University of the Philippines Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

Using Machine Learning Algorithms and Integrons to Predict Antimicrobial Resistance in Common Infection-causing Bacteria

A special problem in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

Lady Edronalee J. Tavu

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 "Using Machine Learning Algorithms and Integrons to Predict Antimicrobial Resistance in Common Infection-causing Bacteria" prepared and submitted by Lady Edronalee J. Tavu in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, Ph.D.
Adviser

EXAMINERS:

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

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

Antimicrobial resistance (AMR) poses a significant global threat to public health. In recent years, machine learning (ML) algorithms have emerged as tools for predicting AMR patterns and guiding antibiotic treatment decisions. This study aimed to explore the predictive capabilities of three ML algorithms, namely Extreme Gradient Boosting (XGBoost), Support Vector Machines (SVM), and Random Forest (RF), with the incorporation of integron features along with AMR genes, to construct a web-based *in silico* antibiogram tool. Utilizing a comprehensive dataset comprising bacterial isolates - *Acinetobacter baumannii*, *Escherichia coli*, and *Klebsiella pneumoniae* and their corresponding resistance profiles to five antibiotics of interest - cefotaxime, ceftriaxone, ciprofloxacin, gentamicin, and levofloxacin, the study train and evaluate the performance of the ML models. Generally, XGBoost and RF perform better than SVM with AUC score up to 0.93. Central to this approach is the integration of integron features, which play a pivotal role in mediating horizontal gene transfer and facilitating the dissemination of resistance genes among bacterial populations. Building upon the robust models, this study developed a user-friendly web application that enables healthcare practitioners and researchers to input whole-genome sequences and rapidly obtain predictions regarding antibiotic resistance profiles based on integrated ML models.

Keywords: Antimicrobial resistance, integrons, whole-genome sequencing, gram-negative bacteria, machine learning

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Rapid diagnostics of Antimicrobial Resistance (AMR) are essential in developing an effective antibiotic treatment for patients. AMR refers to events when microorganisms develop an ability to withstand antibiotic drugs, leading to treatment failure and eventually death. According to the World Health Organization [2], AMR is one of the top 10 global public health threats in the world. In 2019, there were an estimated 4.95 million deaths related to bacterial AMR in 204 countries and territories [3]. Additionally, by 2030, up to 24 million people could experience extreme poverty due to AMR [4].

One of the diagnostic tools of AMR is the traditional Antimicrobial Susceptibility Testing (AST), carried out in the laboratory by a medical technologist. It aids physicians in choosing a specific drug and dosage for the infection of a particular patient. However, AST consists of complex procedures that can only be done in a fully equipped wet laboratory [5]. There are 3 major AST methods, which are disk diffusion, broth dilution, and agar dilution. Both broth dilution and agar dilution are used to quantitatively determine the minimal concentration (usually expressed in mg/ml) of antimicrobial agents to inhibit the bacteria. Meanwhile, disk diffusion is qualitative, wherein it has a susceptibility category that is susceptible, intermediate, or resistant, rather than obtaining the minimum inhibitory concentrations (MIC) [6].

Recently, new approaches are emerging to identify strains that are resistant or susceptible to a certain antibiotic, such as genotype-based machine learning models. The commonly used algorithms for AMR include Naïve Bayes, Decision Trees, Random Forests, Support Vector Machines, and Artificial Neural Networks [7]. The decreasing cost and increasing speed of high-throughput sequencings, such as whole-genome sequencing (WGS), have made it increasingly feasible to perform an in-depth analysis of pathogens for clinical decision-making [8]. Due to the advancement in next-generation sequencing (NGS), it paved the way for numerous techniques in studying AMR as it provides genetic information that can reveal a lot of determinants associated with AMR. These determinants include genetic markers and mobile genetic elements such as resistance integrons. It is established that mutations and integrons contribute to the spread of resistance genes among bacteria leading to accelerated AMR evolution [9][10]. Integrons are genetic elements that, while incapable of self-movement, carry gene cassettes that can be mobilized to other integrons or to a different bacterial genome. This mobility provides integrons with the ability to transfer between and within DNA structures. Consequently, resistance genes can be inserted into integrons as cassettes and subsequently disseminated to other genomic regions. This process contributes to the emergence of diverse multidrug resistance patterns in bacteria.

Moreover, different encoding methods are used in predicting AMR, as the input data are whole-genome sequences consisting of A, T, C, and G. These deoxyribonucleic acid (DNA) sequences are converted into binary data for model construction. With the help of machine learning and other software tools, it is possible to determine the mechanisms of AMR by means of identifying the genetic markers and detecting the integrons on the strains [11][12][13][14][1]. It is significant to develop a comprehensive knowledge of AMR mechanisms to understand the resistance patterns which may aid in developing a public health strategy. In past studies, predictions of AMR were formulated based on known AMR genes and resistance-causing mutations. However, this paper presents a novel approach in predicting AMR based on the presence of integrons and AMR genes on WGS of common infection-causing bacteria using machine learning.

B. Statement of the Problem

AST is a laboratory procedure to identify the effective antimicrobial regimen for individual patients. However, it is time-consuming with a turnaround time that is usually between 12 hours and 48 hours [15]. AST is also labor-intensive because it can only be performed by medical technicians and it needs a support staff when managing a test to avoid laboratory-acquired infections [5]. Additionally, some clinical laboratories do not have access to AST platforms, leading to the liability of AMR to affect most of the low-income sector, as AMR increases their hospital length of stay and healthcare costs.

Furthermore, the current machine learning studies in predicting AMR do not include a tool that can be used for clinicians to identify the AMR of other bacteria. Present-day studies focus on providing and evaluating the accuracy of machine learning algorithms. There was no implementation involved to provide automation in getting the result if an isolate is resistant or susceptible to a specific antibiotic.

This study aims to predict AMR in common infection-causing bacteria based on whole-genome sequencing using machine learning algorithms and provide an implementation of it as a web application. This study focuses on the following research questions:

1. What are the AMR genes associated with integrons in common infection-causing bacteria?

2. Which antibiotics are bacteria isolates susceptible or resistant to?

3. What machine learning algorithm will provide high accuracy even in unseen data?

C. Objectives of the Study

This study aims to predict AMR based on the presence of integrons and AMR genes on WGS of common infection-causing bacteria using machine learning, and establish a web application system to serve as a tool for clinicians. The study has the following objectives:

1. Use Extreme Gradient Boosting, Support Vector Machine and Random Forest algorithms to predict drug resistance to antibiotics including ciprofloxacin, cefotaxime, ceftriaxone, gentamicin, and levofloxacin tested in *Acinetobacter baumannii*, *Escherichia coli*, and *Klebsiella pneumoniae*. According to the Philippine Department of Health (DOH) [16], these microorganisms have an increasing resistance trend to antimicrobials.

   (a) Determine predictive features from the dataset in determining drug resistance in the models.

   (b) Evaluate the performance of the models via Receiver Operating Characteristic Curve (ROC) analysis, Area Under the ROC Curve (AUC), and classification metrics (accuracy, precision, and recall).

2. Develop a web-based application built on the machine learning model to deliver early detection of AMR in pathogens and prescription of more effective antibiotics.

   (a) The application can read a whole-genome sequencing data.

   (b) The application displays the predicted antibiograms.

   (c) The application has a section that displays integron presence and detected AMR genes on WGS.

   (d) The application has demo samples of bacteria isolates for users to see how the application works.

(e) The application has a section where the user can see the details of prediction models used for training the data set, including performance summary.

D. Significance of the Project

The research provides a machine learning approach for resistance prediction that may accelerates the treatment of patients in the future. It addresses the issues that prolong the turnaround time of traditional AST, such as the lack of automation and lack of supplies needed for testing, which may result in inflicting more harm to the patients. The continuous decrease in cost and increasing speed of high-throughput sequencing technologies, such as WGS, significantly improves the feasibility of conducting thorough pathogen analysis for clinical decision-making. Moreover, the analysis of WGS data with machine learning narrows down the need for extensive AST, making the overall process more streamlined and economical.

The web application benefits clinicians by guiding them in making faster decisions on the specific antibiotic regimens needed for a patient. Machine learning algorithms and other software tools also help identify marker genes and integrons linked with AMR. The identification of integrons in bacteria proves useful in developing public health strategies, as they commonly contribute to the spread of antimicrobial resistance. During epidemics, integrons can serve as molecular markers to identify the origin and spread of resistant bacteria. Analyzing integron profiles aids in identifying related strains and tracking the source of infections. This information is crucial for implementing efficient management strategies, locating resistance reservoirs, and preventing future spread within hospital and community settings. Lastly, the findings serve as a valuable reference resource for further study of AMR.

E. Scope and Limitations

1. The study includes the following antibiotics: ciprofloxacin, cefotaxime, ceftriaxone, gentamicin, and levofloxacin.

2. The study implements Extreme Gradient Boosting, Support Vector Machine, and Random Forest algorithms for model construction.

3. The machine learning models are trained on WGS isolates of *Acinetobacter baumannii*, *Escherichia coli*, and *Klebsiella pneumoniae*.

4. The accuracy of prediction depends on the trained model.

5. The dataset used for model development and evaluation only comes from one source, the National Center for Biotechnology Information (NCBI) website.

F. Assumptions

1. There are no invalid inputs / uploads, only the FASTA files of WGS isolates.

2. The system is designed to be used by researchers and health care professionals.

II. Review of Related Literature

Numerous studies show how machine learning can help predict antimicrobial resistance (AMR) in various bacteria. With the assistance of machine learning (ML), it may provide early detection of AMR compared to traditional antimicrobial susceptibility testing (AST), which can take 2-4 days. Several models exhibit high accuracy and have the capacity to guide the clinical decision-making process.

Anahtar et al. [17] enumerated three domains in which ML can help reduce the burden of AMR. The first domain is the use of ML for predicting AMR from pathogen genomic sequences. This domain can provide accurate predictions of AST using a large training data set and by applying feature extraction. The second domain is the use of ML to understand the mechanisms of AMR and antimicrobial discovery. In this domain, ML can discover biochemical inhibitors of essential genes in bacteria. On the other hand, the third domain is about utilizing ML for antimicrobial stewardship. The third domain discussed the analysis of Electronic Health Records (EHR) to create clinical decision support systems for antibiotic stewardship programs.

Various studies are applying ML in the first and second domains due to the challenges in working with EHR data, as it contains entry errors due to human input. Studies with AMR prediction mostly focused on classifying susceptible and resistant phenotypes, while others predict minimum inhibitory concentrations (MIC) based on gene content. For instance, Ren et al. [11] predicted the AMR from the whole-genome sequencing of *Escherichia coli* using different encoding methods and ML models. The encoding methods including label, one-hot, and frequency matrix chaos game representation (FCGR), were utilized to prepare the dataset for ML particularly Support Vector Machine (SVM), Random Forest (RF), Logistic Regression (LR), and Convolutional Neural Network (CNN). The research result found that all encoding methods are effective in putting together the genomic data for machine learning and deep learning algorithms. The whole-genome sequences require encoding methods, as whole-genome sequences consisting of an alphabet are not readable by the computer. The input dataset is converted into a numerical variable, so that the machine can understand the genomic sequence. In this way, the researchers can also identify which encoding method will provide the highest accuracy in ML models. One of the encoding methods included by Ren et al. [11] is label encoding, which converts each alphabet into a unique integer. Meanwhile, one-hot encoding is a process of creating dummy variables. It encodes the genomic sequences into a binary matrix and converts them into a binary vector. The binary vector of the DNA sequence will serve as an input for ML models. The last encoding method is the FCGR encoding, that transforms the sequences into a visual representation, such as images or matrices. FCGR is useful in sequence comparison and phylogenetic analysis.

A study made by Chowdhury et al. [18] predicted AMR in gram-positive bacteria using the game-theory algorithm and SVM model. The Game Theoretic Dynamic Weighting-based Feature Evaluation (GTDWFE) was used for feature selection to identify the most appropriate features for accurate prediction. After identifying the relevant feature, an SVM model with 10-fold cross-validation was trained in it. Other researchers use extreme gradient boosting (XGBoost) for MIC prediction models, such as the study of Nguyen et al. [12] due to the algorithm's scalability and built-in feature selection. Their research predicts the MIC of 15 antibiotics using WGS data of nontyphoidal *Salmonella* through the XGBoost model. The accuracy of the model was computed for each category of isolate sources, but the overall model has an average accuracy of 95%. This is higher compared to the SVM model used by Chowdhury et al. [18] which has an accuracy between 87% and 90%. The XGboost model seems to provide higher accuracy on whole-genome sequence data than the traditional machine learning, such as SVM, LR, and RF, that is used in other studies of AMR. This may be due to the integrated feature importance of the XGBoost model, and the prediction was performed on a 10-fold cross-validation. Additionally, the researcher prevented overfitting by having a validation set. Although the paper of Nguyen et al. [12] predicts the antimicrobial MICs, it is still relevant, as the lower MIC value indicates that the antibiotics are more effective antimicrobial agents and can impede the growth of bacteria. Additionally, if the MIC number is added to the prediction of AMR, then the resistance level may also be identified, and not just the result if an isolate is resistant or susceptible to a specific antibiotic.

In determining the genes associated with antibiotic resistance, the study of Ren et al. [11] was able to identify mutations that may contribute to the resistance through the single nucleotide polymorphisms (SNP) association study and SnpEff Software. The SNP association study was performed using the Ensemble Feature Selection (EFS) package of R, while the SnpEff software was used to annotate the corresponding genes of SNPs. The EFS package has eight feature selection methods for binary classifications, while the SnpEff software annotates and predicts the effects of the difference between a genome and a reference genome on genes and proteins [19][20]. One of the marker genes that they identified is the gene *nhaA* that is associated with cefotaxime, ceftazidime, and gentamicin resistance. This gene has Na+/H+ antiporter function in *E. coli* that may influence antibiotic resistance. Additionally, Nguyen et al. [12] performed a genomic analysis and identified the most important genomic regions of the strains that contribute to the MIC predictions. They used the PATRIC services, such as genome assembly and annotation service, and Basic Local Alignment Search Tool (BLAST), to find if there are significant SNPs in the k-mers.

Van Camp et al. [21] also analyzed the different methods of understanding the mechanisms of AMR using published studies from the past years. One of the approaches was the identification of known genomic signatures of AMR from whole genome sequence data. In this approach, the nucleotide basic local alignment search tool (BLASTn) and Burrows-Wheeler Alignment Maximal Exact Matches (BWA-MEM) tool, a faster sequence aligner than BLASTn, was used to match the assembled sequences with a reference genome while ensuring the quality of genome assembly. According to the article, the tools can generate accurate results, and this approach will eliminate the days needed to grow a bacterium that is required on traditional AST. In this way, a specific patient can immediately start their antibiotic regimen.

There are also other factors that contribute to resistance mechanisms, such as the integrons, which are genetic platforms for the acquisition, transport, and expression of gene cassettes that are usually antibiotic resistance genes [10]. Gene cassettes have the capability of transferring within the organism's genome and transferring to another organism through horizontal gene transfer that can spread resistance and lead to drug resistance in various bacteria. Torres-Elizalde et al. [13] performed detection of AMR integrons in *Salmonella enterica* isolates using IntFinder 1.0 and Integron Visualization and Identification Pipeline (I-VIP) v1.2. The IntFinder 1.0 is an application that detects an integron in next-generation sequencing (NGS) data using k-mer alignment (KMA) [14]. Meanwhile, the I-VIP v1.2 can identify, classify, annotate, and visualize integrons in complete genomes and assembled metagenomes [22]. They were able to effectively identify these integrons in *S. enterica* isolates from countries of the Andean community and their association with antibiotic resistant genes using both tools. They found out that class 2 integrons were more prevalent in the community. Class 2 integrons are commonly associated with the Tn7 transposon family, which encodes resistance to the antibiotics such as trimethoprim and streptomycin.

On the other hand, Néron et al. [1] used an IntFinder 2.0, an updated version of the tool that is more efficient and can handle incomplete genome data, to identify and analyze the integrons of almost 4,000 *Klebsiella pneumoniae* genomes. After refactoring the first version of IntFinder and adding other features, such as unit tests and multi-fasta files as input, they identified 1,855 complete integrons in 1,677 genomes of *K. pneumoniae* that is 42% of the genomes. The integrons found in *K. pneumoniae* have 165 different gene families, and analyzing the 20 most frequent gene families indicates that 12 of which are antibiotic resistance genes connected with resistance to antiseptics, diaminopyrimidine, chloramphenicol, or beta-lactams. Detecting the integrons in a bacterium and having in-depth information on its resistance mechanism will help in developing an appropriate intervention technique.

Most of these studies focus on gram-negative bacteria as they are the common infection-causing bacteria and are one of the public health problems because of their high resistance to antibiotics [23]. Gram-negative bacteria have multiple thin layers of membranes that do not absorb foreign materials, making them more resistant to antibiotics. Meanwhile, the membrane of gram-positive bacteria absorbs foreign material, making it easier to eliminate. *Acinetobacter baumannii*, for instance, is an example of gram-negative bacteria that can infect the blood, urinary tract, lungs, and wounds. Most of the time, *A. baumanni* infections happen in healthcare settings [24] where sharing of equipment and devices takes place. It is highly resistant to antibiotics such as gentamicin and tobramycin [25]. Other common infection-causing gram-negative bacteria include *Escherichia coli* and *Klebsiella pneumoniae*. Some types of *E. coli* can cause diarrhea and intestinal infection. It is also found to be resistant to amoxicillin, cefuroxime, and ceftriaxone [26]. On the other hand, *K. pneumoniae* can cause infections such as urinary tract infection, meningitis, pneumonia, and bloodstream infections and it is known to be resistant to ciprofloxacin, amoxicillin, and ceftazidime [27][28].

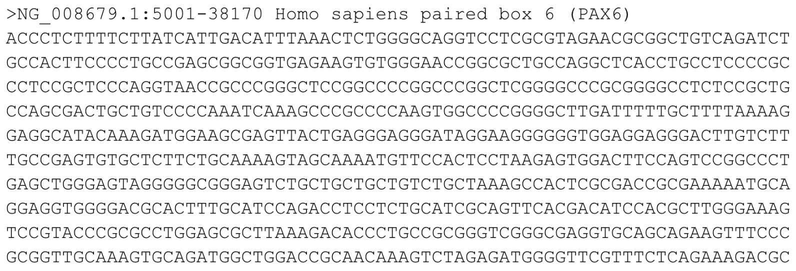
Furthermore, the main goal of this research is to implement a web-based application that delivers early detection of AMR in pathogens and provides prescription recommendations for more effective antibiotics. Chowdhury et al. [18] implemented an application called Prediction of Antimicrobial Resistance via Game Theory (PARGT) tool to identify antimicrobial-resistance genes for bacteria. PARGT accepts user input of sequences and dispenses an output containing the predicted AMR sequences. In this study, the research objective is to also provide a tool for clinicians, and not just to evaluate the machine learning algorithms. However, in Chowdhury's research, the tool they created is a software program that needs to be downloaded on the user's machine. Therefore, end users are required to install a Jupyter Notebook and run the scripts before a Graphical User Interface (GUI) appears. These additional steps may be challenging for users without knowledge of packages and scripts. These shortcomings may be addressed by developing a web-based application to eliminate the previously mentioned unnecessary steps. With a web-based application, users no longer need to download anything, and it is easy to navigate, as almost everyone has experience visiting a website. They only need to have an internet connection and a browser to access the website. Additionally, the application is lightweight and accessible on any device, including smartphones, laptops, and desktop computers.

III. Theoretical Framework

A. Whole-Genome Sequencing Analysis

A WGS data has the ability to uncover genomic knowledge, but one of the challenges in deriving this information is the quality control of the data [29]. A poor-quality WGS can create misleading conclusions. Thus, it requires preparation before undergoing analysis to generate a more accurate and meaningful study.

In the field of sequence analysis, the FASTA format is a text-based format to store either nucleotide sequences or amino acid (protein) sequences. Typically, the initial line of a FASTA file commences with a ">" symbol. This initial line is referred to as the "description line" and provides descriptive details regarding the sequence presented in the following lines. The description often includes the sequence's ID or name, such as gene names. The structure of the FASTA file is illustrated in Figure 1 [30].



```
>NG_008679.1:5001-38170 Homo sapiens paired box 6 (PAX6)
ACCCTCTTTTCTTATCATTGACATTTAAACTCTGGGGCAGGTCCTCGCGTAGAACGCGGCTGTCAGATCT
GCCACTTCCCCTGCCGAGCGGCGGTGAGAAGTGTGGGAACCGGCGCTGCCAGGCTCACCTGCCTCCCCGC
CCTCCGCTCCCAGGTAACCGCCCGGGCTCCGGCCCCGGCCCGGCTCGGGGCCCGCGGGGCCTCTCCGCTG
CCAGCGACTGCTGTCCCCAAATCAAAGCCCGCCCCAAGTGGCCCCGGGGCTTGATTTTTGCTTTTAAAAG
GAGGCATACAAAGATGGAAGCGAGTTACTGAGGGAGGGATAGGAAGGGGGGTGGAGGAGGGACTTGTCTT
TGCCGAGTGTGCTCTTCTGCAAAAGTAGCAAAATGTTCCACTCCTAAGAGTGGACTTCCAGTCCGGCCCT
GAGCTGGGAGTAGGGGGCGGGAGTCTGCTGCTGCTGTCTGCTAAAGCCACTCGCGACCGCGAAAAATGCA
GGAGGTGGGGACGCACTTTGCATCCAGACCTCCTCTGCATCGCAGTTCACGACATCCACGCTTGGGAAAG
TCCGTACCCGCGCCTGGAGCGCTTAAAGACACCCTGCCGCGGGTCGGGCGAGGTGCAGCAGAAGTTTCCC
GCGGTTGCAAAGTGCAGATGGCTGGACCGCAACAAAGTCTAGAGATGGGGTTCGTTTCTCAGAAAGACGC
```

Figure 1: FASTA file of PAX6 gene.

In the workflow of WGS analysis, the procedure can occur into seven steps [31]. These are raw read quality control, data preprocessing, alignment, variant calling, genome assembly, genome annotation, and other advanced analyses like the application of machine learning. In order to achieve accurate and reliable variation detection, raw files like FASTQ files need to be eliminated from poor-quality reads or sequences. Sequence alignment, on the other hand, is the process of arranging the primary sequences to identify regions of similarity that may be a result of functional, structural, or evolutionary relationships between the sequences [32]. This step requires a reference genome. Once reads are aligned to the reference genome, variants can be identified by comparing the sample genome to the reference genome. These detected variants may be associated with a disease or genomic noise. Genome assembly and genome annotation are the process of attaching biological information to sequences, giving meaning to it.

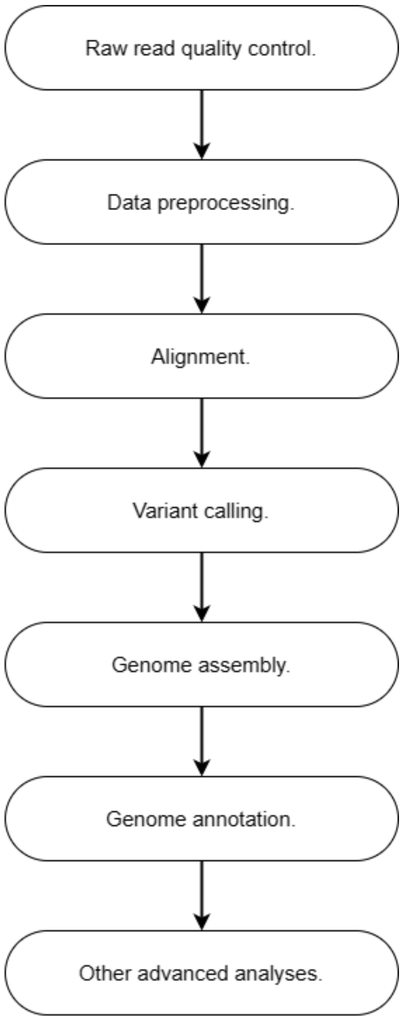


Figure 2: Workflow of whole-genome sequencing.

A.1 Integron Detection: IntFinder

IntegronFinder is a tool that can detect integrons in DNA sequences. The IntFinder version 1 is available both online and can be downloaded to be used in the command line. However, the first version of IntegronFinder is now deprecated as it was coded in Python v2.7 [1]. A new IntegronFinder v2 was built by refactoring the first version to improve its accuracy. The second version now includes databases like AMRFinderPlus and Hidden Markov Model (HMM) Profile as a reference database for annotating antibiotic resistance genes. It also accepts multiple inputs of FASTA files and outputs three files which are the main output file that contains the integron information, the summary file, and a copy of the standard output that can be eliminated using the mute command.

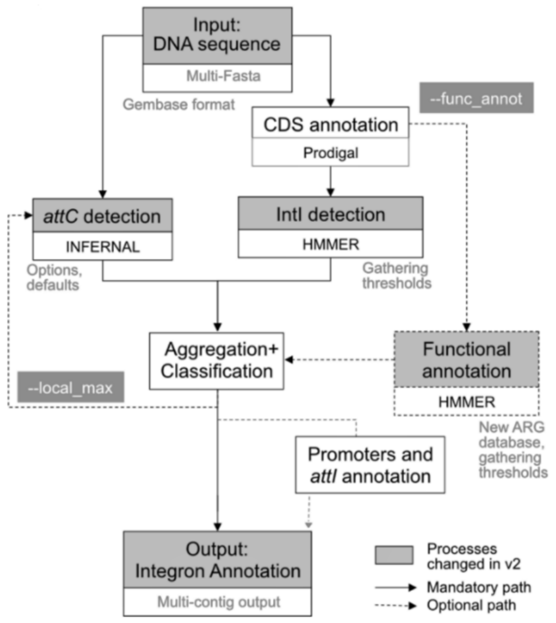


Figure 3: Steps used by IntegronFinder to identify and annotate integrons [1].

Initially, the DNA sequence's coding sequences (CDS) are annotated with Prodigal by IntegronFinder. This step aims to identify and annotate the protein-coding regions. Subsequently, IntegronFinder employs a dual approach to detect both the integron integrase (intI) and *attC* recombination sites independently. The detection of the integron integrase involves utilizing the intersection of two distinct HMM profiles. One profile is specific to tyrosine-recombinase (PF00589), while the other targets integron integrase, particularly in the vicinity of the patch III domain of tyrosine recombinases. On the other hand, the *attC* recombination sites are identified using a covariance model (CM) that encompasses the conserved sequence positions and accounts for the secondary structure.

To consolidate the results, IntegronFinder integrates the outcomes of the previous steps and classifies the elements into three distinct categories. First, complete integron refers to an integron in which the integron integrase is found in proximity to one or more *attC* sites, signifying a fully functional integron system. Second, *In0* element refers to an event where only the integron integrase is present without any adjacent *attC* sites. This suggests the absence of a complete integron structure, potentially indicating the existence of an incomplete or non-functional integron. Third, CALIN (Cluster of *attC* sites Lacking INtegrase nearby) element, this category encompasses a cluster of *attC* sites that lack a nearby integron integrase. To avoid false positive results, it involves filtering out *attC* sites that occur as singular occurrences, or singletons.

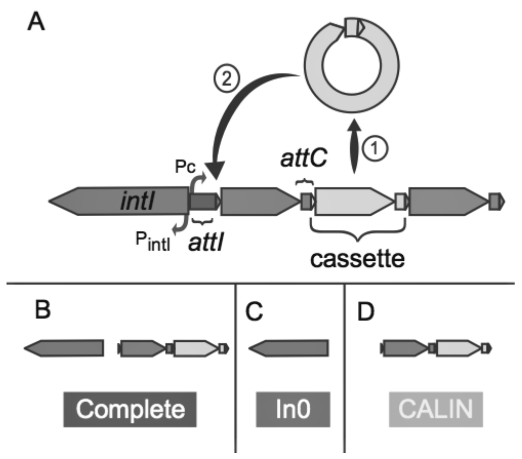


Figure 4: Key genetic elements of integrons. [1].

A.2 AMR Genes Detection: ABRicate

ABRicate is a Perl-based tool designed for efficient screening of contigs to detect antimicrobial resistance or virulence genes. It does not have compatibility with FASTQ reads. The tool includes several bundled databases such as NCBI, CARD, ARG-ANNOT, Resfinder, MEGARES, EcOH, PlasmidFinder, Ecoli VF, and VFDB. By default, the NCBI database is used, but users have the flexibility to select a different database by utilizing the "–db" option on the command line [33].

B. Machine Learning

Machine learning is a field under artificial intelligence and it is generally about machines imitating intelligent human behavior. It is developing computer programs that have the capability to access data and utilize it to learn and produce a data-driven output. The machine learning process starts by observing the data and looking for patterns in it. This will produce inferences based on the samples provided. The two main machine learning types are supervised, and unsupervised learning.

1. Supervised learning is machine learning with models which are trained with labeled data sets to yield the desired output. It can be classified into two types of problems which are classification and regression. In classification, it is an algorithm that assigns the samples into specific categories, while regression is used in understanding the relationship between dependent and independent variables.

2. Unsupervised learning is machine learning which uses algorithms to analyze and cluster unlabeled datasets. The system searches for common characteristics or trends without the need for human intervention. Unsupervised learning is useful in clustering, association, and dimensionality reduction. In clustering, it groups the unlabeled samples in accordance with their similarities and dif-

ferences. Meanwhile, the association is used for finding relationships between variables and dimensionality reduction is a technique to reduce the data to a reasonable size while maintaining the data quality.

B.1 Extreme Gradient Boosting

XGBoost is a supervised machine learning tree-based algorithm. It is a scalable and accurate implementation of a gradient-boosting framework. This algorithm has the capability to create decision trees in sequential form and weights are assigned to independent variables which are distributed in the decision tree to predict results. If the variables are predicted wrong by the tree, the weight of it is increased and the variables are put into the second decision tree. This process is an iterative training of an ensemble of shallow decision trees wherein each iteration will use the error residuals of the previous model to fit the next model [34]. Hence, XGBoost can generate a stronger and more precise model.

B.2 Support Vector Machine

SVM is a supervised machine learning algorithm that is derived from statistical learning theory. The goal of the algorithm is to find the best hyperplane that separates the sample of one category from the other category. The hyperplane is the decision surface whereas the support vectors are the data points that lie closest to the hyperplane [35]. These points are usually the most difficult to classify. The algorithm of SVM is based on kernel methods where features are transformed into different forms using a kernel function [36]. A kernel function maps the data into simplified linear ones which makes it easier to separate the data points. With the help of kernels, SVM became more flexible and has the ability to handle non-linear data.

B.3 Random Forest

RF is a supervised machine learning algorithm that is based on decision trees. It has a number of decision trees operating as an ensemble [37]. The results are computed by calculating the average of all the decision tree values which means that increasing the number of trees will increase the precision. Additionally, the trees are modeled in parallel. For RF, it is necessary to have a low correlation between models or trees to protect each other from individual errors. This will make sure that not all trees will have a wrong outcome.

C. Explainability in Machine Learning

Machine learning models are complex to rationalize, making them black-box models. However, with Explainability in machine learning, humans will be able to understand why and how a machine learning model makes predictions [38]. It will provide transparency and trust to the model as it can justify the attributes that the model depends on [39]. Some machine learning has a built-in technique for performing explainability in machine learning:

C.1 Built-in Feature Importance

Feature importance determines which variable or features are relevant. It helps in realizing the internal logic of a model. The feature importance scores can, later on, improve the model by incorporating a feature selection. Random forest and Extreme Gradient Boosting have built-in feature importance measures which allow to determine which variables in the dataset are most important in a system's decision-making. Furthermore, this analysis will contribute to a better comprehension of the relationship between the presence of integrons and the specific AMR genes that exhibit the strongest association with resistance to a particular antibiotic.

The built-in feature importance in Random Forest and Extreme Gradient Boosting can be computed through Gini importance. The Gini importance measures the variable importance using the Gini impurity index that is used for the calculation of splits in trees. When a variable is constantly used to make key decisions with decision trees, the higher its relative importance is. Meanwhile, feature importance for Support Vector Machine only works for a model with a linear kernel or when the data can be separated using a single line.

D. Evaluation Measures

D.1 Accuracy

The Accuracy metric recognizes the fraction of predictions that the model got correctly. It is defined by the following formula where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Figure 5: The formula of accuracy.

D.2 Precision

The Precision metric recognizes the proportion of positive identifications that was actually correct. It is defined by the following formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Figure 6: The formula of precision.

D.3 Recall

The Recall metric recognizes the proportion of actual positives that were identified correctly. It is defined by the following formula:

$$\text{Recall} = \frac{TP}{TP + FN}$$

Figure 7: The formula of recall.

D.4 Receiver Operating Characteristic (ROC) Curve

The ROC curve plots two parameters which are the True Positive Rate (TPR) and False Positive Rate (FPR). It shows the relationship between sensitivity and specificity for every test.

D.5 Area Under the ROC Curve (AUC)

The AUC measures the ability of the classifier to differentiate between classes. Additionally, it is used as a summary of the ROC curve. A higher AUC indicates better predictive performance in correctly identifying TP and TN. It measures the overall area beneath the ROC curve. Moreover, AUC is unaffected by changes in scale or threshold.

IV. Design and Implementation

A. Dataset

The dataset of this study focuses on gram-negative bacteria including *Acinetobacter baumannii*, *Escherichia coli*, and *Klebsiella pneumoniae*. According to the Philippine DOH [16], there is a significant increase in resistance among these three bacteria. Hence, in this study, the three bacteria are selected. Additionally, *A. baumannii* and *K. pneumoniae* are part of ESKAPE pathogens that exhibit multidrug resistance. The acronym ESKAPE stands for *Enterococcus faecium*, *Staphylococcus aureus*, *Klebsiella pneumoniae*, *Acinetobacter baumannii*, *Pseudomonas aeruginosa*, and *Enterobacter* species. The antibiotics chosen in this study are ciprofloxacin, cefotaxime, ceftriaxone, gentamicin, and levofloxacin. These antibiotics are the common drugs used to treat infections caused by three bacteria. Moreover, it is reported that ceftriaxone and ciprofloxacin have an increasing resistance rate of 41% and 10.3%, respectively.

The isolates of the three bacteria are publicly available and are retrieved from the National Center for Biotechnology Information (NCBI) website. These isolates were obtained from human clinical specimens, environmental, food, and other samples where AST was mostly performed using Vitek, Microscan, and Phoenix.

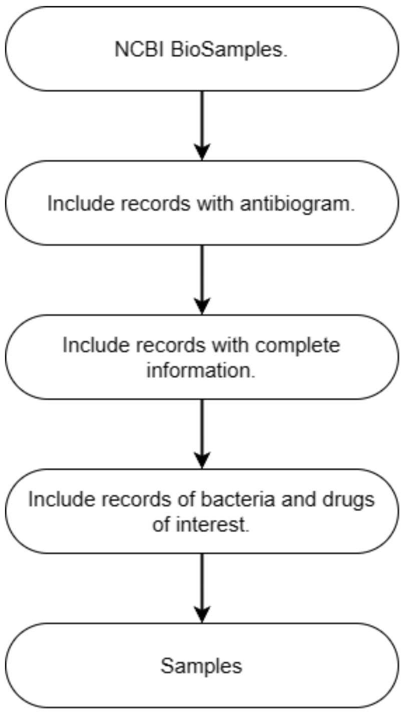

NCBI BioSamples.
Include records with antibiogram.
Include records with complete information.
Include records of bacteria and drugs of interest.
Samples

Figure 8: Data collection.

Figure 8 shows the data collection process where bacterial samples are retrieved from the NCBI BioSample database. The study includes only samples with complete antibiogram records. Records with incomplete information, such as missing the antibiotics of interest, are excluded. Subsequently, the list undergoes further refinement to narrow down to specific bacteria and antibiotics of interest, resulting in a total of 200 samples. The sample counts per species are as follows: *A. baumannii* - 90, *E. coli* - 65, and *K. pneumoniae* - 45. Refer to Table 1 for a summary of the samples used in building and evaluating the machine learning models. Some samples were not tested for each antibiotic, so the total does not add up to 200. Additionally, all “intermediate” antibiotic resistance was converted to “resistant” to project data to a binary classification problem.

Table 1: Summary of the samples.

Antibiotic	**Resistant**	**Susceptible**	**Total**
Cefotaxime	106	21	127
Ceftriaxone	148	40	188
Ciprofloxacin	99	101	200
Gentamicin	87	113	200
Levofloxacin	81	106	187

B. Data Preprocessing

A total of 200 samples were subjected to integron identification using the IntegronFinder software tool to detect the presence of integrons. Additionally, AMR gene detection was performed using the ABRicate tool. The resulting information obtained from IntegronFinder and ABRicate was utilized as predictors or features in the analysis. Meanwhile, the AMR data obtained from the antibiogram served as the response or target variable, with a binary label (0 or 1) indicating whether the bacteria exhibited resistance to a specific antibiotic.

To prepare the data for analysis, a MultiLabelBinarizer from the sklearn library was applied to the column of genes. This transformation enabled handling samples that contained multiple AMR genes, ensuring that the data were appropriately encoded for subsequent modeling.

To assess the performance and generalizability of the developed models, the dataset was split into training and testing sets using a 70:30 ratio. This division allowed for training the models on a substantial portion of the data while preserving an independent subset for evaluation purposes.

Addressing the issue of data imbalance, particular attention was given to balancing the number of susceptible and resistant isolates for each antibiotic. To achieve this, the Synthetic Minority Oversampling TEchnique (SMOTE) was employed. SMOTE is a widely-used technique that generates synthetic samples of the minority class to increase the representation and balance within the dataset. By applying SMOTE, the analysis aimed to mitigate potential biases caused by imbalanced data distribution.

It is worth noting that due to the nature of the dataset, not all samples were tested for every antibiotic. Consequently, rows with missing target variables (e.g., antibiogram data) were excluded from the analysis to ensure the reliability and validity of the results obtained. This data preprocessing step allowed for a focused and accurate assessment of the association between integrons, specific AMR genes, and resistance to antibiotics.

C. Model Implementation

The study employed three machine learning models: XGBoost, SVM, and RF, both with and without the application of SMOTE. This approach aimed to compare the performance of the models under different conditions. Each antibiotic was trained using all three models.

The input data for the machine learning models consisted of the results obtained from the IntegronFinder tool, indicating the presence or absence of integrons, and the list of AMR genes detected by ABRicate, also indicating their presence or absence.

Following data preprocessing and the division of the dataset into training and testing sets, the samples were fed into the machine learning models. The models outputted a binomial classification, labeling the samples as either resistant (= 1) or susceptible (= 0) to the antibiotic of interest. The models were trained using their default parameters.

Once the models were constructed, feature importance analysis was conducted on the RF and XGBoost models to determine the individual impact of each feature in the decision-making process. Specifically, the analysis identified the most important

AMR gene and the presence of integrons for each model.

To evaluate the performance of the models, various statistical measures including the ROC curve, AUC score, and classification measures were utilized. These measures provided insights into the predictive accuracy and effectiveness of the models. Based on the results of the evaluation, the models with the highest AUC score were selected for incorporation into a web-based application.

Furthermore, the study investigated the association between integrons and AMR genes to facilitate additional discussions and insights into their relationship. The comprehensive workflow of the study is depicted below:

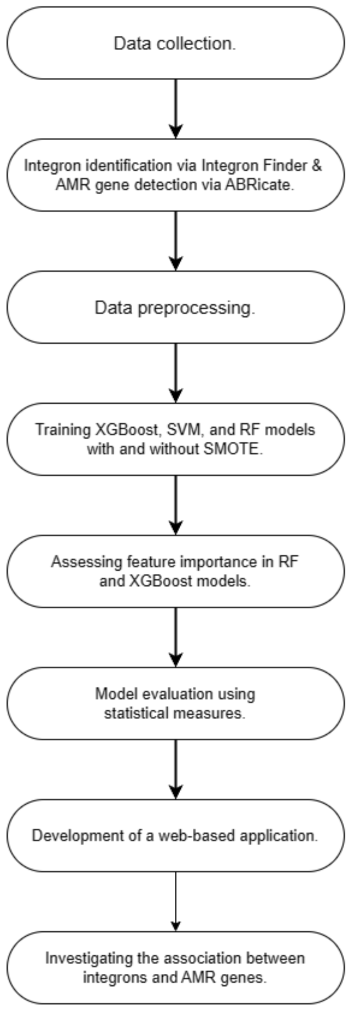

Data collection.
Integron identification via Integron Finder & AMR gene detection via ABRicate.
Data preprocessing.
Training XGBoost, SVM, and RF models with and without SMOTE.
Assessing feature importance in RF and XGBoost models.
Model evaluation using statistical measures.
Development of a web-based application.
Investigating the association between integrons and AMR genes.

Figure 9: Workflow of the study.

D. Use Cases

The system's functionalities are primarily tailored to health professionals and researchers. They can easily input WGS data in FASTA format to obtain the predicted antibiogram results as well as the outputs from IntegronFinder and ABRicate. Furthermore, the system offers the option to use demo samples, enabling users to study pathogens and gain insight into the system's workings.

Additionally, users have access to detailed information about the machine learning model, including performance summaries, statistical measures, and confusion matrices. This comprehensive overview of the system functionalities is illustrated in the figure below.

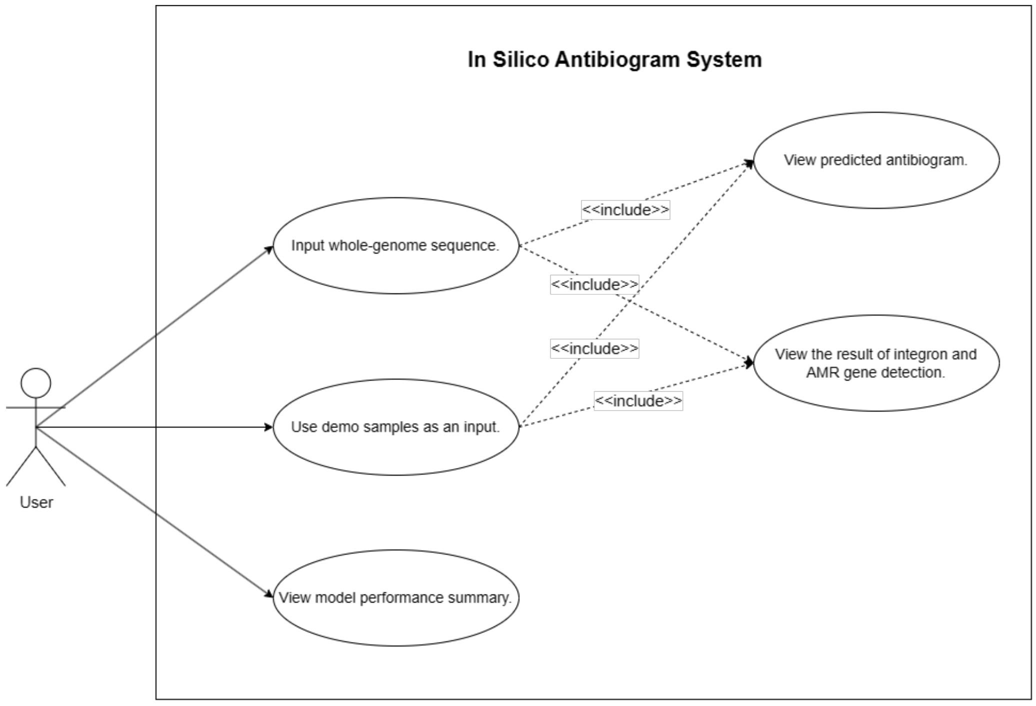

In Silico Antibiogram System
View predicted antibiogram.
<<include>>
Input whole-genome sequence.
<<include>>
<<include>>
View the result of integron and AMR gene detection.
<<include>>
Use demo samples as an input.
User
View model performance summary.

Figure 10: Use case diagram for health professional / researcher.

E. System Architecture

The input and output of the system is defined on the context diagram below:

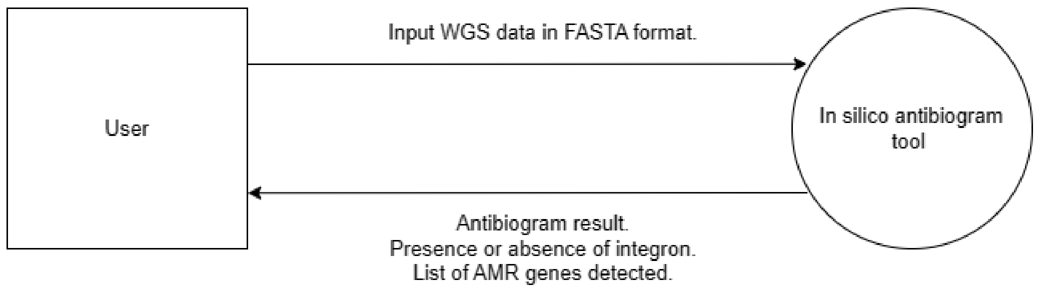

Input WGS data in FASTA format.
User
In silico antibiogram tool
Antibiogram result.
Presence or absence of integron.
List of AMR genes detected.

Figure 11: I/O Context Diagram

The web application is designed to facilitate user input of WGS data in FASTA format. Once the user submits the data, it is sent to a web server for processing. The web server then passes the data to IntegronFinder and ABRicate and uses the output from both tools as input to a machine learning model, which generates the predicted antibiogram result. In addition to the antibiogram, the results from IntegronFinder and ABRicate are also returned. These outcomes are presented to the user through the web application's user interface.

The web application was developed using the Python Django Framework, providing a robust and user-friendly environment for data input and result display. The implementation of the antibiogram tool and analytics was carried out using the sklearn library in Python.

F. Technical Architecture

The system is built using the following components:

1. Python, Django Framework

2. Ubuntu Operating System

3. 6 GB RAM

4. 2.00 GHz CPU

The minimum system requirements for the tool to run smoothly is as follows:

1. At least 4 GB RAM

2. AMD A10/Intel I3 or better

3. Windows/Linux/Mac Operating System

4. Web browser

V. Results

A. Exploratory Data Analysis

Upon analyzing all 200 samples using the IntegronFinder tool, it was found that 88 samples contained integrons, while 112 samples did not. The table below provides a breakdown and overview of the distribution of integrons across different antibiotics and their association with resistance or susceptibility to those antibiotics.

Table 2: Summary of the samples with respect to integrons.

	Resistant		**Susceptible**	
Antibiotic	**With Integron**	**Without Integron**	**With Integron**	**Without Integron**
Cefotaxime	41	65	4	17
Ceftriaxone	74	74	10	30
Ciprofloxacin	60	39	28	73
Gentamicin	55	32	33	80
Levofloxacin	51	30	33	73

Table 2 indicates that bacteria are more likely to be resistant to antibiotics such as Ciprofloxacin, Gentamicin, and Levofloxacin when they possess integrons. This suggests that the presence of integrons in bacteria may contribute to their resistance to these specific antibiotics. Moreover, these antibiotics show a higher proportion of susceptible cases in the ”Without Integron” category compared to the ”With Integron” category. This suggests that the absence of integrons in bacteria may indicate a higher likelihood of susceptibility to these antibiotics.

On the other hand, running the 200 samples on the ABRicate tool, a total of 150 AMR genes were detected and it is listed in table 3. Among the detected AMR genes, some were found to be more prevalent than others. The top 10 frequently identified AMR genes include *blaADC-25* (86 occurrences), *mdf(A)* (65), *sul2* (57), *sul1* (56), *aph(3")-Ib* (49), *aph(6)-Id* (47), *blaCTX-M-15* (41), *oqxA* (40), *oqxB* (39), and *ant(3")-Ia* (32).

Among the AMR genes detected when integrons are present, the top 10 most frequently found genes, along with their corresponding counts, include: *sul1* (55 counts), *aph(3")-Ib* (35), *blaCTX-M-15* (34), *aph(6)-Id* (33), *ant(3")-Ia* (30), *blaADC-25* (29), *sul2* (29), *mdf(A)* (28), *tet(A)* (26), and *oqxA* (25). These genes exhibit the highest occurrence when integrons are present, based on cross-tabulation between integron presence and each AMR gene.

Table 3: List of detected AMR genes.

ARR-3	blaOXA-120	blaSHV-76
aac(3)-IIa	blaOXA-121	blaTEM-141
aac(3)-IId	blaOXA-124	blaTEM-1A
aac(3)-IVa	blaOXA-126	blaTEM-1B
aac(3)-Ia	blaOXA-181	blaTEM-1D
aac(6')-IIa	blaOXA-2	blaVIM-1
aac(6')-Iaf	blaOXA-203	catA1
aac(6')-Ian	blaOXA-208	catA2
aac(6')-Ib	blaOXA-217	catB2
aac(6')-Ib-cr	blaOXA-23	catB3
aadA	blaOXA-235	catB8
aadA1	blaOXA-259	cmlA1
aadA16	blaOXA-260	dfrA1
aadA17	blaOXA-314	dfrA12
aadA2	blaOXA-343	dfrA14
aadA5	blaOXA-407	dfrA17
aadA6	blaOXA-413	dfrA26
ant(2")-Ia	blaOXA-430	dfrA27
ant(3")-Ia	blaOXA-510	dfrA29
aph(3")-Ib	blaOXA-528	dfrA8
aph(3')-Ia	blaOXA-529	ere(A)
aph(3')-VI	blaOXA-531	erm(42)
aph(3')-VIa	blaOXA-558	erm(B)
aph(3')-XV	blaOXA-58	floR
aph(4)-Ia	blaOXA-64	fosA
aph(6)-Id	blaOXA-65	fosA5
armA	blaOXA-66	fosA6
blaADC-25	blaOXA-69	fosA7
blaCARB-10	blaOXA-72	mdf(A)
blaCARB-16	blaOXA-88	mph(A)
blaCMY-2	blaOXA-9	mph(E)
blaCMY-4	blaOXA-90	msr(E)
blaCMY-42	blaOXA-91	oqxA
blaCMY-6	blaOXA-94	oqxB
blaCMY-94	blaOXA-98	qnrB1
blaCTX-M-1	blaPER-1	qnrB19
blaCTX-M-139	blaSHV-101	qnrB2
blaCTX-M-14	blaSHV-106	qnrB4
blaCTX-M-15	blaSHV-108	qnrB9
blaCTX-M-27	blaSHV-11	qnrS1
blaCTX-M-3	blaSHV-110	rmtB
blaCTX-M-55	blaSHV-12	rmtC
blaIMP-4	blaSHV-14	rmtF
blaKPC-2	blaSHV-145	sul1
blaKPC-3	blaSHV-182	sul2
blaNDM-1	blaSHV-187	sul3
blaOXA-1	blaSHV-26	tet(39)
blaOXA-10	blaSHV-27	tet(A)
blaOXA-100	blaSHV-30	tet(B)
blaOXA-106	blaSHV-33	tet(G)

B. Model Performance

This study focuses on comparing different models using AUC as the main metric, which is considered more reliable than accuracy.

Among the three antibiotics—cefotaxime, ceftriaxone, and ciprofloxacin—RF performed better when SMOTE was applied, as these antibiotics had data imbalance issues. In contrast, Gentamicin and Levofloxacin, which did not suffer from data imbalance, performed better with XGBoost and RF, respectively, without applying SMOTE.

All models obtained an AUC score of 0.86 or higher, that is considered excellent. This indicates that the models accurately predict and effectively separate the two classes (resistant vs. susceptible). The high AUC score reflects the models' strong discriminatory power and overall performance.

Based on the high AUC score, the models chosen for implementation on the web application were XGBoost without SMOTE for Gentamicin, RF without SMOTE for Levofloxacin, and RF with SMOTE for the other three antibiotics.

Table 4: XGBoost perfomance statistic without SMOTE.

Without SMOTE	**XGBoost**			
Antibiotic	**Accuracy**	**Recall**	**Precision**	**AUC**
Cefotaxime	0.85	0.94	0.89	0.57
Ceftriaxone	0.81	0.86	0.88	0.75
Ciprofloxacin	0.82	0.80	0.83	0.82
Gentamicin	**0.92**	**0.92**	**0.89**	**0.92**
Levofloxacin	0.82	0.76	0.83	0.82

Table 5: SVM perfomance statistic without SMOTE.

Without SMOTE	**SVM**			
Antibiotic	**Accuracy**	**Recall**	**Precision**	**AUC**
Cefotaxime	0.87	1.00	0.87	0.50
Ceftriaxone	0.81	0.91	0.85	0.66
Ciprofloxacin	0.85	0.81	0.85	0.85
Gentamicin	0.90	0.96	0.84	0.90
Levofloxacin	0.84	0.87	0.84	0.84

Table 6: RF perfomance statistic without SMOTE.

Without SMOTE	**RF**			
Antibiotic	**Accuracy**	**Recall**	**Precision**	**AUC**
Cefotaxime	0.87	1.00	0.87	0.50
Ceftriaxone	0.86	0.95	0.88	0.75
Ciprofloxacin	0.85	0.90	0.81	0.85
Gentamicin	0.92	0.88	0.92	0.91
Levofloxacin	**0.91**	**0.96**	**0.85**	**0.92**

Table 7: XGBoost perfomance statistic with SMOTE.

With SMOTE	**XGBoost**			
Antibiotic	**Accuracy**	**Recall**	**Precision**	**AUC**
Cefotaxime	0.77	0.74	1.00	0.87
Ceftriaxone	0.89	0.91	0.95	0.88
Ciprofloxacin	0.83	0.83	0.83	0.83
Gentamicin	0.92	0.92	0.89	0.92
Levofloxacin	0.86	0.87	0.80	0.86

Table 8: SVM perfomance statistic with SMOTE.

With SMOTE	**SVM**			
Antibiotic	**Accuracy**	**Recall**	**Precision**	**AUC**
Cefotaxime	0.87	0.85	1.00	0.93
Ceftriaxone	0.89	0.89	0.98	0.90
Ciprofloxacin	0.83	0.78	0.84	0.83
Gentamicin	0.92	0.88	0.92	0.91
Levofloxacin	0.84	0.96	0.73	0.86

Table 9: RF perfomance statistic with SMOTE.

With SMOTE	**RF**			
Antibiotic	**Accuracy**	**Recall**	**Precision**	**AUC**
Cefotaxime	**0.87**	**0.85**	**1.00**	**0.93**
Ceftriaxone	**0.93**	**0.95**	**0.95**	**0.91**
Ciprofloxacin	**0.87**	**0.81**	**0.88**	**0.86**
Gentamicin	0.90	0.92	0.86	0.90
Levofloxacin	0.88	0.93	0.85	0.87

C. Feature Importance

Given that all models showed better performance with decision tree algorithms, this study utilized the built-in feature importance functionality of XGBoost and RF to identify the key predictive features for determining drug resistance in the models.

Regarding cefotaxime, the AMR gene *mdf(A)* obtained the highest score, followed by *ant(2")-Ia*, *blaOXA-100*, *sul2*, and *oqxA*. In particular, the presence of integron ranked 11th among the important features.

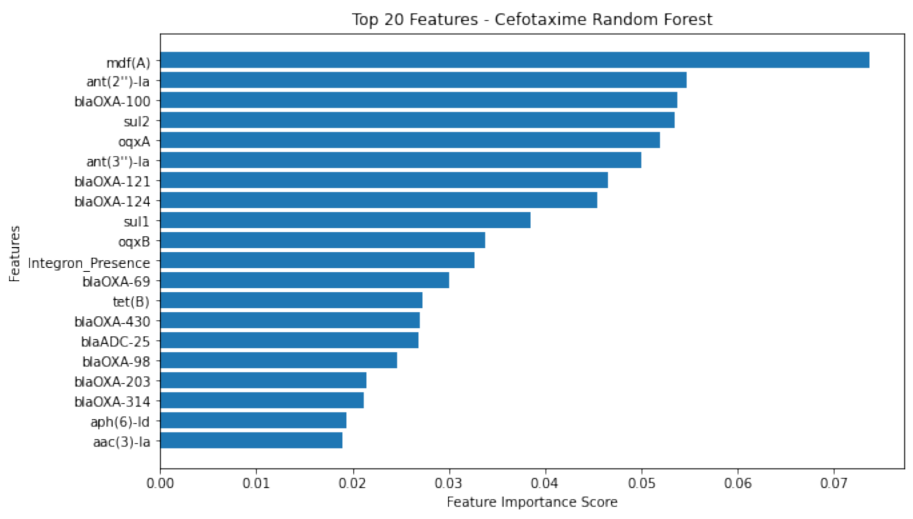


Figure 12: Top 20 most important features for Cefotaxime

For ceftriaxone, the AMR gene *ant(2")-Ia* received the highest score, followed by *sul2*, *mdf(A)*, *blaCMY-2*, and *blaCTX-M-15*. The presence of integron held the 14th position in terms of feature importance.

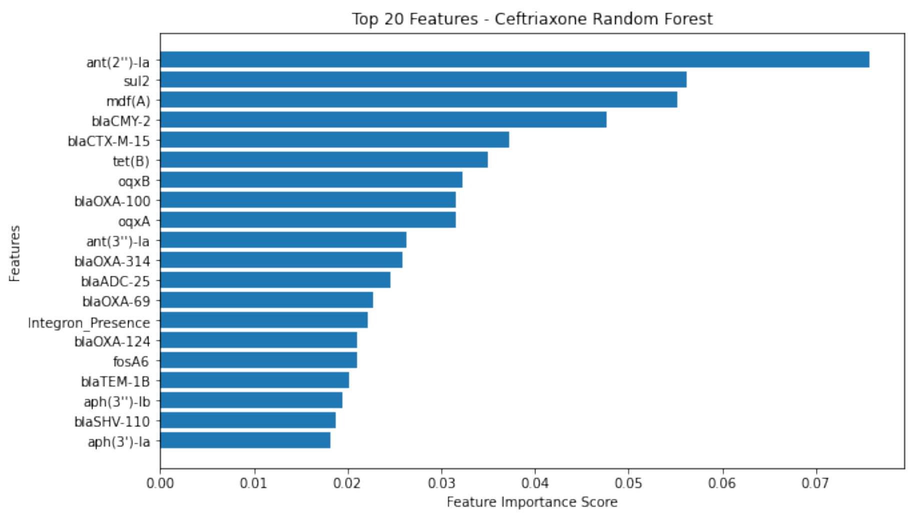


Figure 13: Top 20 most important features for Ceftriaxone

For ciprofloxacin, the AMR gene *blaCMY-2* achieved the highest score, followed by *mdf(A)*, *sul1*, *catA1*, and *ant(3")-Ia*. Meanwhile, the presence of integron was ranked 7th among the important features.

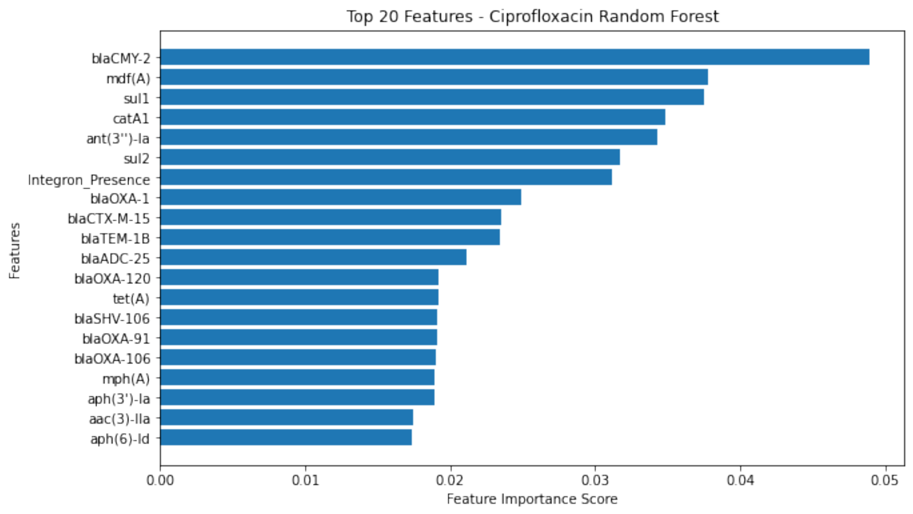


Figure 14: Top 20 most important features for Ciprofloxacin

In the case of gentamicin, the AMR gene *ant(2")-Ia* had the highest score, followed by *aac(3)-Ia*, *aadA2*, *aac(3)-IId*, and *aac(3)-IIa*. The presence of integron was ranked 18th in terms of feature importance.

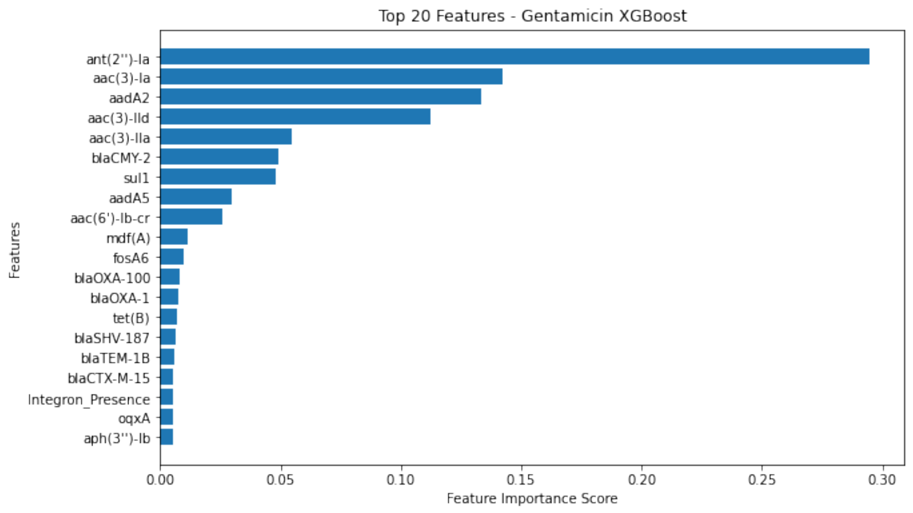


Figure 15: Top 20 most important features for Gentamicinn

Lastly, for levofloxacin, the AMR gene *tet(B)* obtained the highest score, followed by *mdf(A)*, *sul1*, *sul2*, *blaADC-25*, and *blaCMY-2*. The presence of integron held the 10th position among the important features.

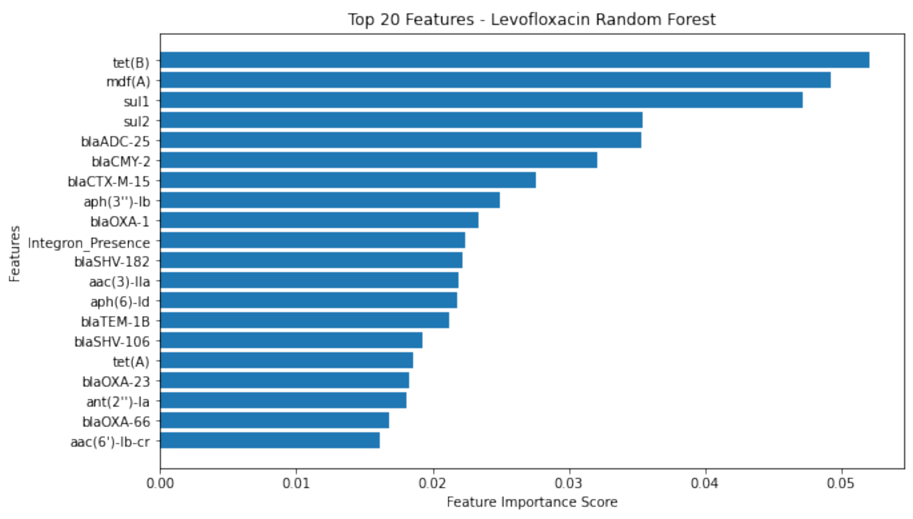


Figure 16: Top 20 most important features for Levofloxacin

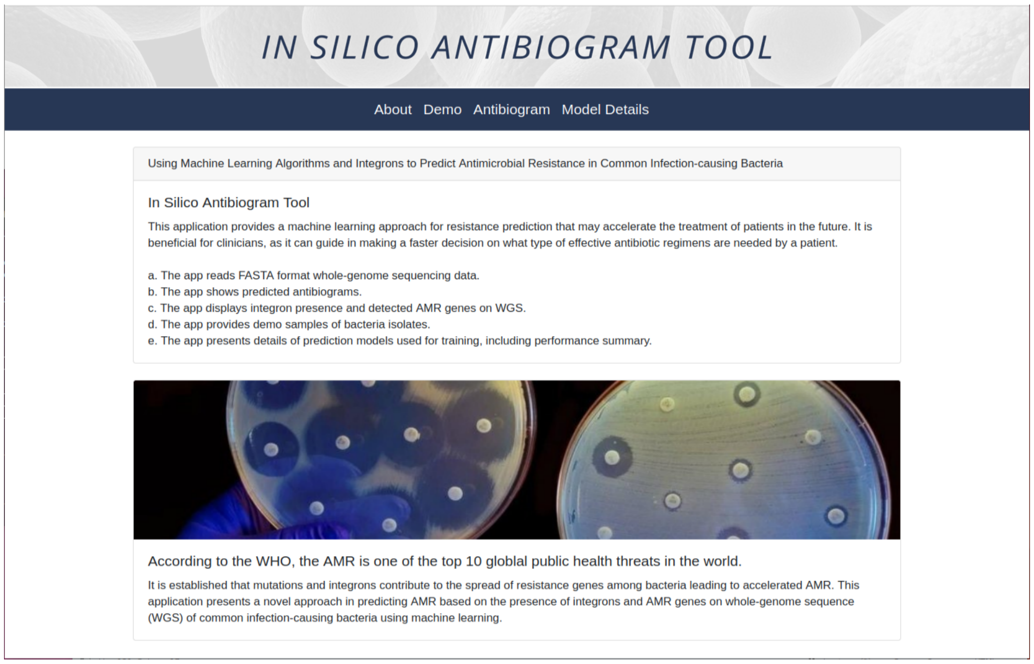
Overall, the feature importance analysis consistently ranks the integron feature below the top 5, suggesting that other features have a stronger influence on the model's predictions. However, it's significant to consider that the absence of high feature importance doesn't necessarily imply that the integron feature is not important or unrelated to AMR.

While feature importance analysis is a commonly used approach to assess the relevance of features, it may not fully capture the complexity of the integron-AMR relationship. Moreover, it is essential to recognize the fundamental role of integrons in horizontal gene transfer, where integrons facilitate the acquisition and dissemination of resistance genes, enabling bacteria to adapt and develop AMR.

D. Web Application

The web application comprises four main pages: About, Demo, Antibiogram, and Model Details. These pages can be accessed through the navigation bar located at the top of the application's interface. The application begins with the About page, which provides an overview of its capabilities and the information users can obtain.

Refer to figure 17 for a visual representation of the About page.


IN SILICO ANTIBIOGRAM TOOL
About Demo Antibiogram Model Details

Using Machine Learning Algorithms and Integrons to Predict Antimicrobial Resistance in Common Infection-causing Bacteria

In Silico Antibiogram Tool

This application provides a machine learning approach for resistance prediction that may accelerate the treatment of patients in the future. It is beneficial for clinicians, as it can guide in making a faster decision on what type of effective antibiotic regimens are needed by a patient.

a. The app reads FASTA format whole-genome sequencing data.
b. The app shows predicted antibiograms.
c. The app displays integron presence and detected AMR genes on WGS.
d. The app provides demo samples of bacteria isolates.
e. The app presents details of prediction models used for training, including performance summary.

According to the WHO, the AMR is one of the top 10 globlal public health threats in the world.

It is established that mutations and integrons contribute to the spread of resistance genes among bacteria leading to accelerated AMR. This application presents a novel approach in predicting AMR based on the presence of integrons and AMR genes on whole-genome sequence (WGS) of common infection-causing bacteria using machine learning.

Figure 17: About page/home page.

On the Demo page, users can find a drop-down menu that allows them to select from a range of available demo samples. When a user chooses a sample, the predicted antibiogram panel displays the resistance or susceptibility results for each antibiotic. Additionally, the integron and AMR genes panel provide information detected by the IntegronFinder and ABRicate tools. Figure 19 illustrates this functionality.

The Antibiogram page shares a similar layout to the Demo page. However, it enables users to upload a FASTA file from their local machine and predict the resistance information associated with it. Furthermore, the values outputted in the Integron and AMR genes panels serve as inputs for the machine learning models. Examples of the in silico antibiogram tool's predictions can be seen in figure 21 and 22.

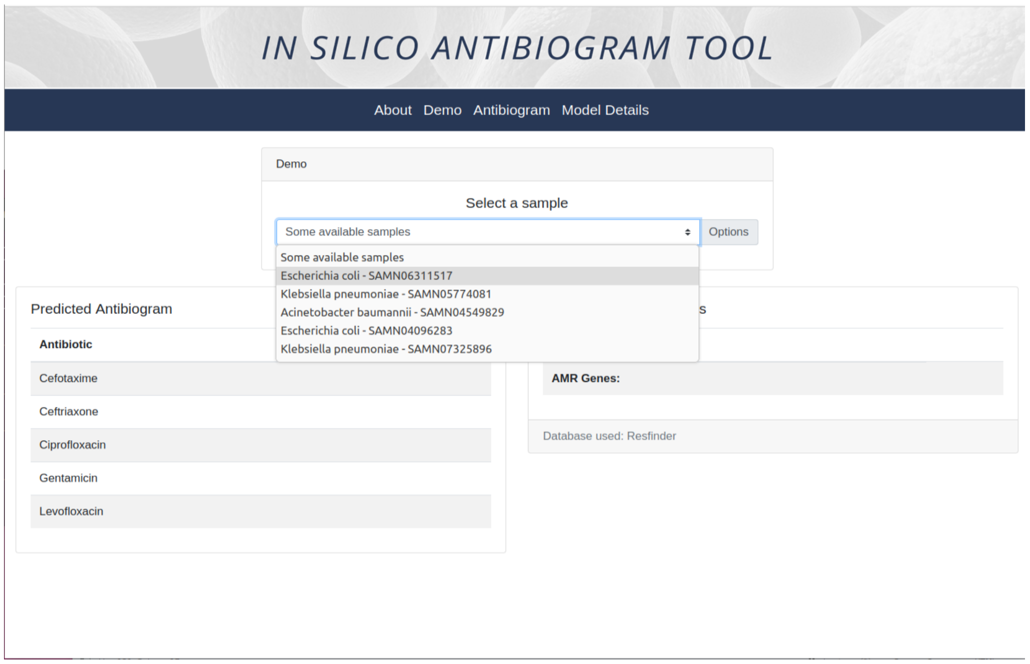
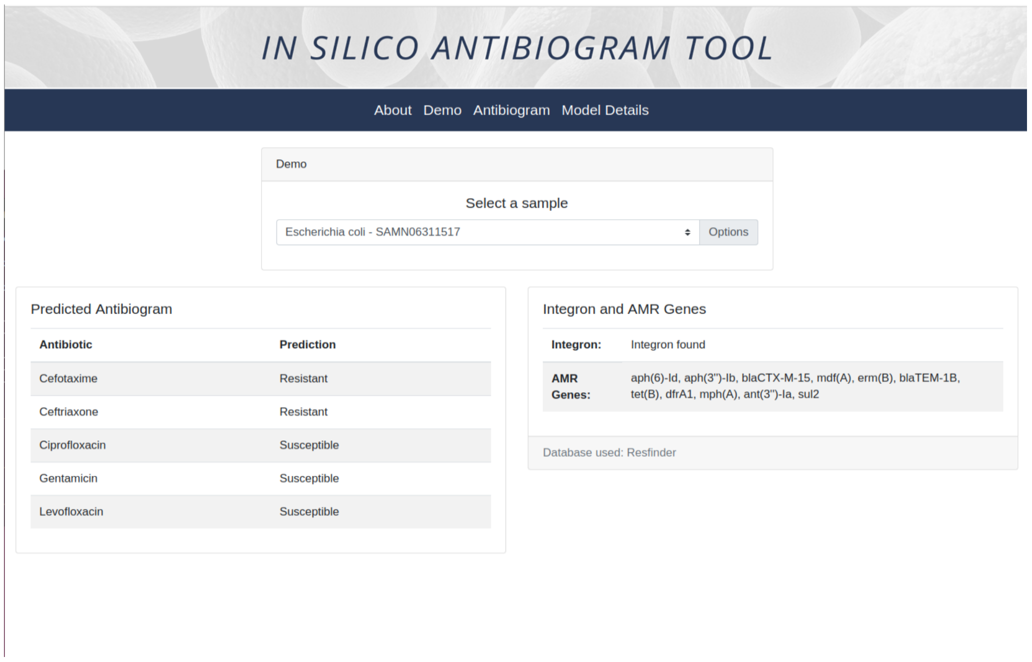

IN SILICO ANTIBIOGRAM TOOL
About Demo Antibiogram Model Details
Demo
Select a sample
Some available samples
Options
Some available samples
Escherichia coli - SAMN06311517
Klebsiella pneumoniae - SAMN05774081
Acinetobacter baumannii - SAMN04549829
Escherichia coli - SAMN04096283
Klebsiella pneumoniae - SAMN07325896
Predicted Antibiogram
Antibiotic
Cefotaxime
Ceftriaxone
Ciprofloxacin
Gentamicin
Levofloxacin
AMR Genes:
Database used: Resfinder

Figure 18: Demo page drop-down menu.


IN SILICO ANTIBIOGRAM TOOL
About Demo Antibiogram Model Details
Demo
Select a sample
Escherichia coli - SAMN06311517
Options

Predicted Antibiogram

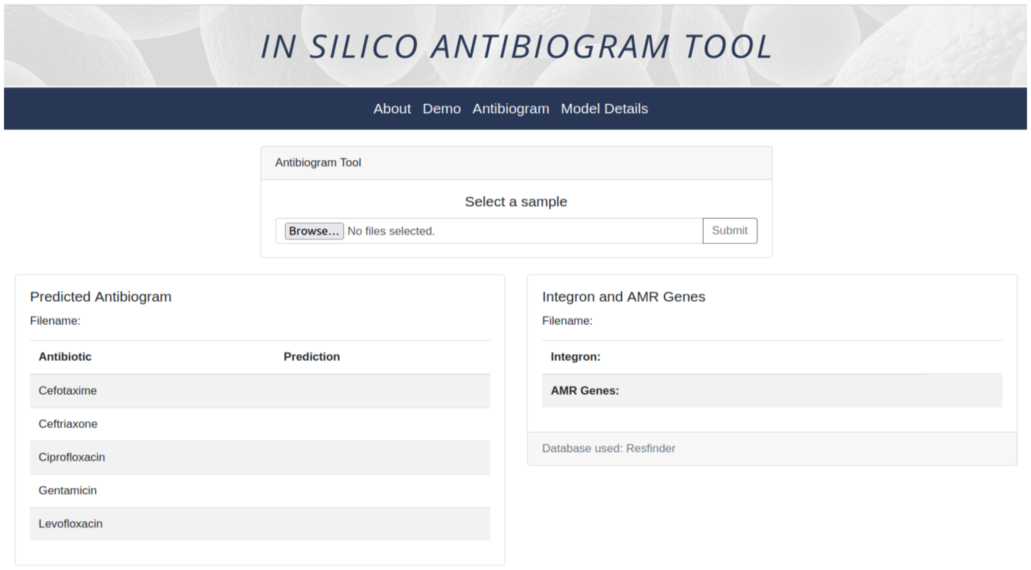
Antibiotic	Prediction
Cefotaxime	Resistant
Ceftriaxone	Resistant
Ciprofloxacin	Susceptible
Gentamicin	Susceptible
Levofloxacin	Susceptible

Integron and AMR Genes

Integron:	Integron found
AMR Genes:	aph(6)-Id, aph(3")-Ib, blaCTX-M-15, mdf(A), erm(B), blaTEM-1B, tet(B), dfrA1, mph(A), ant(3")-Ia, sul2

Database used: Resfinder

Figure 19: Predicting demo samples (*E. coli*).



IN SILICO ANTIBIOGRAM TOOL

About Demo Antibiogram Model Details

Antibiogram Tool

Select a sample

Browse... No files selected. Submit

Predicted Antibiogram

Filename:

Antibiotic	Prediction
Cefotaxime	
Ceftriaxone	
Ciprofloxacin	
Gentamicin	
Levofloxacin	

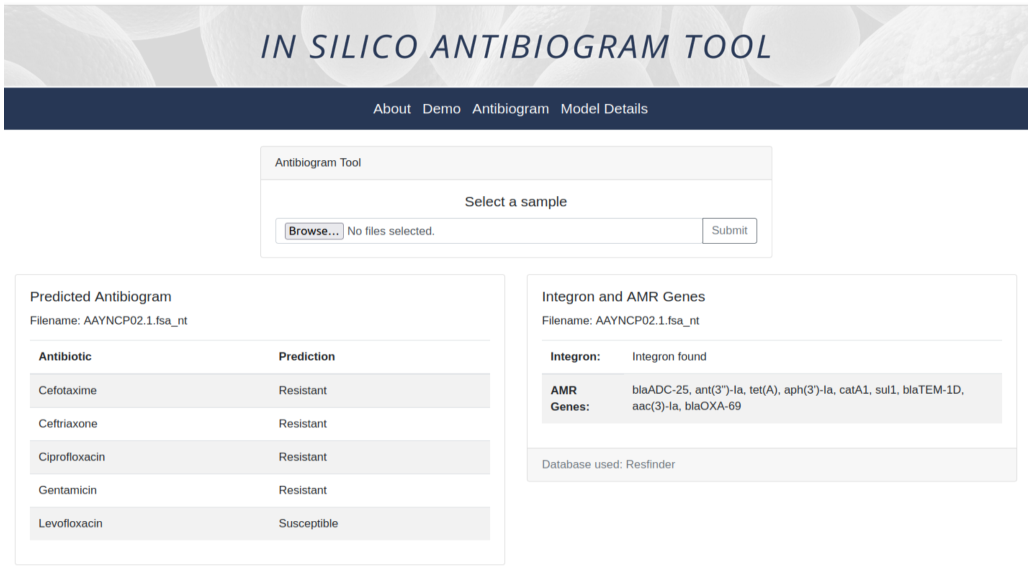
Integron and AMR Genes

Filename:

Integron:	
AMR Genes:	

Database used: Resfinder

Figure 20: Antibiogram page layout.



IN SILICO ANTIBIOGRAM TOOL

About Demo Antibiogram Model Details

Antibiogram Tool

Select a sample

Browse... No files selected. Submit

Predicted Antibiogram

Filename: AAYNCP02.1.fsa_nt

Antibiotic	Prediction
Cefotaxime	Resistant
Ceftriaxone	Resistant
Ciprofloxacin	Resistant
Gentamicin	Resistant
Levofloxacin	Susceptible

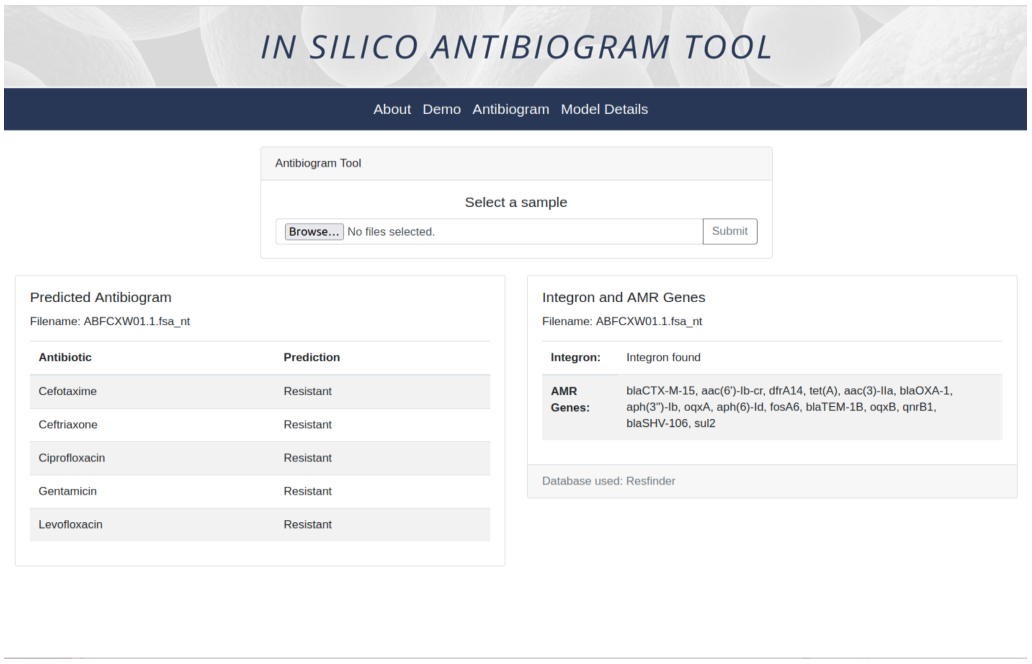
Integron and AMR Genes

Filename: AAYNCP02.1.fsa_nt

Integron:	Integron found
AMR Genes:	blaADC-25, ant(3")-Ia, tet(A), aph(3')-Ia, catA1, sul1, blaTEM-1D, aac(3)-Ia, blaOXA-69

Database used: Resfinder

Figure 21: Predicting a user-input file (*A. baumannii* strain).



Predicted Antibiogram

Filename: ABFCXW01.1.fsa_nt

Antibiotic	Prediction
Cefotaxime	Resistant
Ceftriaxone	Resistant
Ciprofloxacin	Resistant
Gentamicin	Resistant
Levofloxacin	Resistant

Integron and AMR Genes

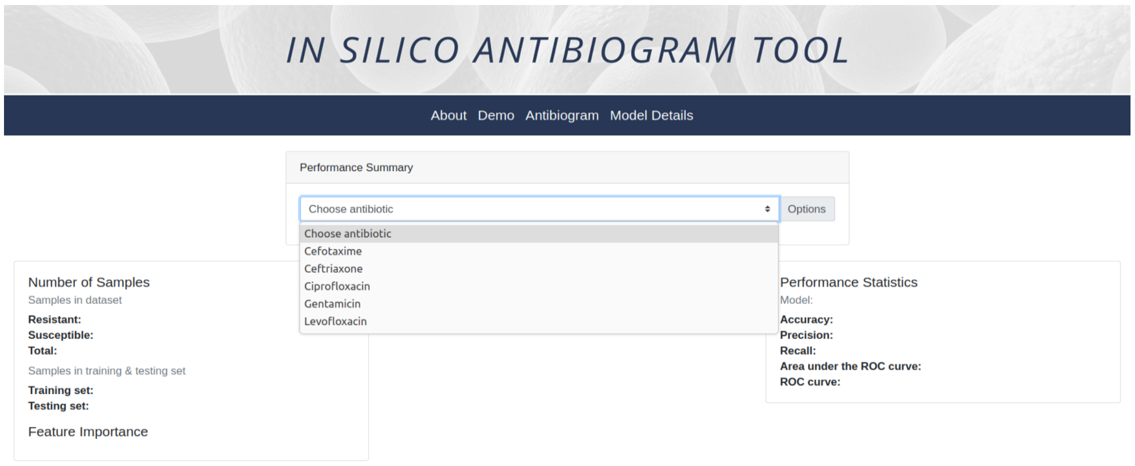
Filename: ABFCXW01.1.fsa_nt

Integron:	Integron found
AMR Genes:	blaCTX-M-15, aac(6')-Ib-cr, dfrA14, tet(A), aac(3)-IIa, blaOXA-1, aph(3")-Ib, oqxA, aph(6)-Id, fosA6, blaTEM-1B, oqxB, qnrB1, blaSHV-106, sul2

Database used: Resfinder

Figure 22: Predicting a user-input file (*K. pneumoniae* strain).

Lastly, the Model Details page features a drop-down menu for the five antibiotics studied. This page displays the number of samples, performance statistics, confusion matrices, and a feature importance section. By exploring this page, users can gain insights into which features are most important and understand the underlying relationships within the data. This allows users to comprehend how the model makes predictions. Figure 24 and 25 offer examples of utilizing the Model Details page.



IN SILICO ANTIBIOGRAM TOOL

About Demo Antibiogram Model Details

Performance Summary

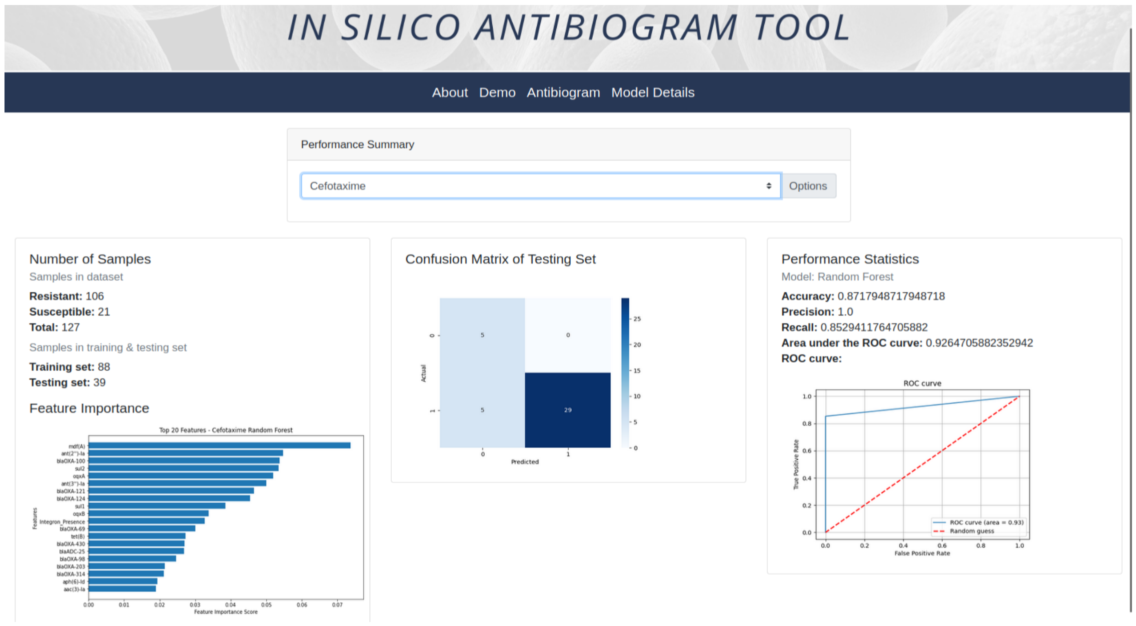
Choose antibiotic | Options

Choose antibiotic
Cefotaxime
Ceftriaxone
Ciprofloxacin
Gentamicin
Levofloxacin

Number of Samples
Samples in dataset
Resistant:
Susceptible:
Total:
Samples in training & testing set
Training set:
Testing set:
Feature Importance

Performance Statistics
Model:
Accuracy:
Precision:
Recall:
Area under the ROC curve:
ROC curve:

Figure 23: Model details page drop-down menu.



IN SILICO ANTIBIOGRAM TOOL

About Demo Antibiogram Model Details

Performance Summary

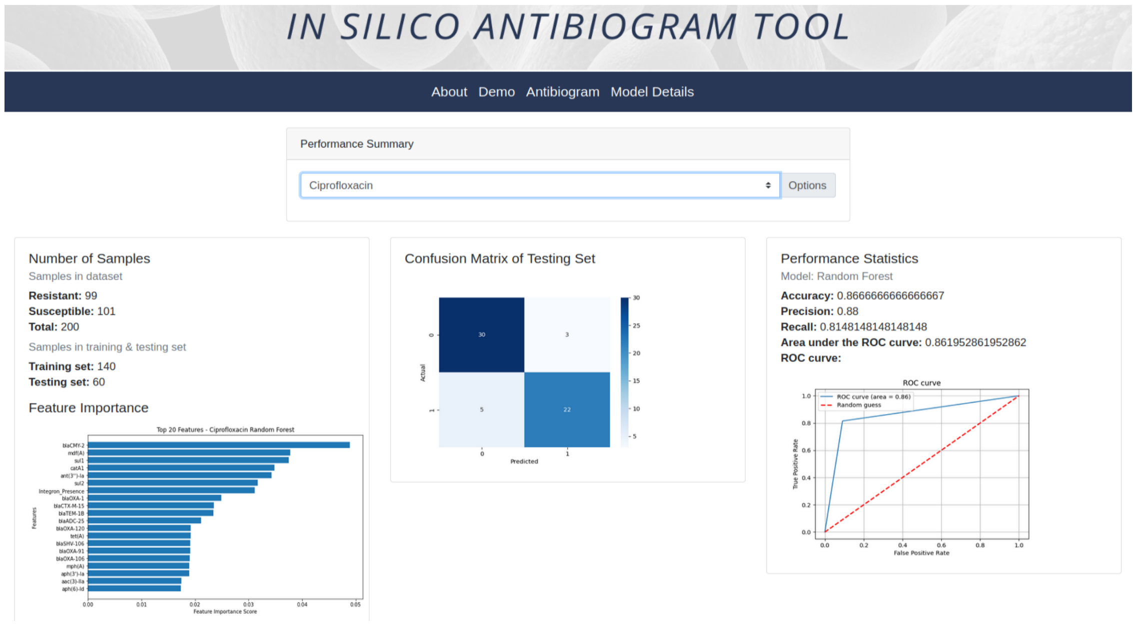
Cefotaxime | Options

Number of Samples
Samples in dataset
Resistant: 106
Susceptible: 21
Total: 127
Samples in training & testing set
Training set: 88
Testing set: 39
Feature Importance

Confusion Matrix of Testing Set

Performance Statistics
Model: Random Forest
Accuracy: 0.8717948717948718
Precision: 1.0
Recall: 0.8529411764705882
Area under the ROC curve: 0.9264705882352942
ROC curve:

Figure 24: Cefotaxime model details.



Performance Summary

Ciprofloxacin

Number of Samples

Samples in dataset

Resistant: 99
Susceptible: 101
Total: 200

Samples in training & testing set

Training set: 140
Testing set: 60

Feature Importance

Confusion Matrix of Testing Set

Performance Statistics

Model: Random Forest

Accuracy: 0.8666666666666667
Precision: 0.88
Recall: 0.8148148148148148
Area under the ROC curve: 0.861952861952862
ROC curve:

Figure 25: Ciprofloxacin model details.

VI. Discussions

This research aimed to evaluate the performance of three machine learning techniques (XGBoost, SVM, and RF) in predicting antibiotic resistance in *Acinetobacter baumannii*, *Escherichia coli*, and *Klebsiella pneumoniae* based on WGS data. A key objective was to incorporate integron as a predictive feature and develop a web-based application utilizing the machine learning model. This application aims to provide early detection of antimicrobial resistance in pathogens and support the prescription of more effective antibiotics.

Traditional methods such as culturing bacteria and obtaining a complete antibiogram typically take 2-4 days, whereas, currently, there is a continuous decrease in cost and increasing speed of high-throughput sequencing technologies, such as WGS, that can serve as an input to the web application that offers rapid testing of bacterial samples, enabling the generation of a preliminary antibiogram to guide initial antibiotic selection. Presenting the results as an early estimate of the antibiogram, rather than individual predictions alone, provides clinicians with a clear and intuitive tool for informing the initial antibiotic choice. This predictive approach could be particularly valuable in settings where resources are limited and a full microbiology laboratory is not feasible.

The results demonstrate that decision tree algorithms, specifically XGBoost and RF, not only outperformed SVM but also provided interpretability, making them valuable tools in predicting antibiotic resistance. By analyzing the built-in feature importance, this study successfully identified key AMR genes associated with resistance against specific antibiotics. Notably, the *mdf(A)* gene consistently exhibited high importance scores, persistently ranking within the top 3 for cefotaxime, ceftriaxone, ciprofloxacin, and levofloxacin. This gene, frequently found in *E. coli*, has been well-established as a multidrug-resistant gene [40].

Furthermore, based on exploratory data analysis, *sul1* and *sul2* is one of the most frequently found AMR genes when integrons are present, confirming previous studies that associate integrons with sulfonamide resistance genes [41][42]. These findings highlight the significance of integrons in conferring resistance and provide insights into the genetic mechanisms involved. Although the integron feature may not rank highly in terms of feature importance, it is essential to recognize the fundamental role of integrons in horizontal gene transfer. Their ability to acquire and disseminate resistance genes is well-established, contributing to the spread of AMR in bacterial populations [10]. Integrons can serve as reservoirs for resistance gene cassettes, allowing bacteria to rapidly acquire and express resistance to multiple antimicrobial agents. AMR is a multifaceted phenomenon influenced by various genetic and environmental factors. While integrons play a significant role, other features, such as specific resistance genes may have a more prominent impact on resistance patterns in the dataset.

In terms of clinical implications, understanding the specific AMR genes associated with resistance can guide clinicians in making informed decisions regarding the antibiotic selection and treatment strategies. The prominence of the *mdf(A)* gene suggests the need for targeted interventions and alternative treatment options in cases where this gene is present. Additionally, the identification of integrons as key features underscore the importance of monitoring and controlling their spread to mitigate antibiotic resistance.

VII. Conclusions

This study showcased the potential for whole-genome sequencing to provide early estimates of the antibiogram for gram-negative bacteria including *E. coli*, *A baumannii*, and *K. pneumoniae* when combined with machine learning. The developed models show good performance and robustness to class-imbalanced data despite being trained exclusively on the presence or absence of AMR genes and integrons. The web application gives a health care professional or researcher an easy way to understand predictions and direct the initial antibiotic decision before the laboratory data are available by showing the results as an antibiogram, shortening the time it takes to begin antibiotic treatment.

The application of decision tree algorithms, along with the identification of key AMR genes and the role of integrons, has shed light on the predictive capabilities and genetic determinants of antibiotic resistance. These findings offer valuable insights for clinical decision-making, emphasize the importance of monitoring integrons, and suggest avenues for future research in combating antibiotic resistance effectively.

VIII. Recommendations

To enhance the performance of the web application - In Silico Antibiogram Tool, several improvements can be implemented. Firstly, increasing the accuracy of the training model can be pursued by augmenting the number of samples per antibiotic in the training set. This larger and more diverse dataset will enable the model to capture a wider range of patterns and improve its predictive capabilities.

Moreover, based on the feature importance analysis conducted in this study, the next step could involve feature extraction techniques. These techniques aim to identify the most informative and relevant features that contribute significantly to the prediction of antibiotic resistance. By selecting and utilizing these essential features, the model's performance can potentially be enhanced even further.

Furthermore, expanding the scope of the study to include additional antibiotics would be valuable. Investigating the predictive performance of a wider range of antibiotics will allow for a more broad evaluation of the model's effectiveness.

Additionally, it is crucial to highlight the significance of integrons, as they are genetic elements that facilitate the acquisition and dissemination of antimicrobial resistance genes among bacteria. These elements can capture and incorporate gene cassettes, which contain resistance genes, into their genetic structure. Further research on integrons can shed light on their functional roles, their prevalence in different bacterial populations, environments, and their potential as targets for intervention strategies.

By implementing these recommendations, the In Silico Antibiogram Tool can be refined and offer improved accuracy and versatility in predicting antibiotic resistance.

IX. Bibliography

[1] B. Néron, E. Littner, M. Haudiquet, A. Perrin, J. Cury, and E. P. C. Rocha, "Integronfinder 2.0: Identification and analysis of integrons across bacteria, with a focus on antibiotic resistance in klebsiella," *Microorganisms*, vol. 10, no. 4, 2022.

[2] "Antimicrobial resistance." https://www.who.int/news-room/fact-sheets/detail/antimicrobial-resistance, Nov 2021.

[3] C. J. Murray, K. S. Ikuta, F. Sharara, L. Swetschinski, G. Robles Aguilar, A. Gray, C. Han, C. Bisignano, P. Rao, E. Wool, and et al., "Global burden of bacterial antimicrobial resistance in 2019: A systematic analysis," *The Lancet*, vol. 399, p. 629–655, Jan 2022.

[4] "New report calls for urgent action to avert antimicrobial resistance crisis." https://www.who.int/news/item/29-04-2019-new-report-calls-for-urgent-action-to-avert-antimicrobial-resistance-crisis, Apr 2019.

[5] M. L. Bayot and B. N. Bragg, "Antimicrobial susceptibility testing." https://www.ncbi.nlm.nih.gov/books/NBK539714/, Oct 2022.

[6] "Antimicrobial susceptibility testing." https://www.tmcc.edu/microbiology-resource-center/lab-protocols/antimicrobial-susceptibility-testing.

[7] J. Lv, S. Deng, and L. Zhang, "A review of artificial intelligence applications for antimicrobial resistance," *Biosafety and Health*, vol. 3, no. 1, pp. 22–31, 2021.

[8] J. Besser, H. Carleton, P. Gerner-Smidt, R. Lindsey, and E. Trees, "Next-generation sequencing technologies and their application to the study and con-

trol of bacterial infections," *Clinical Microbiology and Infection*, vol. 24, no. 4, p. 335–341, 2018.

[9] P. Sabbagh, M. Rajabnia, A. Maali, and E. Ferdosi-Shahandashti, "Integron and its role in antimicrobial resistance: A literature review on some bacterial pathogens," *Iranian journal of basic medical sciences*, vol. 24, p. 136–142, Feb 2021.

[10] C. Souque, J. A. Escudero, and R. C. MacLean, "Integron activity accelerates the evolution of antibiotic resistance," *eLife*, vol. 10, p. e62474, Feb 2021.

[11] Y. Ren, T. Chakraborty, S. Doijad, L. Falgenhauer, J. Falgenhauer, A. Goesmann, A.-C. Hauschild, O. Schwengers, and D. Heider, "Prediction of antimicrobial resistance based on whole-genome sequencing and machine learning," *Bioinformatics*, vol. 38, pp. 325–334, 10 2021.

[12] M. Nguyen, S. W. Long, P. F. McDermott, R. J. Olsen, R. Olson, R. L. Stevens, G. H. Tyson, S. Zhao, and J. J. Davis, "Using machine learning to predict antimicrobial mics and associated genomic features for nontyphoidal salmonella," *Journal of Clinical Microbiology*, vol. 57, no. 2, pp. e01260–18, 2019.

[13] L. Torres-Elizalde, D. Ortega-Paredes, K. Loaiza, E. Fernández-Moreira, and M. Larrea-Álvarez, "In silico detection of antimicrobial resistance integrons in salmonella enterica isolates from countries of the andean community," *Antibiotics*, vol. 10, no. 11, 2021.

[14] K. Loaiza, D. Ortega-Paredes, M. Johansson, A. Florensa, O. Lund, and V. Bortolaia, "Intfinder: a freely-available user-friendly web tool for detection of class 1 integrons in next-generation sequencing data using k-mer alignment," Apr 2020.

[15] A. van Belkum, T. T. Bachmann, G. Lüdke, J. G. Lisby, G. Kahlmeter, A. Mohess, K. Becker, J. P. Hays, N. Woodford, K. Mitsakakis, and et al., "Develop-

mental roadmap for antimicrobial susceptibility testing systems," *Nature Reviews Microbiology*, vol. 17, p. 51–62, Oct 2018.

[16] "The philippine action plan to combat antimicrobial resistance: One health approach." https://icamr.doh.gov.ph/action-plan-2/, Feb 2021.

[17] M. N. Anahtar, J. H. Yang, and S. Kanjilal, "Applications of machine learning to the problem of antimicrobial resistance: an emerging model for translational research," *Journal of Clinical Microbiology*, vol. 59, no. 7, pp. e01260–20, 2021.

[18] A. S. Chowdhury, D. R. Call, and S. L. Broschat, "Pargt: A software tool for predicting antimicrobial resistance in bacteria," *Scientific Reports*, vol. 10, Jul 2020.

[19] U. Neumann, N. Genze, and D. Heider, "Efs: An ensemble feature selection tool implemented as r-package and web-application," *BioData Mining*, vol. 10, Jun 2017.

[20] P. Cingolani, A. Platts, L. L. Wang, M. Coon, T. Nguyen, L. Wang, S. J. Land, X. Lu, and D. M. Ruden, "A program for annotating and predicting the effects of single nucleotide polymorphisms, snpeff," *Fly*, vol. 6, no. 2, pp. 80–92, 2012.

[21] P.-J. Van Camp, D. B. Haslam, and A. Porollo, "Bioinformatics approaches to the understanding of molecular mechanisms in antimicrobial resistance," *International Journal of Molecular Sciences*, vol. 21, no. 4, 2020.

[22] A. N. Zhang, L.-G. Li, L. Ma, M. R. Gillings, J. M. Tiedje, and T. Zhang, "Conserved phylogenetic distribution and limited antibiotic resistance of class 1 integrons revealed by assessing the bacterial genome and plasmid collection," *Microbiome*, vol. 6, Jul 2018.

[23] J. Oliveira and W. Reygaert, "Gram negative bacteria." https://www.ncbi.nlm.nih.gov/books/NBK538213/, Oct 2022.

[24] M.-F. Lin and C.-Y. Lan, "Antimicrobial resistance in acinetobacter baumannii: From bench to bedside," *World Journal of Clinical Cases*, vol. 2, p. 787, Dec 2014.

[25] R. Vazquez-Lopez, S. G. Solano-Galvez, J. J. Juarez Vignon-Whaley, J. A. Abello Vaamonde, L. A. Padro Alonzo, A. Rivera Resendiz, M. Muleiro Alvarez, E. N. Vega Lopez, G. Franyuti-Kelly, D. A. Alvarez-Hernandez, and et al., "Acinetobacter baumannii resistance: A real challenge for clinicians," *Antibiotics*, vol. 9, p. 205, Apr 2020.

[26] D. Wu, Y. Ding, K. Yao, W. Gao, and Y. Wang, "Antimicrobial resistance analysis of clinical escherichia coli isolates in neonatal ward," *Frontiers in Pediatrics*, vol. 9, May 2021.

[27] P. Aminul, S. Anwar, M. M. Molla, and M. R. Miah, "Evaluation of antibiotic resistance patterns in clinical isolates of klebsiella pneumoniae in bangladesh," *Biosafety and Health*, vol. 3, p. 301–306, Nov 2021.

[28] V. Ballen, Y. Gabasa, C. Ratia, R. Ortega, M. Tejero, and S. Soto, "Antibiotic resistance and virulence profiles of klebsiella pneumoniae strains isolated from different clinical sources," *Frontiers in Cellular and Infection Microbiology*, vol. 11, Sep 2021.

[29] C. Gabriele, "The importance of sequencing quality control (data qc)." https://www.fiosgenomics.com/the-importance-of-sequencing-data-control/, Mar 2022.

[30] A. Akalin, "Computational genomics with r." https://compgenomr.github.io/book/fasta-and-fastq-formats.html, Sep 2020.

[31] “Bioinformatics workflow for whole genome sequencing.” https://www.cd-genomics.com/bioinformatics-workflow-for-whole-genome-sequencing.html.

[32] M. Wiltgen, “Algorithms for structure comparison and analysis: Homology modelling of proteins,” *Encyclopedia of Bioinformatics and Computational Biology*, p. 38–61, 2019.

[33] T. Seemann, “Abricate.” https://github.com/tseemann/abricate.

[34] “What is xgboost?.” https://www.nvidia.com/en-us/glossary/data-science/xgboost/.

[35] “Svms: A new generation of learning algorithms.” https://web.mit.edu/6.034/wwwbob/svm.pdf.

[36] “Support vector machine (svm).” https://www.mathworks.com/discovery/support-vector-machine.html.

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

[38] D. Castillo, “Explainability in machine learning.” https://www.seldon.io/explainability-in-machine-learning, Sep 2021.

[39] S. Yang, “Machine learning model interpretability and explainability.” https://towardsdatascience.com/model-interpretability-and-explainability-27fe31cc0688, Sep 2022.

[40] R. Edgar and E. Bibi, “Mdfa, an escherichia coli multidrug resistance protein with an extraordinarily broad spectrum of drug recognition,” *Journal of Bacteriology*, vol. 179, no. 7, p. 2274–2280, 1997.

[41] P. Chaturvedi, A. Singh, P. Chowdhary, A. Pandey, and P. Gupta, "Occurrence of emerging sulfonamide resistance (sul1 and sul2) associated with mobile integrons-integrase (inti1 and inti2) in riverine systems," *Science of The Total Environment*, vol. 751, p. 142217, 2021.

[42] P. Antunes, J. Machado, J. C. Sousa, and L. Peixe, "Dissemination of sulfonamide resistance genes (sul1,sul2, and sul3) in portuguese salmonella enterica strains and relation with integrons," *Antimicrobial Agents and Chemotherapy*, vol. 49, no. 2, p. 836–839, 2005.

X. Appendix

A. List of Samples

Samples from the NCBI database used for training and testing of ML models:

BioSample Accessions	**Bacteria Species**
SAMN04450960	*A. baumannii*
SAMN04450961	*A. baumannii*
SAMN04479995	*A. baumannii*
SAMN04479996	*A. baumannii*
SAMN04480653	*A. baumannii*
SAMN04480664	*A. baumannii*
SAMN04487135	*A. baumannii*
SAMN04487143	*A. baumannii*
SAMN04488291	*A. baumannii*
SAMN04516582	*A. baumannii*
SAMN04515405	*A. baumannii*
SAMN04515392	*A. baumannii*
SAMN04515796	*A. baumannii*
SAMN04515807	*A. baumannii*
SAMN04515808	*A. baumannii*
SAMN04549260	*A. baumannii*
SAMN04549556	*A. baumannii*
SAMN04550076	*A. baumannii*
SAMN04550118	*A. baumannii*
SAMN04555178	*A. baumannii*
SAMN04555193	*A. baumannii*
SAMN04547088	*A. baumannii*
SAMN04547091	*A. baumannii*
SAMN04547092	*A. baumannii*
SAMN04549515	*A. baumannii*
SAMN04549553	*A. baumannii*
SAMN04549876	*A. baumannii*
SAMN04550082	*A. baumannii*
SAMN04555214	*A. baumannii*
SAMN04555216	*A. baumannii*
SAMN04555495	*A. baumannii*
SAMN04555507	*A. baumannii*
SAMN04446471	*A. baumannii*
SAMN04555219	*A. baumannii*
SAMN06837749	*A. baumannii*
SAMN06837788	*A. baumannii*
SAMN06892260	*A. baumannii*
SAMN06892316	*A. baumannii*
SAMN04014886	*A. baumannii*
SAMN04447718	*A. baumannii*
SAMN04446555	*A. baumannii*
SAMN04485914	*A. baumannii*
SAMN04487142	*A. baumannii*
SAMN04545407	*A. baumannii*
SAMN04546723	*A. baumannii*

SAMN04549500	*A. baumannii*
SAMN04549555	*A. baumannii*
SAMN04549559	*A. baumannii*
SAMN04550105	*A. baumannii*
SAMN04550108	*A. baumannii*
SAMN04550110	*A. baumannii*
SAMN04550111	*A. baumannii*
SAMN04550115	*A. baumannii*
SAMN04550117	*A. baumannii*
SAMN04555180	*A. baumannii*
SAMN04555183	*A. baumannii*
SAMN04555305	*A. baumannii*
SAMN04555231	*A. baumannii*
SAMN04555291	*A. baumannii*
SAMN04555300	*A. baumannii*
SAMN04555303	*A. baumannii*
SAMN04555479	*A. baumannii*
SAMN04555486	*A. baumannii*
SAMN04555487	*A. baumannii*
SAMN04555491	*A. baumannii*
SAMN04568505	*A. baumannii*
SAMN04568508	*A. baumannii*
SAMN04568513	*A. baumannii*
SAMN04568520	*A. baumannii*
SAMN04568514	*A. baumannii*
SAMN04568515	*A. baumannii*
SAMN04555197	*A. baumannii*
SAMN04555199	*A. baumannii*
SAMN04555203	*A. baumannii*
SAMN04555208	*A. baumannii*
SAMN04555209	*A. baumannii*
SAMN04555483	*A. baumannii*
SAMN04014893	*A. baumannii*
SAMN04014897	*A. baumannii*
SAMN04014904	*A. baumannii*
SAMN06892210	*A. baumannii*
SAMN06892241	*A. baumannii*
SAMN06892239	*A. baumannii*
SAMN06892227	*A. baumannii*
SAMN06892221	*A. baumannii*
SAMN06892282	*A. baumannii*
SAMN06892275	*A. baumannii*
SAMN06892258	*A. baumannii*
SAMN07602915	*A. baumannii*
SAMN04014943	*A. baumannii*
SAMN04622920	*E. coli*
SAMN05194537	*E. coli*
SAMN05374484	*E. coli*
SAMN05774084	*E. coli*
SAMN05774085	*E. coli*
SAMN05774076	*E. coli*
SAMN05774080	*E. coli*
SAMN06015664	*E. coli*

SAMN05806166	*E. coli*
SAMN06129824	*E. coli*
SAMN06015662	*E. coli*
SAMN05774079	*E. coli*
SAMN05774077	*E. coli*
SAMN05774082	*E. coli*
SAMN05510454	*E. coli*
SAMN05463123	*E. coli*
SAMN05510453	*E. coli*
SAMN07325901	*E. coli*
SAMN07325899	*E. coli*
SAMN07410708	*E. coli*
SAMN07410718	*E. coli*
SAMN07325904	*E. coli*
SAMN06754737	*E. coli*
SAMN04339724	*E. coli*
SAMN03177645	*E. coli*
SAMN04122822	*E. coli*
SAMN04014842	*E. coli*
SAMN04014858	*E. coli*
SAMN04571741	*E. coli*
SAMN05194531	*E. coli*
SAMN04014959	*E. coli*
SAMN04014991	*E. coli*
SAMN06015667	*E. coli*
SAMN07450853	*E. coli*
SAMN07291526	*E. coli*
SAMN04014847	*E. coli*
SAMN04014852	*E. coli*
SAMN04014854	*E. coli*
SAMN04014855	*E. coli*
SAMN04014856	*E. coli*
SAMN04014860	*E. coli*
SAMN04014889	*E. coli*
SAMN04014918	*E. coli*
SAMN04014922	*E. coli*
SAMN04014926	*E. coli*
SAMN04014927	*E. coli*
SAMN04014930	*E. coli*
SAMN04014908	*E. coli*
SAMN04014910	*E. coli*
SAMN04014902	*E. coli*
SAMN07291511	*E. coli*
SAMN07291515	*E. coli*
SAMN07291521	*E. coli*
SAMN07291530	*E. coli*
SAMN07291545	*E. coli*
SAMN07291544	*E. coli*
SAMN07291527	*E. coli*
SAMN07291528	*E. coli*
SAMN07291543	*E. coli*
SAMN05806175	*E. coli*
SAMN05806164	*E. coli*

SAMN05806165	*E. coli*
SAMN05178549	*E. coli*
SAMN04622925	*E. coli*
SAMN04622926	*E. coli*
SAMN03892126	*K. pneumoniae*
SAMN03892119	*K. pneumoniae*
SAMN03892120	*K. pneumoniae*
SAMN04096282	*K. pneumoniae*
SAMN04096276	*K. pneumoniae*
SAMN04393399	*K. pneumoniae*
SAMN04456590	*K. pneumoniae*
SAMN05194535	*K. pneumoniae*
SAMN05510459	*K. pneumoniae*
SAMN04393399	*K. pneumoniae*
SAMN04456592	*K. pneumoniae*
SAMN05806167	*K. pneumoniae*
SAMN05806176	*K. pneumoniae*
SAMN06680409	*K. pneumoniae*
SAMN07325893	*K. pneumoniae*
SAMN07325895	*K. pneumoniae*
SAMN07332514	*K. pneumoniae*
SAMN07332511	*K. pneumoniae*
SAMN06311516	*K. pneumoniae*
SAMN08045836	*K. pneumoniae*
SAMN05510459	*K. pneumoniae*
SAMN04014851	*K. pneumoniae*
SAMN04014853	*K. pneumoniae*
SAMN04014976	*K. pneumoniae*
SAMN04014981	*K. pneumoniae*
SAMN04014982	*K. pneumoniae*
SAMN04015001	*K. pneumoniae*
SAMN04014884	*K. pneumoniae*
SAMN04014875	*K. pneumoniae*
SAMN04014880	*K. pneumoniae*
SAMN04014883	*K. pneumoniae*
SAMN04514120	*K. pneumoniae*
SAMN04514121	*K. pneumoniae*
SAMN04514122	*K. pneumoniae*
SAMN04514123	*K. pneumoniae*
SAMN04514124	*K. pneumoniae*
SAMN04448219	*K. pneumoniae*
SAMN04448241	*K. pneumoniae*
SAMN04014967	*K. pneumoniae*
SAMN05774083	*K. pneumoniae*
SAMN06680398	*K. pneumoniae*
SAMN04014970	*K. pneumoniae*
SAMN04014979	*K. pneumoniae*
SAMN04014980	*K. pneumoniae*
SAMN04014994	*K. pneumoniae*

B. Source Code

FILE: templates/about.html

```
{% load static %}
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Predict AMR</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.
         min.css" integrity="sha384-MCw98/
         SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
  </head>
  <body>
    <img src="{% static 'header.png' %}" class="img-fluid" alt="header image">

    <nav class="navbar navbar-expand-lg" style="background-color: #273755;">
        <div class="container-fluid nav justify-content-center" style="font-size: 20px;">
          <div class="nav justify-content-center" id="navbarNav">
            <ul class="navbar-nav">
              <li class="nav-item">
                <a class="nav-link text-white" aria-current="page" href="{% url 'dashboard:about'
                     %}">About</a>
              </li>
              <li class="nav-item">
                <a class="nav-link text-white" href="{% url 'dashboard:demo' %}">Demo</a>
              </li>
              <li class="nav-item">
                <a class="nav-link text-white" href="{% url 'dashboard:antibiogram' %}">
                     Antibiogram</a>
              </li>
              <li class="nav-item">
                <a class="nav-link text-white" href="{% url 'dashboard:modeldetails' %}">Model
                     Details</a>
              </li>
            </ul>
          </div>
        </div>
    </nav>

      <br>
      <div class="card w-75 mx-auto">
        <div class="card-header">
          Using Machine Learning Algorithms and Integrons to Predict Antimicrobial Resistance in
               Common Infection-causing Bacteria
        </div>
        <div class="card-body">
          <h5 class="card-title">In Silico Antibiogram Tool</h5>
          <p class="card-text">This application provides a machine learning approach for
               resistance prediction that may accelerate the treatment of patients in the future.
               It is beneficial for clinicians, as it can guide in making a faster decision on what
                type of effective antibiotic regimens are needed by a patient.<br><br>
          a. The app reads FASTA format whole-genome sequencing data. <br>
          b. The app shows predicted antibiograms. <br>
          c. The app displays integron presence and detected AMR genes on WGS. <br>
          d. The app provides demo samples of bacteria isolates. <br>
          e. The app presents details of prediction models used for training, including
               performance summary.</p>
        </div>
      </div>

      <br>

      <div class="card w-75 mx-auto">
        <img class="card-img-top" src="{% static 'ast.jpg' %}" style="object-fit: cover;">
        <div class="card-body">
          <h5 class="card-title">According to the WHO, the AMR is one of the top 10 globlal public
                health threats in the world.</h5>
          <p class="card-text">It is established that mutations and integrons contribute to the
               spread of resistance genes among bacteria leading to accelerated AMR. This
               application presents a novel approach in predicting AMR based on the presence of
               integrons and AMR genes on whole-genome sequence (WGS) of common infection-causing
               bacteria using machine learning.</p>
        </div>
      </div>

      <br>

    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965
         DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js" integrity="
         sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="
         anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/js/bootstrap.min.js" integrity
         ="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="
         anonymous"></script>
  </body>
</html>
```

FILE: templates/demo.html

```
{% load static %}
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Predict AMR</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.
         min.css" integrity="sha384-MCw98/
         SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
  </head>
  <body>
    <img src="{% static 'header.png' %}" class="img-fluid" alt="header image">

    <nav class="navbar navbar-expand-lg" style="background-color: #273755;">
      <div class="container-fluid nav justify-content-center" style="font-size: 20px;">
        <div class="nav justify-content-center" id="navbarNav">
          <ul class="navbar-nav">
            <li class="nav-item">
              <a class="nav-link text-white" aria-current="page" href="{% url 'dashboard:about'
                   %}">About</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-white" href="{% url 'dashboard:demo' %}">Demo</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-white" href="{% url 'dashboard:antibiogram' %}">Antibiogram
                   </a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-white" href="{% url 'dashboard:modeldetails' %}">Model
                   Details</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>

    <br>

    <div class="card w-50 mx-auto">
      <div class="card-header">
        Demo
      </div>
      <div class="card-body text-center">
        <h5 class="card-title">Select a sample</h5>
        <div class="input-group mb-3">
          <select class="custom-select" id="inputGroupSelect02">
            <option selected>Some available samples</option>
            <option value="1">Escherichia coli - SAMN06311517</option>
            <option value="2">Klebsiella pneumoniae - SAMN05774081</option>
            <option value="3">Acinetobacter baumannii - SAMN04549829</option>
            <option value="4">Escherichia coli - SAMN04096283</option>
            <option value="5">Klebsiella pneumoniae - SAMN07325896</option>
          </select>
          <div class="input-group-append">
            <label class="input-group-text" for="inputGroupSelect02">Options</label>
          </div>
        </div>
      </div>
    </div>

    <br>

    <div class="row mx-auto">
      <div class="col-sm-6">
        <div class="card">
          <div class="card-body">
            <h5 class="card-title">Predicted Antibiogram</h5>
            <p class="card-text"></p>
            <table class="table table-striped">
              <thead>
                <tr>
                  <th scope="col">Antibiotic</th>
                  <th scope="col">Prediction</th>
                </tr>
              </thead>
              <tbody>
                <tr>
                  <td>Cefotaxime</td>
                  <td><span id="cefotaxime_pred"></span></td>
                </tr>
                <tr>
                  <td>Ceftriaxone</td>
                  <td><span id="ceftriaxone_pred"></span></td>
                </tr>
                <tr>
                  <td>Ciprofloxacin</td>
                  <td><span id="ciprofloxacin_pred"></span></td>
                </tr>
                <tr>
                  <td>Gentamicin</td>
                  <td><span id="gentamicin_pred"></span></td>
                </tr>
                <tr>
                  <td>Levofloxacin</td>
                  <td><span id="levofloxacin_pred"></span></td>
                </tr>
              </tbody>
            </table>
          </div>
        </div>
      </div>
      <div class="col-sm-6">
        <div class="card">
          <div class="card-body">
            <h5 class="card-title">Integron and AMR Genes</h5>
            <p class="card-text"></p>
            <table class="table table-striped">
              <thead>
                <tr>
                  <th scope="row">Integron:</th>
                  <td scope="col"><span id="integron"></span></td>
                </tr>
              </thead>
              <tbody>
                <tr>
                  <th scope="row">AMR Genes:</th>
                  <td><span id="amr_genes"></span></td>
                </tr>
              </tbody>
            </table>
          </div>
          <div class="card-footer text-muted">
            Database used: Resfinder <br>
          </div>
        </div>
      </div>
    </div>

    <script>
      // Add an event listener to the dropdown menu
      var dropdown = document.getElementById("inputGroupSelect02");
      dropdown.addEventListener("change", function() {
        var selectedOption = dropdown.value;
        updatePerformanceStats(selectedOption);
      });

      function updatePerformanceStats(selectedOption) {
      // Update the performance statistics based on the selected antibiotic option
        if (selectedOption === "1") {
          document.getElementById("cefotaxime_pred").innerText = "Resistant";
          document.getElementById("ceftriaxone_pred").innerText = "Resistant";
          document.getElementById("ciprofloxacin_pred").innerText = "Susceptible";
          document.getElementById("gentamicin_pred").innerText = "Susceptible";
          document.getElementById("levofloxacin_pred").innerText = "Susceptible";
          document.getElementById("integron").innerText = "Integron found";
          document.getElementById("amr_genes").innerText = "aph(6)-Id, aph(3'')-Ib, blaCTX-M-15,
              mdf(A), erm(B), blaTEM-1B, tet(B), dfrA1, mph(A), ant(3'')-Ia, sul2";
        }
        else if (selectedOption === "2") {
          document.getElementById("cefotaxime_pred").innerText = "Resistant";
          document.getElementById("ceftriaxone_pred").innerText = "Resistant";
          document.getElementById("ciprofloxacin_pred").innerText = "Resistant";
          document.getElementById("gentamicin_pred").innerText = "Resistant";
          document.getElementById("levofloxacin_pred").innerText = "Resistant";
          document.getElementById("integron").innerText = "Integron found";
          document.getElementById("amr_genes").innerText = "oqxB, aph(6)-Id, qnrB1, fosA6, blaCTX-
              M-15, aph(3'')-Ib, aac(3)-IIa, blaTEM-1B, aac(6')-Ib-cr, blaOXA-1, tet(A), dfrA14,
              blaSHV-106, oqxA, sul2";
        }
        else if (selectedOption === "3") {
          document.getElementById("cefotaxime_pred").innerText = "Resistant";
          document.getElementById("ceftriaxone_pred").innerText = "Resistant";
          document.getElementById("ciprofloxacin_pred").innerText = "Resistant";
          document.getElementById("gentamicin_pred").innerText = "Resistant";
          document.getElementById("levofloxacin_pred").innerText = "Susceptible";
          document.getElementById("integron").innerText = "Integron found";
          document.getElementById("amr_genes").innerText = "catA1, aac(3)-Ia, blaTEM-1D, tet(A),
              aph(3')-Ia, sul1, blaADC-25, ant(3'')-Ia, blaOXA-69";
        }
        else if (selectedOption === "4") {
          document.getElementById("cefotaxime_pred").innerText = "Resistant";
          document.getElementById("ceftriaxone_pred").innerText = "Resistant";
          document.getElementById("ciprofloxacin_pred").innerText = "Resistant";
          document.getElementById("gentamicin_pred").innerText = "Resistant";
          document.getElementById("levofloxacin_pred").innerText = "Resistant";
          document.getElementById("integron").innerText = "Integron found";
          document.getElementById("amr_genes").innerText = "mdf(A), aac(3)-IIa, blaTEM-1B, sul3,
              aph(3')-Ia, qnrS1, blaCTX-M-55, ant(3'')-Ia, dfrA14, floR";
        }
        else if (selectedOption === "5") {
          document.getElementById("cefotaxime_pred").innerText = "Resistant";
          document.getElementById("ceftriaxone_pred").innerText = "Resistant";
          document.getElementById("ciprofloxacin_pred").innerText = "Resistant";
          document.getElementById("gentamicin_pred").innerText = "Resistant";
          document.getElementById("levofloxacin_pred").innerText = "Resistant";
          document.getElementById("integron").innerText = "Integron found";
          document.getElementById("amr_genes").innerText = "oqxB, aph(6)-Id, qnrB1, fosA6, blaCTX-
              M-15, aph(3'')-Ib, aac(3)-IIa, blaTEM-1B, aac(6')-Ib-cr, blaOXA-1, blaCTX-M-156, tet
              (A), dfrA14, blaSHV-110, oqxA, sul2";
        }
      }
    </script>

    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965
        DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js" integrity="
        sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="
        anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/js/bootstrap.min.js" integrity
        ="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="
        anonymous"></script>
  </body>
</html>
```

FILE: templates/antibiogram.html

```
{% load static %}
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Predict AMR</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.
        min.css" integrity="sha384-MCw98/
        SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
  </head>
  <body>
    <img src="{% static 'header.png' %}" class="img-fluid" alt="header image">

    <nav class="navbar navbar-expand-lg" style="background-color: #273755;">
      <div class="container-fluid nav justify-content-center" style="font-size: 20px;">
        <div class="nav justify-content-center" id="navbarNav">
          <ul class="navbar-nav">
            <li class="nav-item">
              <a class="nav-link text-white" aria-current="page" href="{% url 'dashboard:about'
                  %}">About</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-white" href="{% url 'dashboard:demo' %}">Demo</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-white" href="{% url 'dashboard:antibiogram' %}">Antibiogram
                  </a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-white" href="{% url 'dashboard:modeldetails' %}">Model
                  Details</a>
            </li>
          </ul>
        </div>
      </div>
    </nav>

    <br>

    <form method="post" enctype="multipart/form-data" id="antiform">
      {% csrf_token %}
      <div class="card w-50 mx-auto">
        <div class="card-header">
          Antibiogram Tool
        </div>
        <div class="card-body text-center">
          <h5 class="card-title">Select a sample</h5>
          <div class="input-group">
            <div class="custom-file">
              <!-- <input type="file" class="custom-file-input" id="inputGroupFile04" name="
                  fasta_file">
              <label class="custom-file-label" for="inputGroupFile04">Choose file</label> -->
              {{ form.fasta_file }}
            </div>
            <div class="input-group-append">
              <button class="btn btn-outline-secondary" type="submit" id="inputGroupFileAddon04">
                  Submit</button>
            </div>
          </div>
        </div>
      </div>
    </form>


    <br>
```

```
<div class="row mx-auto">
  <div class="col-sm-6">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Predicted Antibiogram</h5>
        <p class="card-text">Filename: {{ filename }}</p>
        <table class="table table-striped">
          <thead>
            <tr>
              <th scope="col">Antibiotic</th>
              <th scope="col">Prediction</th>
              {# <th scope="col">Model Accuracy</th> #}
            </tr>
          </thead>
          <tbody>
            <tr>
              <td>Cefotaxime</td>
              <td>{% if cefotaxime_prediction == 0 %} Susceptible {% elif
                   cefotaxime_prediction == 1 %} Resistant {% else %} {% endif %}</td>
              {# <td>-</td> #}
            </tr>
            <tr>
              <td>Ceftriaxone</td>
              <td>{% if ceftriaxone_prediction == 0 %} Susceptible {% elif
                   ceftriaxone_prediction == 1 %} Resistant {% else %} {% endif %}</td>
              {# <td>-</td> #}
            </tr>
            <tr>
              <td>Ciprofloxacin</td>
              <td>{% if ciprofloxacin_prediction == 0 %} Susceptible {% elif
                   ciprofloxacin_prediction == 1 %} Resistant {% else %} {% endif %}</td>
              {# <td>-</td> #}
            </tr>
            <tr>
              <td>Gentamicin</td>
              <td>{% if gentamicin_prediction == 0 %} Susceptible {% elif
                   gentamicin_prediction == 1 %} Resistant {% else %} {% endif %}</td>
              {# <td>-</td> #}
            </tr>
            <tr>
              <td>Levofloxacin</td>
              <td>{% if levofloxacin_prediction == 0 %} Susceptible {% elif
                   levofloxacin_prediction == 1 %} Resistant {% else %} {% endif %}</td>
              {# <td>-</td> #}
            </tr>
          </tbody>
        </table>
      </div>
    </div>
  </div>
  <div class="col-sm-6">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Integron and AMR Genes</h5>
        <p class="card-text">Filename: {{ filename }}</p>
        <table class="table table-striped">
          <thead>
            <tr>
              <th scope="row">Integron:</th>
              <td scope="col">{% if integron_presence == 0 %} Integron not found {% elif
                   integron_presence == 1 %} Integron found {% else %} {% endif %}</td>
            </tr>
          </thead>
          <tbody>
            <tr>
              <th scope="row">AMR Genes:</th>
              <td>{{ abricate_gene_products | join:", "}}</td>
            </tr>
          </tbody>
        </table>
      </div>
      <div class="card-footer text-muted">
        Database used: Resfinder <br>
      </div>
    </div>
  </div>
</div>

<!-- Add styles for the overlay element -->
<style>
    .overlay {
        position: fixed;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        background-color: rgba(255, 255, 255, 0.8);
        z-index: 9999;
        display: flex;
        justify-content: center;
        align-items: center;
        }

        .spinner {
            display: inline-block;
            width: 40px;
            height: 40px;
            border: 4px solid #f3f3f3;
            border-top: 4px solid #3498db;
            border-radius: 50%;
            animation: spin 2s linear infinite;
        }

        @keyframes spin {
            0% {
                transform: rotate(0deg);
            }
            100% {
                transform: rotate(360deg);
            }
        }
    </style>

    <!-- Add the overlay and spinner elements -->
    <div class="overlay" id="overlay" style="display: none;">
        <div class="spinner"></div>
    </div>

    <!-- Add a JavaScript code snippet to show/hide the overlay -->
    <script>
        // Show the spinner overlay
        function showSpinner() {
            var overlay = document.getElementById('overlay');
            overlay.style.display = 'flex';
        }

        // Hide the spinner overlay
        function hideSpinner() {
            var overlay = document.getElementById('overlay');
            overlay.style.display = 'none';
        }

        // Show the spinner when a link is clicked
        var links = document.querySelectorAll('a');
        links.forEach(function(link) {
            link.addEventListener('click', function() {
                showSpinner();
            });
        });

        // Show the spinner when the form is submitted
        var form = document.getElementById('antiform');
        form.addEventListener('submit', function() {
            showSpinner();
        });

        // Hide the spinner when the page finishes loading
        window.addEventListener('load', function() {
            hideSpinner();
        });
    </script>

    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965
         DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js" integrity="
         sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="
         anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/js/bootstrap.min.js" integrity
         ="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="
         anonymous"></script>
  </body>
</html>
```

FILE: templates/modeldetails.html

```
{% load static %}
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Predict AMR</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/css/bootstrap.
         min.css" integrity="sha384-MCw98/
         SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO" crossorigin="anonymous">
  </head>
  <body>
    <img src="{% static 'header.png' %}" class="img-fluid" alt="header image">

    <nav class="navbar navbar-expand-lg" style="background-color: #273755;">
      <div class="container-fluid nav justify-content-center" style="font-size: 20px;">
        <div class="nav justify-content-center" id="navbarNav">
          <ul class="navbar-nav">
        <li class="nav-item">
          <a class="nav-link text-white" aria-current="page" href="{% url 'dashboard:about'
              %}">About</a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-white" href="{% url 'dashboard:demo' %}">Demo</a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-white" href="{% url 'dashboard:antibiogram' %}">Antibiogram
              </a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-white" href="{% url 'dashboard:modeldetails' %}">Model
              Details</a>
        </li>
      </ul>
    </div>
  </div>
</nav>

<br>

<div class="card w-50 mx-auto">
  <div class="card-header">
    Performance Summary
  </div>
  <div class="card-body text-center">
    <div class="input-group mb-3">
      <select class="custom-select" id="inputGroupSelect02">
        <option selected>Choose antibiotic</option>
        <option value="1">Cefotaxime</option>
        <option value="2">Ceftriaxone</option>
        <option value="3">Ciprofloxacin</option>
        <option value="4">Gentamicin</option>
        <option value="5">Levofloxacin</option>
      </select>
      <div class="input-group-append">
        <label class="input-group-text" for="inputGroupSelect02">Options</label>
      </div>
    </div>
  </div>
</div>

<br>

<div class="row mx-auto">

  <div class="col-sm-4">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Number of Samples</h5>
        <h6 class="card-subtitle mb-2 text-muted">Samples in dataset</h6>
        <p class="card-text">
          <b>Resistant:</b> <span id="resistant"></span><br>
          <b>Susceptible:</b> <span id="susceptible"></span><br>
          <b>Total:</b> <span id="total"></span><br>
        </p>
        <h6 class="card-subtitle mb-2 text-muted">Samples in training & testing set</h6>
        <p class="card-text">
          <b>Training set:</b> <span id="trainingset"></span><br>
          <b>Testing set:</b> <span id="testingset"></span><br>
        </p>
        <h5 class="card-title">Feature Importance</h5>
        <p class="card-text">
          <span id="fi"></span>
        </p>
      </div>
    </div>
  </div>

  <div class="col-sm-4">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Confusion Matrix of Testing Set</h5>
        <p class="card-text">
          <span id="cm"></span>
        </p>
      </div>
    </div>
  </div>

  <div class="col-sm-4">
    <div class="card">
      <div class="card-body">
        <h5 class="card-title">Performance Statistics</h5>
        <h6 class="card-subtitle mb-2 text-muted">Model: <span id="model"></span></h6>
        <p class="card-text">
          <b>Accuracy:</b> <span id="accuracy"></span><br>
          <b>Precision:</b> <span id="precision"></span><br>
          <b>Recall:</b> <span id="recall"></span><br>
          <b>Area under the ROC curve:</b> <span id="auc"></span><br>
          <b>ROC curve:</b> <span id="roc"></span><br>
        </p>
      </div>
    </div>
  </div>

</div>

<script>
  // Add an event listener to the dropdown menu
  var dropdown = document.getElementById("inputGroupSelect02");
  dropdown.addEventListener("change", function() {
    var selectedOption = dropdown.value;
    updatePerformanceStats(selectedOption);
  });

  function updatePerformanceStats(selectedOption) {
  // Update the performance statistics based on the selected antibiotic option
    if (selectedOption === "1") {
      document.getElementById("model").innerText = "Random Forest";
      document.getElementById("resistant").innerText = "{{ num_resistant1 }}";
      document.getElementById("susceptible").innerText = "{{ num_susceptible1 }}";
      document.getElementById("total").innerText = "{{ total_samples1 }}";
      document.getElementById("trainingset").innerText = "{{ num_train_samples1 }}";
      document.getElementById("testingset").innerText = "{{ num_test_samples1 }}";
      document.getElementById("accuracy").innerText = "{{ accuracy1 }}";
      document.getElementById("precision").innerText = "{{ precision1 }}";
      document.getElementById("recall").innerText = "{{ recall1 }}";
      //document.getElementById("mcc").innerText = "{{ mcc1 }}";
      document.getElementById("auc").innerText = "{{ auc1 }}";
      document.getElementById("roc").innerText = "{{ roc_curve }}";
      document.getElementById("roc").innerHTML = '<img src="{% static "cefotaxime_roc_curve.
           png" %}" alt="ROC curve" width="400" height="300">';
      document.getElementById("cm").innerHTML = '<img src="{% static "
           cefotaxime_confusion_matrix.png" %}" alt="Confusion matrix" width="400" height
           ="300">';
      document.getElementById("fi").innerHTML = '<img src="{% static "cefotaxime_features.png"
            %}" alt="Feature Importance" width="490" height="300">';
    }
    else if (selectedOption === "2") {
      document.getElementById("model").innerText = "Random Forest";
      document.getElementById("resistant").innerText = "{{ num_resistant2 }}";
      document.getElementById("susceptible").innerText = "{{ num_susceptible2 }}";
      document.getElementById("total").innerText = "{{ total_samples2 }}";
      document.getElementById("trainingset").innerText = "{{ num_train_samples2 }}";
      document.getElementById("testingset").innerText = "{{ num_test_samples2 }}";
      document.getElementById("accuracy").innerText = "{{ accuracy2 }}";
      document.getElementById("precision").innerText = "{{ precision2 }}";
      document.getElementById("recall").innerText = "{{ recall2 }}";
      //document.getElementById("mcc").innerText = "{{ mcc2 }}";
      document.getElementById("auc").innerText = "{{ auc2 }}";
      document.getElementById("roc").innerHTML = '<img src="{% static "ceftriaxone_roc_curve.
           png" %}" alt="ROC curve" width="400" height="300">';
      document.getElementById("cm").innerHTML = '<img src="{% static "
           ceftriaxone_confusion_matrix.png" %}" alt="Confusion matrix" width="400" height
           ="300">';
      document.getElementById("fi").innerHTML = '<img src="{% static "ceftriaxone_features.png
           " %}" alt="Feature Importance" width="490" height="300">';
    }
    else if (selectedOption === "3") {
      document.getElementById("model").innerText = "Random Forest";
      document.getElementById("resistant").innerText = "{{ num_resistant3 }}";
      document.getElementById("susceptible").innerText = "{{ num_susceptible3 }}";
      document.getElementById("total").innerText = "{{ total_samples3 }}";
      document.getElementById("trainingset").innerText = "{{ num_train_samples3 }}";
      document.getElementById("testingset").innerText = "{{ num_test_samples3 }}";
      document.getElementById("accuracy").innerText = "{{ accuracy3 }}";
      document.getElementById("precision").innerText = "{{ precision3 }}";
      document.getElementById("recall").innerText = "{{ recall3 }}";
      //document.getElementById("mcc").innerText = "{{ mcc3 }}";
      document.getElementById("auc").innerText = "{{ auc3 }}";
      document.getElementById("roc").innerHTML = '<img src="{% static "ciprofloxacin_roc_curve
           .png" %}" alt="ROC curve" width="400" height="300">';
      document.getElementById("cm").innerHTML = '<img src="{% static "
           ciprofloxacin_confusion_matrix.png" %}" alt="Confusion matrix" width="400" height
           ="300">';
      document.getElementById("fi").innerHTML = '<img src="{% static "ciprofloxacin_features.
           png" %}" alt="Feature Importance" width="490" height="300">';
    }
    else if (selectedOption === "4") {
      document.getElementById("model").innerText = "Extreme Gradient Boost";
      document.getElementById("resistant").innerText = "{{ num_resistant4 }}";
      document.getElementById("susceptible").innerText = "{{ num_susceptible4 }}";
      document.getElementById("total").innerText = "{{ total_samples4 }}";
      document.getElementById("trainingset").innerText = "{{ num_train_samples4 }}";
      document.getElementById("testingset").innerText = "{{ num_test_samples4 }}";
      document.getElementById("accuracy").innerText = "{{ accuracy4 }}";
      document.getElementById("precision").innerText = "{{ precision4 }}";
      document.getElementById("recall").innerText = "{{ recall4 }}";
      //document.getElementById("mcc").innerText = "{{ mcc4 }}";
      document.getElementById("auc").innerText = "{{ auc4 }}";
      document.getElementById("roc").innerHTML = '<img src="{% static "gentamicin_roc_curve.
           png" %}" alt="ROC curve" width="400" height="300">';
```

```
            document.getElementById("cm").innerHTML = '<img src="{% static "
                 gentamicin_confusion_matrix.png" %}" alt="Confusion matrix" width="400" height
                 ="300">';
            document.getElementById("fi").innerHTML = '<img src="{% static "gentamicin_features.png"
                  %}" alt="Feature Importance" width="490" height="300">';
          }
          else if (selectedOption === "5") {
            document.getElementById("model").innerText = "Random Forest";
            document.getElementById("resistant").innerText = "{{ num_resistant5 }}";
            document.getElementById("susceptible").innerText = "{{ num_susceptible5 }}";
            document.getElementById("total").innerText = "{{ total_samples5 }}";
            document.getElementById("trainingset").innerText = "{{ num_train_samples5 }}";
            document.getElementById("testingset").innerText = "{{ num_test_samples5 }}";
            document.getElementById("accuracy").innerText = "{{ accuracy5 }}";
            document.getElementById("precision").innerText = "{{ precision5 }}";
            document.getElementById("recall").innerText = "{{ recall5 }}";
            //document.getElementById("mcc").innerText = "{{ mcc5 }}";
            document.getElementById("auc").innerText = "{{ auc5 }}";
            document.getElementById("roc").innerHTML = '<img src="{% static "levofloxacin_roc_curve.
                 png" %}" alt="ROC curve" width="400" height="300">';
            document.getElementById("cm").innerHTML = '<img src="{% static "
                 levofloxacin_confusion_matrix.png" %}" alt="Confusion matrix" width="400" height
                 ="300">';
            document.getElementById("fi").innerHTML = '<img src="{% static "levofloxacin_features.
                 png" %}" alt="Feature Importance" width="490" height="300">';
          }
        }
      </script>


      <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965
           DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
      <script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.3/dist/umd/popper.min.js" integrity="
           sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49" crossorigin="
           anonymous"></script>
      <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.1.3/dist/js/bootstrap.min.js" integrity
           ="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy" crossorigin="
           anonymous"></script>
   </body>
</html>
```

FILE: views.py

```
from django.shortcuts import render
from dashboard.forms import *
import subprocess

import os
from django.conf import settings

#Import and load the ML model
import joblib
import pickle
cefotaxime_svm_model = joblib.load('cefotaxime_svm_model.pkl')
ceftriaxone_rf_model = joblib.load('ceftriaxone_rf_model.pkl')
ciprofloxacin_rf_model = joblib.load('ciprofloxacin_rf_model.pkl')
gentamicin_xgboost_model = joblib.load('gentamicin_xgboost_model.pkl')
levofloxacin_rf_model = joblib.load('levofloxacin_rf_model.pkl')

import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import seaborn as sns


# Create your views here.
def about(request):
    return render(request, 'about.html', {})

def demo(request):
    return render(request, 'demo.html', {})

def antibiogram(request):
    form = FastaUploadForm()
    if request.method == 'POST':
        form = FastaUploadForm(request.POST, request.FILES)
        if form.is_valid():
            # Save the uploaded file to the local machine
            uploaded_file = request.FILES['fasta_file']
            filename = handle_uploaded_file(uploaded_file)

            # Run IntegronFinder
            subprocess.check_call(['integron_finder', '--local-max', filename], cwd = '/home/
                 vboxuser/predict_amr/media')

            # Check if IntegronFinder found an integron
            integron_file_path = '/home/vboxuser/predict_amr/media/Results_Integron_Finder_' + os.
                 path.splitext(filename)[0] + '/' + os.path.splitext(filename)[0] + '.integrons'
```

```
with open(integron_file_path, 'r') as integron_file:
    integron_lines = integron_file.readlines()
    if len(integron_lines) > 1 and integron_lines[1].startswith("# No Integron found")
        :
        integron_presence = 0  # Integron not found
    else:
        integron_presence = 1  # Integron found

# Create a DataFrame for integron presence
integron_df = pd.DataFrame({'Integron_Presence': [integron_presence]})

# Run Abricate
abricate_output = subprocess.check_output(['./abricate/bin/abricate', '--db', '
    resfinder', '/home/vboxuser/predict_amr/media/'+filename], cwd = '/home/vboxuser')
abricate_output = abricate_output.decode('utf-8')

abricate_lines = abricate_output.strip().split('\n')
abricate_gene_products = set(abricate_line.split()[13] for abricate_line in
    abricate_lines[1:])

for abricate_gene_product in abricate_gene_products:
    print(abricate_gene_product)

# Create a feature vector using one-hot encoding
mlb = MultiLabelBinarizer()
encoded_features = mlb.fit_transform([abricate_gene_products])

# Convert the encoded features to a DataFrame if needed
features_df = pd.DataFrame(encoded_features, columns=mlb.classes_)
# print(features_df)

# Concatenate integron presence DataFrame with features DataFrame
processed_data = pd.concat([features_df, integron_df], axis=1)
# print(processed_data)

# Get the list of AMR genes used during training from the SVM model excluding the
    integron_presence column
amr_genes = [
    "ARR-3", "aac(3)-IIa", "aac(3)-IId", "aac(3)-IVa", "aac(3)-Ia", "aac(6')-IIa", "
        aac(6')-Iaf",
    "aac(6')-Ian", "aac(6')-Ib", "aac(6')-Ib-cr", "aadA", "aadA1", "aadA16", "aadA17",
         "aadA2",
    "aadA5", "aadA6", "ant(2'')-Ia", "ant(3'')-Ia", "aph(3'')-Ib", "aph(3')-Ia", "aph
        (3')-VI",
    "aph(3')-VIa", "aph(3')-XV", "aph(4)-Ia", "aph(6)-Id", "armA", "blaADC-25", "
        blaCARB-10",
    "blaCARB-16", "blaCMY-2", "blaCMY-4", "blaCMY-42", "blaCMY-6", "blaCMY-94", "
        blaCTX-M-1",
    "blaCTX-M-139", "blaCTX-M-14", "blaCTX-M-15", "blaCTX-M-27", "blaCTX-M-3", "blaCTX
        -M-55",
    "blaIMP-4", "blaKPC-2", "blaKPC-3", "blaNDM-1", "blaOXA-1", "blaOXA-10", "blaOXA
        -100",
    "blaOXA-106", "blaOXA-120", "blaOXA-121", "blaOXA-124", "blaOXA-126", "blaOXA
        -181", "blaOXA-2",
    "blaOXA-203", "blaOXA-208", "blaOXA-217", "blaOXA-23", "blaOXA-235", "blaOXA-259",
         "blaOXA-260",
    "blaOXA-314", "blaOXA-343", "blaOXA-407", "blaOXA-413", "blaOXA-430", "blaOXA
        -510", "blaOXA-528",
    "blaOXA-529", "blaOXA-531", "blaOXA-558", "blaOXA-58", "blaOXA-64", "blaOXA-65", "
        blaOXA-66",
    "blaOXA-69", "blaOXA-72", "blaOXA-88", "blaOXA-9", "blaOXA-90", "blaOXA-91", "
        blaOXA-94",
    "blaOXA-98", "blaPER-1", "blaSHV-101", "blaSHV-106", "blaSHV-108", "blaSHV-11", "
        blaSHV-110",
    "blaSHV-12", "blaSHV-14", "blaSHV-145", "blaSHV-182", "blaSHV-187", "blaSHV-26", "
        blaSHV-27",
    "blaSHV-30", "blaSHV-33", "blaSHV-76", "blaTEM-141", "blaTEM-1A", "blaTEM-1B", "
        blaTEM-1D",
    "blaVIM-1", "catA1", "catA2", "catB2", "catB3", "catB8", "cmlA1",
    "dfrA1", "dfrA12", "dfrA14", "dfrA17", "dfrA26", "dfrA27", "dfrA29", "dfrA8", "ere
        (A)", "erm(42)",
    "erm(B)", "floR", "fosA", "fosA5", "fosA6", "fosA7", "mdf(A)", "mph(A)", "mph(E)",
         "msr(E)",
    "oqxA", "oqxB", "qnrB1", "qnrB19", "qnrB2", "qnrB4", "qnrB9", "qnrS1", "rmtB", "
        rmtC", "rmtF",
    "sul1", "sul2", "sul3", "tet(39)", "tet(A)", "tet(B)", "tet(G)"
]

# Initialize an empty dictionary to store the feature values
feature_values = {}

# Iterate over each AMR gene and check if it is present in the input fasta file
for amr_gene in amr_genes:
    if amr_gene in abricate_gene_products:
        feature_values[amr_gene] = 1  # Gene is present
    else:
        feature_values[amr_gene] = 0  # Gene is missing

# Create a DataFrame from the feature values
features_df = pd.DataFrame([feature_values])

# Add the integron presence feature to the DataFrame
```

```
            features_df['Integron_Presence'] = integron_presence

            # Load the ML models
            cefotaxime_svm_model = joblib.load('cefotaxime_svm_model.pkl')
            ceftriaxone_rf_model = joblib.load('ceftriaxone_rf_model.pkl')
            ciprofloxacin_rf_model = joblib.load('ciprofloxacin_rf_model.pkl')
            gentamicin_xgboost_model = joblib.load('gentamicin_xgboost_model.pkl')
            levofloxacin_rf_model = joblib.load('levofloxacin_rf_model.pkl')

            # Perform prediction using the models
            cefotaxime_prediction = cefotaxime_svm_model.predict(features_df)
            ceftriaxone_prediction = ceftriaxone_rf_model.predict(features_df)
            ciprofloxacin_prediction = ciprofloxacin_rf_model.predict(features_df)
            gentamicin_prediction = gentamicin_xgboost_model.predict(features_df)
            levofloxacin_prediction = levofloxacin_rf_model.predict(features_df)

            context = {'cefotaxime_prediction':cefotaxime_prediction, 'ceftriaxone_prediction':
                ceftriaxone_prediction,
                       'ciprofloxacin_prediction':ciprofloxacin_prediction, '
                           gentamicin_prediction':gentamicin_prediction,
                       'levofloxacin_prediction':levofloxacin_prediction, 'integron_presence':
                           integron_presence, 'form': form,
                       'abricate_gene_products':abricate_gene_products, 'filename':filename}
            # Render the template with the predictions
            return render(request, 'antibiogram.html', context)
    return render(request, 'antibiogram.html', {'form': form})

def modeldetails(request):
    # Load the test data
    features_data = pd.read_excel('processed_data.xlsx')
    target = pd.read_excel('combined_dataset.xlsx')

    cefotaxime_target = target['Cefotaxime_Resistance']
    ceftriaxone_target = target['Ceftriaxone_Resistance']
    ciprofloxacin_target = target['Ciprofloxacin_Resistance']
    gentamicin_target = target['Gentamicin_Resistance']
    levofloxacin_target = target['Levofloxacin_Resistance']

    # ——— Cefotaxime ———
    # Cefotaxime has missing values, drop the rows
    data1 = pd.concat([features_data, cefotaxime_target], axis=1)
    data1 = data1.dropna(subset=['Cefotaxime_Resistance'])

    # Split the data into training and testing sets
    X_train1, X_test1, y_train1, y_test1 = train_test_split(data1[features_data.columns], data1['
        Cefotaxime_Resistance'], test_size=0.3, random_state=6)

    # Make predictions on the test set using the loaded model
    y_pred1 = cefotaxime_svm_model.predict(X_test1)

    # Calculate the metrics using the test data and predictions
    accuracy1 = accuracy_score(y_test1, y_pred1)
    precision1 = precision_score(y_test1, y_pred1)
    recall1 = recall_score(y_test1, y_pred1)

    # Calculate the ROC curve and AUC
    fpr1, tpr1, thresholds1 = roc_curve(y_test1, y_pred1)
    auc1 = roc_auc_score(y_test1, y_pred1)

    # Calculate the MCC
    mcc1 = matthews_corrcoef(y_test1, y_pred1)

    # Total number of samples in the dataset
    num_resistant1 = data1['Cefotaxime_Resistance'].value_counts()[1]
    num_susceptible1 = data1['Cefotaxime_Resistance'].value_counts()[0]
    total_samples1 = len(data1)

    # Number of samples in the training set
    num_train_samples1 = len(X_train1)

    # Number of samples in the testing set
    num_test_samples1 = len(X_test1)


    # Plot the ROC curve
    fig, ax = plt.subplots()
    ax.plot(fpr1, tpr1, label='ROC curve (area = %.2f)' % auc1)
    ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
    ax.set_title('ROC curve')
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.grid(True)
    ax.legend()
    plt.savefig('static/cefotaxime_roc_curve.png')  # Save the figure as a file
    plt.close(fig)  # Close the figure

    # Generate confusion matrix
    cm1 = confusion_matrix(y_test1, y_pred1)
    fig, ax = plt.subplots()
    sns.heatmap(cm1, annot=True, fmt='g', cmap='Blues', ax=ax)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
```

```
plt.savefig('static/cefotaxime_confusion_matrix.png')  # Save the figure as a file
plt.close(fig)  # Close the figure


# ----- Ceftriaxone -----
# Ceftriaxone has missing values, drop the rows
data2 = pd.concat([features_data, ceftriaxone_target], axis=1)
data2 = data2.dropna(subset=['Ceftriaxone_Resistance'])

# Split the data into training and testing sets
X_train2, X_test2, y_train2, y_test2 = train_test_split(data2[features_data.columns], data2['
     Ceftriaxone_Resistance'], test_size=0.3, random_state=4)

# Make predictions on the test set using the loaded model
y_pred2 = ceftriaxone_rf_model.predict(X_test2)

# Calculate the metrics using the test data and predictions
accuracy2 = accuracy_score(y_test2, y_pred2)
precision2 = precision_score(y_test2, y_pred2)
recall2 = recall_score(y_test2, y_pred2)

# Calculate the ROC curve and AUC
fpr2, tpr2, thresholds2 = roc_curve(y_test2, y_pred2)
auc2 = roc_auc_score(y_test2, y_pred2)

# Calculate the MCC
mcc2 = matthews_corrcoef(y_test2, y_pred2)

# Total number of samples in the dataset
num_resistant2 = data2['Ceftriaxone_Resistance'].value_counts()[1]
num_susceptible2 = data2['Ceftriaxone_Resistance'].value_counts()[0]
total_samples2 = len(data2)

# Number of samples in the training set
num_train_samples2 = len(X_train2)

# Number of samples in the testing set
num_test_samples2 = len(X_test2)

# Plot the ROC curve
fig, ax = plt.subplots()
ax.plot(fpr2, tpr2, label='ROC curve (area = %.2f)' % auc2)
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
ax.set_title('ROC curve')
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.grid(True)
ax.legend()
plt.savefig('static/ceftriaxone_roc_curve.png')  # Save the figure as a file
plt.close(fig)  # Close the figure

# Generate confusion matrix
cm2 = confusion_matrix(y_test2, y_pred2)
fig, ax = plt.subplots()
sns.heatmap(cm2, annot=True, fmt='g', cmap='Blues', ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.savefig('static/ceftriaxone_confusion_matrix.png')  # Save the figure as a file
plt.close(fig)  # Close the figure


# ----- Ciprofloxacin -----
# Split the data into training and testing sets
X_train3, X_test3, y_train3, y_test3 = train_test_split(features_data, ciprofloxacin_target,
     test_size=0.3, random_state=72)

# Make predictions on the test set using the loaded model
y_pred3 = ciprofloxacin_rf_model.predict(X_test3)

# Calculate the metrics using the test data and predictions
accuracy3 = accuracy_score(y_test3, y_pred3)
precision3 = precision_score(y_test3, y_pred3)
recall3 = recall_score(y_test3, y_pred3)

# Calculate the ROC curve and AUC
fpr3, tpr3, thresholds3 = roc_curve(y_test3, y_pred3)
auc3 = roc_auc_score(y_test3, y_pred3)

# Calculate the MCC
mcc3 = matthews_corrcoef(y_test3, y_pred3)

# Total number of samples in the dataset
num_resistant3 = target['Ciprofloxacin_Resistance'].value_counts()[1]
num_susceptible3 = target['Ciprofloxacin_Resistance'].value_counts()[0]
total_samples3 = len(ciprofloxacin_target)

# Number of samples in the training set
num_train_samples3 = len(X_train3)

# Number of samples in the testing set
num_test_samples3 = len(X_test3)
```

```
# Plot the ROC curve
fig, ax = plt.subplots()
ax.plot(fpr3, tpr3, label='ROC curve (area = %.2f)' % auc3)
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
ax.set_title('ROC curve')
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.grid(True)
ax.legend()
plt.savefig('static/ciprofloxacin_roc_curve.png')  # Save the figure as a file
plt.close(fig)  # Close the figure

# Generate confusion matrix
cm3 = confusion_matrix(y_test3, y_pred3)
fig, ax = plt.subplots()
sns.heatmap(cm3, annot=True, fmt='g', cmap='Blues', ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.savefig('static/ciprofloxacin_confusion_matrix.png')  # Save the figure as a file
plt.close(fig)  # Close the figure


# ----- Gentamicin -----
# Split the data into training and testing sets
X_train4, X_test4, y_train4, y_test4 = train_test_split(features_data, gentamicin_target,
     test_size=0.3, random_state=2)

# Make predictions on the test set using the loaded model
y_pred4 = gentamicin_xgboost_model.predict(X_test4)

# Calculate the metrics using the test data and predictions
accuracy4 = accuracy_score(y_test4, y_pred4)
precision4 = precision_score(y_test4, y_pred4)
recall4 = recall_score(y_test4, y_pred4)

# Calculate the ROC curve and AUC
fpr4, tpr4, thresholds4 = roc_curve(y_test4, y_pred4)
auc4 = roc_auc_score(y_test4, y_pred4)

# Calculate the MCC
mcc4 = matthews_corrcoef(y_test4, y_pred4)

# Total number of samples in the dataset
num_resistant4 = target['Gentamicin_Resistance'].value_counts()[1]
num_susceptible4 = target['Gentamicin_Resistance'].value_counts()[0]
total_samples4 = len(gentamicin_target)

# Number of samples in the training set
num_train_samples4 = len(X_train4)

# Number of samples in the testing set
num_test_samples4 = len(X_test4)

# Plot the ROC curve
fig, ax = plt.subplots()
ax.plot(fpr4, tpr4, label='ROC curve (area = %.2f)' % auc4)
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
ax.set_title('ROC curve')
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.grid(True)
ax.legend()
plt.savefig('static/gentamicin_roc_curve.png')  # Save the figure as a file
plt.close(fig)  # Close the figure

# Generate confusion matrix
cm4 = confusion_matrix(y_test4, y_pred4)
fig, ax = plt.subplots()
sns.heatmap(cm4, annot=True, fmt='g', cmap='Blues', ax=ax)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.savefig('static/gentamicin_confusion_matrix.png')  # Save the figure as a file
plt.close(fig)  # Close the figure


# ----- Levofloxacin -----
# Levofloxacin has missing values, drop the rows
data3 = pd.concat([features_data, levofloxacin_target], axis=1)
data3 = data3.dropna(subset=['Levofloxacin_Resistance'])

# Split the data into training and testing sets
X_train5, X_test5, y_train5, y_test5 = train_test_split(data3[features_data.columns], data3['
     Levofloxacin_Resistance'], test_size=0.3, random_state=38)

# Make predictions on the test set using the loaded model
y_pred5 = levofloxacin_rf_model.predict(X_test5)

# Calculate the metrics using the test data and predictions
accuracy5 = accuracy_score(y_test5, y_pred5)
precision5 = precision_score(y_test5, y_pred5)
recall5 = recall_score(y_test5, y_pred5)
```

```
    # Calculate the ROC curve and AUC
    fpr5, tpr5, thresholds5 = roc_curve(y_test5, y_pred5)
    auc5 = roc_auc_score(y_test5, y_pred5)

    # Calculate the MCC
    mcc5 = matthews_corrcoef(y_test5, y_pred5)

    # Total number of samples in the dataset
    num_resistant5 = data3['Levofloxacin_Resistance'].value_counts()[1]
    num_susceptible5 = data3['Levofloxacin_Resistance'].value_counts()[0]
    total_samples5 = len(data3)

    # Number of samples in the training set
    num_train_samples5 = len(X_train5)

    # Number of samples in the testing set
    num_test_samples5 = len(X_test5)

    # Plot the ROC curve
    fig, ax = plt.subplots()
    ax.plot(fpr5, tpr5, label='ROC curve (area = %.2f)' % auc5)
    ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
    ax.set_title('ROC curve')
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.grid(True)
    ax.legend()
    plt.savefig('static/levofloxacin_roc_curve.png')  # Save the figure as a file
    plt.close(fig)  # Close the figure

    # Generate confusion matrix
    cm5 = confusion_matrix(y_test5, y_pred5)
    fig, ax = plt.subplots()
    sns.heatmap(cm5, annot=True, fmt='g', cmap='Blues', ax=ax)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
    plt.savefig('static/levofloxacin_confusion_matrix.png')  # Save the figure as a file
    plt.close(fig)  # Close the figure


    # Pass the metrics scores to the template
    context = {
        'accuracy1': accuracy1, 'precision1': precision1, 'recall1': recall1, 'mcc1':mcc1, 'auc1':
            auc1, 'num_test_samples1':num_test_samples1, 'num_train_samples1':num_train_samples1,
            'num_resistant1':num_resistant1,'num_susceptible1':num_susceptible1, 'total_samples1':
            total_samples1,
        'accuracy2': accuracy2, 'precision2': precision2, 'recall2': recall2, 'mcc2':mcc2, 'auc2':
            auc2, 'num_test_samples2':num_test_samples2, 'num_train_samples2':num_train_samples2,
            'num_resistant2':num_resistant2,'num_susceptible2':num_susceptible2, 'total_samples2':
            total_samples2,
        'accuracy3': accuracy3, 'precision3': precision3, 'recall3': recall3, 'mcc3':mcc3, 'auc3':
            auc3, 'num_test_samples3':num_test_samples3, 'num_train_samples3':num_train_samples3,
            'num_resistant3':num_resistant3,'num_susceptible3':num_susceptible3, 'total_samples3':
            total_samples3,
        'accuracy4': accuracy4, 'precision4': precision4, 'recall4': recall4, 'mcc4':mcc4, 'auc4':
            auc4, 'num_test_samples4':num_test_samples4, 'num_train_samples4':num_train_samples4,
            'num_resistant4':num_resistant4,'num_susceptible4':num_susceptible4, 'total_samples4':
            total_samples4,
        'accuracy5': accuracy5, 'precision5': precision5, 'recall5': recall5, 'mcc5':mcc5, 'auc5':
            auc5, 'num_test_samples5':num_test_samples5, 'num_train_samples5':num_train_samples5,
            'num_resistant5':num_resistant5,'num_susceptible5':num_susceptible5, 'total_samples5':
            total_samples5
    }
    return render(request, 'modeldetails.html', context)

def handle_uploaded_file(uploaded_file):
    """
    Save the uploaded file to the MEDIA_ROOT directory with a unique name.
    Return the filename.
    """
    filename = uploaded_file.name
    file_path = os.path.join(settings.MEDIA_ROOT, filename)

    # Write the uploaded file data to the new file
    with open(file_path, 'wb+') as destination:
        for chunk in uploaded_file.chunks():
            destination.write(chunk)

    return filename
```

FILE: forms.py

```
from django import forms

class FastaUploadForm(forms.Form):
    fasta_file = forms.FileField(label='Upload Fasta File', widget=forms.ClearableFileInput(attrs
        ={'multiple': True, 'class':'form-control'}))
```

FILE: models.py

```
from django.db import models

class FastaFile(models.Model):
    title = models.CharField(max_length=255)
    file = models.FileField(upload_to='fasta_files/')

    def __str__(self):
        return self.title
```

FILE: urls.py

```
from django.urls import path
from . import views

app_name = 'dashboard'

urlpatterns = [
    path('', views.about, name='about'),
    path('demo/', views.demo, name='demo'),
    path('antibiogram/', views.antibiogram, name='antibiogram'),
    path('modeldetails/', views.modeldetails, name='modeldetails'),
]
```

FILE: ml_models_cefotaxime.py

```
#!/usr/bin/env python
# coding: utf-8

# EDA packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# package for reading and writing excel files
# pip install openpyxl

a_baumannii = pd.read_excel("acinetobacter_dataset.xlsx")
e_coli = pd.read_excel("escherichia_dataset.xlsx")
k_pneumoniae = pd.read_excel("klebsiella_dataset.xlsx")

# Concatenate the three datasets row-wise
combined_df = pd.concat([a_baumannii, e_coli, k_pneumoniae], ignore_index=True)
combined_df.to_excel("combined_dataset.xlsx")

# Check missing values
combined_df.isnull().sum()

# Basic information / check the datatypes
combined_df.info()


# Plot the unique values
# Get the value counts of the Integron_Presence column
integron_presence_counts = combined_df['Integron_Presence'].value_counts()
print(integron_presence_counts)

# plot the bar chart
sns.barplot(x=integron_presence_counts.index, y=integron_presence_counts)
plt.show()

# Plot the unique values
# Get the value counts of the Cefotaxime_Resistance column
cefotaxime_resistance_counts = combined_df['Cefotaxime_Resistance'].value_counts()
print(cefotaxime_resistance_counts)

# plot the bar chart
sns.barplot(x=cefotaxime_resistance_counts.index, y=cefotaxime_resistance_counts)
plt.show()

count = ((combined_df['Cefotaxime_Resistance'] == 1) & (combined_df['Integron_Presence'] == 1)).
    sum()
print("Number of samples with Cefotaxime_Resistance = 1 and Integron_Presence = 1:", count)

count = ((combined_df['Cefotaxime_Resistance'] == 1) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Cefotaxime_Resistance = 1 and Integron_Presence = 0:", count)

count = ((combined_df['Ceftriaxone_Resistance'] == 1) & (combined_df['Integron_Presence'] == 1)).
    sum()
print("Number of samples with Ceftriaxone_Resistance = 1 and Integron_Presence = 1:", count)

count = ((combined_df['Ceftriaxone_Resistance'] == 1) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Ceftriaxone_Resistance = 1 and Integron_Presence = 0:", count)

count = ((combined_df['Ciprofloxacin_Resistance'] == 1) & (combined_df['Integron_Presence'] == 1))
    .sum()
print("Number of samples with Ciprofloxacin_Resistance = 1 and Integron_Presence = 1:", count)

count = ((combined_df['Ciprofloxacin_Resistance'] == 1) & (combined_df['Integron_Presence'] == 0))
    .sum()
print("Number of samples with Ciprofloxacin_Resistance = 1 and Integron_Presence = 0:", count)

count = ((combined_df['Gentamicin_Resistance'] == 1) & (combined_df['Integron_Presence'] == 1)).
    sum()
```

```
print("Number of samples with Gentamicin_Resistance = 1 and Integron_Presence = 1:", count)

count = ((combined_df['Gentamicin_Resistance'] == 1) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Gentamicin_Resistance = 1 and Integron_Presence = 0:", count)

count = ((combined_df['Levofloxacin_Resistance'] == 1) & (combined_df['Integron_Presence'] == 1)).
    sum()
print("Number of samples with Levofloxacin_Resistance = 1 and Integron_Presence = 1:", count)

count = ((combined_df['Levofloxacin_Resistance'] == 1) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Levofloxacin_Resistance = 1 and Integron_Presence = 0:", count)

count = ((combined_df['Cefotaxime_Resistance'] == 0) & (combined_df['Integron_Presence'] == 1)).
    sum()
print("Number of samples with Cefotaxime_Resistance = 0 and Integron_Presence = 1:", count)

count = ((combined_df['Cefotaxime_Resistance'] == 0) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Cefotaxime_Resistance = 0 and Integron_Presence = 0:", count)

count = ((combined_df['Ceftriaxone_Resistance'] == 0) & (combined_df['Integron_Presence'] == 1)).
    sum()
print("Number of samples with Ceftriaxone_Resistance = 0 and Integron_Presence = 1:", count)

count = ((combined_df['Ceftriaxone_Resistance'] == 0) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Ceftriaxone_Resistance = 0 and Integron_Presence = 0:", count)

count = ((combined_df['Ciprofloxacin_Resistance'] == 0) & (combined_df['Integron_Presence'] == 1))
    .sum()
print("Number of samples with Ciprofloxacin_Resistance = 0 and Integron_Presence = 1:", count)

count = ((combined_df['Ciprofloxacin_Resistance'] == 0) & (combined_df['Integron_Presence'] == 0))
    .sum()
print("Number of samples with Ciprofloxacin_Resistance = 0 and Integron_Presence = 0:", count)

count = ((combined_df['Gentamicin_Resistance'] == 0) & (combined_df['Integron_Presence'] == 1)).
    sum()
print("Number of samples with Gentamicin_Resistance = 0 and Integron_Presence = 1:", count)

count = ((combined_df['Gentamicin_Resistance'] == 0) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Gentamicin_Resistance = 0 and Integron_Presence = 0:", count)

count = ((combined_df['Levofloxacin_Resistance'] == 0) & (combined_df['Integron_Presence'] == 1)).
    sum()
print("Number of samples with Levofloxacin_Resistance = 0 and Integron_Presence = 1:", count)

count = ((combined_df['Levofloxacin_Resistance'] == 0) & (combined_df['Integron_Presence'] == 0)).
    sum()
print("Number of samples with Levofloxacin_Resistance = 0 and Integron_Presence = 0:", count)

# Unique values in Gene_Cassette_Array
# Split the AMR genes into separate columns
combined_df['Gene_Cassette_Array'] = combined_df['Gene_Cassette_Array'].astype(str)
combined_df['Gene_Cassette_Array'] = combined_df['Gene_Cassette_Array'].str.split(';')
combined_df = combined_df.explode('Gene_Cassette_Array')

# Reset the index
combined_df.reset_index(drop=True, inplace=True)

combined_df['Gene_Cassette_Array'].unique()

pd.set_option('display.max_rows', 500)
# Plot the unique values
# Get the value counts of the Gene_Cassette_Array column
gene_cassettes_counts = combined_df['Gene_Cassette_Array'].value_counts()
print(gene_cassettes_counts)

# plot the bar chart
plt.figure(figsize=(17, 6))
sns.barplot(x=gene_cassettes_counts.index, y=gene_cassettes_counts, dodge=True)
plt.xticks(rotation=90)
plt.show()

# putting the dataset to original form
# Concatenate the three datasets row-wise
combined_df = pd.concat([a_baumannii, e_coli, k_pneumoniae], ignore_index=True)
combined_df

# Machine Learning Algorithms

from sklearn.preprocessing import MultiLabelBinarizer
# Separate features and target variable

# Exclude non-relevant columns and the target variable
features = combined_df.drop(["Accession_no", "Bacteria_Species", "Cefotaxime_Resistance", "
    Ceftriaxone_Resistance", "Ciprofloxacin_Resistance", "Gentamicin_Resistance", "
    Levofloxacin_Resistance"], axis=1)
target = combined_df['Cefotaxime_Resistance']
```

```
# Split the gene names in the Gene_Cassette_Array column
features['Gene_Cassette_Array'] = features['Gene_Cassette_Array'].str.split(';')

mlb = MultiLabelBinarizer()
encoded_features = pd.DataFrame(mlb.fit_transform(features['Gene_Cassette_Array']), columns=mlb.
     classes_)
# Rename the encoded feature columns with the gene names
encoded_features.columns = mlb.classes_

# Concatenate the encoded features with the remaining columns
processed_data = pd.concat([encoded_features, features.drop(['Gene_Cassette_Array'], axis=1)],
     axis=1)
processed_data.to_excel("processed_data.xlsx")

# ------------------------------------ XGBoost (Cefotaxime) ------------------------------------

# Drop rows with missing values in the target variable
data = pd.concat([processed_data, target], axis=1)
data = data.dropna(subset=['Cefotaxime_Resistance'])

data

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
import xgboost as xgb

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Cefotaxime_Resistance'], test_size=0.3, random_state=6)

# Train the XGBoost model
model = xgb.XGBClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Cefotaxime_Resistance'], test_size=0.3, random_state=6)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the XGBoost model on the resampled data
model = xgb.XGBClassifier()
model.fit(X_train_resampled, y_train_resampled)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()


# ----------------------------- Support Vector Machine (Cefotaxime) -----------------------------


from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Cefotaxime_Resistance'], test_size=0.3, random_state=6)

# Train the SVM model
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()
```

```
# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Cefotaxime_Resistance'], test_size=0.3, random_state=6)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the SVM model on the resampled data
svm_model = SVC()
svm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

import joblib
joblib.dump(svm_model, 'svm_model.pkl')

# ---------------------------- Random Forest (Cefotaxime) ----------------------------

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Cefotaxime_Resistance'], test_size=0.3, random_state=6)

# Train the Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

```
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
    Cefotaxime_Resistance'], test_size=0.3, random_state=6)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

model = RandomForestClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

FILE: ml_models_ceftriaxone.py

```
#!/usr/bin/env python
# coding: utf-8

# EDA packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# package for reading and writing excel files
# pip install openpyxl

a_baumannii = pd.read_excel("acinetobacter_dataset.xlsx")
e_coli = pd.read_excel("escherichia_dataset.xlsx")
k_pneumoniae = pd.read_excel("klebsiella_dataset.xlsx")

# Concatenate the three datasets row-wise
combined_df = pd.concat([a_baumannii, e_coli, k_pneumoniae], ignore_index=True)
#combined_df.to_excel("combined_dataset.xlsx")

# Check missing values
combined_df.isnull().sum()

# Basic information / check the datatypes
combined_df.info()

# Plot the unique values
# Get the value counts of the Integron_Presence column
integron_presence_counts = combined_df['Integron_Presence'].value_counts()
print(integron_presence_counts)

# plot the bar chart
sns.barplot(x=integron_presence_counts.index, y=integron_presence_counts)
plt.show()

# Plot the unique values
# Get the value counts of the Ceftriaxone_Resistance column
ceftriaxone_resistance_counts = combined_df['Ceftriaxone_Resistance'].value_counts()
print(ceftriaxone_resistance_counts)

# plot the bar chart
sns.barplot(x=ceftriaxone_resistance_counts.index, y=ceftriaxone_resistance_counts)
plt.show()

# Machine Learning Algorithms

from sklearn.preprocessing import MultiLabelBinarizer
# Separate features and target variable

# Exclude non-relevant columns and the target variable
features = combined_df.drop(["Accession_no", "Bacteria_Species", "Cefotaxime_Resistance", "
     Ceftriaxone_Resistance", "Ciprofloxacin_Resistance", "Gentamicin_Resistance", "
     Levofloxacin_Resistance"], axis=1)
target = combined_df['Ceftriaxone_Resistance']

# Split the gene names in the Gene_Cassette_Array column
features['Gene_Cassette_Array'] = features['Gene_Cassette_Array'].str.split(';')

mlb = MultiLabelBinarizer()
encoded_features = pd.DataFrame(mlb.fit_transform(features['Gene_Cassette_Array']), columns=mlb.
     classes_)
# Rename the encoded feature columns with the gene names
encoded_features.columns = mlb.classes_

# Concatenate the encoded features with the remaining columns
processed_data = pd.concat([encoded_features, features.drop(['Gene_Cassette_Array'], axis=1)],
     axis=1)
#processed_data.to_excel("processed_data.xlsx")

# ------------------- XGBoost (Ceftriaxone) -------------------

# Drop rows with missing values in the target variable
data = pd.concat([processed_data, target], axis=1)
data = data.dropna(subset=['Ceftriaxone_Resistance'])

data

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
import xgboost as xgb

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Ceftriaxone_Resistance'], test_size=0.3, random_state=14)

# Train the XGBoost model
model = xgb.XGBClassifier()
model.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
    Ceftriaxone_Resistance'], test_size=0.3, random_state=4)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the XGBoost model on the resampled data
model = xgb.XGBClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```
# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()


# ---------------------------- Support Vector Machine (Ceftriaxone) ----------------------------

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Ceftriaxone_Resistance'], test_size=0.3, random_state=22)

# Train the SVM model
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Ceftriaxone_Resistance'], test_size=0.3, random_state=24)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the SVM model on the resampled data
svm_model = SVC()
svm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)
```

```
# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

import joblib
joblib.dump(svm_model, 'svm_model.pkl')

# ---------------------------- Random Forest (Ceftriaxone) ----------------------------

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
    Ceftriaxone_Resistance'], test_size=0.3, random_state=42)

# Train the Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
    Ceftriaxone_Resistance'], test_size=0.3, random_state=4)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

model = RandomForestClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

FILE: ml_models_ciprofloxacin.py

```
#!/usr/bin/env python
# coding: utf-8

# EDA packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# package for reading and writing excel files
# pip install openpyxl

a_baumannii = pd.read_excel("acinetobacter_dataset.xlsx")
e_coli = pd.read_excel("escherichia_dataset.xlsx")
k_pneumoniae = pd.read_excel("klebsiella_dataset.xlsx")

# Concatenate the three datasets row-wise
combined_df = pd.concat([a_baumannii, e_coli, k_pneumoniae], ignore_index=True)

# Check missing values
combined_df.isnull().sum()

# Basic information / check the datatypes
combined_df.info()

# Plot the unique values
# Get the value counts of the Integron_Presence column
integron_presence_counts = combined_df['Integron_Presence'].value_counts()
print(integron_presence_counts)

# plot the bar chart
sns.barplot(x=integron_presence_counts.index, y=integron_presence_counts)
plt.show()

# Plot the unique values
# Get the value counts of the Ciprofloxacin_Resistance column
ciprofloxacin_resistance_counts = combined_df['Ciprofloxacin_Resistance'].value_counts()
print(ciprofloxacin_resistance_counts)

# plot the bar chart
sns.barplot(x=ciprofloxacin_resistance_counts.index, y=ciprofloxacin_resistance_counts)
plt.show()


# Machine Learning Algorithms

from sklearn.preprocessing import MultiLabelBinarizer
# Separate features and target variable

# Exclude non-relevant columns and the target variable
features = combined_df.drop(["Accession_no", "Bacteria_Species", "Cefotaxime_Resistance", "
     Ceftriaxone_Resistance", "Ciprofloxacin_Resistance", "Gentamicin_Resistance", "
     Levofloxacin_Resistance"], axis=1)
target = combined_df['Ciprofloxacin_Resistance']

# Split the gene names in the Gene_Cassette_Array column
features['Gene_Cassette_Array'] = features['Gene_Cassette_Array'].str.split(';')

mlb = MultiLabelBinarizer()
encoded_features = pd.DataFrame(mlb.fit_transform(features['Gene_Cassette_Array']), columns=mlb.
     classes_)
# Rename the encoded feature columns with the gene names
encoded_features.columns = mlb.classes_

# Concatenate the encoded features with the remaining columns
processed_data = pd.concat([encoded_features, features.drop(['Gene_Cassette_Array'], axis=1)],
     axis=1)

# -------------------- XGBoost (Ciprofloxacin) --------------------

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
import xgboost as xgb

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
     random_state=69)

# Train the XGBoost model
model = xgb.XGBClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# using smote
from imblearn.over_sampling import SMOTE
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
     random_state=11)

# Apply SMOTE to the training data
smote = SMOTE(random_state=5)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the XGBoost model on the resampled data
model = xgb.XGBClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ----------------------------- Support Vector Machine (Ciprofloxacin) -----------------------------

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
     random_state=72)

# Train the SVM model
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()


from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
    roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
    random_state=72)

# Apply SMOTE to the training data
smote = SMOTE(random_state=5)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the SVM model on the resampled data
svm_model = SVC()
svm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ------------------------------ Random Forest (Cefotaxime) ------------------------------

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
    random_state=47)

# Train the Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
    random_state=72)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

model = RandomForestClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

FILE: ml_models_gentamicin.py

```
#!/usr/bin/env python
# coding: utf-8

# EDA packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# package for reading and writing excel files
# pip install openpyxl

a_baumannii = pd.read_excel("acinetobacter_dataset.xlsx")
e_coli = pd.read_excel("escherichia_dataset.xlsx")
k_pneumoniae = pd.read_excel("klebsiella_dataset.xlsx")

# Concatenate the three datasets row-wise
combined_df = pd.concat([a_baumannii, e_coli, k_pneumoniae], ignore_index=True)

# Check missing values
combined_df.isnull().sum()

# Basic information / check the datatypes
combined_df.info()

# Plot the unique values
# Get the value counts of the Integron_Presence column
integron_presence_counts = combined_df['Integron_Presence'].value_counts()
print(integron_presence_counts)

# plot the bar chart
sns.barplot(x=integron_presence_counts.index, y=integron_presence_counts)
plt.show()

# Plot the unique values
# Get the value counts of the Gentamicin_Resistance column
gentamicin_resistance_counts = combined_df['Gentamicin_Resistance'].value_counts()
print(ciprofloxacin_resistance_counts)

# plot the bar chart
sns.barplot(x=gentamicin_resistance_counts.index, y=gentamicin_resistance_counts)
plt.show()

# Machine Learning Algorithms

from sklearn.preprocessing import MultiLabelBinarizer
# Separate features and target variable

# Exclude non-relevant columns and the target variable
features = combined_df.drop(["Accession_no", "Bacteria_Species", "Cefotaxime_Resistance", "
     Ceftriaxone_Resistance", "Ciprofloxacin_Resistance", "Gentamicin_Resistance", "
     Levofloxacin_Resistance"], axis=1)
target = combined_df['Gentamicin_Resistance']

# Split the gene names in the Gene_Cassette_Array column
features['Gene_Cassette_Array'] = features['Gene_Cassette_Array'].str.split(';')

mlb = MultiLabelBinarizer()
encoded_features = pd.DataFrame(mlb.fit_transform(features['Gene_Cassette_Array']), columns=mlb.
     classes_)
# Rename the encoded feature columns with the gene names
encoded_features.columns = mlb.classes_

# Concatenate the encoded features with the remaining columns
processed_data = pd.concat([encoded_features, features.drop(['Gene_Cassette_Array'], axis=1)],
     axis=1)

# ------------------- XGBoost (Gentamicin) -------------------

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
import xgboost as xgb

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
     random_state=2)

# Train the XGBoost model
```

```
model = xgb.XGBClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
     random_state=2)

# Apply SMOTE to the training data
smote = SMOTE(random_state=8)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the XGBoost model on the resampled data
model = xgb.XGBClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()
```

```
# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# ----------------------------- Support Vector Machine (Gentamicin) -----------------------------

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
    roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
    random_state=1)

# Train the SVM model
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
    roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
    random_state=2)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the XGBoost model on the resampled data
svm_model = SVC()
svm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)
```

```
# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

import joblib
joblib.dump(svm_model, 'svm_model.pkl')

# ------------------------------ Random Forest (Gentamicin) ------------------------------

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
     random_state=2)

# Train the Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
```

```
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(processed_data, target, test_size=0.3,
    random_state=2)

# Apply SMOTE to the training data
smote = SMOTE(random_state=8)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

model = RandomForestClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

FILE: ml_models_levofloxacin.py

```
#!/usr/bin/env python
# coding: utf-8

# EDA packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# package for reading and writing excel files
# pip install openpyxl

a_baumannii = pd.read_excel("acinetobacter_dataset.xlsx")
e_coli = pd.read_excel("escherichia_dataset.xlsx")
k_pneumoniae = pd.read_excel("klebsiella_dataset.xlsx")

# Concatenate the three datasets row-wise
combined_df = pd.concat([a_baumannii, e_coli, k_pneumoniae], ignore_index=True)
#combined_df.to_excel("combined_dataset.xlsx")

# Check missing values
combined_df.isnull().sum()

# Basic information / check the datatypes
combined_df.info()

# Plot the unique values
# Get the value counts of the Integron_Presence column
integron_presence_counts = combined_df['Integron_Presence'].value_counts()
print(integron_presence_counts)
```

```
# plot the bar chart
sns.barplot(x=integron_presence_counts.index, y=integron_presence_counts)
plt.show()

# Plot the unique values
# Get the value counts of the Levofloxacin_Resistance column
levofloxacin_resistance_counts = combined_df['Levofloxacin_Resistance'].value_counts()
print(levofloxacin_resistance_counts)

# plot the bar chart
sns.barplot(x=levofloxacin_resistance_counts.index, y=levofloxacin_resistance_counts)
plt.show()

# Machine Learning Algorithms

from sklearn.preprocessing import MultiLabelBinarizer
# Separate features and target variable

# Exclude non-relevant columns and the target variable
features = combined_df.drop(["Accession_no", "Bacteria_Species", "Cefotaxime_Resistance", "
     Ceftriaxone_Resistance", "Ciprofloxacin_Resistance", "Gentamicin_Resistance", "
     Levofloxacin_Resistance"], axis=1)
target = combined_df['Levofloxacin_Resistance']

# Split the gene names in the Gene_Cassette_Array column
features['Gene_Cassette_Array'] = features['Gene_Cassette_Array'].str.split(';')

mlb = MultiLabelBinarizer()
encoded_features = pd.DataFrame(mlb.fit_transform(features['Gene_Cassette_Array']), columns=mlb.
     classes_)
# Rename the encoded feature columns with the gene names
encoded_features.columns = mlb.classes_

# Concatenate the encoded features with the remaining columns
processed_data = pd.concat([encoded_features, features.drop(['Gene_Cassette_Array'], axis=1)],
     axis=1)
#processed_data.to_excel("processed_data.xlsx")

# ------------------ XGBoost (Levofloxacin) ------------------

# Drop rows with missing values in the target variable
data = pd.concat([processed_data, target], axis=1)
data = data.dropna(subset=['Levofloxacin_Resistance'])

data

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
import xgboost as xgb

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Levofloxacin_Resistance'], test_size=0.3, random_state=3)

# Train the XGBoost model
model = xgb.XGBClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()
```

```
# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Levofloxacin_Resistance'], test_size=0.3, random_state=38)

# Apply SMOTE to the training data
smote = SMOTE(random_state=4)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the XGBoost model on the resampled data
model = xgb.XGBClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
# ---------------------------- Support Vector Machine (Levofloxacin ----------------------------

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Levofloxacin_Resistance'], test_size=0.3, random_state=4)

# Train the SVM model
svm_model = SVC()
svm_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve,
     roc_auc_score, matthews_corrcoef, confusion_matrix
from sklearn.svm import SVC

# using smote
from imblearn.over_sampling import SMOTE

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Levofloxacin_Resistance'], test_size=0.3, random_state=37)

# Apply SMOTE to the training data
smote = SMOTE(random_state=10)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Train the SVM model on the resampled data
svm_model = SVC()
svm_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
import joblib
joblib.dump(svm_model, 'svm_model.pkl')

# ------------------------------ Random Forest (Levofloxacin) ------------------------------

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Levofloxacin_Resistance'], test_size=0.3, random_state=38)

# Train the Random Forest model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

from sklearn.ensemble import RandomForestClassifier

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data[processed_data.columns], data['
     Levofloxacin_Resistance'], test_size=0.3, random_state=4)

# Apply SMOTE to the training data
smote = SMOTE(random_state=3)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

model = RandomForestClassifier()
model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate the ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred)

# Calculate the MCC
mcc = matthews_corrcoef(y_test, y_pred)

# Calculate the classification metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

# Print the results
print(f"ROC curve: {fpr}, {tpr}")
print(f"AUC: {auc}")
print(f"MCC: {mcc}")
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = %.2f)' %auc)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix as heatmap
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

XI. Acknowledgment

I am deeply grateful to the Lord for His relentless guidance and blessings that have accompanied me throughout this SP journey. His constant support and strength have been the foundation of my success.

I extend my sincerest appreciation to my thesis adviser, Doc Geoffrey Solano, for his guidance and encouragement. Your mentorship has profoundly shaped the direction of this work. I am fortunate for your advice during this research endeavor and your support in applying for the NIH Student Research Grant, which we were later awarded.

I would also like to express my gratitude to Sir Zachary Lara for his significant contributions and valuable insights. Your expertise and input have greatly enriched the research and its outcomes. Your assistance has been instrumental, and I am immensely grateful for your support throughout my SP.

I am also grateful to Ma'am Eden for her guidance throughout my UP journey. I will never forget how you assisted me throughout the enrollment process. As a lost T2, your support meant a lot to me.

A special thank you goes to my sister, Lady Edlenill, for her persistent support, encouragement, and understanding. Your belief in my abilities and continuous motivation has served as a constant source of inspiration. Even during my lowest moments, you have been there for me, and I cannot express my gratitude enough. You are truly the reason why I find the strength to persevere.

I also want to express my appreciation to my little brother, Lord Edrichard III. Thank you for being someone with whom I can share my enthusiasm for F1, running, and various other sports. Your companionship and shared interests have made this journey all the more enjoyable.

Finally, I would like to acknowledge all individuals who have played a part, no matter how big or small, in supporting and inspiring me during this thesis and college life. Your presence, encouragement, and belief in my abilities have been integral to my success. Your contributions have undoubtedly shaped my academic and personal growth, and for that, I am truly thankful.