Group members :

Christian RWIBUTSO HAKIZINKA , Abdullah Amin , Shamsun Nahar , Abdelmola Albadwi

Data Preparation/Feature Engineering

1. Overview

Data preparation and feature engineering are critical phases in our machine learning project, "Predicting Antimicrobial Resistance Using Machine Learning for Improved Healthcare Outcomes." These steps ensure that the data is clean, relevant, and in a format suitable for modeling, which ultimately enhances the model's performance and accuracy.

2. Data Collection

The datasets for our project will be sourced from reputable institutions such as the World Health Organization (WHO) and various healthcare research centers. These datasets include demographic, clinical, laboratory, and microbiological data essential for understanding antimicrobial resistance (AMR) patterns. During data collection, preprocessing steps will involve integrating data from multiple sources and standardizing formats.

Link: Global Antimicrobial Resistance and Use Surveillance System (GLASS) Dashboard - Publication - WHO AFRO Health Data Hub

3. Data Cleaning

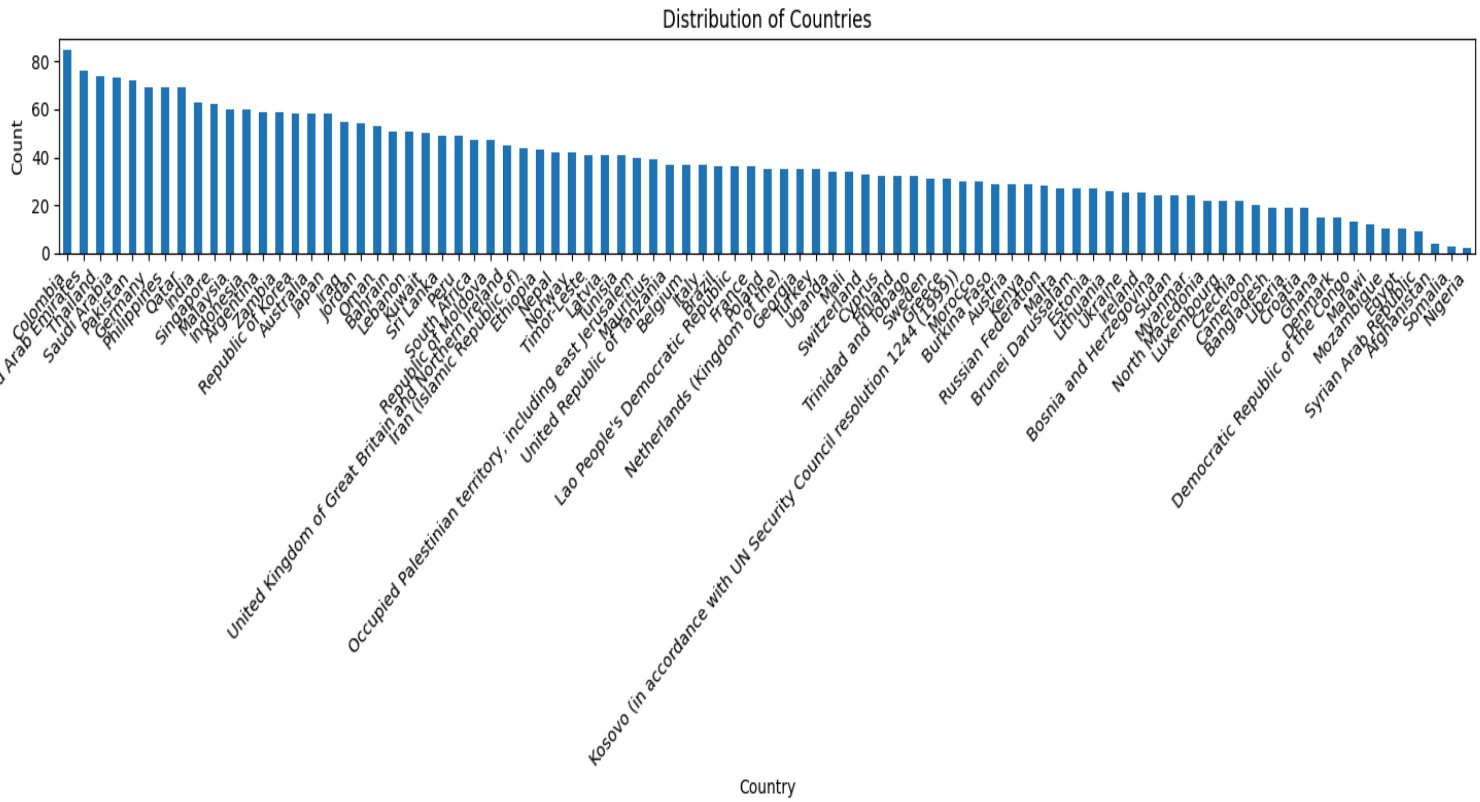
Data cleaning involves several steps:

- **Handling Missing Values**: Imputation techniques will be used to fill in missing data points where appropriate.
- **Outlier Detection and Removal**: Statistical methods and domain knowledge will help identify and manage outliers.
- **Consistency Checks**: Ensuring data consistency across different sources by standardizing units and formats.

4. Exploratory Data Analysis (EDA)

Exploratory data analysis (EDA) will involve:

- **Visualizations**: Using histograms, box plots, and scatter plots to understand data distributions and relationships.
- **Insights**: Identifying trends and patterns in the data, such as common resistance patterns and their correlation with specific antibiotics.

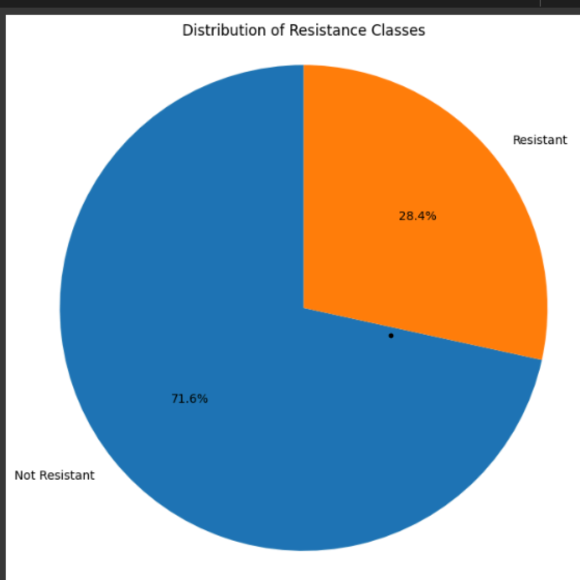


5. Feature Engineering

```python
# prompt: pie chart of resistance_class column

# Calculate the value counts for the 'Resistance_Class' column.
resistance_counts = amr['Resistance_Class'].value_counts()

# Create the pie chart.
plt.figure(figsize=(8, 8))
plt.pie(resistance_counts, labels=resistance_counts.index, autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Resistance Classes')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



6. Data Transformation

```
[56] features =amr[['Specimen','PathogenName','CountryTerritoryArea','AntibioticName','ResistancePercentage','Resistance_Class']]
```

```
[58] features.head()
```

	Specimen	PathogenName	CountryTerritoryArea	AntibioticName	ResistancePercentage	Resistance_Class
0	BLOOD	Escherichia coli	United Arab Emirates	Ampicillin	71.740891	Resistant
1	BLOOD	Escherichia coli	Brunei Darussalam	Ampicillin	53.125000	Resistant
2	BLOOD	Escherichia coli	Pakistan	Ampicillin	95.990966	Resistant
3	BLOOD	Escherichia coli	Iran (Islamic Republic of)	Ampicillin	92.248062	Resistant
4	BLOOD	Escherichia coli	Jordan	Ampicillin	79.166667	Resistant

Next steps: Generate code with features | View recommended plots | New interactive sheet

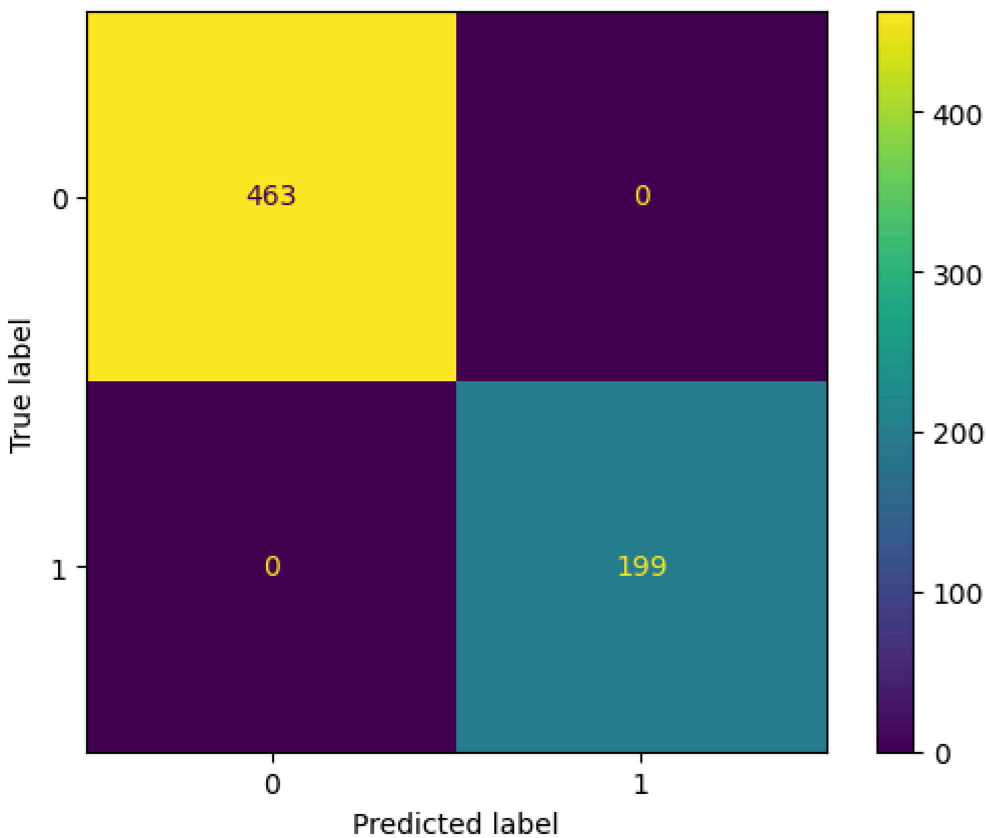
```
[59] from sklearn.preprocessing import LabelEncoder
     # Assuming your DataFrame is named 'features'
     features = features.apply(LabelEncoder().fit_transform)
```

```
[60] import xgboost as xgb
     from sklearn.model_selection import train_test_split
     X = features.drop('Resistance_Class', axis=1)
     y = features['Resistance_Class']
```

```
[ ] Start coding or generate with AI.
```

```
[61] from sklearn.preprocessing import LabelEncoder # import LabelEncoder

     #le = LabelEncoder() # create a LabelEncoder object
     #y = le.fit_transform(y) # fit and transform the target variable
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```



Model Exploration
1. Model Selection

Random Forest: An ensemble method that builds multiple decision trees and combines their predictions to improve accuracy and reduce overfitting. It handles large datasets well and provides feature importance, but can be computationally intensive.

Logistic Regression: A simple, interpretable method for binary classification that estimates probabilities using a logistic function. It's efficient and effective for linearly separable data but may struggle with complex, non-linear relationships.

XGBoost: An advanced boosting technique that builds decision trees sequentially, where each tree corrects errors of the previous ones. It excels in performance and accuracy, handles missing data well, and includes regularization to prevent overfitting.

Choosing Logistic Regression makes sense for your situation due to several key reasons:

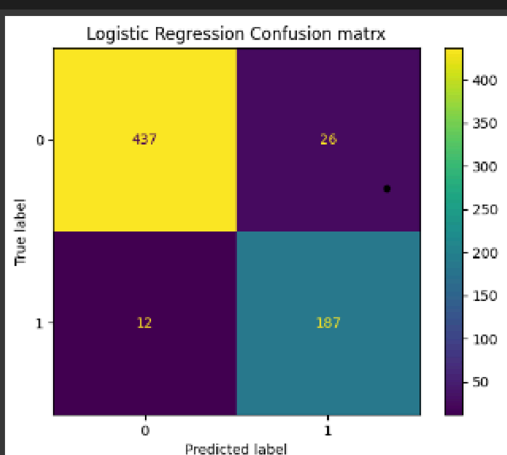
1. **Interpretability**: Logistic Regression provides clear, interpretable results, making it easy to understand and explain how each feature influences the predictions.
2. **Suitability for Small Datasets**: It performs well with smaller datasets because it has fewer parameters, reducing the risk of overfitting and ensuring stable performance.
3. **Good Accuracy**: With an accuracy of 91%, Logistic Regression delivers strong performance, effectively capturing patterns in your data despite its simplicity compared to more complex models.

Logistic regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
# Initialize the Logistic Regression model
logreg_model = LogisticRegression(random_state=42)
# Fit the model to the training data
logreg_model.fit(X_train, y_train)
# Make predictions on the test data
predictions = logreg_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")
cm = confusion_matrix(y_test, predictions)
print(f"Confusion Matrix:\n {cm}")
```

```
Accuracy: 0.9425981873111783
Confusion Matrix:
 [[437  26]
 [ 12 187]]
```

[72]
```
# Assuming you have your confusion matrix in the 'cm' variable
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('Logistic Regression Confusion matrx')
plt.show()
```



4. Code Implementation

```
# prompt: want to create a column from resistant, if resistant is above 50 will
classsify as resistant if else not resistant

amr['Resistance_Class'] = np.where(amr['ResistancePercentage'] > 50, 'Resistant',
'Not Resistant')

# prompt: pie chart of resistance_class column

# Calculate the value counts for the 'Resistance_Class' column.
resistance_counts = amr['Resistance_Class'].value_counts()

# Create the pie chart.
plt.figure(figsize=(8, 8))
plt.pie(resistance_counts, labels=resistance_counts.index, autopct='%1.1f%%',
startangle=90)
plt.title('Distribution of Resistance Classes')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
# prompt: bar chart of column CountryTerritoryArea to know the distribution of
country of this

# Count the occurrences of each country.
country_counts = amr['CountryTerritoryArea'].value_counts()

# Create the bar chart.
plt.figure(figsize=(15, 6))
country_counts.plot(kind='bar')
plt.xlabel('Country')
plt.ylabel('Count')
plt.title('Distribution of Countries')
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()

features
=amr[['Specimen','PathogenName','CountryTerritoryArea','AntibioticName','Resistanc
ePercentage','Resistance_Class']]
from sklearn.preprocessing import LabelEncoder
```

```python
# Assuming your DataFrame is named 'features'
features = features.apply(LabelEncoder().fit_transform)
import xgboost as xgb
from sklearn.model_selection import train_test_split
X = features.drop('Resistance_Class', axis=1)
y = features['Resistance_Class']
from sklearn.preprocessing import LabelEncoder # import LabelEncoder

#le = LabelEncoder() # create a LabelEncoder object
#y = le.fit_transform(y) # fit and transform the target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Initialize the XGBoost model with enable_categorical
model = xgb.XGBClassifier(enable_categorical=True)

# Fit the model to the training data
model.fit(X_train, y_train)
# Assuming you have your trained model in the 'model' variable and your test set
in 'X_test'
predictions = model.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix

# Calculate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")

# Generate confusion matrix
cm = confusion_matrix(y_test, predictions)
print(f"Confusion Matrix:\n {cm}")
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Assuming you have your confusion matrix in the 'cm' variable
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
import pickle

# Save the model
with open("xgboost_model.pkl", "wb") as f:
```

```
    pickle.dump(model, f)
from sklearn.ensemble import RandomForestClassifier
#from sklearn.model_selection import train_test_split
#from sklearn.metrics import accuracy_score, confusion_matrix

# Assuming you have your features in 'X' and target variable in 'y'
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)

# Fit the model to the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test data
predictions = rf_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
# Initialize the Logistic Regression model
logreg_model = LogisticRegression(random_state=42)
# Fit the model to the training data
logreg_model.fit(X_train, y_train)
# Make predictions on the test data
predictions = logreg_model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy}")
cm = confusion_matrix(y_test, predictions)
print(f"Confusion Matrix:\n {cm}")
# Assuming you have your confusion matrix in the 'cm' variable
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.title('Logistic Regression Confusion matrx')
plt.show()
```