Predictive Model for Analytics Hackathon

Dataset Overview

Loading the training and testing datasets (X_Train_Data_Input.csv, Y_Train_Data_Target.csv, etc.).

Uses the pandas library to read and check the shapes of the datasets.

```
df_train = pd.read_csv("X_Train_Data_Input.csv")
df_train_target = pd.read_csv("Y_Train_Data_Target.csv")
df_test = pd.read_csv("X_Test_Data_Input.csv")
df_test_target = pd.read_csv("Y_Test_Data_Target.csv")

print(f'Training Data Shape: {df_train.shape}, Test Data Shape: {df_test.shape}')
```

Basic Exploration

- Check for missing values in the dataset.
- Show basic statistics (mean, median, min, max).

```
print(df_train.isnull().sum())
 print(df_train.describe())
```

Handling Missing Values

- Drop unnecessary columns with a high percentage of missing values.
- Drop the ID column as it's not useful for model training.
- Impute missing values using mean or another strategy.

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='mean')
df_train_imputed = pd.DataFrame(imputer.fit_transform(df_train), columns=df_train.columns)
```

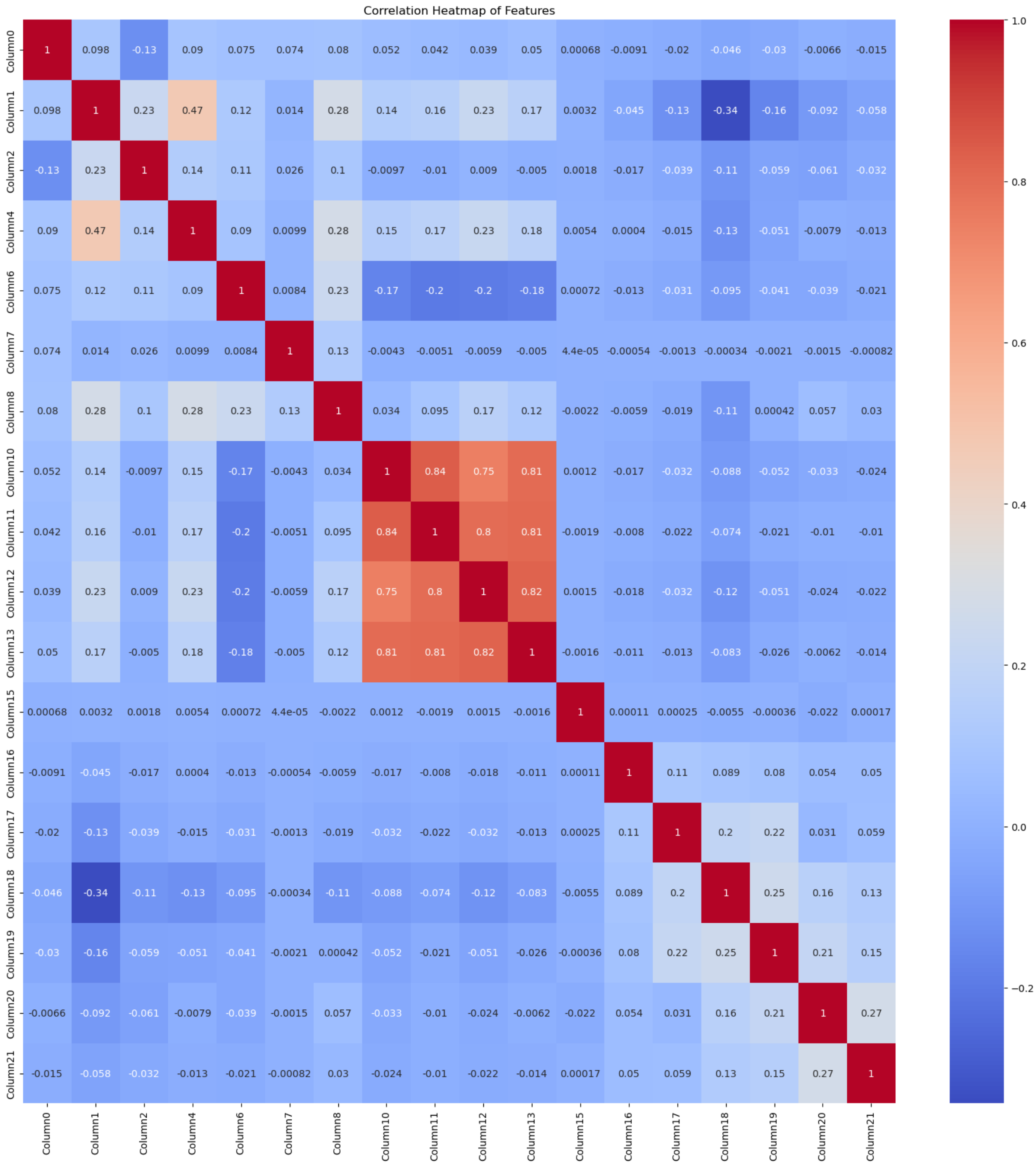
Exploratory Data Analysis (EDA)

Correlation Analysis

- Use a heatmap to visualize correlations between features.

```
plt.figure(figsize=(20,20))
```

```
sns.heatmap(df_train.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```



Feature Reduction

- Use correlation analysis to remove highly correlated features (correlation > 0.8) to avoid multicollinearity.

```
corr_matrix = df_train.corr().abs()
upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [column for column in upper_triangle.columns if any(upper_triangle[column] > 0.8)]
df_train = df_train.drop(columns=to_drop)
df_test = df_test.drop(columns=to_drop)
```

Data Preprocessing

4.1 Scaling

- Apply different scaling methods like Standardization, Normalization, and Robust Scaling.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
# Standardization
scaler = StandardScaler()
df_train_scaled = pd.DataFrame(scaler.fit_transform(df_train_imputed),
columns=df_train_imputed.columns)
df_test_scaled = pd.DataFrame(scaler.transform(df_test_imputed),
columns=df_test_imputed.columns)
```

Handling Imbalanced Data

- Use SMOTE (Synthetic Minority Over-sampling Technique) to handle class imbalance in the target variable.

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_resample, y_resample = smote.fit_resample(df_train_scaled, df_train_target)
```

Model Building

Model Selection

- Choose several models for training, including:
  - RandomForest
  - DecisionTree
  - GradientBoosting
  - XGBClassifier

- CatBoostClassifier
- AdaBoost

```
models = {
"Random Forest": RandomForestClassifier(),
"Decision Tree": DecisionTreeClassifier(),
"Gradient Boosting": GradientBoostingClassifier(),
"XGBClassifier": XGBClassifier(),
 "CatBoosting Classifier": CatBoostClassifier(verbose=False),
"AdaBoost Classifier": AdaBoostClassifier(),
}
```

Model Evaluation Function

- Create a function to evaluate each model based on different metrics like accuracy, precision, recall, f1-score, and ROC AUC.

```
def evaluate_clf(true, predicted, prob):
    acc = accuracy_score(true, predicted)
    f1 = f1_score(true, predicted)
    precision = precision_score(true, predicted)
    recall = recall_score(true, predicted)
    roc_auc = roc_auc_score(true, predicted)
    log_loss_value = log_loss(true, prob)
    return acc, f1, precision, recall, roc_auc, log_loss_value
```

Model Performance on different Algorithms:

Random Forest

Model performance for Training set

- Accuracy: 0.9945

- F1 score: 0.9945

- Precision: 0.9892

- Recall: 1.0000

- Roc Auc Score: 0.9945

- Log Loss Score0.017734301638717413

Model performance for Test set

- Accuracy: 0.9858

- F1 score: 0.9860

- Precision: 0.9733

- Recall: 0.9989

- Roc Auc Score: 0.9857

- Logg Loss: 0.048060573111745046

===================================

Decision Tree

Model performance for Training set

- Accuracy: 0.9997

- F1 score: 0.9997

- Precision: 0.9997

- Recall: 0.9996

- Roc Auc Score: 0.9997

- Log Loss Score0.0007500408477461758

Model performance for Test set

- Accuracy: 0.9790

- F1 score: 0.9791

- Precision: 0.9774

- Recall: 0.9807

- Roc Auc Score: 0.9790

- Logg Loss: 0.7501836573360774

===================================

Gradient Boosting

Model performance for Training set

- Accuracy: 0.9829

- F1 score: 0.9832

- Precision: 0.9679

- Recall: 0.9989

- Roc Auc Score: 0.9829

- Log Loss Score0.06189489197258645

Model performance for Test set

- Accuracy: 0.9826

- F1 score: 0.9829

- Precision: 0.9675

- Recall: 0.9989

- Roc Auc Score: 0.9826

- Logg Loss: 0.062479410484319486

===================================

XGBClassifier

Model performance for Training set

- Accuracy: 0.9850

- F1 score: 0.9852

- Precision: 0.9720

- Recall: 0.9987

- Roc Auc Score: 0.9850

- Log Loss Score0.05091280556281581

Model performance for Test set

- Accuracy: 0.9840

- F1 score: 0.9842

- Precision: 0.9707

- Recall: 0.9981

- Roc Auc Score: 0.9839

- Logg Loss: 0.05397204008610946

===================================

CatBoosting Classifier

Model performance for Training set

- Accuracy: 0.9858

- F1 score: 0.9860

- Precision: 0.9737

- Recall: 0.9987

- Roc Auc Score: 0.9858

- Log Loss Score0.04746268587159103

Model performance for Test set

- Accuracy: 0.9846

- F1 score: 0.9848

- Precision: 0.9720

- Recall: 0.9979

- Roc Auc Score: 0.9845

- Logg Loss: 0.05098771480954667

===================================

AdaBoost Classifier

Model performance for Training set

- Accuracy: 0.9826

- F1 score: 0.9829

- Precision: 0.9674

- Recall: 0.9990

- Roc Auc Score: 0.9827

- Log Loss Score0.5217571858476999

Model performance for Test set

- Accuracy: 0.9824

- F1 score: 0.9828

- Precision: 0.9672

- Recall: 0.9989

- Roc Auc Score: 0.9824

- Logg Loss: 0.5222482834493352

===================================

Model Name	f1_score
Random Forest	0.985974
CatBoosting Classifier	0.984806
XGBClassifier	0.984230
Gradient Boosting	0.982924
AdaBoost Classifier	0.982772

Decision Tree	0.979064

Final Model Selection

- After evaluating all models, select the one that performs best (RandomForest).

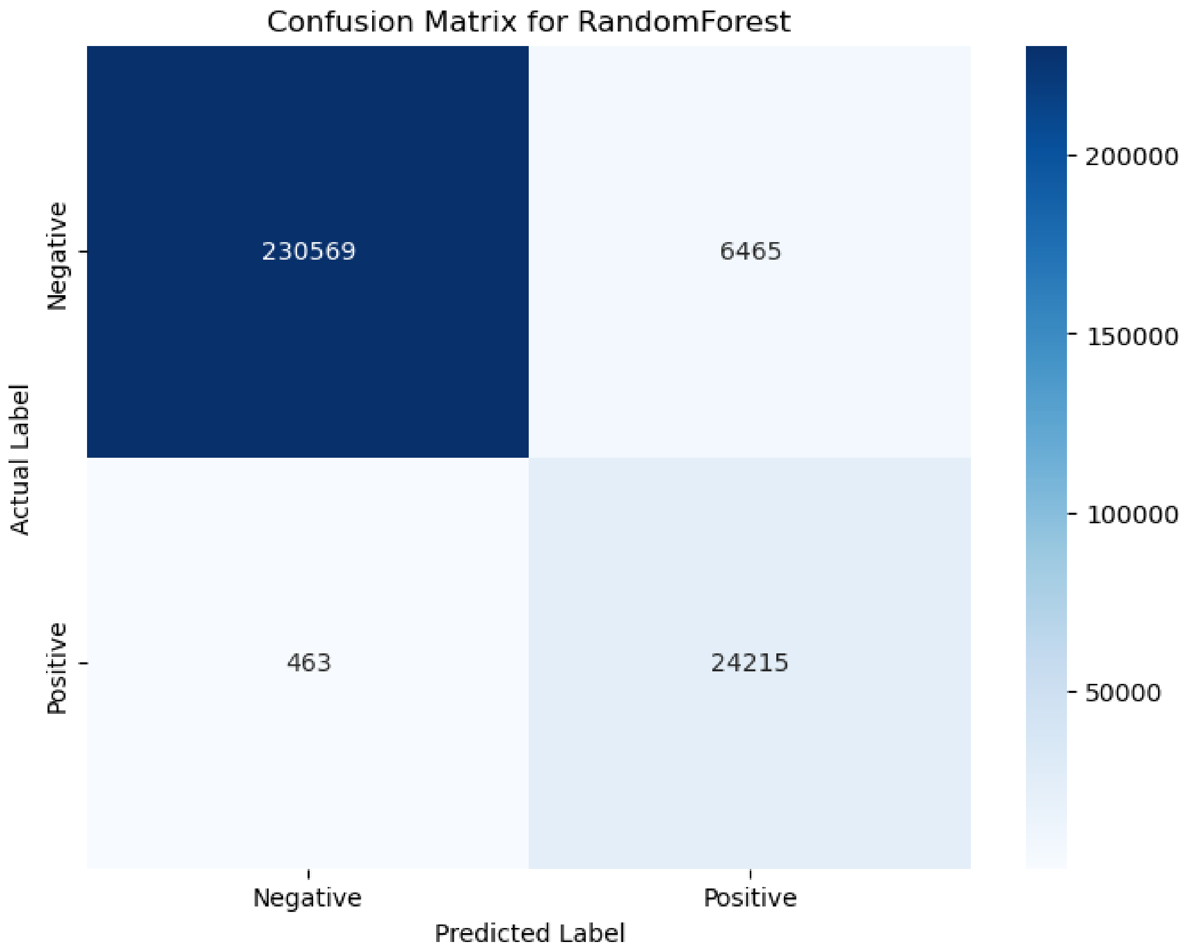
```
random_forest = RandomForestClassifier()
random_forest.fit(X_resample, y_resample)
y_pred_rf = random_forest.predict(df_test_scaled)
accuracy_rf = accuracy_score(df_test_target, y_pred_rf)
precision_rf = precision_score(df_test_target, y_pred_rf)
recall_rf = recall_score(df_test_target, y_pred_rf)
f1_rf = f1_score(df_test_target, y_pred_rf)
roc_auc_rf = roc_auc_score(df_test_target, y_pred_rf)

print(f"Accuracy: {accuracy_rf}, Precision: {precision_rf}, Recall: {recall_rf}, F1 Score: {f1_rf},
AUC-ROC Score: {roc_auc_rf}")
```

Confusion Matrix

- Visualize the confusion matrix to understand the distribution of predictions.

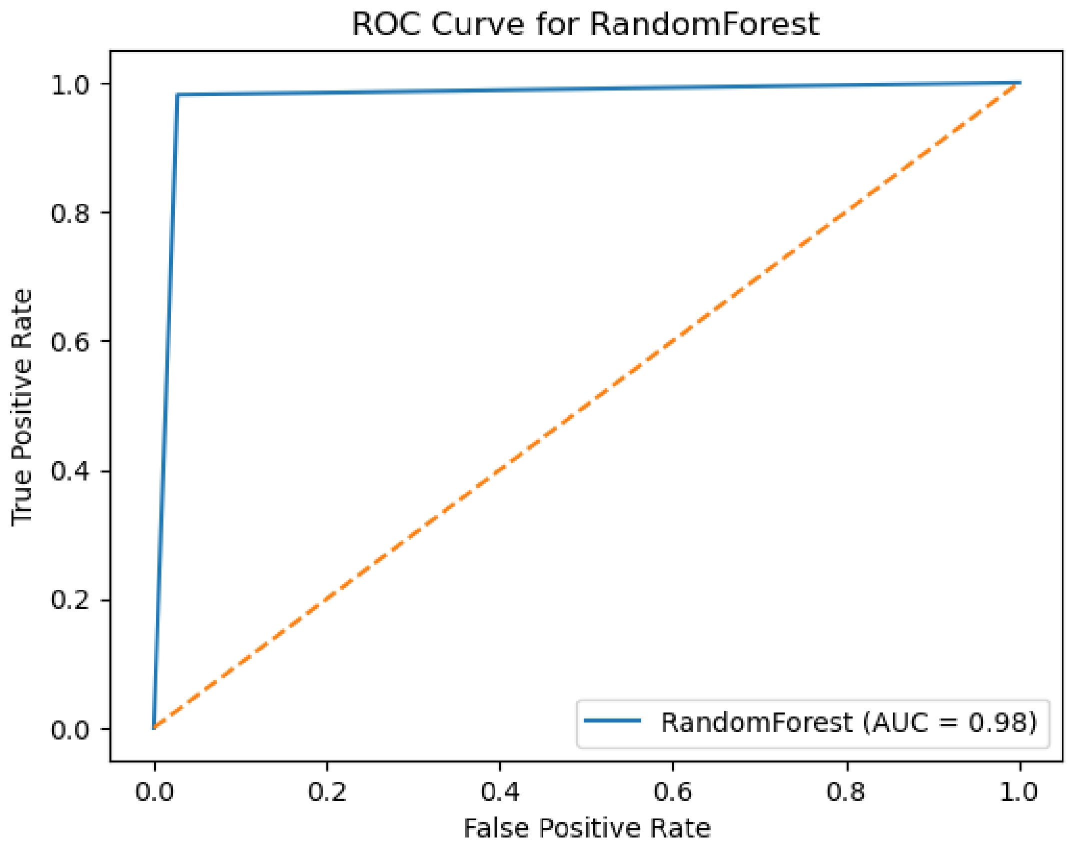
```
cm = confusion_matrix(df_test_target, y_pred_rf)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'],
yticklabels=['Negative', 'Positive'])
plt.title('Confusion Matrix')
plt.show()
```



ROC Curve

- Plot the ROC Curve for the final model.

```
fpr, tpr, _ = roc_curve(df_test_target, y_pred_rf)
plt.plot(fpr, tpr, label='RandomForest (AUC = %0.2f)' % roc_auc_rf)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



ROC Curve for RandomForest
True Positive Rate
False Positive Rate
RandomForest (AUC = 0.98)
0.0
0.2
0.4
0.6
0.8
1.0