Data Science Project

Credit Card Fraud Detection

Phase 5 - Documentation

Team ID: TG34
Team Members:
1. Prinitha S - 211521106126
2. Shalini S- 211521106145
3. Poojashree S- 211521106119
4. Haritha B- 211521106057
5. Prateekshitha M– 211521106122

Problem Statement:

This project focuses on leveraging data science techniques to develop an efficient credit card fraud detection system. The project involves *data collection, preprocessing, feature engineering, model selection, and deployment, with continuous monitoring and a feedback loop* to adapt to evolving fraud patterns. Success hinges on high accuracy and real-time efficiency while maintaining the security of electronic transactions

Objectives:

- **High Accuracy:** Develop a credit card fraud detection system with a primary objective of achieving a high level of accuracy in distinguishing between legitimate and fraudulent transactions.
- **Real-Time Efficiency:** Ensure that the system operates in real-time, promptly classifying transactions, and minimizing any disruptions to legitimate transactions.
- **Adaptability:** Create a system that can continuously adapt to evolving fraud patterns by monitoring and updating the model with new data.
- **Compliance and Security:** Establish robust legal compliance to protect the interests of financial institutions and cardholders while effectively detecting and preventing fraud.

Investigation:

1. **Exploratory Data Analysis (EDA):** To understand the distribution of characteristics, spot possible outliers, and learn more about transaction trends for both fraudulent and non-fraudulent situations, start by thoroughly examining the historical transaction data.

2. **Feature Importance Analysis:** Assess the relevance and impact of various features in fraud detection. Use techniques like feature importance scores and correlations to select and engineer the most informative attributes for the models.

3. **Model Evaluation and Selection:** To determine which machine learning models are best for detecting fraud, compare the effectiveness of several models, including support vector machines, random forests, and logistic regression.

4. **Threshold Optimization**: Investigate different decision thresholds for classifying transactions as fraudulent or non-fraudulent. Fine-tune these thresholds to balance the trade-off between false positives and false negatives, aligning with specific business requirements.

Example Program:

```
[ ]: #Uploading and Displaying Dataset
     import pandas as pd
     df = pd.read_csv('/content/creditcard.csv')
     print(df)
```

```
        Time        V1        V2        V3        V4        V5        V6  \
0          0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1          0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2          1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3          1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4          2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
...      ...       ...       ...       ...       ...       ...       ...
61486  49862 -0.987401  1.056011  1.184880 -0.738908 -0.080874 -0.167563
61487  49862  0.910338 -0.976578  1.308543  0.352233 -1.228617  1.053119
61488  49863  1.033813 -0.261495  1.329732  1.820041 -0.974670  0.384867
61489  49863  1.446884 -0.263871 -0.192448 -0.660946 -0.598712 -1.387964
61490  49863 -1.256173  0.190250  0.486835  0.174933  0.615589  0.667585

             V7        V8        V9  ...       V21       V22       V23  \
0      0.239599  0.098698  0.363787  ... -0.018307  0.277838 -0.110474
1     -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288
2      0.791461  0.247676 -1.514654  ...  0.247998  0.771679  0.909412
3      0.237609  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321
4      0.592941 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458
...         ...       ...       ...  ...       ...       ...       ...
61486  0.224570  0.650254 -0.454139  ... -0.069511 -0.149341 -0.022932
61487 -1.189630  0.422435 -0.756288  ...  0.081774  0.571815 -0.057231
61488 -0.580740  0.317767  1.384514  ... -0.398950 -0.748977  0.126709
61489 -0.001654 -0.430535 -1.207218  ...  0.194922  0.511846 -0.170092
61490  0.983123  0.049528  0.077344  ... -0.024591  0.195936 -0.204945

            V24       V25       V26       V27       V28  Amount  Class
0      0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62    0.0
1     -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69    0.0
2     -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66    0.0
3     -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50    0.0
4      0.141267 -0.206010  0.502292  0.219422  0.215153   69.99    0.0
...         ...       ...       ...       ...       ...     ...    ...

1

61486  0.040407 -0.291335  0.307241  0.190307  0.104660    1.00    0.0
61487 -0.292605  0.065126 -0.226463  0.114948  0.054243  129.50    0.0
61488  0.367238  0.342699 -0.527094  0.082915  0.031702   22.02    0.0
61489  0.445334  0.805625 -0.086404 -0.026795  0.002982   15.00    0.0
61490 -0.850277 -0.393942 -0.584722 -0.117240  0.287411     NaN    NaN
```

```
[ ]: import pandas as pd
     df = pd.read_csv('/content/creditcard.csv')

     # creating a list of column names
     Column_name = pd.DataFrame(df.columns)
     print('List of column names :',Column_name)
```

```
List of column names :          0
0     Time
1       V1
2       V2
3       V3
4       V4
5       V5
6       V6
7       V7
8       V8
9       V9
10     V10
11     V11
12     V12
13     V13
14     V14
15     V15
16     V16
17     V17
18     V18
19     V19
20     V20
21     V21
22     V22
23     V23
24     V24
25     V25
26     V26
27     V27
28     V28
29  Amount
30   Class
```



```
[ ]: import pandas as pd
     df = pd.read_csv('/content/creditcard.csv')

     #Displaying First 5 rows
     print(df.head())
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4     2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62    0.0
1  0.125895 -0.008983  0.014724    2.69    0.0
2 -0.139097 -0.055353 -0.059752  378.66    0.0
3 -0.221929  0.062723  0.061458  123.50    0.0
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns]
```

```
[ ]: import pandas as pd
     df = pd.read_csv('/content/creditcard.csv')

     #Displaying Last 5 rows
     print(df.tail())
```

```
        Time        V1        V2        V3        V4        V5        V6  \
75352  56019 -0.330203  1.265306  0.703968  0.943073  0.020692 -0.334221
75353  56020 -0.824159  0.689632  0.238364  0.843827 -1.723679  2.003802
75354  56020  0.929735 -0.299633  0.394750  0.540756  0.272645  1.656454
75355  56021 -1.657683 -0.426294  1.687198 -1.454421 -2.383477  1.305648
75356  56021 -1.711121 -0.369377  1.440787  0.136071  0.517265  0.196718

             V7        V8        V9  _       V21       V22       V23  \
75352  0.406527  0.313829 -0.924290  _  0.222834  0.653481 -0.024697
75353  3.194546 -0.727345 -0.680086  _ -0.092779  0.491220 -0.376126
75354 -0.410577  0.600046  0.184701  _ -0.038120  0.102605  0.094452
75355  0.658390  0.543973 -1.068176  _  0.166907  0.460901 -0.059376
75356 -0.006951  0.325528 -1.817459  _ -0.365864 -0.524456  0.085383
```

3

```
            V24       V25       V26       V27       V28  Amount  Class
75352  0.123168 -0.154517 -0.303343 -0.020581 -0.017914   17.45    0.0
75353 -0.768536 -0.152191 -0.255962  0.014022 -0.227837  570.00    0.0
75354 -0.985089  0.094890  0.376660  0.038140 -0.005591   44.43    0.0
75355  0.079649  0.353229 -0.323458  0.023944 -0.176926  375.01    0.0
75356 -0.342706  0.281718 -0.280851       NaN       NaN     NaN    NaN

[5 rows x 31 columns]
```

```
[ ]: import pandas as pd
     df = pd.read_csv('/content/creditcard.csv')

     #Dropping Empty Cells
     df = df.drop(columns=df.columns[1:3], inplace=True)
     print(df)
```

```
None
```

```
[ ]: import pandas as pd
     df = pd.read_csv('/content/creditcard.csv')

     #Change data to Numpy
     df.to_numpy()
```

```
<ipython-input-14-7a866dd08d87>:2: DtypeWarning: Columns (28) have mixed types.
Specify dtype option on import or set low_memory=False.
  df = pd.read_csv('/content/creditcard.csv')
```

```
[ ]: array([[0, -1.3598071336738, -0.0727811733098497, _,
             -0.0210530534538215, 149.62, 0.0],
            [0, 1.19185711131486, 0.26615071205963, _, 0.0147241691924927,
             2.69, 0.0],
            [1, -1.35835406159823, -1.34016307473609, _,
             -0.0597518405929204, 378.66, 0.0],
            _,
            [81845, 1.06740445339533, 0.0871526496833023, _,
             '0.0222817106909724', 85.9, 0.0],
            [81846, -0.833219444483895, 0.746825586488521, _,
             '0.0384589819489235', 6.8, 0.0],
            [81847, -0.418804707094463, 0.740210337181224, _, '-', nan, nan]],
           dtype=object)
```

Naan Mudhalvan

Data Science - Credit Card Fraud Detection (PHASE 4)

[3]:
```
import pandas as pd
import numpy as np
# read dataset
df = pd.read_csv('/content/creditcard.csv')
print(df)
```

```
        Time        V1        V2        V3        V4        V5        V6  \
0          0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1          0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2          1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3          1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4          2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
...      ...       ...       ...       ...       ...       ...       ...
11954  20631  1.504204 -0.411728  0.200090 -0.778753 -0.442232 -0.119677
11955  20636  1.134994  0.096340  0.277921  0.319692  0.742800  1.611803
11956  20638 -6.305012  3.944886 -4.707362  1.539602 -3.934785 -1.730565
11957  20638  1.161960 -0.398297  1.123732 -0.474237 -1.226667 -0.519325
11958  20642  1.291096 -0.226628  0.708386 -0.719236 -0.659099 -0.273757

             V7        V8        V9  ...       V21       V22       V23  \
0      0.239599  0.098698  0.363787  ... -0.018307  0.277838 -0.110474
1     -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288
2      0.791461  0.247676 -1.514654  ...  0.247998  0.771679  0.909412
3      0.237609  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321
4      0.592941 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458
...         ...       ...       ...  ...       ...       ...       ...
11954 -0.782660 -0.165178  0.691819  ... -0.136231 -0.217274 -0.143260
11955 -0.458649  0.390012  1.424541  ... -0.395605 -0.743542  0.222256
11956 -2.104936  3.843447  0.863458  ...  0.073140 -0.039935 -0.108896
11957 -0.804179  0.070134  3.262926  ... -0.121191  0.097255  0.050903
11958 -0.612042 -0.111488  3.032258  ...       NaN       NaN       NaN

            V24       V25       V26       V27       V28  Amount  Class
0      0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62    0.0
1     -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69    0.0
```

```
2      -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66    0.0
3      -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50    0.0
4       0.141267 -0.206010  0.502292  0.219422  0.215153   69.99    0.0
...          ...       ...       ...       ...       ...     ...    ...
11954 -1.057332  0.529188 -0.235062 -0.012089  0.000905    9.00    0.0
11955 -1.859104 -0.109777  0.279049  0.012398 -0.009090    0.99    0.0
11956  0.691434 -0.261979 -0.447540  0.212900 -0.031021   89.99    0.0
11957  0.330479  0.315692 -0.712765  0.073836  0.028055   11.85    0.0
11958       NaN       NaN       NaN       NaN       NaN     NaN    NaN

[11959 rows x 31 columns]
```

[4]:
```python
df['Hour'] = df['Time'] // 3600  # Convert seconds to hours
df['Day'] = df['Time'] // (3600 * 24)  # Convert seconds to days
df['Weekday'] = pd.to_datetime(df['Time']).dt.dayofweek  # Extract weekday (0 =
↪Monday, 6 = Sunday)

#Amount-Based Features
bins = [0, 50, 100, 500, 1000, np.inf]
labels = ['Very Low', 'Low', 'Medium', 'High', 'Very High']
df['Amount_Category'] = pd.cut(df['Amount'], bins=bins, labels=labels)

# Transaction Frequency
df['Transactions_Last_Hour'] = df.groupby('Hour')['Hour'].transform('count')

# Statistical Aggregations
for i in range(1, 29):
    df[f'V{i}_Mean'] = df.groupby('Class')[f'V{i}'].transform('mean')
    df[f'V{i}_StdDev'] = df.groupby('Class')[f'V{i}'].transform('std')

# Interaction Features
df['V1_V2_Multiplication'] = df['V1'] * df['V2']

X = df.drop('Class', axis=1)  # Features
y = df['Class']  # Target variable

from sklearn.impute import SimpleImputer
# Create an imputer object
imputer = SimpleImputer(strategy='mean')

# Select only numeric columns
X_numeric = X.select_dtypes(include=[np.number])

# Impute missing values in features
X_imputed = imputer.fit_transform(X_numeric)
```

[8]:
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.
↪2, random_state=42)

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

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

print(y.isnull().sum())
# Remove rows with NaN values in the target variable
y = y.dropna()
X_imputed = X_imputed[y.index]  # Update X_imputed to match the updated y

from sklearn.impute import SimpleImputer

# Create an imputer object for target variable y
y_imputer = SimpleImputer(strategy='mean')
y_imputed = y_imputer.fit_transform(y.values.reshape(-1, 1))

# Replace y with imputed values
y = pd.Series(y_imputed.ravel(), index=y.index)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(classification_rep)
```

```
0
Accuracy: 100.00%
Confusion Matrix:
[[2378    0]
```

```
 [    0    14]]
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2378
         1.0       1.00      1.00      1.00        14

    accuracy                           1.00      2392
   macro avg       1.00      1.00      1.00      2392
weighted avg       1.00      1.00      1.00      2392
```

[9]:

```
from sklearn.metrics import accuracy_score
#Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Accuracy: 100.00%
```

[10]:

```
from sklearn.metrics import precision_score, recall_score, f1_score
#Precision, Recall, and F1-Score
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1))
```

```
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

[11]:

```
from sklearn.metrics import confusion_matrix
#Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Confusion Matrix:
[[2378    0]
 [   0   14]]
```

[12]:

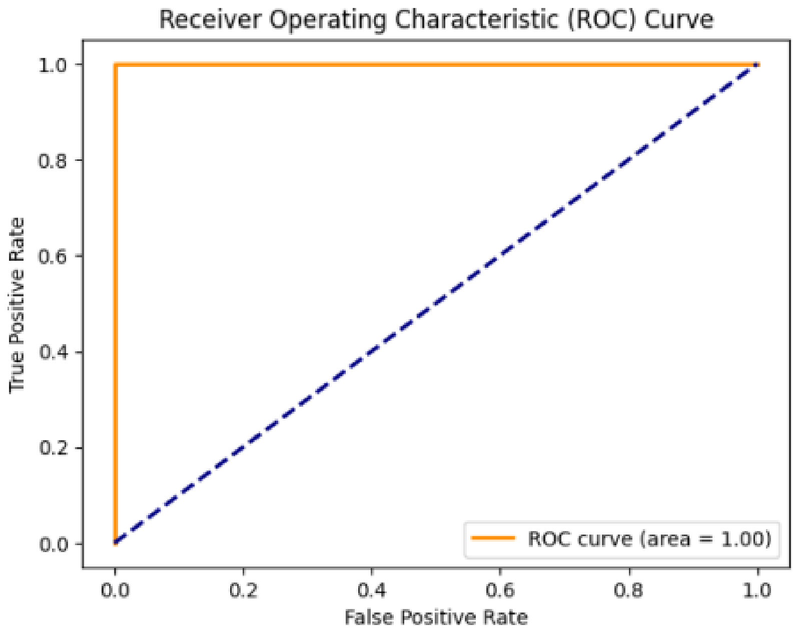
```
from sklearn.metrics import roc_auc_score, roc_curve
import matplotlib.pyplot as plt

#Receiver Operating Characteristic (ROC) Curve and Area Under the Curve
 ↪(AUC-ROC):
```

```python
# Calculate ROC-AUC score
roc_auc = roc_auc_score(y_test, rf_classifier.predict_proba(X_test)[:, 1])

# Plot ROC curve
fpr, tpr, _ = roc_curve(y_test, rf_classifier.predict_proba(X_test)[:, 1])
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.
 ↪format(roc_auc))
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

print("ROC-AUC Score: {:.2f}".format(roc_auc))
```




```
ROC-AUC Score: 1.00
```

[15]:

```python
from sklearn.model_selection import cross_val_score

# Cross-Validation
# Perform 10-fold cross-validation

cv_scores = cross_val_score(rf_classifier, X_imputed, y, cv=10,␣
 ↪scoring='accuracy')
print("Cross-Validation Scores:", cv_scores)
print("Mean Accuracy: {:.2f}".format(cv_scores.mean()))
```

```
Cross-Validation Scores: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Mean Accuracy: 1.00
```

Understanding and Impact:

1. **Enhanced Fraud Detection:** By deploying a data-driven credit card fraud detection system, financial institutions can significantly improve their ability to detect and prevent fraudulent transactions, reducing financial losses.

2. **Customer Trust:** Increased accuracy in fraud detection leads to enhanced trust among cardholders, as they experience fewer disruptions and can rely on their financial institutions to protect their assets.

3. **Operational Efficiency:** Real-time processing and automation of fraud detection processes streamline operations, saving time and resources for financial organizations.

4. **Compliance:** The system ensures compliance with regulatory requirements, safeguarding financial institutions from legal repercussions and reputational damage.

5. **Adaptability:** The ability to continuously adapt to evolving fraud patterns ensures that the system remains effective in detecting new and emerging forms of credit card fraud, providing long-term security.

Description of Dataset:

1. Transaction Information:
- ❖ Transaction Date and Time: Timestamps indicating when each transaction occurred.
- ❖ Transaction Amount: The monetary value of the transaction.
- ❖ Transaction Type: Categorization of the transaction, e.g., purchase, withdrawal, online payment, etc.

2. Cardholder Information:
- ❖ Cardholder Name: The name of the cardholder (often anonymized).
- ❖ Card Number: The unique identifier for the credit card (typically anonymized or tokenized).
- ❖ Card Expiration Date: The date when the credit card expires (often anonymized).

3. Merchant Information:
- ❖ Merchant Name: The name of the establishment where the transaction took place.
- ❖ Merchant Category: The type of business or industry the merchant belongs to.

4. Transaction Status:
- ❖ Fraud Label: A binary indicator (0 or 1) specifying whether the transaction is fraudulent (1) or legitimate (0). This is the ground truth used for model training and evaluation.

5. Additional Features:
- ❖ Location: Information about the geographical location of the transaction, such as city or country.
- ❖ Device Information: Data about the device used for the transaction, which may include the device type, operating system, or device identifier.
- ❖ IP Address: The internet protocol (IP) address associated with the transaction.

Conclusion:

In summary, credit card fraud detection is an essential safeguard against financial fraud. Through data analysis and machine learning, it offers real-time protection while minimizing disruptions to legitimate transactions. Continuous monitoring and adaptation are pivotal to address evolving fraud patterns. Key findings drive system improvements, ensuring a balance between security and convenience. Ultimately, credit card fraud detection remains a cornerstone of trust in the digital financial world.

References:

1. Dataset Link:

https://drive.google.com/drive/folders/1eM-GTnt0p1v_yM2_RNobhYX9pC-wC6in?usp=drive_link

2. Kaggle Link:

https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

3. Github Links:

1. https://github.com/shalinisubburaj11/NM-Data-science.git
2. https://github.com/POOJA20SHREE/NM-Datascience.git
3. https://github.com/prinitha0310/NM
4. https://github.com/prateekshitha/NM-Data-science.git
5. https://github.com/Haritha3228/NM-data-science-.git