University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Stroke Prediction System Using Machine Learning Methods

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Glaiza Rein F. La Rosa

June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Stroke Prediction System Using Machine Learning Methods" prepared and submitted by Glaiza Rein F. La Rosa in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Perlita E. Gasmen, M.Sc. (*cand.*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.	_____	_____
2. Richard Bryann L. Chua, M.Sc.	_____	_____
3. Ma. Sheila A. Magboo, Ph.D.(*cand.*)	_____	_____
4. Vincent Peter C. Magboo, M.D.	_____	_____
5. Marbert John C. Marasigan, M.Sc.(*cand.*)	_____	_____
6. Geoffrey A. Solano, Ph.D.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Maria Constancia O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

Stroke, a deadly disease affecting the brain, has damaging outcomes which may result to death. Its burden has significantly increased in developing countries due to the lack of resources focusing on stroke healthcare and prevention. The need to minimize its effects surged the need to be cautious against the disease and use digital instruments to improve identifying stroke risk. This study implemented different machine learning techniques to predict the probable occurrence of stroke. After removing noise and outliers, data pre-processing was applied along with KNN Imputation to impute missing values. SMOTE was used to handle the imbalance present in the data and after conducting feature selection with the use of ExtraTreesClassifier, XGBoost generated the highest performance metrics among the 7 classifiers. The model was then integrated to the web application making it possible for users to predict whether or not they have the likelihood of having the disease.

Keywords: Stroke, Mean Value and Most Frequent Imputation, KNN Imputation, SMOTE, SMOTE-Tomek, ExtraTreesClassifier, Logistic Regression, Random Forest, Support Vector Machine, Multilayer Perceptron, XGBoost, AdaBoost, KNN

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Stroke is a non-communicable disease that lasts for a long period of time and it is heavily influenced by many factors which leads to its development [1]. It is responsible for being the second leading cause of death in the Philippines [2] and globally, with stroke-related cases significantly higher in low-income groups than high-income groups in terms of age-standardised stroke-related mortality rate and age-standardised stroke-related DALY rate, according to the recent 2019 Global Burden of Diseases, Injuries, and Risk Factors Study (GBD) study [3]. Limited access to sufficient stroke care units, healthcare providers and awareness about the disease attributed to this cause, substantially increasing the risk of cases of stroke in the low-income countries or LIMCs [4]. Additionally, the number of prevalent, incident, stroke-related DALYS and stroke-related deaths increased among young adults [3] with affected age groups younger than 65 years increased by 25% [5]. Risk is significantly high among young adults as there is no formal stroke management governing them and the modifiable risk factors of stroke are highly prevalent in their age group [6].

Past GDB reports have stated that prevention strategies should start by focusing on modifiable risk factors, which causes most of the stroke cases [3][4]. An INTERSTROKE study [7] was able to demonstrate this by associating 10 modifiable risk factors to 90% of population-attributable risk or stroke risk. Although there are also prevention strategies that focus on secondary prevention, some patients suffered disabilities post-stroke and needed rehabilitation. A study [8] indicated that the follow-up rate of stroke patients after a decade was approximately half, some reportedly experienced recurrent stroke or have died. These patients have experienced psychological and cognitive disabilities or illnesses along with poor functional conditions. Based on their results, even if 1 out of 5 can survive in the duration of 15 years after experiencing a stroke, impairments can be experi-

enced by one-third of the survivors. In addition to this, the cost of treatment on stroke patients is quite expensive. According to [5], the more severe the stroke the patient has, the more expensive their inpatient hospital costs. Costs even vary per country, depending on their healthcare systems and stroke, along with other diseases, had a significant impact on the economic burden at a national level, which occurred to some countries. With this in mind, it is best to detect stroke by identifying the risk factors earlier and reduce exposure to these factors to prevent the emergence of the disease.

B. Statement of the Problem

Stroke burden has significantly increased, evident by the rise of related cases[3]. One of its challenges is handling the disease whereas screening for its presence may require the use of imaging tests and additional ones which depend on the state of a person's physiological variables [9]. Moreover, the appropriateness of the test needs to checked since some requires a person to have a particular condition for it to be efficient [10].

[11] accentuated that there is a need to minimize its effects, especially on developing countries, as the prevalence of the disease is common in their areas. They also made the point of focusing on prevention rather treatment as its impact is more advantageous than the latter. In addition, the rate of medical expenses for stroke starts to increase just from the actual onset of the disease. There are currently limitations of healthcare systems focusing on stroke in terms of medical specialists and facilities. These constituents are mostly accessed on an urban setting where the restraints of finances became one of the barriers that motivates this framework [12]. As an effect, this also hinders prevention efforts as people are less aware on the disease and its physiological determinants. [3] has stated that prevention efforts has been futile compared to stroke treatments these past few decades which is why there has been an emphasis on identifying the risk factors as a way to prevent stroke emergence at a population-level. Nonetheless, they also

recommended the idea of stroke prevention strategies at an inexpensive means and at an individual level which should be supported by improving assessments by health professionals, assisting their findings with the application of digital tools to determine people at risk for stroke, which in turn will produce better prevention treatments.

C. Objectives of the Study

The research aims to develop a stroke predictive system that can determine if the user has the likelihood of having a stroke or not based on the physiological variables. Moreover, the research aims to know which of the model configurations employed is the most efficient in terms of model performance and whether appropriate data cleaning has a significant effect on the performance of the model. Specifically, the system can implement the following actions:

1. General User
    (a) can input the values of the variables to the system.
    (b) can view the classification results (stroke or not stroke) based on the values inputted.
2. Admin
    (a) can input the dataset used for training and testing.
    (b) can perform data visualization and exploration
    (c) can pre-process the data using the following techniques:
        i. Handling Imbalance
            A. SMOTE
            B. SMOTE-Tomek
        ii. Impute Missing Values
            A. Mean Value and Most Frequent Imputation

B. KNN Imputation

(d) can split the data into training and testing sets

(e) can perform feature selection on the data using Extra Tree Classifier

(f) classify cleaned data using the following machine learning methods:

i. Logistic Regression

ii. Random Forest

iii. Support Vector Machine

A. Linear

B. Polynomial

C. Radial

iv. Artificial Neural Network (Multilayer Perceptron)

v. XGBoost

vi. AdaBoost

vii. KNN

(g) evaluate the models with the use of k-cross validation

(h) view classification results of both training and testing data.

(i) select the model with the highest performance

D. Significance of the Project

When developed, the project can be used as a tool to create prevention strategies for stroke at a primordial and primary prevention level. Furthermore, by only using physiological variables for an early detection of stroke, it can lessen the number of instruments needed to test for the disease, which in turn, would lessen the cost of medical tests by a potential patient.

E. Scope and Limitations

1. The training data will only be limited to the supplementary dataset by Liu [13] which is a benchmark dataset taken from HealthData.gov.

2. Physiological variables will only be limited by the 10 predictors used in the dataset:

   (a) Gender

   (b) Age

   (c) Hypertension

   (d) Heart Disease

   (e) Ever Married

   (f) Work Type

   (g) Residence

   (h) Average Glucose Level

   (i) BMI

   (j) Smoking Status

3. Prediction of stroke is only limited to two classes: stroke and non-stroke.

II. Review of Related Literature

There had been previous studies that focused on developing prediction systems on stroke with different methods and datasets. A study by [14] compared 12 machine learning methods namely Logistic Regression, SVM, KNeighbors, GaussianNB, BernoulliNB, Decision Tree, Random Forest, XGBoost, AdaBoost, Bagging, LGBM and Neural Network using the Stroke Prediction Dataset which consists of 5110 observations. Neural Network was deemed the most optimal of the methods with ROC score 0.843. Although most prediction systems emphasize accuracy as their evaluation metric to support their results, this study did not follow the same trend. Most of the methods achieved approximately 90% accuracy however they were not considered efficient considering their percentages of ROC-AUC and F1 scores.

The research by [15] also used other metrics on their research which compared Artificial Neural Network against optimal classifiers such as Random Forest and XGBoost on a Kaggle Dataset with around 10,000 entries. Other than accuracy, the researchers used efficiency rate as their basis of comparing model performance. Efficiency rate, as described by their study, encompassed sensitivity, specificity, precision, recall and F1 scores. They used CSL to generate which weights are optimal to apply on the model; however, their proposed method failed in contrast to the optimal methods. Nevertheless, they deduced that it can be improved with a large or increase in the number of data used for training.

According to [13], false positive and false negative rates should be highlighted since these metrics have a significant influence on the clinical diagnosis and psychological burden to the patients, making these metrics the basis of significance of their proposed model. It is important to address the accuracy paradox as it is an effect of the presence of imbalanced data. Imbalanced data pertains to the uneven proportion of the data which results in the existence of the majority and minority class. Evaluation of the model may result with accuracy accounted mostly by the majority class. This was illustrated by [14] where their data generated more True

Positive values than the other metrics with an observed low precision but high accuracy which is an illustration of the accuracy paradox. Although accuracy is an important metric, classification of diseases, especially those which are used along clinical diagnosis and have devastating consequences when misdiagnosed, should not only rely on accuracy and consider aiming for a high recall [16].

Development of predictive systems does not only rely on machine learning techniques applied to the dataset. Results of these systems can be further improved by pre-processing techniques before the actual comparison of machine learning methods. The study by [13] used Random Forest Regression to impute missing values on the Stroke Dataset taken from HealthData.gov before introducing the data to an AutoHPO-based DNN model. While accuracy amounted to 71.6%, it showed a small false positive rate of 19.1%. Models with different applications of balancing techniques such as Random Oversampling (ROS), Random Undersampling (RUS) and SMOTE were compared in [17] among machine learning methods such as Regularized Logistic Regression, Support Vector Machine and Random Forest. The study showed that there is a significant increase in the classification results when balancing techniques were applied and models applied with SMOTE showed high AUC scores compared to the other balancing techniques. On the other hand, 3 data selection methods- without data resampling, with data imputation and with data resampling were applied and evaluated in [18] where they used the data from the National Health and Nutrition Examination Survey or NHANES. From the results, data resampling was the method that gives the most optimal metrics on the machine learning classifiers applied: Naive Bayes, BayesNet, J48 and Random Forest, especially with the latter, having an AUC score of 0.97 and 0.96 accuracy. Moreover, the study also used 10-fold cross validation to allocate observations for training and validation and used other performance measures than accuracy such as sensitivity, specificity, positive predictive value or precision, negative predictive value and AUC score. Researchers in [14] used SMOTE-Tomek where SMOTE performs oversampling on the data while Tomek links is assigned to perform un-

dersampling after SMOTE before training the data with multiple machine learning methods. In contrast, [19] only used SMOTE to handle the imbalance and applied it to classifiers Logistic Regression, Random Forest and XGBoost. Random Forest had the most optimal results with accuracy of 99.07%, precision and recall scores of 99%. Furthermore, this study implemented the Mean Value Imputation to address missing values, label encoding for re-encoding of categorical variables and feature scaling for data normalization. Exploratory Data Analysis was also used to illustrate the correlation between variables and how it relates to the results. The research of [20] also used SMOTE to address imbalance and used mean value imputation to fill missing values. Out of the four machine learning methods applied: Decision Tree, Logistic Regression, Voting Classifier and Random Forest, the latter had the highest performance with 96% accuracy. Although no feature selection or imbalance- handling techniques were mentioned, outliers were removed in [21]. They used k-cross validation to split the training and testing sets before applying it to linear, quadratic and cubic Support Vector Machine models. Out of the three SVM methods deployed in the study, linear and quadratic functions generated approximately 90% accuracy. K-cross validation was also used in [22] for hyperparameter tuning however, the method was only used on the training set. Their data was derived from the China Kadoorie Biobank or CKB, a prospective cohort study and models were developed with use of the following methods: Cox, Random Survival Forest, Logistic Regression, Support Vector Machine, Gradient Boosted Tree and Multilayer Perceptrons. The researchers developed models that predict stroke risk within 9 years of the survey and 3 follow-up intervals after the survey: 0-3 years, 3-6 years, 6-9 years. Although Gradient Boosted Tree was said to be the most optimal in terms of AUROC and calibration performance, the models the researchers created surpasses that of the Framingham Stroke Risk Profile. The data pre-processing techniques stated in the study are one-hot coding of the geographical areas and imputing missing values using Mean Value Imputation. From the results, observed first stroke risk factors indicated are age and people

with a record on CHD, diabetes and hypertension. Although also a cohort, the study of [23] classified 10-year probability of stroke with the data derived from the NHIS in Korea. Outliers were removed and Cox's proportional hazard regression model was applied to classify stroke risk prediction into 5 ranges namely normal, slightly high, high, risky and very risky. From their study, the indicated significant risk factors for stroke are age, smoking, diabetes and hypertension. Both cohort studies did not include the feature atrial fibrillation in their research and analyzed the values separately by gender.

III. Theoretical Framework

A. Stroke

Stroke is a very complicated disease where its intricacy lies in its root causes which can be several and uncertain in number. According to [24], there are three main types of this disease namely, Ischemic, Transient Ischemic stroke and Hemorrhagic stroke. Ischemic stroke happens when a blood clot prevents the blood flow to the brain. Transient Ischemic Stroke also functions just the same but the damage of the blood clot is not that severe and the presence of the blood clot is only temporary. However, this type of stroke is susceptible to a full stroke onset, making this type a warning or a mini-stroke. The last type is the Hemorrhagic stroke which refers to the damage caused by blood leakage inside the brain. Symptoms and severity of the disease depends on the affected part and patients who experienced this can suffer long-term disability and death [8].

Development in research and medicine revealed that additional risk factors that influence this disease have been discovered. Although these factors are many, they can be classified into modifiable and non-modifiable types, where the former is mostly highlighted by researchers as they can control these variables to prevent the emergence of the disease [5][25]. It was stated in [3] that Body Mass Index is the fastest-growing risk factor in the last two decades with other leading risk factors such as high systolic blood pressure or hypertension, high fasting plasma glucose or diabetes, ambient particulate matter pollution and smoking. Reducing the exposure to these risk factors along with managing physical activity and diet is the recommended effective prevention strategy however it was stated that this has yet to improve as statistics implied that stroke prevention has not been fully utilized [3].

B. Machine Learning Methods

Machine learning methods have been proven to be valuable in the development of decision-making tools in healthcare and medicine. Some of its applications include systems that predict diseases which can help formulate the appropriate treatments for the patients, cost-effective clinical data management, medical imaging analysis and computerized drug discovery [26]. Since they are an automated process that can find the underlying pattern in the data, they are efficient to use and robust against large data. Moreover, with large data, the model can generate more accurate results as it learns from the data itself.

1. Logistic Regression

   Logistic Regression is a supervised learning algorithm that classifies classes based on probability. It is only applicable in events where the target variables are dichotomous, no multicollinearity in the data, used significant variables and a large sample size [27].

2. Random Forest

   Random Forest is a group of Decision Trees that functions as an ensemble classifier. Prediction is made based on the majority of the predictions of these trees and each of these predictions have low correlations with each other. This classifier executes the concept of bagging where the trees resample the data with replacement [28].

3. Support Vector Machine

   Support Vector Machine operates with the help of support vectors in which hyperplanes depend on. These hyperplanes help classify data points and it is ideal to find support vectors that maximise the margin between data points and the hyperplanes. This classifier also has less computational power while giving a high performance model [29]. There are many types of kernels used in SVM but the ones to be used in the study will be the following:

   (a) Linear

When using linear kernels, data points are linearly separated and fast in execution. It is highly ideal in classification problems on textual data [30].

(b) Polynomial

This kernel is the generalized form of the linear function and is used mostly when the data points are not linearly separable; however it is not always used as it performs poorly compared to other kernels . The most important parameter to consider for this kernel is the degree value [31].

(c) Radial Basis Function

Radial Basis Function or RBF is one of the most commonly used kernels since it can make use of infinite dimensions to map feature space. The most important parameter to consider for this kernel is the Gamma value [31].

4. Artificial Neural Network

Artificial Neural Networks make use of layers- the input, the hidden and the output layer. These layers output weights which depend on the function of the layers and then feed these weights to the activation function within the output layer. This classifier learns depending on the weight calculated and it cannot be limited by datasets [32].

(a) Multi-layer Perceptron

Multi-layer Perceptron is a neural network that is based on a feed-forward algorithm, wherein each linear combination generated at each layer is consumed by the next one until it stops at the last layer. In addition, it also adopts backpropagation which is responsible for cost function minimization. Convergence is achieved when a specified threshold has been reached, after an iteration process of computing Mean Squared Error gradient and updating the gradient value of the first hidden layer.

Different from a Perceptron, it is applicable to non-linear data and there is no limit on what neurons can use as an activation function [33].

5. XGBoost

   Extreme Gradient Boosting or XGBoost uses the gradient boosting decision tree algorithm which reduces the loss by boosting weak learners and applying systems optimization and algorithmic enhancements, which makes it less prone to overfitting. Execution of this algorithm is fast and highly ideal on tabular datasets [34].

6. AdaBoost

   Adaptive Boosting or AdaBoost is an ensemble boosting method which makes use of weak learners or classifiers iteratively to create a highly accurate classifier. It uses weighted training examples as a basis to track which classifier makes a lot of wrong observations and what weight to set per iteration. Since it uses boosting like XGBoost, it is also less prone to overfitting [35].

7. KNN

   K-Nearest Neighbors or KNN is a non-parametric algorithm that uses distance measures from k or the majority of data points of a selected k to assign a label for classification. Parameters k and distance measures are the most essential factors to look for in this classifier. Examples of distance measures which can be used are Euclidean and Hamming distance [36].

C. Handling Imbalance and Missing Values

1. Balancing Techniques

   (a) SMOTE

   Synthetic Minority Oversampling Technique or SMOTE is an oversampling method where it creates synthetic samples using the distance

between random data from the minority class and its k nearest neighbours [37]. Application of this method is only done to a proportion of the dataset, typically the training set, which studies have failed to apply correctly [14].

(b) SMOTE-Tomek

SMOTE-Tomek is a combination of oversampling method SMOTE and undersampling method Tomek links. This method addresses both minority and majority classes [14].

2. Imputation Techniques

(a) Mean Value Imputation

Mean Value Imputation replaces the missing numerical values with the mean value taken from the other columns. It does not reduce the sample size and is easy to apply, however there is a chance of biased estimates depending on the kind of response mechanisms [38].

(b) Most Frequent Imputation

Most Frequent Imputation replaces the missing values with the most frequent value of the feature with missing values. Because of this, it is commonly known as Mode Imputation as it uses the mode value for imputation. It is usually applied on categorical variables as opposed to Mean Value Imputation [39].

(c) KNN/ Nearest Neighbour Imputation

KNN is a non-parametric method that uses distance measures such as Euclidean distances and k-neighbours to impute missing values. This algorithm is applicable to any kind of data however there are factors that limit this function mainly, the k number of neighbours or hyperparameter k, aggregation method, attribute distance and data normalization [40].

D. Feature Selection

To find which predictors have the most significant effect on the model performance, Extremely Randomized Trees Classifier or Extra Trees Classifier will be conducted. It is an ensemble learning algorithm where each Decision Tree selects the best feature to split based on a criteria such as the Gini Index. Feature selection using this algorithm can be implemented by using the Gini Importance of each feature [41]. Feature selection can reduce overfitting, improve accuracy and reduce training time [42].

E. Performance Metrics

The methods applied will be evaluated using the classification metrics with the use of the contingency table below.

	Predicted	
Actual Value	Stroke	Not Stroke
Stroke	True Positive (TP)	False Negative (FN)
Not Stroke	False Positive (FP)	True Negative (TN)

Table 1: Contingency Table

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad Sensitivity/Recall = \frac{TP}{TP+FN}$$

$$Specificity = \frac{TN}{TN+FP} \qquad Precision = \frac{TP}{TP+FP}$$

$$f1score = \frac{TP}{TP+\frac{1}{2}(FP+FN)}$$

The Receiver Operating Characteristic (ROC) with the Area Under the Curve (AUC) will also be plotted and analysed.

IV. Design and Implementation

A. Dataset

The study used a supplement dataset from [13] where it was originally used as a benchmark dataset in a Kaggle competition with origins from HealthData.gov. It is publicly available at Mendeley Data [43]. It consisted of 12 features and 43,400 samples with an observed imbalance characteristic in the outcome class. The stroke class were comprised of 783 rows while the non-stroke class had 42,617 rows, making the former only ∼1.8% of the outcome class.

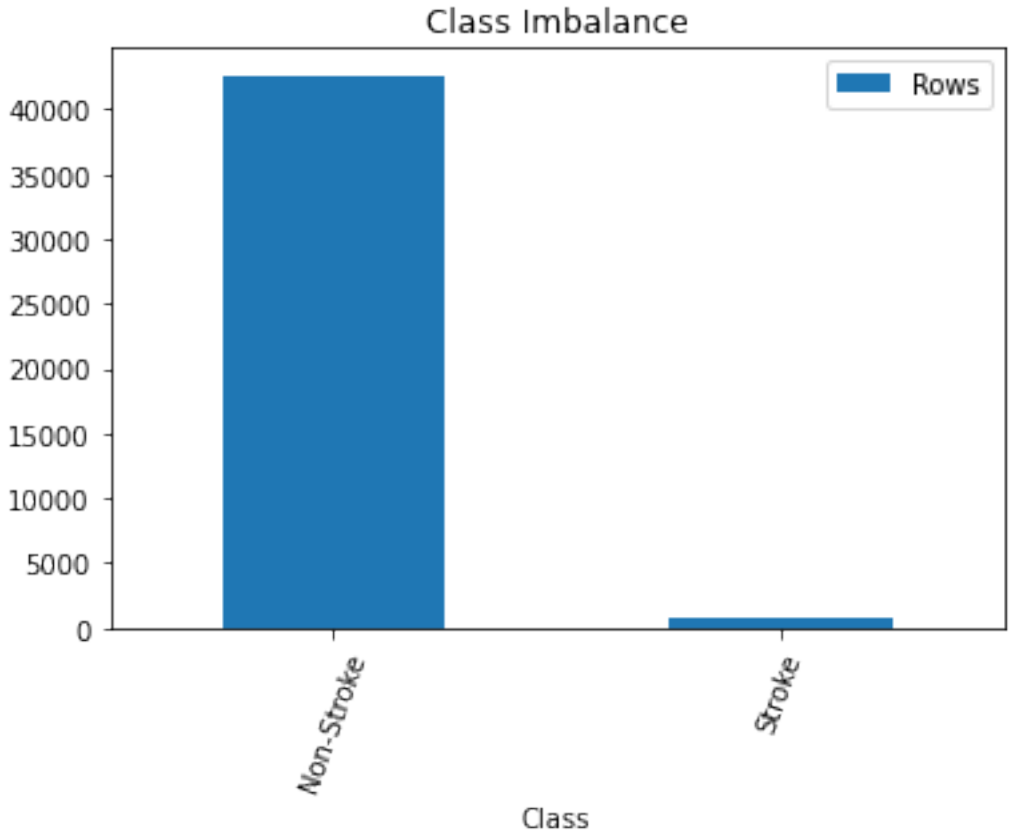

Figure 1: Class Imbalance

Features 'BMI' and 'Smoking Status' also have some noticeable traits. The former has 1,462 (∼3%) missing rows while the latter has 13, 292 (∼30%) missing values. Details about the features are illustrated in table 1. Since Patient ID is not to be used in the study, it is to be excluded in further research.

Feature	Value/Range
Gender	Male Female Other
Age	Min: 0.08 Max: 82.0
Hypertension	1 0
Heart Disease	1 0
Ever Married	Yes No
Work Type	Govt_job Never_worked Private Self-employed children
Residence	Rural Urban
Average Glucose Level	Min: 55.0 Max: 291.05
BMI	Min: 10.1 Max: 97.6
Smoking Status	formerly smoked never smoked smokes

Table 2: Feature Details of Raw data

B. System Overview

The system classifies the variables inputted by the user into probable stroke or not stroke. Generally, the following are the steps executed for each machine learning technique. First, the data was split into training(80%) and testing(20%) sets wherein the former was cleaned - noise and outliers were removed before explored for further analysis. To avoid data leakage during hyperparameter optimization, the data was fed to a pipeline containing the pre-processing techniques (ordinal encoding, one hot encoding and robust scaling with imputation techniques Mean Value and Most Frequent Imputation or KNN Imputation), imbalance handling techniques (SMOTE or SMOTE-Tomek), feature selection (ExtraTreesClassifier) and classifier before evaluated with the help of 10-cross validation.

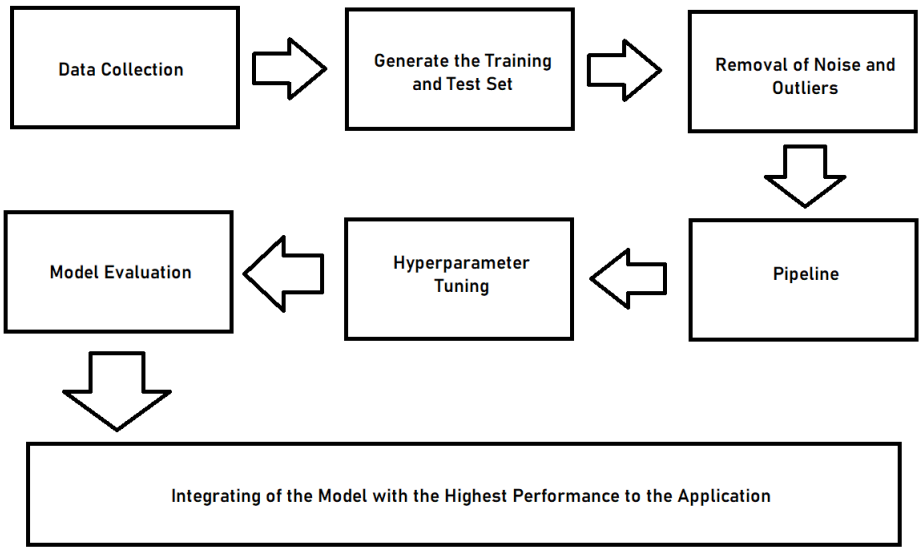

Figure 2: Flow Diagram

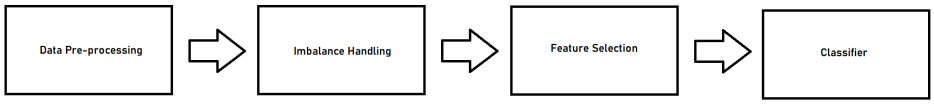

Figure 3: Pipeline Definition

The following are the hyperparameters optimized for each classifiers. AUC score is used as a metric to evaluate the cross-validation process with hyperpa-

rameter technique GridSearch.

Classifier	Hyperparameters
Logistic Regression	C:[0.0001,0.001, 0.01, 0.1, 1, 10, 100] solver:['sag', 'saga','newton-cg', 'lbfgs'] max_iter:[1000,2000,3000]
Random Forest	n_estimators:[10,100,500] min_samples_leaf:[1, 3, 5] max_features:['sqrt','log2']
Support Vector Machine	kernel:['linear','poly','rbf'] C:[0.1,1,10]
Multi-layer Perceptron	solver:['lbfgs','sgd','adam'] activation:['logistic', 'tanh', 'relu']
XGBoost	learning_rate[0, 0.0001, 0.001, 0.01] max_depth:np.arange(1, 11, 3) min_child_weight:np.arange(1, 11, 3)
AdaBoost	learning_rate:[0.0001,0.001,0.01,0.1,1] n_estimators:[10,50,100,500]
KNN	n_neighbors:list(range(1,21,1)) weights:['uniform', 'distance'] metric:['euclidean', 'manhattan', 'minkowski']

Table 3: Hyperparameters for Optimization

The following illustration summarizes the order of the pre-processing techniques which differs between the 2 imputation techniques in the form of a pipeline.

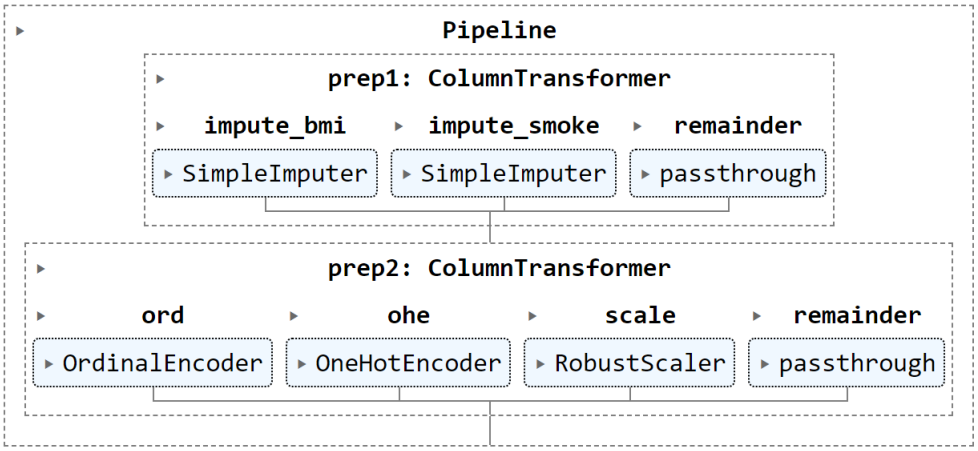

Figure 4: Pre-processing with Mean Value and Most Frequent Imputation

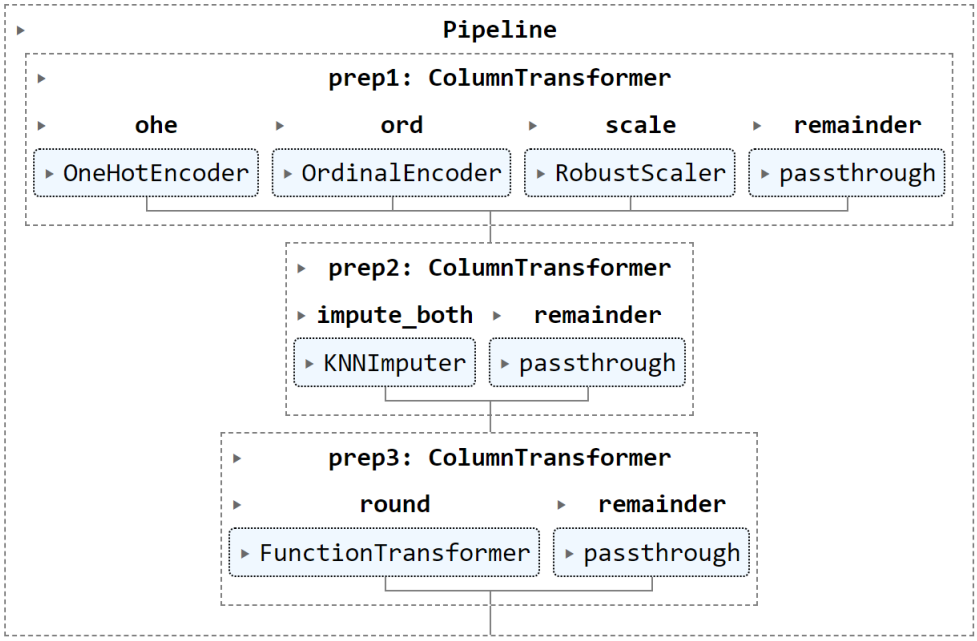

Figure 5: Pre-processing with KNN Imputation

For Mean Value and Most Frequent Imputation (Fig. 4), the imputation comes first before the techniques mentioned while it is the opposite for KNNImputation (Fig. 5), as it requires numerical values for it to work [44].

After determining the model with the highest performance, it was deployed into a web application wherein the user can input data and classify whether the feature input has a probable stroke or not.

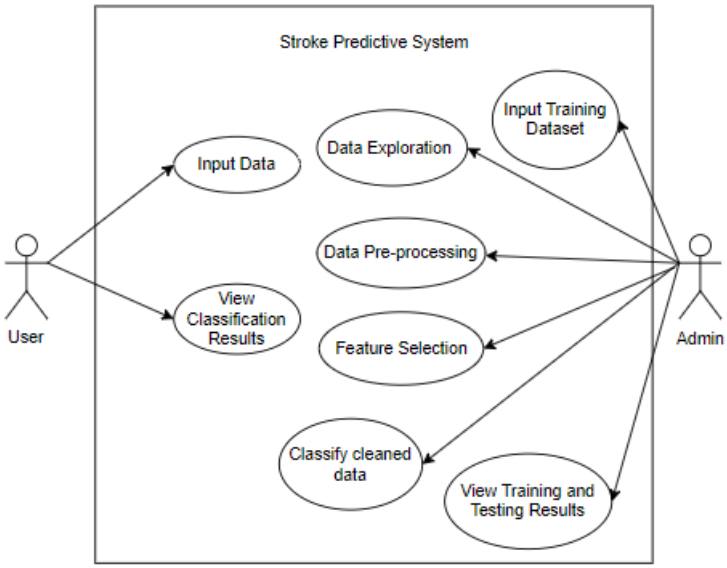

Figure 6: Use Case Diagram

C. System Architecture

The system was deployed with the use of Django Framework. Python was used as the programming language and for the implementation of machine learning methods, pandas, numpy, sci-kit learn and imblearn library were used. The Admin page of the system uses Jazzmin as the UI theme.

V. Results

A. Exploratory Data Analysis

Before conducting exploratory data analysis, rows containing the value 'Other' for the feature 'Gender' was removed from the training set. The study used [13] as a reference to determine noise and outlier values on the data as the process indicated in the paper utilizes medical expertise. Rows with BMI values exceeding 60 were removed along with rows with age less than 25.

	Original	Gender:Other	Age $<$ 25	BMI $>$ 60
Shape	(34720,10)	(34710,10)	(25871,10)	(25802,10)
Difference of Rows from the Preceding	-	10	8839	69
Percentage	-	∼0.03%	∼25.47%	∼0.27%

Table 4: Changes in the Shape of the Training Set (X) After Each Succeeding Noise and Outlier Removal Method

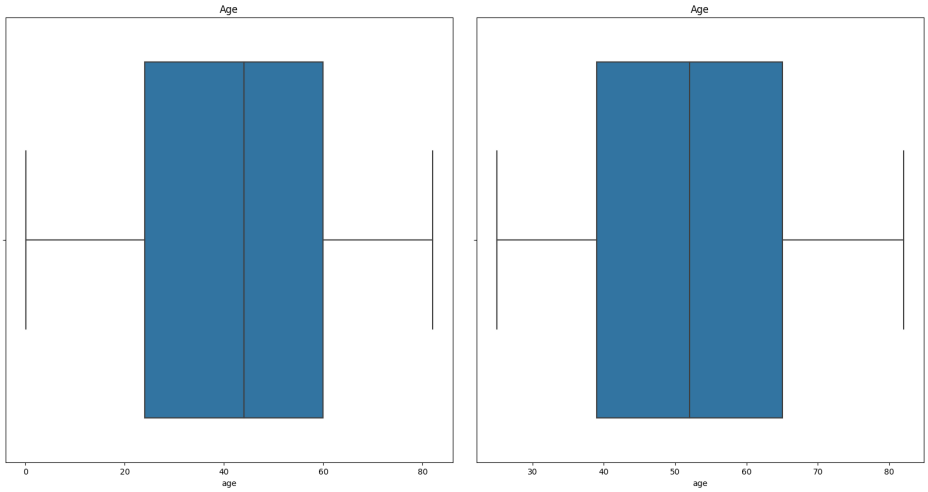

Figure 7: Age Before and After Removal

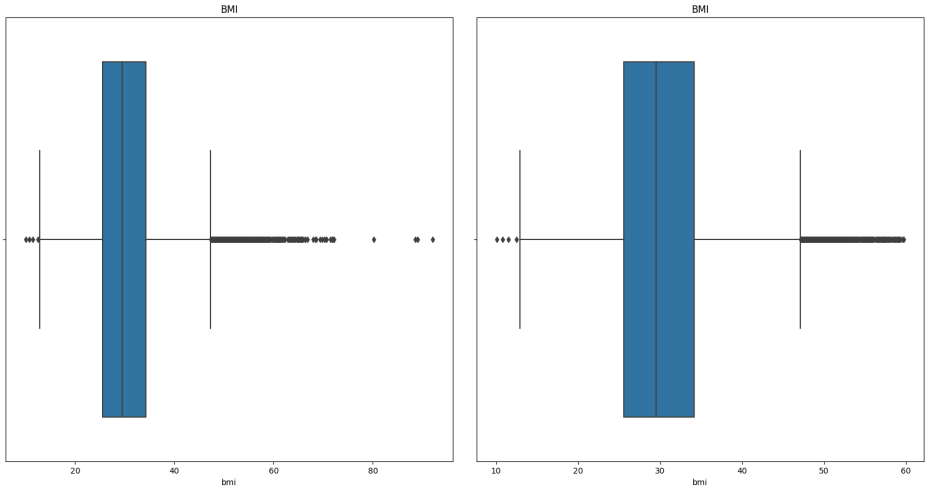

Figure 8: BMI Before and After Removal

Since ages less than 25 has been removed, there are no more values that pertains to children hence the category 'children' of feature 'work_type' has been removed. As the resulting training set (X) now only 25,802 rows and 10 columns, the training set(y) were filtered with the indices of the former. From its rows, 630($\sim$2.44%)are of stroke class while 25,172($\sim$97.56%) are of non-stroke class, illustrating a huge imbalance in the outcome class.

After the application of pre-processing techniques, vif values were computed.

Feature	VIF
smoking_status	1.026927
gender_Male	1.014489
ever_married_Yes	1.092446
work_type_Govt_job	2.666644
work_type_Never_worked	1.004002
work_type_Private	7.650195
work_type_Self-employed	3.139766
Residence_type_Urban	1.000471
age	1.338180
avg_glucose_level	1.091406
bmi	1.042370
hypertension	1.072770
heart_disease	1.087913

Table 5: VIF-Mean Value and Most Frequent Imputed

Among features in the mean value and most frequent imputed data, feature 'work_type_Private' has the highest VIF value followed by 'work_type_Self-employed' and 'work_type_Govt_job'. This affects their relationship with other features, supported by the heatmap below.

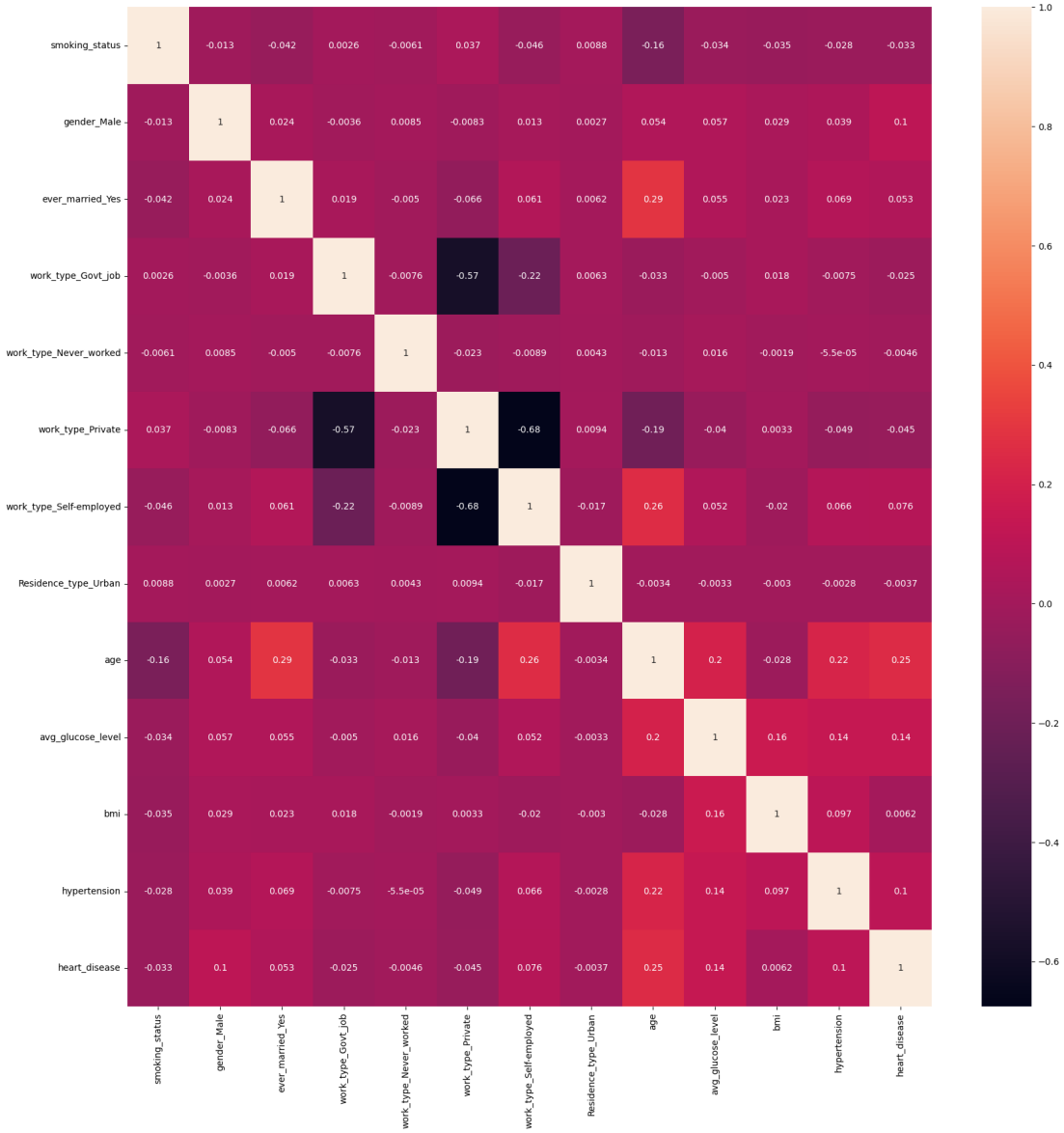

Figure 9: Mean Value and Most Frequent Imputed Data (Heatmap)

At first glance, it can be observed that features 'work_type_Self-employed' and 'work_type_Govt_job' has a high negative correlation with feature 'work_type_Private'. The two also has a negative correlation with each other. Other features correlated with 'work_type_Private' does not seem to show any correlation at all with the exception of age (-0.19).

Interestingly, among features, feature age has a lot of positive correlations equal or greater than 0.20 with features ever_married_Yes, work_type_Self-employed, heart_disease, hypertension and avg_glucose_level. Other than fea-

ture work_type_Private, age also has another negative correlation with feature smoking_status (-0.16).

Additionally, here are other notable positive correlations, with value equal or greater than 0.10, although not of high value : heart_disease with features gender_Male, avg_glucose_level and hypertension and avg_glucose_level with hypertension and bmi. Only features Residence_type_Urban and work_type_Never_worked do not have any significant correlations with other features.

The same observations can be observed on the KNN imputed data, with vif values illustrated below.

Feature	VIF
smoking_status	1.027286
bmi	1.041878
gender_Male	1.014534
ever_married_Yes	1.092409
work_type_Govt_job	2.642848
work_type_Never_worked	1.003971
work_type_Private	7.549625
work_type_Self-employed	3.107659
Residence_type_Urban	1.000436
age	1.336993
avg_glucose_level	1.089896
hypertension	1.072057
heart_disease	1.087926

Table 6: VIF-KNN Imputed

The same correlations also show in the following heatmap representing KNN imputed data.

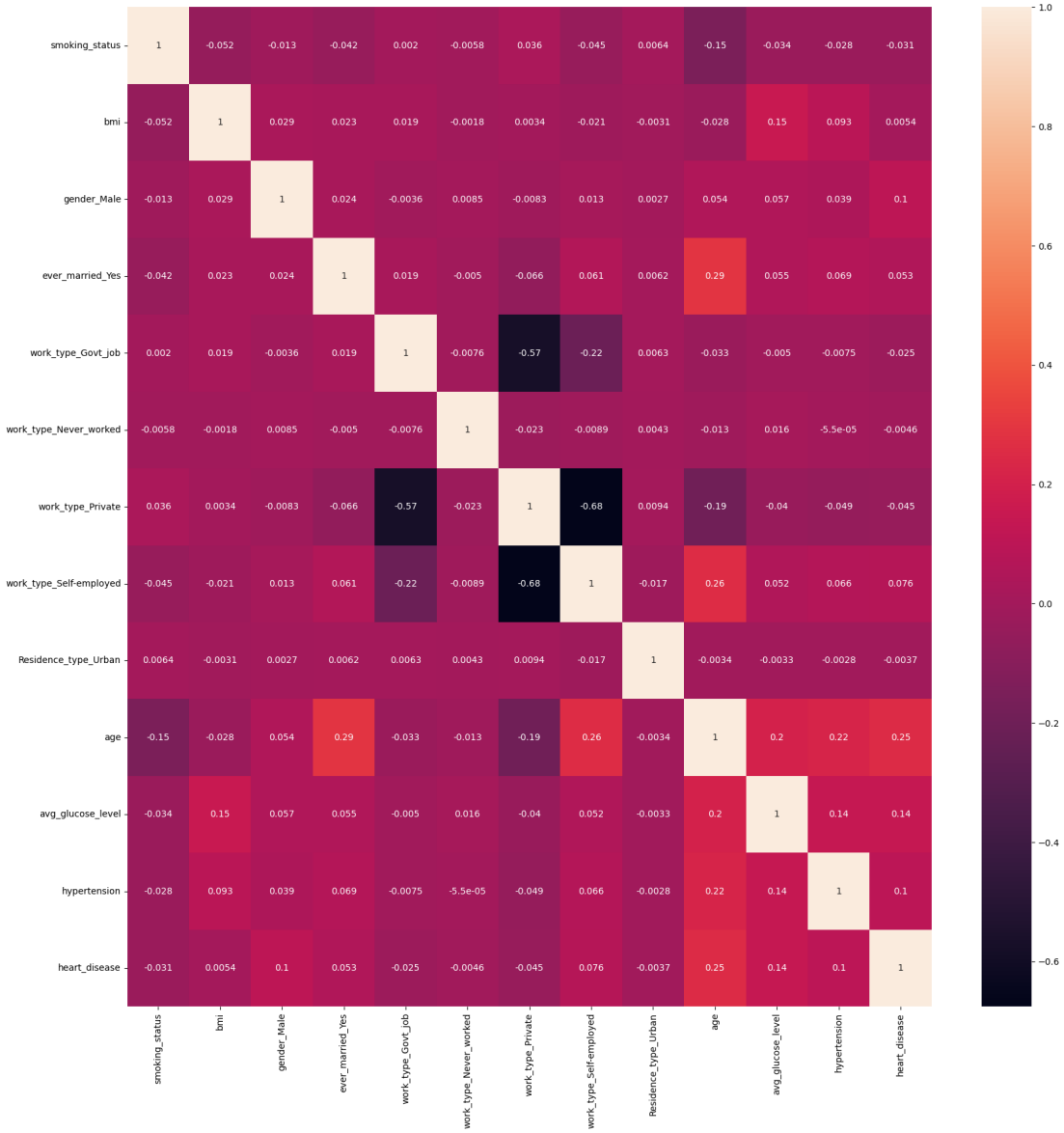

Figure 10: KNN Imputed Data(Heatmap)

To further understand the data, it was explored without the application of RobustScaling. The following figure shows the pairwise relationships of the continuous features present in the data. Plotted diagonally are the univariate plots of each feature while those which are off the diagonal subplots are the scatterplots that show the relationship between the 2 features.

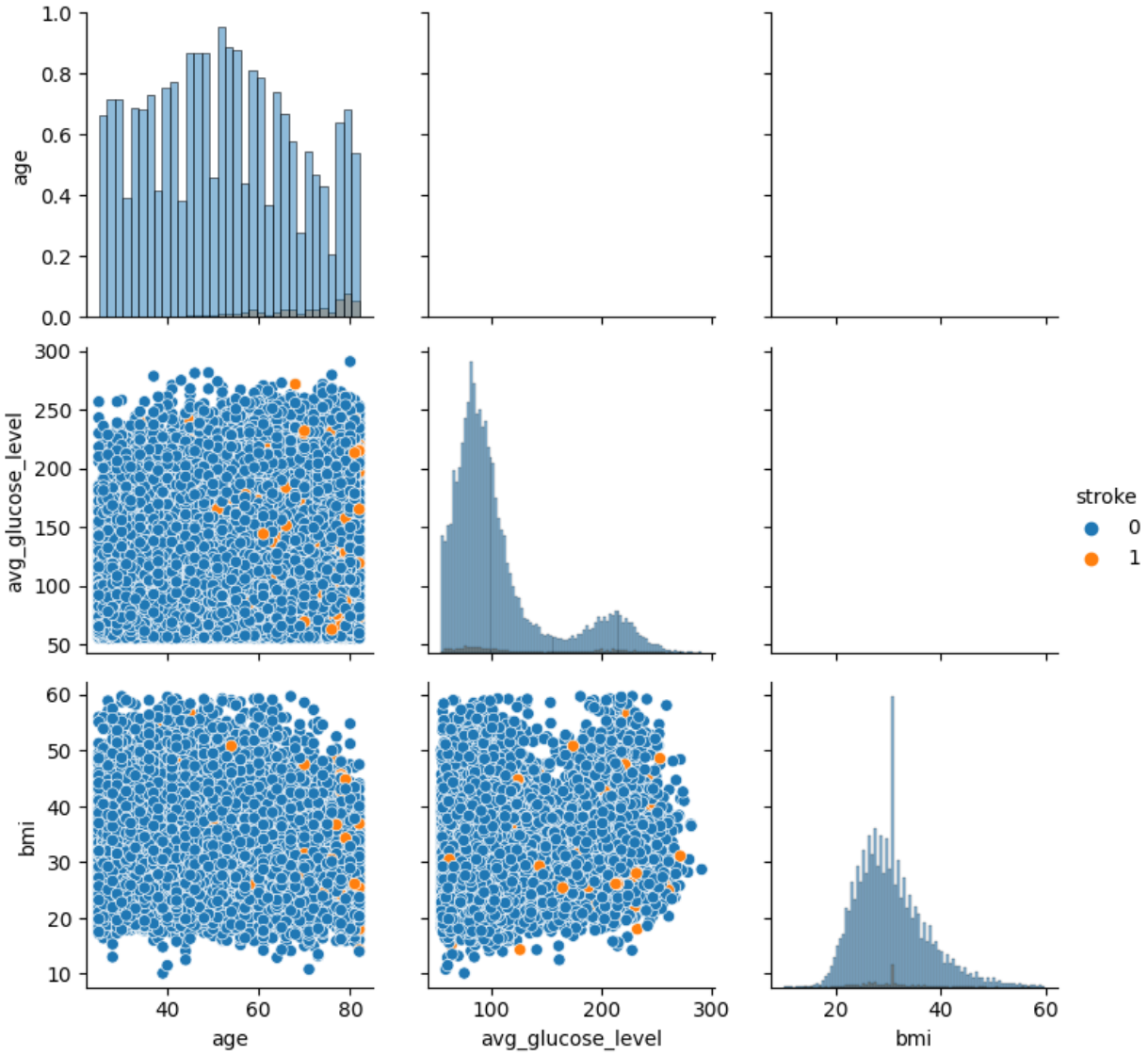

Figure 11: Pairwise Relationship of Continuous Features (MV and MF Imputed Data)

It can be seen in Fig. 11 that age shows approximately a normal distribution while both average glucose level and bmi do not. Average glucose level is heavily skewed to the right, with average values ranging around 100. BMI values are also somewhat skewed to the right however there is a distinct data point around the value 30 in the plot.

Some additional observations also include the following: regardless of age, the average glucose level is around 260 and at least bmi values of 15 were included in

the data. Correlations are not distinct in the plots.

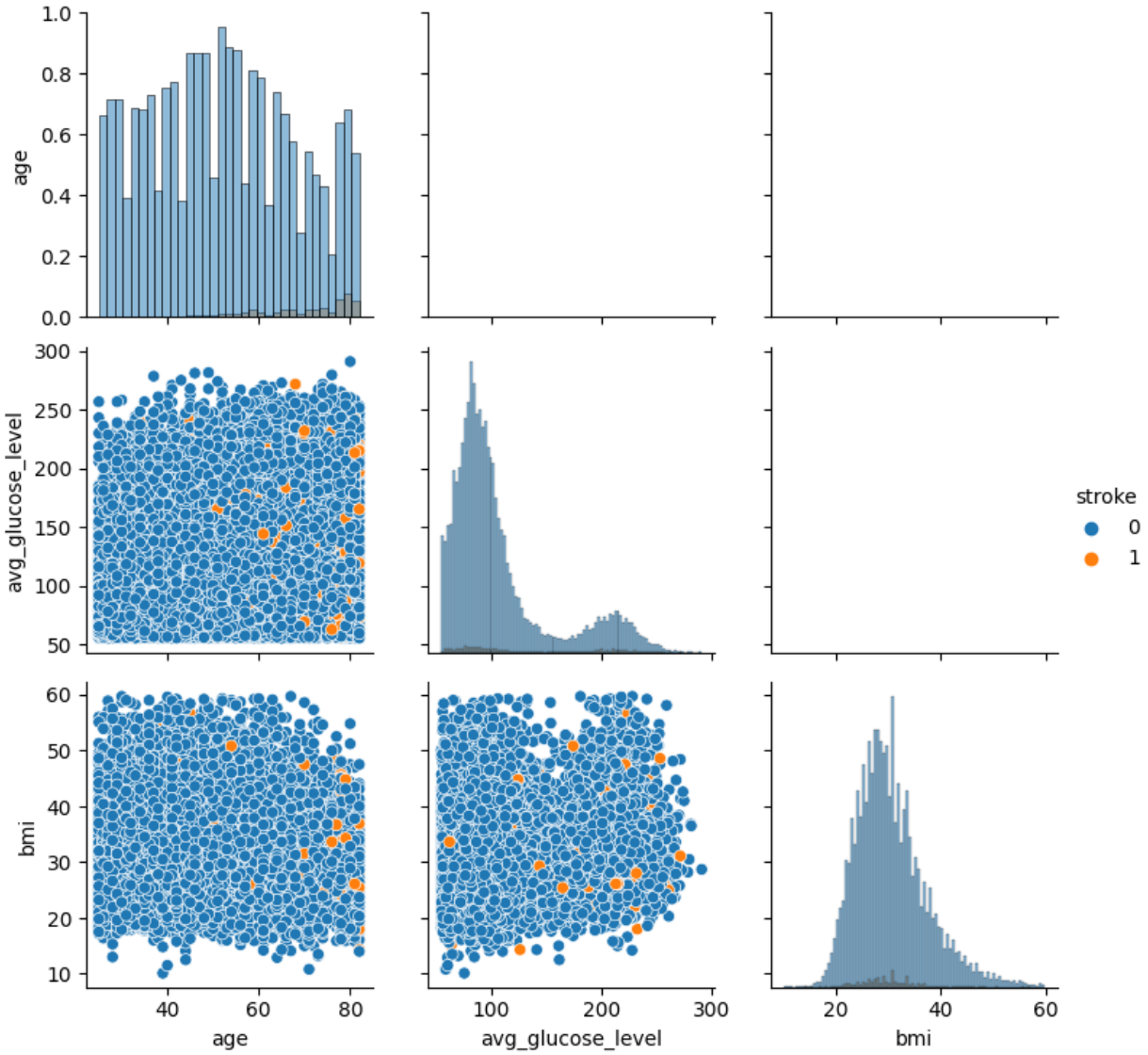

Figure 12: Pairwise Relationship of Continuous Features (KNN Imputed Data)

Like in Fig. 11, age also shows an approximately normal distriution while average glucose level is also heavily skewed to the right. The distribution of the BMI values however displays a different distribution plot instead of a near flat distribution showed in Fig. 11. Correlations are not distinct in the plots.

In both figures, the scatter plots showed that the distribution of stroke classes are heavily imbalanced. There is a huge distinction between stroke and not stroke among the distribution of the features. Stroke seems to be more prevalent in

people who are 50 and above, most likely to older ones and those who have BMI values more than 25. Stroke also seems to be prevalent no matter what the average glucose level is.

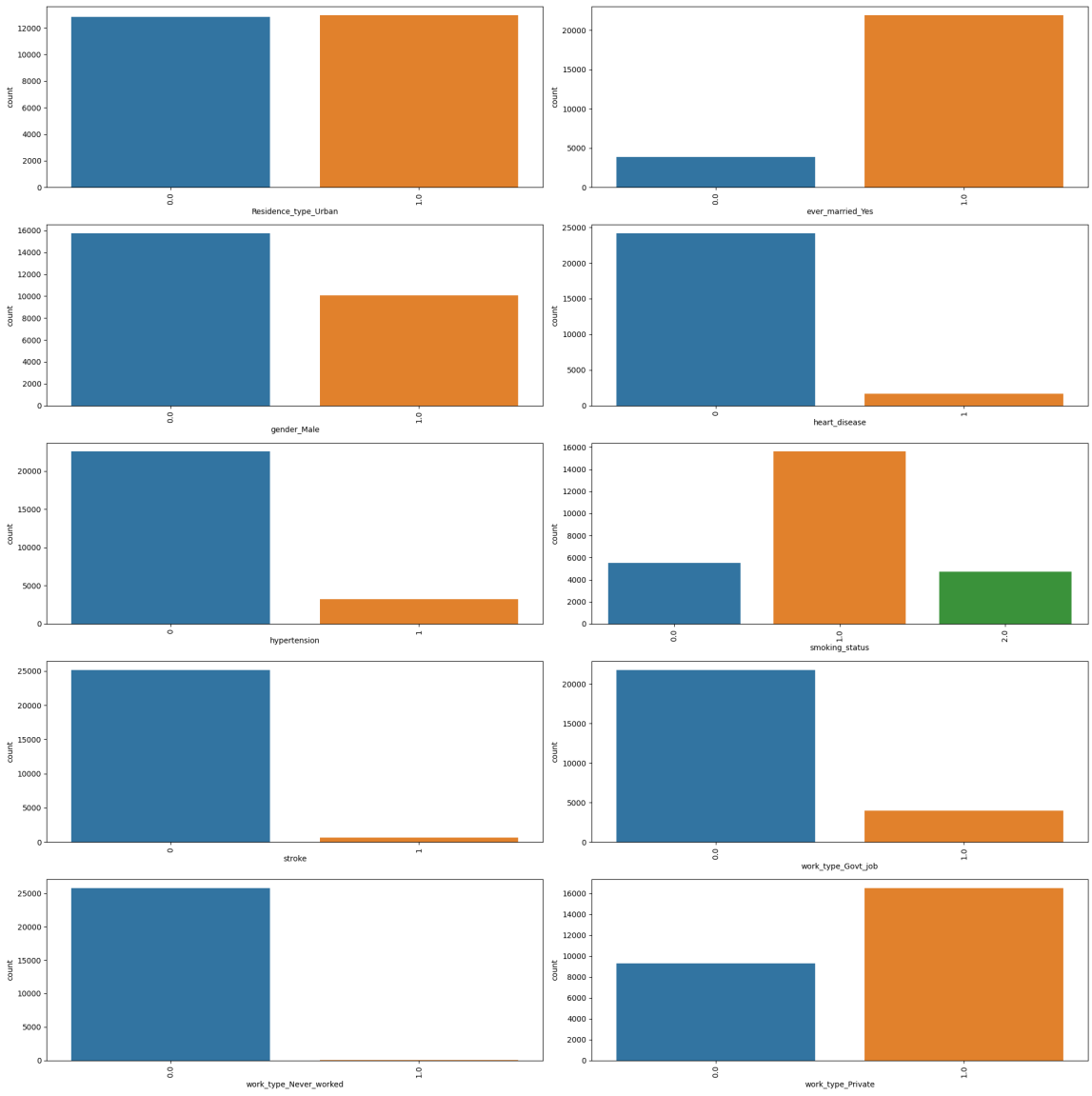

Figure 13: Count Plots of Categorical variables (Mean Value and Most Frequent Imputed Data)

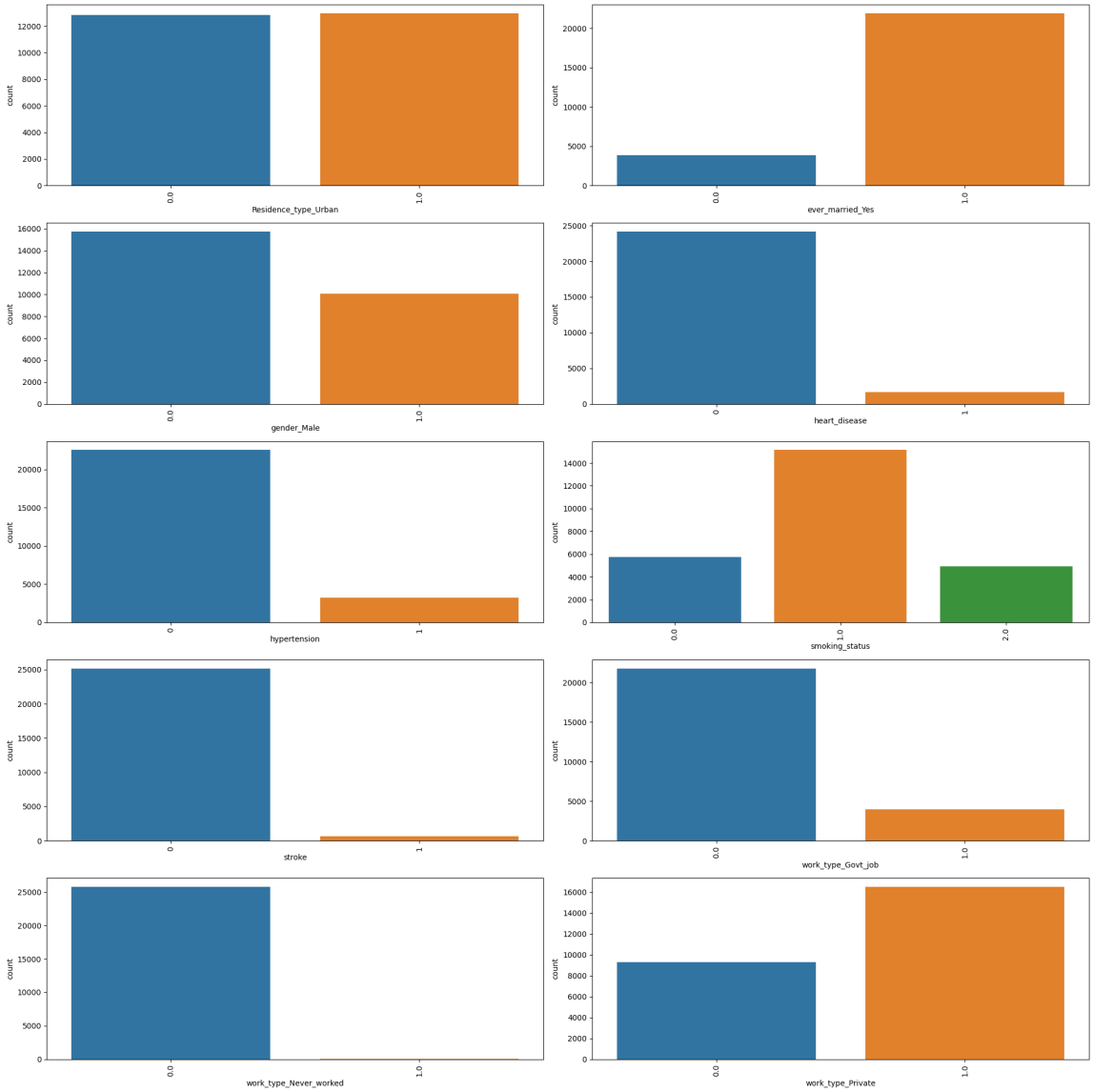

Figure 14: Count Plots of Categorical variables (KNN Imputed Data)

Among the countplots above, it can be observed that only the feature Residence_type_Urban has almost the same distribution for both data. Features ever_married_Yes and work_type_Private has more distribution of positive target class than that of the negative class. Among the 3 categories of feature smoking_status, category 1(never_smoked) has the most distribution followed by 0 (formerly_smoked) then 2 (smokes). The rest of the features has more negative class than that of the positive class. Both figures displaying count plots of categorical variables showed no difference in the distribution of classes.

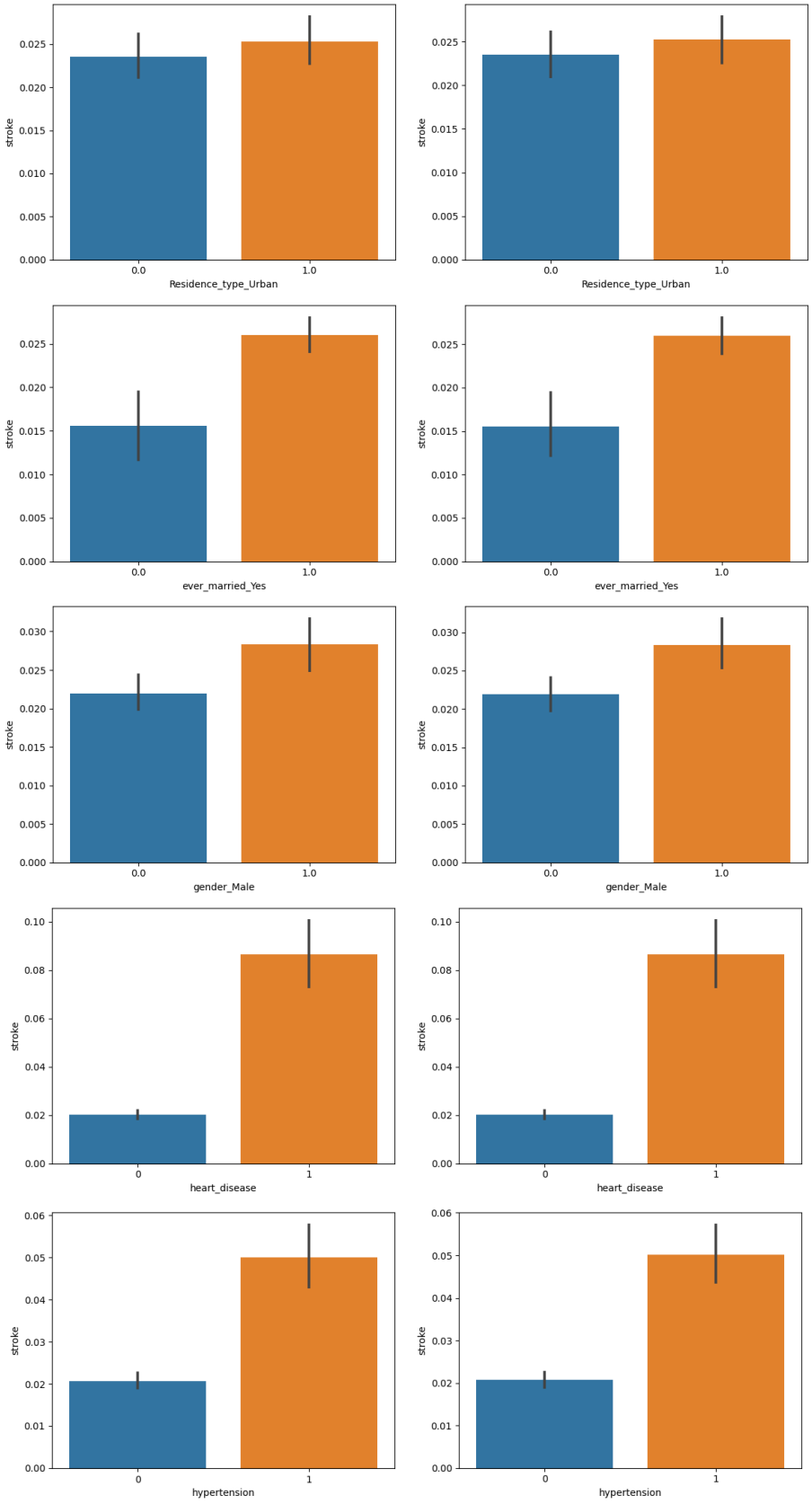

Figure 15: Bivariate Analysis of Categorical Variables VS Target Class Mean Value and Most Frequent Imputed Data(Left) | KNN Imputed Data(Right)

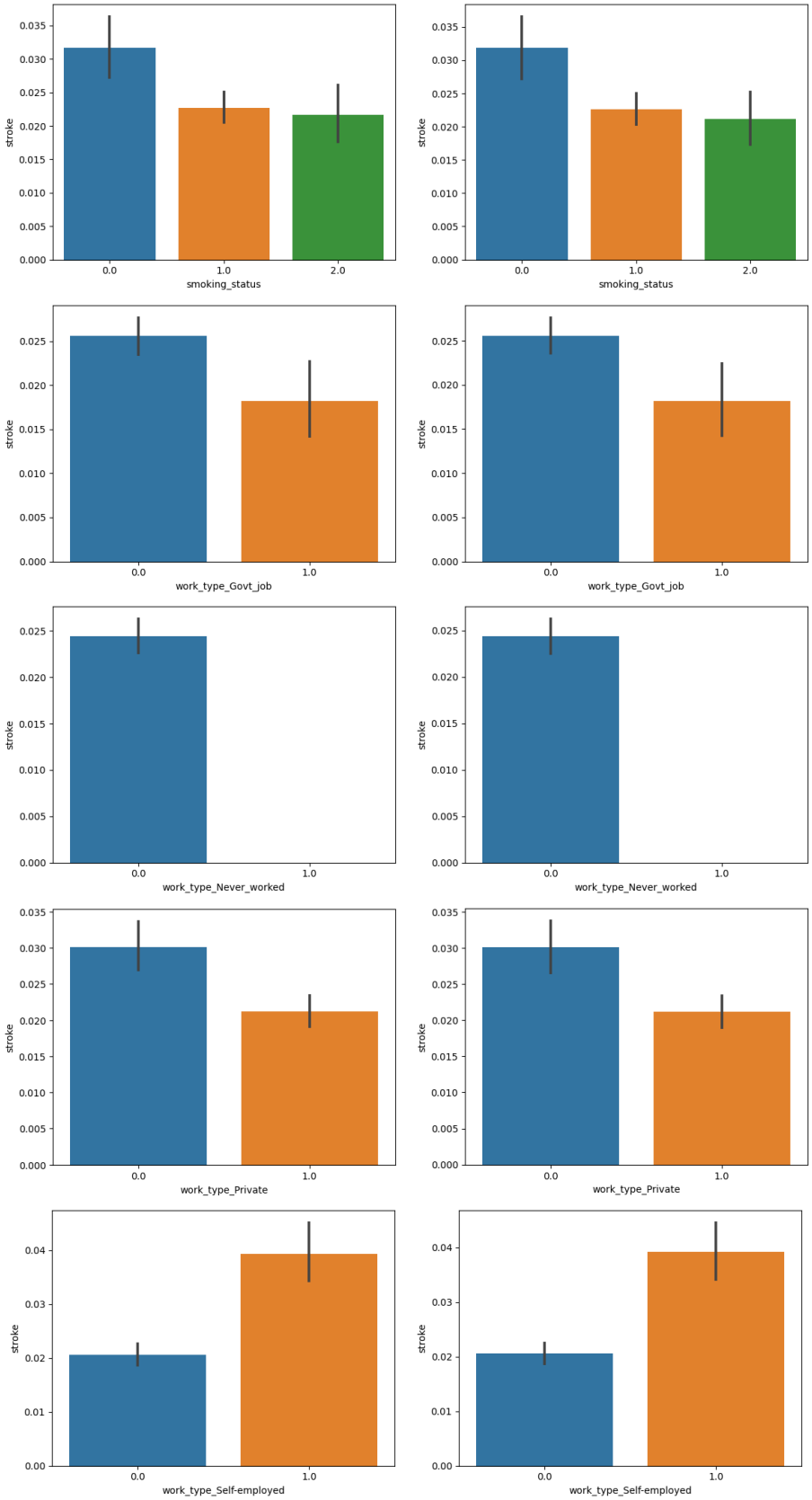

Figure 16: Bivariate Analysis of Categorical Variables VS Target Class[cont.] Mean Value and Most Frequent Imputed Data(Left) | KNN Imputed Data(Right)

Barplots by Fig. 15 and Fig. 16 showed that there is no significant difference between the Mean Value and Most Frequent imputed data to that of the KNN imputed data.

From the bar plots, it can be inferred that those who live on urban residences are more likely to have a stroke. The same can be said for those that are either have married or men. The disease seems to be common to those who either have hypertension or heart disease. People who have formerly smoked are more likely to have the disease than people who currently smokes and have never smoked. Data also suggested that those who have never worked or does not currently hold a government job or a private job are not likely to have a stroke than those who are self-employed.

B. Hyperparameter Tuning Results

The following tables are the hyperparameters which resulted from the hyperparameter tuning on the classifiers using GridSearch with 10-cross validation.

Classifier	Mean Value and Most Frequent Imputation	KNN Imputaion
Logistic Regression	C:0.01 solver: saga max_iter:1000	C:0.001 solver: sag max_iter:1000
Random Forest	max_features:sqrt min_samples_leaf:5 n_estimators:500	max_features:sqrt min_samples_leaf:5 n_estimators:10
Support Vector Machine	C:0.1 kernel:rbf	C:0.1 kernel:rbf
Multi-Layer Perceptron	activation:tanh solver:sgd	activation:relu solver:sgd
XGBoost	learning_rate:0.001 max_depth:1 min_child_weight:1	learning_rate:0.01 max_depth:1 min_child_weight:1
AdaBoost	learning_rate:0.1 n_estimators:50	learning_rate:0.01 n_estimators:500
KNN	metric:euclidean n_neighbors:17 weights:uniform	metric:euclidean n_neighbors:20 weights:uniform

Table 7: Hyperparameters Optimized w/ SMOTE + Feature Selection

Classifier	Mean Value and Most Frequent Imputation	KNN Imputaion
Logistic Regression	C:0.01 solver: saga max_iter:1000	C:0.001 solver: sag max_iter:1000
Random Forest	max_features:sqrt min_samples_leaf:5 n_estimators:10	max_features:sqrt min_samples_leaf:5 n_estimators:10
Support Vector Machine	C:0.1 kernel:rbf	C:0.1 kernel:rbf
Multi-Layer Perceptron	activation:tanh solver:sgd	activation:logistic solver:sgd
XGBoost	learning_rate:0.01 max_depth:1 min_child_weight:1	learning_rate:0.01 max_depth:1 min_child_weight:1
AdaBoost	learning_rate:0.1 n_estimators:50	learning_rate:0.01 n_estimators:500
KNN	metric:euclidean n_neighbors:17 weights:uniform	metric:euclidean n_neighbors:19 weights:uniform

Table 8: Hyperparameters Optimized w/ SMOTE-Tomek + Feature Selection

C. Performance Metrics

The table below displays the performance metrics at each model configuration.

Imbalance Handling	Imputation Technique	Metrics	LR	RF	SVM	MLP	XGB	ADA	KNN
SMOTE	Mean Value and Most Frequent Imputation	Accuracy	0.78	0.9	0.771	0.779	0.721	0.779	0.787
		Recall	0.7	0.34	0.72	0.7	0.78	0.68	0.567
		Specificity	0.781	0.91	0.772	0.781	0.72	0.781	0.79
		Precision	0.053	0.062	0.053	0.053	0.047	0.052	0.045
		f1 score	0.099	0.105	0.098	0.099	0.088	0.096	0.084
		AUC score	0.741	0.625	0.746	0.74	0.75	0.731	0.679
	KNN Imputation	Accuracy	0.778	0.942	0.771	0.779	0.721	0.769	0.822
		Recall	0.707	0.14	0.727	0.7	0.78	0.7	0.493
		Specificity	0.779	0.956	0.772	0.78	0.72	0.77	0.828
		Precision	0.053	0.053	0.053	0.053	0.047	0.051	0.048
		f1 score	0.099	0.077	0.099	0.098	0.088	0.095	0.087
		AUC score	0.743	0.548	0.749	0.74	0.75	0.735	0.661
SMOTE-Tomek	Mean Value and Most Frequent Imputation	Accuracy	0.779	0.887	0.763	0.776	0.721	0.779	0.786
		Recall	0.7	0.347	0.727	0.713	0.78	0.68	0.567
		Specificity	0.78	0.897	0.763	0.777	0.72	0.781	0.79
		Precision	0.053	0.056	0.051	0.053	0.047	0.052	0.045
		f1 score	0.099	0.096	0.096	0.099	0.088	0.096	0.084
		AUC score	0.74	0.622	0.745	0.745	0.75	0.731	0.678
	KNN Imputation	Accuracy	0.778	0.945	0.769	0.782	0.721	0.769	0.812
		Recall	0.707	0.133	0.72	0.693	0.78	0.7	0.533
		Specificity	0.779	0.959	0.77	0.783	0.72	0.77	0.817
		Precision	0.053	0.054	0.052	0.053	0.047	0.051	0.049
		f1 score	0.099	0.077	0.097	0.099	0.088	0.095	0.089
		AUC score	0.743	0.546	0.745	0.738	0.75	0.735	0.675

Table 9: Performance Metrics

D. Stroke Prediction System

Home Page

The Home Page is the first page the user sees which contains some information about stroke.

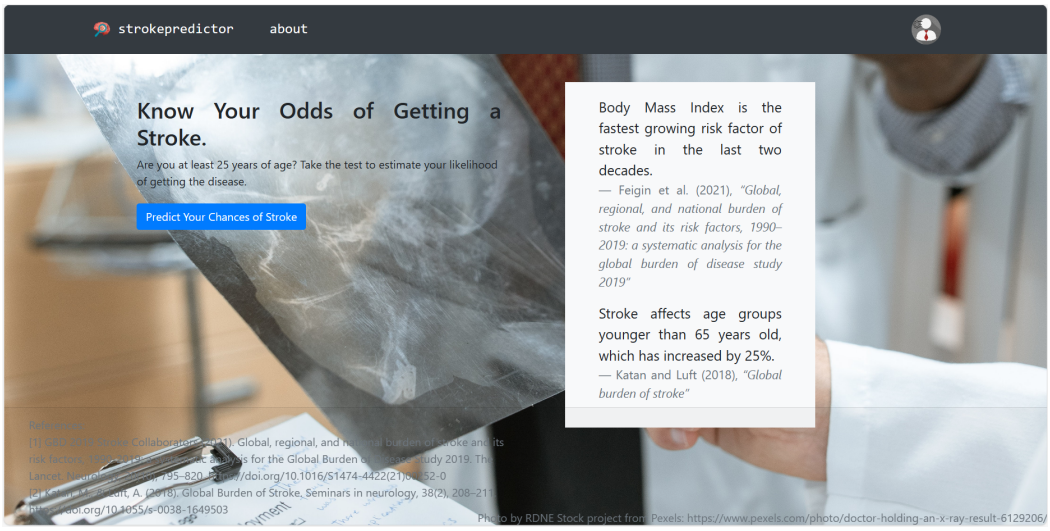

Figure 17: Home Page

Prediction Page

The Prediction Page shows all the fields the user needs to be filled out.

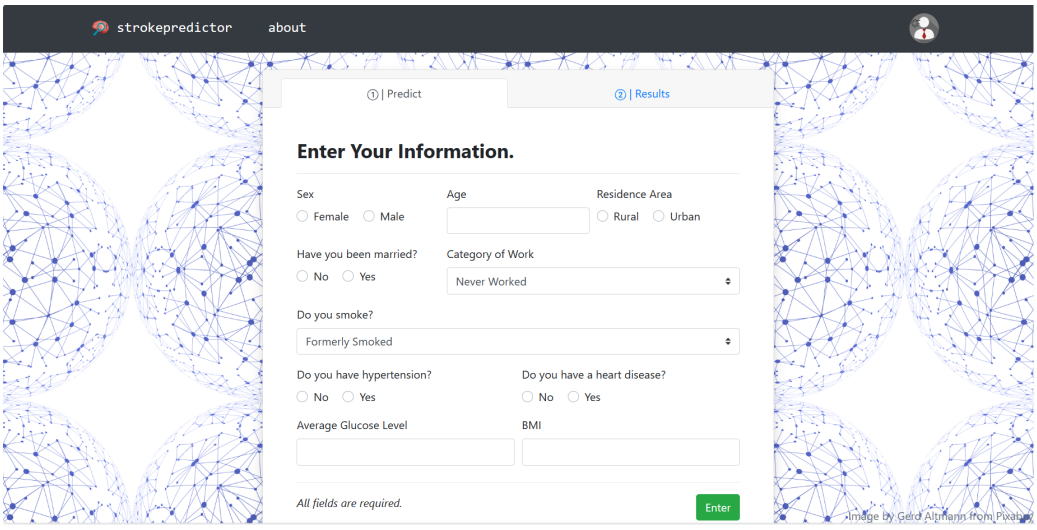

Figure 18: Prediction Page

Results Page

The Results Page shows the predicted result using the user's inputted details. The user will be redirected to this page after entering their information. They can also review what they inputted in the Prediction Page.

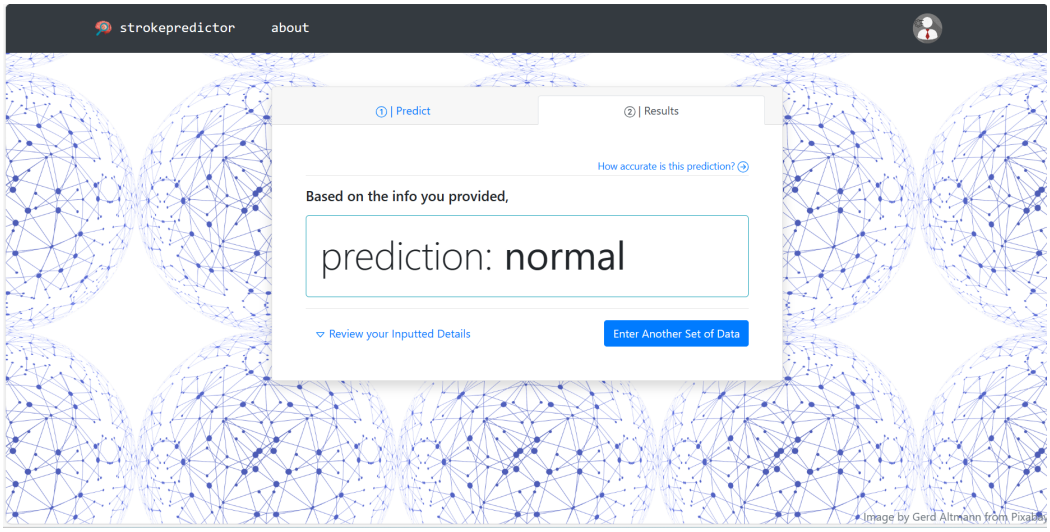

Figure 19: Results Page (Normal Result)

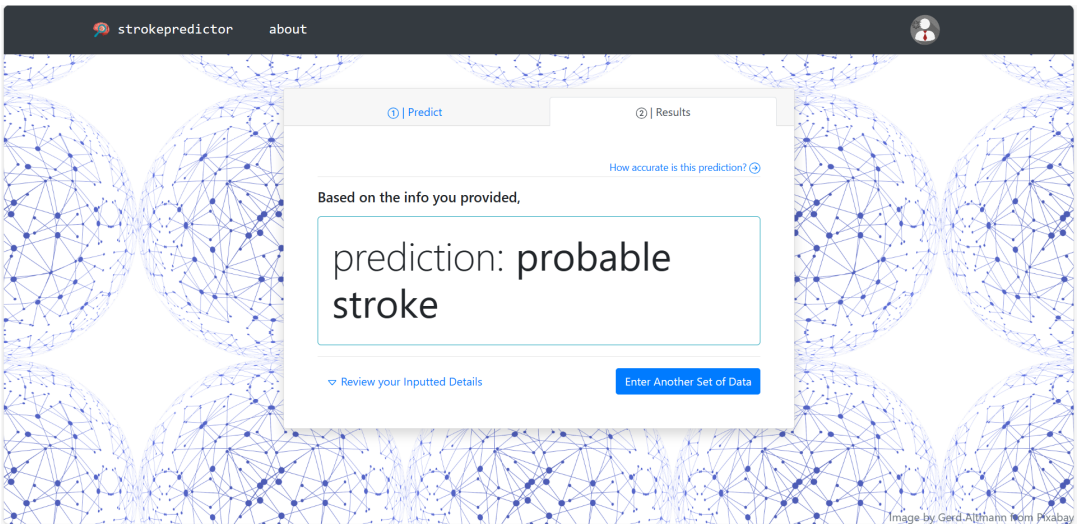

Figure 20: Results Page (Probable Stroke Result)

About Page

The About Page shows the details behind the system - the classifier used along with the pre-processing steps and parameters implemented in the model that was

integrated to the system. It also shows the performance metrics of the chosen model, the AUC-ROC plot, features selected and the dataset that made the model possible.

Figure 21: About Page

Admin Page

The Admin Page shows the system admin part of the webapp. It features a dashboard that shows the details inputted in the system.

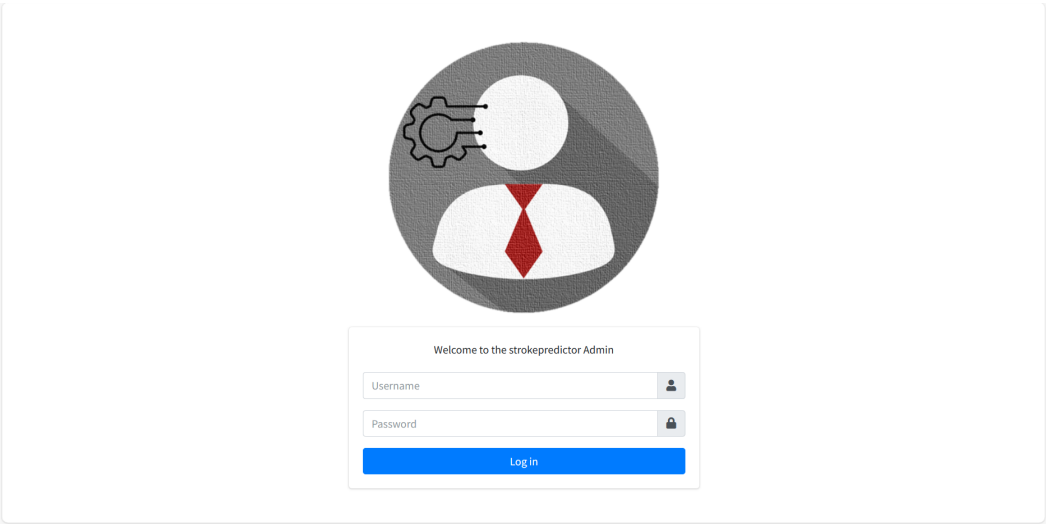

Figure 22: Admin Log In Page

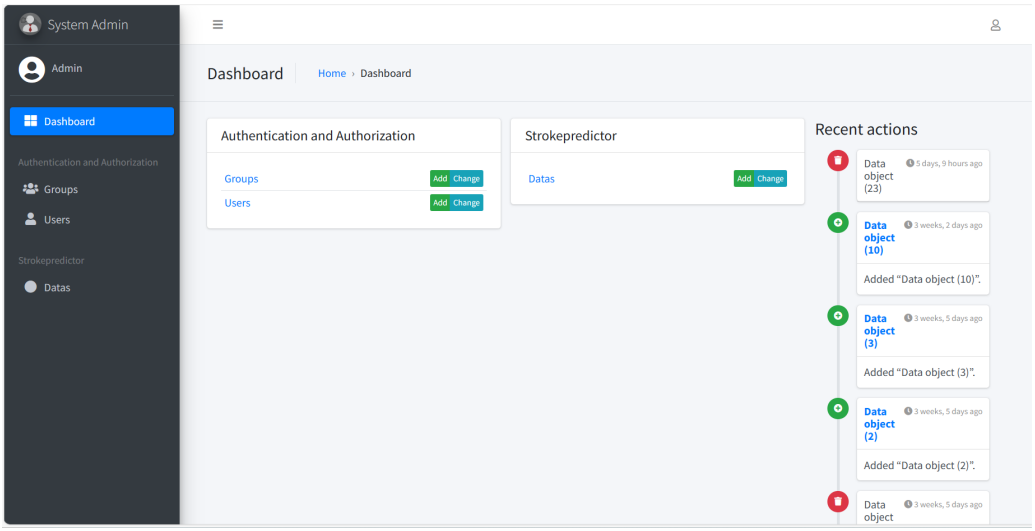

Figure 23: Admin Dashboard

VI. Discussions

There were 4 model configurations generated and evaluated for each of the classifiers: Logistic Regression, Random Forest, Support Vector Machine, Multilayer Perceptron, XGBoost, AdaBoost and KNN. The dataset from [13] was cleaned and pre-processed with OneHotEncoding, OrdinalEncoding and RobustScaling to make it applicable for model implementation. Pipelines were used for cross-validation to avoid data leakage[45]. These were only applied on the training set. Below illustrates the model configurations generated.

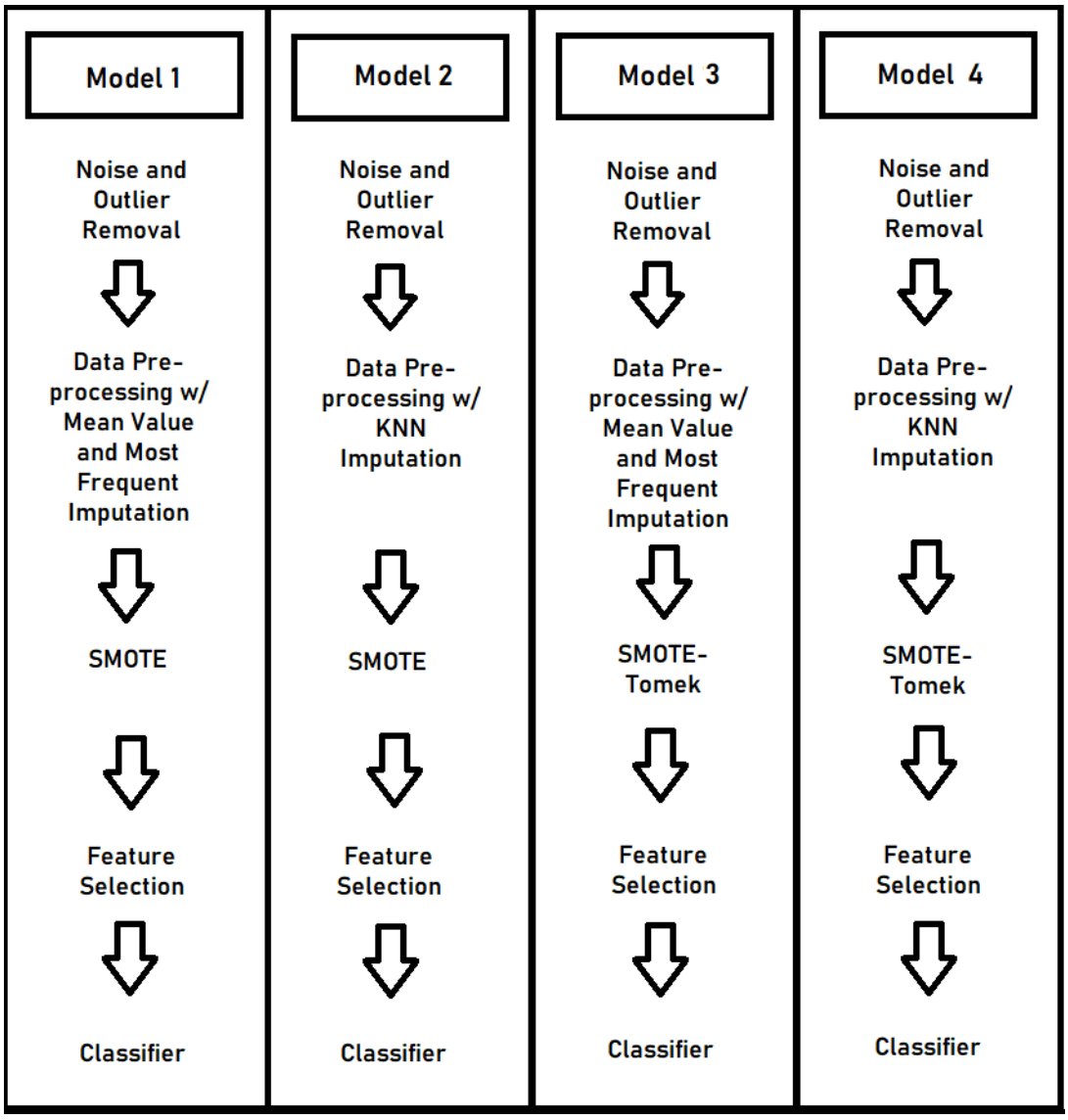

Figure 24: Model Configurations

The following tables discusses performance metrics of models with and without the applied techniques mentioned to see whether they have a significant influence or not.

Imbalance Handling	Imputation Technique	Metrics	LR	RF	SVM	MLP	XGB	ADA	KNN
None	Mean Value and Most Frequent Imputation	Accuracy	0.983	0.981	0.983	0.983	0.982	0.983	0.982
		Recall	0	0.013	0	0	0.007	0	0
		Specificity	1	0.998	1	1	0.999	1	0.999
		Precision	0	0.133	0	0	0.125	0	0
		f1 score	0	0.024	0	0	0.013	0	0
		AUC score	0.5	0.506	0.5	0.5	0.503	0.5	0.5
	KNN Imputation	Accuracy	0.983	0.982	0.983	0.983	0.982	0.983	0.982
		Recall	0	0.007	0	0	0	0	0
		Specificity	1	1	1	1	1	1	0.999
		Precision	0	0.2	0	0	0	0	0
		f1 score	0	0.013	0	0	0	0	0
		AUC score	0.5	0.503	0.5	0.5	0.5	0.5	0.5

Table 10: No Imbalance Handling

Table 10 displays metrics which resulted from classifiers without the application of imbalance handling. Here, the models without imbalance handling were set with default hyperparameters and generated favorable results in metrics accuracy and specificity however, at the same time, they generated less favorable results in metrics recall, precision and f1 score which is present across classifiers Logistic Regression, Support Vector Machine , Multilayer Perceptron and AdaBoost. Although other classifiers hold approximately a different estimate, their difference are low in value. The models also demonstrated weak performance through their AUC scores as it never significantly resulted beyond the value of 0.5. In addition, there seems to be no difference in metrics of classifiers applied with Mean Value and Most Frequent Imputation than that with KNN Imputation. Most of the metrics has values that mirror each other except some noticable ones which are

produced by Random Forest and XGBoost.

Imbalance Handling	Metrics	LR	RF	SVM	MLP	XGB	ADA	KNN
SMOTE	Accuracy	0.78	0.9	0.771	0.779	0.721	0.779	0.787
	Recall	0.7	0.34	0.72	0.7	0.78	0.68	0.567
	Specificity	0.781	0.91	0.772	0.781	0.72	0.781	0.79
	Precision	0.053	0.062	0.053	0.053	0.047	0.052	0.045
	f1 score	0.099	0.105	0.098	0.099	0.088	0.096	0.084
	AUC score	0.741	0.625	0.746	0.74	0.75	0.731	0.679
SMOTE-Tomek	Accuracy	0.779	0.887	0.763	0.776	0.721	0.779	0.786
	Recall	0.7	0.347	0.727	0.713	0.78	0.68	0.567
	Specificity	0.78	0.897	0.763	0.777	0.72	0.781	0.79
	Precision	0.053	0.056	0.051	0.053	0.047	0.052	0.045
	f1 score	0.099	0.096	0.096	0.099	0.088	0.096	0.084
	AUC score	0.74	0.622	0.745	0.745	0.75	0.731	0.678

Table 11: SMOTE VS SMOTE-Tomek (Mean Value and Most Frequent Imputed Data)

On the other hand, Table 11 shows metrics of the classifiers applied with SMOTE and SMOTE-Tomek on Mean Value and Most Frequent imputed data. It can be observed that despite the drop in accuracy and specificity, there is an improvement in other metrics when compared to the results showed in Table 10. Most of the metrics, however, showed that those applied with SMOTE has little to no difference with those applied with SMOTE-Tomek.

Imbalance Handling	Metrics	LR	RF	SVM	MLP	XGB	ADA	KNN
SMOTE	Accuracy	0.778	0.942	0.771	0.779	0.721	0.769	0.822
	Recall	0.707	0.14	0.727	0.7	0.78	0.7	0.493
	Specificity	0.779	0.956	0.772	0.78	0.72	0.77	0.828
	Precision	0.053	0.053	0.053	0.053	0.047	0.051	0.048
	f1 score	0.099	0.077	0.099	0.098	0.088	0.095	0.087
	AUC score	0.743	0.548	0.749	0.74	0.75	0.735	0.661
SMOTE-Tomek	Accuracy	0.778	0.945	0.769	0.782	0.721	0.769	0.812
	Recall	0.707	0.133	0.72	0.693	0.78	0.7	0.533
	Specificity	0.779	0.959	0.77	0.783	0.72	0.77	0.817
	Precision	0.053	0.054	0.052	0.053	0.047	0.051	0.049
	f1 score	0.099	0.077	0.097	0.099	0.088	0.095	0.089
	AUC score	0.743	0.546	0.745	0.738	0.75	0.735	0.675

Table 12: SMOTE VS SMOTE-Tomek (KNN Imputed Data)

Table 12 also exhibits the comparison of imbalance handling techniques using metrics except it was implemented on KNN imputed data. While the same observations as Table 11 can be seen in this table, Mean Value and Most Frequent imputed data showed that only XGBoost and AdaBoost had no difference in performance metrics between the two imbalance handling techniques in contrast with KNN imputed data which presented 3 classifiers - XGBoost, AdaBoost and Logistic Regression. In addition, when employed with SMOTE-Tomek, a portion of models such as Random Forest, Multilayer Perceptron and KNN showed a slight increase in some of their performance metrics and a decrease in all of the metrics of Support Vector Machine, albeit small.

Some similarities which are apparent on the Table 11 and 12 is that, out of all classifiers, Random Forest had the highest accuracy, specificity and precision metrics while XGBoost had the highest recall and AUC scores regardless of the

imbalance handling and imputation techniques. Furthermore, among classifiers, only AdaBoost and KNN showed no significant metrics compared to the other classifiers.

Imbalance Handling	Imputation Technique	Metrics	LR	RF	SVM	MLP	XGB	ADA	KNN
SMOTE w/o Feasture Selection	Mean Value and Most Frequent Imputation	Accuracy	0.78	0.925	0.768	0.779	0.721	0.759	0.825
		Recall	0.707	0.16	0.707	0.693	0.78	0.727	0.46
		Specificity	0.781	0.938	0.769	0.781	0.72	0.76	0.831
		Precision	0.054	0.044	0.051	0.053	0.047	0.05	0.046
		f1 score	0.1	0.068	0.095	0.098	0.088	0.094	0.083
		AUC score	0.744	0.549	0.738	0.737	0.75	0.743	0.646
	KNN Imputation	Accuracy	0.779	0.927	0.768	0.784	0.721	0.749	0.83
		Recall	0.707	0.167	0.707	0.693	0.78	0.72	0.447
		Specificity	0.781	0.94	0.769	0.786	0.72	0.749	0.837
		Precision	0.054	0.047	0.051	0.054	0.047	0.048	0.046
		f1 score	0.1	0.073	0.095	0.1	0.088	0.09	0.083
		AUC score	0.744	0.554	0.738	0.74	0.75	0.735	0.642
SMOTE w/ Feature Selection	Mean Value and Most Frequent Imputation	Accuracy	0.78	0.9	0.771	0.779	0.721	0.779	0.787
		Recall	0.7	0.34	0.72	0.7	0.78	0.68	0.567
		Specificity	0.781	0.91	0.772	0.781	0.72	0.781	0.79
		Precision	0.053	0.062	0.053	0.053	0.047	0.052	0.045
		f1 score	0.099	0.105	0.098	0.099	0.088	0.096	0.084
		AUC score	0.741	0.625	0.746	0.74	0.75	0.731	0.679
	KNN Imputation	Accuracy	0.778	0.942	0.771	0.779	0.721	0.769	0.822
		Recall	0.707	0.14	0.727	0.7	0.78	0.7	0.493
		Specificity	0.779	0.956	0.772	0.78	0.72	0.77	0.828
		Precision	0.053	0.053	0.053	0.053	0.047	0.051	0.048
		f1 score	0.099	0.077	0.099	0.098	0.088	0.095	0.087
		AUC score	0.743	0.548	0.749	0.74	0.75	0.735	0.661

Table 13: Comparison of SMOTE with and without Feature Selection

Table 13 differentiated whether feature selection had a notable effect on the

metrics of the SMOTE applied models. Although not distinct, there is a slight increase in all of the metrics of Support Vector Machine in both imputed data. There is also an inconsiderable increase in some of the performance metrics of Random Forest, Multilayer Perceptron, AdaBoost and KNN and decrease in some of the performance metrics of Logistic Regression. The performance metrics of XGBoost remains the same with or without feature selection in SMOTE applied models.

Table 14 also described the difference of the performance metrics of SMOTE-Tomek applied models with and without the utilization of feature selection. It also produced similar observations as Table 13 wherein there is minimal increase in some performance metrics of most of the models except that of Logistic Regression which showed the opposite behavior and XGBoost, which remained the same values.

Imbalance Handling	Imputation Technique	Metrics	LR	RF	SVM	MLP	XGB	ADA	KNN
SMOTE-Tomek w/o Feature Selection	Mean Value and Most Frequent Imputation	Accuracy	0.78	0.929	0.769	0.784	0.721	0.759	0.824
		Recall	0.707	0.193	0.707	0.693	0.78	0.733	0.467
		Specificity	0.781	0.942	0.77	0.785	0.72	0.759	0.83
		Precision	0.054	0.055	0.051	0.054	0.047	0.051	0.046
		f1 score	0.1	0.086	0.095	0.1	0.088	0.095	0.084
		AUC score	0.744	0.568	0.738	0.739	0.75	0.746	0.648
	KNN Imputation	Accuracy	0.779	0.929	0.768	0.78	0.721	0.752	0.825
		Recall	0.707	0.147	0.707	0.693	0.78	0.713	0.46
		Specificity	0.781	0.943	0.77	0.782	0.72	0.753	0.832
		Precision	0.054	0.043	0.051	0.053	0.047	0.048	0.046
		f1 score	0.1	0.067	0.095	0.098	0.088	0.09	0.083
		AUC score	0.744	0.545	0.738	0.738	0.75	0.733	0.646
SMOTE-Tomek w/ Feature Selection	Mean Value and Most Frequent Imputation	Accuracy	0.779	0.887	0.763	0.776	0.721	0.779	0.786
		Recall	0.7	0.347	0.727	0.713	0.78	0.68	0.567
		Specificity	0.78	0.897	0.763	0.777	0.72	0.781	0.79
		Precision	0.053	0.056	0.051	0.053	0.047	0.052	0.045
		f1 score	0.099	0.096	0.096	0.099	0.088	0.096	0.084
		AUC score	0.74	0.622	0.745	0.745	0.75	0.731	0.678
	KNN Imputation	Accuracy	0.778	0.945	0.769	0.782	0.721	0.769	0.812
		Recall	0.707	0.133	0.72	0.693	0.78	0.7	0.533
		Specificity	0.779	0.959	0.77	0.783	0.72	0.77	0.817
		Precision	0.053	0.054	0.052	0.053	0.047	0.051	0.049
		f1 score	0.099	0.077	0.097	0.099	0.088	0.095	0.089
		AUC score	0.743	0.546	0.745	0.738	0.75	0.735	0.675

Table 14: Comparison of SMOTE-Tomek with and without Feature Selection

After considering the performance metrics, the classifier chosen to be integrated in the application is XGBoost since it showed the highest AUC value, the metric that determines the discriminating performance of the model [46],among the classifiers. Since there are four model configurations which showed the same per-

formance metrics, the cross-validation scores were compared since it is the average of the metric's iteration results[47].

Model 1	Model 2	Model 3	Model 4
0.720011899	0.720012845	0.720012845	0.720012845

Table 15: Comparison of Cross-Validation Scores of XGBoost Among Model Configurations

As model configurations 2, 3 and 4 had the same CV scores, any of them can be used for the model integration. In the end, model configuration 2 with classifier XGBoost was integrated in the application. Below shows the ROC-AUC plot that illustrates the chosen model's classification performance.

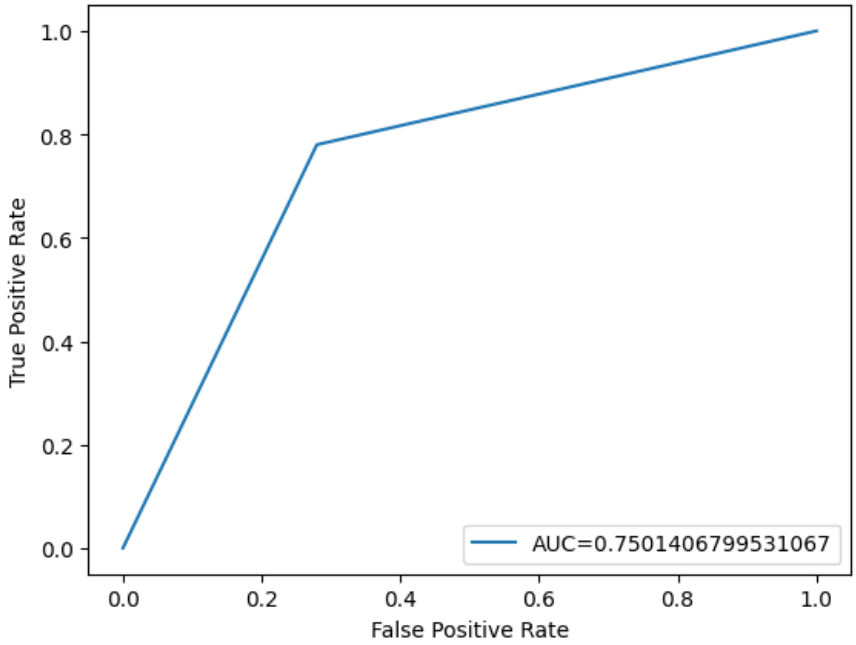

Figure 25: AUCROC Plot

To better understand the different metrics generated by the confusion matrix, the following figures illustrate examples of each using some sample data from the test set.

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
37883	Male	52	0	0	Yes	Private	Rural	61.88	26.5	smokes

Actual: 0 | Predicted: 0 | True Negative

Figure 26: True Negative Example

The figure above shows that the predicted value of the sample data is '0' or non-stroke. This corresponds with the actual value of the data which also has the same outcome class hence it is considered a True Negative.

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
23479	Male	70	1	1	Yes	Private	Urban	197.43	31.4	smokes

Actual: 1 | Predicted: 1 | True Positive

Figure 27: True Positive Example

The figure above shows that the predicted value of the sample data is '1' or stroke. This corresponds with the actual value of the data which also has the same outcome class hence it is considered a True Positive.

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
20297	Female	74	0	0	Yes	Private	Rural	66.02	25.6	never smoked

Actual: 0 | Predicted: 1 | False Positive

Figure 28: False Positive Example

The figure above shows that the predicted value of the sample data is '1' or stroke. This does not correspond with the actual value of the data which has the outcome class '0' or non-stroke hence it is considered a False Positive.

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status
12827	Female	48	0	0	Yes	Private	Urban	74.11	20.5	never smoked

Actual: 1 | Predicted: 0 | False Negative

Figure 29: False Negative Example

Lastly, the figure above shows that the predicted value of the sample data is '0' or non-stroke. This does not correspond with the actual value of the data which has the outcome class '1' or stroke hence it is considered a False Negative.

Both Figures 26 and 27 represent the correct prediction of the model with reference to the actual data while Figures 28 and 29 represent the error made by the model, having done the opposite. The figure below indicates the confusion matrix generated by the model along with the classification report done on the testing set.

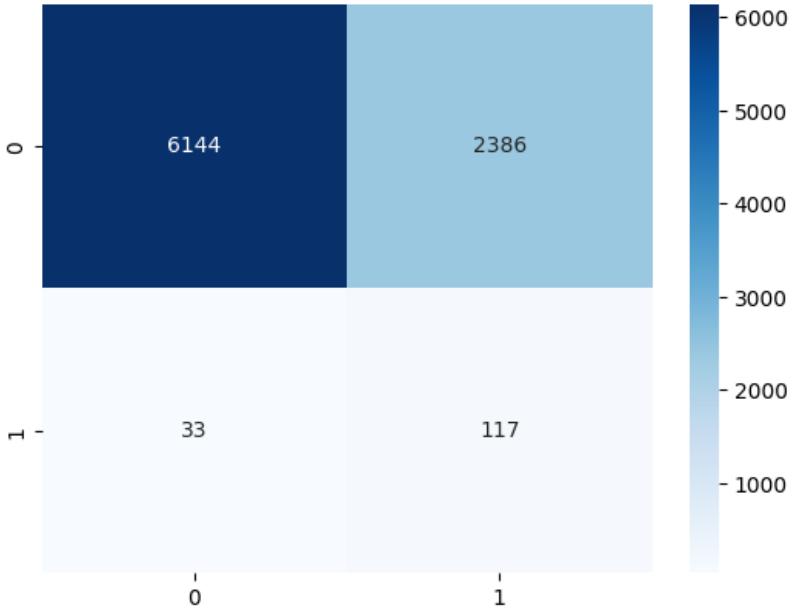

Figure 30: Confusion Matrix

	precision	recall	f1-score	support
0	0.72	0.99	0.84	6177
1	0.78	0.05	0.09	2503
accuracy			0.72	8680
macro avg	0.75	0.52	0.46	8680
weighted avg	0.74	0.72	0.62	8680

Figure 31: Classification Report

To summarize, Fig. 30 showed that the model yielded high True Negatives

(TN) followed by False Positives(FP) then True Positives(TP) and lastly False Negatives(FN). This means that the model has correctly identified values which are actually not stroke however, at the same time, the model has incorrectly identified values which were actually not stroke as stroke. This is reflected on Fig. 31 where it can be seen that the f1 score and recall are very low in value. In the test set, 6177 are actually not stroke while 2503 are stroke values and the report states that the model correctly predicted only about 5% of the actual stroke values which affected the f1 score. On the other hand, from the predicted stroke values, the model correctly identified the actual stroke values about 78%. Although the model did poorly on predicting whether a person has a stroke or not, False Negatives are less in value, which meant that the outcomes where the model incorrectly predicts a stroke as not stroke also reduced as well.

Since there is feature selection applied in the mentioned model configuration, we can determine which features were selected by the ExtraTreesClassifier.

Smoking Status
BMI
Age
Average Glucose Level

Table 16: Selected Features

It can be seen that Body Mass Index or BMI, Smoking Status and Average Glucose Level are included in the selected features by the model configuration. This suggests that the features have a significant influence on the model and as the latter feature is used to determine diabetes [48], this further supports the insights by [3] where they were considered leading risk factors of stroke. Although hypertension is also considered as one along with these factors, it may be due to the threshold set which limited the selection of the feature. Age is also mentioned to be one of the determining factors of stroke as young adults are more susceptible to the disease [6]. The selected features also reflects the significant risk factors

for stroke yielded by studies [22] and [23] wherein similar factors are age, diabetes and the latter paper adding the smoking aspect as well. However, at the same time, these studies also stipulated that the feature hypertension also indicates the chances of stroke which was overlooked by the model.

When considering the relationship of features according to Fig 9 and Fig. 10, as age has a positive correlation with average glucose level, this means that, as a person ages, the average glucose level also increases. In addition to that, average glucose level also has a positive correlation with bmi values, which can suggest that an increase in average glucose level also indicates an increase in bmi values. This supports both figures Fig. 11 and Fig. 12 wherein stroke datapoints are more visible as age increases. Likewise, stroke values occur more on people who have formerly smoked, as illustrated in Fig. 16.

Finally, the performance metrics of the study were compared to the best model generated by other studies, specifically with [13] since it is the study that utilized the same dataset. However, since their study emphasized the use of FN and FP rate, the False Positive (FP) and False Negative(FN) rate were calculated using the following formulas and then compared:

$$FP_{rate} = \frac{FP}{FP+TN} \qquad FN_{rate} = \frac{FN}{FN+TP}$$

	Dataset	Accuracy	Sensitivity (Recall)	AUC	FP Rate	FN Rate
Our Study	43,400	76.9	78	75	27.87	22
[13]	43,400	71.6	67.4	-	33.1	19.1
[15]	10,000	-	92	94	-	-
[14]	5,110	86	-	84	-	-

Table 17: Comparison of Studies (Percentage)

It can be seen in Table 17 that the last two papers bested the study in terms of AUC scores, with the former paper being better than the study in recall and the latter paper being better in the accuracy. Although they have the same features,

they have different number of rows and techniques applied. As it happens, these two employed the use of neural networks. However, when compared to [13], the study has the highest acccuracy and recall. Although the study generated high FN rate, the study had an upper hand on the FP rate which is low in value.

It was also discovered that the noise and outlier removal method used by the study were not appropriate to be applied in the dataset. Before the application of machine learning methods, the dataset were split into training and testing set. The training set which will be processed had its noise and outliers removed using the techniques mentioned in [13]. However, by following this procedure, specifically removing rows with age < 25, approximately 25% of the rows were removed (Table 4). As the number of rows dismissed were too large to be considered negligible, it can be said that the resulting data used in the model has assumed bias.

VII. Conclusions

The Stroke Predictions System produced a web application that predicts whether a user has a probable stroke or not with the use of a predictive model derived from a dataset taken from [43], which were cleaned and pre-processed along with KNN imputation and the imbalance present in the data was handled with SMOTE. Since it uses a pipeline, it automatically selects which features are significant before implemented on the XGBClassifier. From the results, it can be seen that,

1. Imbalance handling had a significant impact on the performance metrics of the classifiers. Although there seems to be minimal difference between the effect of SMOTE and SMOTE-Tomek, there has been an improvement in the metrics Recall, Precision, f1 score and AUC scores when applied with an imbalance handling technique.

2. There is no significant effect between the application of Mean Value and Most Frequent Imputation and KNN Imputation however results showed that there has been an increase in data imputed by KNN when applied with imbalance handling and feature selection, albeit small.

3. Although results showed that there has been a slight increase in the performance metrics of some models, the application of feature selection do not show any significant effect, regardless of the imputation or the imbalance handling technique implemented.

4. Among classifiers, XGBoost showed consistent values regardless of the model configuration having the highest recall of 0.78 and AUC score of 0.75 which makes it the model suitable for integration with the web application.

VIII. Recommendations

The stroke prediction system can be further improved if the models having values near estimates with the AUC value of 0.75 were further hyperparameter tuned. It would also be better if a different feature engineering or an outlier removal technique was applied and tested on the dataset along with different training and test set to see if there is a significant difference. Since there are a few features, it would be best not to apply feature selection. Additionally, it is a good practice to experiment and compare other imputation and imbalance handling techniques with the ones defined in this study.

For the web application, it would be better if there is an interactive data training making it possible for the system admin to customize the model to be used for prediction and deployed so that it can be used without depending on the local.

IX. Bibliography

[1] World Health Organization and others, "Noncommunicable diseases." https://www.who.int/health-topics/noncommunicable-diseases#tab=tab_1.

[2] Institute of Health Metrics, "GBD Compare | IHME Viz Hub." http://vizhub.healthdata.org/gbd-compare.

[3] V. L. Feigin, B. A. Stark, C. O. Johnson, G. A. Roth, C. Bisignano, G. G. Abady, M. Abbasifard, M. Abbasi-Kangevari, F. Abd-Allah, V. Abedi, A. Abualhasan, N. M. Abu-Rmeileh, A. I. Abushouk, O. M. Adebayo, G. Agarwal, P. Agasthi, B. O. Ahinkorah, S. Ahmad, S. Ahmadi, Y. A. Salih, B. Aji, S. Akbarpour, R. O. Akinyemi, H. A. Hamad, F. Alahdab, S. M. Alif, V. Alipour, S. M. Aljunid, S. Almustanyir, R. M. Al-Raddadi, R. A.-S. Salman, N. Alvis-Guzman, R. Ancuceanu, D. Anderlini, J. A. Anderson, A. Ansar, I. C. Antonazzo, J. Arabloo, J. Ärnlöv, K. D. Artanti, Z. Aryan, S. Asgari, T. Ashraf, M. Athar, A. Atreya, M. Ausloos, A. A. Baig, O. C. Baltatu, M. Banach, M. A. Barboza, S. L. Barker-Collo, T. W. Bärnighausen, M. T. U. Barone, S. Basu, G. Bazmandegan, E. Beghi, M. Beheshti, Y. Béjot, A. W. Bell, D. A. Bennett, I. M. Bensenor, W. M. Bezabhe, Y. M. Bezabih, A. S. Bhagavathula, P. Bhardwaj, K. Bhattacharyya, A. Bijani, B. Bikbov, M. M. Birhanu, A. Boloor, A. Bonny, M. Brauer, H. Brenner, D. Bryazka, Z. A. Butt, F. L. C. dos Santos, I. R. Campos-Nonato, C. Cantu-Brito, J. J. Carrero, C. A. Castañeda-Orjuela, A. L. Catapano, P. A. Chakraborty, J. Charan, S. G. Choudhari, E. K. Chowdhury, D.-T. Chu, S.-C. Chung, D. Colozza, V. M. Costa, S. Costanzo, M. H. Criqui, O. Dadras, B. Dagnew, X. Dai, K. Dalal, A. A. M. Damasceno, E. D'Amico, L. Dandona, R. Dandona, J. D. Gela, K. Davletov, V. D. la Cruz-Góngora, R. Desai, D. Dhamnetiya, S. D. Dharmaratne, M. L. Dhimal, M. Dhimal, D. Diaz, M. Dichgans, K. Dokova, R. Doshi, A. Douiri, B. B. Duncan, S. Eftekharzadeh, M. Ekholuenetale, N. E. Nahas, I. Y. Elgendy, M. El-

hadi, S. I. El-Jaafary, M. Endres, A. Y. Endries, D. A. Erku, E. J. A. Faraon, U. Farooque, F. Farzadfar, A. H. Feroze, I. Filip, F. Fischer, D. Flood, M. M. Gad, S. Gaidhane, R. G. Gheshlagh, A. Ghashghaee, N. Ghith, G. Ghozali, S. Ghozy, A. Gialluisi, S. Giampaoli, S. A. Gilani, P. S. Gill, E. V. Gnedovskaya, M. Golechha, A. C. Goulart, Y. Guo, R. Gupta, V. B. Gupta, V. K. Gupta, P. Gyanwali, N. Hafezi-Nejad, S. Hamidi, A. Hanif, G. J. Hankey, A. Hargono, A. Hashi, T. S. Hassan, H. Y. Hassen, R. J. Havmoeller, S. I. Hay, K. Hayat, M. I. Hegazy, C. Herteliu, R. Holla, S. Hostiuc, M. Househ, J. Huang, A. Humayun, B.-F. Hwang, L. Iacoviello, I. Iavicoli, S. E. Ibitoye, O. S. Ilesanmi, I. M. Ilic, M. D. Ilic, U. Iqbal, S. S. N. Irvani, S. M. S. Islam, N. E. Ismail, H. Iso, G. Isola, M. Iwagami, L. Jacob, V. Jain, S.-I. Jang, S. K. Jayapal, S. Jayaram, R. Jayawardena, P. Jeemon, R. P. Jha, W. D. Johnson, J. B. Jonas, N. Joseph, J. J. Jozwiak, M. Jürisson, R. Kalani, R. Kalhor, Y. Kalkonde, A. Kamath, Z. Kamiab, T. Kanchan, H. Kandel, A. Karch, P. D. Katoto, G. A. Kayode, P. Keshavarz, Y. S. Khader, E. A. Khan, I. A. Khan, M. Khan, M. A. Khan, M. N. Khatib, J. Khubchandani, G. R. Kim, M. S. Kim, Y. J. Kim, A. Kisa, S. Kisa, M. Kivimäki, D. Kolte, A. Koolivand, S. L. K. Laxminarayana, A. Koyanagi, K. Krishan, V. Krishnamoorthy, R. V. Krishnamurthi, G. A. Kumar, D. Kusuma, C. L. Vecchia, B. Lacey, H. M. Lak, T. Lallukka, S. Lasrado, P. M. Lavados, M. Leonardi, B. Li, S. Li, H. Lin, R.-T. Lin, X. Liu, W. D. Lo, S. Lorkowski, G. Lucchetti, R. L. Saute, H. M. A. E. Razek, F. G. Magnani, P. B. Mahajan, A. Majeed, A. Makki, R. Malekzadeh, A. A. Malik, N. Manafi, M. A. Mansournia, L. G. Mantovani, S. Martini, G. Mazzaglia, M. M. Mehndiratta, R. G. Menezes, A. Meretoja, A. G. Mersha, J. M. Jonasson, B. Miazgowski, T. Miazgowski, I. M. Michalek, E. M. Mirrakhimov, Y. Mohammad, A. Mohammadian-Hafshejani, S. Mohammed, A. H. Mokdad, Y. Mokhayeri, M. Molokhia, M. A. Moni, A. A. Montasir, R. Moradzadeh, L. Morawska, J. Morze, W. Muruet, K. I. Musa, A. J. Nagarajan, M. Naghavi, S. N. Swamy, B. R. Nascimento, R. I. Negoi,

S. N. Kandel, T. H. Nguyen, B. Norrving, J. J. Noubiap, V. E. Nwatah, B. Oancea, O. O. Odukoya, A. T. Olagunju, H. Orru, M. O. Owolabi, J. R. Padubidri, A. Pana, T. Parekh, E.-C. Park, F. P. Kan, M. Pathak, M. F. P. Peres, A. Perianayagam, T.-M. Pham, M. A. Piradov, V. Podder, S. Polinder, M. J. Postma, A. Pourshams, A. Radfar, A. Rafiei, A. Raggi, F. Rahim, V. Rahimi-Movaghar, M. Rahman, M. A. Rahman, A. M. Rahmani, N. Rajai, P. Ranasinghe, C. R. Rao, S. J. Rao, P. Rathi, D. L. Rawaf, S. Rawaf, M. B. Reitsma, V. Renjith, A. M. N. Renzaho, A. Rezapour, J. A. B. Rodriguez, L. Roever, M. Romoli, A. Rynkiewicz, S. Sacco, M. Sadeghi, S. S. Moghaddam, A. Sahebkar, K. Saif-Ur-Rahman, R. Salah, M. Samaei, A. M. Samy, I. S. Santos, M. M. Santric-Milicevic, N. Sarrafzadegan, B. Sathian, D. Sattin, S. Schiavolin, M. P. Schlaich, M. I. Schmidt, A. E. Schutte, S. G. Sepanlou, A. Seylani, F. Sha, S. Shahabi, M. A. Shaikh, M. Shannawaz, M. S. R. Shawon, A. Sheikh, S. Sheikhbahaei, K. Shibuya, S. Siabani, D. A. S. Silva, J. A. Singh, J. K. Singh, V. Y. Skryabin, A. A. Skryabina, B. H. Sobaih, S. Stortecky, S. Stranges, E. G. Tadesse, I. U. Tarigan, M.-H. Temsah, Y. Teuschl, A. G. Thrift, M. Tonelli, M. R. Tovani-Palone, B. X. Tran, M. Tripathi, G. W. Tsegaye, A. Ullah, B. Unim, B. Unnikrishnan, A. Vakilian, S. V. Tahbaz, T. J. Vasankari, N. Venketasubramanian, D. Vervoort, B. Vo, V. Volovici, K. Vosoughi, G. T. Vu, L. G. Vu, H. A. Wafa, Y. Waheed, Y. Wang, T. Wijeratne, A. S. Winkler, C. D. A. Wolfe, M. Woodward, J. H. Wu, S. W. Hanson, X. Xu, L. Yadav, A. Yadollahpour, S. H. Y. Jabbari, K. Yamagishi, H. Yatsuya, N. Yonemoto, C. Yu, I. Yunusa, M. S. Zaman, S. B. Zaman, M. Zamanian, R. Zand, A. Zandifar, M. S. Zastrozhin, A. Zastrozhina, Y. Zhang, Z.-J. Zhang, C. Zhong, Y. M. H. Zuniga, and C. J. L. Murray, "Global, regional, and national burden of stroke and its risk factors, 1990–2019: a systematic analysis for the global burden of disease study 2019," *The Lancet Neurology*, vol. 20, pp. 795–820, Oct. 2021.

[4] J. D. Pandian, S. L. Gall, M. P. Kate, G. S. Silva, R. O. Akinyemi, B. I.

Ovbiagele, P. M. Lavados, D. B. C. Gandhi, and A. G. Thrift, "Prevention of stroke: a global perspective," *The Lancet*, vol. 392, pp. 1269–1278, Oct. 2018.

[5] M. Katan and A. Luft, "Global burden of stroke," *Seminars in Neurology*, vol. 38, pp. 208–211, Apr. 2018.

[6] D. Smajlovic, "Strokes in young adults: epidemiology and prevention," *Vascular Health and Risk Management*, p. 157, Feb. 2015.

[7] M. J. O'Donnell, S. L. Chin, S. Rangarajan, D. Xavier, L. Liu, H. Zhang, P. Rao-Melacini, X. Zhang, P. Pais, S. Agapay, P. Lopez-Jaramillo, A. Damasceno, P. Langhorne, M. J. McQueen, A. Rosengren, M. Dehghan, G. J. Hankey, A. L. Dans, A. Elsayed, A. Avezum, C. Mondo, H.-C. Diener, D. Ryglewicz, A. Czlonkowska, N. Pogosova, C. Weimar, R. Iqbal, R. Diaz, K. Yusoff, A. Yusufali, A. Oguz, X. Wang, E. Penaherrera, F. Lanas, O. S. Ogah, A. Ogunniyi, H. K. Iversen, G. Malaga, Z. Rumboldt, S. Oveisgharan, F. A. Hussain, D. Magazi, Y. Nilanont, J. Ferguson, G. Pare, and S. Yusuf, "Global and regional effects of potentially modifiable risk factors associated with acute stroke in 32 countries (INTERSTROKE): a case-control study," *The Lancet*, vol. 388, pp. 761–775, Aug. 2016.

[8] S. L. Crichton, B. D. Bray, C. McKevitt, A. G. Rudd, and C. D. Wolfe, "Patient outcomes up to 15 years after stroke: survival, disability, quality of life, cognition and mental health," *Journal of Neurology, Neurosurgery & Psychiatry*, vol. 87, no. 10, pp. 1091–1098, 2016.

[9] H. Moawad, "7 tests that measure your stroke risk." https://www.verywellhealth.com/simple-tests-that-measure-your-stroke-risk-4020539, May 2021.

[10] Harvard Health Publishing Harvard Medical School, "Is there an early warning test for stroke?." https://www.health.harvard.edu/heart-health/is-there-an-early-warning-test-for-stroke, Sept. 2016.

[11] Y. V. Kalkonde, S. Alladi, S. Kaul, and V. Hachinski, "Stroke prevention strategies in the developing world," *Stroke*, vol. 49, pp. 3092–3097, Dec. 2018.

[12] M. V. Collantes, Y. H. Zuñiga, C. N. Granada, D. R. Uezono, L. C. D. Castillo, C. G. Enriquez, K. D. Ignacio, S. D. Ignacio, and R. D. Jamora, "Current state of stroke care in the philippines," *Frontiers in Neurology*, vol. 12, Aug. 2021.

[13] T. Liu, W. Fan, and C. Wu, "A hybrid machine learning approach to cerebral stroke prediction based on imbalanced medical dataset," *Artificial Intelligence in Medicine*, vol. 101, p. 101723, Nov. 2019.

[14] C. Rana, N. Chitre, B. Poyekar, and P. Bide, "Stroke prediction using smote-tomek and neural network," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, July 2021.

[15] P. A, N. G, V. K. R, P. P, and S. R. R.V.T, "Stroke prediction system using artificial neural network," in *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, pp. 1898–1902, 2021.

[16] A. Kashyap, "Precision and recall — a simplified view." https://towardsdatascience.com/precision-and-recall-a-simplified-view-bc25978d81e6, Dec 2019.

[17] Y. Wu and Y. Fang, "Stroke prediction with machine learning methods among older chinese," *International Journal of Environmental Research and Public Health*, vol. 17, p. 1828, Mar. 2020.

[18] E. M. Alanazi, A. Abdou, and J. Luo, "Predicting risk of stroke from lab tests using machine learning algorithms: Development and evaluation of prediction models," *JMIR Formative Research*, vol. 5, p. e23440, Dec. 2021.

[19] Ferdib-Al-Islam and M. Ghosh, "An enhanced stroke prediction scheme using SMOTE and machine learning techniques," in *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, IEEE, July 2021.

[20] T. Tazin, M. N. Alam, N. N. Dola, M. S. Bari, S. Bourouis, and M. M. Khan, "Stroke disease detection and prediction using robust learning approaches," *Journal of Healthcare Engineering*, vol. 2021, pp. 1–12, Nov. 2021.

[21] H. Puri, J. Chaudhary, K. R. Raghavendra, R. Mantri, and K. Bingi, "Prediction of heart stroke using support vector machine algorithm," in *2021 8th International Conference on Smart Computing and Communications (ICSCC)*, IEEE, July 2021.

[22] M. Chun, R. Clarke, B. J. Cairns, D. Clifton, D. Bennett, Y. Chen, Y. Guo, P. Pei, J. Lv, C. Yu, L. Yang, L. Li, Z. Chen, and T. Z. and, "Stroke risk prediction using machine learning: a prospective cohort study of 0.5 million chinese adults," *Journal of the American Medical Informatics Association*, vol. 28, pp. 1719–1727, May 2021.

[23] J. woo Lee, H. sun Lim, D. wook Kim, S. ae Shin, J. Kim, B. Yoo, and K. hee Cho, "The development and implementation of stroke risk prediction model in national health insurance service's personal health record," *Computer Methods and Programs in Biomedicine*, vol. 153, pp. 253–257, Jan. 2018.

[24] National Heart, Lung and Blood Institute, "Stroke." https://www.nhlbi.nih.gov/health-topics/stroke.

[25] A. K. Boehme, C. Esenwa, and M. S. Elkind, "Stroke risk factors, genetics, and prevention," *Circulation Research*, vol. 120, pp. 472–495, Feb. 2017.

[26] S. Santosh, "Data science in healthcare." https://www.analyticsvidhya.com/blog/2021/05/data-science-in-healthcare/, May 2021.

[27] Tutorialspoint, "Machine learning - logistic regression." https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_logistic_regression.htm.

[28] T. Yiu, "Understanding random forest." https://towardsdatascience.com/understanding-random-forest-58381e0602d2, Sep 2021.

[29] R. Gandhi, "Support vector machine — introduction to machine learning algorithms." https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444f Jul 2018.

[30] S. Awasthi, "Seven most popular svm kernels." https://dataaspirant.com/svm-kernels/, Dec 2020.

[31] A. Navlani, "Support vector machines with scikit-learn." https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python, Dec 2019.

[32] S. K. Agrawal, "Understanding the basics of artificial neural network." https://www.analyticsvidhya.com/blog/2021/07/understanding-the-basics-of-artificial-neural-network-ann/, Jul 2021.

[33] C. Bento, "Multilayer perceptron explained with a real-life example and python code: Sentiment analysis." https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-se Sep 2021.

[34] V. Morde, "Xgboost algorithm: Long may she reign!." https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be6 Apr 2019.

[35] A. Navlani, "Adaboost classifier in python." https://www.datacamp.com/community/tutorials/adaboost-classifier-python, Nov 2018.

[36] S. Patwardhan, "Simple understanding and implementation of knn algorithm!." https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/, Apr 2021.

[37] R. A. A. Viadinugroho, "Imbalanced classification in python: Smote-tomek links method." https://towardsdatascience.com/imbalanced-classification-in-python-smote-tomek-links-method-6e48dfe69bbc, Apr 2021.

[38] "Mean imputation for missing data (example in r & spss)." https://statisticsglobe.com/mean-imputation-for-missing-data/.

[39] S. Singhal, "Defining, analysing, and implementing imputation techniques." https://www.analyticsvidhya.com/blog/2021/06/defining-analysing-and-implementing-imputation-techniques/, Jun 2021.

[40] Y. Obadia, "The use of knn for missing values." https://towardsdatascience.com/the-use-of-knn-for-missing-values-cf33d935c637, Feb 2018.

[41] GeeksforGeeks, "Ml | extra tree classifier for feature selection." https://www.geeksforgeeks.org/ml-extra-tree-classifier-for-feature-selection/, Jul 2019.

[42] R. Shaikh, "Feature selection techniques in machine learning with python." https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e, Oct 2018.

[43] T. Liu, "Data for: A hybrid machine learning approach to cerebral stroke prediction based on imbalanced medical-datasets." https://data.mendeley.com/datasets/x8ygrw87jw/1, 2019.

[44] K. S. Htoon, "A guide to knn imputation." https://medium.com/@kyawsawhtoon/a-guide-to-knn-imputation-95e2dc496e, Jul 2020.

[45] K. Zhao, "Pre-process data with pipeline to prevent data leakage during cross-validation.." https://towardsdatascience.com/pre-process-data-with-pipeline-to-prevent-data-leakage-during-cross-validat Sep 2019.

[46] "Machine Learning Glossary." https://developers.google.com/machine-learning/glossary#AUC.

[47] "3.1. cross-validation: evaluating estimator performance." https://scikit-learn.org/stable/modules/cross_validation.html.

[48] CDC, "Diabetes testing." https://www.cdc.gov/diabetes/basics/getting-tested.html, Feb 2023.

X. Appendix

A. Source Code

Stroke Prediction System

A..1 models.py

```
from django.db import models
from django.core.validators import MinValueValidator,
    ↪ MaxValueValidator
import joblib
import pandas as pd
import numpy as np

# Create your models here.
class Data(models.Model):
    GENDER = (
        ("Female", "Female"),
        ("Male", "Male")
    )

    OPTION = (
        (0, 'No'),
        (1, 'Yes')
    )

    OPTION2 = (
        ("No", "No"),
        ("Yes", "Yes")
    )

    RESIDENCE_TYPE = (
        ("Rural", "Rural"),
        ("Urban", "Urban")
    )

    SMOKING_STATUS = (
        ("formerly smoked", "Formerly Smoked"),
        ("never smoked", "Never Smoked"),
        ("smokes", "Smokes")
    )

    WORK_TYPE = (
        ("Never_worked", "Never Worked"),
        ("Private", "Private"),
        ("Govt_job", "Government Job"),
        ("Self-employed", "Self-Employed")
    )

    age = models.PositiveIntegerField(validators=[
      ↪ MinValueValidator(25),MaxValueValidator(99)],null=True
      ↪ )
    bmi = models.FloatField(validators=[MinValueValidator(10.0)
      ↪ ,MaxValueValidator(60.0)],null=True, verbose_name='BMI
      ↪ ')
    avg_glucose_level = models.FloatField(validators=[
      ↪ MinValueValidator(55.0),MaxValueValidator(292.0)],null
      ↪ =True, verbose_name='Average Glucose Level')

    hypertension = models.PositiveIntegerField(choices=OPTION,
      ↪ null=True, verbose_name='Do you have hypertension?',
      ↪ default=None)
    heart_disease = models.PositiveIntegerField(choices=OPTION,
      ↪  null=True, verbose_name='Do you have a heart disease
      ↪ ?',default=None)

    sex = models.CharField(choices=GENDER, null=True,
      ↪ max_length = 20,default='')
    ever_married =  models.CharField(choices=OPTION2, null=True
      ↪ , max_length = 20, verbose_name='Have you been married
      ↪ ?',default='')
    residence_type = models.CharField(choices=RESIDENCE_TYPE,
      ↪ null=True, max_length = 20, verbose_name='Residence
      ↪ Area',default='')
    smoking_status = models.CharField(choices=SMOKING_STATUS,
      ↪ null=True, max_length = 20, verbose_name='Do you smoke
      ↪ ?',default='')
    work_type = models.CharField(choices=WORK_TYPE,null=True,
      ↪ max_length = 20, verbose_name='Category of Work',
      ↪ default='')
    predictions = models.PositiveIntegerField(blank=True, null=
      ↪ True)

    def save(self, *args, **kwargs):

        ml_model = joblib.load('ml_model/final_model_check.
      ↪ joblib')

        col_sample = ['gender','age','hypertension','
      ↪ heart_disease','ever_married','work_type','
      ↪ Residence_type','avg_glucose_level','bmi','
      ↪ smoking_status']
        features = [self.sex,self.age,self.hypertension,self.
      ↪ heart_disease,self.ever_married, self.work_type,
                    self.residence_type,self.avg_glucose_level,
      ↪ self.bmi,self.smoking_status]
        data_type = {'gender': np.dtype('O'),'age': np.dtype('
      ↪ float64'),'hypertension': np.dtype('int64'),'
      ↪ heart_disease': np.dtype('int64'),'ever_married': np.
      ↪ dtype('O'),
                    'work_type': np.dtype('O'),'Residence_type
      ↪ ': np.dtype('O'),'avg_glucose_level': np.dtype('
      ↪ float64'),'bmi': np.dtype('float64'),'smoking_status':
      ↪  np.dtype('O')}

        new_data = pd.DataFrame(np.array(features).reshape(-1,
      ↪ len(features)), columns=col_sample)
        new_data = new_data.astype(data_type)
        self.predictions = ml_model.predict(new_data)

        return super().save(*args,**kwargs)
```

A..2 forms.py

```
from django import forms
from .models import Data
from crispy_forms.helper import FormHelper
from crispy_forms.layout import Row, Column,Layout,Submit,HTML
from crispy_forms.bootstrap import InlineRadios
from django.urls import reverse_lazy

class DataForm(forms.ModelForm):

    def __init__(self, *args, **kwargs):
        super(DataForm, self).__init__(*args, **kwargs)

        self.helper = FormHelper(self)
        self.helper.form_method = 'post'
        self.helper.form_action = reverse_lazy('userpage-
    ↪ results')

        self.helper.layout = Layout(
            Row(
            Column(InlineRadios('sex'),css_class='form-group
    ↪ col-md-4 mb-0'),
            Column('age', css_class='form-group col-md-4 mb-0')
    ↪ ,
            Column(InlineRadios('residence_type'), css_class='
    ↪ form-group col-md-4 mb-0'),
            css_class='form-row'
            ),
            Row(
            Column(InlineRadios('ever_married'),css_class='form
    ↪ -group col-md-4 mb-0'),
            Column('work_type',css_class='form-group col-md-8
    ↪ mb-0'),
            css_class='form-row'
            ),
            'smoking_status',
            Row(
            Column(InlineRadios('hypertension'),css_class='form
    ↪ -group col-md-6 mb-0'),
            Column(InlineRadios('heart_disease'),css_class='
    ↪ form-group col-md-6 mb-0'),
            css_class='form-row'
            ),
            Row(
            Column('avg_glucose_level',css_class='form-group
    ↪ col-md-6 mb-0'),
            Column('bmi',css_class='form-group col-md-6 mb-0'),
            css_class='form-row'
            ),
            Row(
            css_class='dropdown-divider mb-3'
            ),
            Row(
            Column(HTML('<p><i>All fields are required.</i></p
    ↪ >')),
            Column(Submit('submit', 'Enter', css_class='btn btn
    ↪ -success float-right')),
            css_class='form-row'
            )
        )

    class Meta:
        model = Data
        fields = ['sex', 'age', 'ever_married', 'residence_type
    ↪ ','work_type','smoking_status','hypertension', '
    ↪ heart_disease',
                  'avg_glucose_level', 'bmi']

        widgets = {
            'sex': forms.RadioSelect,
            'ever_married': forms.RadioSelect,
            'residence_type': forms.RadioSelect,
            'hypertension': forms.RadioSelect,
            'heart_disease': forms.RadioSelect
        }
```

A..3 views.py

```
from django.shortcuts import render, redirect
from .forms import DataForm
from .models import Data

# Create your views here.
def index(request):
    return render(request, 'userpage/index.html')

def input_data(request):
    if request.method == 'POST':
        form = DataForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('userpage-results')
    else:
        form =DataForm()
    context = {
        'form':form,
    }
    return render(request, 'userpage/input_data.html',context)

def show_results(request):
    predicted_results = Data.objects.last()

    context={
        'predicted_results': predicted_results
    }
    return render(request, 'userpage/results.html',context)

def about(request):
    return render(request, 'userpage/about.html')
```

A..4 admin.py

```
from django.contrib import admin
from .models import Data

# Register your models here.
class DataAdmin(admin.ModelAdmin):
```

```
    list_display = ('sex', 'age', 'hypertension',  '
      ↪ heart_disease','ever_married', 'work_type', '
      ↪ residence_type',
                  'avg_glucose_level',  'bmi', '
      ↪ smoking_status', 'predictions')

admin.site.register(Data, DataAdmin)
```

Machine Learning Model

A..5 Hyperparameter Tuning

```
{
  "cells": [
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "99d4a419"
      },
      "outputs": [],
      "source": [
        "import pandas as pd\n",
        "import numpy as np"
      ],
      "id": "99d4a419"
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "25760d29"
      },
      "source": [
        "## Basic Info About the Data"
      ],
      "id": "25760d29"
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "5cbf54eb"
      },
      "outputs": [],
      "source": [
        "df = pd.read_csv(\"dataset.csv\")\n",
        "df.head()"
      ],
      "id": "5cbf54eb"
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "95d848f2"
      },
      "outputs": [],
      "source": [
        "print(\"Shape:\",df.shape)\n",
        "print()\n",
        "print(df.info())"
      ],
      "id": "95d848f2"
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "ac045baf"
      },
      "outputs": [],
      "source": [
        "#summary statistics of numerical values\n",
        "df.describe()"
      ],
      "id": "ac045baf"
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "7b122840"
      },
      "outputs": [],
      "source": [
        "#Number of unique values per feature\n",
        "df.nunique()"
      ],
      "id": "7b122840"
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "18692b15"
      },
      "outputs": [],
      "source": [
        "# unique values of each categorical values\n",
        "column_names = list(df.columns.values)\n",
        "\n",
        "for col in column_names:\n",
        "  if df[col].dtype == 'object':\n",
        "    print(df[col].unique())"
      ],
      "id": "18692b15"
    },
    {
      "cell_type": "markdown",
      "metadata": {
        "id": "6a9cc9bf"
      },
      "source": [
        "<b>Is the data clean enough for data visualization and
      ↪  exploration?</b>"
      ],
      "id": "6a9cc9bf"
    },
```

```
{
  "cell_type": "markdown",
  "metadata": {
    "id": "30c5b30d"
  },
  "source": [
    "Check for duplicates"
  ],
  "id": "30c5b30d"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "031781ce"
  },
  "outputs": [],
  "source": [
    "dups = df.duplicated()\n",
    "# report if there are any duplicates\n",
    "print(dups.any())"
  ],
  "id": "031781ce"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "748dcbac"
  },
  "source": [
    "Missing Values in raw data"
  ],
  "id": "748dcbac"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8ad709ff"
  },
  "outputs": [],
  "source": [
    "print(df.isnull().sum())"
  ],
  "id": "8ad709ff"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "a9956a99"
  },
  "outputs": [],
  "source": [
    "df.gender.unique()"
  ],
  "id": "a9956a99"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "747fcc22"
  },
  "source": [
    ">Since number of unique features of gender is 3
    ↪ because of the additional 'Other', consider it to be
    ↪ removed.\n",
    "Also, according to Liu, Fan, Wu (2019),\n",
    ">*   Minimum Age for Smoke-Monitoring is 25 years old\
    ↪ n",
    ">*   Reference Values of BMI should be no more than
    ↪ 60%\n",
    ">*   Patient ID is irrelevant hence it will be dropped
    ↪  from the dataset.\n",
    "\n",
    ">Hence, to remove outliers, data where age < 25 and
    ↪ BMI > 60% should be removed."
  ],
  "id": "747fcc22"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "781327fa"
  },
  "source": [
    "# Data Preparation"
  ],
  "id": "781327fa"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "674d3f88"
  },
  "source": [
    "Imputation and transformation of data into applicable
    ↪ ones (cat to numerical)"
  ],
  "id": "674d3f88"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "30a884e2"
  },
  "outputs": [],
  "source": [
    "#all features except ID\n",
    "X = df.iloc[:,1:len(column_names)-1]\n",
    "y = pd.DataFrame(df.iloc[:,-1])"
  ],
  "id": "30a884e2"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "63d7004c"
```

```
  },
  "outputs": [],
  "source": [
    "print(X)\n",
    "print(y)"
  ],
  "id": "63d7004c"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "235b5971"
  },
  "outputs": [],
  "source": [
    "import matplotlib.pyplot as plt\n",
    "import seaborn as sns\n",
    "from sklearn.model_selection import train_test_split"
  ],
  "id": "235b5971"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "2f51f7ff"
  },
  "outputs": [],
  "source": [
    "X_train, X_test, y_train, y_test = train_test_split(X,
  ↪ y, test_size = 0.2, random_state=100)"
  ],
  "id": "2f51f7ff"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "591f9cc5"
  },
  "outputs": [],
  "source": [
    "print(\"X Train: \", X_train.shape)\n",
    "print(\"X Test: \", X_test.shape)\n",
    "print(\"y Train: \", y_train.shape)\n",
    "print(\"y Test: \", y_test.shape)"
  ],
  "id": "591f9cc5"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "4a97e303"
  },
  "outputs": [],
  "source": [
    "orig_X_train=X_train.copy()\n",
    "orig_y_train=y_train.copy()"
  ],
  "id": "4a97e303"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "408c5a51"
  },
  "source": [
    "Before imputation, data needs to be encoded and
  ↪ irrelevant info should be removed(outliers). Only
  ↪ applied on training data."
  ],
  "id": "408c5a51"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "53c91bd2"
  },
  "source": [
    "#### Outlier Removal"
  ],
  "id": "53c91bd2"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "670f8311"
  },
  "source": [
    "Gender:Other"
  ],
  "id": "670f8311"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "016ef74e"
  },
  "outputs": [],
  "source": [
    "X_train.gender.unique()"
  ],
  "id": "016ef74e"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "a51cc68c"
  },
  "outputs": [],
  "source": [
    "other_results = X_train.loc[X_train['gender']  != '
  ↪ Other']\n",
    "other_results.shape"
  ],
  "id": "a51cc68c"
```

```
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8fd14fd0"
  },
  "outputs": [],
  "source": [
    "X_train= X_train.loc[other_results.index]\n",
    "X_train.shape"
  ],
  "id": "8fd14fd0"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "9285cbad"
  },
  "outputs": [],
  "source": [
    "X_train.gender.unique()"
  ],
  "id": "9285cbad"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "0badafa1"
  },
  "source": [
    "Age>=25%"
  ],
  "id": "0badafa1"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "0bc4cc30"
  },
  "outputs": [],
  "source": [
    "#visualize age boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='age').set(title='Age')"
  ],
  "id": "0bc4cc30"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "50724a87"
  },
  "outputs": [],
  "source": [
    "age_results = X_train.loc[X_train['age'] >= 25]\n",
    "age_results.shape"
  ],
  "id": "50724a87"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "5f67a807"
  },
  "outputs": [],
  "source": [
    "X_train = X_train.loc[age_results.index]\n",
    "X_train.shape"
  ],
  "id": "5f67a807"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "cadadc20"
  },
  "outputs": [],
  "source": [
    "#revisualize age boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='age').set(title='Age')"
  ],
  "id": "cadadc20"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "TbycuEt-XK1N"
  },
  "outputs": [],
  "source": [
    "print(X_train.isnull().sum())"
  ],
  "id": "TbycuEt-XK1N"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "498c9a6c"
  },
  "source": [
    "BMI<=60% (With the rows with missing values)"
  ],
  "id": "498c9a6c"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ad55f0a4"
  },
  "outputs": [],
  "source": [
```

```
    "#visualize bmi boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='bmi').set(title='BMI')"
  ],
  "id": "ad55f0a4"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "EDsdbCghSmbp"
  },
  "outputs": [],
  "source": [
    "more_than = X_train.loc[X_train['bmi'] > 60]"
  ],
  "id": "EDsdbCghSmbp"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "1oWhna8EJN_d"
  },
  "outputs": [],
  "source": [
    "#remove bmi reference values more than 60\n",
    "bmi_results = X_train[~X_train.index.isin(more_than.
↪ index)]"
  ],
  "id": "1oWhna8EJN_d"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "colab": {
      "base_uri": "https://localhost:8080/"
    },
    "id": "D9jcbZSJI9Vc",
    "outputId": "e750aa2e-4c28-4a81-e292-ac492e435e9f"
  },
  "outputs": [
    {
      "data": {
        "text/plain": [
          "(25871, 10)"
        ]
      },
      "execution_count": 31,
      "metadata": {},
      "output_type": "execute_result"
    }
  ],
  "source": [
    "X_train.shape"
  ],
  "id": "D9jcbZSJI9Vc"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "3IFHkewfVIB8"
  },
  "outputs": [],
  "source": [
    "print(bmi_results.shape)\n",
    "print(more_than.shape)"
  ],
  "id": "3IFHkewfVIB8"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "THJBHjsQVaPF"
  },
  "outputs": [],
  "source": [
    "X_train.shape"
  ],
  "id": "THJBHjsQVaPF"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "lKHwJXMxP11S"
  },
  "outputs": [],
  "source": [
    "more_than.isnull().sum()"
  ],
  "id": "lKHwJXMxP11S"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "5O2__cQyTHgz"
  },
  "outputs": [],
  "source": [
    "bmi_results.isnull().sum()"
  ],
  "id": "5O2__cQyTHgz"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "uf5k9sY6UTJu"
  },
  "outputs": [],
  "source": [
    "X_train.isnull().sum()"
  ],
  "id": "uf5k9sY6UTJu"
},
```

```
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "b784e226"
  },
  "outputs": [],
  "source": [
    "X_train = X_train.loc[bmi_results.index]\n",
    "X_train.shape"
  ],
  "id": "b784e226"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "qa5FOJvWXSy4"
  },
  "outputs": [],
  "source": [
    "print(X_train.isnull().sum())"
  ],
  "id": "qa5FOJvWXSy4"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "c9e0e8d7"
  },
  "outputs": [],
  "source": [
    "#revisualize bmi boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='bmi').set(title='BMI')"
  ],
  "id": "c9e0e8d7"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "ff857a43"
  },
  "source": [
    "Since Ages < 25 has been removed, there are no more
  ↪ values that pertains to children hence the feature
  ↪ children has been removed."
  ],
  "id": "ff857a43"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "a6dd348e"
  },
  "outputs": [],
  "source": [
    "X_train.work_type.unique()"
  ],
  "id": "a6dd348e"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "5e7c075c"
  },
  "source": [
    "Filter y with the indexes of the X_train"
  ],
  "id": "5e7c075c"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "80e19575"
  },
  "outputs": [],
  "source": [
    "y_train = y_train.loc[X_train.index]"
  ],
  "id": "80e19575"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "962b445e"
  },
  "outputs": [],
  "source": [
    "print(\"Filtered Shapes Before and After Removal\")\n
  ↪ ",
    "print(\"Before| X_train: \", orig_X_train.shape, \" |
  ↪ y_train: \", orig_y_train.shape)\n",
    "print(\"After| X_train: \", X_train.shape, \" |
  ↪ y_train: \", y_train.shape)"
  ],
  "id": "962b445e"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "7d2b5eb4"
  },
  "source": [
    "#### Customizing ColumnTransfer"
  ],
  "id": "7d2b5eb4"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "187b2ec5"
  },
  "source": [
    "Separating categorical and numerical data for encoding
  ↪ ."
```

```
  ],
  "id": "187b2ec5"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "96203e27"
  },
  "outputs": [],
  "source": [
    "cat_data = X_train.select_dtypes(include=['object','
  ↪ int64'])"
  ],
  "id": "96203e27"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d71f6ccc",
    "scrolled": true
  },
  "outputs": [],
  "source": [
    "cat_data"
  ],
  "id": "d71f6ccc"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "6c2d7e48"
  },
  "source": [
    ">* hypertension and heart_disease can be left as is.\n
  ↪ ",
    ">* gender, ever_married, work_type and residence_type
  ↪ needs to be one hot encoded(dummy variable trap
  ↪ prevention).\n",
    ">* smoking status needs to be ordinal encoded before
  ↪ imputed since knnimputer only works on numerical
  ↪ values."
  ],
  "id": "6c2d7e48"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "85c1eaa7"
  },
  "outputs": [],
  "source": [
    "pass_feats = ['hypertension','heart_disease']\n",
    "ohe_feats = ['gender','ever_married','work_type','
  ↪ Residence_type']"
  ],
  "id": "85c1eaa7"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d52d9b5d"
  },
  "outputs": [],
  "source": [
    "num_data = X_train.select_dtypes(include=['float64'])"
  ],
  "id": "d52d9b5d"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "710168df",
    "scrolled": true
  },
  "outputs": [],
  "source": [
    "num_data"
  ],
  "id": "710168df"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "f24f8363"
  },
  "source": [
    "Create a Pipeline for 2 different imputation methods
  ↪ but with the same feature engineering techniques(does
  ↪ not contain the scaling, feature selection and model
  ↪ since we need to visualize the cleaned data)."
  ],
  "id": "f24f8363"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "6d4cdb8f"
  },
  "outputs": [],
  "source": [
    "from sklearn.impute import SimpleImputer, KNNImputer\n
  ↪ ",
    "from sklearn.preprocessing import OneHotEncoder,
  ↪ RobustScaler, OrdinalEncoder\n",
    "from sklearn.pipeline import Pipeline\n",
    "from sklearn.compose import ColumnTransformer,
  ↪ make_column_transformer"
  ],
  "id": "6d4cdb8f"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
```

```
    "id": "1b1ace7b"
   },
   "outputs": [],
   "source": [
    "#display the diagram\n",
    "from sklearn import set_config"
   ],
   "id": "1b1ace7b"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "24d8dfdb"
   },
   "outputs": [],
   "source": [
    "num_data_col = num_data.columns.tolist()"
   ],
   "id": "24d8dfdb"
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "549cb706"
   },
   "source": [
    "### <b>1st Imputation Method<b/>\n",
    ">Impute both bmi and smoke. Next, encode the
   ↪ categorical smoking_status feature then one-hot encode
   ↪  the rest of the categorcial values."
   ],
   "id": "549cb706"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "d72c2e03"
   },
   "outputs": [],
   "source": [
    "simple_ct1 = ColumnTransformer([\n",
    "    ('impute_bmi', SimpleImputer(strategy='mean',
   ↪ missing_values=np.nan),['bmi']),\n",
    "    ('impute_smoke',SimpleImputer(strategy='
   ↪ most_frequent',missing_values=np.NaN),['smoking_status
   ↪ ']),\n",
    "    ],remainder='passthrough',
   ↪ verbose_feature_names_out=False,n_jobs=-1)"
   ],
   "id": "d72c2e03"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "78c1e22d"
   },
   "outputs": [],
   "source": [
    "simple_ct2 = ColumnTransformer([\n",
    "    ('ord', OrdinalEncoder(handle_unknown='
   ↪ use_encoded_value', unknown_value=np.nan),['
   ↪ smoking_status']),\n",
    "    ('ohe', OneHotEncoder(sparse_output=False,
   ↪ handle_unknown='ignore',drop=\"if_binary\"),ohe_feats)
   ↪ ,\n",
    "    ('scale', RobustScaler(), num_data_col)\n",
    "    ],verbose_feature_names_out=False,remainder='
   ↪ passthrough',n_jobs=-1)"
   ],
   "id": "78c1e22d"
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "9ebbbd42"
   },
   "source": [
    "> run only for data visualization process of the
   ↪ prepared data. only applicable on column transformers,
   ↪  not with imbalance handling because of fit_transform
   ↪ method."
   ],
   "id": "9ebbbd42"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "849685c2"
   },
   "outputs": [],
   "source": [
    "ct_simple = Pipeline(steps = [('prep1', simple_ct1),
   ↪ ('prep2',simple_ct2)])"
   ],
   "id": "849685c2"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "54b72eab",
    "scrolled": false
   },
   "outputs": [],
   "source": [
    "set_config(transform_output=\"pandas\",display='
   ↪ diagram')\n",
    "display(ct_simple)"
   ],
   "id": "54b72eab"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "b7a321f3",
```

```
    "scrolled": false
   },
   "outputs": [],
   "source": [
    "prep1_data = X_train.copy()\n",
    "prep1_data"
   ],
   "id": "b7a321f3"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "mq-XTyDHyhC4"
   },
   "outputs": [],
   "source": [
    "print(prep1_data.shape)\n",
    "print(prep1_data.isnull().sum())"
   ],
   "id": "mq-XTyDHyhC4"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "1c300f44"
   },
   "outputs": [],
   "source": [
    "prep1_data = ct_simple.fit_transform(prep1_data)"
   ],
   "id": "1c300f44"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "3dfcd88d"
   },
   "outputs": [],
   "source": [
    "prep1_data.columns"
   ],
   "id": "3dfcd88d"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "ba902377",
    "scrolled": true
   },
   "outputs": [],
   "source": [
    "prep1_data.head()"
   ],
   "id": "ba902377"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "UudQI3Ztz2C9"
   },
   "outputs": [],
   "source": [
    "X_train.head()"
   ],
   "id": "UudQI3Ztz2C9"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "40a18314"
   },
   "outputs": [],
   "source": [
    "print(prep1_data.shape)\n",
    "print(prep1_data.isnull().sum())"
   ],
   "id": "40a18314"
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "9d55ff1f"
   },
   "source": [
    "###<b>2nd Imputation Method</b>\n",
    "> Encode first the smoking_status since KNNImputer does not accept non-numerical values then impute the data before finally one-hot encoding the rest of the categorical variables."
   ],
   "id": "9d55ff1f"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "b-STeKBt8IJZ"
   },
   "outputs": [],
   "source": [
    "from sklearn.impute import KNNImputer"
   ],
   "id": "b-STeKBt8IJZ"
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "ahScsOnX8PpS"
   },
   "source": [
    "first convert text data into numerical data to make it suitable for knn imputaion"
   ],
   "id": "ahScsOnX8PpS"
```

```
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "q93etr6FwLV_"
  },
  "outputs": [],
  "source": [
    "from sklearn.preprocessing import FunctionTransformer"
  ],
  "id": "q93etr6FwLV_"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "u9bP1H_78XPd"
  },
  "outputs": [],
  "source": [
    "round_values = FunctionTransformer(np.round)"
  ],
  "id": "u9bP1H_78XPd"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "OOkHh0qQ8ZK4"
  },
  "outputs": [],
  "source": [
    "knn_ct1 = ColumnTransformer([\n",
    "    ('ohe', OneHotEncoder(sparse_output=False,
↪ handle_unknown='ignore',drop=\"if_binary\"),ohe_feats)
↪ ,\n",
    "    ('ord', OrdinalEncoder(handle_unknown='
↪ use_encoded_value', unknown_value=np.nan),['
↪ smoking_status']),\n",
    "    ('scale', RobustScaler(), num_data_col),\n",
    "    ],remainder='passthrough',
↪ verbose_feature_names_out=False,n_jobs=-1)"
  ],
  "id": "OOkHh0qQ8ZK4"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "MfBcL6vku-gZ"
  },
  "outputs": [],
  "source": [
    "knn_ct2 = ColumnTransformer([\n",
    "    ('impute_both', KNNImputer(),['bmi','
↪ smoking_status']),\n",
    "    ],remainder='passthrough',
↪ verbose_feature_names_out=False,n_jobs=-1)"
  ],
  "id": "MfBcL6vku-gZ"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "E2nFUO2WzOsk"
  },
  "outputs": [],
  "source": [
    "knn_ct3 = ColumnTransformer([\n",
    "    ('round', round_values, ['smoking_status'])\n",
    "    ],remainder='passthrough',
↪ verbose_feature_names_out=False,n_jobs=-1)"
  ],
  "id": "E2nFUO2WzOsk"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "V3PZV9XZtJbu"
  },
  "source": [
    "According to https://medium.com/@kyawsawhtoon/a-guide-
↪ to-knn-imputation-95e2dc496e, since KNN Imputation is
↪ distance-based, it is better to scale the data to
↪ prevent any biased replacements for the missing values
↪ ."
  ],
  "id": "V3PZV9XZtJbu"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "rITWQIkt8aXS"
  },
  "outputs": [],
  "source": [
    "ct_knn = Pipeline(steps = [('prep1', knn_ct1),('prep2
↪ ', knn_ct2),('prep3',knn_ct3)])"
  ],
  "id": "rITWQIkt8aXS"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "kncHe7hA8e2F"
  },
  "outputs": [],
  "source": [
    "set_config(transform_output=\"pandas\",display='
↪ diagram')"
  ],
  "id": "kncHe7hA8e2F"
},
{
  "cell_type": "code",
  "execution_count": null,
```

```
  "metadata": {
    "id": "_lMTvmFS8cJ0"
  },
  "outputs": [],
  "source": [
    "display(ct_knn)"
  ],
  "id": "_lMTvmFS8cJ0"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "_jivMW1fuRVM"
  },
  "outputs": [],
  "source": [
    "df_sample = ct_knn.fit_transform(test)"
  ],
  "id": "_jivMW1fuRVM"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "0o_LsbrnuiW3"
  },
  "outputs": [],
  "source": [
    "df_sample.shape"
  ],
  "id": "0o_LsbrnuiW3"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "0deK7D1zulOU"
  },
  "outputs": [],
  "source": [
    "df_sample.head()"
  ],
  "id": "0deK7D1zulOU"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "uwueeWrGuszD"
  },
  "outputs": [],
  "source": [
    "print(df_sample.isnull().sum())"
  ],
  "id": "uwueeWrGuszD"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "fpVIN4opvSWa"
  },
  "outputs": [],
  "source": [
    "df_sample_col = df_sample.columns"
  ],
  "id": "fpVIN4opvSWa"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "HnCxKFinvjNQ"
  },
  "outputs": [],
  "source": [
    "df_sample.describe()"
  ],
  "id": "HnCxKFinvjNQ"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "CQU6q_EFvUm6"
  },
  "outputs": [],
  "source": [
    "for col in df_sample_col:\n",
    "  print(col)\n",
    "  print(df_sample[col].unique())"
  ],
  "id": "CQU6q_EFvUm6"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "2da63f98"
  },
  "source": [
    "# Model Configuration"
  ],
  "id": "2da63f98"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "cee175ce"
  },
  "outputs": [],
  "source": [
    "from imblearn.pipeline import Pipeline as imbpipeline\n",
    "from sklearn.model_selection import GridSearchCV"
  ],
  "id": "cee175ce"
},
{
```

```
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d544fcbf"
  },
  "outputs": [],
  "source": [
    "from imblearn.pipeline import Pipeline as imbpipeline,
  ↪  make_pipeline"
  ],
  "id": "d544fcbf"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "bab5ab4a"
  },
  "outputs": [],
  "source": [
    "from sklearn.feature_selection import SelectFromModel\
  ↪ n",
    "from sklearn.ensemble import ExtraTreesClassifier"
  ],
  "id": "bab5ab4a"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d6f38a55"
  },
  "outputs": [],
  "source": [
    "from imblearn.over_sampling import SMOTE\n",
    "from imblearn.combine import SMOTETomek"
  ],
  "id": "d6f38a55"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "3617c7c2"
  },
  "outputs": [],
  "source": [
    "from sklearn.linear_model import LogisticRegression\n
  ↪ ",
    "from sklearn.ensemble import RandomForestClassifier\n
  ↪ ",
    "from sklearn.svm import SVC\n",
    "import xgboost as xgb\n",
    "from sklearn.ensemble import AdaBoostClassifier\n",
    "from sklearn.neighbors import KNeighborsClassifier\n",
    "from sklearn.neural_network import MLPClassifier"
  ],
  "id": "3617c7c2"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "wyv_77enlOLk"
  },
  "outputs": [],
  "source": [
    "from sklearn.tree import DecisionTreeClassifier\n",
    "import time"
  ],
  "id": "wyv_77enlOLk"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "6d263afb"
  },
  "outputs": [],
  "source": [
    "from sklearn.metrics import confusion_matrix,
  ↪ classification_report\n",
    "from sklearn.metrics import accuracy_score, f1_score,
  ↪ precision_score,recall_score,roc_auc_score,
  ↪ make_scorer"
  ],
  "id": "6d263afb"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "oQMTe6ZbsH-z"
  },
  "outputs": [],
  "source": [
    "import csv"
  ],
  "id": "oQMTe6ZbsH-z"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "defee15e"
  },
  "outputs": [],
  "source": [
    "smote = SMOTE(random_state = 100)\n",
    "smote_tomek = SMOTETomek(random_state=100)"
  ],
  "id": "defee15e"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "66177783"
  },
  "source": [
    "## GridSearch"
```

```
  ],
  "id": "66177783"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "55e7a93d"
  },
  "outputs": [],
  "source": [
    "lr_clf = LogisticRegression(random_state = 100)\n",
    "lr_param_grid = {'clf__C':[0.0001,0.001, 0.01, 0.1, 1,
↪  10, 100],\n",
    "              'clf__solver':['sag', 'saga','newton-cg
↪ ', 'lbfgs'],\n",
    "              'clf__max_iter':[1000,2000,3000]}"
  ],
  "id": "55e7a93d"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "72de3b93"
  },
  "outputs": [],
  "source": [
    "rf_clf = RandomForestClassifier(random_state = 100)\n
↪ ",
    "rf_param_grid = {'clf__n_estimators':[10,100,500],'
↪ clf__min_samples_leaf' : [1, 3, 5],\n",
    "              'clf__max_features':['sqrt','log2']}"
  ],
  "id": "72de3b93"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "9d5712c2"
  },
  "outputs": [],
  "source": [
    "svm_clf = SVC(random_state = 100)\n",
    "svm_param_grid = {'clf__kernel':['linear','poly','rbf
↪ '],'clf__C':[0.1,1,10]}"
  ],
  "id": "9d5712c2"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "KGIZubOyIUKT"
  },
  "outputs": [],
  "source": [
    "mlp_clf = MLPClassifier(random_state=100)\n",
    "mlp_param_grid = {\n",
    "    'clf__solver':['lbfgs','sgd','adam'],\n",
    "    'clf__activation':['logistic', 'tanh', 'relu']\n",
    "}"
  ],
  "id": "KGIZubOyIUKT"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "e7b09e69"
  },
  "outputs": [],
  "source": [
    "xgb_clf = xgb.XGBClassifier(random_state = 100)\n",
    "xgb_param_grid = {'clf__learning_rate':[0, 0.0001,
↪ 0.001, 0.01],\n",
    "              'clf__max_depth':np.arange(1, 11, 3),\n
↪ ",
    "              'clf__min_child_weight': np.arange(1,
↪ 11, 3)}"
  ],
  "id": "e7b09e69"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "27d97980"
  },
  "outputs": [],
  "source": [
    "ada_clf = AdaBoostClassifier(random_state = 100)\n",
    "ada_param_grid = {'clf__learning_rate
↪ ':[0.0001,0.001,0.01,0.1,1],\n",
    "              'clf__n_estimators':[10,50,100,500]}"
  ],
  "id": "27d97980"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d91cc454"
  },
  "outputs": [],
  "source": [
    "knn_clf = KNeighborsClassifier()\n",
    "knn_param_grid = {'clf__n_neighbors':list(range
↪ (1,21,1)),\n",
    "              'clf__weights' : ['uniform', 'distance
↪ '],\n",
    "              'clf__metric': ['euclidean', 'manhattan
↪ ', 'minkowski']}"
  ],
  "id": "d91cc454"
},
{
  "cell_type": "markdown",
  "metadata": {
```

```
    "id": "7435456c"
   },
   "source": [
    "### <b>Simple Imputation(Mean Value and Most Frequent)
   ↪ </b>"
   ],
   "id": "7435456c"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "9c73a2bc"
   },
   "outputs": [],
   "source": [
    "def ht_simple(imb, clf, feat, param_grid, x_train,
   ↪ y_train, x_test, y_test):\n",
    "\n",
    "    if feat == True:\n",
    "        xtra_trees = ExtraTreesClassifier(random_state
   ↪  = 100)\n",
    "        feat_selection = SelectFromModel(xtra_trees)\n
   ↪ ",
    "\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
   ↪ simple_ct1],['prep2', simple_ct2],\n",
    "                                       ['imb', imb],['
   ↪ fs', feat_selection],['clf', clf]])\n",
    "    else:\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
   ↪ simple_ct1],['prep2', simple_ct2],\n",
    "                                       ['imb', imb],['
   ↪ clf', clf]])\n",
    "\n",
    "    pipe_ml.set_output(transform=\"pandas\")\n",
    "\n",
    "    grid_search = GridSearchCV(estimator=pipe_ml,
   ↪ param_grid=param_grid, scoring=make_scorer(
   ↪ roc_auc_score),\n",
    "                               verbose =4, cv=10,
   ↪ n_jobs=-1)\n",
    "    start_time = time.time()\n",
    "    grid_search.fit(x_train, np.ravel(y_train))\n",
    "\n",
    "    exec_time = str((time.time() - start_time))\n",
    "\n",
    "    cv_score = grid_search.best_score_\n",
    "    test_score = grid_search.score(x_test, y_test)\n",
    "    best_param = grid_search.best_params_\n",
    "    best_estimator = grid_search.best_estimator_\n",
    "\n",
    "    result = {\"Cross-validation score\": cv_score,\n
   ↪ ",
    "             \"Test Score\": test_score,\n",
    "              \"Execution Time\" : exec_time,\n",
    "            \"Best Paramaters\": best_param}\n",
    "\n",
    "    return result, best_estimator"
  ],
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "507fa322"
   },
   "source": [
    "### **KNN Imputation**"
   ],
   "id": "507fa322"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "9745f594"
   },
   "outputs": [],
   "source": [
    "def ht_knn(imb, clf, feat, param_grid, x_train,
   ↪ y_train, x_test, y_test):\n",
    "\n",
    "    if feat == True:\n",
    "        xtra_trees = ExtraTreesClassifier(random_state
   ↪  = 100)\n",
    "        feat_selection = SelectFromModel(xtra_trees)\n
   ↪ ",
    "\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
   ↪ knn_ct1], ['prep2',knn_ct2],['prep3',knn_ct3],\n",
    "                                       ['imb', imb],['fs',
   ↪  feat_selection],['clf', clf]])\n",
    "    else:\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
   ↪ knn_ct1], ['prep2',knn_ct2],['prep3',knn_ct3],\n",
    "                                       ['imb', imb],['clf
   ↪ ', clf]])\n",
    "\n",
    "    pipe_ml.set_output(transform=\"pandas\")\n",
    "\n",
    "    grid_search = GridSearchCV(estimator=pipe_ml,
   ↪ param_grid=param_grid, scoring=make_scorer(
   ↪ roc_auc_score),\n",
    "                               verbose=4, cv=10,
   ↪ n_jobs=-1)\n",
    "\n",
    "    start_time = time.time()\n",
    "    grid_search.fit(x_train, np.ravel(y_train))\n",
    "\n",
    "    exec_time = str((time.time() - start_time))\n",
    "\n",
    "    cv_score = grid_search.best_score_\n",
    "    test_score = grid_search.score(x_test, y_test)\n",
    "    best_param = grid_search.best_params_\n",
    "    best_estimator = grid_search.best_estimator_\n",
    "\n",
    "    result = {\"Cross-validation score\": cv_score,\n
   ↪ ",
    "            \"Test Score\": test_score,\n",
```

```
    "             \"Execution Time\" : exec_time,\n",
    "            \"Best Paramaters\": best_param}\n",
    "\n",
    "    return result, best_estimator"
   ],
   "id": "9745f594"
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "c7ee73f2"
   },
   "source": [
    "### Definition of Pipes"
   ],
   "id": "c7ee73f2"
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "499aa321"
   },
   "source": [
    "There will be 4 pipes (Model Configurations) to be conducted:\n",
    "1. SMOTE w/oFS\n",
    "2. SMOTETomek w/o FS\n",
    "3. SMOTE with FS\n",
    "4. SMOTETomek with FS\n",
    "\n",
    "These pipes will be done on both imputations."
   ],
   "id": "499aa321"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "57538a08"
   },
   "outputs": [],
   "source": [
    "def create_pipes(ht_func, clf, param_grid, x_train, y_train, x_test, y_test):\n",
    "    results = []\n",
    "    pipes = []\n",
    "\n",
    "    out1, smote_ = ht_func(smote, clf, False, param_grid, x_train, y_train, x_test, y_test)\n",
    "    print(\"---pipe1 done---\")\n",
    "    print(out1)\n",
    "    out2, smotet_= ht_func(smote_tomek, clf, False, param_grid, x_train, y_train, x_test, y_test)\n",
    "    print(\"---pipe2 done---\")\n",
    "    print(out2)\n",
    "    out3, smote_fs = ht_func(smote, clf, True, param_grid, x_train, y_train, x_test, y_test)\n",
    "    print(\"---pipe3 done---\")\n",
    "    print(out3)\n",
    "    out4, smotet_fs = ht_func(smote_tomek, clf, True, param_grid, x_train, y_train, x_test, y_test)\n",
    "    print(\"---all pipes done---\")\n",
    "    print(out4)\n",
    "\n",
    "    results.extend([out1,out2,out3,out4])\n",
    "    pipes.extend([smote_,smotet_,smote_fs,smotet_fs])\n",
    "    output_df = pd.DataFrame(results)\n",
    "\n",
    "    return output_df, pipes"
   ],
   "id": "57538a08"
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "HQ0CCcp6Vp4L"
   },
   "outputs": [],
   "source": [
    "def evaluate_model(model, x_test, y_test):\n",
    "  y_pred = model.predict(x_test)\n",
    "\n",
    "  print(\"-------------\")\n",
    "  print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))\n",
    "  print('Precision: %.3f' % precision_score(y_test, y_pred))\n",
    "  print('F1 Score: %.3f' % f1_score(y_test, y_pred))\n",
    "  print('Recall: %.3f' % recall_score(y_test, y_pred))\n",
    "  print('Specificity: %.3f' % recall_score(y_test, y_pred, pos_label =0))\n",
    "  print('ROC AUC: %.3f' % roc_auc_score(y_test, y_pred))\n",
    "  print(\"-------------\")\n",
    "\n",
    "  print(classification_report(y_pred,y_test))\n",
    "\n",
    "  print(\"-------------\")\n",
    "  conf = confusion_matrix(y_test, y_pred)\n",
    "  sns.heatmap(conf,annot = True, fmt='g',cmap ='Blues')"
   ],
   "id": "HQ0CCcp6Vp4L"
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "d7d3781b"
   },
   "source": [
    "#### Logistic Regression"
   ],
   "id": "d7d3781b"
  },
  {
   "cell_type": "markdown",
   "cell_type": "markdown",
```

```
  "metadata": {
    "id": "e-8Pk8tYm-Np"
  },
  "source": [
    "##### Simple"
  ],
  "id": "e-8Pk8tYm-Np"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "O9d31dbc"
  },
  "outputs": [],
  "source": [
    "simple_lr_out, simple_lr_pipes = create_pipes(
  ↪ ht_simple, lr_clf, lr_param_grid, X_train, y_train,
  ↪ X_test, y_test)"
  ],
  "id": "O9d31dbc"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "4qVkkdeF74Ch"
  },
  "source": [
    "##### KNN Imputation"
  ],
  "id": "4qVkkdeF74Ch"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "EaAgnTJV74Ci"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_knn(smote, lr_clf, False,
  ↪ lr_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "EaAgnTJV74Ci"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "MR9w4Eda74Ci"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_knn(smote_tomek, lr_clf, False,
  ↪ lr_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "MR9w4Eda74Ci"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "BGkPBtAd74Cj"
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_knn(smote, lr_clf, True,
  ↪ lr_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "BGkPBtAd74Cj"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "DUTTC3lY74Cj"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_knn(smote_tomek, lr_clf, True,
  ↪ lr_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "DUTTC3lY74Cj"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "O4o1-vvM74Cl"
  },
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1])"
  ],
  "id": "O4o1-vvM74Cl"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "wvDLTAxP74Cl"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "wvDLTAxP74Cl"
},
{
  "cell_type": "code",
  "execution_count": null,
```

```
"metadata": {
  "colab": {
    "base_uri": "https://localhost:8080/",
    "height": 81
  },
  "id": "sNEsS1fg74Cl",
  "outputId": "ba70dbe3-2d8e-4c01-c66f-f1fe898a4c0d"
},
"outputs": [
  {
    "data": {
      "text/html": [
        "\n",
        "  <div id=\"df-77f30cc2-dbd6-4e94-85fa-
↪ ea199c2a8fcb\">\n",
        "    <div class=\"colab-df-container\">\n",
        "      <div>\n",
        "<style scoped>\n",
        "    .dataframe tbody tr th:only-of-type {\n",
        "        vertical-align: middle;\n",
        "    }\n",
        "\n",
        "    .dataframe tbody tr th {\n",
        "        vertical-align: top;\n",
        "    }\n",
        "\n",
        "    .dataframe thead th {\n",
        "        text-align: right;\n",
        "    }\n",
        "</style>\n",
        "<table border=\"1\" class=\"dataframe\">\n",
        "  <thead>\n",
        "    <tr style=\"text-align: right;\">\n",
        "      <th></th>\n",
        "      <th>Cross-validation score</th>\n",
        "      <th>Test Score</th>\n",
        "      <th>Execution Time</th>\n",
        "      <th>Best Paramaters</th>\n",
        "    </tr>\n",
        "  </thead>\n",
        "  <tbody>\n",
        "    <tr>\n",
        "      <th>0</th>\n",
        "      <td>0.734678</td>\n",
        "      <td>0.743603</td>\n",
        "      <td>9171.80990409851</td>\n",
        "      <td>{’clf__C’: 0.001, ’clf__max_iter’:
↪ 1000, ’clf_...</td>\n",
        "    </tr>\n",
        "  </tbody>\n",
        "</table>\n",
        "</div>\n",
        "      <button class=\"colab-df-convert\" onclick
↪ =\"convertToInteractive(’df-77f30cc2-dbd6-4e94-85fa-
↪ ea199c2a8fcb’)\"\n",
        "              title=\"Convert this dataframe to
↪ an interactive table.\"\n",
        "              style=\"display:none;\">\n",
        "        \n",
        "  <svg xmlns=\"http://www.w3.org/2000/svg\"
↪ height=\"24px\"viewBox=\"0 0 24 24\"\n",
        "       width=\"24px\">\n",
        "    <path d=\"M0 0h24v24H0V0z\" fill=\"none\"/>\
↪ n",
        "    <path d=\"M18.56 5.441.94 2.06.94-2.06
↪ 2.06-.94-2.06-.94-.94-2.06-.94 2.06-2.06.94zm-11 1L8.5
↪  8.51.94-2.06 2.06-.94-2.06-.94L8.5 2.51-.94
↪ 2.06-2.06.94zm10 10l.94 2.06.94-2.06
↪ 2.06-.94-2.06-.94-.94-2.06-.94 2.06-2.06.94z\"/><path
↪ d=\"M17.41 7.96l-1.37-1.37c-.4-.4-.92-.59-1.43-.59-.52
↪  0-1.04.2-1.43.59L10.3 9.45l-7.72 7.72c-.78.78-.78
↪ 2.05 0 2.83L4 21.41c.39.39.9.59 1.41.59.51 0 1.02-.2
↪ 1.41-.59l7.78-7.78 2.81-2.81c.8-.78.8-2.07 0-2.86zM5
↪ .41 20L4 18.59l7.72-7.72 1.47 1.35L5.41 20z\"/>\n",
        "  </svg>\n",
        "      </button>\n",
        "      \n",
        "  <style>\n",
        "    .colab-df-container {\n",
        "      display:flex;\n",
        "      flex-wrap:wrap;\n",
        "      gap: 12px;\n",
        "    }\n",
        "\n",
        "    .colab-df-convert {\n",
        "      background-color: #E8F0FE;\n",
        "      border: none;\n",
        "      border-radius: 50%;\n",
        "      cursor: pointer;\n",
        "      display: none;\n",
        "      fill: #1967D2;\n",
        "      height: 32px;\n",
        "      padding: 0 0 0 0;\n",
        "      width: 32px;\n",
        "    }\n",
        "\n",
        "    .colab-df-convert:hover {\n",
        "      background-color: #E2EBFA;\n",
        "      box-shadow: 0px 1px 2px rgba(60, 64, 67,
↪ 0.3), 0px 1px 3px 1px rgba(60, 64, 67, 0.15);\n",
        "      fill: #174EA6;\n",
        "    }\n",
        "\n",
        "    [theme=dark] .colab-df-convert {\n",
        "      background-color: #3B4455;\n",
        "      fill: #D2E3FC;\n",
        "    }\n",
        "\n",
        "    [theme=dark] .colab-df-convert:hover {\n",
        "      background-color: #434B5C;\n",
        "      box-shadow: 0px 1px 3px 1px rgba(0, 0, 0,
↪ 0.15);\n",
        "      filter: drop-shadow(0px 1px 2px rgba(0, 0,
↪  0, 0.3));\n",
        "      fill: #FFFFFF;\n",
        "    }\n",
        "  </style>\n",
        "\n",
        "      <script>\n",
        "        const buttonEl =\n",
```

```
        "          document.querySelector('#df-77f30cc2-
↪ dbd6-4e94-85fa-ea199c2a8fcb button.colab-df-convert')
↪ ;\n",
        "        buttonEl.style.display =\n",
        "          google.colab.kernel.accessAllowed ? '
↪ block' : 'none';\n",
        "\n",
        "        async function convertToInteractive(key)
↪  {\n",
        "          const element = document.querySelector
↪ ('#df-77f30cc2-dbd6-4e94-85fa-ea199c2a8fcb');\n",
        "          const dataTable =\n",
        "            await google.colab.kernel.
↪ invokeFunction('convertToInteractive',\n",
        "
↪       [key], {});\n",
        "          if (!dataTable) return;\n",
        "\n",
        "          const docLinkHtml = 'Like what you see
↪ ? Visit the ' +\n",
        "            '<a target=\"_blank\" href=https://
↪ colab.research.google.com/notebooks/data_table.ipynb>
↪ data table notebook</a>'\n",
        "            + ' to learn more about interactive
↪ tables.';\n",
        "          element.innerHTML = '';\n",
        "          dataTable['output_type'] = '
↪ display_data';\n",
        "          await google.colab.output.renderOutput
↪ (dataTable, element);\n",
        "          const docLink = document.createElement
↪ ('div');\n",
        "          docLink.innerHTML = docLinkHtml;\n",
        "          element.appendChild(docLink);\n",
        "        }\n",
        "      </script>\n",
        "    </div>\n",
        "  </div>\n",
        "  "
      ],
      "text/plain": [
        "   Cross-validation score  Test Score
↪ Execution Time  \\\n",
        "0               0.734678    0.743603
↪ 9171.80990409851   \n",
        "\n",
        "                                      Best
↪ Paramaters  \n",
        "0  {'clf__C': 0.001, 'clf__max_iter': 1000, '
↪ clf_...  "
      ]
    },
    "execution_count": 72,
    "metadata": {},
    "output_type": "execute_result"
  }
],
"source": [
  "output_df"
],
  "id": "sNEsS1fg74Cl"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "HbPKJol074Cl"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('knn_lr_out_imb.csv')"
  ],
  "id": "HbPKJol074Cl"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "sBbv2fKOcUF4"
  },
  "source": [
    "#### Random Forest"
  ],
  "id": "sBbv2fKOcUF4"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "zsQh4KYvmsy3"
  },
  "source": [
    "##### Simple"
  ],
  "id": "zsQh4KYvmsy3"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "4aSyXDsccXkd"
  },
  "outputs": [],
  "source": [
    "simple_rf_out, simple_rf_pipes = create_pipes(
↪ ht_simple, rf_clf, rf_param_grid, X_train, y_train,
↪ X_test, y_test)"
  ],
  "id": "4aSyXDsccXkd"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "aERa5JL-uYDo"
  },
  "outputs": [],
  "source": [
    "simple_rf_out.to_csv('simple_rf_out_imb.csv')"
  ],
  "id": "aERa5JL-uYDo"
},
```

```
{
  "cell_type": "markdown",
  "metadata": {
    "id": "r7MfPDJNAxP8"
  },
  "source": [
    "##### KNN Imputation\n"
  ],
  "id": "r7MfPDJNAxP8"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "J0WN3L5yAw4x"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_knn(smote, rf_clf, False,
  ↪ rf_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "J0WN3L5yAw4x"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "BNYw6GA3OtLa"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_knn(smote_tomek, rf_clf, False,
  ↪ rf_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "BNYw6GA3OtLa"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Lj8mMeTnG8-P"
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_knn(smote, rf_clf, True,
  ↪ rf_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "Lj8mMeTnG8-P"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Hn9nLgqSHLtF"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_knn(smote_tomek, rf_clf, True,
  ↪ rf_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "Hn9nLgqSHLtF"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "F-3LGIy3Jzy7"
  },
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1,out2,out3,out4])"
  ],
  "id": "F-3LGIy3Jzy7"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "BkKa8UdhJ5OV"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "BkKa8UdhJ5OV"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "MJS3hpCuJ67y"
  },
  "outputs": [],
  "source": [
    "output_df"
  ],
  "id": "MJS3hpCuJ67y"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "f8i0cmtHKBj6"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('knn_rf_out_imb.csv')"
  ],
  "id": "f8i0cmtHKBj6"
},
{
```

```
  "cell_type": "markdown",
  "metadata": {
    "id": "DtYYtEsgrC1D"
  },
  "source": [
    "####SVM"
  ],
  "id": "DtYYtEsgrC1D"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "y6OFP8x3rE6n"
  },
  "outputs": [],
  "source": [
    "simple_svm_out, simple_svm_pipes = create_pipes(
  ↪ ht_simple, svm_clf, svm_param_grid, X_train, y_train,
  ↪ X_test, y_test)"
  ],
  "id": "y6OFP8x3rE6n"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "5jEar_2VrKOZ"
  },
  "outputs": [],
  "source": [
    "knn_svm_out, knn_svm_pipes = create_pipes(ht_knn,
  ↪ svm_clf, svm_param_grid, X_train, y_train, X_test,
  ↪ y_test)"
  ],
  "id": "5jEar_2VrKOZ"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "48_kSBQ082DU"
  },
  "source": [
    "##### Simple"
  ],
  "id": "48_kSBQ082DU"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8O4bDeVhuKui"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_simple(smote, svm_clf, False,
  ↪ svm_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "8O4bDeVhuKui"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ikqWDpd-82Dm"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_simple(smote_tomek, svm_clf, False,
  ↪ svm_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "ikqWDpd-82Dm"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "colab": {
      "background_save": true
    },
    "id": "HqY_BI5S82Dn"
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_simple(smote, svm_clf, True,
  ↪ svm_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "HqY_BI5S82Dn"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "colab": {
      "background_save": true
    },
    "id": "jrFJeM8x82Do"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_simple(smote_tomek, svm_clf, True
  ↪ , svm_param_grid, X_train, y_train, X_test, y_test)\n
  ↪ ",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "jrFJeM8x82Do"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "wBR12pGy82Dw"
  },
```

```
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1,out2,out3,out4])"
  ],
  "id": "wBR12pGy82Dw"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "D15TD1-G82Dx"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "D15TD1-G82Dx"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "1o8WQCcL82Dx"
  },
  "outputs": [],
  "source": [
    "output_df"
  ],
  "id": "1o8WQCcL82Dx"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "gJDhx6AW82Dy"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('simple_svm_out_imb.csv')"
  ],
  "id": "gJDhx6AW82Dy"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "F7bZJEqyf_vh"
  },
  "source": [
    "##### KNN Imputation\n"
  ],
  "id": "F7bZJEqyf_vh"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Sf0DvOx3f_vi"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_knn(smote, svm_clf, False,
↪ svm_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "Sf0DvOx3f_vi"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "3J4TNaRjf_vj"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_knn(smote_tomek, svm_clf, False,
↪ svm_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "3J4TNaRjf_vj"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Msre9dCsf_vj"
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_knn(smote, svm_clf, True,
↪ svm_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "Msre9dCsf_vj"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "WntOeMf6f_vj"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_knn(smote_tomek, svm_clf, True,
↪ svm_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "WntOeMf6f_vj"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "6JsiRXdnf_vn"
  },
  "outputs": [],
```

```
    "source": [
      "results=[]\n",
      "results.extend([out1,out2,out3,out4])"
    ],
    "id": "6JsiRXdnf_vn"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "D5hfLsMvf_vn"
    },
    "outputs": [],
    "source": [
      "output_df = pd.DataFrame(results)"
    ],
    "id": "D5hfLsMvf_vn"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "-7zzmiE8f_vn"
    },
    "outputs": [],
    "source": [
      "output_df"
    ],
    "id": "-7zzmiE8f_vn"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "6Fk44N4pf_vo"
    },
    "outputs": [],
    "source": [
      "output_df.to_csv('knn_svm_out_imb.csv')"
    ],
    "id": "6Fk44N4pf_vo"
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "I3TG6kEnHmmf"
    },
    "source": [
      "#### ANN\n"
    ],
    "id": "I3TG6kEnHmmf"
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "solq2zcW_Fzr"
    },
    "source": [
      "##### Simple"
    ],
    "id": "solq2zcW_Fzr"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "VQPCo3pSZNtU"
    },
    "outputs": [],
    "source": [
      "out1, smote_ = ht_simple(smote, mlp_clf, False,
      ↪ mlp_param_grid, X_train, y_train, X_test, y_test)\n",
      "print(\"---pipe1 done---\")\n",
      "print(out1)"
    ],
    "id": "VQPCo3pSZNtU"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "lJzU8XZ5-07L"
    },
    "outputs": [],
    "source": [
      "out2, smotet_= ht_simple(smote_tomek, mlp_clf, False,
      ↪ mlp_param_grid, X_train, y_train, X_test, y_test)\n",
      "print(\"---pipe2 done---\")\n",
      "print(out2)"
    ],
    "id": "lJzU8XZ5-07L"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "QXcD_fg-9ht0"
    },
    "outputs": [],
    "source": [
      "out3, smote_fs = ht_simple(smote, mlp_clf, True,
      ↪ mlp_param_grid, X_train, y_train, X_test, y_test)\n",
      "print(\"---pipe3 done---\")\n",
      "print(out3)"
    ],
    "id": "QXcD_fg-9ht0"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "VwFAAS91_Hmw"
    },
    "outputs": [],
    "source": [
      "out4, smotet_fs = ht_simple(smote_tomek, mlp_clf, True
      ↪ , mlp_param_grid, X_train, y_train, X_test, y_test)\n
      ↪ ",
      "print(\"---all pipes done---\")\n",
      "print(out4)"
```

```
  ],
  "id": "VwFAAS91_Hmw"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "TnNnt7Qn8q50"
  },
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1,out2,out3,out4])"
  ],
  "id": "TnNnt7Qn8q50"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ow6fQvj58tq2"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "ow6fQvj58tq2"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "FXT4k5LM8woO"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('simple_mlp_out_imb.csv')"
  ],
  "id": "FXT4k5LM8woO"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "FKS14j4i2oU9"
  },
  "source": [
    "##### KNN Imputation\n"
  ],
  "id": "FKS14j4i2oU9"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "JMv2GAiP2oU-"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_knn(smote, mlp_clf, False,
    ↪ mlp_param_grid, X_train, y_train, X_test, y_test)\n",
    "  "print(\"---pipe1 done---\")\n",
    "  "print(out1)"
  ],
  "id": "JMv2GAiP2oU-"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "EhCL5hHp2oU-"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_knn(smote_tomek, mlp_clf, False,
    ↪ mlp_param_grid, X_train, y_train, X_test, y_test)\n",
    "  "print(\"---pipe2 done---\")\n",
    "  "print(out2)"
  ],
  "id": "EhCL5hHp2oU-"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ISeyEh1A2oU-"
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_knn(smote,mlp_clf, True,
    ↪ mlp_param_grid, X_train, y_train, X_test, y_test)\n",
    "  "print(\"---pipe3 done---\")\n",
    "  "print(out3)"
  ],
  "id": "ISeyEh1A2oU-"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "jQafa3Uf2oU-"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_knn(smote_tomek, mlp_clf, True,
    ↪ mlp_param_grid, X_train, y_train, X_test, y_test)\n",
    "  "print(\"---all pipes done---\")\n",
    "  "print(out4)"
  ],
  "id": "jQafa3Uf2oU-"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "l11dE5Ay2oU_"
  },
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1,out2,out3,out4])"
```

```
  ],
  "id": "l11dE5Ay2oU_"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "_kjwW1VD2oU_"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "_kjwW1VD2oU_"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "PqDA-F3Z2oU_"
  },
  "outputs": [],
  "source": [
    "output_df"
  ],
  "id": "PqDA-F3Z2oU_"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "1NC-CAck2oU_"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('knn_mlp_out_imb.csv')"
  ],
  "id": "1NC-CAck2oU_"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "WXWhjuGtqyn1"
  },
  "source": [
    "#### XGBoost"
  ],
  "id": "WXWhjuGtqyn1"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "_XBcakNOminh"
  },
  "source": [
    "##### Simple"
  ],
  "id": "_XBcakNOminh"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "9C89T_urqxx-"
  },
  "outputs": [],
  "source": [
    "simple_xgb_out, simple_xgb_pipes = create_pipes(
    ↪ ht_simple, xgb_clf, xgb_param_grid, X_train, y_train,
    ↪ X_test, y_test)"
  ],
  "id": "9C89T_urqxx-"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "br2bV68eGefL"
  },
  "outputs": [],
  "source": [
    "simple_xgb_out.to_csv(\"simple_xgb_out_imb.csv\")"
  ],
  "id": "br2bV68eGefL"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Kngsmi5Otht1"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_simple(smote, xgb_clf, False,
    ↪ xgb_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "Kngsmi5Otht1"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "9EPEUcEy78O6"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_simple(smote_tomek, xgb_clf, False,
    ↪ xgb_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "9EPEUcEy78O6"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "5Evzd1EhG9VK"
```

```
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_simple(smote,xgb_clf, True,
  ↪ xgb_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "5Evzd1EhG9VK"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "rCo4BftBHBmb"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_simple(smote_tomek, xgb_clf, True
  ↪ , xgb_param_grid, X_train, y_train, X_test, y_test)\n
  ↪ ",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "rCo4BftBHBmb"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Wr7j_sHOVQEW"
  },
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1,out2,out3,out4])"
  ],
  "id": "Wr7j_sHOVQEW"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "MRRON8-5VTjz"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "MRRON8-5VTjz"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "_2EWPNrHVXY7"
  },
  "outputs": [],
  "source": [
    "output_df"
  ],
  "id": "_2EWPNrHVXY7"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "VHaVav8el4Fj"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('simple_xgb_out_imb.csv')"
  ],
  "id": "VHaVav8el4Fj"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "DtyhKWkcYf1_"
  },
  "source": [
    "##### KNN Imputation"
  ],
  "id": "DtyhKWkcYf1_"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Fj8qFxIBYf2A"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_knn(smote, xgb_clf, False,
  ↪ xgb_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "Fj8qFxIBYf2A"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "yGuWX5sHYf2A"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_knn(smote_tomek, xgb_clf, False,
  ↪ xgb_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "yGuWX5sHYf2A"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8uF7ErjbYf2A"
```

```
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_knn(smote,xgb_clf, True,
  ↪ xgb_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "8uF7ErjbYf2A"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "66pFyu7ZYf2B"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_knn(smote_tomek, xgb_clf, True,
  ↪ xgb_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "66pFyu7ZYf2B"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Z25Jw042Yf2B"
  },
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1,out2])"
  ],
  "id": "Z25Jw042Yf2B"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "CS8eTABBYf2B"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "CS8eTABBYf2B"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "6lgkTUKGYf2B"
  },
  "outputs": [],
  "source": [
    "output_df"
  ],
  "id": "6lgkTUKGYf2B"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "sxzsN4c3Yf2B"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('knn_xgb_out_imb.csv')"
  ],
  "id": "sxzsN4c3Yf2B"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "s5aNpUvAFkCF"
  },
  "source": [
    "#### Adaboost"
  ],
  "id": "s5aNpUvAFkCF"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "Kmi8t1IimpDw"
  },
  "source": [
    "##### Simple"
  ],
  "id": "Kmi8t1IimpDw"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "KFANGM1ZFmTV"
  },
  "outputs": [],
  "source": [
    "simple_ada_out, simple_ada_pipes = create_pipes(
  ↪ ht_simple, ada_clf, ada_param_grid, X_train, y_train,
  ↪ X_test, y_test)"
  ],
  "id": "KFANGM1ZFmTV"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "xuf_4s8iQupV"
  },
  "outputs": [],
  "source": [
    "simple_ada_out.to_csv(\"simple_ada_out_imb.csv\")"
  ],
  "id": "xuf_4s8iQupV"
},
```

```
{
  "cell_type": "markdown",
  "metadata": {
    "id": "123EnoT7SCkJ"
  },
  "source": [
    "##### KNN Imputation"
  ],
  "id": "123EnoT7SCkJ"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "L0QZnTANSCkJ"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_knn(smote, ada_clf, False,
  ↪ ada_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "L0QZnTANSCkJ"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "RSjdMr5cSCkK"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_knn(smote_tomek, ada_clf, False,
  ↪ ada_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "RSjdMr5cSCkK"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "HjgX9b8gSCkK"
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_knn(smote,ada_clf, True,
  ↪ ada_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "HjgX9b8gSCkK"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "G4gCz8NJSCkK"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_knn(smote_tomek, ada_clf, True,
  ↪ ada_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "G4gCz8NJSCkK"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "jlFByqYTSCkK"
  },
  "outputs": [],
  "source": [
    "results=[]\n",
    "results.extend([out1,out2,out3,out4])"
  ],
  "id": "jlFByqYTSCkK"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "_Wy7Z6ZGSCkL"
  },
  "outputs": [],
  "source": [
    "output_df = pd.DataFrame(results)"
  ],
  "id": "_Wy7Z6ZGSCkL"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "D-ciR8zBSCkL"
  },
  "outputs": [],
  "source": [
    "output_df"
  ],
  "id": "D-ciR8zBSCkL"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "-9Gznz-YSCkL"
  },
  "outputs": [],
  "source": [
    "output_df.to_csv('knn_ada_out_imb.csv')"
  ],
  "id": "-9Gznz-YSCkL"
},
{
```

```
  "cell_type": "markdown",
  "metadata": {
    "id": "Rl6GcLYVGCnF"
  },
  "source": [
    "#### KNN"
  ],
  "id": "Rl6GcLYVGCnF"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "RA8f-n_OmmjK"
  },
  "source": [
    "##### Simple"
  ],
  "id": "RA8f-n_OmmjK"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "XoOk8YpgGDfn"
  },
  "outputs": [],
  "source": [
    "simple_knn_out, simple_knn_pipes = create_pipes(
  ↪ ht_simple, knn_clf, knn_param_grid, X_train, y_train,
  ↪ X_test, y_test)"
  ],
  "id": "XoOk8YpgGDfn"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "gkuERQzKtIH5"
  },
  "outputs": [],
  "source": [
    "simple_knn_out.to_csv(\"simple_knn_out_imb.csv\")"
  ],
  "id": "gkuERQzKtIH5"
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "2yHF_38cLoWB"
  },
  "source": [
    "##### KNN Imputation"
  ],
  "id": "2yHF_38cLoWB"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "GqKZH6EuLoWC"
  },
  "outputs": [],
  "source": [
    "out1, smote_ = ht_knn(smote, knn_clf, False,
  ↪ knn_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe1 done---\")\n",
    "print(out1)"
  ],
  "id": "GqKZH6EuLoWC"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8OJPZIoALoWD"
  },
  "outputs": [],
  "source": [
    "out2, smotet_= ht_knn(smote_tomek, knn_clf, False,
  ↪ knn_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe2 done---\")\n",
    "print(out2)"
  ],
  "id": "8OJPZIoALoWD"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "fo6wwx9OLoWD"
  },
  "outputs": [],
  "source": [
    "out3, smote_fs = ht_knn(smote, knn_clf, True,
  ↪ knn_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---pipe3 done---\")\n",
    "print(out3)"
  ],
  "id": "fo6wwx9OLoWD"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ztCUqTSmLoWE"
  },
  "outputs": [],
  "source": [
    "out4, smotet_fs = ht_knn(smote_tomek, knn_clf, True,
  ↪ knn_param_grid, X_train, y_train, X_test, y_test)\n",
    "print(\"---all pipes done---\")\n",
    "print(out4)"
  ],
  "id": "ztCUqTSmLoWE"
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "y3sp4RQELoWE"
```

```
    },
    "outputs": [],
    "source": [
      "results=[]\n",
      "results.extend([out1,out2,out3])"
    ],
    "id": "y3sp4RQELoWE"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "jUb6d2tLLoWF"
    },
    "outputs": [],
    "source": [
      "output_df = pd.DataFrame(results)"
    ],
    "id": "jUb6d2tLLoWF"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "vdrCHq7ELoWF"
    },
    "outputs": [],
    "source": [
      "output_df"
    ],
    "id": "vdrCHq7ELoWF"
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "NwVzYkKNLoWG"
    },
    "outputs": [],
    "source": [
      "output_df.to_csv('knn_knn_out_imb.csv')"
    ],
    "id": "NwVzYkKNLoWG"
  }
],
"metadata": {
  "colab": {
    "collapsed_sections": [
      "25760d29",
      "53c91bd2",
      "rvFOnSy8sEWK",
      "T1A5hUllZJLS",
      "mPAP2kIyGOjU"
    ],
    "provenance": [],
    "toc_visible": true
  },
  "gpuClass": "standard",
  "kernelspec": {
    "display_name": "Python 3",
    "name": "python3"
  },
  "language_info": {
    "name": "python"
  }
},
"nbformat": 4,
"nbformat_minor": 5
}
```


A..6 EDA and Evaluation


```
{
  "nbformat": 4,
  "nbformat_minor": 0,
  "metadata": {
    "colab": {
      "provenance": [],
      "toc_visible": true
    },
    "kernelspec": {
      "name": "python3",
      "display_name": "Python 3"
    },
    "language_info": {
      "name": "python"
    }
  },
  "cells": [
    {
      "cell_type": "markdown",
      "source": [
        "**Final EDA and Model Evaluation**"
      ],
      "metadata": {
        "id": "sYvxYQrD8RKJ"
      }
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "8WjKosvp8L1W"
      },
      "outputs": [],
      "source": [
        "import pandas as pd\n",
        "import numpy as np"
      ]
    },
    {
      "cell_type": "code",
      "execution_count": null,
      "metadata": {
        "id": "235b5971"
      },
      "outputs": [],
      "source": [
        "import matplotlib.pyplot as plt\n",
        "import seaborn as sns\n",
```

```
    "from sklearn.model_selection import train_test_split"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "25760d29"
   },
   "source": [
    "## Basic Info About the Data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "5cbf54eb"
   },
   "outputs": [],
   "source": [
    "df = pd.read_csv(\"dataset.csv\")\n",
    "df.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "95d848f2"
   },
   "outputs": [],
   "source": [
    "print(\"Shape:\",df.shape)\n",
    "print()\n",
    "print(df.info())"
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "*There are 43,400 rows and 12 columns.Among the
   ↪ features, there are 4 which are of datatype int64 (id,
   ↪  hypertension, heart_disease and smoking status, the
   ↪ target feature), 3 of which are of float64 (age,
   ↪ avg_glucose_level and bmi) and the rest are of object
   ↪ datatype.*\n",
    "\n",
    "*It can be observed that there are null rows in
   ↪ features bmi( approx. 3.37%) and smoking_status (
   ↪ approx. 30.63%).*"
   ],
   "metadata": {
    "id": "2yr-Fhgv3JOg"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "display(df.describe().round(2))"
   ],
   "metadata": {
    "id": "homQAAkUDGFw"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "*The above display summarizes the statistics of the
   ↪ features with numerical values. Features hypertension,
   ↪  heart_disease and stroke has the same min and max
   ↪ values (0,1) since they are contain binary values.
   ↪ Features age, avg_glucose_level and bmi are continuous
   ↪ , with the following min and max values:*\n",
    "*   *age : (0.08,82.00)*\n",
    "*   *avg_glucose_level: (55.00,291.05)*\n",
    "*   *bmi: (10.10,97.60)*\n",
    "\n",
    "\n"
   ],
   "metadata": {
    "id": "TcB4zmKS9A4p"
   }
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "7b122840"
   },
   "outputs": [],
   "source": [
    "df.nunique()"
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "*Almost all of the categorical features have 2 unique
   ↪ values except work_type, smoking_status and gender.*"
   ],
   "metadata": {
    "id": "RD6cMUxOHhUP"
   }
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "18692b15"
   },
   "outputs": [],
   "source": [
    "column_names = list(df.columns.values)\n",
    "\n",
    "for col in column_names:\n",
    "  if df[col].dtype == 'object':\n",
    "    print(df[col].unique())"
   ]
```

```
},
{
  "cell_type": "code",
  "source": [
    "for col in column_names:\n",
    "  print(col)\n",
    "  print(df[col].unique())"
  ],
  "metadata": {
    "id": "ylNBGcV9U35N"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "6a9cc9bf"
  },
  "source": [
    "<b>Is the data clean enough for data visualization and
  ↪ exploration?</b>"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "30c5b30d"
  },
  "source": [
    "Check for duplicates"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "031781ce"
  },
  "outputs": [],
  "source": [
    "dups = df.duplicated()\n",
    "print(dups.any())"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "748dcbac"
  },
  "source": [
    "Missing Values in raw data"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8ad709ff"
  },
  "outputs": [],
  "source": [
    "print(df.isnull().sum())"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "3398d534"
  },
  "source": [
    "Check for outliers"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "a9956a99"
  },
  "outputs": [],
  "source": [
    "df.gender.unique()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "781327fa"
  },
  "source": [
    "# Data Preparation"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "674d3f88"
  },
  "source": [
    "Imputation and transformation of data into applicable
  ↪ ones (cat to numerical)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "30a884e2"
  },
  "outputs": [],
  "source": [
    "# Patient ID is irrelevant hence dropped from the
  ↪ dataset used.\n",
    "X = df.iloc[:,1:len(column_names)-1]\n",
    "y = pd.DataFrame(df.iloc[:,-1])"
  ]
},
{
  "cell_type": "code",
```

```
  "execution_count": null,
  "metadata": {
    "id": "63d7004c"
  },
  "outputs": [],
  "source": [
    "print(X)\n",
    "print(y)"
  ]
},
{
  "cell_type": "code",
  "source": [
    "counter = Counter(y['stroke'])\n",
    "counter"
  ],
  "metadata": {
    "id": "6LU-x7gKggFC"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "2f51f7ff"
  },
  "outputs": [],
  "source": [
    "X_train, X_test, y_train, y_test = train_test_split(X,
  ↪ y, test_size = 0.2, random_state=100)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "591f9cc5"
  },
  "outputs": [],
  "source": [
    "print(\"X Train: \", X_train.shape)\n",
    "print(\"X Test: \", X_test.shape)\n",
    "print(\"y Train: \", y_train.shape)\n",
    "print(\"y Test: \", y_test.shape)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "4a97e303"
  },
  "outputs": [],
  "source": [
    "orig_X_train=X_train.copy()\n",
    "orig_y_train=y_train.copy()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "53c91bd2"
  },
  "source": [
    "#### Outlier Removal"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "408c5a51"
  },
  "source": [
    "Before imputation, data needs to be encoded and
  ↪ irrelevant info should be removed(outliers). Only
  ↪ applied on training data."
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "According to Liu, Fan, Wu (2019),\n",
    "*   Minimum Age for Smoke-Monitoring is 25 years old\n
  ↪ ",
    "*   Reference Values of BMI should be no more than
  ↪ 60%\n",
    "\n",
    "Hence, to remove outliers and noise, data where age <
  ↪ 25 and BMI > 60% should be removed."
  ],
  "metadata": {
    "id": "BDffevM4Mz_9"
  }
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "670f8311"
  },
  "source": [
    "##### Gender:Other"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "016ef74e"
  },
  "outputs": [],
  "source": [
    "X_train.gender.unique()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
```

```
    "id": "a51cc68c"
   },
   "outputs": [],
   "source": [
    "other_results = X_train.loc[X_train['gender']  != '
   ↪ Other']\n",
    "other_results.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "8fd14fd0"
   },
   "outputs": [],
   "source": [
    "X_train= X_train.loc[other_results.index]\n",
    "X_train.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "9285cbad"
   },
   "outputs": [],
   "source": [
    "X_train.gender.unique()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "0badafa1"
   },
   "source": [
    "##### Age>=25%"
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "print(X_train.isnull().sum())"
   ],
   "metadata": {
    "id": "FOnIt3EXKXN2"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "0bc4cc30"
   },
   "outputs": [],
   "source": [
    "#visualize age boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='age').set(title='Age')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "50724a87"
   },
   "outputs": [],
   "source": [
    "age_results = X_train.loc[X_train['age'] >= 25]\n",
    "age_results.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "5f67a807"
   },
   "outputs": [],
   "source": [
    "X_train = X_train.loc[age_results.index]\n",
    "X_train.shape"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "cadadc20"
   },
   "outputs": [],
   "source": [
    "#revisualize age boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='age').set(title='Age')"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "TbycuEt-XK1N"
   },
   "outputs": [],
   "source": [
    "print(X_train.isnull().sum())"
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "*Missing values:*\n",
    "\n",
    "*Before (BMI):  1188  | After (BMI):  1040  |
   ↪ Difference:  148*\n",
```

```
    "\n",
    "*Before(Smoking Status):  10635  | After (Smoking
   ↪ Status):  5046  | Difference:  5589*"
  ],
  "metadata": {
    "id": "z44YPPQVNs_h"
  }
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "498c9a6c"
  },
  "source": [
    "##### BMI<=60% (With the rows with missing values)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ad55f0a4"
  },
  "outputs": [],
  "source": [
    "#visualize bmi boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='bmi').set(title='BMI')"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "EDsdbCghSmbp"
  },
  "outputs": [],
  "source": [
    "more_than = X_train.loc[X_train['bmi'] > 60]"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "1oWhna8EJN_d"
  },
  "outputs": [],
  "source": [
    "#remove bmi reference values more than 60\n",
    "bmi_results = X_train[~X_train.index.isin(more_than.
   ↪ index)]"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "D9jcbZSJI9Vc"
  },
  "outputs": [],
  "source": [
    "X_train.shape"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "3IFHkewfVIB8"
  },
  "outputs": [],
  "source": [
    "print(bmi_results.shape)\n",
    "print(more_than.shape)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "lKHwJXMxP11S"
  },
  "outputs": [],
  "source": [
    "more_than.isnull().sum()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "5O2__cQyTHgz"
  },
  "outputs": [],
  "source": [
    "bmi_results.isnull().sum()"
  ]
},
{
  "cell_type": "code",
  "source": [
    "more_than = bmi_results.loc[bmi_results['bmi'] > 60]"
  ],
  "metadata": {
    "id": "mc7ZQXr6JzIV"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "more_than"
  ],
  "metadata": {
    "id": "U23irEBvJ4RE"
  },
  "execution_count": null,
  "outputs": []
```

```
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "uf5k9sY6UTJu"
  },
  "outputs": [],
  "source": [
    "X_train.isnull().sum()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "b784e226"
  },
  "outputs": [],
  "source": [
    "X_train = X_train.loc[bmi_results.index]\n",
    "X_train.shape"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "qa5FOJvWXSy4"
  },
  "outputs": [],
  "source": [
    "print(X_train.isnull().sum())"
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "*Missing Values:*\n",
    "\n",
    "*Before (BMI): 1040 | After (BMI): 1040 |
↪ Difference: 0*\n",
    "\n",
    "*Before(Smoking Status): 5046 | After (Smoking
↪ Status): 5031 | Difference: 15*"
  ],
  "metadata": {
    "id": "DYa6cn3wPpmX"
  }
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "c9e0e8d7"
  },
  "outputs": [],
  "source": [
    "#revisualize bmi boxplot\n",
    "plt.figure(figsize = (10,10))\n",
    "sns.boxplot(data=X_train, x='bmi').set(title='BMI')"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "a6dd348e"
  },
  "outputs": [],
  "source": [
    "X_train.work_type.unique()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "ff857a43"
  },
  "source": [
    "*Since Ages < 25 has been removed, there are no more
↪ values that pertains to children hence the category '
↪ children' of feature 'work_type' has been removed.*"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "5e7c075c"
  },
  "source": [
    "Filter y with the indexes of the X_train"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "80e19575"
  },
  "outputs": [],
  "source": [
    "y_train = y_train.loc[X_train.index]"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "962b445e"
  },
  "outputs": [],
  "source": [
    "print(\"Filtered Shapes Before and After Removal\")\n
↪ ",
    "print(\"Before| X_train: \", orig_X_train.shape, \" |
↪ y_train: \", orig_y_train.shape)\n",
    "print(\"After| X_train: \", X_train.shape, \" |
↪ y_train: \", y_train.shape)"
  ]
```

```
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "7d2b5eb4"
  },
  "source": [
    "#### Customizing ColumnTransfer"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "187b2ec5"
  },
  "source": [
    "Separating categorical and numerical data for encoding
  ↪ ."
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "96203e27"
  },
  "outputs": [],
  "source": [
    "cat_data = X_train.select_dtypes(include=['object','
  ↪ int64'])"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d71f6ccc",
    "scrolled": true
  },
  "outputs": [],
  "source": [
    "cat_data"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "6c2d7e48"
  },
  "source": [
    ">* hypertension and heart_disease can be left as is.\n
  ↪ ",
    ">* gender, ever_married, work_type and residence_type
  ↪ needs to be one hot encoded(dummy variable trap
  ↪ prevention).\n",
    ">* smoking status needs to be ordinal encoded before
  ↪ imputed since knnimputer only works on numerical
  ↪ values."
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "85c1eaa7"
  },
  "outputs": [],
  "source": [
    "pass_feats = ['hypertension','heart_disease']\n",
    "ohe_feats = ['gender','ever_married','work_type','
  ↪ Residence_type']"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d52d9b5d"
  },
  "outputs": [],
  "source": [
    "num_data = X_train.select_dtypes(include=['float64'])"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "710168df",
    "scrolled": true
  },
  "outputs": [],
  "source": [
    "num_data"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "f24f8363"
  },
  "source": [
    "Create a Pipeline for 2 different imputation methods
  ↪ but with the same feature engineering techniques(does
  ↪ not contain the scaling, feature selection and model
  ↪ since we need to visualize the cleaned data)."
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "6d4cdb8f"
  },
  "outputs": [],
  "source": [
    "from sklearn.impute import SimpleImputer, KNNImputer\n
  ↪ ",
    "from sklearn.preprocessing import OneHotEncoder,
  ↪ RobustScaler, OrdinalEncoder\n",
```

```
    "# from sklearn.pipeline import Pipeline\n",
    "from imblearn.pipeline import Pipeline as imbpipeline\
↪ n",
    "from sklearn.compose import ColumnTransformer,
↪ make_column_transformer"
  ]
},
{
  "cell_type": "code",
  "source": [
    "from imblearn.over_sampling import SMOTE\n",
    "from imblearn.combine import SMOTETomek"
  ],
  "metadata": {
    "id": "w1PSTxDMBQ6i"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "1b1ace7b"
  },
  "outputs": [],
  "source": [
    "from sklearn import set_config"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "24d8dfdb"
  },
  "outputs": [],
  "source": [
    "num_data_col = num_data.columns.tolist()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "549cb706"
  },
  "source": [
    "### <b>1st Imputation Method<b/>\n",
    ">Impute both bmi and smoke. Next, encode the
↪ categorical smoking_status feature then one-hot encode
↪  the rest of the categorcial values."
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "Don't scale to compare the values."
  ],
  "metadata": {
    "id": "XWPI6GLEx4Oq"
  }
  }
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "d72c2e03"
  },
  "outputs": [],
  "source": [
    "simple_ct1 = ColumnTransformer([\n",
    "    ('impute_bmi', SimpleImputer(strategy='mean',
↪ missing_values=np.nan),['bmi']),\n",
    "    ('impute_smoke',SimpleImputer(strategy='
↪ most_frequent',missing_values=np.NaN),['smoking_status
↪ ']),\n",
    "    ],remainder='passthrough',
↪ verbose_feature_names_out=False,n_jobs=-1)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "78c1e22d"
  },
  "outputs": [],
  "source": [
    "simple_ct2 = ColumnTransformer([\n",
    "    ('ord', OrdinalEncoder(handle_unknown='
↪ use_encoded_value', unknown_value=np.nan),['
↪ smoking_status']),\n",
    "    ('ohe', OneHotEncoder(handle_unknown='ignore',
↪ sparse_output=False,drop=\"if_binary\"),ohe_feats),\n
↪ ",
    "    ('scale', RobustScaler(), num_data_col)\n",
    "    ],verbose_feature_names_out=False,remainder='
↪ passthrough',n_jobs=-1)"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "KOvKoq6eFwhg"
  },
  "source": [
    "If Before Imputation:\n",
    "\n",
    "OrdinalEncoder(handle_unknown='use_encoded_value',
↪ unknown_value=np.nan)\n",
    "\n",
    "OneHotEncoder(handle_unknown='ignore')"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "2rOCyLlt3Bcp"
  },
  "source": [
```

```
    "#### Data Visualization"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "9ebbbd42"
   },
   "source": [
    "> run only for data visualization process of the
   ↪ prepared data. only applicable on column transformers,
   ↪  not with imbalance handling because of fit_transform
   ↪ method."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "849685c2"
   },
   "outputs": [],
   "source": [
    "ct_simple = imbpipeline(steps = [('prep1', simple_ct1)
   ↪ , ('prep2',simple_ct2)])"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "54b72eab",
    "scrolled": false
   },
   "outputs": [],
   "source": [
    "set_config(transform_output=\"pandas\",display='
   ↪ diagram')\n",
    "display(ct_simple)"
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "from sklearn.utils import estimator_html_repr"
   ],
   "metadata": {
    "id": "OdN0Rnpk6P8s"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "# with open(\"ct_simple_diagram.html\", \"w\") as f:\n
   ↪ ",
    "#     f.write(estimator_html_repr(ct_simple))"
   ],
   "metadata": {
    "id": "DI3GKyUX6IqD"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "b7a321f3",
    "scrolled": false
   },
   "outputs": [],
   "source": [
    "simple_imputed = X_train.copy()\n",
    "simple_imputed.head()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "mq-XTyDHyhC4"
   },
   "outputs": [],
   "source": [
    "print(\"Shape of Data:\",simple_imputed.shape)\n",
    "print(\"Sum of Missing Values:\")\n",
    "print(simple_imputed.isnull().sum())"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "K1lCM7KL5DhA"
   },
   "source": [
    "Transform the data"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "1c300f44"
   },
   "outputs": [],
   "source": [
    "simple_imputed = ct_simple.fit_transform(
   ↪ simple_imputed)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "3dfcd88d"
   },
   "outputs": [],
   "source": [
```

```
    "print(\"Old Columns(\",len(X_train.columns),\") : \",
   ↪ list(X_train.columns))\n",
    "print(\"New Columns(\",len(simple_imputed.columns),\")
   ↪  : \", list(simple_imputed.columns))"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ba902377",
    "scrolled": true
  },
  "outputs": [],
  "source": [
    "simple_imputed.head()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "UudQI3Ztz2C9"
  },
  "outputs": [],
  "source": [
    "X_train.head()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "40a18314"
  },
  "outputs": [],
  "source": [
    "print(\"Shape of Data:\",simple_imputed.shape)\n",
    "print(\"Sum of Missing Values:\")\n",
    "print(simple_imputed.isnull().sum())"
  ]
},
{
  "cell_type": "code",
  "source": [
    "temp = simple_imputed.copy()"
  ],
  "metadata": {
    "id": "1zqNekFzyjwg"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "simple_imputed = pd.concat([simple_imputed,y_train['
   ↪ stroke']], axis=1)"
  ],
  "metadata": {
    "id": "j13lvdRaxZVW"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "3VBEQiLzy4-g"
  },
  "source": [
    "#### Check the unique values"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "S7Ah4PF4zpNM"
  },
  "outputs": [],
  "source": [
    "simple_imputed.nunique()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "-cRG33syy4kl"
  },
  "outputs": [],
  "source": [
    "col_tf1 = simple_imputed.columns"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "vdeJ4Ei-y-FO"
  },
  "outputs": [],
  "source": [
    "for col in col_tf1:\n",
    "  print(col)\n",
    "  print(simple_imputed[col].unique())"
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "Skewness and Kurtosis"
  ],
  "metadata": {
    "id": "pO2vKPNgIgJh"
  }
},
{
  "cell_type": "code",
```

```
    "source": [
      "print(\"Skewness and Kurtosis\")\n",
      "print(\"Skewness (Age): %f\" %simple_imputed['age'].
    ↪ skew())\n",
      "print(\"Kurtosis (Age): %f\" %simple_imputed['age'].
    ↪ kurt())\n",
      "print('-----------------------')\n",
      "print(\"Skewness (Average Glucose Level): %f\" %
    ↪ simple_imputed['avg_glucose_level'].skew())\n",
      "print(\"Kurtosis (Average Glucose Level): %f\" %
    ↪ simple_imputed['avg_glucose_level'].kurt())\n",
      "print('-----------------------')\n",
      "print(\"Skewness (BMI): %f\" %simple_imputed['bmi'].
    ↪ skew())\n",
      "print(\"Kurtosis (BMI): %f\" %simple_imputed['bmi'].
    ↪ kurt())"
    ],
    "metadata": {
      "id": "TglilvrIIfid"
    },
    "execution_count": null,
    "outputs": []
  },
  {
    "cell_type": "markdown",
    "source": [
      "*Age is heavily symmetrical with few outliers. Average
    ↪  Glucose Level is asymmetrical with a heavy tail/more
    ↪ outliers. BMI is also asymmetrical and has more
    ↪ observed outliers like Average Glucose Level.*"
    ],
    "metadata": {
      "id": "lkLtgayJJusY"
    }
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "2LcGbych9OVS"
    },
    "source": [
      "#### Check for Multicollinearity"
    ]
  },
  {
    "cell_type": "markdown",
    "metadata": {
      "id": "tZBRbVlxQQ4E"
    },
    "source": [
      "Need to be observed for regression models and also to
    ↪ avoid the dummy trap."
    ]
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "A78KGQPI9OVT"
    },
    "outputs": [],
    "source": [
      "from statsmodels.stats.outliers_influence import
    ↪ variance_inflation_factor"
    ]
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "En34DInL9OVT"
    },
    "outputs": [],
    "source": [
      "vif_simple = pd.DataFrame()\n",
      "vif_simple[\"feature\"] = simple_imputed.columns"
    ]
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "EMWqqDk59OVU"
    },
    "outputs": [],
    "source": [
      "# calculating VIF for each feature\n",
      "\n",
      "vif_simple[\"VIF\"] = [variance_inflation_factor(
    ↪ simple_imputed.values, i)\n",
      "                     for i in range(len(
    ↪ simple_imputed.columns))]\n",
      "print(vif_simple)"
    ]
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "jNVtzTQpCJ9Y"
    },
    "outputs": [],
    "source": [
      "vif_simple = pd.DataFrame(vif_simple)"
    ]
  },
  {
    "cell_type": "code",
    "execution_count": null,
    "metadata": {
      "id": "DWh0sC3oCbx0"
    },
    "outputs": [],
    "source": [
      "# vif_simple.to_csv('vif_simple.csv', index=False)"
    ]
  },
  {
    "cell_type": "markdown",
    "metadata": {
```

```
    "id": "G5_76LJV-F9k"
   },
   "source": [
    "Only the feature Private has high vif."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "1aTH9OQq83tX"
   },
   "source": [
    "#### Graphs"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "V6qF_bM_cjqN"
   },
   "source": [
    "##### Correlation Matrix (Heatmap)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "ZGWzQ5V6818Y"
   },
   "outputs": [],
   "source": [
    "corr_simple = simple_imputed.corr()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "tDLIYkET8_FG"
   },
   "outputs": [],
   "source": [
    "plt.subplots(figsize=(20, 20))\n",
    "sns.heatmap(corr_simple, xticklabels = corr_simple.
    ↪ columns, yticklabels = corr_simple.columns, annot =
    ↪ True)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "fp9lvNmJQ7M9"
   },
   "source": [
    "*At first glance, it can be observed that features
    ↪ Self_Employed and Govt_job has a high negative
    ↪ correlation with feature Private.This supports the
    ↪ values of VIF illustrated earlier wherein the 3
    ↪ mentioned features has the highest values of VIF among
    ↪  the features.*\n",
    "\n",
    "1. *Private (~7.65)*\n",
    "2. *Self_Employed (~3.14)*\n",
    "3. *Govt_job (~2.67)*\n",
    "\n",
    "*Other features correlated with Private does not seem
    ↪ to show any correlation at all with the exception of
    ↪ Age (-0.19).*"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "MPK6Jo55Zv_h"
   },
   "source": [
    "*Interestingly, among features, feature Age has a lot
    ↪ of positive correlations.*\n",
    "\n",
    "1. *Age X Ever_Married (0.29)*\n",
    "2. *Age X Self_Employed (0.26)*\n",
    "3. *Age X Heart_Disease (0.25)*\n",
    "4. *Age X Hypertension (0.22)*\n",
    "5. *Age X Glucose_Level (0.2)*\n",
    "\n",
    "*Other than feature Private, age also has another
    ↪ negative correlation with feature Smoking_Status
    ↪ (-0.16)*\n",
    "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "xn4vu6X3awme"
   },
   "source": [
    "*Additionally, here are other notable positive
    ↪ correlations, although not of high value.*\n",
    "\n",
    "1. *Heart_disease X Male (0.1)*\n",
    "2. *Heart_disease X Avg_Glucose_level (0.14)*\n",
    "3. *Heart_disease X hypertension (0.1)*\n",
    "\n",
    "4. *hypertension X Avg-_glucose_level (0.14)*\n",
    "5. *BMI X avg_glucose_level (0.16)*\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "gV0fQs78ZUUO"
   },
   "source": [
    "*Only features Urban and Never_Worked do not have any
    ↪ significant correlations with other features.*"
   ]
  },
  {
```

```
  "cell_type": "markdown",
  "metadata": {
    "id": "7uOhGiaOcw6c"
  },
  "source": [
    "##### Univariate Analysis"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "yEtQOGgYoc9F"
  },
  "source": [
    "**Summary Statistics**"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "uVtEM6X-ogIo"
  },
  "outputs": [],
  "source": [
    "simple_imputed.describe()"
  ]
},
{
  "cell_type": "code",
  "source": [
    "# simple_imputed.describe().to_csv(\"
  ↪ simpleimputed_summarystat.csv\")"
  ],
  "metadata": {
    "id": "KTyw3YYAIsOt"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "TaBHmvnYc1mv"
  },
  "source": [
    "**Target Values**"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "CeNZlnimczvz"
  },
  "outputs": [],
  "source": [
    "sns.countplot(x='stroke', data = y_train)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "94ldWJANoUhO"
  },
  "outputs": [],
  "source": [
    "from collections import Counter\n",
    "\n",
    "counter = Counter(y_train['stroke'])\n",
    "counter"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "AHjhUjIUdPFe"
  },
  "source": [
    "*Data is heavily Imbalanced.*\n"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "_G3edoZZf4te"
  },
  "source": [
    "**Numerical Distributions and Boxplots**"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "gWgOjdePmHjM"
  },
  "source": [
    "Before Transformation"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "hP6eNwU3l1SZ"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
  ↪ ()):\n",
    "    sns.histplot(x = X_train[variable], ax=subplot)\n
  ↪ ",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
```

```
  "cell_type": "markdown",
  "metadata": {
    "id": "t1UOTgDhmJv-"
  },
  "source": [
    "After Transformation"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "LNKi-WBbgVxP"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
↪ ()):\n",
    "    sns.histplot(x = simple_imputed[variable], ax=
↪ subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "IMj_ALePnDXk"
  },
  "source": [
    "Before Transformation"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "zeGa01O1m2j-"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
↪ ()):\n",
    "    sns.boxplot(x = X_train[variable], ax=subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "zbOKAo2InGv-"
  },
  "source": [
    "After Transformation"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "-tGx22Dimkf-"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
↪ ()):\n",
    "    sns.boxplot(x = simple_imputed[variable], ax=
↪ subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "*Age has no observed outliers. Meanwhile, average
↪ glucose level has more observed outliers than that of
↪ the feature BMI.*"
  ],
  "metadata": {
    "id": "5utc9y1HV_lF"
  }
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "skgGdSNqnf17"
  },
  "source": [
    "Categorcial Countplots"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Z3N1FGNYnkdH"
  },
  "outputs": [],
  "source": [
    "cat_data_col = simple_imputed.columns.difference(
↪ num_data_col)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "e6jHDFilniDW"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(5, 2, figsize=(20, 20))\n",
```

```
    "for variable, subplot in zip(cat_data_col, ax.flatten
  ↪ ()):\n",
    "    sns.countplot(x = simple_imputed[variable], ax=
  ↪ subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "Bf_VVGWtIJ42"
  },
  "source": [
    "*Among the countplots above, it can be observed that
  ↪ only the feature Residence_type_Urban has almost the
  ↪ same distribution for both data.*\n",
    "*Features ever_married_Yes and work_type_Private has
  ↪ more distribution of positive target class than that
  ↪ of the negative class.*\n",
    "*Among the 3 categories of feature smoking_status,
  ↪ category 1 has the most distribution.*\n",
    "*The rest of the features has more negative class than
  ↪  that of the positive class.*"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "5VjaCVj6smsR"
  },
  "source": [
    "##### Bivariate Analysis"
  ]
},
{
  "cell_type": "code",
  "source": [
    "num_data_col"
  ],
  "metadata": {
    "id": "S0m5jSvbzLoZ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "g = sns.PairGrid(simple_imputed, diag_sharey=False,
  ↪ hue=\"stroke\", vars=num_data_col)\n",
    "g.map_lower(sns.scatterplot)\n",
    "g.map_diag(sns.histplot)\n",
    "g.add_legend()"
  ],
  "metadata": {
    "id": "FWfmIlQ8NOZn"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "cat_data_col"
  ],
  "metadata": {
    "id": "K3cfYhusIIzS"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "##### Bivariate Categorical"
  ],
  "metadata": {
    "id": "BfXX8CJSLd1K"
  }
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='work_type_Self-employed',y='stroke',
  ↪ data=simple_imputed)"
  ],
  "metadata": {
    "id": "kNMlABc7LNEO"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='work_type_Private',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "r3BTOYwULFj0"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='work_type_Never_worked',y='stroke',data
  ↪ =simple_imputed)"
  ],
  "metadata": {
    "id": "2uXztp4rK_ec"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
```

```
  "source": [
    "sns.barplot(x='work_type_Govt_job',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "rdb-8Vm4K4VZ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='smoking_status',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "Nt6MW0daKvjl"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='hypertension',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "vcmpvTOnKncq"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='heart_disease',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "aLZ3qWQIKfCN"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='gender_Male',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "ZEPOGjNCKZe-"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='Residence_type_Urban',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "4BuHiSLtJdrI"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='ever_married_Yes',y='stroke',data=
  ↪ simple_imputed)"
  ],
  "metadata": {
    "id": "FlUu_i6nEe1g"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "simple_imputed.groupby('ever_married_Yes')['stroke'].
  ↪ mean().plot.bar()\n",
    "plt.show()"
  ],
  "metadata": {
    "id": "VIQ9CS5hD_oc"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "Qp9KAJPACeSz"
  },
  "source": [
    "##### Numerical Features VS Target Feature (Box Plots)
  ↪ "
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "_vtYgAtGNZf3"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(15, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
  ↪ ()):\n",
    "    sns.boxplot(x=y_train['stroke'], y=simple_imputed[
  ↪ variable], ax=subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
```

```
    "fig.tight_layout()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "QYvoSmu2Chi6"
   },
   "source": [
    "Categorical Features VS Target Feature (Comparative
   ↪ Bar Graphs)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "DAU7Uvo8uxew"
   },
   "outputs": [],
   "source": [
    "df_cat = simple_imputed[cat_data_col]"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "r7PsRwidvDPF"
   },
   "outputs": [],
   "source": [
    "cat_df = pd.concat([df_cat,y_train],axis=1)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "tDEFE8UNvM4K"
   },
   "outputs": [],
   "source": [
    "cat_df"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "FeQIChhMWLJF"
   },
   "outputs": [],
   "source": [
    "fig, ax = plt.subplots(3, 3, figsize=(15, 10))\n",
    "for var, subplot in zip(cat_data_col, ax.flatten()):\n
   ↪ ",
    "  distribution = pd.crosstab(cat_df[var], cat_df['
   ↪ stroke'], normalize = 'index')\n",
    "\n",
    "  sns.barplot(data = distribution.cumsum(axis=1).stack
   ↪ ().reset_index(name='Dist'),\n",
    "              x=var, y='Dist', hue ='stroke',\n",
    "              hue_order=distribution.columns[::-1],
   ↪ dodge=False, ax=subplot)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "BNwYXcrItLOU"
   },
   "outputs": [],
   "source": [
    "simple_imputed.columns"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "Oq63xF-fjXJz"
   },
   "source": [
    "#### Check the SMOTE"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "_A1GSxBLj9F2"
   },
   "outputs": [],
   "source": [
    "smote = SMOTE(random_state = 100)\n",
    "smotet= SMOTETomek(random_state=100)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "R6iXa5HxqtKM"
   },
   "outputs": [],
   "source": [
    "y_prep = y_train.copy()\n",
    "y_prep1 = y_train.copy()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "Zerz1b-3RiPe"
   },
   "outputs": [],
   "source": [
    "y_prep2 = y_train.copy()"
```

```
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "qqdJNyiGqwww"
  },
  "outputs": [],
  "source": [
    "y_prep"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "tp_Wo879Q3zk"
  },
  "outputs": [],
  "source": [
    "prep1_data1 = prep1_data.copy()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "31rnMbeDRsq9"
  },
  "outputs": [],
  "source": [
    "prep1_data2 = prep1_data.copy()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "262KybTvqpF_"
  },
  "outputs": [],
  "source": [
    "prep1_data, y_prep = smote.fit_resample(prep1_data,
  ↪ y_prep)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "zYcBGANSQ2nF"
  },
  "outputs": [],
  "source": [
    "prep1_data1, y_prep1 = smote_tomek.fit_resample(
  ↪ prep1_data1, y_prep1)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Ezt6Mt-KRw7y"
  },
  "outputs": [],
  "source": [
    "prep1_data2, y_prep2 = smotet.fit_resample(prep1_data2
  ↪ , y_prep2)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "WhcyiMncRBff"
  },
  "outputs": [],
  "source": [
    "print(prep1_data.shape)\n",
    "print(prep1_data1.shape)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "3LnWC-idR1_z"
  },
  "outputs": [],
  "source": [
    "print(prep1_data2.shape)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "13QZf9afR7LW"
  },
  "outputs": [],
  "source": [
    "print(y_prep2.shape)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "-ZM95ornRFWe"
  },
  "outputs": [],
  "source": [
    "print(y_prep.shape)\n",
    "print(y_prep1.shape)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
```

```
  "metadata": {
    "id": "6JOegs89R-f-"
  },
  "outputs": [],
  "source": [
    "print(y_prep['stroke'].value_counts())\n",
    "print(\"-----\")\n",
    "print(y_prep1['stroke'].value_counts())\n",
    "print(\"-----\")\n",
    "print(y_prep2['stroke'].value_counts())"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "fNcuxICPrMmm"
  },
  "outputs": [],
  "source": [
    "y_train['stroke'].unique()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "AeWrkx75rVLg"
  },
  "outputs": [],
  "source": [
    "sns.countplot(x='stroke', data = y_train)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8zTq2YLDrH7U"
  },
  "outputs": [],
  "source": [
    "sns.countplot(x='stroke', data = y_prep)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "BCqn8BlERPzF"
  },
  "outputs": [],
  "source": [
    "sns.countplot(x='stroke', data = y_prep1)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "Q4VYOFzXS3xk"
  },
  "outputs": [],
  "source": [
    "sns.countplot(x='stroke', data = y_prep2)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "a8YIer0jrv0A"
  },
  "outputs": [],
  "source": [
    "print(y_train.shape)\n",
    "print(y_prep.shape)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "dnQKs2N-r1Lo"
  },
  "outputs": [],
  "source": [
    "print(X_train.shape)\n",
    "print(prep1_data.shape)"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "9d55ff1f"
  },
  "source": [
    "###<b>2nd Imputation Method</b>\n",
    "> Encode first convert text data into numerical data
    ↪ to make it suitable for knn imputation before scaling
    ↪ since it will affect the imputation then impute the
    ↪ data before finally rounding the values of smoking
    ↪ status."
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "b-STeKBt8IJZ"
  },
  "outputs": [],
  "source": [
    "from sklearn.impute import KNNImputer"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "ahScsOnX8PpS"
```

```
  },
  "source": [
    "first convert text data into numerical data to make it
  ↪  suitable for knn imputaion"
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "remove scale for data visualization"
  ],
  "metadata": {
    "id": "J194Zi4o2aZJ"
  }
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "q93etr6FwLV_"
  },
  "outputs": [],
  "source": [
    "from sklearn.preprocessing import FunctionTransformer"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "u9bP1H_78XPd"
  },
  "outputs": [],
  "source": [
    "round_values = FunctionTransformer(np.round)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "OOkHhOqQ8ZK4"
  },
  "outputs": [],
  "source": [
    "knn_ct1 = ColumnTransformer([\n",
    "    ('ohe', OneHotEncoder(sparse_output=False,
  ↪ handle_unknown='ignore',drop=\"if_binary\"),ohe_feats)
  ↪ ,\n",
    "    ('ord', OrdinalEncoder(handle_unknown='
  ↪ use_encoded_value', unknown_value=np.nan),['
  ↪ smoking_status']),\n",
    "    ('scale', RobustScaler(), num_data_col),\n",
    "    ],remainder='passthrough',
  ↪ verbose_feature_names_out=False,n_jobs=-1)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "MfBcL6vku-gZ"
  },
  "outputs": [],
  "source": [
    "knn_ct2 = ColumnTransformer([\n",
    "    ('impute_both', KNNImputer(),['bmi','
  ↪ smoking_status']),\n",
    "    ],remainder='passthrough',
  ↪ verbose_feature_names_out=False,n_jobs=-1)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "E2nFUO2WzOsk"
  },
  "outputs": [],
  "source": [
    "knn_ct3 = ColumnTransformer([\n",
    "    ('round', round_values, ['smoking_status'])\n",
    "    ],remainder='passthrough',
  ↪ verbose_feature_names_out=False,n_jobs=-1)"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "CetJIcKH9AZv"
  },
  "source": [
    "#### Data Visualization"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "9BI_OYOJ9AZw"
  },
  "source": [
    "> run only for data visualization process of the
  ↪ prepared data. only applicable on column transformers,
  ↪  not with imbalance handling because of fit_transform
  ↪ method."
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "rITWQIkt8aXS"
  },
  "outputs": [],
  "source": [
    "ct_knn = imbpipeline(steps = [('prep1', knn_ct1),('
  ↪ prep2', knn_ct2),('prep3',knn_ct3)])"
  ]
},
{
```

```
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "kncHe7hA8e2F"
  },
  "outputs": [],
  "source": [
    "set_config(transform_output=\"pandas\",display='
  ↪ diagram')"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "_lMTvmFS8cJ0"
  },
  "outputs": [],
  "source": [
    "display(ct_knn)"
  ]
},
{
  "cell_type": "code",
  "source": [
    "# with open(\"ct_knn_diagram.html\", \"w\") as f:\n",
    "#     f.write(estimator_html_repr(ct_knn))"
  ],
  "metadata": {
    "id": "orLuDSxb9AZy"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "scrolled": false,
    "id": "E8uW27oV9AZy"
  },
  "outputs": [],
  "source": [
    "knn_imputed = X_train.copy()\n",
    "knn_imputed.head()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "my8MUZ009AZz"
  },
  "outputs": [],
  "source": [
    "print(\"Shape of Data:\",knn_imputed.shape)\n",
    "print(\"Sum of Missing Values:\")\n",
    "print(knn_imputed.isnull().sum())"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "KFGCyeKL9AZz"
  },
  "source": [
    "Transform the data"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8d6SBD4C9AZz"
  },
  "outputs": [],
  "source": [
    "knn_imputed = ct_knn.fit_transform(knn_imputed)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "UBXuUvMB9AZ0"
  },
  "outputs": [],
  "source": [
    "print(\"Old Columns(\",len(X_train.columns),\") : \",
  ↪ list(X_train.columns))\n",
    "print(\"New Columns(\",len(knn_imputed.columns),\") :
  ↪ \", list(knn_imputed.columns))"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "scrolled": true,
    "id": "foOx1cBr9AZ0"
  },
  "outputs": [],
  "source": [
    "knn_imputed.head()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "a_i2aSD19AZ1"
  },
  "outputs": [],
  "source": [
    "X_train.head()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
```

```
  "metadata": {
    "id": "9pYWFoAv9AZ1"
  },
  "outputs": [],
  "source": [
    "print(\"Shape of Data:\",knn_imputed.shape)\n",
    "print(\"Sum of Missing Values:\")\n",
    "print(knn_imputed.isnull().sum())"
  ]
},
{
  "cell_type": "code",
  "source": [
    "temp_knn = knn_imputed.copy()"
  ],
  "metadata": {
    "id": "ij9S240u2pYz"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "knn_imputed = pd.concat([knn_imputed,y_train['stroke
  ↪ ']], axis=1)"
  ],
  "metadata": {
    "id": "pLAVb1bJ2v7U"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "knn_imputed"
  ],
  "metadata": {
    "id": "3l7jNXgx2y1K"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "ymI4vr0wHFR0"
  },
  "source": [
    "#### Check the unique values"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "9ciMXzzYHFSD"
  },
  "outputs": [],
  "source": [
    "knn_imputed.nunique()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "YfV_9SWUHFSE"
  },
  "outputs": [],
  "source": [
    "col_tf2 = knn_imputed.columns"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "A61FRQAJHFSE"
  },
  "outputs": [],
  "source": [
    "for col in col_tf2:\n",
    "  print(col)\n",
    "  print(knn_imputed[col].unique())"
  ]
},
{
  "cell_type": "code",
  "source": [
    "print(\"Skewness and Kurtosis\")\n",
    "print(\"Skewness (Age): %f\" %knn_imputed['age'].skew
  ↪ ())\n",
    "print(\"Kurtosis (Age): %f\" %knn_imputed['age'].kurt
  ↪ ())\n",
    "print('-----------------------')\n",
    "print(\"Skewness (Average Glucose Level): %f\" %
  ↪ knn_imputed['avg_glucose_level'].skew())\n",
    "print(\"Kurtosis (Average Glucose Level): %f\" %
  ↪ knn_imputed['avg_glucose_level'].kurt())\n",
    "print('-----------------------')\n",
    "print(\"Skewness (BMI): %f\" %knn_imputed['bmi'].skew
  ↪ ())\n",
    "print(\"Kurtosis (BMI): %f\" %knn_imputed['bmi'].kurt
  ↪ ())"
  ],
  "metadata": {
    "id": "vJr8uiNZ_BqR"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "X3raYjkHHWP9"
  },
  "source": [
    "#### Check for Multicollinearity"
```

```
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "iJWZReumHWQQ"
  },
  "source": [
    "Need to be observed for regression models"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "tfH4-o5PHWQS"
  },
  "outputs": [],
  "source": [
    "vif_knn = pd.DataFrame()\n",
    "vif_knn[\"feature\"] = knn_imputed.columns"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "2yd6UlNTHWQT"
  },
  "outputs": [],
  "source": [
    "# calculating VIF for each feature\n",
    "\n",
    "vif_knn[\"VIF\"] = [variance_inflation_factor(
↪ knn_imputed.values, i)\n",
    "                          for i in range(len(
↪ knn_imputed.columns))]\n",
    "print(vif_knn)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "OAELqb3QHWQU"
  },
  "outputs": [],
  "source": [
    "vif_knn = pd.DataFrame(vif_knn)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "u2UcBGOQHWQU"
  },
  "outputs": [],
  "source": [
    "# vif_knn.to_csv('vif_knn.csv', index=False)"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "cvEdDOBTHWQV"
  },
  "source": [
    "Only the feature Private has high vif."
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "ucaplPVmHvrD"
  },
  "source": [
    "#### Graphs"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "x-8BvKc-HvrM"
  },
  "source": [
    "##### Correlation Matrix (Heatmap)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "8wgZizr-HvrN"
  },
  "outputs": [],
  "source": [
    "corr_knn = knn_imputed.corr()"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "_3Uo2pKcHvrN"
  },
  "outputs": [],
  "source": [
    "plt.subplots(figsize=(20, 20))\n",
    "sns.heatmap(corr_knn, xticklabels = corr_knn.columns,
↪ yticklabels = corr_knn.columns, annot = True)"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "PMCCZxHPHvrN"
  },
  "source": [
    "*At first glance, it can be observed that features
```

```
    ↪ Self_Employed and Govt_job has a high negative
    ↪ correlation with feature Private.This supports the
    ↪ values of VIF illustrated earlier wherein the 3
    ↪ mentioned features has the highest values of VIF among
    ↪  the features.*\n",
     "\n",
     "1. *Private (~7.54)*\n",
     "2. *Self_Employed (~3.10)*\n",
     "3. *Govt_job (~2.64)*\n",
     "\n",
     "*Other features correlated with Private does not seem
    ↪ to show any correlation at all with the exception of
    ↪ Age (-0.19).*"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
     "id": "H7Fkiad7HvrO"
   },
   "source": [
     "*Interestingly, among features, feature Age has a lot
    ↪ of positive correlations.*\n",
     "\n",
     "1. *Age X Ever_Married (0.29)*\n",
     "2. *Age X Self_Employed (0.26)*\n",
     "3. *Age X Heart_Disease (0.25)*\n",
     "4. *Age X Hypertension (0.22)*\n",
     "5. *Age X Glucose_Level (0.2)*\n",
     "\n",
     "*Other than feature Private, age also has another
    ↪ negative correlation with feature Smoking_Status
    ↪ (-0.15)*\n",
     "\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
     "id": "5IVHONEwHvrO"
   },
   "source": [
     "*Additionally, here are other notable positive
    ↪ correlations, although not of high value.*\n",
     "\n",
     "1. *Heart_disease X Male (0.1)*\n",
     "2. *Heart_disease X Avg_Glucose_level (0.14)*\n",
     "3. *Heart_disease X hypertension (0.1)*\n",
     "\n",
     "4. *hypertension X Avg-_glucose_level (0.14)*\n",
     "5. *BMI X avg_glucose_level (0.16)*\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
     "id": "XafJLTO5HvrO"
   },
   "source": [
     "*Only features Urban and Never_Worked do not have any
    ↪ significant correlations with other features.*"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
     "id": "NA-BeLzDHvrO"
   },
   "source": [
     "##### Univariate Analysis"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
     "id": "jgrYG1qPHvrP"
   },
   "source": [
     "**Summary Statistics**"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
     "id": "1R1JMYL1HvrP"
   },
   "outputs": [],
   "source": [
     "knn_imputed.describe()"
   ]
  },
  {
   "cell_type": "code",
   "source": [
     "# knn_imputed.describe().to_csv(\"
    ↪ knnimputed_summarystat.csv\")"
   ],
   "metadata": {
     "id": "8iOhVyirI4ew"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "metadata": {
     "id": "5BW24Y3PHvrP"
   },
   "source": [
     "**Target Values**"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
     "id": "Fy9GFwQJHvrP"
   },
   "outputs": [],
   "source": [
```

```
  "source": [
    "sns.countplot(x='stroke', data = y_train)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "2JI7kQqIHvrQ"
  },
  "outputs": [],
  "source": [
    "counter = Counter(y_train['stroke'])\n",
    "counter"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "vwiOinOVHvrQ"
  },
  "source": [
    "*Data is heavily Imbalanced.*\n"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "l9_COzKlHvrQ"
  },
  "source": [
    "**Numerical Distributions and Boxplots**"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "uQlJhjp-HvrQ"
  },
  "source": [
    "Before Transformation"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "ApPQXLQHHvrQ"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
    ↪ ()):\n",
    "    sns.histplot(x = X_train[variable], ax=subplot)\n
    ↪ ",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "2Cuq3HDeHvrR"
  },
  "source": [
    "After Transformation"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "xHNXzjSRHvrR"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
    ↪ ()):\n",
    "    sns.histplot(x = knn_imputed[variable], ax=subplot
    ↪ )\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "t6ZjoVOQHvrR"
  },
  "source": [
    "Before Transformation"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "QfW3hfhOHvrR"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
    ↪ ()):\n",
    "    sns.boxplot(x = X_train[variable], ax=subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "QlnwUY_qHvrR"
  },
  "source": [
```

```
    "After Transformation"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "w7PTv_j8HvrR"
   },
   "outputs": [],
   "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(20, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
    ↪ ()):\n",
    "    sns.boxplot(x = knn_imputed[variable], ax=subplot)
    ↪ \n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "*Age has no observed outliers. Meanwhile, average
    ↪ glucose level has more observed outliers than that of
    ↪ the feature BMI.*"
   ],
   "metadata": {
    "id": "wC2Wem65HvrR"
   }
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "1SAS_fNPHvrS"
   },
   "source": [
    "Categorcial Countplots"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "UCLKv13DHvrS"
   },
   "outputs": [],
   "source": [
    "cat_data_col = knn_imputed.columns.difference(
    ↪ num_data_col)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "id": "3nznujZJHvrS"
   },
   "outputs": [],
   "source": [
    "fig, ax = plt.subplots(5, 2, figsize=(20, 20))\n",
    "for variable, subplot in zip(cat_data_col, ax.flatten
    ↪ ()):\n",
    "    sns.countplot(x = knn_imputed[variable], ax=
    ↪ subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
   ]
  },
  {
   "cell_type": "markdown",
   "source": [
    "Need to modify"
   ],
   "metadata": {
    "id": "cQUawRmUJpG4"
   }
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "JzqlhTiKHvrS"
   },
   "source": [
    "*Among the countplots above, it can be observed that
    ↪ only the feature Residence_type_Urban has almost the
    ↪ same distribution for both data.*\n",
    "*Features ever_married_Yes and work_type_Private has
    ↪ more distribution of positive target class than that
    ↪ of the negative class.*\n",
    "*Among the 3 categories of feature smoking_status,
    ↪ category 1 has the most distribution.*\n",
    "*The rest of the features has more negative class than
    ↪  that of the positive class.*"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "id": "5fXNBpMdHvrS"
   },
   "source": [
    "##### Bivariate Analysis"
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "g = sns.PairGrid(knn_imputed, diag_sharey=False, hue
    ↪ =\"stroke\", vars=num_data_col)\n",
    "g.map_lower(sns.scatterplot)\n",
    "g.map_diag(sns.histplot)\n",
    "g.add_legend()"
   ],
   "metadata": {
    "id": "514LeW5u3BwV"
   },
   "execution_count": null,
```

```
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "##### Bivariate Categorical"
  ],
  "metadata": {
    "id": "2yQ7QaUjLzu1"
  }
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='work_type_Self-employed',y='stroke',
    ↪ data=knn_imputed)"
  ],
  "metadata": {
    "id": "8coindBkLzu2"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='work_type_Private',y='stroke',data=
    ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "A83svMwtLzu2"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='work_type_Never_worked',y='stroke',data
    ↪ =knn_imputed)"
  ],
  "metadata": {
    "id": "h1MRayKhLzu3"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='work_type_Govt_job',y='stroke',data=
    ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "5zOWiFkOLzu4"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='smoking_status',y='stroke',data=
    ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "-tWVy17GLzu4"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='hypertension',y='stroke',data=
    ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "fDP4NHYgLzu5"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='heart_disease',y='stroke',data=
    ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "ulcrUJJOLzu5"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='gender_Male',y='stroke',data=
    ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "_RFZMKb_Lzu6"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "sns.barplot(x='Residence_type_Urban',y='stroke',data=
    ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "HtsvZ5O7Lzu6"
  },
  "execution_count": null,
  "outputs": []
},
{
```

```
  "cell_type": "code",
  "source": [
    "sns.barplot(x='ever_married_Yes',y='stroke',data=
  ↪ knn_imputed)"
  ],
  "metadata": {
    "id": "FBUXGBFBLzu7"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "TD3nOBp5HvrS"
  },
  "source": [
    "##### |Numerical Features VS Target Feature (Box Plots
  ↪ )"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "LL-Nw65mHvrS"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 1, figsize=(15, 20))\n",
    "for variable, subplot in zip(num_data_col, ax.flatten
  ↪ ()):\n",
    "    sns.boxplot(x=y_train['stroke'], y=knn_imputed[
  ↪ variable], ax=subplot)\n",
    "    for label in subplot.get_xticklabels():\n",
    "        label.set_rotation(90)\n",
    "fig.tight_layout()"
  ]
},
{
  "cell_type": "markdown",
  "metadata": {
    "id": "YJLkfPNcHvrS"
  },
  "source": [
    "Categorical Features VS Target Feature (Comparative
  ↪ Bar Graphs)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "sjjcMxLkHvrT"
  },
  "outputs": [],
  "source": [
    "df_cat = knn_imputed[cat_data_col]"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "l3UN_-5EHvrT"
  },
  "outputs": [],
  "source": [
    "cat_df = pd.concat([df_cat,y_train],axis=1)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "pT4l1uTeHvrT"
  },
  "outputs": [],
  "source": [
    "cat_df"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "SOfnthuIHvrT"
  },
  "outputs": [],
  "source": [
    "fig, ax = plt.subplots(3, 3, figsize=(15, 10))\n",
    "for var, subplot in zip(cat_data_col, ax.flatten()):\n
  ↪ ",
    "  distribution = pd.crosstab(cat_df[var], cat_df['
  ↪ stroke'], normalize = 'index')\n",
    "\n",
    "  sns.barplot(data = distribution.cumsum(axis=1).stack
  ↪ ().reset_index(name='Dist'),\n",
    "              x=var, y='Dist', hue ='stroke',\n",
    "              hue_order=distribution.columns[::-1],
  ↪ dodge=False, ax=subplot)"
  ]
},
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {
    "id": "p7wvPbuGHvrT"
  },
  "outputs": [],
  "source": [
    "knn_imputed.columns"
  ]
},
{
  "cell_type": "markdown",
  "source": [
    "# Model Evaluation"
  ],
  "metadata": {
```

```
    "id": "VKdNuqOfOC3g"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "from sklearn.feature_selection import SelectFromModel\
   ↪ n",
    "from sklearn.ensemble import ExtraTreesClassifier\n",
    "from imblearn.over_sampling import SMOTE\n",
    "from imblearn.combine import SMOTETomek"
   ],
   "metadata": {
    "id": "YRy5OM_uOGSg"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "from sklearn.metrics import confusion_matrix,
   ↪ classification_report,roc_curve\n",
    "from sklearn.metrics import accuracy_score, f1_score,
   ↪ precision_score,recall_score,roc_auc_score,
   ↪ make_scorer"
   ],
   "metadata": {
    "id": "qYnskO4uOWik"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "import joblib\n",
    "import ast"
   ],
   "metadata": {
    "id": "3MfJCV5fOQWz"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "from sklearn.linear_model import LogisticRegression\n
   ↪ ",
    "from sklearn.ensemble import RandomForestClassifier\n
   ↪ ",
    "from sklearn.svm import SVC\n",
    "import xgboost as xgb\n",
    "from sklearn.ensemble import AdaBoostClassifier\n",
    "from sklearn.neighbors import KNeighborsClassifier\n",
    "from sklearn.neural_network import MLPClassifier"
   ],
   "metadata": {
    "id": "gs0k4M7iQZPl"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "def evaluate_model(model, x_test, y_test):\n",
    "  y_pred = model.predict(x_test)\n",
    "\n",
    "  print(\"-------------\")\n",
    "  print('Accuracy: %.3f' % accuracy_score(y_test,
   ↪ y_pred))\n",
    "  print('Precision: %.3f' % precision_score(y_test,
   ↪ y_pred))\n",
    "  print('F1 Score: %.3f' % f1_score(y_test, y_pred))\n
   ↪ ",
    "  print('Recall: %.3f' % recall_score(y_test, y_pred))
   ↪ \n",
    "  print('Specificity: %.3f' % recall_score(y_test,
   ↪ y_pred, pos_label =0))\n",
    "  print('ROC AUC: %.3f' % roc_auc_score(y_test, y_pred
   ↪ ))\n",
    "  print(\"-------------\")\n",
    "  fpr, tpr, _ = roc_curve(y_test,  y_pred)\n",
    "  auc = roc_auc_score(y_test, y_pred)\n",
    "  plt.plot(fpr,tpr,label=\"AUC=\"+str(auc))\n",
    "  plt.ylabel('True Positive Rate')\n",
    "  plt.xlabel('False Positive Rate')\n",
    "  plt.legend(loc=4)\n",
    "  plt.show()\n",
    "  print(\"-------------\")\n",
    "  print(classification_report(y_pred,y_test))\n",
    "\n",
    "  print(\"-------------\")\n",
    "  conf = confusion_matrix(y_test, y_pred)\n",
    "  sns.heatmap(conf,annot = True, fmt='g',cmap ='Blues
   ↪ ')"
   ],
   "metadata": {
    "id": "atH6Pbc5OdfO"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "smote = SMOTE(random_state = 100)\n",
    "smote_tomek = SMOTETomek(random_state=100)"
   ],
   "metadata": {
    "id": "isp1n3inO1vp"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
```

```
    "#only applies on pipelines withe feature selection\n",
    "def get_features(pipe,columns):\n",
    "  pipe_mask = pipe.named_steps['fs'].get_support()\n",
    "  new_features = []\n",
    "\n",
    "  for bool, feature in zip(pipe_mask, columns):\n",
    "    if bool:\n",
    "        new_features.append(feature)\n",
    "  print(\"Threshold:\",pipe.named_steps['fs'].
↪ threshold_)\n",
    "  return new_features"
  ],
  "metadata": {
    "id": "87fmTngtOBOy"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "def extract_features(pipe):\n",
    "  # data = list(zip(model.feature_names_in_, model.
↪ feature_importances_))\n",
    "  # df_importances = pd.DataFrame(data, columns=['
↪ Feature', 'Importance']).sort_values(by='Importance',
↪ ascending=False)\n",
    "  # print(df_importances)\n",
    "  # df_importances.plot.barh(x='Feature', y='
↪ Importance')\n",
    "  clf = pipe.named_steps['clf']\n",
    "  return clf.feature_names_in_"
  ],
  "metadata": {
    "id": "WUkTerlt8PHW"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "For complete pipelines only"
  ],
  "metadata": {
    "id": "wtmsyz5lPWG2"
  }
},
{
  "cell_type": "code",
  "source": [
    "def build_model_simple_imb(imb, feat, clf, x_train,
↪ y_train, **params):\n",
    "  print(params)\n",
    "  temp = params.copy()\n",
    "  for key in temp.keys():\n",
    "    print(key)\n",
    "    new_key = key.replace(\"clf__\",\"\")\n",
    "    params[new_key] = params.pop(key)\n",
    "\n",
    "  print(\"Params:\",params)\n",
    "  clf.set_params(**params)\n",
    "\n",
    "  if feat == True:\n",
    "       xtra_trees = ExtraTreesClassifier(random_state
↪  = 100)\n",
    "       feat_selection = SelectFromModel(xtra_trees)\n
↪ ",
    "\n",
    "       pipe_ml = imbpipeline(steps = [['prep1',
↪ simple_ct1],['prep2', simple_ct2],\n",
    "                                      ['imb', imb],['
↪ fs', feat_selection],['clf', clf]])\n",
    "  else:\n",
    "       pipe_ml = imbpipeline(steps = [['prep1',
↪ simple_ct1],['prep2', simple_ct2],\n",
    "                                      ['imb', imb],['
↪ clf', clf]])\n",
    "  pipe_ml.set_output(transform=\"pandas\")\n",
    "\n",
    "  display(pipe_ml)\n",
    "\n",
    "  pipe_ml.fit(x_train, y_train.values.ravel())\n",
    "  return pipe_ml\n",
    "  # joblib.dump(pipe_ml, 'model_simple.joblib')\n",
    "\n",
    "  # print(\"Model Saved!\")"
  ],
  "metadata": {
    "id": "n1hv0wFVO8fK"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "def build_model_knn_imb(imb, feat, clf, x_train,
↪ y_train, **params):\n",
    "  print(params)\n",
    "  temp = params.copy()\n",
    "  for key in temp.keys():\n",
    "    print(key)\n",
    "    new_key = key.replace(\"clf__\",\"\")\n",
    "    params[new_key] = params.pop(key)\n",
    "\n",
    "  print(\"Params:\",params)\n",
    "  clf.set_params(**params)\n",
    "\n",
    "  if feat == True:\n",
    "       xtra_trees = ExtraTreesClassifier(random_state
↪  = 100)\n",
    "       feat_selection = SelectFromModel(xtra_trees)\n
↪ ",
    "\n",
    "       pipe_ml = imbpipeline(steps = [['prep1',
↪ knn_ct1], ['prep2',knn_ct2],['prep3',knn_ct3],\n",
    "                                      ['imb', imb],['
↪ fs', feat_selection],['clf', clf]])\n",
    "  else:\n",
```

```
    "        pipe_ml = imbpipeline(steps = [['prep1',
↪ knn_ct1], ['prep2',knn_ct2],['prep3',knn_ct3],\n",
    "                                       ['imb', imb],['
↪ clf', clf]])\n",
    "  pipe_ml.set_output(transform=\"pandas\")\n",
    "\n",
    "  display(pipe_ml)\n",
    "\n",
    "  pipe_ml.fit(x_train, y_train.values.ravel())\n",
    "  return pipe_ml\n",
    "  # joblib.dump(pipe_ml, 'model_knn.joblib')\n",
    "\n",
    "  # print(\"Model Saved!\")"
  ],
  "metadata": {
    "id": "rgFPfO0GPPfv"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "## No Imbalance handling"
  ],
  "metadata": {
    "id": "4dO6w8AsO3DR"
  }
},
{
  "cell_type": "markdown",
  "source": [
    "(default hyperparameter values)"
  ],
  "metadata": {
    "id": "9sjsumQqP7R2"
  }
},
{
  "cell_type": "code",
  "source": [
    "def build_model_simple(feat, clf, x_train, y_train):\n
↪ ",
    "\n",
    "  if feat == True:\n",
    "        xtra_trees = ExtraTreesClassifier(random_state
↪  = 100)\n",
    "        feat_selection = SelectFromModel(xtra_trees)\n
↪ ",
    "\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
↪ simple_ct1],['prep2', simple_ct2],\n",
    "                                       ['fs',
↪ feat_selection],['clf', clf]])\n",
    "  else:\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
↪ simple_ct1],['prep2', simple_ct2],\n",
    "                                       ['clf', clf]])
↪ \n",
    "  pipe_ml.set_output(transform=\"pandas\")\n",
    "\n",
    "  display(pipe_ml)\n",
    "\n",
    "  pipe_ml.fit(x_train, y_train.values.ravel())\n",
    "  return pipe_ml"
  ],
  "metadata": {
    "id": "C119fwV2P3LN"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "def build_model_knn(feat, clf, x_train, y_train):\n",
    "\n",
    "  if feat == True:\n",
    "        xtra_trees = ExtraTreesClassifier(random_state
↪  = 100)\n",
    "        feat_selection = SelectFromModel(xtra_trees)\n
↪ ",
    "\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
↪ knn_ct1], ['prep2',knn_ct2],['prep3',knn_ct3],\n",
    "                                       ['fs',
↪ feat_selection],['clf', clf]])\n",
    "  else:\n",
    "        pipe_ml = imbpipeline(steps = [['prep1',
↪ knn_ct1], ['prep2',knn_ct2],['prep3',knn_ct3],\n",
    "                                       ['clf', clf]])
↪ \n",
    "  pipe_ml.set_output(transform=\"pandas\")\n",
    "\n",
    "  display(pipe_ml)\n",
    "\n",
    "  pipe_ml.fit(x_train, y_train.values.ravel())\n",
    "  return pipe_ml"
  ],
  "metadata": {
    "id": "BaUVRc-cPfwV"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "### No Feature Selection"
  ],
  "metadata": {
    "id": "8uVyRp7GQKRQ"
  }
},
{
  "cell_type": "code",
  "source": [
    "eval_data = X_train.copy()"
  ],
  "metadata": {
```

```
    "id": "7NYdM61GQk6P"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### Logistic Regression"
   ],
   "metadata": {
    "id": "vD3FP18SR92y"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_lr = build_model_simple(False,LogisticRegression(
   ↪ random_state=100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "P7HJfE7DQHzM"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
   ],
   "metadata": {
    "id": "9Zv7EnvNQs6C"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_lr = build_model_knn(False,LogisticRegression(
   ↪ random_state=100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "ln5oZd4_RiuY"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
   ],
   "metadata": {
    "id": "YGskPXGSRmPm"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### Random Forest"
   ],
   "metadata": {
    "id": "mF6owdMaSDNp"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_rf = build_model_simple(False,
   ↪ RandomForestClassifier(random_state=100),eval_data,
   ↪ y_train)"
   ],
   "metadata": {
    "id": "FCMynbvgSDN3"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
   ],
   "metadata": {
    "id": "gUPJx-G-SDN3"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_rf = build_model_knn(False,RandomForestClassifier
   ↪ (random_state=100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "ZvIfDZgxSDN4"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
   ],
   "metadata": {
    "id": "7Iga3E87SDN4"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### Support Vector Machine"
```

```
  ],
  "metadata": {
    "id": "4wW0Vs2DT1-c"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_simple(False,SVC(random_state
  ↪ =100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "K-iS-PaqT1-w"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "uae4Ih8mT1-w"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_knn(False,SVC(random_state=100)
  ↪ ,eval_data,y_train)"
  ],
  "metadata": {
    "id": "lYjdbujPT1-x"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "O9yEsfoRT1-x"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### MLP"
  ],
  "metadata": {
    "id": "vUHGrLcqUq7e"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_mlp = build_model_simple(False,MLPClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "iOJJyINcUq7w"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
  ],
  "metadata": {
    "id": "usJseLDeUq7x"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_mlp = build_model_knn(False,MLPClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "5vsP8p2cUq7y"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
  ],
  "metadata": {
    "id": "5PkIGvbnUq7y"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### XGBoost"
  ],
  "metadata": {
    "id": "K-VxYwFKVMIc"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_simple(False,xgb.XGBClassifier(
  ↪ random_state=100),eval_data,y_train)"
```

```
  ],
  "metadata": {
    "id": "HyD1apeBVMIs"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "mRMA7lzsVMIt"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_knn(False,xgb.XGBClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "6tx4gWZMVMIu"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "eO0d1q4eVMIv"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### AdaBoost"
  ],
  "metadata": {
    "id": "qW36kl-tWOse"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_simple(False,AdaBoostClassifier
  ↪ (random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "nUAm5SVcWOsn"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "kYkCNzFXWOsn"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_knn(False,AdaBoostClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "Z6ZV2vI_WOsn"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "KhUPaTFrWOsn"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### KNN"
  ],
  "metadata": {
    "id": "1kDv2KL9Wm-q"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_simple(False,
  ↪ KNeighborsClassifier(),eval_data,y_train)"
  ],
  "metadata": {
    "id": "kmhNkke9Wm-1"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
```

```
    "evaluate_model(pipe_knn,X_test,y_test)"
   ],
   "metadata": {
    "id": "It8gJNakWm-1"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_knn = build_model_knn(False,KNeighborsClassifier
   ↪ (),eval_data,y_train)"
   ],
   "metadata": {
    "id": "lldZBx7oWm-1"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
   ],
   "metadata": {
    "id": "Ou8G-lpJWm-1"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "### With Feature Selection"
   ],
   "metadata": {
    "id": "Q8W6QgliXAkb"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "eval_data = X_train.copy()"
   ],
   "metadata": {
    "id": "XiKGOjSuXAkh"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### Logistic Regression"
   ],
   "metadata": {
    "id": "bgbz_oggXAkh"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_lr = build_model_simple(True,LogisticRegression(
   ↪ random_state=100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "HDHpHA7UXAkh"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
   ],
   "metadata": {
    "id": "xP_AngR2XAkh"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_lr = build_model_knn(True,LogisticRegression(
   ↪ random_state=100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "LpPeMpvvXAkh"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
   ],
   "metadata": {
    "id": "ZgnKoJpZXAkh"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### Random Forest"
   ],
   "metadata": {
    "id": "yoaTV-M8XAkh"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_rf = build_model_simple(True,
   ↪ RandomForestClassifier(random_state=100),eval_data,
```

```
    ↪ y_train)"
   ],
   "metadata": {
    "id": "d9KnFHUgXAkh"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
   ],
   "metadata": {
    "id": "IpbH1CB4XAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_rf = build_model_knn(True,RandomForestClassifier(
    ↪ random_state=100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "TsF-z7ndXAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
   ],
   "metadata": {
    "id": "n6OF35qZXAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### Support Vector Machine"
   ],
   "metadata": {
    "id": "8dOVJ88SXAki"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_svm = build_model_simple(True,SVC(random_state
    ↪ =100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "-qp-TzozXAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
   ],
   "metadata": {
    "id": "bWjIaPysXAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_svm = build_model_knn(True,SVC(random_state=100),
    ↪ eval_data,y_train)"
   ],
   "metadata": {
    "id": "FJkrH7uAXAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
   ],
   "metadata": {
    "id": "5czp4EBWXAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### MLP"
   ],
   "metadata": {
    "id": "mdOToKFNXAki"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_mlp = build_model_simple(True,MLPClassifier(
    ↪ random_state=100),eval_data,y_train)"
   ],
   "metadata": {
    "id": "Dr0LFSiXXAki"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
```

```
  "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
  ],
  "metadata": {
    "id": "g5QSCGgBXAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_mlp = build_model_knn(True,MLPClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "wfrLVKViXAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
  ],
  "metadata": {
    "id": "ApwTQCCyXAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### XGBoost"
  ],
  "metadata": {
    "id": "VLV8kCyKXAkj"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_simple(True,xgb.XGBClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "3UL4sGj8XAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "WyuCh8Y-XAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_knn(True,xgb.XGBClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "h5POm3xlXAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "O_U-Xiv3XAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### AdaBoost"
  ],
  "metadata": {
    "id": "oMM2y7JvXAkj"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_simple(True,AdaBoostClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "AfXLrsYHXAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "Iir6h_-cXAkj"
  },
  "execution_count": null,
  "outputs": []
},
{
```

```
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_knn(True,AdaBoostClassifier(
  ↪ random_state=100),eval_data,y_train)"
  ],
  "metadata": {
    "id": "tVR1rbkwXAkk"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "rhqfTWz_XAkk"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### KNN"
  ],
  "metadata": {
    "id": "AAs4HLEqXAkk"
  }
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_simple(True,
  ↪ KNeighborsClassifier(),eval_data,y_train)"
  ],
  "metadata": {
    "id": "TOUNPuDZXAkk"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "EbwWZ50wXAkk"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_knn(True,KNeighborsClassifier()
  ↪ ,eval_data,y_train)"
  ],
  "metadata": {
    "id": "DTdgI7P1XAkk"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "Nf1FXryVXAkk"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "## With Imbalance Handling (SMOTE)"
  ],
  "metadata": {
    "id": "UuyA6dFcYOmC"
  }
},
{
  "cell_type": "markdown",
  "source": [
    "### No Feature Selection"
  ],
  "metadata": {
    "id": "tu4TFoIEZD4Q"
  }
},
{
  "cell_type": "code",
  "source": [
    "eval_data = X_train.copy()"
  ],
  "metadata": {
    "id": "v6eu_YEZZD4R"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Logistic Regression"
  ],
  "metadata": {
    "id": "-Us8bqbRZD4R"
  }
},
{
  "cell_type": "code",
  "source": [
    "lr_params_simple = {'clf__C': 0.001, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'sag'}"
```

```
  ],
  "metadata": {
    "id": "dx4apOvUnE0L"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_simple_imb(smote,False,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_simple)"
  ],
  "metadata": {
    "id": "_i3u_RKhZD4R"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "oppiRqw1ZD4R"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "lr_params_knn = {'clf__C': 0.001, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'sag'}"
  ],
  "metadata": {
    "id": "kESiIJEyngKX"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_knn_imb(smote,False,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_knn)"
  ],
  "metadata": {
    "id": "kAxAu99HZD4R"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "1j_P2LU-ZD4R"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Random Forest"
  ],
  "metadata": {
    "id": "SNoFI9C0ZD4R"
  }
},
{
  "cell_type": "code",
  "source": [
    "rf_params_simple ={'clf__max_features': 'sqrt', '
  ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 10}"
  ],
  "metadata": {
    "id": "CD4YQBeKoYSb"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_rf = build_model_simple_imb(smote,False,
  ↪ RandomForestClassifier(random_state=100),eval_data,
  ↪ y_train,**rf_params_simple)"
  ],
  "metadata": {
    "id": "71dpXscKZD4S"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
  ],
  "metadata": {
    "id": "9tjFxWu2ZD4S"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "prep = pipe_rf.named_steps['prep2']"
  ],
  "metadata": {
    "id": "e3jm-n7vIEeE"
  },
```

```
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "columns=prep.transform(X_test).columns"
  ],
  "metadata": {
    "id": "9W_IoPW6IICW"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "columns"
  ],
  "metadata": {
    "id": "QShsxML3ILX9"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "model = pipe_rf.named_steps['clf']"
  ],
  "metadata": {
    "id": "1XXyizWiIRBL"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "extract_features(model)"
  ],
  "metadata": {
    "id": "7H4KpQPDIfwC"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "rf_params_knn={'clf__max_features': 'sqrt', '
  ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 10}"
  ],
  "metadata": {
    "id": "2fbZ9un2osZE"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_rf = build_model_knn_imb(smote,False,
  ↪ RandomForestClassifier(random_state=100),eval_data,
  ↪ y_train,**rf_params_knn)"
  ],
  "metadata": {
    "id": "JaR0aMUFZD4S"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
  ],
  "metadata": {
    "id": "Z3PVMISiZD4S"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Support Vector Machine"
  ],
  "metadata": {
    "id": "czRAKXXfZD4S"
  }
},
{
  "cell_type": "code",
  "source": [
    "svm_params_simple={'clf__C': 0.1, 'clf__kernel': '
  ↪ linear'}"
  ],
  "metadata": {
    "id": "Owxxiz6fo8Dy"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_simple_imb(smote,False,SVC(
  ↪ random_state=100),eval_data,y_train,**
  ↪ svm_params_simple)"
  ],
  "metadata": {
    "id": "gOY4zABxZD4S"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
```

```
    "evaluate_model(pipe_svm,X_test,y_test)"
   ],
   "metadata": {
    "id": "cIg_b_mxZD4S"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "svm_params_knn={'clf__C': 1, 'clf__kernel': 'linear'}"
   ],
   "metadata": {
    "id": "Cwbb4IPGpOi0"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_svm = build_model_knn_imb(smote,False,SVC(
   ↪ random_state=100),eval_data,y_train,**svm_params_knn)"
   ],
   "metadata": {
    "id": "s37W6Z9sZD4S"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
   ],
   "metadata": {
    "id": "s6e_JFYqZD4S"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### MLP"
   ],
   "metadata": {
    "id": "JZwrqtDmZD4S"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "mlp_params_simple={'clf__activation': 'logistic', '
   ↪ clf__solver': 'sgd'}"
   ],
   "metadata": {
    "id": "gxKT1gtDqcWE"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_mlp = build_model_simple_imb(smote,False,
   ↪ MLPClassifier(random_state=100),eval_data,y_train,**
   ↪ mlp_params_simple)"
   ],
   "metadata": {
    "id": "PIlqxPGbZD4S"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
   ],
   "metadata": {
    "id": "Ojsd7IchZD4S"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "mlp_params_knn={'clf__activation': 'logistic', '
   ↪ clf__solver': 'sgd'}"
   ],
   "metadata": {
    "id": "WaoE4_EIqpSH"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_mlp = build_model_knn_imb(smote,False,
   ↪ MLPClassifier(random_state=100),eval_data,y_train,**
   ↪ mlp_params_knn)"
   ],
   "metadata": {
    "id": "OUmfxR4vZD4T"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
   ],
   "metadata": {
    "id": "7LaFzVy7ZD4T"
   },
```

```
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### XGBoost"
  ],
  "metadata": {
    "id": "HXlTLOIbZD4T"
  }
},
{
  "cell_type": "code",
  "source": [
    "xgb_params_simple={'clf__learning_rate': 0.001, '
  ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
  ],
  "metadata": {
    "id": "m9R4-iB4rZKf"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_simple_imb(smote,False,xgb.
  ↪ XGBClassifier(random_state=100),eval_data,y_train,**
  ↪ xgb_params_simple)"
  ],
  "metadata": {
    "id": "ITcp5aAwZD4T"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "JpXraeEPZD4T"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "xgb_params_knn={'clf__learning_rate': 0.01, '
  ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
  ],
  "metadata": {
    "id": "Fq0EPI8KrgL4"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_knn_imb(smote,False,xgb.
  ↪ XGBClassifier(random_state=100),eval_data,y_train,**
  ↪ xgb_params_knn)"
  ],
  "metadata": {
    "id": "RrylCUODZD4T"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "NSzeyp2YZD4T"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### AdaBoost"
  ],
  "metadata": {
    "id": "v47fN-L5ZD4T"
  }
},
{
  "cell_type": "code",
  "source": [
    "ada_params_simple={'clf__learning_rate': 0.1, '
  ↪ clf__n_estimators': 50}"
  ],
  "metadata": {
    "id": "p2-DMgLxsbYH"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_simple_imb(smote,False,
  ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
  ↪ ,**ada_params_simple)"
  ],
  "metadata": {
    "id": "T9LFifGYZD4T"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
```

```
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "2IQcmukWZD4T"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "ada_params_knn={'clf__learning_rate': 0.01, '
  ↪ clf__n_estimators': 500}"
  ],
  "metadata": {
    "id": "P9tOqam_sgB2"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_knn_imb(smote,False,
  ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
  ↪ ,**ada_params_knn)"
  ],
  "metadata": {
    "id": "jS_9p8JgZD4T"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "51VSQKuNZD4U"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### KNN"
  ],
  "metadata": {
    "id": "5YjsCHrSZD4U"
  }
},
{
  "cell_type": "code",
  "source": [
    "knn_params_simple ={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 19, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "cpmTQWptsyA4"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_simple_imb(smote,False,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_simple)"
  ],
  "metadata": {
    "id": "0rS-LwBOZD4U"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "F_3en5LdZD4U"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "knn_params_knn={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 20, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "3CW6-sgXs27i"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_knn_imb(smote,False,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_knn)"
  ],
  "metadata": {
    "id": "fza2g3-xZD4U"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
```

```
  "metadata": {
    "id": "MUFinWA5ZD4U"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "### With Feature Selection"
  ],
  "metadata": {
    "id": "P2H6JQCkuGTl"
  }
},
{
  "cell_type": "code",
  "source": [
    "eval_data = X_train.copy()"
  ],
  "metadata": {
    "id": "juC5XezpuGTr"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Logistic Regression"
  ],
  "metadata": {
    "id": "HDq3G2enuGTr"
  }
},
{
  "cell_type": "code",
  "source": [
    "lr_params_simple = {'clf__C': 0.01, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'sag'}"
  ],
  "metadata": {
    "id": "ZJSM5fNuuGTr"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_simple_imb(smote,True,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_simple)"
  ],
  "metadata": {
    "id": "eF_zTiPiuGTr"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "-kujMTn7uGTs"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "lr_params_knn = {'clf__C': 0.001, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'sag'}"
  ],
  "metadata": {
    "id": "TJhJIyMCuGTs"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_knn_imb(smote,True,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_knn)"
  ],
  "metadata": {
    "id": "RDNNSZ4IuGTs"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "cF5eDPFhuGTs"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Random Forest"
  ],
  "metadata": {
    "id": "DJGXVgVquGTs"
  }
},
{
  "cell_type": "code",
  "source": [
    "rf_params_simple ={'clf__max_features': 'sqrt', '
```

```
    ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 500}"
   ],
   "metadata": {
    "id": "WpnjaW7luGTs"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_rf = build_model_simple_imb(smote,True,
    ↪ RandomForestClassifier(random_state=100),eval_data,
    ↪ y_train,**rf_params_simple)"
   ],
   "metadata": {
    "id": "2f2N481cuGTs"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
   ],
   "metadata": {
    "id": "ystWQSpJuGTs"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "prep = pipe_rf.named_steps['prep2']\n",
    "columns=prep.transform(X_test).columns"
   ],
   "metadata": {
    "id": "HKI5xLq99T6P"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "columns=prep.transform(X_test).columns"
   ],
   "metadata": {
    "id": "QiKvymv89VG-"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "columns"
   ],
   "metadata": {
    "id": "kYYFwFWq9ZTI"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "get_features(pipe_rf,columns)"
   ],
   "metadata": {
    "id": "p-ycHV6-0EXH"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "model = pipe_rf.named_steps['clf']"
   ],
   "metadata": {
    "id": "iQfMcusU5hld"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "extract_features(model)"
   ],
   "metadata": {
    "id": "G2RkYuwo8rPR"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "rf_params_knn={'clf__max_features': 'sqrt', '
    ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 10}"
   ],
   "metadata": {
    "id": "_s_IniG_uGTt"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_rf = build_model_knn_imb(smote,True,
    ↪ RandomForestClassifier(random_state=100),eval_data,
    ↪ y_train,**rf_params_knn)"
   ],
   "metadata": {
    "id": "dL357MqhuGTt"
```

```
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
  ],
  "metadata": {
    "id": "-51DKejfuGTt"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Support Vector Machine"
  ],
  "metadata": {
    "id": "EQIXxnecuGTt"
  }
},
{
  "cell_type": "code",
  "source": [
    "svm_params_simple={'clf__C': 0.1, 'clf__kernel': '
  ↪ linear'}"
  ],
  "metadata": {
    "id": "-LSoHpSduGTt"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_simple_imb(smote,True,SVC(
  ↪ random_state=100),eval_data,y_train,**
  ↪ svm_params_simple)"
  ],
  "metadata": {
    "id": "OIXxEY6puGTt"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "l1EA2zbhuGTu"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "svm_params_knn={'clf__C': 1, 'clf__kernel': 'linear'}"
  ],
  "metadata": {
    "id": "E9NAbdT5uGTu"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_knn_imb(smote,True,SVC(
  ↪ random_state=100),eval_data,y_train,**svm_params_knn)"
  ],
  "metadata": {
    "id": "oHOhNaOauGTu"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "ImiQmSkiuGTu"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### MLP"
  ],
  "metadata": {
    "id": "WQXZfAwbuGTu"
  }
},
{
  "cell_type": "code",
  "source": [
    "mlp_params_simple={'clf__activation': 'logistic', '
  ↪ clf__solver': 'sgd'}"
  ],
  "metadata": {
    "id": "ZBhMEcnSuGTu"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_mlp = build_model_simple_imb(smote,True,
  ↪ MLPClassifier(random_state=100),eval_data,y_train,**
```

```
    ↪ mlp_params_simple)"
   ],
   "metadata": {
    "id": "WohjEoEpuGTv"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
   ],
   "metadata": {
    "id": "rcOMmVpVuGTv"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "mlp_params_knn={'clf__activation': 'logistic', '
    ↪ clf__solver': 'sgd'}"
   ],
   "metadata": {
    "id": "BG5TJ_UDuGTv"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_mlp = build_model_knn_imb(smote,True,
    ↪ MLPClassifier(random_state=100),eval_data,y_train,**
    ↪ mlp_params_knn)"
   ],
   "metadata": {
    "id": "dXT7EyjiuGTv"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
   ],
   "metadata": {
    "id": "UNnC8uabuGTv"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### XGBoost"
   ],
   "metadata": {
    "id": "ntHOg1cRuGTv"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "xgb_params_simple={'clf__learning_rate': 0.001, '
    ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
   ],
   "metadata": {
    "id": "-yKESAdyuGTv"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_xgb = build_model_simple_imb(smote,True,xgb.
    ↪ XGBClassifier(random_state=100),eval_data,y_train,**
    ↪ xgb_params_simple)"
   ],
   "metadata": {
    "id": "CewQATzDuGTv"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
   ],
   "metadata": {
    "id": "NGXswB74uGTw"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "xgb_params_knn={'clf__learning_rate': 0.01, '
    ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
   ],
   "metadata": {
    "id": "NhD_HQGauGTw"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_xgb = build_model_knn_imb(smote,True,xgb.
    ↪ XGBClassifier(random_state=100),X_train,y_train,**
    ↪ xgb_params_knn)"
   ],
   "metadata": {
```

```
    "id": "olORSLeyuGTw"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
   ],
   "metadata": {
    "id": "Y7GOEk9ouGTw"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "joblib.dump(pipe_xgb, 'final_model_check.joblib')"
   ],
   "metadata": {
    "id": "N17dBtHlO_JJ"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "get_features(pipe_xgb,columns)"
   ],
   "metadata": {
    "id": "oVEFdFPzJtMX"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "model = pipe_xgb.named_steps['clf']"
   ],
   "metadata": {
    "id": "CpMlduE7K3I4"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "extract_features(model)"
   ],
   "metadata": {
    "id": "cSq4FP4DK5e6"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### AdaBoost"
   ],
   "metadata": {
    "id": "vQJSaybiuGTw"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "ada_params_simple={'clf__learning_rate': 0.1, '
    ↪ clf__n_estimators': 50}"
   ],
   "metadata": {
    "id": "rmAVdbtAuGTx"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_ada = build_model_simple_imb(smote,True,
    ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
    ↪ ,**ada_params_simple)"
   ],
   "metadata": {
    "id": "edwrhPwsuGTx"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
   ],
   "metadata": {
    "id": "y3DEUy3ouGTx"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "ada_params_knn={'clf__learning_rate': 0.01, '
    ↪ clf__n_estimators': 500}"
   ],
   "metadata": {
    "id": "RXO7JPY7uGTx"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
```

```
    "pipe_ada = build_model_knn_imb(smote,True,
  ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
  ↪ ,**ada_params_knn)"
  ],
  "metadata": {
    "id": "3waTiMW6uGTx"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "DGYGv_d2uGTx"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### KNN"
  ],
  "metadata": {
    "id": "OgihqBVyuGTx"
  }
},
{
  "cell_type": "code",
  "source": [
    "knn_params_simple ={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 17, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "d9rdESMvuGTx"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_simple_imb(smote,True,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_simple)"
  ],
  "metadata": {
    "id": "96wlKOlxuGTx"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "ei3ImZiXuGTx"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "knn_params_knn={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 20, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "7eedDErUuGTy"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_knn_imb(smote,True,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_knn)"
  ],
  "metadata": {
    "id": "BEAPxZPtuGTy"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "b2xVfqA9uGTy"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "## With Imbalance Handling (SMOTETomek)"
  ],
  "metadata": {
    "id": "e3AfDcCLmqmg"
  }
},
{
  "cell_type": "markdown",
  "source": [
    "### No Feature Selection"
  ],
  "metadata": {
    "id": "v7yr7-A4mqmz"
  }
},
```

```
{
  "cell_type": "code",
  "source": [
    "eval_data = X_train.copy()"
  ],
  "metadata": {
    "id": "cOJ4PdbAmqm0"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Logistic Regression"
  ],
  "metadata": {
    "id": "QW32rBqGmqm0"
  }
},
{
  "cell_type": "code",
  "source": [
    "lr_params_simple = {'clf__C': 0.001, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'sag'}"
  ],
  "metadata": {
    "id": "rGVC3Fl8mqm1"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_simple_imb(smote_tomek,False,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_simple)"
  ],
  "metadata": {
    "id": "DF1fhAvNmqm1"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "3kY0Ub0_mqm2"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "lr_params_knn = {'clf__C': 0.001, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'saga'}"
  ],
  "metadata": {
    "id": "dk9J5kaMmqm3"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_knn_imb(smote_tomek,False,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_knn)"
  ],
  "metadata": {
    "id": "dcIvTWGsmqm3"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "_mL4BrAFmqm3"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Random Forest"
  ],
  "metadata": {
    "id": "1qYlgmF9mqm4"
  }
},
{
  "cell_type": "code",
  "source": [
    "rf_params_simple ={'clf__max_features': 'sqrt', '
  ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 10}"
  ],
  "metadata": {
    "id": "ljdYynYwmqm4"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_rf = build_model_simple_imb(smote_tomek,False,
  ↪ RandomForestClassifier(random_state=100),eval_data,
  ↪ y_train,**rf_params_simple)"
  ],
```

```
  "metadata": {
    "id": "Pp0lNdNHmqm4"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
  ],
  "metadata": {
    "id": "xwgICaVzmqm4"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "rf_params_knn={'clf__max_features': 'sqrt', '
  ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 10}"
  ],
  "metadata": {
    "id": "xUbeCZ5Umqm5"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_rf = build_model_knn_imb(smote_tomek,False,
  ↪ RandomForestClassifier(random_state=100),eval_data,
  ↪ y_train,**rf_params_knn)"
  ],
  "metadata": {
    "id": "sZ1iA9apmqm5"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
  ],
  "metadata": {
    "id": "twX_Z3vjmqm5"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Support Vector Machine"
  ],
  "metadata": {
    "id": "sHco7rkXmqm5"
  }
},
{
  "cell_type": "code",
  "source": [
    "svm_params_simple={'clf__C': 0.1, 'clf__kernel': '
  ↪ linear'}"
  ],
  "metadata": {
    "id": "zsyXsMGCmqm5"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_simple_imb(smote_tomek,False,
  ↪ SVC(random_state=100),eval_data,y_train,**
  ↪ svm_params_simple)"
  ],
  "metadata": {
    "id": "z5AWTsnXmqm6"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "K3yEcuemmqm6"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "svm_params_knn={'clf__C': 1, 'clf__kernel': 'linear'}"
  ],
  "metadata": {
    "id": "jALDSLSmmqm6"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_knn_imb(smote_tomek,False,SVC(
  ↪ random_state=100),eval_data,y_train,**svm_params_knn)"
  ],
  "metadata": {
    "id": "zcAEG49Zmqm7"
  },
  "execution_count": null,
  "outputs": []
```

```
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "BD5V-BFwmqm7"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### MLP"
  ],
  "metadata": {
    "id": "-vctnK5imqm7"
  }
},
{
  "cell_type": "code",
  "source": [
    "mlp_params_simple={'clf__activation': 'logistic', '
  ↪ clf__solver': 'sgd'}"
  ],
  "metadata": {
    "id": "iUUzlgu-mqm7"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_mlp = build_model_simple_imb(smote_tomek,False,
  ↪ MLPClassifier(random_state=100),eval_data,y_train,**
  ↪ mlp_params_simple)"
  ],
  "metadata": {
    "id": "zzjuRZKimqnB"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
  ],
  "metadata": {
    "id": "zHbYh1m6mqnC"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "mlp_params_knn={'clf__activation': 'logistic', '
  ↪ clf__solver': 'sgd'}"
  ],
  "metadata": {
    "id": "QOGpguBemqnC"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_mlp = build_model_knn_imb(smote_tomek,False,
  ↪ MLPClassifier(random_state=100),eval_data,y_train,**
  ↪ mlp_params_knn)"
  ],
  "metadata": {
    "id": "_JeF9J6jmqnC"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
  ],
  "metadata": {
    "id": "LRAdomyCmqnC"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### XGBoost"
  ],
  "metadata": {
    "id": "AVP5vJoVmqnC"
  }
},
{
  "cell_type": "code",
  "source": [
    "xgb_params_simple={'clf__learning_rate': 0.01, '
  ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
  ],
  "metadata": {
    "id": "SopKNGhLmqnD"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_simple_imb(smote_tomek,False,
  ↪ xgb.XGBClassifier(random_state=100),eval_data,y_train
  ↪ ,**xgb_params_simple)"
```

```
  ],
  "metadata": {
    "id": "q27Jct9DmqnD"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "QYaZG6mHmqnD"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "xgb_params_knn={'clf__learning_rate': 0.01, '
  ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
  ],
  "metadata": {
    "id": "py_wei6RmqnD"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_knn_imb(smote_tomek,False,xgb.
  ↪ XGBClassifier(random_state=100),eval_data,y_train,**
  ↪ xgb_params_knn)"
  ],
  "metadata": {
    "id": "QRXoGcIamqnE"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "p9ehpH57mqnE"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### AdaBoost"
  ],
  "metadata": {
    "id": "sSXa9TX_mqnE"
  }
},
{
  "cell_type": "code",
  "source": [
    "ada_params_simple={'clf__learning_rate': 0.1, '
  ↪ clf__n_estimators': 50}"
  ],
  "metadata": {
    "id": "Z2NXnOxhmqnF"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_simple_imb(smote_tomek,False,
  ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
  ↪ ,**ada_params_simple)"
  ],
  "metadata": {
    "id": "73MuZamPmqnF"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "Z0egBj6RmqnF"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "ada_params_knn={'clf__learning_rate': 0.1, '
  ↪ clf__n_estimators': 50}"
  ],
  "metadata": {
    "id": "2MvkeMREmqnF"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_knn_imb(smote_tomek,False,
  ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
  ↪ ,**ada_params_knn)"
  ],
  "metadata": {
    "id": "NFYBwIpkmqnG"
```

```
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "eui_JZ5emqnG"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### KNN"
  ],
  "metadata": {
    "id": "Lxhy9pE7mqnG"
  }
},
{
  "cell_type": "code",
  "source": [
    "knn_params_simple ={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 19, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "LkgMXiK7mqnG"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_simple_imb(smote_tomek,False,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_simple)"
  ],
  "metadata": {
    "id": "A_eTbfITmqnG"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "v6kygUobmqnH"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "knn_params_knn={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 19, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "cPS2FySPmqnH"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_knn_imb(smote_tomek,False,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_knn)"
  ],
  "metadata": {
    "id": "RhyCxSmAmqnH"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "dn2xfvogmqnH"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "### With Feature Selection"
  ],
  "metadata": {
    "id": "hDJOz6X7mqnI"
  }
},
{
  "cell_type": "code",
  "source": [
    "eval_data = X_train.copy()"
  ],
  "metadata": {
    "id": "DMOo7BUBmqnI"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Logistic Regression"
```

```
  ],
  "metadata": {
    "id": "KQO57vqymqnI"
  }
},
{
  "cell_type": "code",
  "source": [
    "lr_params_simple = {'clf__C': 0.01, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'saga'}"
  ],
  "metadata": {
    "id": "TslqqD6gmqnI"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_simple_imb(smote_tomek,True,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_simple)"
  ],
  "metadata": {
    "id": "ibh0Fdw8mqnI"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "FQJZMGelmqnJ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "lr_params_knn = {'clf__C': 0.001, 'clf__max_iter':
  ↪ 1000, 'clf__solver': 'sag'}"
  ],
  "metadata": {
    "id": "DujooxZbmqnJ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_lr = build_model_knn_imb(smote_tomek,True,
  ↪ LogisticRegression(random_state=100),eval_data,y_train
  ↪ ,**lr_params_knn)"
  ],
  "metadata": {
    "id": "h2tkqmohmqnJ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_lr,X_test,y_test)"
  ],
  "metadata": {
    "id": "v-VeQZslmqnK"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Random Forest"
  ],
  "metadata": {
    "id": "FRZeS-5ImqnK"
  }
},
{
  "cell_type": "code",
  "source": [
    "rf_params_simple ={'clf__max_features': 'sqrt', '
  ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 10}"
  ],
  "metadata": {
    "id": "4c8pOUsRmqnK"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_rf = build_model_simple_imb(smote_tomek,True,
  ↪ RandomForestClassifier(random_state=100),eval_data,
  ↪ y_train,**rf_params_simple)"
  ],
  "metadata": {
    "id": "Sl-PVvHGmqnK"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
  ],
  "metadata": {
    "id": "QwAsoy-smqnK"
  },
  "execution_count": null,
```

```
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "rf_params_knn={'clf__max_features': 'sqrt', '
  ↪ clf__min_samples_leaf': 5, 'clf__n_estimators': 10}"
  ],
  "metadata": {
    "id": "0VGB19LjmqnK"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_rf = build_model_knn_imb(smote_tomek,True,
  ↪ RandomForestClassifier(random_state=100),eval_data,
  ↪ y_train,**rf_params_knn)"
  ],
  "metadata": {
    "id": "ED7eeeY5mqnL"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_rf,X_test,y_test)"
  ],
  "metadata": {
    "id": "cA6didjbmqnL"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### Support Vector Machine"
  ],
  "metadata": {
    "id": "TPXcse63mqnL"
  }
},
{
  "cell_type": "code",
  "source": [
    "svm_params_simple={'clf__C': 0.1, 'clf__kernel': 'rbf
  ↪ '}"
  ],
  "metadata": {
    "id": "0ZGI4Sl0mqnL"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_simple_imb(smote_tomek,True,SVC
  ↪ (random_state=100),eval_data,y_train,**
  ↪ svm_params_simple)"
  ],
  "metadata": {
    "id": "5GaFsBf7mqnL"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "65DUOsgHmqnL"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "svm_params_knn={'clf__C': 0.1, 'clf__kernel': 'rbf'}"
  ],
  "metadata": {
    "id": "M8xVjf5emqnL"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_svm = build_model_knn_imb(smote_tomek,True,SVC(
  ↪ random_state=100),eval_data,y_train,**svm_params_knn)"
  ],
  "metadata": {
    "id": "NvpMFcHPmqnM"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_svm,X_test,y_test)"
  ],
  "metadata": {
    "id": "DCyGBVX6mqnM"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
```

```
    "#### MLP"
   ],
   "metadata": {
    "id": "-lHkprFNmqnM"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "mlp_params_simple={'clf__activation': 'tanh', '
   ↪ clf__solver': 'sgd'}"
   ],
   "metadata": {
    "id": "UDzT7JWbmqnM"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_mlp = build_model_simple_imb(smote_tomek,True,
   ↪ MLPClassifier(random_state=100),eval_data,y_train,**
   ↪ mlp_params_simple)"
   ],
   "metadata": {
    "id": "sw1wjqrWmqnM"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
   ],
   "metadata": {
    "id": "NOYanHM9mqnM"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "mlp_params_knn={'clf__activation': 'logistic', '
   ↪ clf__solver': 'sgd'}"
   ],
   "metadata": {
    "id": "kAHDHPJRmqnN"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_mlp = build_model_knn_imb(smote_tomek,True,
   ↪ MLPClassifier(random_state=100),eval_data,y_train,**
   ↪ mlp_params_knn)"
   ],
   "metadata": {
    "id": "XOUTinQImqnN"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_mlp,X_test,y_test)"
   ],
   "metadata": {
    "id": "_rbG39IomqnN"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "markdown",
   "source": [
    "#### XGBoost"
   ],
   "metadata": {
    "id": "XSyAPQJmmqnN"
   }
  },
  {
   "cell_type": "code",
   "source": [
    "xgb_params_simple={'clf__learning_rate': 0.01, '
   ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
   ],
   "metadata": {
    "id": "i8a8qaJFmqnN"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "pipe_xgb = build_model_simple_imb(smote_tomek,True,xgb
   ↪ .XGBClassifier(random_state=100),eval_data,y_train,**
   ↪ xgb_params_simple)"
   ],
   "metadata": {
    "id": "GKgaBBvsmqnN"
   },
   "execution_count": null,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
   ],
   "metadata": {
    "id": "GgbZo_rTmqnN"
   },
```

```
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "xgb_params_knn={'clf__learning_rate': 0.01, '
  ↪ clf__max_depth': 1, 'clf__min_child_weight': 1}"
  ],
  "metadata": {
    "id": "zviW54yBmqn0"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_xgb = build_model_knn_imb(smote_tomek,True,xgb.
  ↪ XGBClassifier(random_state=100),eval_data,y_train,**
  ↪ xgb_params_knn)"
  ],
  "metadata": {
    "id": "LaeUBz8Emqn0"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_xgb,X_test,y_test)"
  ],
  "metadata": {
    "id": "WzwOXgfBmqn0"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "markdown",
  "source": [
    "#### AdaBoost"
  ],
  "metadata": {
    "id": "Iu7eZmLrmqn0"
  }
},
{
  "cell_type": "code",
  "source": [
    "ada_params_simple={'clf__learning_rate': 0.1, '
  ↪ clf__n_estimators': 50}"
  ],
  "metadata": {
    "id": "c13wuop2mqn0"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_simple_imb(smote_tomek,True,
  ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
  ↪ ,**ada_params_simple)"
  ],
  "metadata": {
    "id": "63HhyrwkmqnO"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "EhLBfWCNmqnO"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "ada_params_knn={'clf__learning_rate': 0.01, '
  ↪ clf__n_estimators': 500}"
  ],
  "metadata": {
    "id": "1USojROhmqnP"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_ada = build_model_knn_imb(smote_tomek,True,
  ↪ AdaBoostClassifier(random_state=100),eval_data,y_train
  ↪ ,**ada_params_knn)"
  ],
  "metadata": {
    "id": "bYgJg2frmqnP"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_ada,X_test,y_test)"
  ],
  "metadata": {
    "id": "ZqWQPxEcmqnP"
  },
  "execution_count": null,
  "outputs": []
},
```

```
{
  "cell_type": "markdown",
  "source": [
    "#### KNN"
  ],
  "metadata": {
    "id": "NTQCXmiZmqnP"
  }
},
{
  "cell_type": "code",
  "source": [
    "knn_params_simple ={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 17, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "nHga7ao7mqnQ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_simple_imb(smote_tomek,True,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_simple)"
  ],
  "metadata": {
    "id": "A24noil8mqnQ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "_8VOgXPMmqnQ"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "knn_params_knn={'clf__metric': 'euclidean', '
  ↪ clf__n_neighbors': 19, 'clf__weights': 'uniform'}"
  ],
  "metadata": {
    "id": "DOlxfSJYmqnR"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "pipe_knn = build_model_knn_imb(smote_tomek,True,
  ↪ KNeighborsClassifier(),eval_data,y_train,**
  ↪ knn_params_knn)"
  ],
  "metadata": {
    "id": "krYIOVOZmqnR"
  },
  "execution_count": null,
  "outputs": []
},
{
  "cell_type": "code",
  "source": [
    "evaluate_model(pipe_knn,X_test,y_test)"
  ],
  "metadata": {
    "id": "IkU2ZvXFmqnS"
  },
  "execution_count": null,
  "outputs": []
}
]
}
```

XI. Acknowledgement

First of all, I would like to thank my thesis adviser Ma'am Perl, who gave me helpful suggestions every time I have a concern regarding my thesis and guidance during the defense. I would also like to thank the panel who gave numerous insights during the defense which enabled me to see the shortcomings of my thesis and what I need to improve if I would ever involve myself in future data science projects.

I am endlessly thankful to my wonderful mother and my grandmother, who supported me throughout college, encouraged me and tried to understand me, especially on my later years in the university. I would also like to give my sincerest appreciation to my friends. My first college friend and my game buddy Carl, even if you think that you are a B.I. or a bad influence, without your support and sometimes persuading me to unwind through games, I wouldn't be able to stand the pressure of studies. My stellar friend Alona, thank you for always checking up on me and my progress on life. You are an inspiration of hard work along with my levelheaded friend Regina, who showed me that resiliency pays off. Thank you for tolerating me throughout college. My dark humor buddy, Carms, aside from being an example of perseverance, thank you for the laughs especially the hugots, which gave me solace that some parts of life do not need to be that serious. My friend Jen, your compassion and life advices are a blessing. Thank you for reassuring me when acads and life hit hard. My friend Ella, I'm forever grateful to the different opportunities you introduced to me to upskill myself. You also taught me the importance of taking things step by step but not giving up on the task. Last but not the least, my SP buddy Gi, thank you for introducing me to productivity stuff and locations like the National Library and coworking spaces that enabled me to finish my SP.

Finally, I would like to express my deepest gratitude to DOST-SEI for all their support during my college years. I would also like to thank their staff for always accommodating my inquiries.