Name: Shivashish Bhunia

R/No: 2110993833

Train/Test Notebooks: https://drive.google.com/drive/folders/1zIDvSDvBLa2Tj3QNEDT0lMN3GsOwcmgH?usp=sharing

GitHub Link: https://github.com/sh1vvy/CreditCardFraudDetection

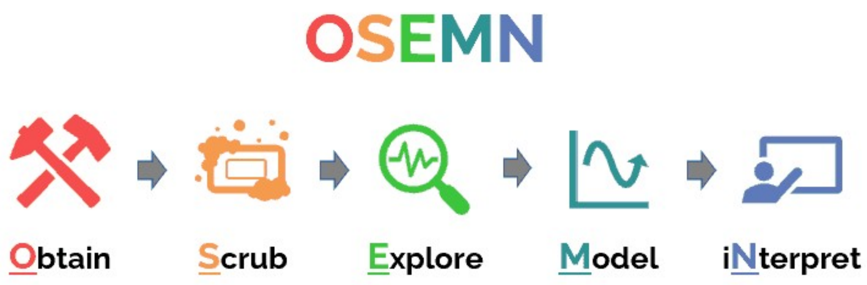
Live Project Website: https://creditcardfrauddetection-el22.onrender.com/

(more information regarding usage below) (TTL approx. 50s)

Applications used: Google Colab, Tableau, MS-Word/Excel

Documenting the process of solving the credit card fraud detection problem using Data Science techniques

I am using the **OSEMN** Framework for the entire process of solving this problem



Obtain:

- Collect and acquire the data you need for analysis.
- This might involve accessing databases, scraping websites, using APIs, or retrieving data from other sources.

Scrub:

- Clean the data to make it usable.
- Handle missing values, correct inconsistencies, remove duplicates, and ensure data is in the correct format.

Explore:

- Conduct exploratory data analysis (EDA) to understand the data's structure and patterns.
- Use statistical summaries, visualizations, and other techniques to uncover relationships, distributions, and anomalies.

Model:

- Develop predictive or descriptive models based on the cleaned and explored data.
- Choose appropriate algorithms and techniques (e.g., regression, classification, clustering) depending on the problem at hand.
- Train, validate, and fine-tune the models to improve performance.

i**Nterpret**:

- Interpret the results and translate them into actionable insights or decisions.
- Communicate findings to stakeholders, often through reports, visualizations, or presentations.

Note:

- Dependencies have been imported in the code given below as per examples.
- They are included all at once in the .ipynb notebook

1. Obtain (Data Acquisition)

The data for this project was obtained from [source]. It includes transactional records with attributes relevant to detecting credit card fraud. The data was loaded into the environment using the `pandas` library.

Example Code:

```
python
import pandas as pd
df = pd.read_csv('path/to/dataset.csv')
df.head()
```

2. Scrub (Data Cleaning)

Several steps were taken to clean the dataset, including handling missing values and removing duplicates. The data was transformed where necessary to ensure consistency and accuracy.

Example Code:

```
python
# Check for missing values
df.isnull().sum()

# Remove duplicates
df = df.drop_duplicates()

# Data transformations
df['column'] = df['column'].apply(transformation_function)
```

3. Explore (Data Exploration)

Exploratory Data Analysis (EDA) was performed to understand the distribution of the data and uncover any underlying patterns. This included generating descriptive statistics and visualizing key features.

Example Code:
python

```
# Summary statistics
df.describe()

# Visualizations
import matplotlib.pyplot as plt
df['column'].hist()
plt.show()
```

4. Model (Modeling)

Various machine learning models were trained and evaluated to predict credit card fraud. The models were tuned for optimal performance using techniques such as cross-validation.

Example Code:
python

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = RandomForestClassifier(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

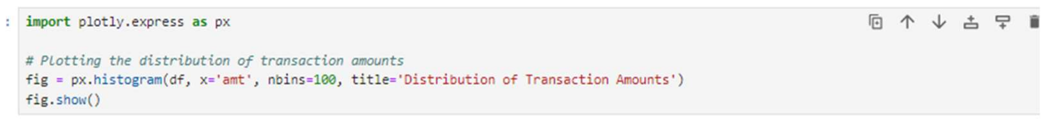
5. iNterpret (Interpretation)

The model's performance was evaluated based on precision, recall, F1-score, and accuracy. Key insights include the model's ability to identify fraudulent transactions with a balance between precision and recall.

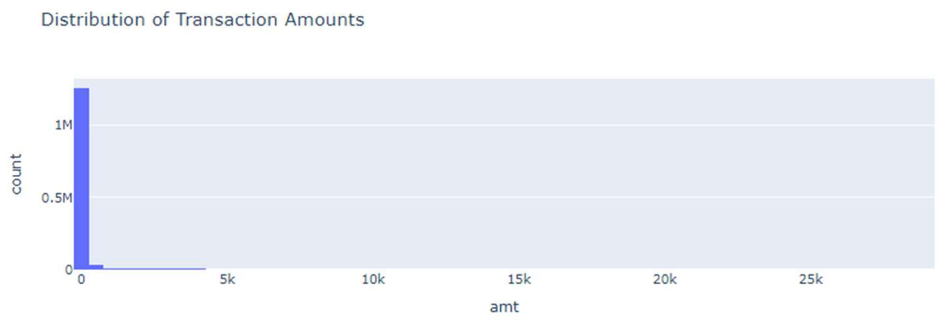
Example Code:
python

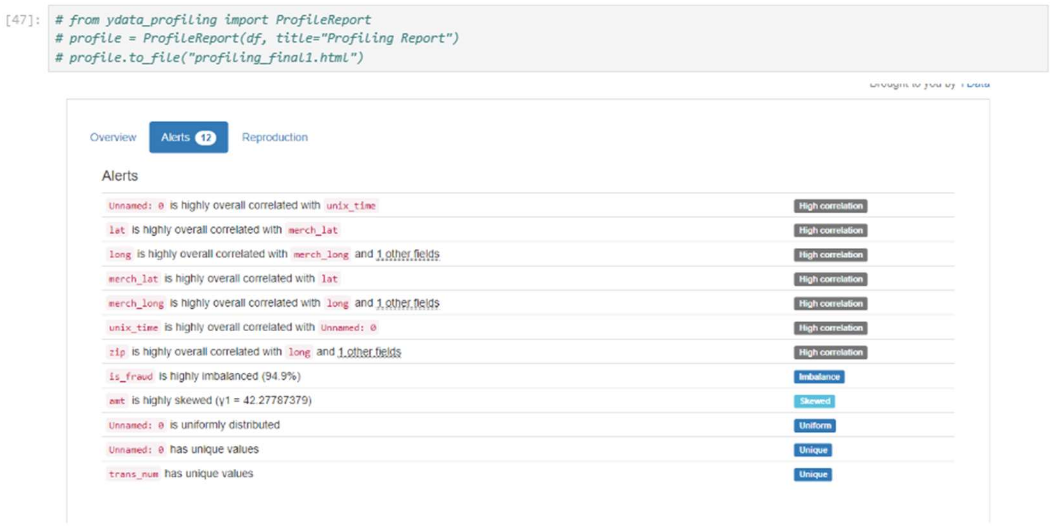
```
from sklearn.metrics import confusion_matrix, roc_auc_score

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
# ROC-AUC score
roc_auc = roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
print(f'ROC-AUC Score: {roc_auc:.2f}')
```

Initial Investigation done during Obtain & Scrub Phase/Exploring (Visualizations)



```
import plotly.express as px

# Plotting the distribution of transaction amounts
fig = px.histogram(df, x='amt', nbins=100, title='Distribution of Transaction Amounts')
fig.show()
```

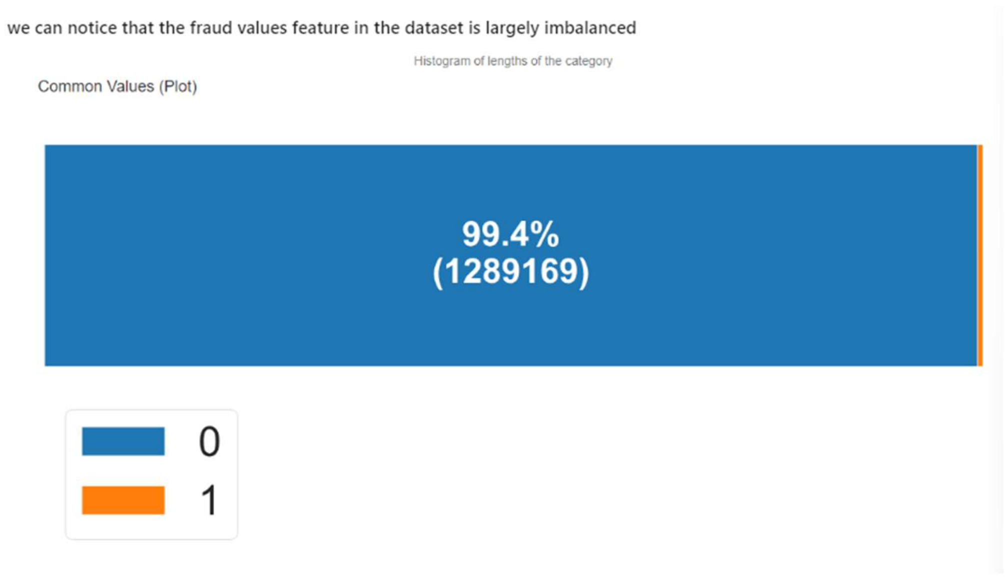




```
[47]: # from ydata_profiling import ProfileReport
      # profile = ProfileReport(df, title="Profiling Report")
      # profile.to_file("profiling_final1.html")
```

Alerts

`Unnamed: 0` is highly overall correlated with `unix_time`	High correlation
`lat` is highly overall correlated with `merch_lat`	High correlation
`long` is highly overall correlated with `merch_long` and 1 other fields	High correlation
`merch_lat` is highly overall correlated with `lat`	High correlation
`merch_long` is highly overall correlated with `long` and 1 other fields	High correlation
`unix_time` is highly overall correlated with `Unnamed: 0`	High correlation
`zip` is highly overall correlated with `long` and 1 other fields	High correlation
`is_fraud` is highly imbalanced (94.9%)	Imbalance
`amt` is highly skewed (γ1 = 42.27787379)	Skewed
`Unnamed: 0` is uniformly distributed	Uniform
`Unnamed: 0` has unique values	Unique
`trans_num` has unique values	Unique

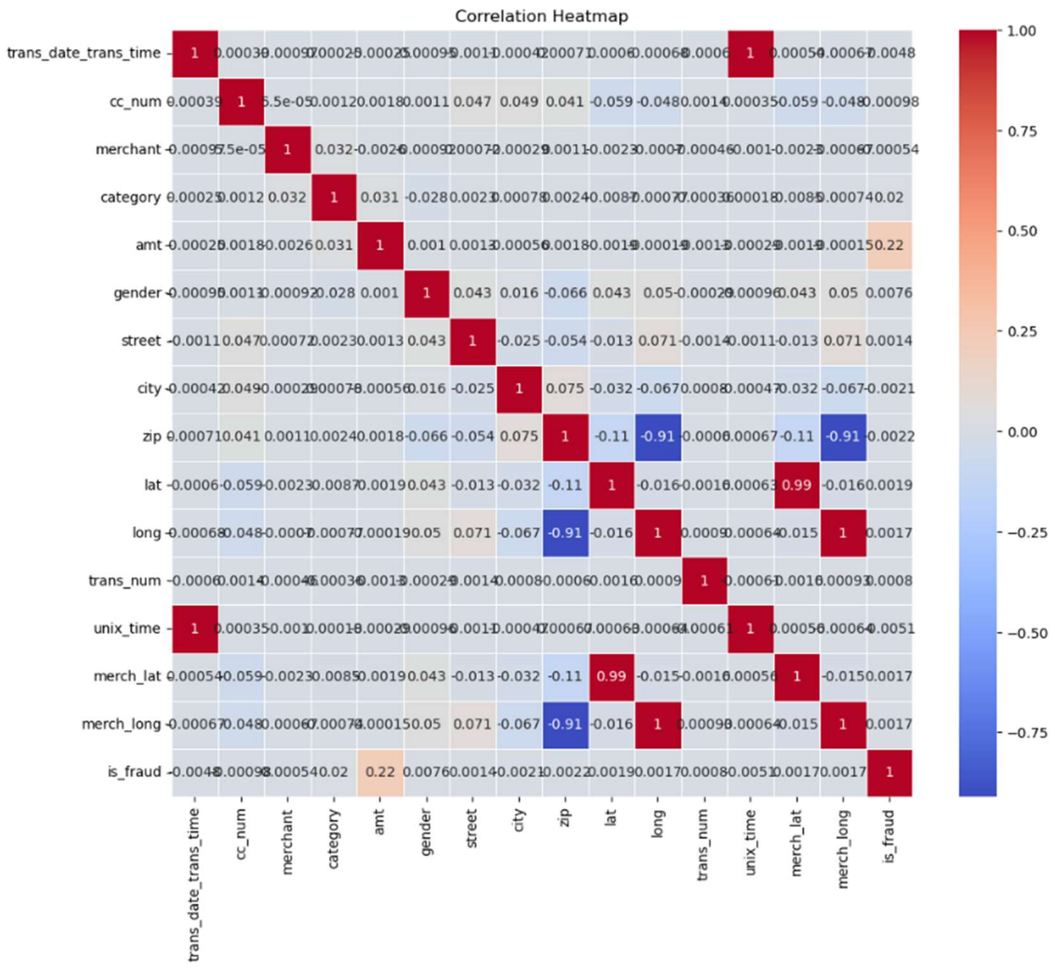

we can notice that the fraud values feature in the dataset is largely imbalanced

Data profiling tool used: https://github.com/ydataai/ydata-profiling

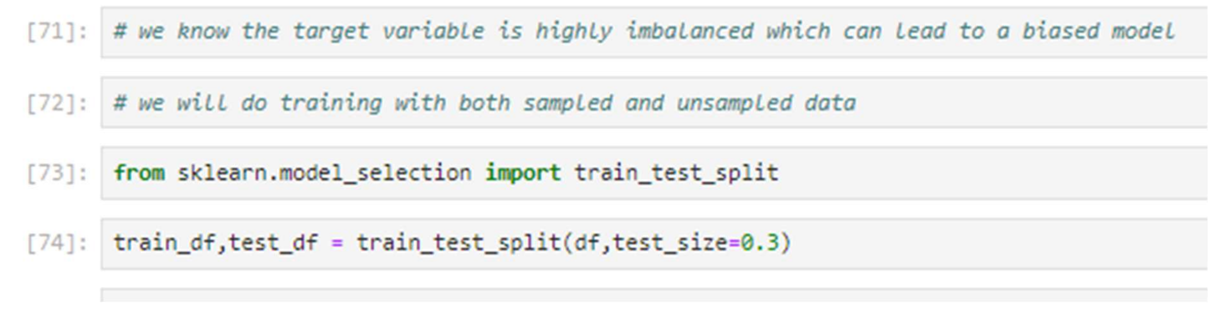
LabelEncoding and checking correlations between columns.

```
[59]: # For 'category' column
      category_enc = LabelEncoder()
      df['category'] = category_enc.fit_transform(df['category'])
      joblib.dump(category_enc, 'category_encoder.pkl')

      # For 'street' column
      street_enc = LabelEncoder()
      df['street'] = street_enc.fit_transform(df['street'])
      joblib.dump(street_enc, 'street_encoder.pkl')

      # For 'gender' column
      gender_enc = LabelEncoder()
      df['gender'] = gender_enc.fit_transform(df['gender'])
      joblib.dump(gender_enc, 'gender_encoder.pkl')

      # For 'city' column
      city_enc = LabelEncoder()
      df['city'] = city_enc.fit_transform(df['city'])
      joblib.dump(city_enc, 'city_encoder.pkl')

      # For 'merchant' column
      merchant_enc = LabelEncoder()
      df['merchant'] = merchant_enc.fit_transform(df['merchant'])
      joblib.dump(merchant_enc, 'merchant_encoder.pkl')

      # For 'trans_num' column
      trans_num_enc = LabelEncoder()
      df['trans_num'] = trans_num_enc.fit_transform(df['trans_num'])
      joblib.dump(trans_num_enc, 'trans_num_encoder.pkl')

      # For 'trans_date_trans_time' column
      trans_date_trans_time_enc = LabelEncoder()
      df['trans_date_trans_time'] = trans_date_trans_time_enc.fit_transform(df['trans_date_trans_time'])
      joblib.dump(trans_date_trans_time_enc, 'trans_date_trans_time_encoder.pkl')
```

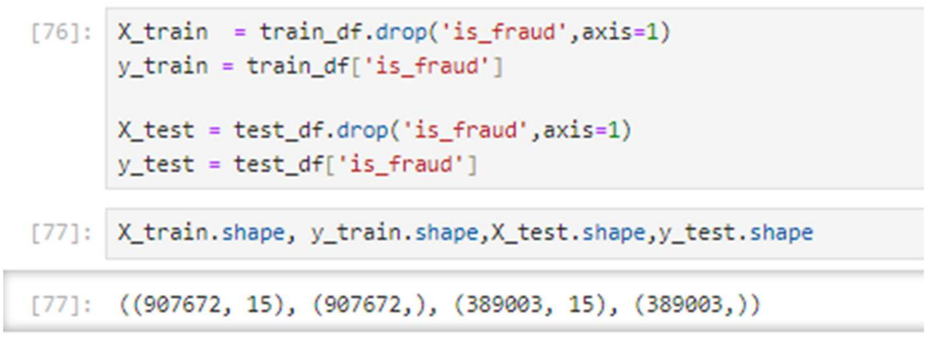
```
[59]: ['trans_date_trans_time_encoder.pkl']
```



Modelling the Data



```
[71]: # we know the target variable is highly imbalanced which can lead to a biased model

[72]: # we will do training with both sampled and unsampled data

[73]: from sklearn.model_selection import train_test_split

[74]: train_df,test_df = train_test_split(df,test_size=0.3)
```



```
[76]: X_train  = train_df.drop('is_fraud',axis=1)
      y_train = train_df['is_fraud']

      X_test = test_df.drop('is_fraud',axis=1)
      y_test = test_df['is_fraud']

[77]: X_train.shape, y_train.shape,X_test.shape,y_test.shape

[77]: ((907672, 15), (907672,), (389003, 15), (389003,))
```

We train the models

i) Without Sampling
ii) With SMOTE Sampling (for managing imbalance)

On approaching (i) we find that due to the high imbalance in the fraud(target) variable, the predictions made by the models are very inacceptable.

We applied several machine learning models to predict fraudulent transactions. The primary models we used include:

- Logistic Regression
- Decision Trees
- Random Forest
- Gradient Boosting
- Deep Learning (Neural Network)

Given the nature of the data, which was highly imbalanced, we applied the Synthetic Minority Over-sampling Technique (SMOTE) to balance the dataset. After balancing the data, we found that the deep learning model provided the most accurate predictions.

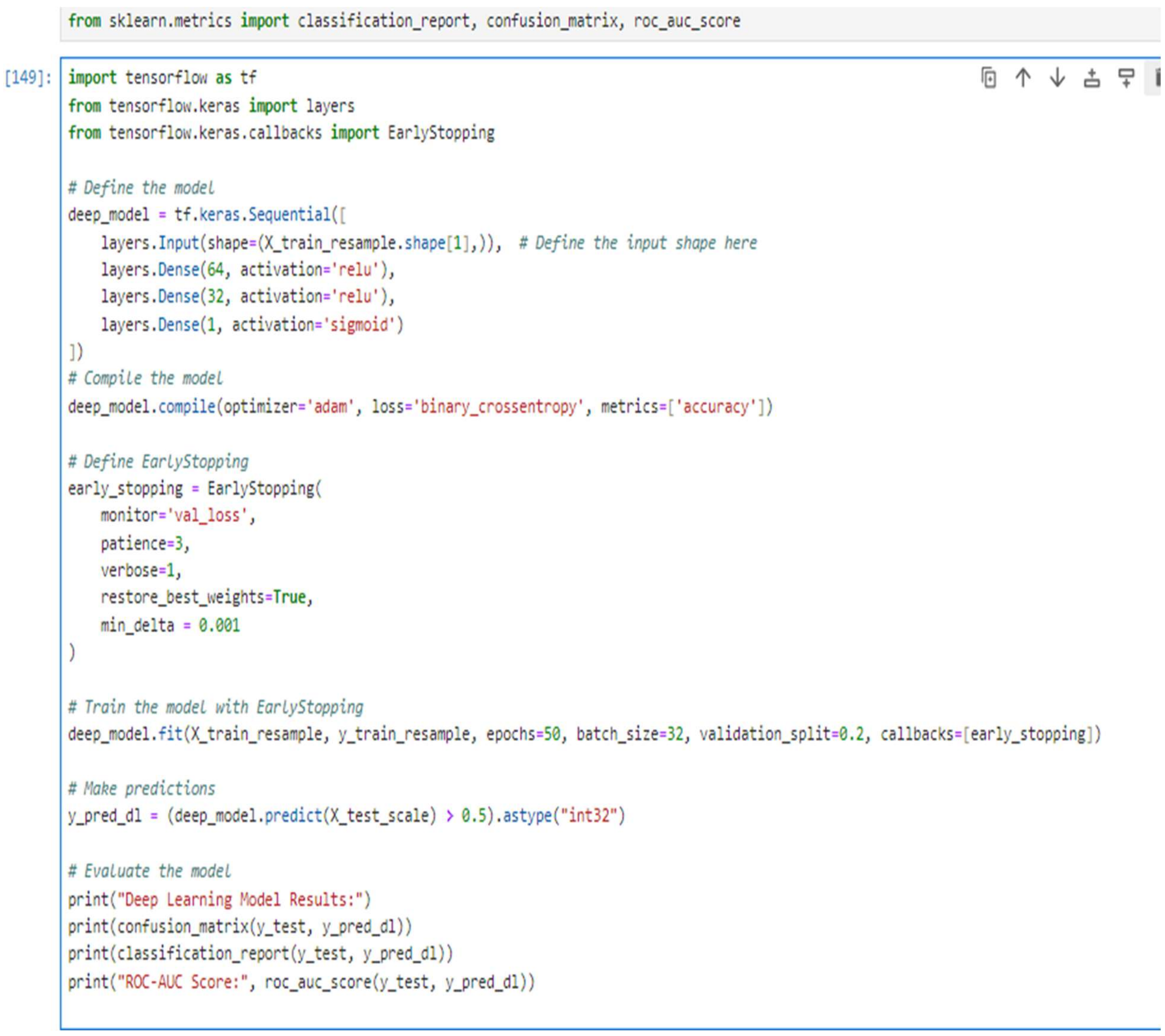
Handling Data Imbalance with SMOTE

Initially, the dataset was highly imbalanced, with a small percentage of fraudulent transactions compared to non-fraudulent ones. This imbalance caused issues with the performance of our models, particularly in correctly identifying fraudulent transactions.

To address this issue, we applied SMOTE, which generates synthetic samples for the minority class (fraudulent transactions) to create a more balanced dataset. After applying SMOTE, we re-trained all the models, including the deep learning model, which ultimately provided the best results.

Deep Learning Model Implementation

Below is the code used to implement the deep learning model. The model consists of a neural network with two hidden layers and an output layer designed for binary classification (fraud vs. not fraud). We also incorporated EarlyStopping to prevent overfitting by stopping training when the model's performance on the validation set stopped improving.



```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

```
[149]: import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping

# Define the model
deep_model = tf.keras.Sequential([
    layers.Input(shape=(X_train_resample.shape[1],)),  # Define the input shape here
    layers.Dense(64, activation='relu'),
    layers.Dense(32, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
# Compile the model
deep_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Define EarlyStopping
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    verbose=1,
    restore_best_weights=True,
    min_delta = 0.001
)

# Train the model with EarlyStopping
deep_model.fit(X_train_resample, y_train_resample, epochs=50, batch_size=32, validation_split=0.2, callbacks=[early_stopping])

# Make predictions
y_pred_dl = (deep_model.predict(X_test_scale) > 0.5).astype("int32")

# Evaluate the model
print("Deep Learning Model Results:")
print(confusion_matrix(y_test, y_pred_dl))
print(classification_report(y_test, y_pred_dl))
print("ROC-AUC Score:", roc_auc_score(y_test, y_pred_dl))
```

Results and Conclusion

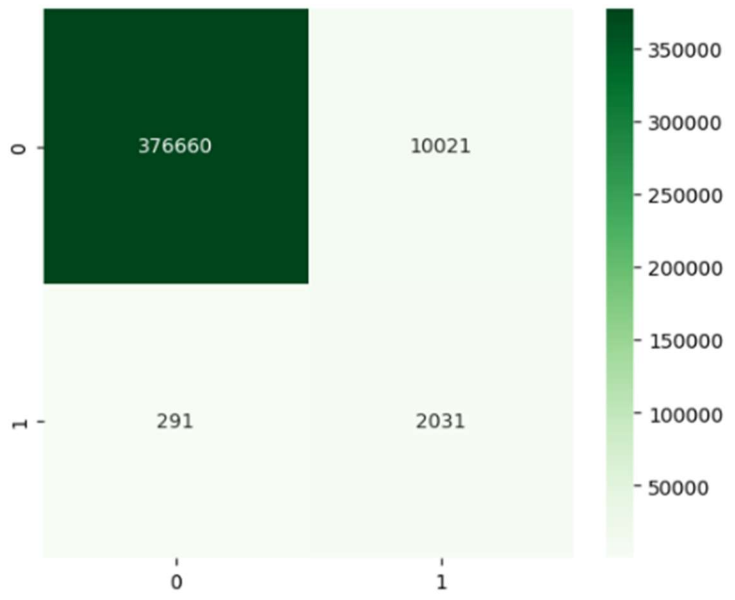
- **Logistic Regression, Decision Trees, Random Forest, and Gradient Boosting**: These models were trained and evaluated but struggled with the imbalanced nature of the dataset. After applying SMOTE, their performance improved, but they were still outperformed by the deep learning model.
- **Deep Learning Model**: The deep learning model, after being trained on the SMOTE-resampled data, provided the best performance. The ROC-AUC score and other evaluation metrics indicated that the model was able to effectively distinguish between fraudulent and non-fraudulent transactions.
- **Final Model**: Based on the evaluation metrics, the deep learning model was selected as the final model for deployment. This model, combined with the preprocessing steps, is robust in identifying fraudulent transactions in the provided dataset.

Training Data

```
[150]: DL_conf_mat = confusion_matrix(y_test, y_pred_dl)
```

```
[151]: fig,axes = plt.subplots(figsize=(6,5))
       sns.heatmap(DL_conf_mat,annot=True,axes=axes,cmap='Greens',fmt='d')
```

```
[151]: <Axes: >
```

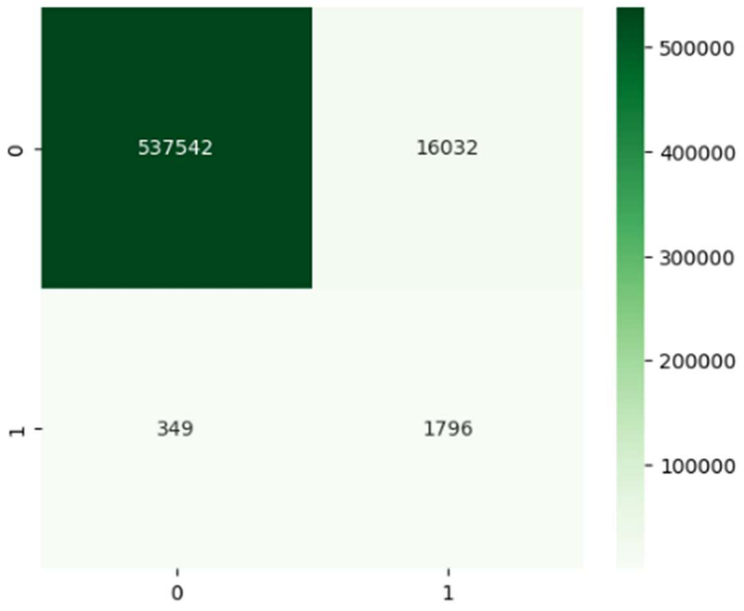


Testing Data

```
[17]: DL_conf_mat = confusion_matrix(y_test, y_pred_binary)
```

```
[22]: fig,axes = plt.subplots(figsize=(6,5))
      sns.heatmap(DL_conf_mat,annot=True,axes=axes,cmap='Greens',fmt='d')
```

```
[22]: <Axes: >
```



Flask Website and Deployment on Render

Overview

As part of this project, we developed a Flask web application to deploy our trained deep learning model. The website allows users to input transaction data in JSON format and receive predictions on whether the transaction is fraudulent or not.

Website Features

- **Frontend**: The website has a simple, user-friendly interface where users can input transaction data in JSON format. The input is processed through the Flask application, which uses the trained deep learning model to predict whether the transaction is fraudulent.
- **Backend**: The backend is powered by Flask, a lightweight web framework for Python. The backend handles the data preprocessing, model prediction, and returns the result (either "Fraud" or "Not Fraud") based on the model's output.

Deployment on Render

The Flask application was successfully deployed on Render, a popular platform for hosting web applications. The deployment included both the frontend interface and the backend API that handles predictions.

Issue with Render Deployment:

- **Memory Limitation**: While the website runs successfully on Render, the platform encounters memory limitations when attempting to make predictions. The deep learning model, when loaded into memory and used for prediction, consumes more resources than what is allocated by the free tier on Render.
- **Result**: Due to this, the application is able to handle the frontend operations (displaying the website, taking input), but fails when it comes to processing the prediction request, resulting in the application running out of memory and crashing.

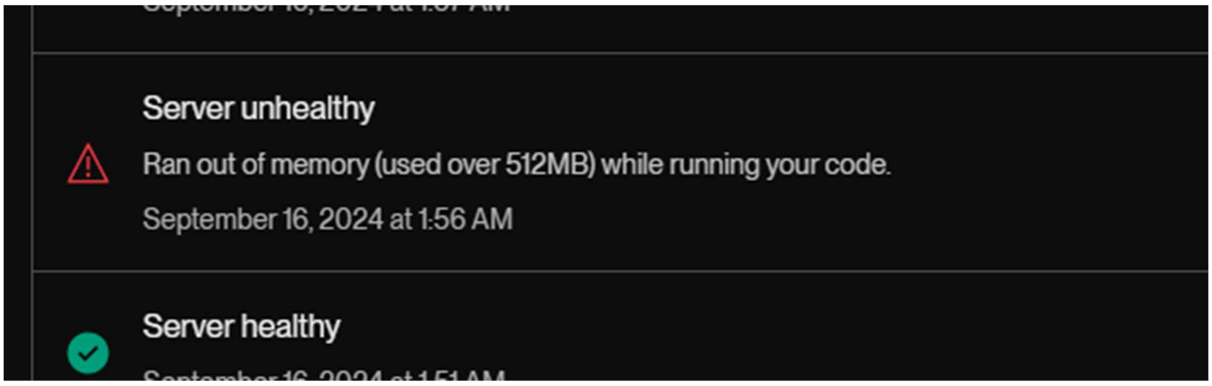
Local Execution

- **Successful Operation**: When run locally, the Flask application performs as expected. The model can be loaded into memory, predictions are made without issues, and the application does not encounter memory limitations.
- **Recommendation**: For testing and use, it is recommended to run the application locally, especially when working with large models or data that requires significant computational resources.

Conclusion

- **Render Deployment**: While the application was successfully deployed on Render, the memory limitations of the platform prevent it from handling the prediction process. This issue highlights the importance of considering resource allocation and platform capabilities when deploying machine learning models.
- **Local Use**: The application runs smoothly when executed locally, making it a reliable tool for making predictions using the trained deep learning model. For future deployments, considering a platform with higher memory capacity or optimizing the model for lower memory usage could help mitigate this issue.

This summary outlines the development and deployment of the Flask web application, its limitations on the Render platform, and the successful operation of the application in a local environment.

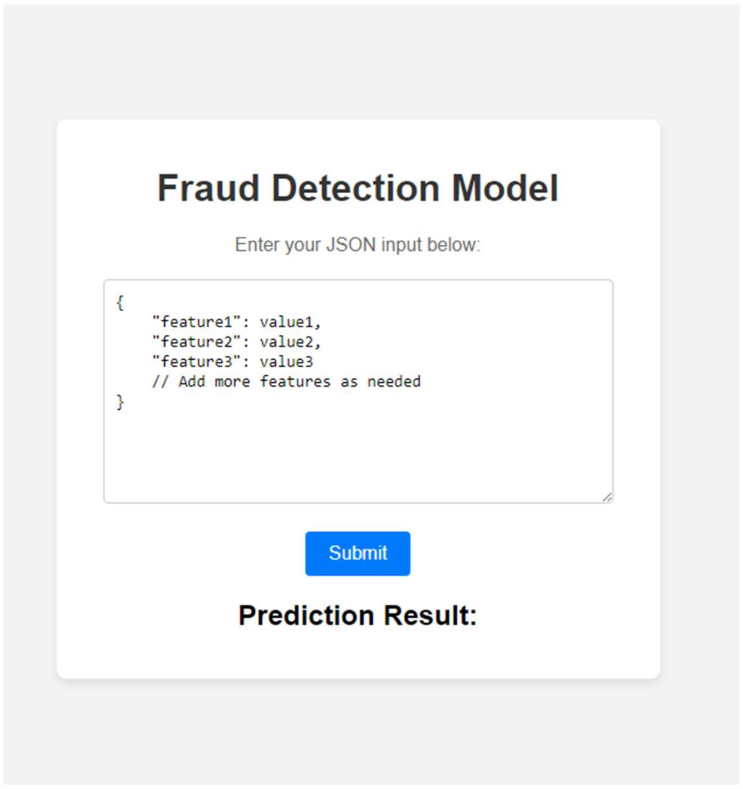


Server unhealthy

Ran out of memory (used over 512MB) while running your code.

September 16, 2024 at 1:56 AM

Server healthy

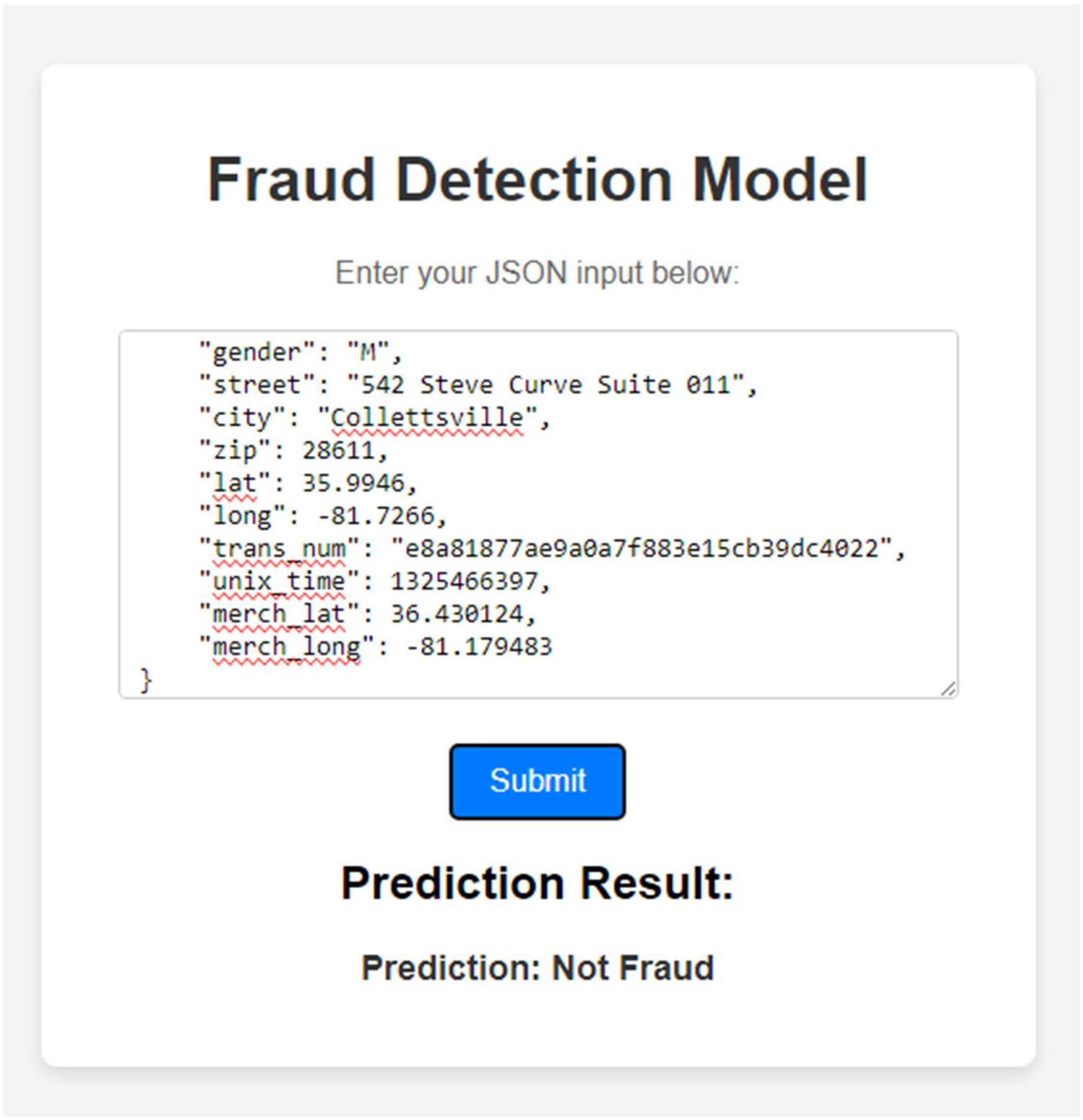

Fraud Detection Model

Enter your JSON input below:

```
{
    "feature1": value1,
    "feature2": value2,
    "feature3": value3
    // Add more features as needed
}
```

Prediction Result:

When run locally, it predicts whether the transaction is fraudulent or not



Fraud Detection Model

Enter your JSON input below:

```
    "gender": "M",
    "street": "542 Steve Curve Suite 011",
    "city": "Collettsville",
    "zip": 28611,
    "lat": 35.9946,
    "long": -81.7266,
    "trans_num": "e8a81877ae9a0a7f883e15cb39dc4022",
    "unix_time": 1325466397,
    "merch_lat": 36.430124,
    "merch_long": -81.179483
}
```

Prediction Result:

Prediction: Not Fraud