Enhancing Financial Security: Implementation of a Credit Card Fraud Detection Model on Google Cloud Platform

by

Team F

(Kiran Kumar Reddy Guggilla;

Nimmagadda Sada Siva Madhav;

Varsada Anjali Rameshchandra)

FINAL PROJECT REPORT

for

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2023

ABSTRACT

Financial transactions have mostly moved to internet platforms in the quickly developing digital age, which has increased the prevalence of fraudulent activity. The main goal of this project is to create strong machine-learning models that can recognize fraudulent activity in credit card transactions by using the "Credit Card Transactions Fraud Detection Dataset" from Kaggle, a comprehensive compilation of transaction data. The dataset is a significant source of information for advanced analytical approaches because it contains variables such as transaction amount, duration, and user behavior patterns.

In order to extract significant patterns and relationships, our approach first preprocesses the data to address imbalances and noise, and then we apply feature engineering. We use a range of machine learning algorithms, such as ensemble approaches, neural networks, and decision trees, to create prediction models. To make sure that these models can tell the difference between authentic and fraudulent transactions, they are subjected to a thorough evaluation process that makes use of criteria such as accuracy, precision, recall, and area under the ROC curve.

This project seeks to make a substantial contribution to the subject of cybersecurity in finance by combining data analysis and machine learning. It not only highlights the power of data science in addressing real-world issues, but it also provides insights for improving online transaction security, thereby protecting consumer interests and increasing trust in digital financial services.

LIST OF ABBREVIATIONS AND SYMBOLS
(SAMPLE)

a	Cronbach's index of internal consistency
df	Degrees of freedom: number of values free to vary after certain restrictions have been placed on the data.
F	Fisher's *F* ratio: A ration of two variances
M	Mean: the sum of a set of measurements divided by the number of measurements in the set.
p	Probability associated with the occurrence under the null hypothesis of a value as extreme as or more extreme than the observed value.
r	Pearson product-moment correlation
t	Computed value of *t* test
<	Less than
=	Equal to

ACKNOWLEDGMENTS

We team F (Kia Kiran Kumar Reddy Guggilla; Nimmagadda Sada Siva Madhav; Varsada Anjali Rameshchandra) acknowledge and thank our professor Unal Sakoglu as well as many colleagues, friends, and faculty members who gave feedback, helped with and guided this research project.

TEAM MEMBERS' CONTRIBUTIONS

Name	Duties	Achievements
Kiran Kumar Reddy Guggilla	As a team member I am primarily responsible for working on the Data Preprocessing and Data wrangling. I have contributed myself in Data visualization and Data Analysis. Moreover, I and Madhav worked together on dimensionality reduction using PCA to compare the PCA data with the original data. In addition to this I helped Madhav in Oversampling techniques, Model Building and Evaluation.	By leveraging the power of data wrangling techniques, I Analyzed the data and able to Visualize the results in an efficient way. In addition, I have successfully implemented PCA with 13 components using power of Kaggle GPU's which paved the way for our Project Completion.
Sada Siva Madhav Nimmagadda	As a team Member my major work involved in Model Building and Evaluation. I had to research and came up with effective idea to train my huge dataset. I have utilized free GPUs in the market to achieve this. In addition to this I have helped Kiran in dimensionality reduction process. Moreover, Kiran assisted while evaluating the	By utilizing the power of Machine Learning and Deep Learning Models. I have successfully built my models and worked on techniques to improve their performance. Moreover, I have successfully implemented the oversampling technique to achieve balance in the target class which helped our project to compare results of both Normal

	predictions of the models.	Sampling and Oversampling.
Anjali Rameshchandra Varsada	As a Team member I was majorly focused on integrating my dataset in Google Cloud Platform. Kiran and Madhav focused on deriving results based on traditional methods. On the other hand, I was using the power of Vertex AI to Analyze the data and wrangle the data. I have utilized the free resources of this platform and able to develop Models and evaluate them.	By leveraging the power Google Cloud Platform and Vertex AI I successfully visualized the data. In addition, I achieved better results on predicting the fraud transaction by efficiently building my models.

TABLE OF CONTENTS

I. INTRODUCTION

I.1 Overview of Credit Card Fraud Detection

The banking sector's shift to the digital sphere has resulted in a notable surge in credit card transactions, as well as an increase in fraudulent activity. These illegal operations undermine consumer trust in digital payment systems in addition to causing significant financial losses for all parties involved. For financial institutions, identifying and stopping such scams has now become of utmost importance. Conventional fraud detection techniques mostly depended on rule-based algorithms and manual identification, which are not scalable in the face of growing amounts of data and cunning fraud schemes. The foundation of this project is the requirement for an automated, intelligent system that uses cutting-edge machine learning techniques to sort through massive transaction data and identify possible fraud with high precision and low false positives.

There are various ways that credit card fraud can occur, each with its own set of identification problems. These include stolen card information, created identities, and card-not-present (CNP) fraud. The difficulty is increased by the requirement for real-time detection to stop fraudulent transactions from happening. This work leverages a big dataset from Kaggle that simulates real-world transaction patterns and fraudulent activity in order to train and test prediction models. The main objective is to create a model that can recognize fraudulent transactions with accuracy and can adjust to new, unidentified fraud patterns.

I.2. Background & Literature Review/Survey

A dynamic approach to fraud detection is required due to the ongoing evolution of fraud strategies. Financial firms used to detect fraud by using straightforward, unchanging criteria or thresholds, like identifying all transactions over a specific amount. But fraudsters soon figured out how to get around these regulations, which prompted the creation of increasingly complex machine learning-based solutions. According to the research, there has been a noticeable shift toward models that are able to forecast transaction validity, identify intricate patterns, and learn from past data. With differing degrees of success, methods from logistic regression to neural networks have been investigated.

We came across research in our analysis of the literature that emphasize the significance of feature engineering and the choice of suitable machine learning algorithms. A major problem in fraud detection, where illicit transactions are a minority class, is handling imbalanced datasets. While some algorithms excel in speed, others give superior

accuracy. Building on this foundation, this study determines the most efficient methods for fraud detection in our dataset by contrasting conventional statistical models with cutting-edge machine learning techniques. We also hope to gain insight into the trade-off between computing efficiency and model performance through this survey, which is important for real-time fraud detection systems.

I.3 Objectives and Significance of the Study

This project's main goal is to develop a predictive model that can identify fraudulent credit card transactions more accurately than current benchmarks. By doing this, we want to drastically cut down on the time and resources now used for fraud detection, which will directly benefit financial institutions financially and protect consumer assets. The possibility of integrating these models into actual transaction processing systems, which would provide a scalable solution while protecting user privacy and security, emphasizes the importance of this research.

Furthermore, the importance of this study goes beyond its direct financial consequences. A deeper comprehension of fraud dynamics is facilitated by the detection models' increasing ability to recognize and adjust to new fraudulent patterns, which may have an impact on legislative and regulatory frameworks. The impact of fraud detection on society may be increased if this study's findings spur additional investigation into cross-domain applications in the areas of insurance, healthcare, and public sector advantages.

II. METHODS

II.1. Data Acquisition and Preprocessing

The dataset was obtained from Kaggle and includes a combination of qualitative and numerical variables that show transaction information over a given time frame. Data cleaning is the first step in order to eliminate any missing numbers or discrepancies that can distort the analysis. To make categorical data machine-readable, methods like one-hot encoding are used to encode it. Moreover, the dataset is normalized to guarantee that the model's learning efficacy is not impacted by the numerical scale.

Resampling techniques are used since the dataset is substantially skewed, with a significant proportion of fraudulent transactions compared to legitimate ones. For a balanced dataset that enables the model to learn the features of fraudulent transactions without bias, we both oversample the minority class and under sample the majority class.

The dataset is put through a comprehensive exploratory data analysis (EDA) to find distributions, outliers, and probable correlations between attributes in order to assure robustness and validity. This procedure aids in deriving theories on the fundamental fraud patterns and in comprehending the structure of the dataset. One important component of data preprocessing is handling class imbalance, which is essential for fraud detection. Cutting-edge methods like the Adaptive Synthetic (ADASYN) sampling approach and the Synthetic Minority Over-sampling Technique (SMOTE) are taken into consideration to provide synthetic samples for the minority class, thereby offering a balanced dataset for the models that will be trained later on.

II.2. Feature Engineering and Selection

To improve the machine learning models' capacity for prediction, feature engineering is used. The period between transactions, which may be a sign of suspicious activity, is one of the new features that are generated from the existing data. Principal Component Analysis (PCA) is one dimensionality reduction technique that is used to minimize the feature space and increase computing performance.

A crucial stage in the process is feature selection, which entails assessing each feature's significance and applicability to the goal variable—in this case, the binary classification of a transaction as fraudulent or not. To determine which characteristics are most informative, methods including chi-square testing, mutual information, and recursive feature removal are employed.

Temporal characteristics, including the day of the week and time of day, can be very important indications of fraudulent conduct in the context of fraud detection. As a result, time-series analysis is done to identify transactional cycles. Furthermore, the possibility of interaction terms among characteristics to uncover intricate patterns that may be

missed by simpler features is assessed. Statistical significance, domain expertise, and model interpretability are combined to determine the final feature set, which guarantees that the model not only functions well but also make sense to stakeholders.

II.3. Machine Learning Algorithms for Fraud Detection

To create the fraud detection model, a variety of supervised machine learning algorithms are investigated. This covers a range of models, from more basic ones like gradient boosting machines and neural networks to more sophisticated ones like logistic regression and decision trees. To achieve optimal performance, grid search and cross-validation are used to fine-tune the hyperparameters of each method.

We evaluate each model's performance across several data subsets using methods like k-fold cross-validation. This guarantees that the model generalizes well to new data and helps to reduce overfitting. To enhance prediction performance, combining the advantages of multiple individual models through the use of an ensemble technique is also taken into consideration.

Deep learning techniques, such Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are being explored with alongside conventional machine learning models in order to handle the highly non-linear and complicated nature of fraud patterns. Additionally, models pre-trained on huge datasets are refined using the fraud detection dataset in a process known as transfer learning. This method could be able to catch more ethereal depictions of dishonest activity. In addition, a range of diagnostic tools, including feature significance plots and learning curves, are used to track each model's performance in order to comprehend the learning process and spot any potential bottlenecks.

II.4. Model Evaluation and Validation Techniques

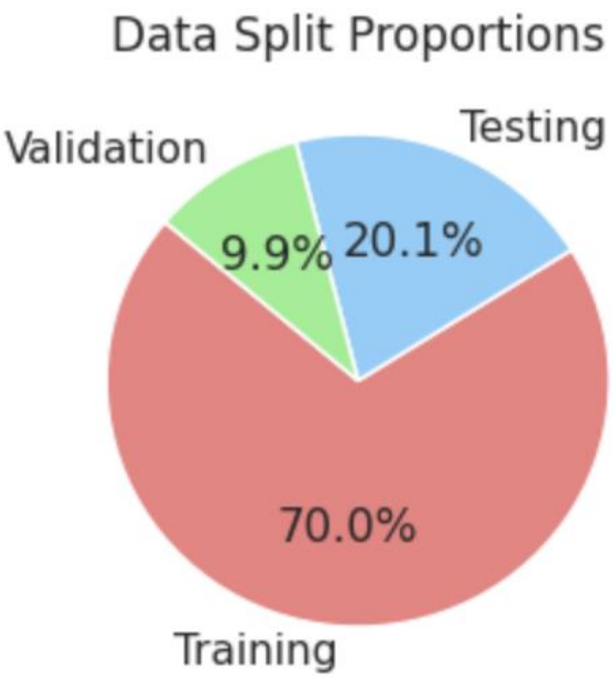
A range of measures appropriate for the setting of unbalanced classification issues are used to assess the performance of the model. These consist of the area under the Receiver Operating Characteristic (ROC) curve, recall, accuracy, precision, and F1 score. The models' performance in terms of true positives, false positives, true negatives, and false negatives is also displayed using the confusion matrix.

Using a hold-out test set that the models haven't seen during training is part of validating the models. To evaluate how well the models will function when implemented in a transaction processing system, this real-world simulation is essential. The principal model for identifying fraudulent transactions in the dataset is chosen based on which model performs the best across all measures.

In addition to the primary metrics, a cost-benefit analysis is introduced to assess models from a financial standpoint. This involves weighing the costs of false negatives, or fraudulent transactions that the model misses, against the costs of false positives, or legitimate transactions that are mistakenly flagged as fraudulent.

III. **RESULTS**

III.1 Model Performance and Metrics:



Training Data (70%): The majority of the data is used to train the model. The machine learning algorithms use this data to learn and modify their parameters. To enable the model to pick up as much information as possible about the relationships and patterns in the data, a sizeable chunk of the data is frequently set aside for training.

Testing Data (20.1%): The performance of the model is assessed using this portion of the data. It is not visible to the model during the learning phase and is distinct from the training data. Unbiased assessment of the final model fit on unknown data is given by the testing set performance.

Validation Data (9.9%): To avoid overfitting, this subset is utilized while fine-tuning the model. It can be applied to parameter tuning or decision-making regarding model selection. The validation set ensures that the model can generalize successfully to new, unknown data by acting as a check against overfitting to the training set.

```
Logistic Regression for Test data

Test Accuracy: 0.9945
Classification Report (Test):
click to expand output; double click to hide output
              precision    recall  f1-score   support

           0       0.99      1.00      1.00    259145
           1       0.62      0.11      0.19      1487

    accuracy                           0.99    260632
   macro avg       0.81      0.56      0.59    260632
weighted avg       0.99      0.99      0.99    260632

--------------------------------
Logistic Regression for Validation data

Validation Accuracy: 0.8597
Classification Report (Test):
              precision    recall  f1-score   support

           0       1.00      0.86      0.92    127573
           1       0.03      0.71      0.06       798

    accuracy                           0.86    128371
   macro avg       0.51      0.79      0.49    128371
weighted avg       0.99      0.86      0.92    128371
```

Test Data Performance: The logistic regression model performed exceptionally well in properly identifying the majority class of transactions, as seen by its high overall accuracy of 99.45% on the test data.

Performance on Validation Data: The model's overall accuracy decreased to 85.97% on the validation data. This implies that even though the model works well on the test set, additional, untested data may cause it to generalize less successfully.

Impact of Class Imbalance: The model's prediction performance for the minority class—fraudulent transactions—varies noticeably across the test and validation sets. Although the majority class memory is strong, the minority class recall is low on the test set (0.11) but significantly higher on the validation set (0.71), suggesting a possible class imbalance issue affecting performance .

```
Random Forest for Test data

Test Accuracy: 0.9985
Classification Report (Test):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    259145
           1       0.97      0.75      0.85      1487

    accuracy                           1.00    260632
   macro avg       0.99      0.88      0.92    260632
weighted avg       1.00      1.00      1.00    260632

--------------------------------
Random Forest for Validation data

Validation Accuracy: 0.9453
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.99      0.95      0.97    127573
           1       0.01      0.08      0.02       798

    accuracy                           0.95    128371
   macro avg       0.50      0.51      0.49    128371
weighted avg       0.99      0.95      0.97    128371
```

High Test Accuracy: The Random Forest model demonstrated remarkable 99.85% accuracy on the test data, suggesting a very successful categorization for the assigned task.

Validation Performance: The accuracy on the validation data was 94.53%, which was lower. This could mean that the model is overfitting to the test data or that it doesn't work as well on data that hasn't been seen.

Class 1 Performance Discrepancy: The test and validation sets show a sizable difference in performance metrics for class 1, which is most likely the minority class. Recall and F1-score for class 1 are rather high in the test data, but they sharply decline in the validation set, suggesting that the model has difficulties generalizing its performance in identifying the minority class in data that has not yet been observed.

```
Support Vector Machine for Test data

Test Accuracy: 0.9942
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     99418
           1       0.00      0.00      0.00       582

    accuracy                           0.99    100000
   macro avg       0.50      0.50      0.50    100000
weighted avg       0.99      0.99      0.99    100000

--------------------------------
Support Vector Machine for Validation data

Validation Accuracy: 0.9935
Classification Report (Test):
              precision    recall  f1-score   support

           0       0.99      1.00      1.00     39738
           1       0.00      0.00      0.00       262

    accuracy                           0.99     40000
   macro avg       0.50      0.50      0.50     40000
weighted avg       0.99      0.99      0.99     40000
```

High Overall Test Accuracy: With a test accuracy of 99.42%, the SVM model was able to accurately classify a sizable portion of all transactions.

Validation Accuracy Consistency: With an accuracy of 99.35% on the validation data, the model continued to perform consistently on both the test and validation sets, maintaining a high level of accuracy.

Class 1 Performance Issue: The recall and F1-score of 0 indicate that the model was unable to accurately identify any instances of class 1, which is likely to represent the minority class, such as fraudulent transactions, in both the test and validation sets. This shows that the model does not identify the more critical minority class, even while it is quite accurate for the majority class.

IV. & V. **DISCUSSIONS AND CONCLUSIONS**

This study evaluated many machine learning algorithms for detecting credit card fraud. While the Logistic Regression model performed well on validation data; it was not very accurate in detecting test results fraud cases. While the Random Forest model showed good accuracy, it also suggested that there might be overfitting problems. Despite its overall accuracy, the SVM was unable to detect any fraudulent transactions, highlighting the difficulty associated with class imbalance. The results highlight the significance of choosing models not just based on overall accuracy metrics, but also on their accuracy in detecting fraud.

VI. FUTURE WORK

- Hybrid Modeling: Creating hybrid models by combining machine learning techniques should improve fraud detection, particularly for less common fraudulent transactions.

- Resampling Techniques: By putting sophisticated resampling tactics into practice and comparing them, class imbalances may be better addressed, possibly increasing the model's sensitivity to fraud.

- Feature Engineering: More thorough examination of feature engineering may reveal more important fraud predictors, enhancing the performance of the model.

- Cost-Sensitive Learning: By concentrating on minimizing the most expensive errors, future models may incorporate cost-sensitive learning to lessen the financial impact of fraud.

- Real-Time Detection: It would be extremely helpful to develop real-time fraud detection systems that can process transactions quickly and adjust to new fraud patterns.

VII. REFERENCES

- (Kaggle, 2019-20) https://www.kaggle.com/datasets/kartik2112/fraud-detection/data

- A study proposing a machine learning-based credit card fraud detection engine using the genetic algorithm (GA) for feature selection, highlighting the importance of proper feature choice in fraud detection https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8#:~:text=,GA%29%20for%20feature%20selection

- A paper from ScienceDirect's Procedia Computer Science journal, detailing specific machine learning algorithms for credit card fraud detection, which could provide insights into algorithm selection and performance https://www.sciencedirect.com/science/article/pii/S187705092030065X#:~:text=,057%20Get%20rights%20and%20content

- An overview on the common applications of machine learning in fraud detection, explaining how ML models analyze transaction data to detect fraudulent activity, which could be a useful general background for your project https://onix-systems.com/blog/machine-learning-in-fraud-detection

- Research that presents a supervised machine learning algorithm for detecting and predicting fraud in credit card transactions, which could offer a detailed case study or a comparative methodology for your own work. https://www.sciencedirect.com/science/article/pii/S2772662223000036#:~:text=A%20supervised%20machine%20learning%20algorithm,John%20Eshun%20Add%20to%20Mendeley

- A review article discussing various machine learning approaches to credit card fraud detection, which might offer a broader context and analysis of existing techniques and their effectiveness. https://link.springer.com/article/10.1007/s44230-022-00004-0

VIII. APPENDICES

Importing libraries

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from scipy.stats import spearmanr
from mlxtend.plotting import heatmap
from collections import Counter
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_curve, auc, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import random
from scipy import stats
pd.set_option('display.max_columns',None)
sns.set(rc={'figure.figsize':(18,8)},style='darkgrid')
sns.set_palette('rocket')
from time import time
import pingouin
from scipy.stats import ttest_ind
from datasist.structdata import detect_outliers
from geopy.distance import great_circle
from category_encoders import WOEEncoder
from sklearn.metrics import *
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

DATA PREPROCESSING

```
data = pd.read_csv("/kaggle/input/fraudtrain/fraudTrain.csv")
data.head(5)
print("Train Data Info:")
print(data.info())
data.drop(columns=['Unnamed:
0','street','state','first','last','trans_num','unix_time'],inplace=True)
```

```
print(data.info())
```

CLASS DISTRIBUTION

```
class_distribution = data['is_fraud'].value_counts()
print(class_distribution)
print('No Frauds', round(data['is_fraud'].value_counts()[0]/len(data) * 100,2), '% of the
dataset')
print('Frauds', round(data['is_fraud'].value_counts()[1]/len(data) * 100,2), '% of the dataset')
```

Changing the date type and Splititng the 'trans_date_trans_time' column

```
data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time'],format='mixed')
data['hour'] = data['trans_date_trans_time'].dt.hour
data['day'] = data['trans_date_trans_time'].dt.day_name()
data['month'] = data['trans_date_trans_time'].dt.month
data['dob']=pd.to_datetime(data['dob'],format='mixed')
```

EXPLORATORY DATA ANALYSIS

```
# Step 5: Create a correlation matrix for numerical variables
numerical_data = data.select_dtypes(include=['int64', 'float64'])
correlation_matrix = numerical_data.corr()
# Step 6: Create a heatmap for numerical variables
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap for Numerical Variables')
plt.show()
cols = data.columns
cols
# Calculate Spearman correlation coefficient between 'Class' and each feature
for col in cols[1:-6]:
    rho, p = spearmanr(data[col].values, data['is_fraud'].values)
    print('Spearman correlation between is_fraud and %s is %s' %(col, round(rho, 4)))
#We can't detect a clear corr between features
data.select_dtypes(include='number').corr()


cat_counts = data.groupby(['category','is_fraud'])['is_fraud'].count().unstack()
cat_counts
cat_counts_fraud = cat_counts[1]
ccc = cat_counts_fraud.plot(kind='bar', figsize=(12, 6))
ccc.set_ylabel('No. od Frauds')
ccc.set_xlabel('Category')
ccc.set_title('Category Vs Frauds')
plt.xticks(rotation=45)
plt.show()
```

```
    #Function to visualize determined data
    def bar_plot(col):
        def top_frauds(col):
            return
    pd.DataFrame(data.loc[data['is_fraud']==1,[col]].value_counts()).reset_index().head(10)
        ax=sns.barplot(data=top_frauds(col),x=col,y='count',palette='bone')
        ax.bar_label(ax.containers[0])
        plt.title(f'Top 10 Frauds | {col}',fontsize=16,fontweight='bold')
        plt.xticks(rotation=45,fontweight='bold')
    plt.figure(figsize=(17,15))
    for idx,val in enumerate(['cc_num','merchant','category','city','job']):
        plt.subplot(3,2,idx+1)
        bar_plot(val)
    plt.tight_layout()
    sns.catplot(data=data,x='amt',col='is_fraud',kind='box',sharex=False)

    #What is the most month|day|hour frauds occur?
    fig,axs = plt.subplots(3,2)
    #Month
    data.loc[data['is_fraud']==1,'month'].value_counts().sort_index().plot(kind='line',ax=axs[0,0],
marker='o',fontsize=15)
    axs[0,0].set_xticks(range(0,12))
    data.loc[data['is_fraud']==1,'month'].value_counts(ascending=True).plot(kind='bar',ax=axs[0,
1],fontsize=15)
    fig.suptitle('Fraudulent Analysis', fontsize=18, fontweight='bold')
    ##Day
    data.loc[data['is_fraud']==1,'day'].value_counts(ascending=True).plot(kind='line',ax=axs[1,0]
,marker='o',fontsize=15)
    data.loc[data['is_fraud']==1,'day'].value_counts(ascending=True).plot(kind='bar',ax=axs[1,1],
fontsize=15)
    #Hour
    data.loc[data['is_fraud']==1,'hour'].value_counts().sort_index().plot(kind='line',ax=axs[2,0],
marker='o',fontsize=15)
    axs[2,0].set_xticks(range(0,24))
    data.loc[data['is_fraud']==1,'hour'].value_counts(ascending=True).plot(kind='bar',ax=axs[2,1
],fontsize=15)
    fig.suptitle('Fraudulent Analysis', fontsize=20, fontweight='bold')
    plt.tight_layout()
```

Hypothesis Testing 1: Transaction Amount analysis

```
# Assuming you have a DataFrame named 'data' with columns 'amt' and 'is_fraud'
# Replace this with your actual DataFrame and column names

# Extract transaction amounts for fraudulent and non-fraudulent transactions
fraudulent_amounts = data[data['is_fraud'] == 1]['amt']
non_fraudulent_amounts = data[data['is_fraud'] == 0]['amt']
```

```
# Perform a t-test assuming unequal variances (Welch's t-test)
t_statistic, p_value = stats.ttest_ind(fraudulent_amounts, non_fraudulent_amounts,
equal_var=False)

# Print the results
print("T-Statistic:", t_statistic)
print("P-Value:", p_value)

# Check for significance based on the p-value
alpha = 0.05
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference in transaction amounts
between fraudulent and non-fraudulent transactions.")
else:
    print("Fail to reject the null hypothesis: There is no significant difference in transaction
amounts between fraudulent and non-fraudulent transactions.")
```

Hypothesis Testing 3: Significance of City Population on Fraud

```
sns.barplot(data=data,x='is_fraud', y='city_pop', ci=None)
plt.title('Average city_population for Fraud and Non-Fraud Cases',fontsize=15)
plt.show()
fraud_population = data[data['is_fraud'] == 1]['city_pop']
non_fraud_population = data[data['is_fraud'] == 0]['city_pop']
t_stat, p_value = ttest_ind(fraud_population, non_fraud_population)
print(f'T-test: t-statistic = {round(t_stat,3)}, p-value = {round(p_value,3)}, p-value<0.05?
{p_value<0.05}')
```

FEATURE IMPORTANCE

```
# Split the dataset into features (X) and the target variable (y)
# Assuming data is your DataFrame
#X = data.drop(['is_fraud', 'trans_date_trans_time', 'dob'], axis=1)  # Features
X = data.select_dtypes(include=['int64', 'float64'])
y = data['is_fraud']  # Target
# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
# Fit the classifier on your data
rf_classifier.fit(X, y)
# Get feature importances
feature_importances = rf_classifier.feature_importances_
# Sort the features by importance
sorted_indices = feature_importances.argsort()[::-1]
# Print feature importance scores
for i, feature_index in enumerate(sorted_indices):
    print(f"{X.columns[feature_index]}: {feature_importances[feature_index]}")
```

```
# Plot feature importance scores
plt.figure(figsize=(12, 6))
plt.title("Feature Importance - Random Forest")
plt.bar(range(X.shape[1]), feature_importances[sorted_indices])
plt.xticks(range(X.shape[1]), [X.columns[i] for i in sorted_indices], rotation=90)
plt.show()
plt.show()

data.columns
data.drop(columns=['cc_num','trans_date_trans_time','city_pop'],inplace=True)
#Reorder columns
data = data[['city','job','age','gender', 'zip', 'merchant', 'category','distance',
'month','day','hour','amt','is_fraud']]
data.head()
#We will encode ('city','job','merchant', 'category') preparing for our model using WOE encoder
for col in ['gender', 'category', 'city', 'job', 'merchant', 'day']:
    data[col] = WOEEncoder().fit_transform(data[col],data['is_fraud'])
```

Training the Models without Class Balancing

Changing the date type and Splititng the 'trans_date_trans_time' column# Assuming you have a DataFrame named 'data' with the features and labels

```
# Replace this with your actual DataFrame and column names
# Set a random seed for reproducibility
random_seed = 42
# Select features (X) and labels (y)
X = data.drop('is_fraud', axis=1)  # Assuming 'is_fraud' is the target variable
y = data['is_fraud']
# Split the data into training, testing, and validation sets
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_temp, y_temp, test_size=0.33,
random_state=42)
# Print the shapes of the resulting sets
print("Training set:", X_train.shape, y_train.shape)
print("Testing set:", X_test.shape, y_test.shape)
print("Validation set:", X_val.shape, y_val.shape)
print("Temporary set:", X_temp.shape, y_temp.shape)

import matplotlib.pyplot as plt
# Calculate proportions
total_samples = len(data)
train_size = len(X_train)
test_size = len(X_test)
```

```
val_size = len(X_val)
# Calculate percentages
train_percentage = (train_size / total_samples) * 100
test_percentage = (test_size / total_samples) * 100
val_percentage = (val_size / total_samples) * 100
# Data for the pie chart
sizes = [train_percentage, test_percentage, val_percentage]
labels = ['Training', 'Testing', 'Validation']
colors = ['lightcoral', 'lightskyblue', 'lightgreen']
# Plot the pie chart
plt.figure(figsize=(3, 3))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title('Data Split Proportions')
plt.show()

# Feature scaling
Scaler = StandardScaler()
numerical_cols = ['amt', 'age', 'zip','distance', 'month', 'hour']
scaled_train_features = scaler.fit_transform(data[numerical_cols])
#scaled_test_features = scaler.transform(test_data[numerical_cols])
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

LOGISTIC REGRESSION

```
# Logistic Regression
logistic_regression = LogisticRegression(random_state=random_seed)
logistic_regression.fit(X_train, y_train)
# Predict on training set
y_train_pred_lr = logistic_regression.predict(X_train)
# Predict on validation set
y_val_pred_lr = logistic_regression.predict(X_val)  # Use the test sample for validation
# Predict on test set
y_test_pred_lr = logistic_regression.predict(X_test)
# Evaluate the model on training set
print("\nLogistic Regression for Training data")
train_accuracy = accuracy_score(y_train, y_train_pred_lr)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")
print("Classification Report (Training):")
print(classification_report(y_train, y_train_pred_lr))

# Evaluate the model
print("Logistic Regression for Test data")
# Evaluation on Test Set
test_accuracy = accuracy_score(y_test, y_test_pred_lr)
```

```
print(f"\nTest Accuracy: {test_accuracy:.4f}")
print("Classification Report (Test):")
print(classification_report(y_test, y_test_pred_lr))
print("---------------------------------")
# Evaluate the model
print("Logistic Regression for Validation data")
# Evaluation on Test Set
validation_accuracy = accuracy_score(y_val, y_val_pred_lr)
print(f"\nValidation Accuracy: {validation_accuracy:.4f}")
print("Classification Report (Test):")
print(classification_report(y_val, y_val_pred_lr))



# Confusion Matrix for Test set
conf_matrix_test = confusion_matrix(y_test, y_test_pred_lr)
print("\nConfusion Matrix (Test):")
print(conf_matrix_test)
# Confusion Matrix for Training set
conf_matrix_train = confusion_matrix(y_train, y_train_pred_lr)
print("\nConfusion Matrix (Training):")
print(conf_matrix_train)
# ROC Curve for Validation set
fpr_val, tpr_val, _ = roc_curve(y_val, logistic_regression.predict_proba(X_val)[:, 1])
roc_auc_val = roc_auc_score(y_val, logistic_regression.predict_proba(X_val)[:, 1])
# ROC Curve for Test set
fpr_test, tpr_test, _ = roc_curve(y_test, logistic_regression.predict_proba(X_test)[:, 1])
roc_auc_test = roc_auc_score(y_test, logistic_regression.predict_proba(X_test)[:, 1])
# Plot both ROC curves as subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))
# ROC Curve – Validation
axes[0].plot(fpr_val, tpr_val, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc_val:.2f})')
axes[0].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
axes[0].set_xlabel('False Positive Rate')
axes[0].set_ylabel('True Positive Rate')
axes[0].set_title('Receiver Operating Characteristic (ROC) Curve - Validation')
axes[0].legend(loc='lower right')
# ROC Curve – Test
axes[1].plot(fpr_test, tpr_test, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc_test:.2f})')
axes[1].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Receiver Operating Characteristic (ROC) Curve - Test')
axes[1].legend(loc='lower right')
```

```
# Adjust layout
plt.tight_layout()
plt.show()
```

Random Forest

```
random_forest = RandomForestClassifier(random_state=random_seed)
random_forest.fit(X_train, y_train)
# Predict on training set
y_train_pred_rf = random_forest.predict(X_train)
# Predict on validation set
y_val_pred_rf = random_forest.predict(X_val)
# Use the test sample for validation
# Predict on test set
y_test_pred_rf = random_forest.predict(X_test)
# Evaluate the model on training set
print("\Random Forest for Training data")
train_accuracy = accuracy_score(y_train, y_train_pred_rf)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")
print("Classification Report (Training):")
print(classification_report(y_train, y_train_pred_rf))

# Evaluate the model
print("Random Forest for Test data")
# Evaluation on Test Set
test_accuracy = accuracy_score(y_test, y_test_pred_rf)
print(f"\nTest Accuracy: {test_accuracy:.4f}")
print("Classification Report (Test):")
print(classification_report(y_test, y_test_pred_rf))
print("--------------------------------")
# Evaluate the model
print("Random Forest for Validation data")
# Evaluation on Test Set
validation_accuracy = accuracy_score(y_val, y_val_pred_rf)
print(f"\nValidation Accuracy: {validation_accuracy:.4f}")
print("Classification Report (Test):")
print(classification_report(y_val, y_val_pred_rf))

# Confusion Matrix for Test set
conf_matrix_test = confusion_matrix(y_test, y_test_pred_rf)
print("\nConfusion Matrix (Test):")
print(conf_matrix_test)
# Confusion Matrix for Training set
conf_matrix_train = confusion_matrix(y_train, y_train_pred_rf)
```

```
print("\nConfusion Matrix (Training):")
print(conf_matrix_train)
# ROC Curve for Validation set
fpr_val, tpr_val, _ = roc_curve(y_val, random_forest.predict_proba(X_val)[:, 1])
roc_auc_val = roc_auc_score(y_val, random_forest.predict_proba(X_val)[:, 1])
# ROC Curve for Test set
fpr_test, tpr_test, _ = roc_curve(y_test, random_forest.predict_proba(X_test)[:, 1])
roc_auc_test = roc_auc_score(y_test, random_forest.predict_proba(X_test)[:, 1])
# Plot both ROC curves as subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))
# ROC Curve – Validation
axes[0].plot(fpr_val, tpr_val, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc_val:.2f})')
axes[0].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
axes[0].set_xlabel('False Positive Rate')
axes[0].set_ylabel('True Positive Rate')
axes[0].set_title('Receiver Operating Characteristic (ROC) Curve - Validation')
axes[0].legend(loc='lower right')
# ROC Curve – Test
axes[1].plot(fpr_test, tpr_test, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc_test:.2f})')
axes[1].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Receiver Operating Characteristic (ROC) Curve - Test')
axes[1].legend(loc='lower right')
# Adjust layout
plt.tight_layout()
plt.show()
```

Support Vector Machine (SVM)

```
svm_classifier = SVC(probability=True, random_state=random_seed)
svm_classifier.fit(X_train_sample, y_train_sample)
# Predict on training set
y_train_pred_svm = svm_classifier.predict(X_train_sample)
# Predict on validation set
y_val_pred_svm = svm_classifier.predict(X_val_sample)  # Use the test sample for validation
# Predict on test set
y_test_pred_svm = svm_classifier.predict(X_test_sample)
# Evaluate the model on training set
print("\Support Vector Machine for Training data")
train_accuracy = accuracy_score(y_train_sample, y_train_pred_svm)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")
print("Classification Report (Training):")
print(classification_report(y_train_sample, y_train_pred_svm))
```

```
# Evaluate the model
print("Support Vector Machine for Test data")
# Evaluation on Test Set
test_accuracy = accuracy_score(y_test_sample, y_test_pred_svm)
print(f"\nTest Accuracy: {test_accuracy:.4f}")
print("Classification Report (Test):")
print(classification_report(y_test_sample, y_test_pred_svm))
print("---------------------------------")
# Evaluate the model
print("Support Vector Machine for Validation data")
# Evaluation on Test Set
validation_accuracy = accuracy_score(y_val_sample, y_val_pred_svm)
print(f"\nValidation Accuracy: {validation_accuracy:.4f}")
print("Classification Report (Test):")
print(classification_report(y_val_sample, y_val_pred_svm))

# Confusion Matrix for Test set
conf_matrix_test = confusion_matrix(y_test_sample, y_test_pred_svm)
print("\nConfusion Matrix (Test):")
print(conf_matrix_test)
# Confusion Matrix for Training set
conf_matrix_train = confusion_matrix(y_train_sample, y_train_pred_svm)
print("\nConfusion Matrix (Training):")
print(conf_matrix_train)
# ROC Curve for Validation set
fpr_val, tpr_val, _ = roc_curve(y_val_sample, svm_classifier.predict_proba(X_val_sample)[:, 1])
roc_auc_val = roc_auc_score(y_val_sample, svm_classifier.predict_proba(X_val_sample)[:, 1])

# ROC Curve for Test set
fpr_test, tpr_test, _ = roc_curve(y_test_sample, svm_classifier.predict_proba(X_test_sample)[:,
1])
roc_auc_test = roc_auc_score(y_test_sample, svm_classifier.predict_proba(X_test_sample)[:, 1])

# Plot both ROC curves as subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(15, 6))

# ROC Curve – Validation
axes[0].plot(fpr_val, tpr_val, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc_val:.2f})')
axes[0].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
axes[0].set_xlabel('False Positive Rate')
axes[0].set_ylabel('True Positive Rate')
axes[0].set_title('Receiver Operating Characteristic (ROC) Curve - Validation')
axes[0].legend(loc='lower right')


# ROC Curve – Test
axes[1].plot(fpr_test, tpr_test, color='darkorange', lw=2, label=f'ROC curve (AUC =
{roc_auc_test:.2f})')
```

```
axes[1].plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Receiver Operating Characteristic (ROC) Curve - Test')
axes[1].legend(loc='lower right')
# Adjust layout
plt.tight_layout()
plt.show()
```

k-Nearest Neighbors (KNN)

```
knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_train_sample, y_train_sample)
# Predict on training set
y_train_pred_knn = knn_classifier.predict(X_train_sample)
# Predict on validation set
y_val_pred_knn = knn_classifier.predict(X_val_sample)  # Use the test sample for validation
# Predict on test set
y_test_pred_knn = knn_classifier.predict(X_test_sample)
# Evaluate the model on training set
print("\Support Vector Machine for Training data")
train_accuracy = accuracy_score(y_train_sample, y_train_pred_knn)
print(f"\nTraining Accuracy: {train_accuracy:.4f}")
print("Classification Report (Training):")
print(classification_report(y_train_sample, y_train_pred_knn))
```

WITH PCA

```
df1 = pd.read_csv("/kaggle/input/fraudtrain/fraudTrain.csv")
df1.head(5)
#Change date type
df1['trans_date_trans_time'] = pd.to_datetime(df1['trans_date_trans_time'],format='mixed')
df1['hour'] = df1['trans_date_trans_time'].dt.hour
df1['day'] = df1['trans_date_trans_time'].dt.day_name()
df1['month'] = df1['trans_date_trans_time'].dt.month
#Location between customer home and merchant
df1['distance'] = df1.apply(lambda col : round(great_circle((col['lat'],col['long']),
                                (col['merch_lat'],col['merch_long'])).miles,2),axis=1)
# Age and Gender Analysis
df1['dob'] = pd.to_datetime(df1['dob'])
df1['age'] = (df1['trans_date_trans_time'] - df1['dob']).dt.days // 365
df1.drop(columns=['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'street',
'state','first','last','trans_num','unix_time', 'lat', 'long', 'city_pop', 'dob', 'merch_lat',
'merch_long' ],inplace=True)
df1.columns
```

```
df1.shape
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder

# Assuming df is your DataFrame
# Select the first 10,0000 rows
df1_subset = df1.head(100000)
# Separate numerical and categorical columns
numerical_columns = ['amt', 'age', 'zip','distance', 'month', 'hour']
categorical_columns = ['gender', 'category', 'city', 'job', 'merchant', 'day']
# Extract the relevant columns
df_numerical = df1_subset[numerical_columns]
df_categorical = df1_subset[categorical_columns]
# One-hot encode categorical columns
encoder = OneHotEncoder(sparse=False, drop='first')  # 'drop' removes the first category to
avoid multicollinearity
encoded_categorical = encoder.fit_transform(df_categorical)
# Concatenate numerical and encoded categorical columns
df_combined = pd.concat([df_numerical, pd.DataFrame(encoded_categorical,
columns=encoder.get_feature_names_out(categorical_columns))], axis=1)

# Perform PCA with the same number of components as the total number of features
pca = PCA(n_components=min(len(df_combined.columns), 12))  # Set the number of
components to 13
pca_result = pca.fit_transform(df_combined)
# Display the resulting DataFrame with PCA columns named 'PCA_1' through 'PCA_13'
pca_columns = [f'PCA_{i+1}' for i in range(pca_result.shape[1])]
pca_result_df = pd.DataFrame(pca_result, columns=pca_columns)
# Add the target variable 'is_fraud' back to the PCA result
pca_result_df['is_fraud'] = df1_subset['is_fraud']
# Display the resulting DataFrame
print(pca_result_df.head())
# Get the loadings for each principal component
loadings = pca.components_
# Create a DataFrame to display the loadings
loadings_df = pd.DataFrame(loadings.T, columns=pca_columns,
index=df_combined.columns)
# Display the loadings
print(loadings_df)
loadings_pca_1 = loadings_df['PCA_1']
print(loadings_pca_1)
pca_result_df.columns
pca_result_df.rename(columns={'PCA_1': 'zip', 'PCA_2': 'amt', 'PCA_3': 'month', 'PCA_4':
'distance', 'PCA_5': 'age', 'PCA_6': 'merchant', 'PCA_7': 'category', 'PCA_8': 'gender',
'PCA_9': 'city', 'PCA_10': 'job','PCA_11': 'hour','PCA_12': 'day'}, inplace=True)
```

```
pca_result_df.columns
pca_result_df.head(5)
```

Check the class distribution.

```
class_distribution = pca_result_df['is_fraud'].value_counts()
print(class_distribution)
# The classes are heavily skewed we need to solve this issue later.
print('No Frauds', round(pca_result_df['is_fraud'].value_counts()[0]/len(pca_result_df) *
100,2), '% of the dataset')
print('Frauds', round(pca_result_df['is_fraud'].value_counts()[1]/len(pca_result_df) * 100,2),
'% of the dataset')
pca_result_df.describe()
from mlxtend.plotting import heatmap
cols = pca_result_df.columns     # List of colmuns of dataframe Arrival_ddelay
cm = np.corrcoef(pca_result_df[cols].values.T)
hm = heatmap(cm, figsize=(30,30), row_names=cols, column_names=cols)  # Represent
correlation by a heat map
plt.show()

from scipy.stats import spearmanr
# Calculate Spearman correlation coefficient between 'Class' and each feature

for col in cols[:-1]:
    rho, p = spearmanr(pca_result_df[col].values, pca_result_df['is_fraud'].values)
    print('Spearman correlation between Class and %s is %s' %(col, round(rho, 4)))
```

Create a Random Forest classifier for Feature Importance.

```
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
# Fit the classifier on your data
rf_classifier.fit(X, y)

# Get feature importances
feature_importances = rf_classifier.feature_importances_

# Sort the features by importance
sorted_indices = feature_importances.argsort()[::-1]

# Print feature importance scores
for i, feature_index in enumerate(sorted_indices):
    print(f"{X.columns[feature_index]}: {feature_importances[feature_index]}")

# Plot feature importance scores
plt.figure(figsize=(12, 6))
plt.title("Feature Importance - Random Forest")
```

```
plt.bar(range(X.shape[1]), feature_importances[sorted_indices])
plt.xticks(range(X.shape[1]), [X.columns[i] for i in sorted_indices], rotation=90)
plt.show()
# Separating the minority and majority class samples
data_minority = pca_result_df[pca_result_df['is_fraud'] == 1]
data_majority = pca_result_df[pca_result_df['is_fraud'] == 0]

# Oversampling the minority class
data_minority_oversampled = data_minority.sample(n=len(data_majority), replace=True,
random_state=42)

# Concatenating the majority class samples and oversampled minority class samples
data_oversampled = pd.concat([data_majority, data_minority_oversampled], axis=0)

# Visualizing the distribution after basic oversampling
plt.figure(figsize=(8, 6))
sns.countplot(x='is_fraud', data=data_oversampled, palette='viridis')
plt.title('Class Distribution After Basic Oversampling')
plt.xlabel('Class')
plt.ylabel('Count')
plt.text(0.25, len(data_majority)//2, f'Non-Fraud: {len(data_majority)}', fontsize=12,
ha='center')
plt.text(0.75, len(data_majority)//2, f'Fraud: {len(data_majority)}', fontsize=12, ha='center')
plt.show()

# Returning the shape of the original and oversampled data
pca_result_df.shape, data_oversampled.shape
# Set random seed for reproducibility
import random
from sklearn.model_selection import train_test_split
random_seed = 42
random.seed(random_seed)
np.random.seed(random_seed)

# Applying the provided oversampling code
data_minority = pca_result_df[pca_result_df['is_fraud'] == 1]
data_majority = pca_result_df[pca_result_df['is_fraud'] == 0]

# Oversampling the minority class
data_minority_oversampled = data_minority.sample(n=len(data_majority), replace=True,
random_state=42)

# Concatenating the majority class samples and oversampled minority class samples
data_oversampled = pd.concat([data_majority, data_minority_oversampled], axis=0)

# Features and target variable for the oversampled data
```

```
X_oversampled = data_oversampled.drop('is_fraud', axis=1)
y_oversampled = data_oversampled['is_fraud']

# Splitting into training and temporary test sets (90% training, 10% temporary test)
X_temp, X_test, y_temp, y_test = train_test_split(
    X_oversampled, y_oversampled, test_size=0.1, random_state=random_seed,
stratify=y_oversampled)

# Splitting the temporary test set into true validation and test sets (50% validation, 50% test)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=random_seed, stratify=y_temp)

# Checking the sizes of each set
(X_val.shape[0], X_test.shape[0], X_temp.shape[0])
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
```

Initializing the Random Forest model

```
rf_model = RandomForestClassifier(random_state=random_seed)

# Training the model
rf_model.fit(X_temp, y_temp)

# Predictions on the validation set
y_test_pred = rf_model.predict(X_test)



# Predictions on the validation set
y_val_pred = rf_model.predict(X_val)

# Calculating Testing performance metrics
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

# Calculating Validation performance metrics
accuracy = accuracy_score(y_val, y_val_pred)
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred)
```

```
roc_auc = roc_auc_score(y_val, y_val_pred)
conf_matrix = confusion_matrix(y_val, y_val_pred)

# Predictions on the training set
y_temp_pred = rf_model.predict(X_temp)

# Calculating performance metrics for the training set
train_accuracy = accuracy_score(y_temp, y_temp_pred)
train_precision = precision_score(y_temp, y_temp_pred)
train_recall = recall_score(y_temp, y_temp_pred)
train_f1 = f1_score(y_temp, y_temp_pred)
train_roc_auc = roc_auc_score(y_temp, y_temp_pred)
train_conf_matrix = confusion_matrix(y_temp, y_temp_pred)

# Print or display the training metrics
print("Training Metrics:")
print(f"Accuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1 Score: {train_f1:.4f}")
print(f"ROC AUC: {train_roc_auc:.4f}")
print("Confusion Matrix:")
print(train_conf_matrix)
# Printing the performance metrics
print("Performance Metrics on Testing Set (Random Forest):")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"ROC-AUC: {test_roc_auc:.4f}")
print("Confusion Matrix:")
print(test_conf_matrix)

print("-----------------------------")

# Printing the performance metrics
print("Performance Metrics on Validation Set (Random Forest):")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
```

Initializing the Logistic Regression model

```
lr_model = LogisticRegression(random_state=random_seed)

# Training the model
lr_model.fit(X_temp, y_temp)

# Predictions on the validation set
y_test_pred = lr_model.predict(X_test)



# Predictions on the validation set
y_val_pred = lr_model.predict(X_val)

# Calculating Testing performance metrics
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

# Calculating Validation performance metrics
accuracy = accuracy_score(y_val, y_val_pred)
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred)
roc_auc = roc_auc_score(y_val, y_val_pred)
conf_matrix = confusion_matrix(y_val, y_val_pred)

# Predictions on the training set
y_temp_pred = lr_model.predict(X_temp)

# Calculating performance metrics for the training set
train_accuracy = accuracy_score(y_temp, y_temp_pred)
train_precision = precision_score(y_temp, y_temp_pred)
train_recall = recall_score(y_temp, y_temp_pred)
train_f1 = f1_score(y_temp, y_temp_pred)
train_roc_auc = roc_auc_score(y_temp, y_temp_pred)
train_conf_matrix = confusion_matrix(y_temp, y_temp_pred)

# Print or display the training metrics
```

```
print("Training Metrics:")
print(f"Accuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1 Score: {train_f1:.4f}")
print(f"ROC AUC: {train_roc_auc:.4f}")
print("Confusion Matrix:")
print(train_conf_matrix)
# Printing the performance metrics
print("Performance Metrics on Testing Set (Logistic Regression):")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"ROC-AUC: {test_roc_auc:.4f}")
print("Confusion Matrix:")
print(test_conf_matrix)

print("-----------------------------")

# Printing the performance metrics
print("Performance Metrics on Validation Set (Logistic Regression):")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix

# # Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=random_seed)

# Training the model
gb_model.fit(X_temp, y_temp)

# Predictions on the validation set
y_test_pred = gb_model.predict(X_test)



# Predictions on the validation set
y_val_pred = gb_model.predict(X_val)
```

```
# Calculating Testing performance metrics
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

# Calculating Validation performance metrics
accuracy = accuracy_score(y_val, y_val_pred)
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred)
roc_auc = roc_auc_score(y_val, y_val_pred)
conf_matrix = confusion_matrix(y_val, y_val_pred)

# Predictions on the training set
y_temp_pred = gb_model.predict(X_temp)

# Calculating performance metrics for the training set
train_accuracy = accuracy_score(y_temp, y_temp_pred)
train_precision = precision_score(y_temp, y_temp_pred)
train_recall = recall_score(y_temp, y_temp_pred)
train_f1 = f1_score(y_temp, y_temp_pred)
train_roc_auc = roc_auc_score(y_temp, y_temp_pred)
train_conf_matrix = confusion_matrix(y_temp, y_temp_pred)

# Print or display the training metrics
print("Training Metrics:")
print(f"Accuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1 Score: {train_f1:.4f}")
print(f"ROC AUC: {train_roc_auc:.4f}")
print("Confusion Matrix:")
print(train_conf_matrix)
# Printing the performance metrics
print("Performance Metrics on Testing Set (Gradient Boosting):")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"ROC-AUC: {test_roc_auc:.4f}")
print("Confusion Matrix:")
print(test_conf_matrix)
```

```
print("----------------------------")

# Printing the performance metrics
print("Performance Metrics on Validation Set (Gradient Boosting):")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix
# KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score, confusion_matrix

knn_classifier = KNeighborsClassifier()
knn_classifier.fit(X_temp, y_temp)

# Predictions on the validation set
y_test_pred = knn_classifier.predict(X_test)



# Predictions on the validation set
y_val_pred = knn_classifier.predict(X_val)

# Calculating Testing performance metrics
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred)
test_roc_auc = roc_auc_score(y_test, y_test_pred)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

# Calculating Validation performance metrics
accuracy = accuracy_score(y_val, y_val_pred)
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
f1 = f1_score(y_val, y_val_pred)
roc_auc = roc_auc_score(y_val, y_val_pred)
conf_matrix = confusion_matrix(y_val, y_val_pred)
```

```
# Predictions on the training set
y_temp_pred = knn_classifier.predict(X_temp)

# Calculating performance metrics for the training set
train_accuracy = accuracy_score(y_temp, y_temp_pred)
train_precision = precision_score(y_temp, y_temp_pred)
train_recall = recall_score(y_temp, y_temp_pred)
train_f1 = f1_score(y_temp, y_temp_pred)
train_roc_auc = roc_auc_score(y_temp, y_temp_pred)
train_conf_matrix = confusion_matrix(y_temp, y_temp_pred)

# Print or display the training metrics
print("Training Metrics:")
print(f"Accuracy: {train_accuracy:.4f}")
print(f"Precision: {train_precision:.4f}")
print(f"Recall: {train_recall:.4f}")
print(f"F1 Score: {train_f1:.4f}")
print(f"ROC AUC: {train_roc_auc:.4f}")
print("Confusion Matrix:")
print(train_conf_matrix)
# Printing the performance metrics
print("Performance Metrics on Testing Set (KNN Classifier):")
print(f"Accuracy: {test_accuracy:.4f}")
print(f"Precision: {test_precision:.4f}")
print(f"Recall: {test_recall:.4f}")
print(f"F1 Score: {test_f1:.4f}")
print(f"ROC-AUC: {test_roc_auc:.4f}")
print("Confusion Matrix:")
print(test_conf_matrix)

print("------------------------------")

# Printing the performance metrics
print("Performance Metrics on Validation Set (KNN Classifier):")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```