VCC Project Code Documentation

Harnessing Cloud Computing for Fraud Detection in Online Payments

Team Members:

1. Shubhankit Dutt (G23AI2070)
2. Nihar Dave (G23AI2097)
3. Priyank Bhardwaj (G23AI2038)

Under the Supervision of

Prof. Sumit Kalra

PG DIPLOMA in DATA ENGINEERING

DEPARTMENT OF AIDE

INDIAN INSTITUTE OF TECHNOLOGY JODHPUR

SEP, 2024

Project Description:

Harnessing Cloud Computing for Fraud Detection in Online Payment using Google Cloud Platform.

Problem Statement:

The rise of online payment fraud is a growing concern in the rapidly evolving digital transaction landscape. It presents significant risks to both financial institutions and consumers, resulting in monetary losses and diminishing trust in online payment systems. The goal of this project is to develop an advanced machine learning solution capable of detecting fraudulent transactions in real time, helping safeguard the integrity of online payment systems.

Technologies:

- Cloud Platform: Google Cloud Platform (GCP)
- Machine Learning Framework: Random Forest Classifier
- Libraries: NumPy, Pandas, Matplotlib, Seaborn
- Programming language: Python
- Application: Flask
- UI: Html
- Deployment Tools: GCP services, Google App Engine

Code Overview

1. Preprocessing and Exploratory Data Analysis (EDA)

Steps:

- Import Libraries: Core libraries like numpy, pandas, matplotlib, and seaborn were imported to handle data manipulation and visualization.
- Handling Missing Data: The dataset is cleansed by dropping null values, ensuring data quality and reliability.
- Duplicate Removal: Duplicate rows and columns are identified and removed to avoid data redundancy.
- Range Checks: Minimum and maximum values of key features are inspected for potential outliers or anomalies.

2. Data Visualization

Univariate Data Visualization:

Univariate visualizations provide insight into the distribution of individual features. Techniques like histograms and box plots are used to highlight the central tendency, spread, and presence of outliers.

Bivariate Data Visualization:

Bivariate visualizations reveal relationships between two variables, essential for detecting interactions between transaction amounts and other factors like location or time. Techniques like scatter plots and heatmaps help uncover these relationships.

Multivariate Data Visualization:

Advanced visualizations are employed to understand relationships across multiple dimensions. Scatter plot matrices and heatmaps reveal intricate patterns and interactions between features.

3. Random Forest Model for Fraud Detection

The Random Forest model is a powerful ensemble-based algorithm used to detect fraudulent transactions. By training multiple decision trees on random subsets of data, the model achieves high accuracy and robustness.

Steps:

1. Data Preparation: Features are selected, and the data is split into training and test sets.
2. Model Training: The Random Forest classifier is trained on the data, using bootstrapped samples and random feature selection to reduce overfitting.
3. Model Evaluation: Metrics such as accuracy, precision, recall, and F1-score are computed to evaluate the model's effectiveness.
4. Feature Importance Analysis: Insights into key variables are drawn by analyzing feature importance scores from the Random Forest model.

Deployment on GCP

The model is deployed on Google Cloud Platform (GCP), leveraging cloud-based scalability for handling large transaction datasets and performing real-time fraud detection. Services such as Google AI Platform, Cloud Storage, and BigQuery are utilized to host the model and store transaction data.

Key Components:

Google Cloud SDK: Google Cloud SDK is a set of command-line tools and libraries that allow developers to interact with Google Cloud Platform (GCP) services and manage resources. It provides a comprehensive toolkit for building, testing, and deploying applications on GCP. Google Cloud SDK is commonly used to deploy applications from your local directory to Google Cloud Platform.

Google App Engine: Google App Engine is a fully managed platform that allows developers to build, deploy, and scale web applications without having to worry about managing infrastructure.
It provides a scalable environment for running applications in a variety of programming languages.

Google Cloud Storage: It is a massively scalable object storage service offered by Google Cloud Platform. It allows to store and retrieve data of any size from anywhere on the web. It is a cloud-based file storage system that can handle enormous amounts of data, making it ideal for storing images, videos, documents, and other digital assets.

Advantages of Google Cloud App Engine over traditional methods of hosting web applications:

Fully Managed: App Engine handles infrastructure management tasks like server provisioning, load balancing, and scaling, allowing developers to focus on writing code.

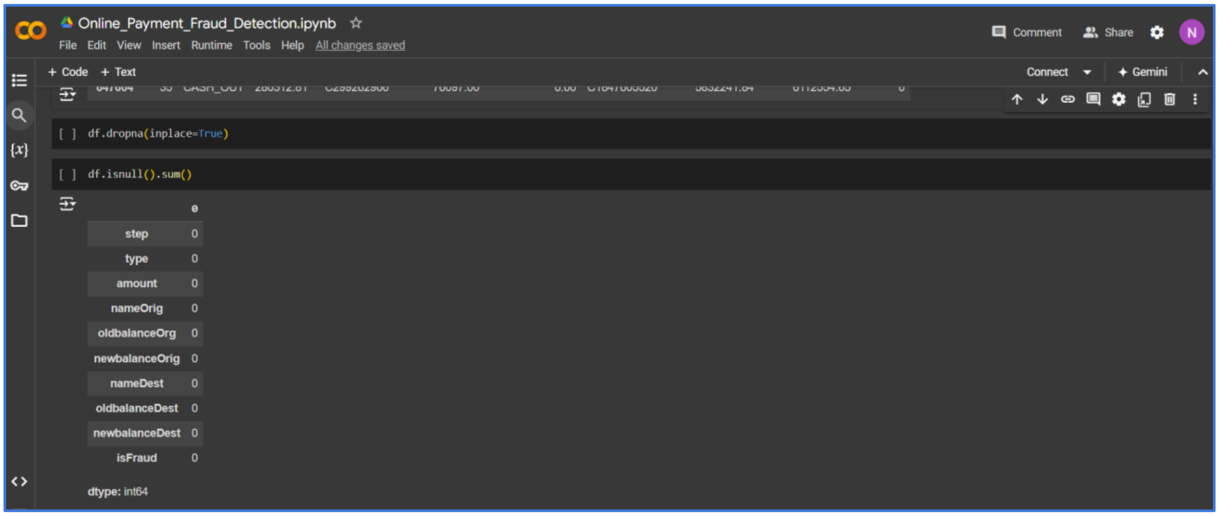
Scalability: App Engine automatically scales resources based on demand, ensuring optimal performance and cost-efficiency.

Reliability: Google's infrastructure provides high availability and reliability, minimizing downtime and ensuring consistent performance.

Cost-Effective: App Engine's pay-as-you-go pricing model makes it cost-effective for various application sizes.

Integration: It seamlessly integrates with other Google Cloud Platform services, offering a comprehensive solution for building and deploying applications.

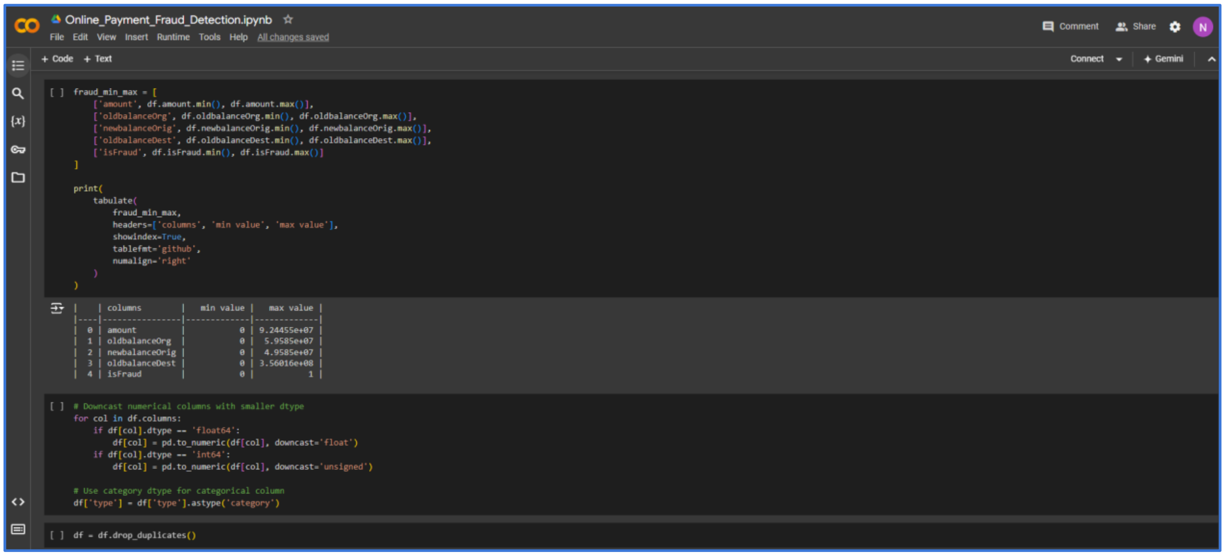
Code snippets and screenshots


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
N
+ Code + Text
Connect
Gemini

```
[ ] df.dropna(inplace=True)
```

```
[ ] df.isnull().sum()
```

```
                  0
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0

dtype: int64
```

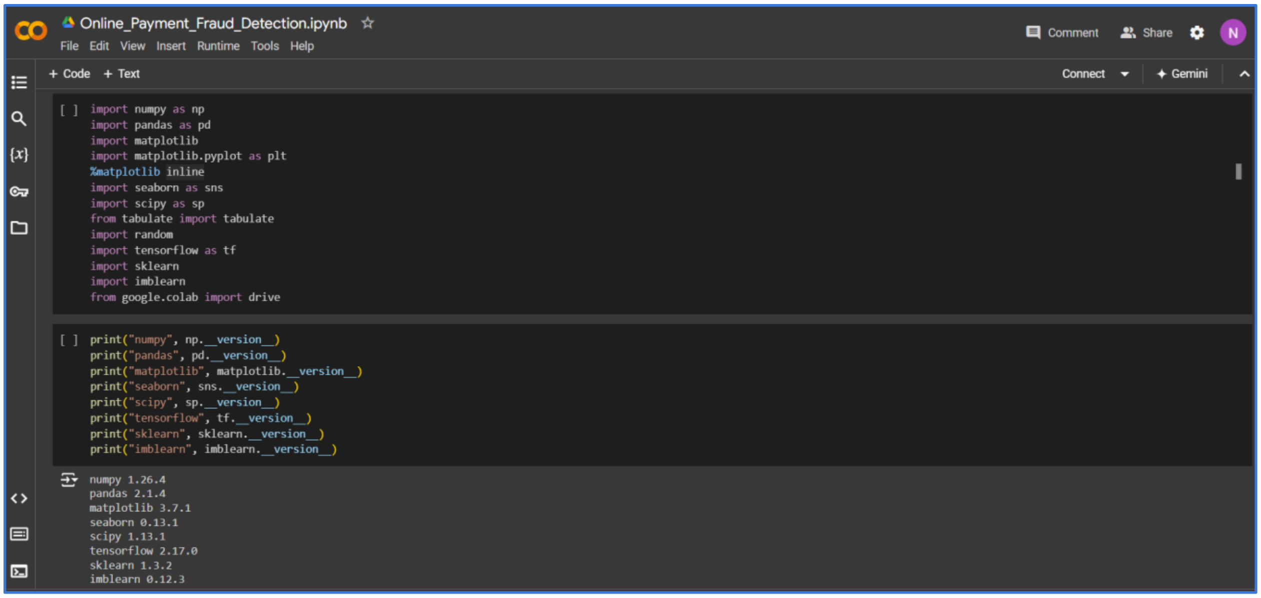

Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
N
+ Code + Text
Connect
Gemini

```
[ ] fraud_min_max = [
        ['amount', df.amount.min(), df.amount.max()],
        ['oldbalanceOrg', df.oldbalanceOrg.min(), df.oldbalanceOrg.max()],
        ['newbalanceOrig', df.newbalanceOrig.min(), df.newbalanceOrig.max()],
        ['oldbalanceDest', df.oldbalanceDest.min(), df.oldbalanceDest.max()],
        ['isFraud', df.isFraud.min(), df.isFraud.max()]
    ]

    print(
        tabulate(
            fraud_min_max,
            headers=['columns', 'min value', 'max value'],
            showindex=True,
            tablefmt='github',
            numalign='right'
        )
    )
```

```
|    | columns        |   min value |   max value |
|----|----------------|-------------|-------------|
|  0 | amount         |           0 | 9.24455e+07 |
|  1 | oldbalanceOrg  |           0 |  5.9585e+07 |
|  2 | newbalanceOrig |           0 |  4.9585e+07 |
|  3 | oldbalanceDest |           0 | 3.56016e+08 |
|  4 | isFraud        |           0 |           1 |
```

```
[ ] # Downcast numerical columns with smaller dtype
    for col in df.columns:
        if df[col].dtype == 'float64':
            df[col] = pd.to_numeric(df[col], downcast='float')
        if df[col].dtype == 'int64':
            df[col] = pd.to_numeric(df[col], downcast='unsigned')

    # Use category dtype for categorical column
    df['type'] = df['type'].astype('category')
```

```
[ ] df = df.drop_duplicates()
```

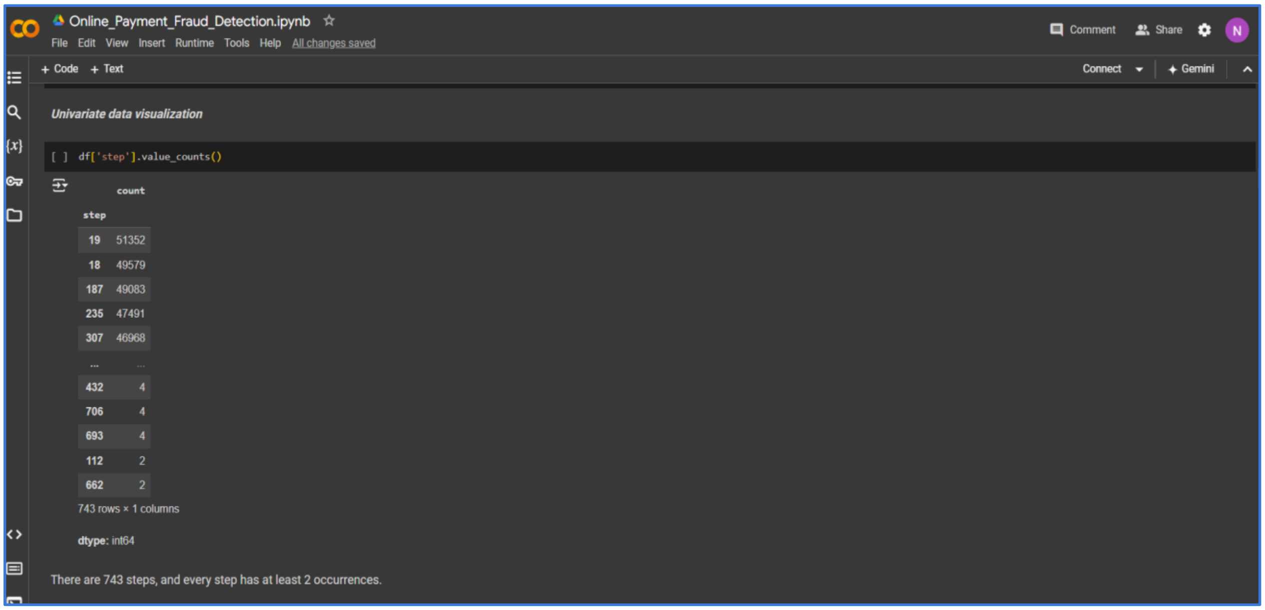

Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
N
+ Code
+ Text
Connect
Gemini

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import scipy as sp
from tabulate import tabulate
import random
import tensorflow as tf
import sklearn
import imblearn
from google.colab import drive
```

```
print("numpy", np.__version__)
print("pandas", pd.__version__)
print("matplotlib", matplotlib.__version__)
print("seaborn", sns.__version__)
print("scipy", sp.__version__)
print("tensorflow", tf.__version__)
print("sklearn", sklearn.__version__)
print("imblearn", imblearn.__version__)
```

```
numpy 1.26.4
pandas 2.1.4
matplotlib 3.7.1
seaborn 0.13.1
scipy 1.13.1
tensorflow 2.17.0
sklearn 1.3.2
imblearn 0.12.3
```

Univariate Data Visualization


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
N
+ Code
+ Text
Connect
Gemini

Univariate data visualization

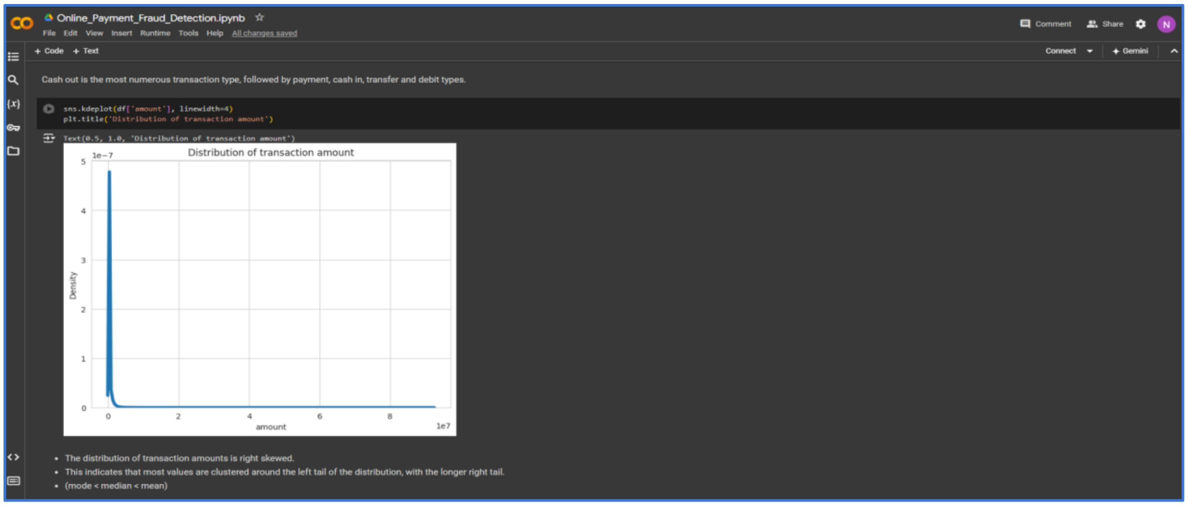
```
df['step'].value_counts()
```

	count
step	
19	51352
18	49579
187	49083
235	47491
307	46968
...	...
432	4
706	4
693	4
112	2
662	2

743 rows × 1 columns

dtype: int64

There are 743 steps, and every step has at least 2 occurrences.

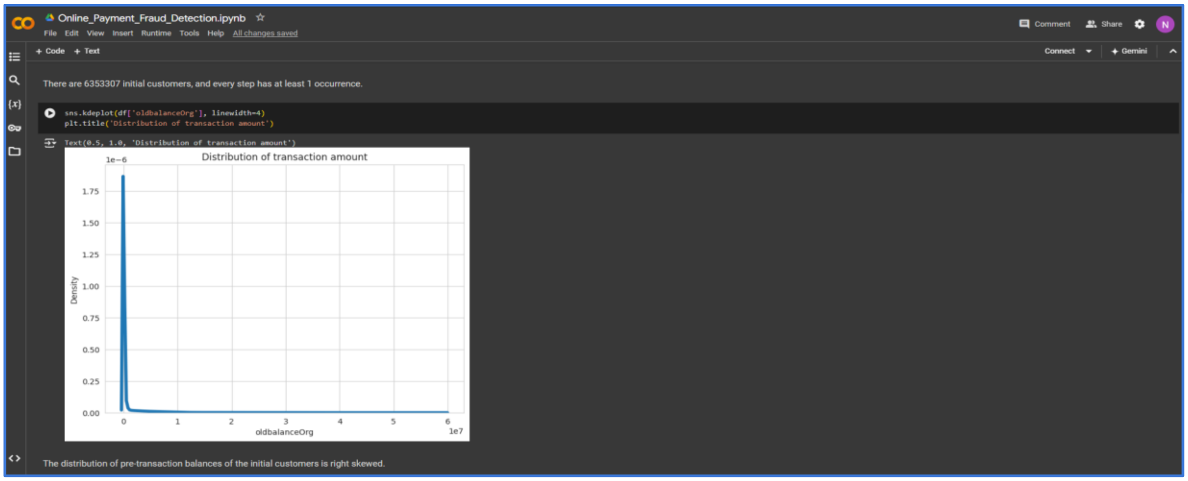

Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini
Distribution of transaction amount
1e−7
5
4
3
2
1
0
Density
0
2
4
6
8
1e7
amount

Cash out is the most numerous transaction type, followed by payment, cash in, transfer and debit types.

```
sns.kdeplot(df['amount'], linewidth=4)
plt.title('Distribution of transaction amount')
```

```
Text(0.5, 1.0, 'Distribution of transaction amount')
```

- The distribution of transaction amounts is right skewed.
- This indicates that most values are clustered around the left tail of the distribution, with the longer right tail.
- (mode < median < mean)

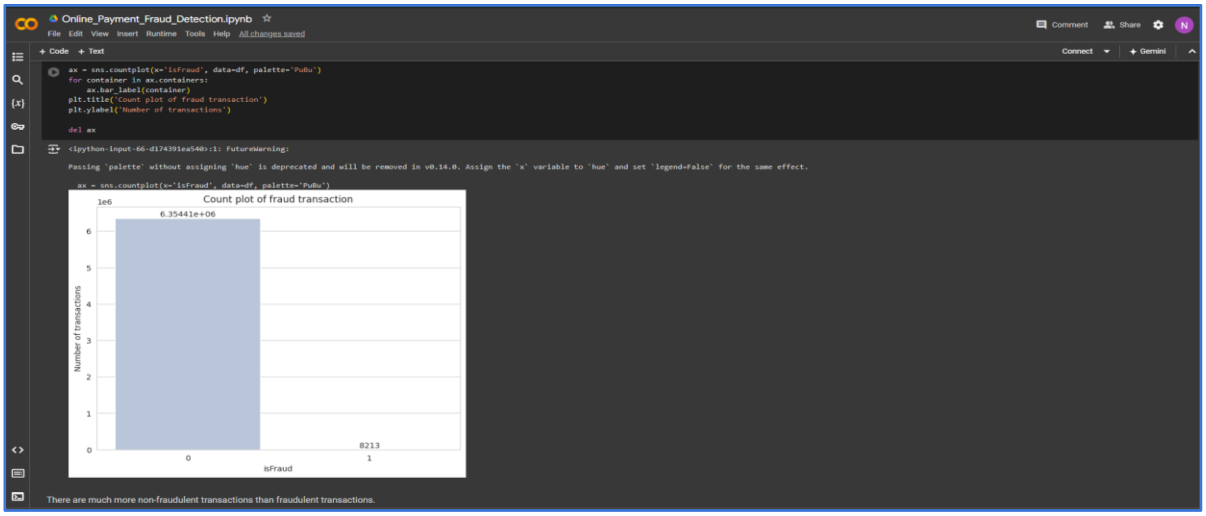

Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini
Distribution of transaction amount
1e−6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
0
1
2
3
4
5
6
1e7
oldbalanceOrg

There are 6353307 initial customers, and every step has at least 1 occurrence.

```
sns.kdeplot(df['oldbalanceOrg'], linewidth=4)
plt.title('Distribution of transaction amount')
```

```
Text(0.5, 1.0, 'Distribution of transaction amount')
```

The distribution of pre-transaction balances of the initial customers is right skewed.

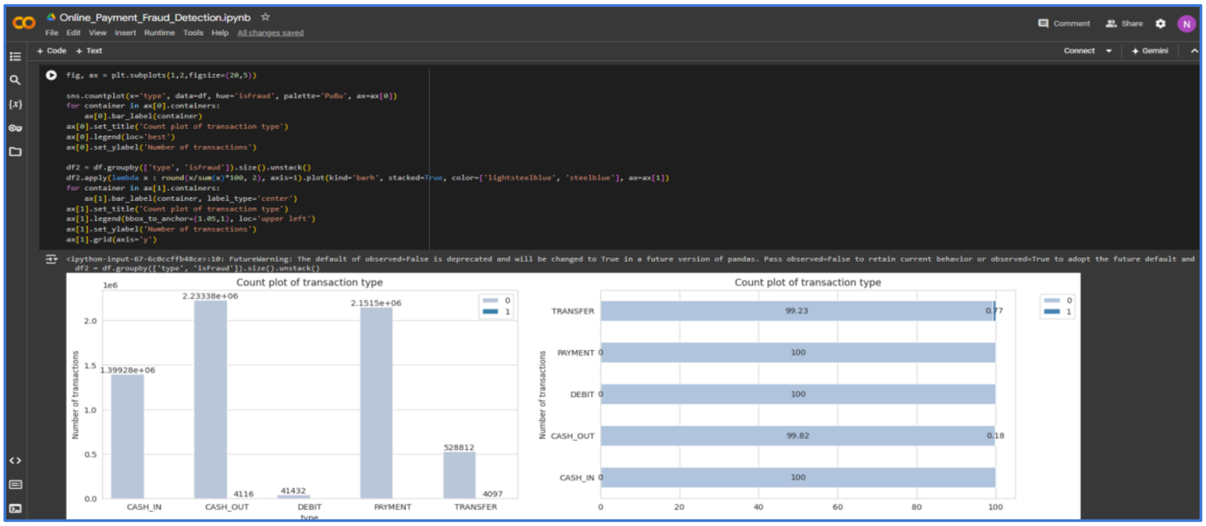

Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini
Count plot of fraud transaction
1e6
6.35441e+06
8213
6
5
4
3
2
1
0
Number of transactions
0
1
isFraud

```
ax = sns.countplot(x='isFraud', data=df, palette='PuBu')
for container in ax.containers:
    ax.bar_label(container)
plt.title('Count plot of fraud transaction')
plt.ylabel('Number of transactions')

del ax
```

```
<ipython-input-66-d174391ea540>:1: FutureWarning: 

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  ax = sns.countplot(x='isFraud', data=df, palette='PuBu')
```

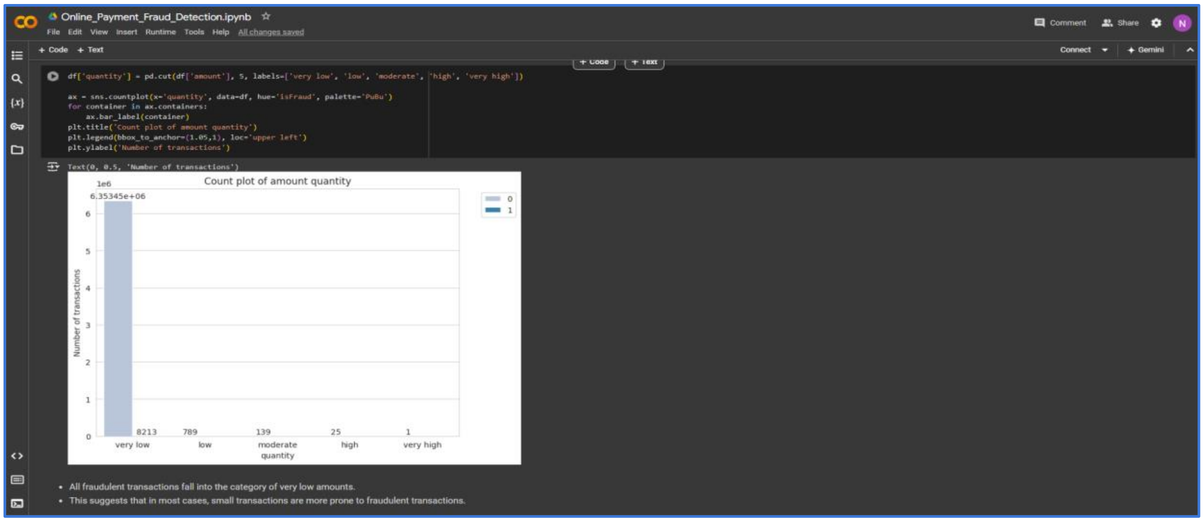
There are much more non-fraudulent transactions than fraudulent transactions.

Bivariate Data Visualization


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini

```
fig, ax = plt.subplots(1,2,figsize=(20,5))

sns.countplot(x='type', data=df, hue='isFraud', palette='PuBu', ax=ax[0])
for container in ax[0].containers:
    ax[0].bar_label(container)
ax[0].set_title('Count plot of transaction type')
ax[0].legend(loc='best')
ax[0].set_ylabel('Number of transactions')

df2 = df.groupby(['type', 'isFraud']).size().unstack()
df2.apply(lambda x : round(x/sum(x)*100, 2), axis=1).plot(kind='barh', stacked=True, color=['lightsteelblue', 'steelblue'], ax=ax[1])
for container in ax[1].containers:
    ax[1].bar_label(container, label_type='center')
ax[1].set_title('Count plot of transaction type')
ax[1].legend(bbox_to_anchor=(1.05,1), loc='upper left')
ax[1].set_ylabel('Number of transactions')
ax[1].grid(axis='y')
```

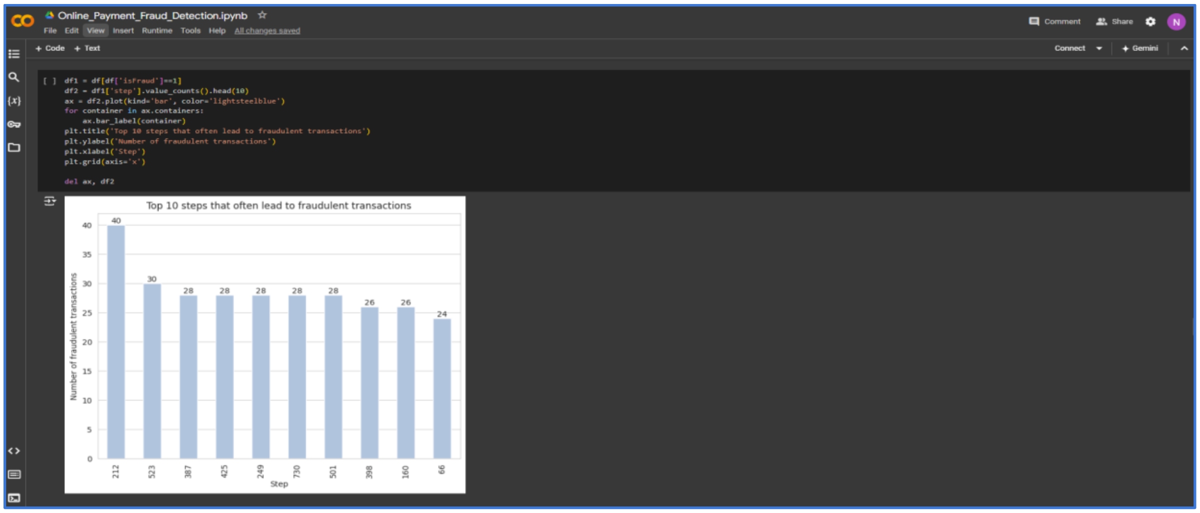
```
<ipython-input-67-6c0ccffb48ce>:10: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and
  df2 = df.groupby(['type', 'isFraud']).size().unstack()
```


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini

```
df['quantity'] = pd.cut(df['amount'], 5, labels=['very low', 'low', 'moderate', 'high', 'very high'])

ax = sns.countplot(x='quantity', data=df, hue='isFraud', palette='PuBu')
for container in ax.containers:
    ax.bar_label(container)
plt.title('Count plot of amount quantity')
plt.legend(bbox_to_anchor=(1.05,1), loc='upper left')
plt.ylabel('Number of transactions')
```

```
Text(0, 0.5, 'Number of transactions')
```

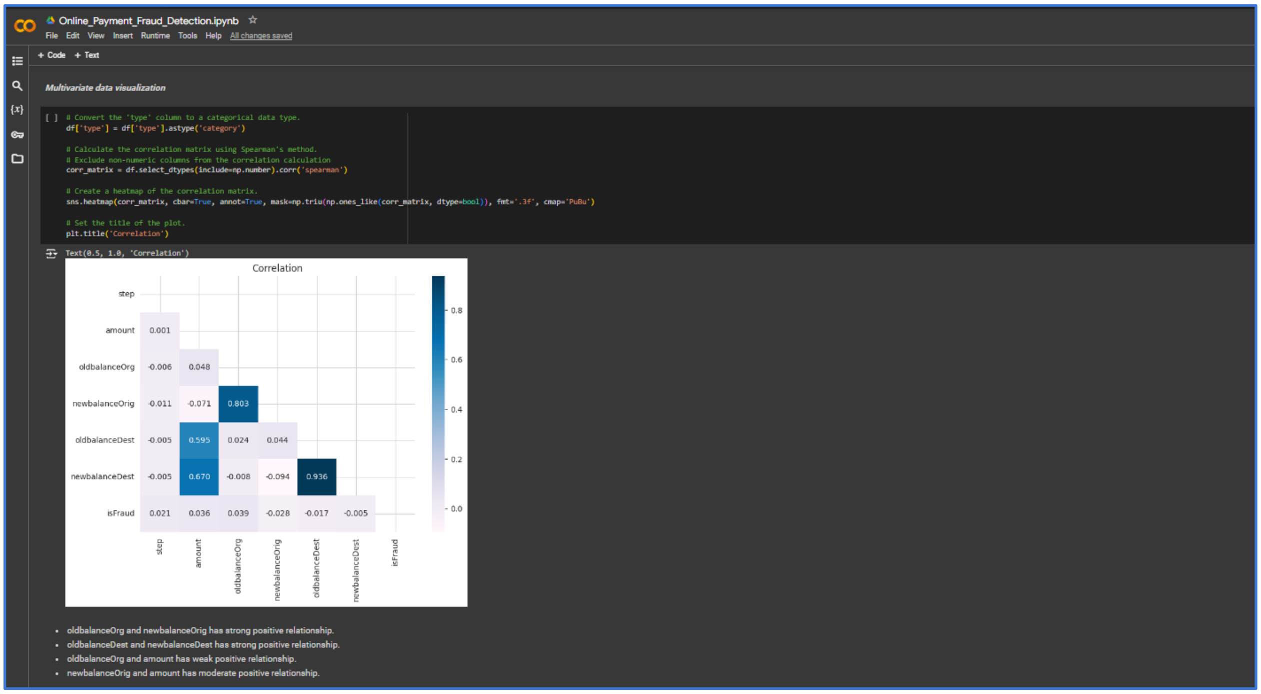
- All fraudulent transactions fall into the category of very low amounts.
- This suggests that in most cases, small transactions are more prone to fraudulent transactions.


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini

```
[ ] df1 = df[df['isFraud']==1]
    df2 = df1['step'].value_counts().head(10)
    ax = df2.plot(kind='bar', color='lightsteelblue')
    for container in ax.containers:
        ax.bar_label(container)
    plt.title('Top 10 steps that often lead to fraudulent transactions')
    plt.ylabel('Number of fraudulent transactions')
    plt.xlabel('Step')
    plt.grid(axis='x')

    del ax, df2
```

Top 10 steps that often lead to fraudulent transactions
40
35
30
25
20
15
10
5
0
Number of fraudulent transactions
40
30
28
28
28
28
28
26
26
24
212
523
387
425
249
730
501
398
160
66
Step

Multivariate Data Visualization


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

Multivariate data visualization

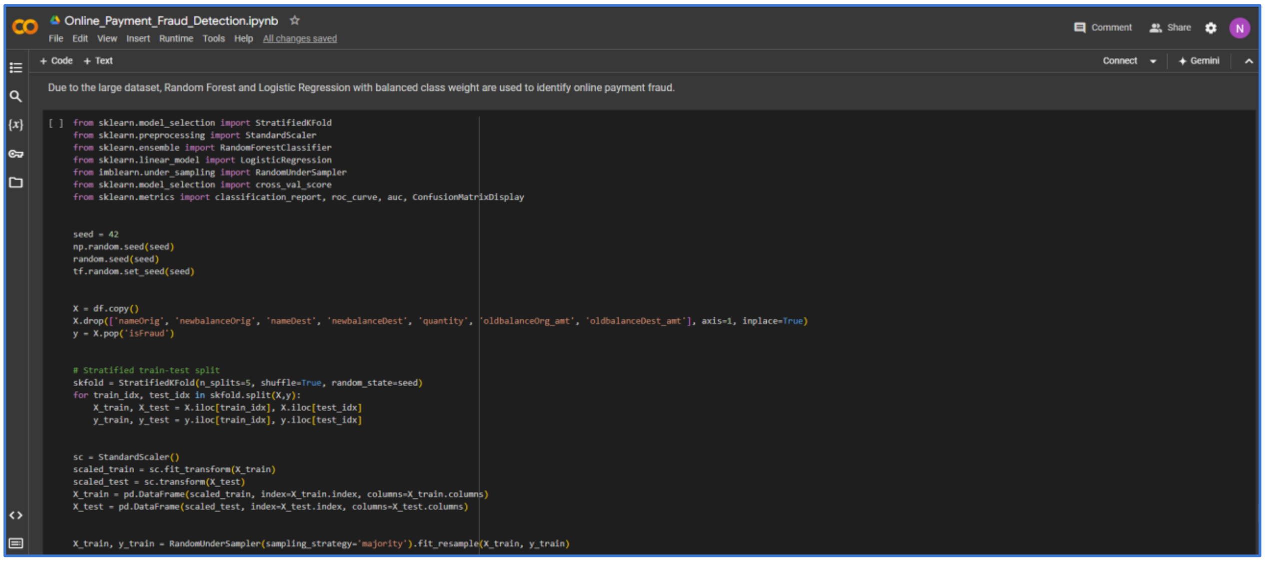
```
# Convert the 'type' column to a categorical data type.
df['type'] = df['type'].astype('category')

# Calculate the correlation matrix using Spearman's method.
# Exclude non-numeric columns from the correlation calculation
corr_matrix = df.select_dtypes(include=np.number).corr('spearman')

# Create a heatmap of the correlation matrix.
sns.heatmap(corr_matrix, cbar=True, annot=True, mask=np.triu(np.ones_like(corr_matrix, dtype=bool)), fmt='.3f', cmap='PuBu')

# Set the title of the plot.
plt.title('Correlation')
```

```
Text(0.5, 1.0, 'Correlation')
```

Correlation
step
amount 0.001
oldbalanceOrg -0.006 0.048
newbalanceOrig -0.011 -0.071 0.803
oldbalanceDest -0.005 0.595 0.024 0.044
newbalanceDest -0.005 0.670 -0.008 -0.094 0.936
isFraud 0.021 0.036 0.039 -0.028 -0.017 -0.005
step
amount
oldbalanceOrg
newbalanceOrig
oldbalanceDest
newbalanceDest
isFraud
0.8
0.6
0.4
0.2
0.0

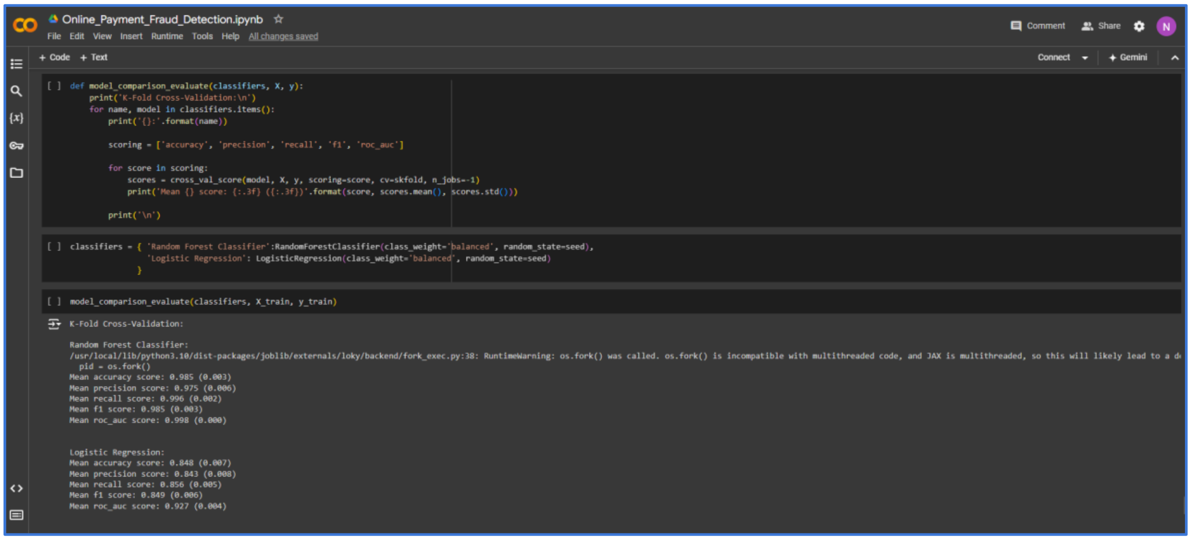
- oldbalanceOrg and newbalanceOrig has strong positive relationship.
- oldbalanceDest and newbalanceDest has strong positive relationship.
- oldbalanceOrg and amount has weak positive relationship.
- newbalanceOrig and amount has moderate positive relationship.

Random Forest Model


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
N
+ Code + Text
Connect
Gemini

Due to the large dataset, Random Forest and Logistic Regression with balanced class weight are used to identify online payment fraud.

```
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, roc_curve, auc, ConfusionMatrixDisplay


seed = 42
np.random.seed(seed)
random.seed(seed)
tf.random.set_seed(seed)


X = df.copy()
X.drop(['nameOrig', 'newbalanceOrig', 'nameDest', 'newbalanceDest', 'quantity', 'oldbalanceOrg_amt', 'oldbalanceDest_amt'], axis=1, inplace=True)
y = X.pop('isFraud')


# Stratified train-test split
skfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
for train_idx, test_idx in skfold.split(X,y):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]


sc = StandardScaler()
scaled_train = sc.fit_transform(X_train)
scaled_test = sc.transform(X_test)
X_train = pd.DataFrame(scaled_train, index=X_train.index, columns=X_train.columns)
X_test = pd.DataFrame(scaled_test, index=X_test.index, columns=X_test.columns)


X_train, y_train = RandomUnderSampler(sampling_strategy='majority').fit_resample(X_train, y_train)
```

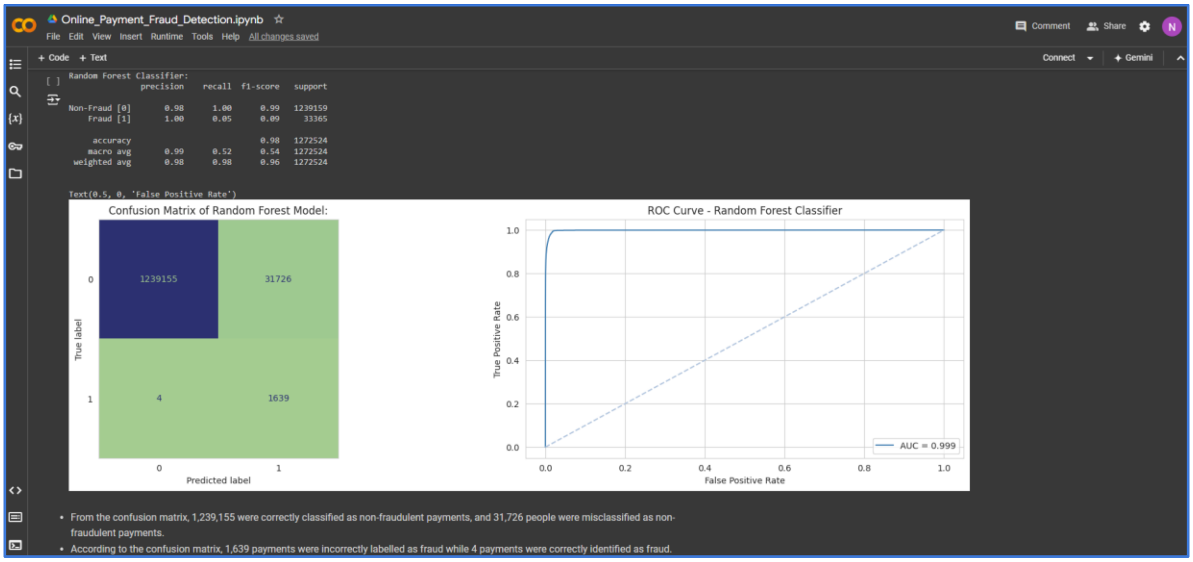

Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini

```
def model_comparison_evaluate(classifiers, X, y):
    print('K-Fold Cross-Validation:\n')
    for name, model in classifiers.items():
        print('{}:'.format(name))

        scoring = ['accuracy', 'precision', 'recall', 'f1', 'roc_auc']

        for score in scoring:
            scores = cross_val_score(model, X, y, scoring=score, cv=skfold, n_jobs=-1)
            print('Mean {} score: {:.3f} ({:.3f})'.format(score, scores.mean(), scores.std()))

        print('\n')
```

```
classifiers = { 'Random Forest Classifier':RandomForestClassifier(class_weight='balanced', random_state=seed),
                'Logistic Regression': LogisticRegression(class_weight='balanced', random_state=seed)
              }
```

```
model_comparison_evaluate(classifiers, X_train, y_train)
```

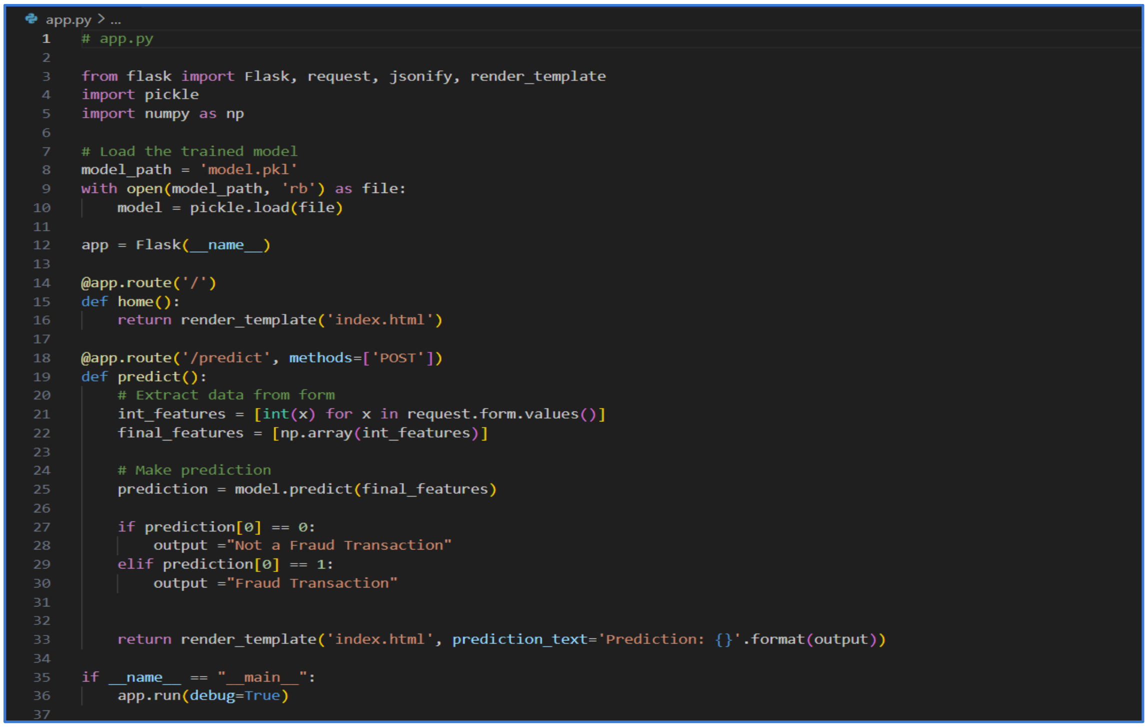
```
K-Fold Cross-Validation:

Random Forest Classifier:
/usr/local/lib/python3.10/dist-packages/joblib/externals/loky/backend/fork_exec.py:38: RuntimeWarning: os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so this will likely lead to a de
  pid = os.fork()
Mean accuracy score: 0.985 (0.003)
Mean precision score: 0.975 (0.006)
Mean recall score: 0.996 (0.002)
Mean f1 score: 0.985 (0.003)
Mean roc_auc score: 0.998 (0.000)


Logistic Regression:
Mean accuracy score: 0.848 (0.007)
Mean precision score: 0.843 (0.008)
Mean recall score: 0.856 (0.005)
Mean f1 score: 0.849 (0.006)
Mean roc_auc score: 0.927 (0.004)
```


Online_Payment_Fraud_Detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Connect
Gemini
Confusion Matrix of Random Forest Model:
0
1
1239155
31726
4
1639
True label
Predicted label
0
1
ROC Curve - Random Forest Classifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.999

```
Random Forest Classifier:
                precision    recall  f1-score   support

Non-Fraud [0]       0.98      1.00      0.99   1239159
    Fraud [1]       1.00      0.05      0.09     33365

     accuracy                           0.98   1272524
    macro avg       0.99      0.52      0.54   1272524
 weighted avg       0.98      0.98      0.96   1272524


Text(0.5, 0, 'False Positive Rate')
```

- From the confusion matrix, 1,239,155 were correctly classified as non-fraudulent payments, and 31,726 people were misclassified as non-fraudulent payments.
- According to the confusion matrix, 1,639 payments were incorrectly labelled as fraud while 4 payments were correctly identified as fraud.

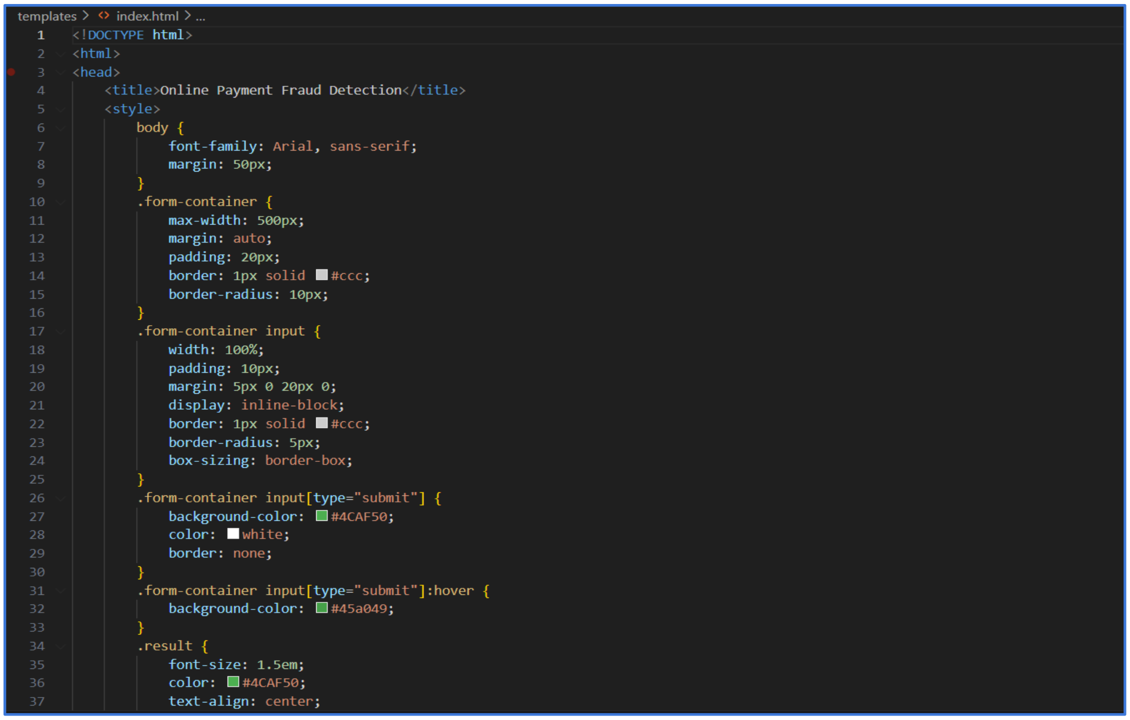
Flask Application Code



app.py > ...

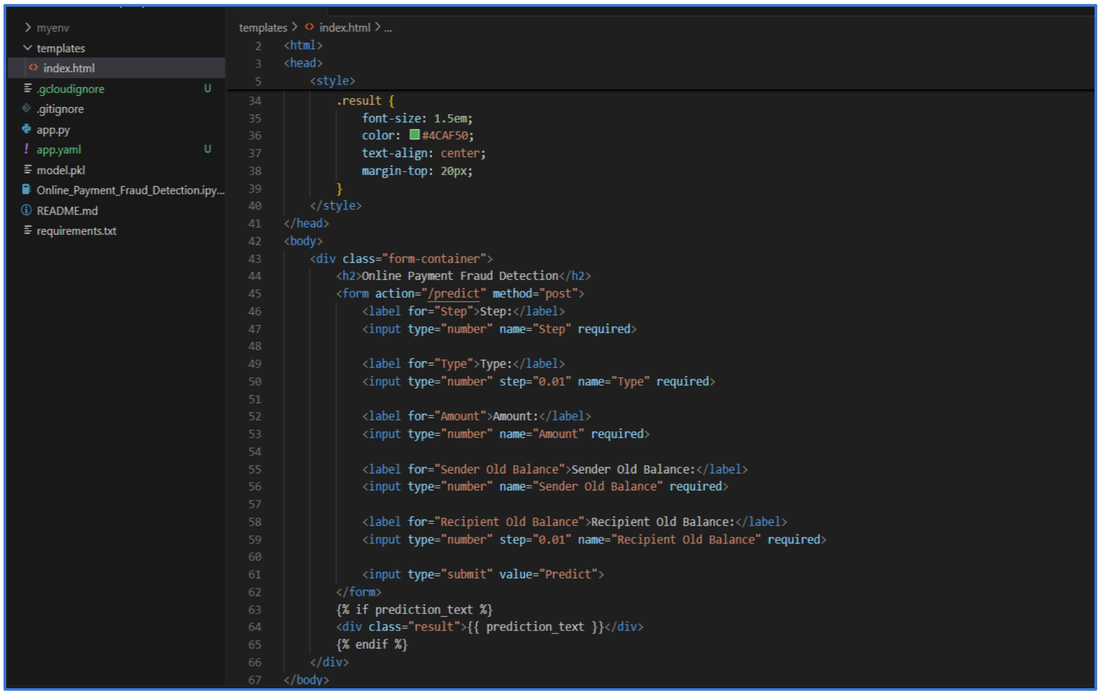
```python
# app.py

from flask import Flask, request, jsonify, render_template
import pickle
import numpy as np

# Load the trained model
model_path = 'model.pkl'
with open(model_path, 'rb') as file:
    model = pickle.load(file)

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    # Extract data from form
    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]

    # Make prediction
    prediction = model.predict(final_features)

    if prediction[0] == 0:
        output ="Not a Fraud Transaction"
    elif prediction[0] == 1:
        output ="Fraud Transaction"


    return render_template('index.html', prediction_text='Prediction: {}'.format(output))

if __name__ == "__main__":
    app.run(debug=True)

```

HTML Script for UI



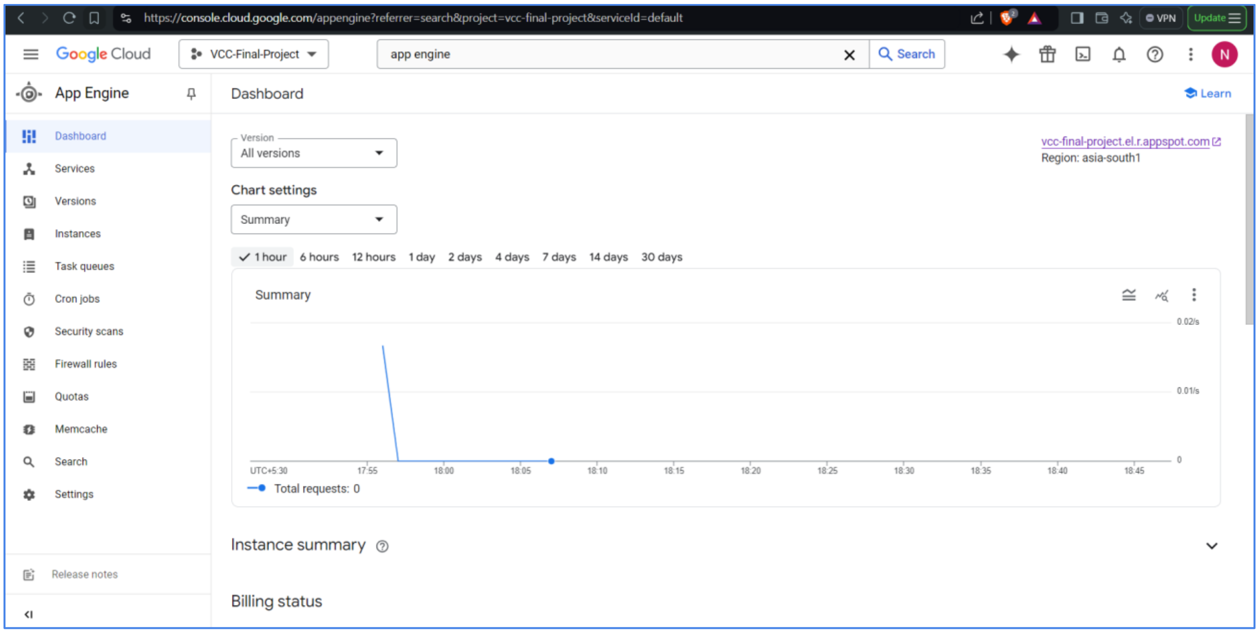
templates > index.html > ...

```html
<!DOCTYPE html>
<html>
<head>
    <title>Online Payment Fraud Detection</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 50px;
        }
        .form-container {
            max-width: 500px;
            margin: auto;
            padding: 20px;
            border: 1px solid □#ccc;
            border-radius: 10px;
        }
        .form-container input {
            width: 100%;
            padding: 10px;
            margin: 5px 0 20px 0;
            display: inline-block;
            border: 1px solid □#ccc;
            border-radius: 5px;
            box-sizing: border-box;
        }
        .form-container input[type="submit"] {
            background-color: 🟩#4CAF50;
            color: □white;
            border: none;
        }
        .form-container input[type="submit"]:hover {
            background-color: 🟩#45a049;
        }
        .result {
            font-size: 1.5em;
            color: 🟩#4CAF50;
            text-align: center;
```



```html
<html>
<head>
    <style>
        .result {
            font-size: 1.5em;
            color: 🟩#4CAF50;
            text-align: center;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="form-container">
        <h2>Online Payment Fraud Detection</h2>
        <form action="/predict" method="post">
            <label for="Step">Step:</label>
            <input type="number" name="Step" required>

            <label for="Type">Type:</label>
            <input type="number" step="0.01" name="Type" required>

            <label for="Amount">Amount:</label>
            <input type="number" name="Amount" required>

            <label for="Sender Old Balance">Sender Old Balance:</label>
            <input type="number" name="Sender Old Balance" required>

            <label for="Recipient Old Balance">Recipient Old Balance:</label>
            <input type="number" step="0.01" name="Recipient Old Balance" required>

            <input type="submit" value="Predict">
        </form>
        {% if prediction_text %}
        <div class="result">{{ prediction_text }}</div>
        {% endif %}
    </div>
</body>
```

Google Cloud App Engine Deployment



Dashboard

Version: All versions

vcc-final-project.el.r.appspot.com

Region: asia-south1

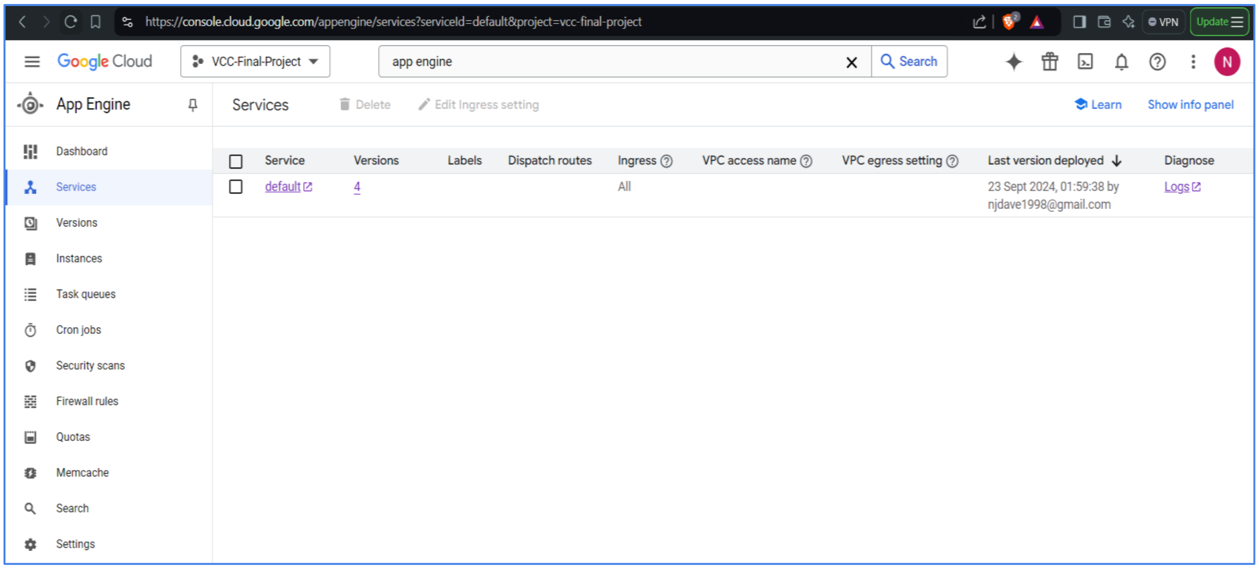
Chart settings: Summary

✓ 1 hour 6 hours 12 hours 1 day 2 days 4 days 7 days 14 days 30 days

Total requests: 0

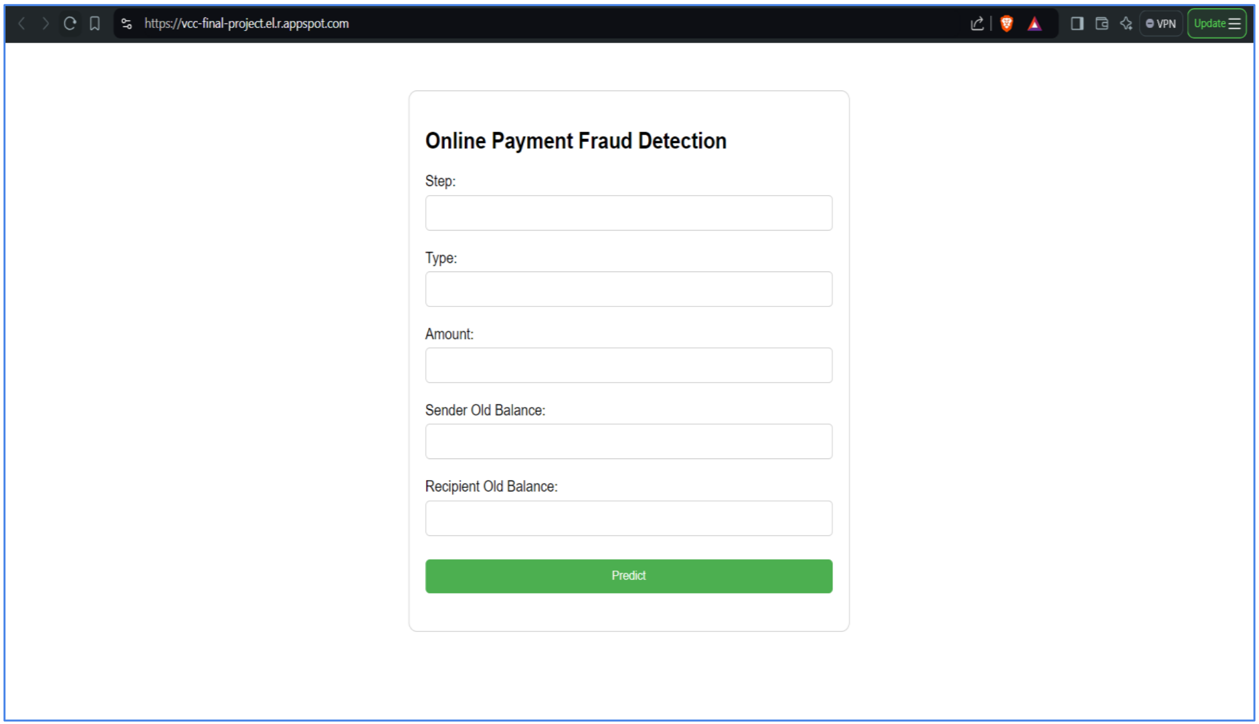
Instance summary

Billing status



Service	Versions	Labels	Dispatch routes	Ingress	VPC access name	VPC egress setting	Last version deployed	Diagnose
default	4			All			23 Sept 2024, 01:59:38 by njdave1998@gmail.com	Logs

App UI



Online Payment Fraud Detection

Step:

Type:

Amount:

Sender Old Balance:

Recipient Old Balance:

Thank You