Tree Health Prediction

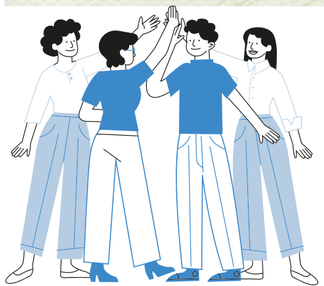
Prepared by: Darsh Patel
Priyanshi Yadav
Krinal Akbari
Dimpal Lad

Darsh Patel
20025221

Priyanshi Yadav
20019419

Krinal Akbari
20018932

Dimpal Lad
20023001



TODAY'S AGENDA

01 Introduction

02 Problem Statement

03 Exploratory Data Analysis

04 Classification Algorithms

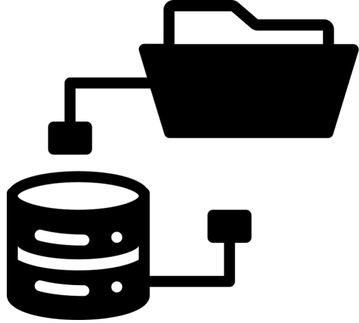
05 Model Conclusion

INTRODUCTION

- Urban trees are vital to city environments, environmental, and social benefits. However, maintaining their health and addressing potential problems can be challenging. The TreesCount! 2015 Street Tree Census, conducted by NYC Parks & Recreation and partners, offers a detailed dataset containing tree species, diameter, and perceived health information.

- This project aims to develop machine learning models to predict urban tree health. Using attributes such as tree diameter, species, location, and stewardship information, we have created models to predict the "health" from the 2015 Street Tree Census dataset. These models will assist urban planners in making informed decisions about tree maintenance and management, ultimately enhancing the health and sustainability of urban tree populations.

Description of Dataset



Source : https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/uvpi-gqnh/about_data

[14]:

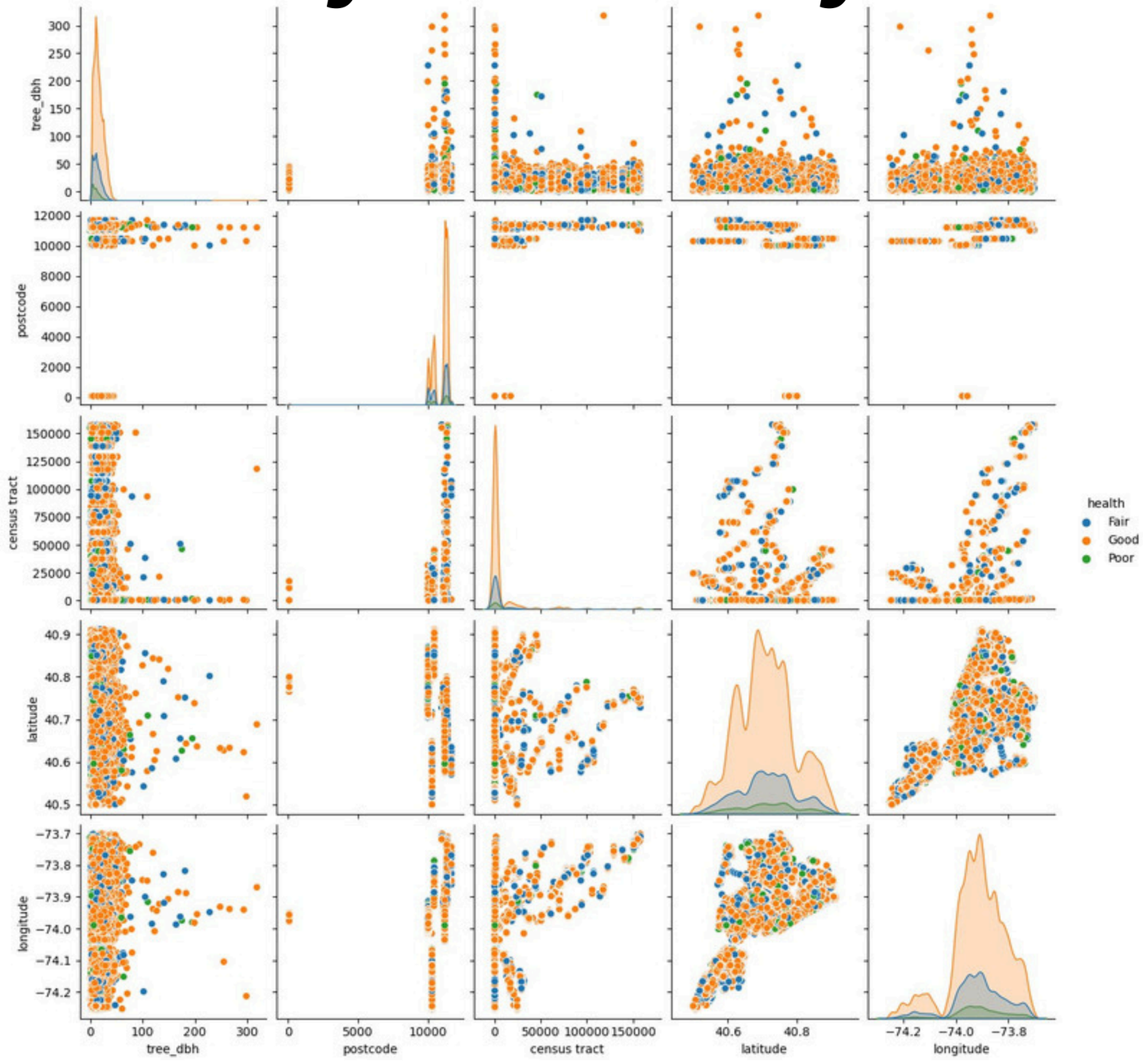
	tree_id	block_id	created_at	tree_dbh	stump_diam	curb_loc	status	health	spc_latin	spc_common	...	boro_ct	state	latitude	longitude	x_sp	
1	200540	315986	09/03/2015	21	0	OnCurb	Alive	Fair	Quercus palustris	pin oak	...	4097300	New York	40.794111	-73.818679	1.034456e+06	22
3	204337	217969	09/05/2015	10	0	OnCurb	Alive	Good	Gleditsia triacanthos var. inermis	honeylocust	...	3044900	New York	40.713537	-73.934456	1.002420e+06	19
4	189565	223043	08/30/2015	21	0	OnCurb	Alive	Good	Tilia americana	American linden	...	3016500	New York	40.666778	-73.975979	9.909138e+05	18
7	208649	103940	09/07/2015	9	0	OnCurb	Alive	Good	Tilia americana	American linden	...	1012700	New York	40.762724	-73.987297	9.877691e+05	21

4 rows × 44 columns

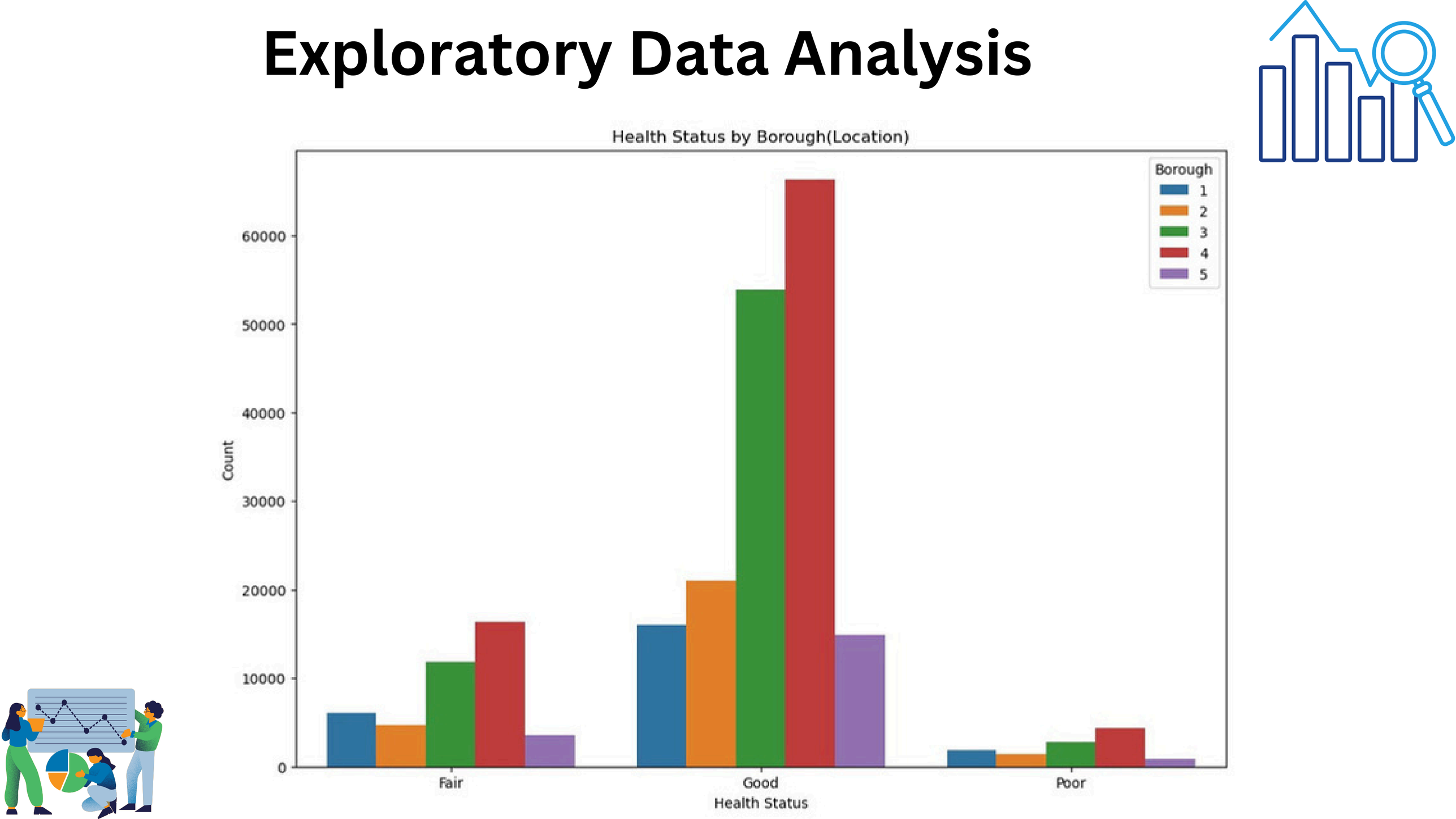
Our dataset consist of :

- **Number of Columns :**44
- **Number of Rows:** 225,844
- **Categorical columns :** 25
- **Numerical columns :** 19
- **Target Variable :** ‘Health’

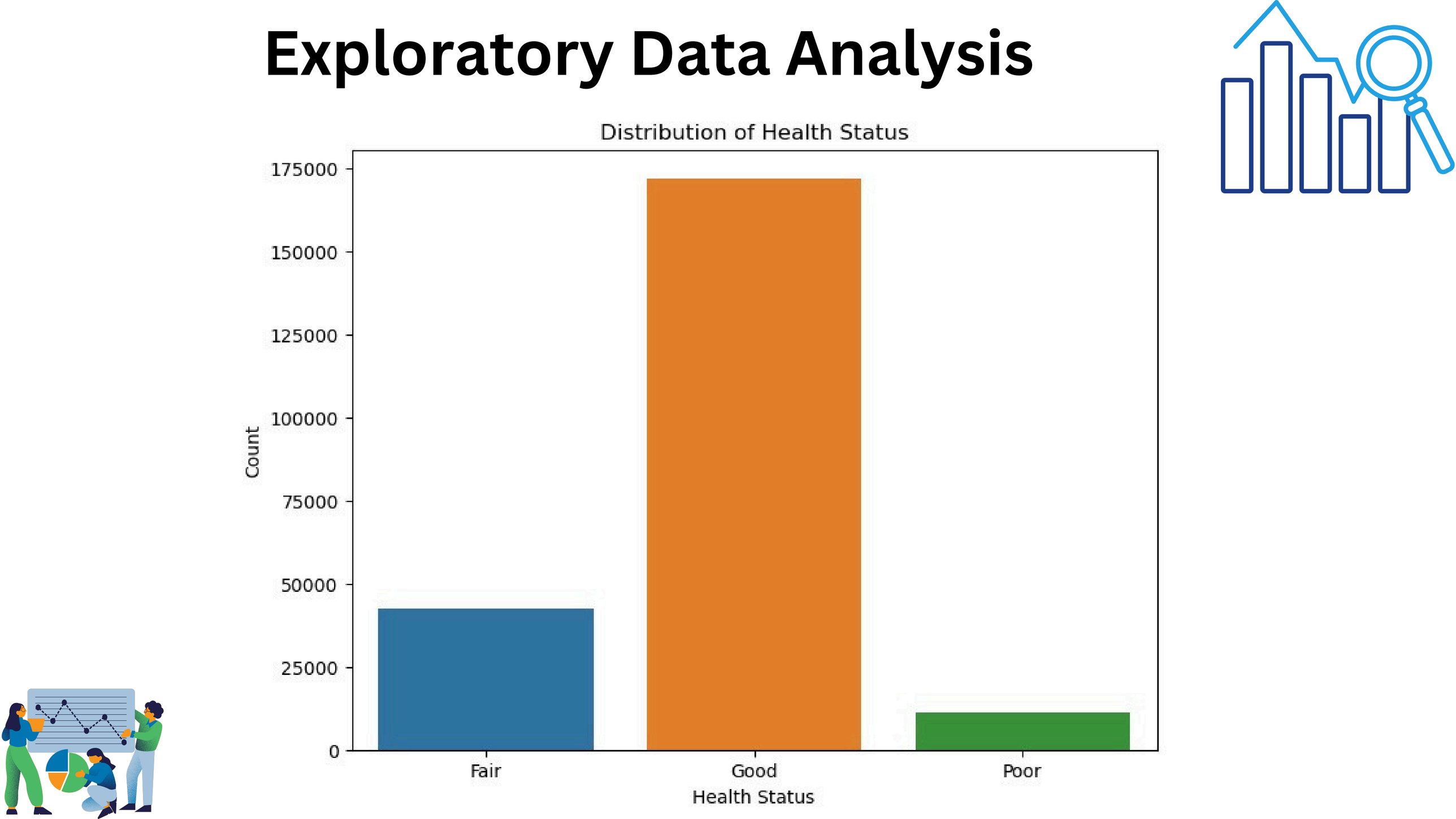
Exploratory Data Analysis



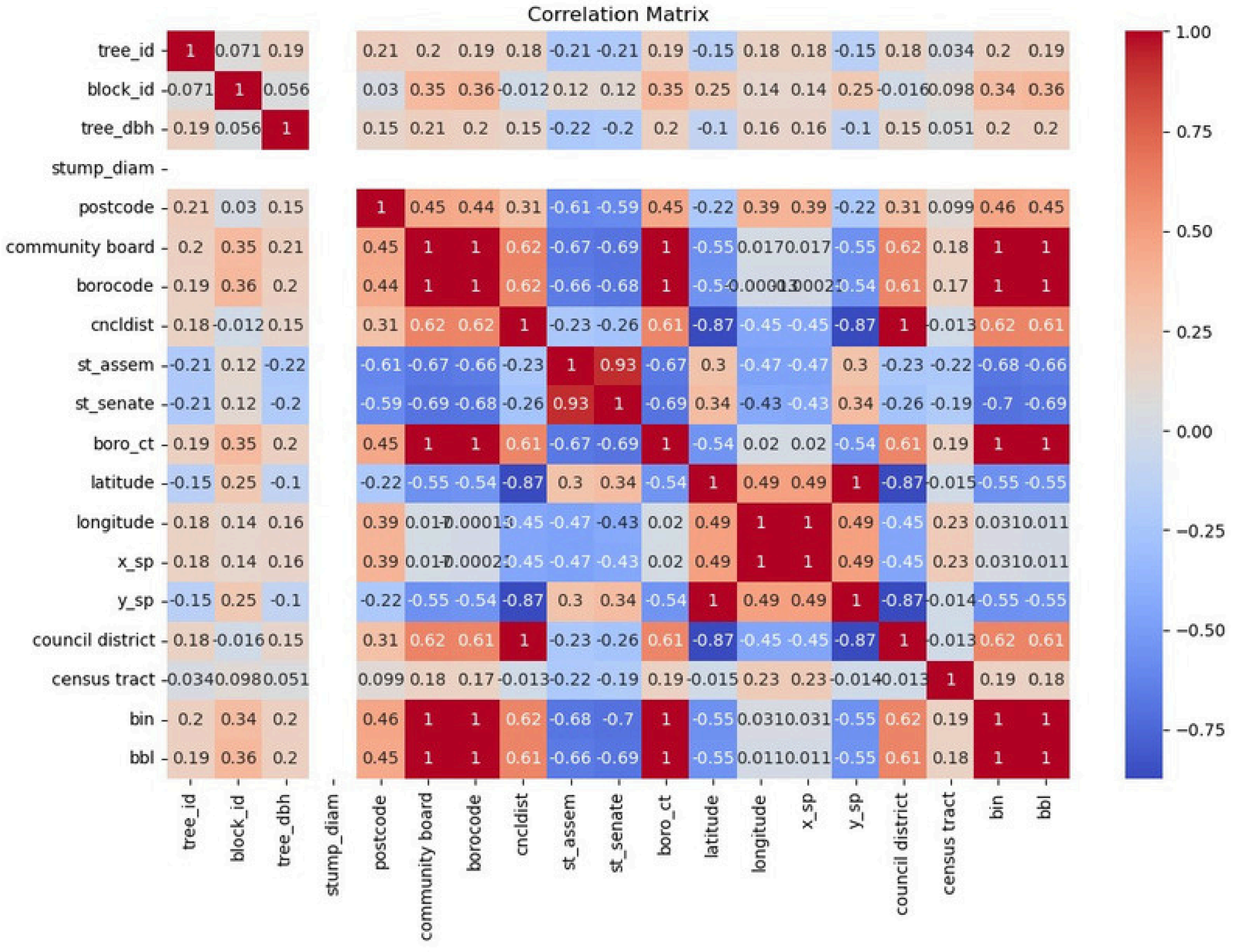
Exploratory Data Analysis



Exploratory Data Analysis



Exploratory Data Analysis



Data Preprocessing Techniques

After understanding our dataset through EDA visualization we applied, the following data preprocessing techniques:

- **Identification of Null Values**
  - Use df.isna().sum() to identify the NaN value for each column.

- **Imputation**
  - For few columns like spc_latin, spc_common we address the missing values using mode()
  - For a few records in four columns it was not possible to replace the missing values in that case we dropped those records

- **Descritize the target variable**
  - Target variable “health” consists of values “Good”, “Fair” and “Poor”
  - Labeled it as “Good” set to 1 and other values set to 0

- **Sampling Techniques**
  - Through EDA visualization we concluded that our target variable has imbalance classes
  - So model prediction would have been biased
  - To avoid that we used undersampling technique
  - Library: sklearn.utils import resample

Data Preprocessing Techniques

- **Handling Categorical Variables**
  - Used Label Encoder() to convert categorical variables to numeric

- **Feature Scaling**
  - Perform normalization using MinMaxScaler() to normalize our data

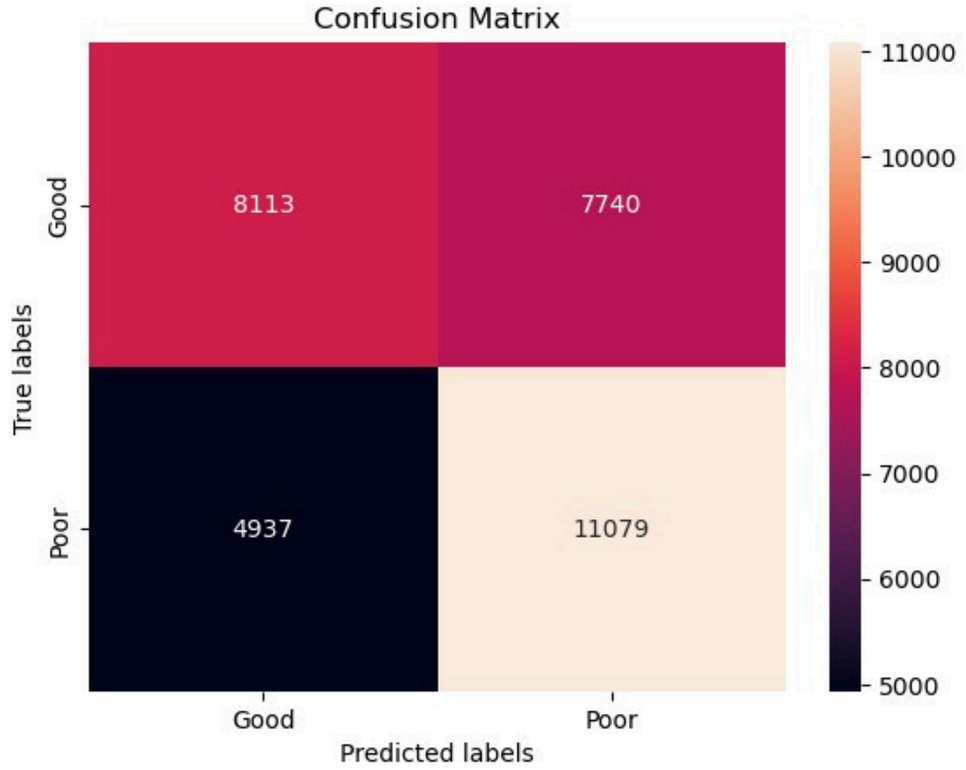
- **Feature Selection**
  - Applied chi-square test to select the best features from our dataset

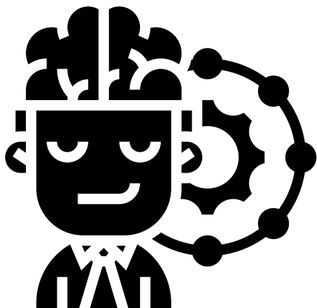
```
# Encode categorical variables
label_encoders = {}
categorical_cols = data2=['curb_loc','status','spc_latin','spc_common','steward','sidewalk','user_type','problems','addre
for col in categorical_cols:
    label_encoders[col] = LabelEncoder()
    df[col] = label_encoders[col].fit_transform(df[col])

# Split dataset into features and target variable
X = df.drop(columns=["health",'tree_id'])
y = df["health"]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Apply Min-Max scaling to make features non-negative
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply Chi-square test for feature selection
num_features = 15 # You can choose the number of features you want to select
selector = SelectKBest(score_func=chi2, k=num_features)
X_train_selected = selector.fit_transform(X_train_scaled, y_train)
X_test_selected = selector.transform(X_test_scaled)

# Get selected feature indices
selected_feature_indices = selector.get_support(indices=True)

# Get selected feature names
selected_feature_names = X.columns[selected_feature_indices]
print("Selected Features:", selected_feature_names)
```

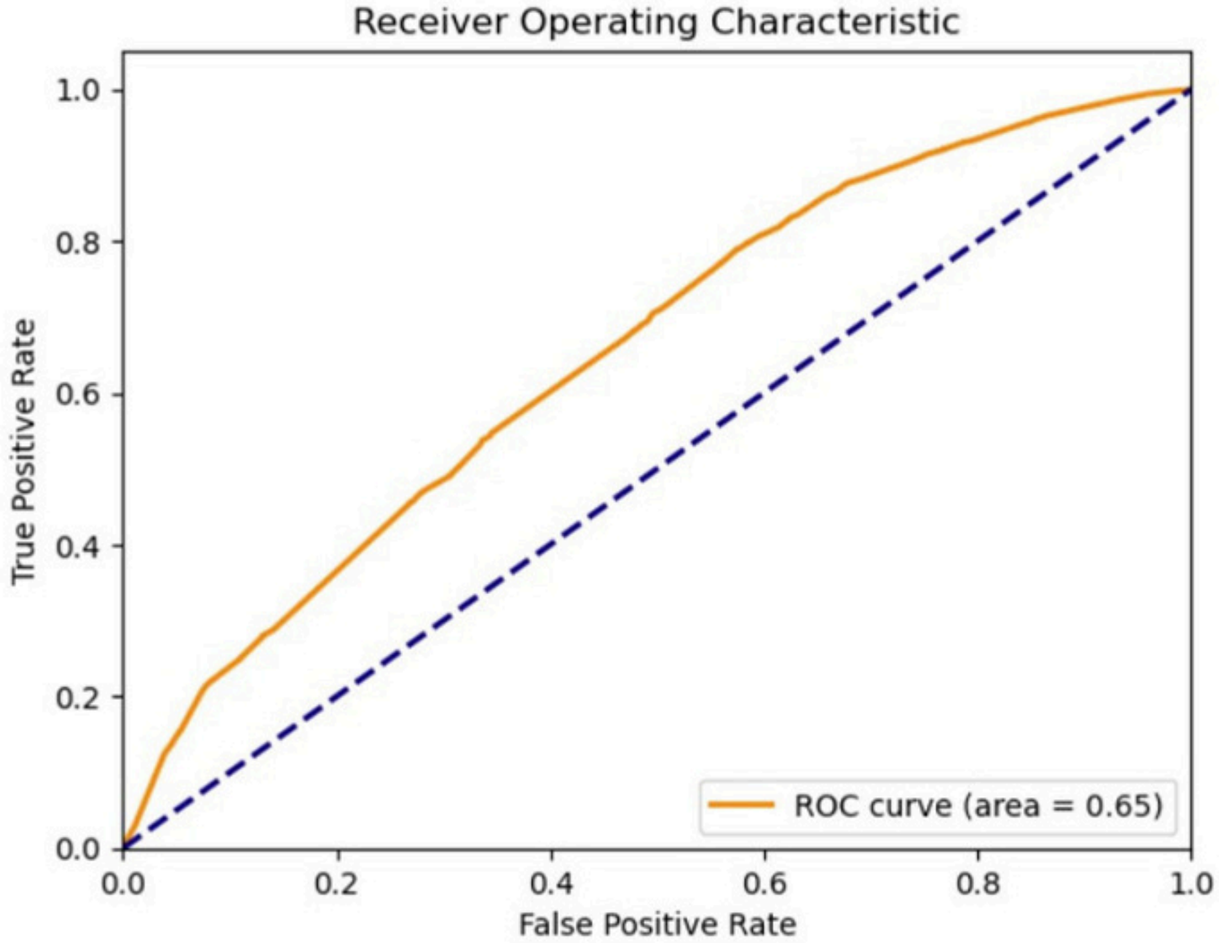
Naive Bayes Classifier

```
Confusion Matrix
[[ 8113  7740]
 [ 4937 11079]]

Accuracy score
0.6022153189620006

Classification Report
              precision    recall  f1-score   support

           0       0.62      0.51      0.56     15853
           1       0.59      0.69      0.64     16016

    accuracy                           0.60     31869
   macro avg       0.61      0.60      0.60     31869
weighted avg       0.61      0.60      0.60     31869
```

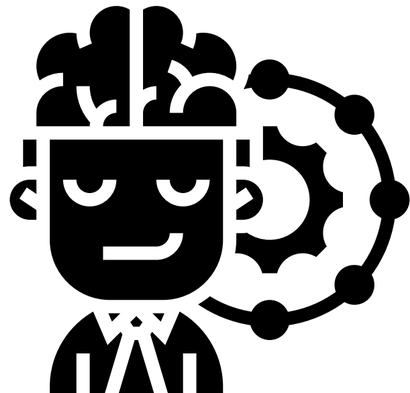




Naive Bayes Classifier

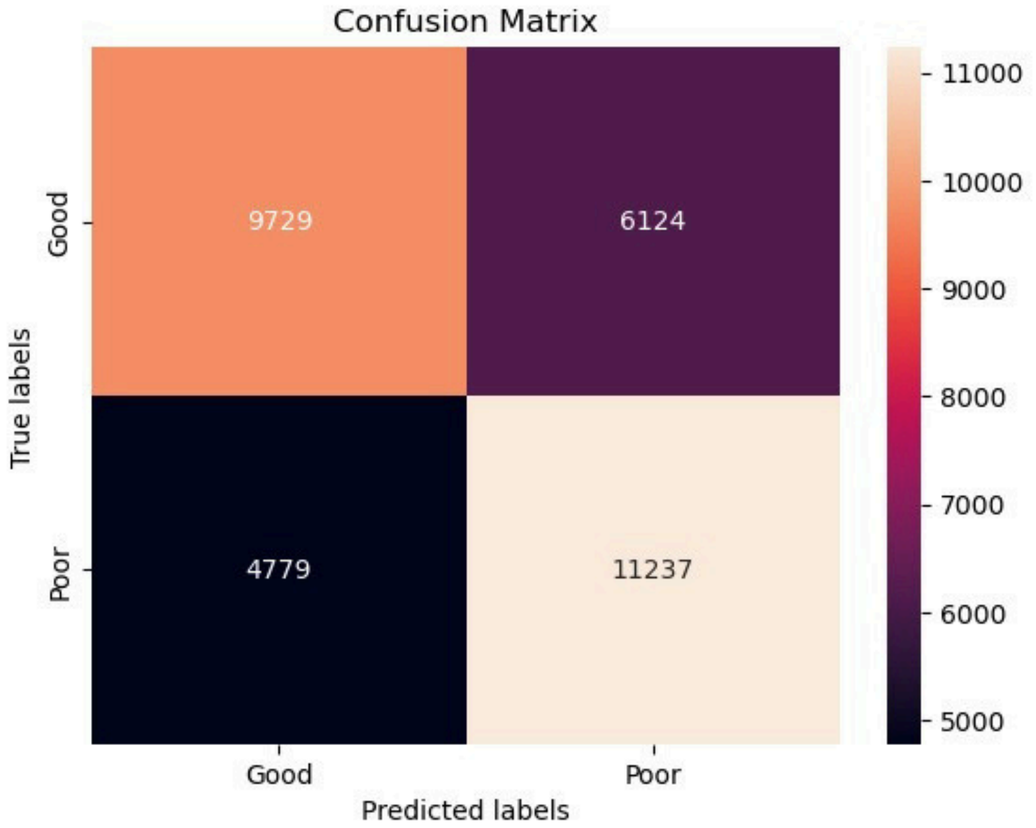


AUC: 0.6537983085481982

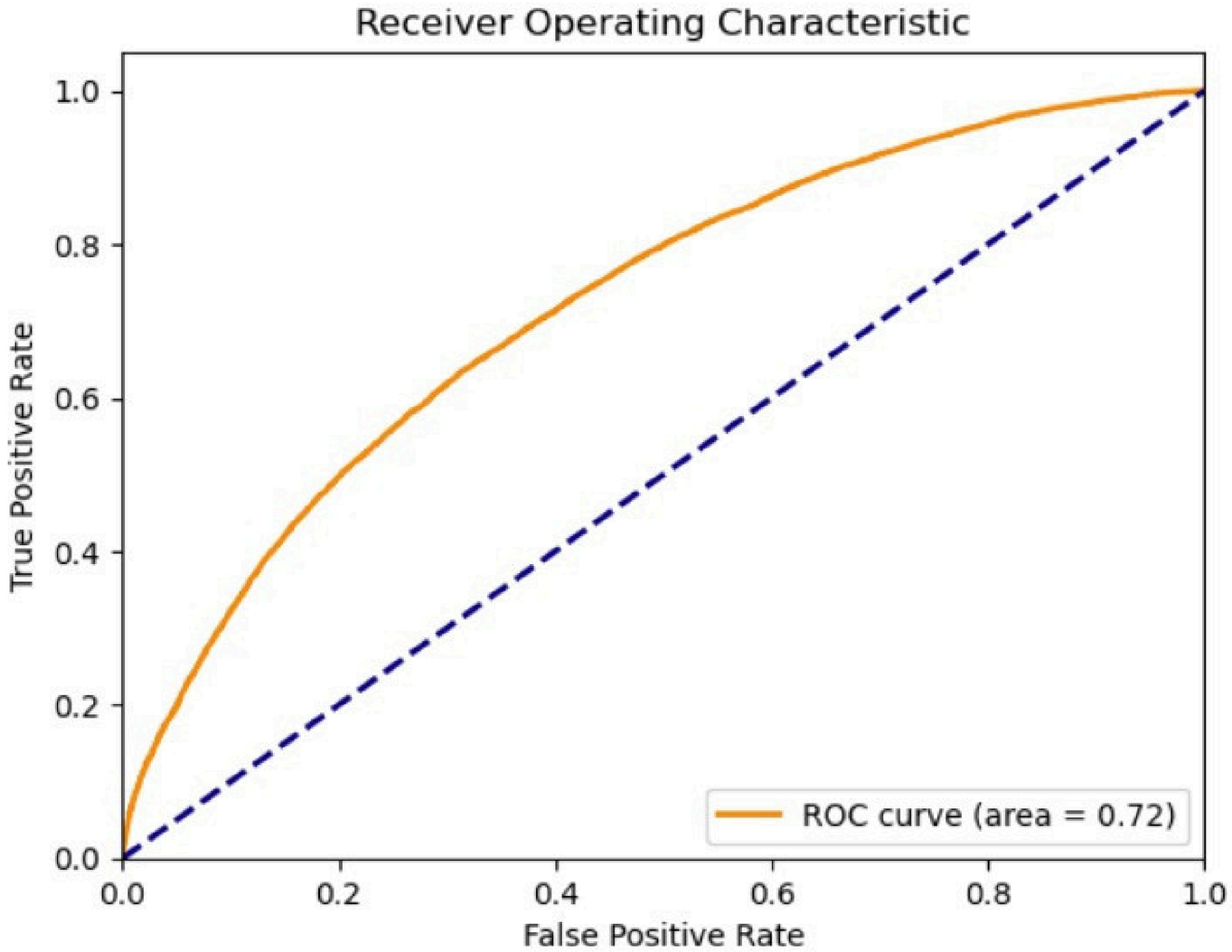


Gradient Boosting Classifier

```
Confusion Matrix
[[ 9729  6124]
 [ 4779 11237]]

Accuracy score
0.6578806991119897

Classification Report
              precision    recall  f1-score   support

           0       0.67      0.61      0.64     15853
           1       0.65      0.70      0.67     16016

    accuracy                           0.66     31869
   macro avg       0.66      0.66      0.66     31869
weighted avg       0.66      0.66      0.66     31869
```



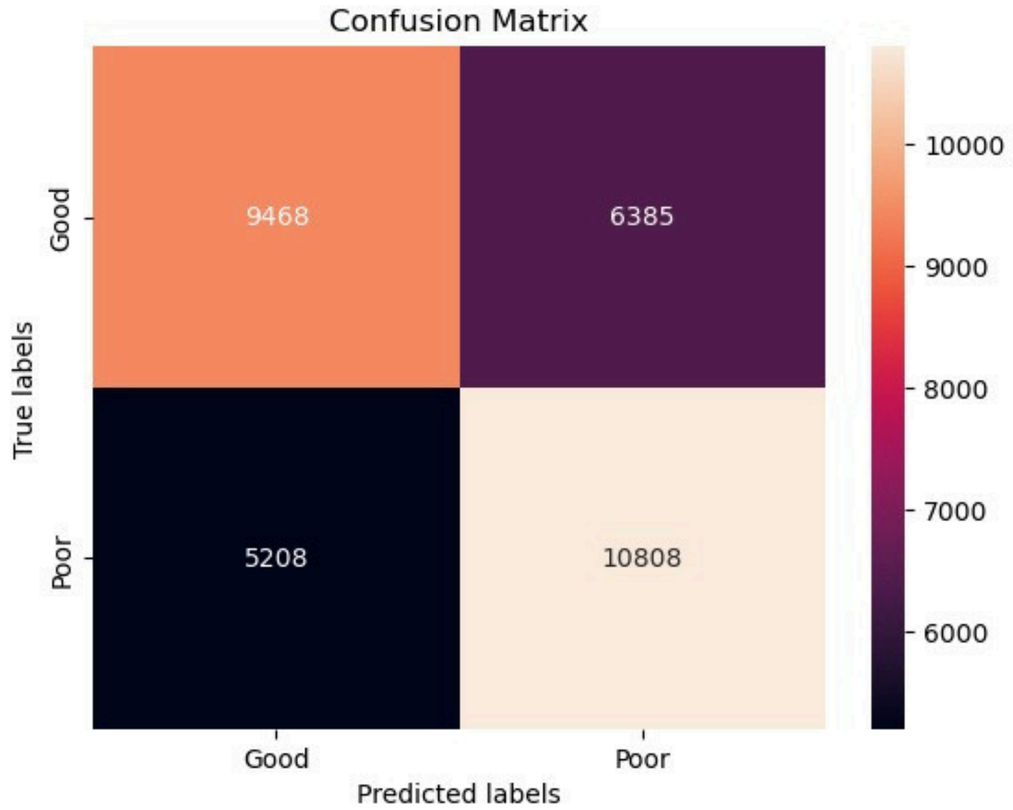
Gradient Boosting Classifier



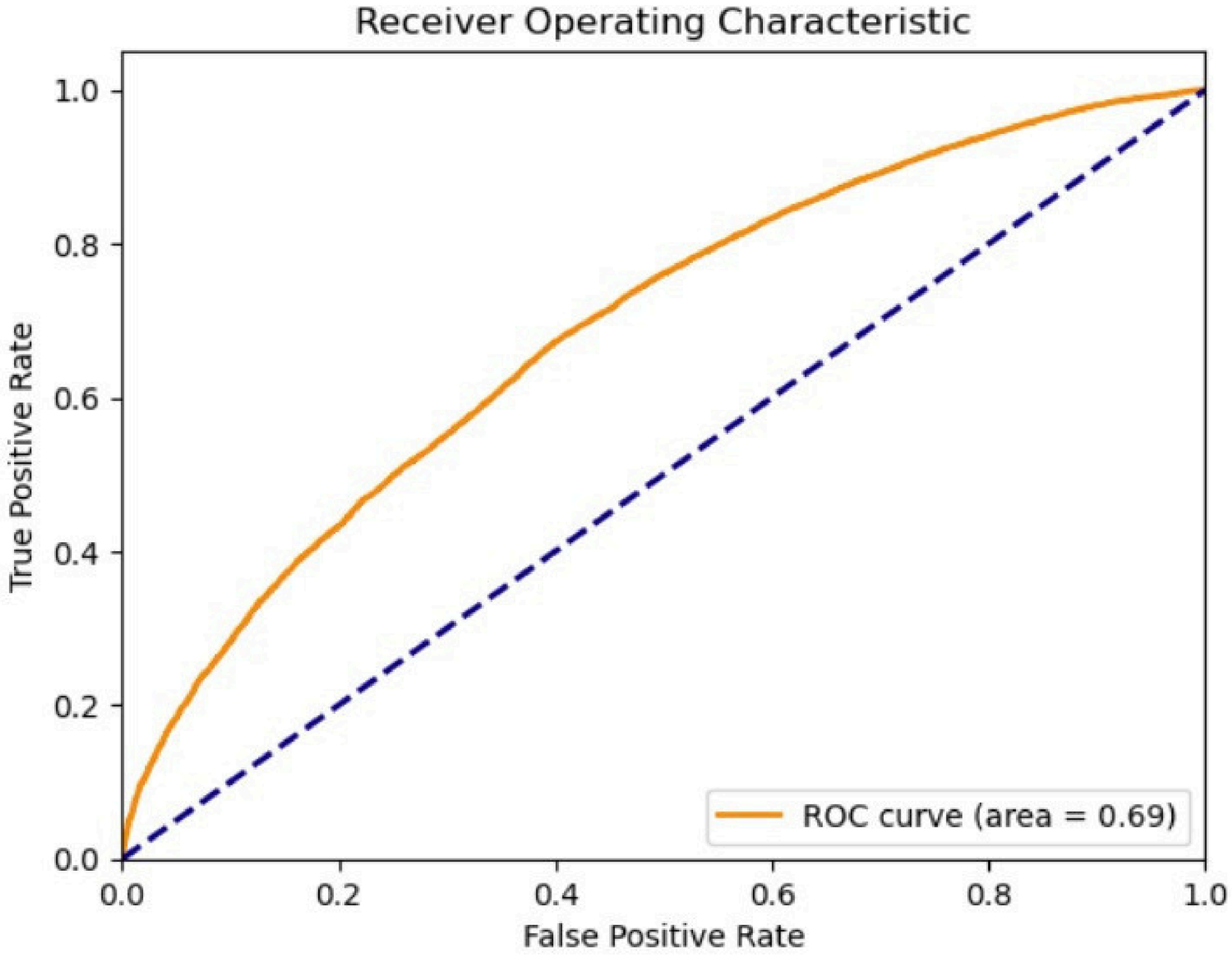
AUC: 0.7224949835693859

Logistic Regression Classifier

```
Confusion Matrix
[[ 9468  6385]
 [ 5208 10808]]

Accuracy score
0.6362295647808215

Classification Report
              precision    recall  f1-score   support

           0       0.65      0.60      0.62     15853
           1       0.63      0.67      0.65     16016

    accuracy                           0.64     31869
   macro avg       0.64      0.64      0.64     31869
weighted avg       0.64      0.64      0.64     31869
```



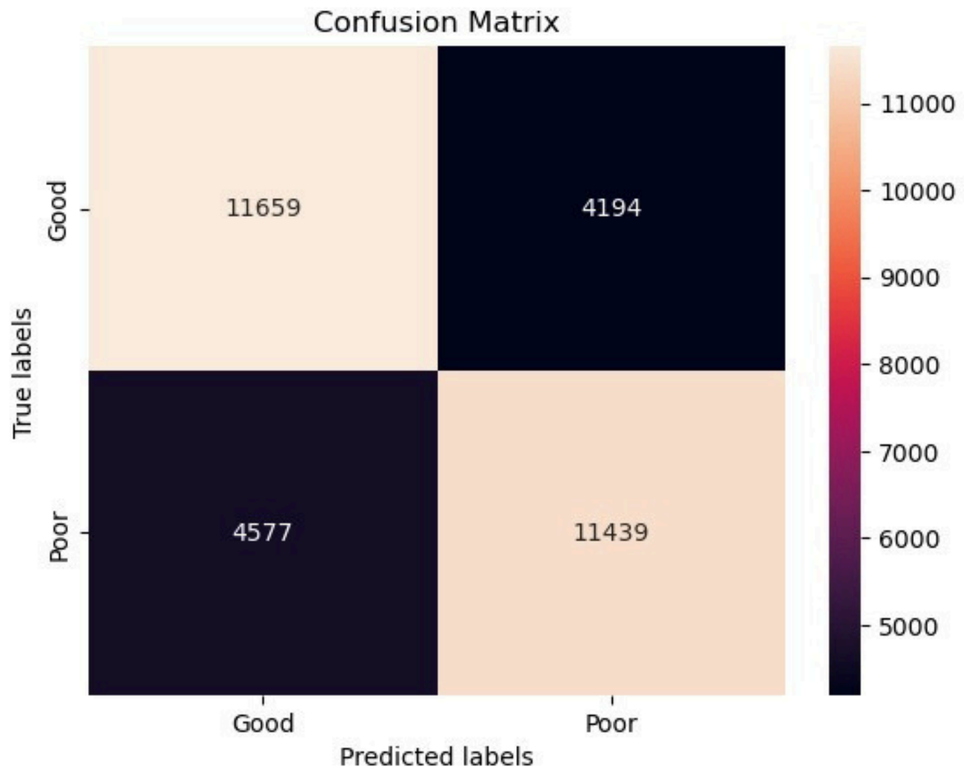
Logistic Regression Classifier

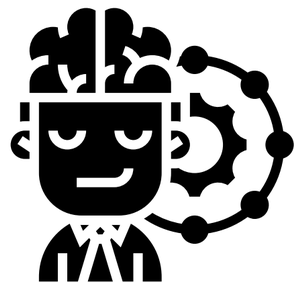


AUC: 0.6894805011269560

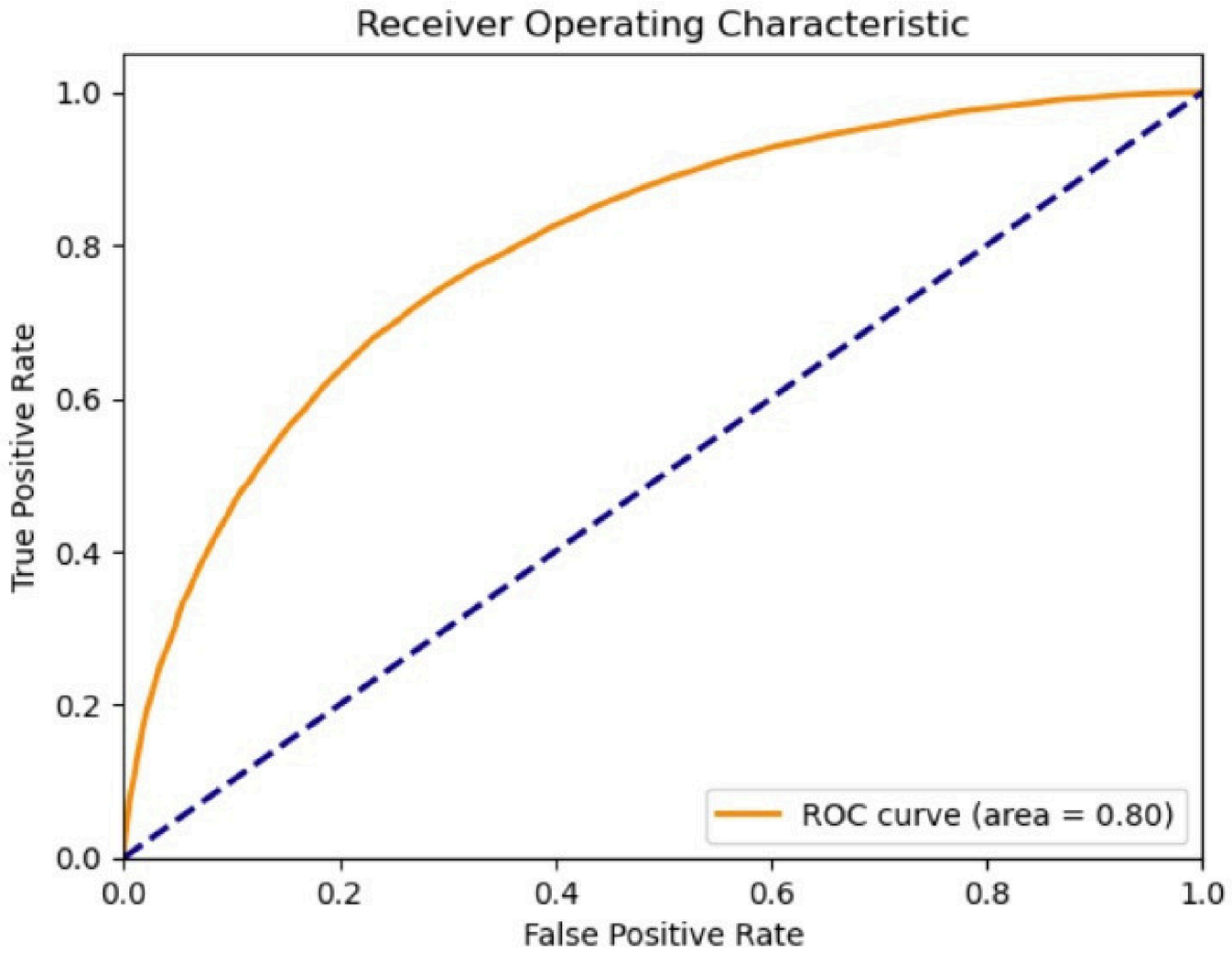
Random Forest Classifier

```
Confusion Matrix
[[11659  4194]
 [ 4577 11439]]

Accuracy score
0.7247795663497443

Classification Report
              precision    recall  f1-score   support

           0       0.72      0.74      0.73     15853
           1       0.73      0.71      0.72     16016

    accuracy                           0.72     31869
   macro avg       0.72      0.72      0.72     31869
weighted avg       0.72      0.72      0.72     31869
```





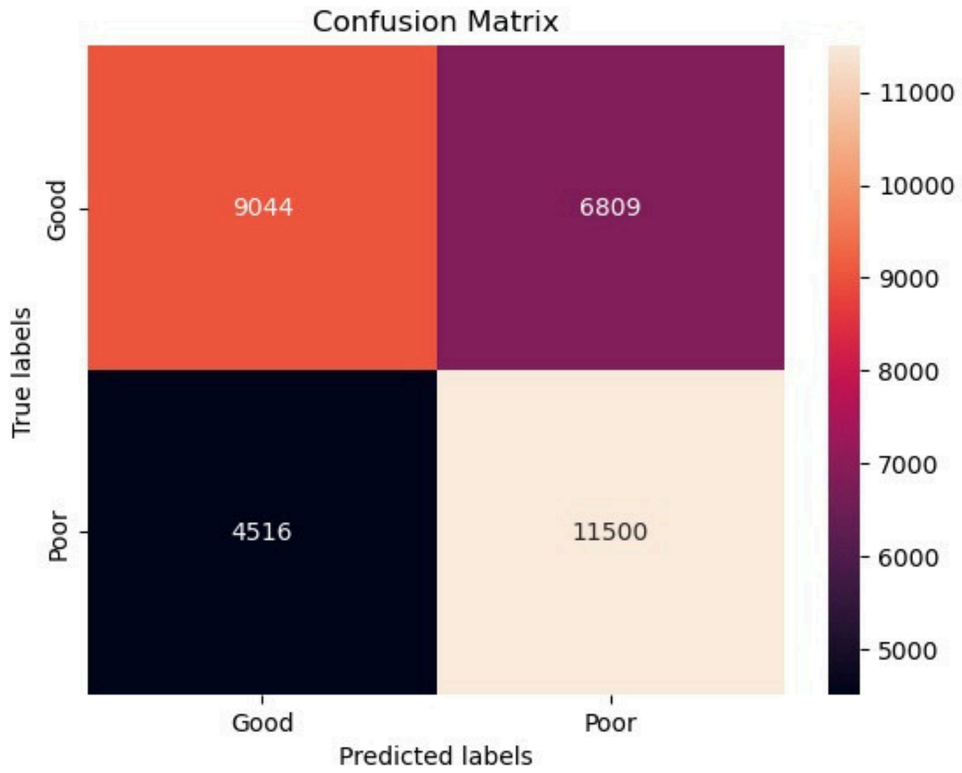
Random Forest Classifier



AUC: 0.7990899511609315

Support Vector Machine Classifier

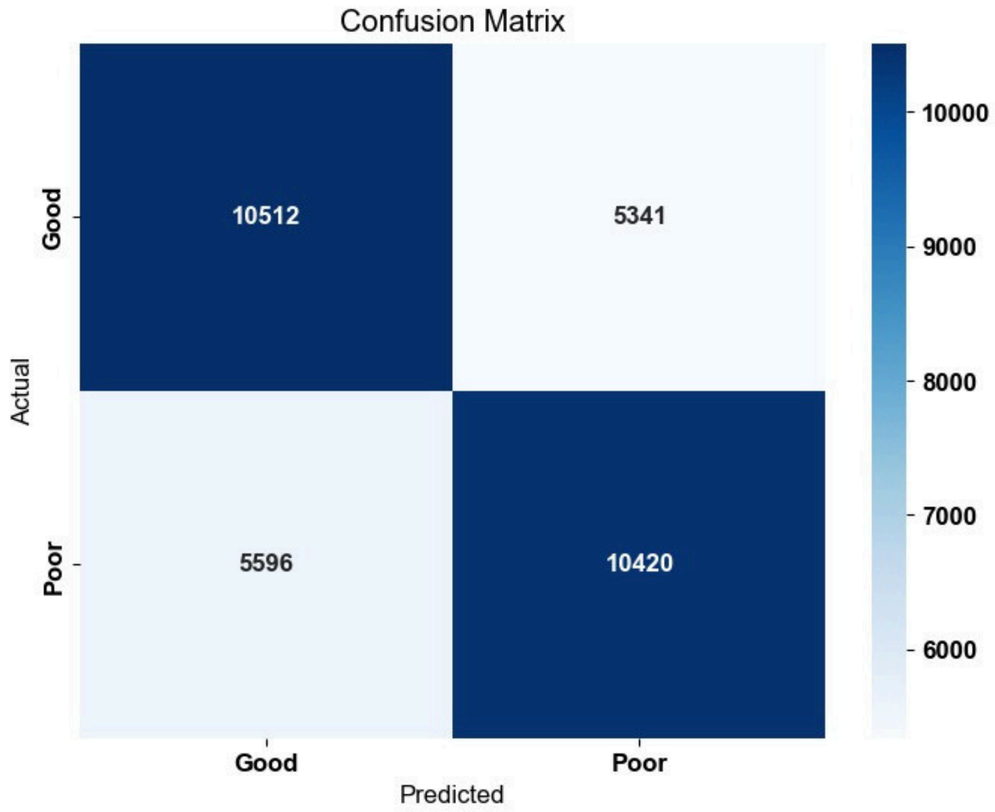
```
Confusion Matrix
[[ 9044  6809]
 [ 4516 11500]]

Accuracy score
0.6446389908688694

Classification Report
              precision    recall  f1-score   support

           0       0.67      0.57      0.61     15853
           1       0.63      0.72      0.67     16016

    accuracy                           0.64     31869
   macro avg       0.65      0.64      0.64     31869
weighted avg       0.65      0.64      0.64     31869
```



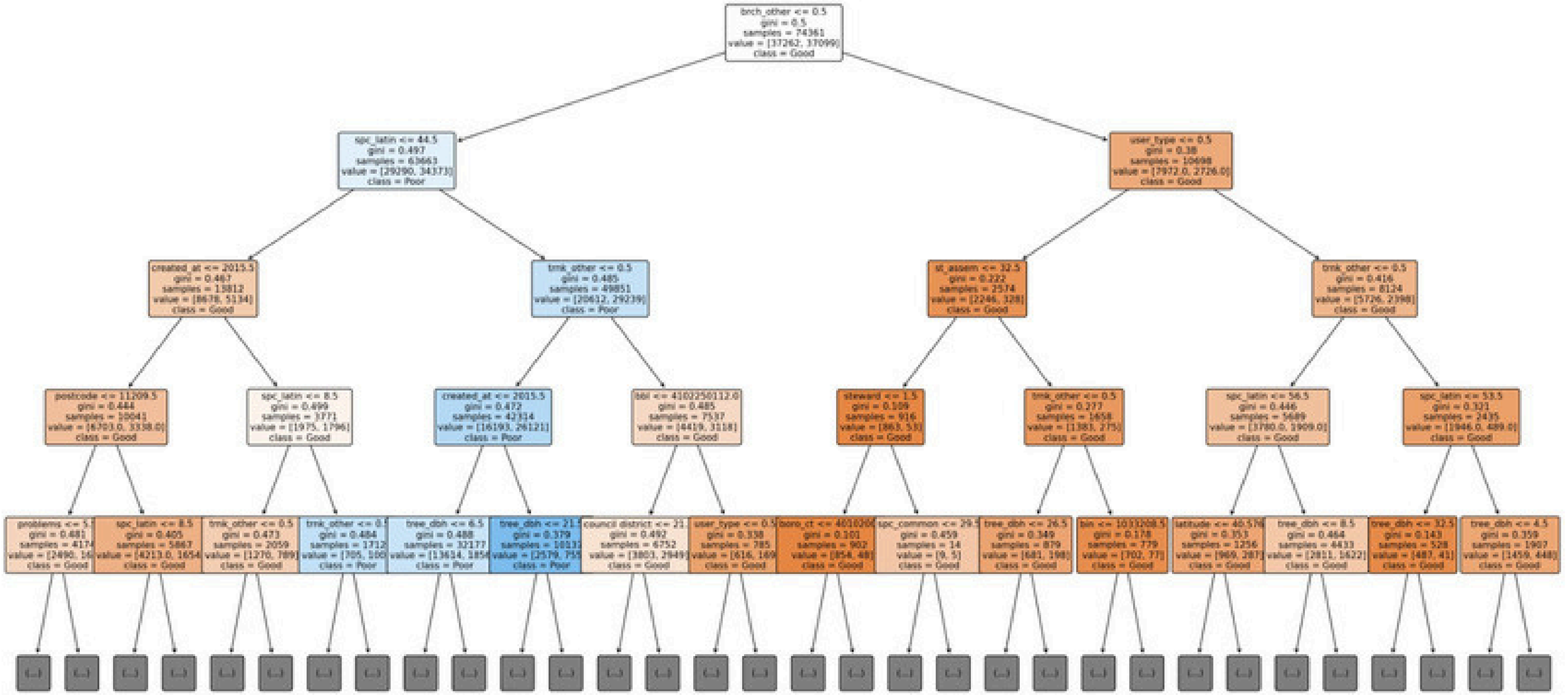
Decision Tree Classifier

Accuracy: 0.6568138316232075
Classification Report:

	precision	recall	f1-score	support
0	0.65	0.66	0.66	15853
1	0.66	0.65	0.66	16016
accuracy			0.66	31869
macro avg	0.66	0.66	0.66	31869
weighted avg	0.66	0.66	0.66	31869

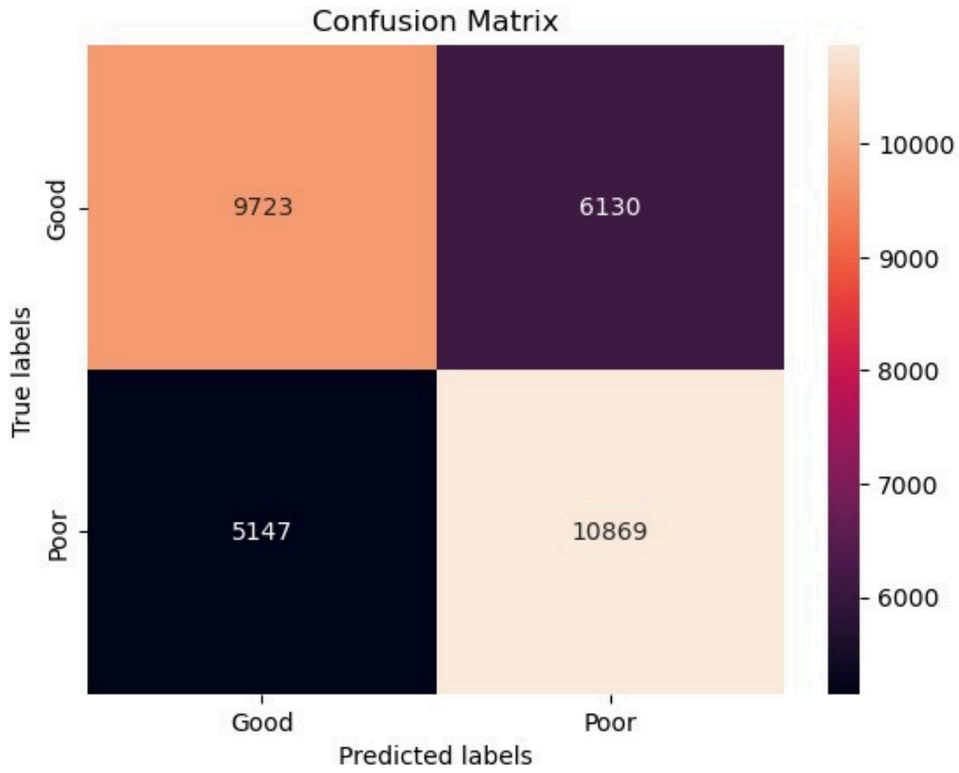


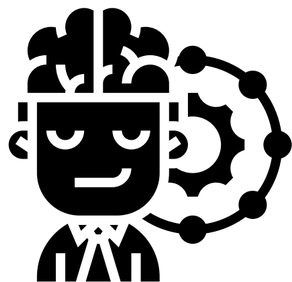
Decision Tree Classifier



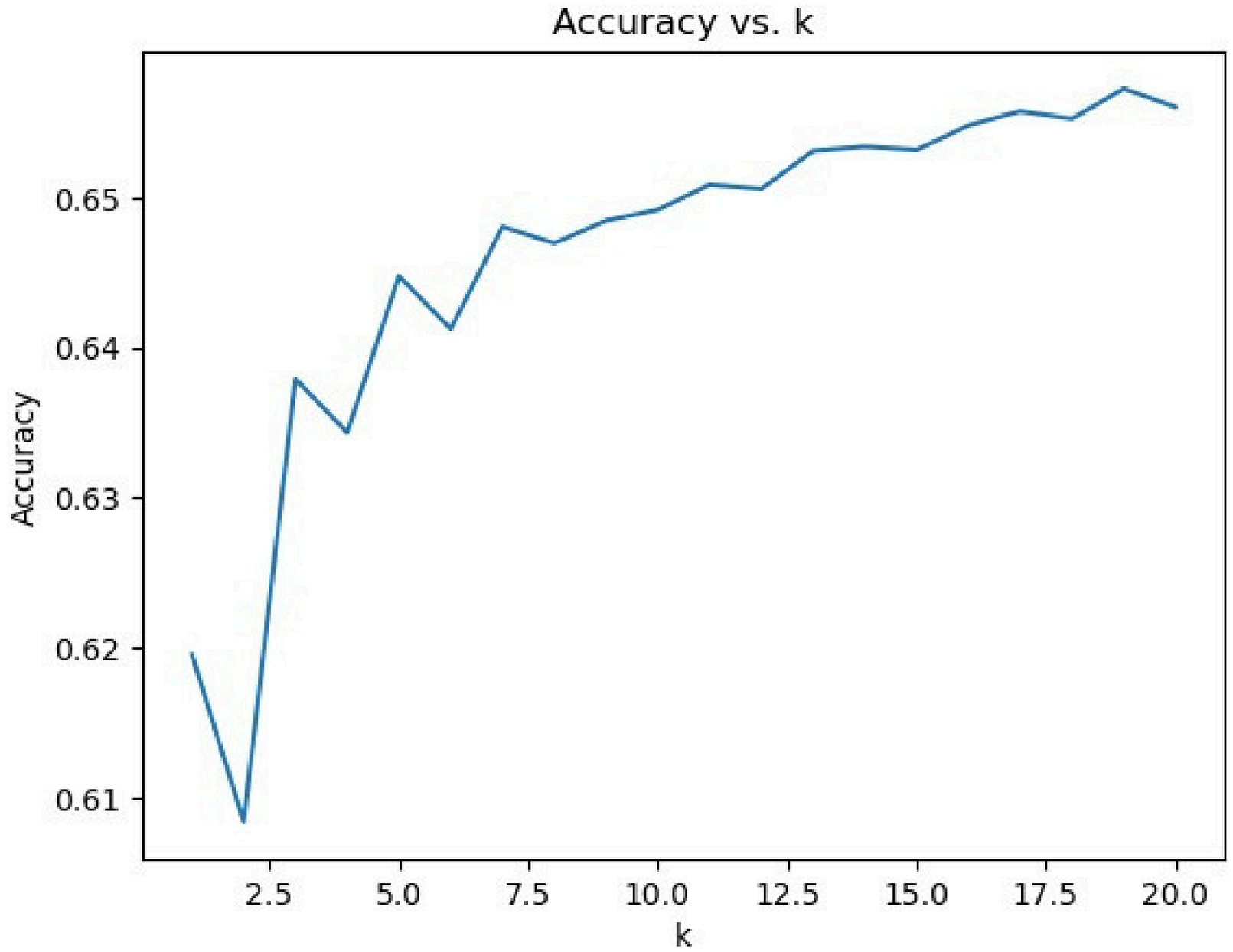
Artificial Neural Network Classifier

```
Confusion Matrix
[[ 9723  6130]
 [ 5147 10869]]

Accuracy score
0.6461451567353855

Classification Report
              precision    recall  f1-score   support

           0       0.65      0.61      0.63     15853
           1       0.64      0.68      0.66     16016

    accuracy                           0.65     31869
   macro avg       0.65      0.65      0.65     31869
weighted avg       0.65      0.65      0.65     31869
```





K-Nearest Neighbors Classifier

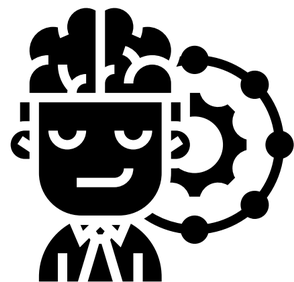


K-Nearest Neighbors Classifier

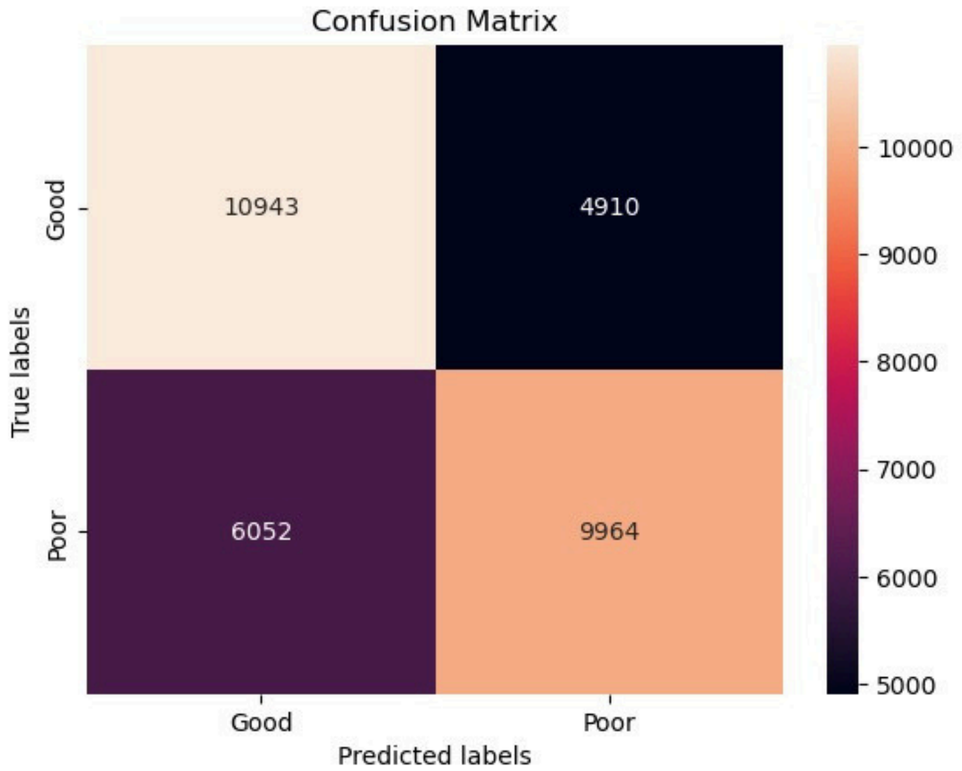
```
Accuracy of model with k = 2: 0.6560293702343971%

Confusion Matrix:
[[10943  4910]
 [ 6052  9964]]

Classification Report
              precision    recall  f1-score   support

           0       0.64      0.69      0.67     15853
           1       0.67      0.62      0.65     16016

    accuracy                           0.66     31869
   macro avg       0.66      0.66      0.66     31869
weighted avg       0.66      0.66      0.66     31869
```

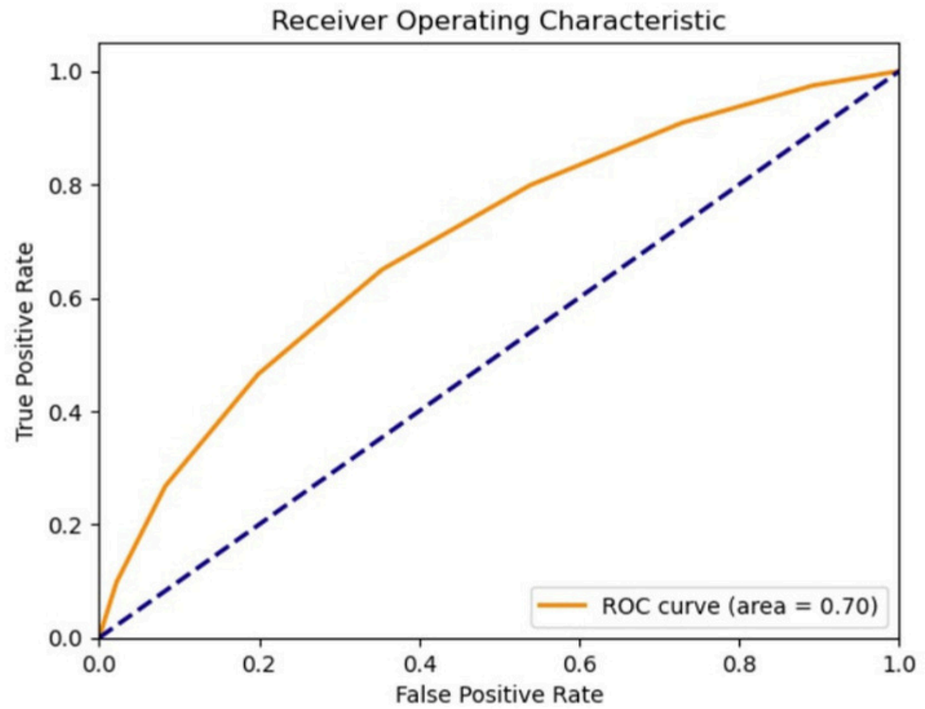
```
Accuracy of model with k = 4: 0.6560293702343971%

Confusion Matrix:
[[10943  4910]
 [ 6052  9964]]

Classification Report
              precision    recall  f1-score   support

           0       0.64      0.69      0.67     15853
           1       0.67      0.62      0.65     16016

    accuracy                           0.66     31869
   macro avg       0.66      0.66      0.66     31869
weighted avg       0.66      0.66      0.66     31869
```

```
Accuracy of model with k = 7: 0.6560293702343971%

Confusion Matrix:
[[10943  4910]
 [ 6052  9964]]

Classification Report
              precision    recall  f1-score   support

           0       0.64      0.69      0.67     15853
           1       0.67      0.62      0.65     16016

    accuracy                           0.66     31869
   macro avg       0.66      0.66      0.66     31869
weighted avg       0.66      0.66      0.66     31869
```

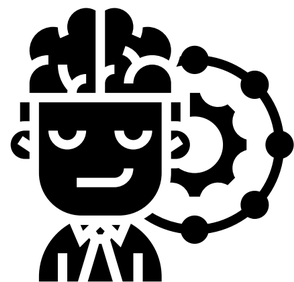


K-Nearest Neighbors Classifier





AUC: 0.6976996659745982



Model Comparison & Conclusion

Following is the conclusion using the various performance metrics for all the applied binary classification algorithms on our dataset:

- **Accuracy:** Random Forest showed the highest accuracy of 72.47 %
- **Precision:** Random Forest has the highest precision of 0.73
- **Recall:** Random Forest has the highest recall of 0.71
- **F1-score** for Random Forest is 0.72
- **Confusion matrix:** False positive and False negative values are the least for Random Forest

So, concluding from the all the above-mentioned metrics **Random Forest Classification** model is the best binary classification for our dataset.

Thank You!