DATA SCIENCE AND ANALYTICS

SECURITY ANALYSIS

Fraud Detection System for Finance

- Mahima S [2022115018]

1. Introduction

In the dynamic financial industry, fraud is a significant and persistent challenge. Financial institutions and e-commerce platforms face billions in losses due to fraudulent activities such as credit card fraud and identity theft. Machine learning models, particularly fraud detection systems, have become essential tools in identifying and preventing such financial threats. This project focuses on developing a fraud detection system using machine learning, deployed as an interactive web app using **Streamlit**, a Python framework for building data applications.

Project Objective

The goal of this project is to develop a real-time fraud detection system that can classify transactions as legitimate or fraudulent based on transaction details. The system was deployed as a **Streamlit** web application to allow users to input transaction details and get immediate fraud predictions.

2. Problem Definition

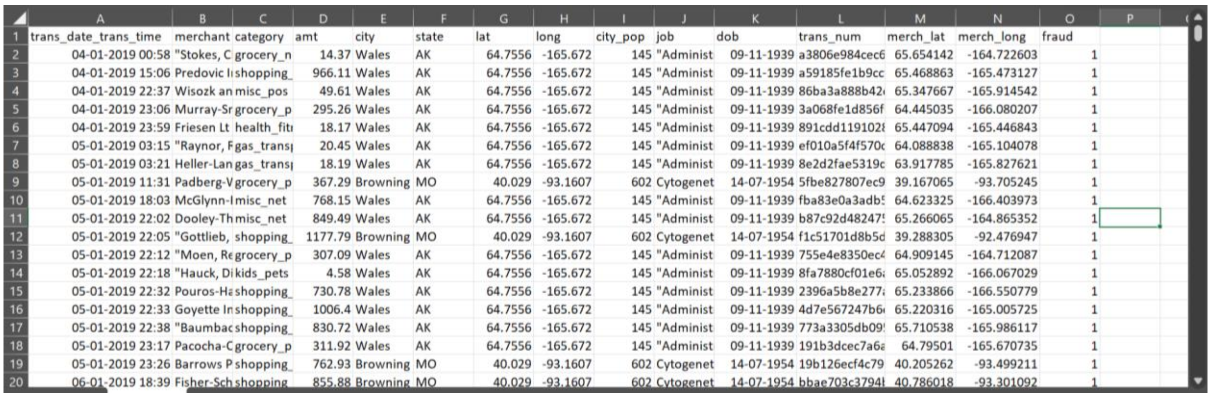
The primary challenge addressed by this system is the identification of fraudulent transactions on an e-commerce platform. The system is built to detect potentially fraudulent transactions based on various input features such as the transaction amount, user behavior, and geographical information.

3. Data Collection

The dataset used for this project is the **IEEE-CIS Fraud Detection Dataset**, which contains transaction records with features like transaction amount, user ID, merchant ID, device information, and labels indicating whether the transaction is fraudulent or legitimate.

- **Dataset Overview**:
  - **Features**: The dataset includes user and merchant information, transaction amount, time, and various other transaction details.
  - **Label**: A binary label indicating whether the transaction is fraudulent (1) or legitimate (0).
  - **Imbalance**: The dataset is imbalanced with a larger proportion of legitimate transactions compared to fraudulent ones.

  Dateset:



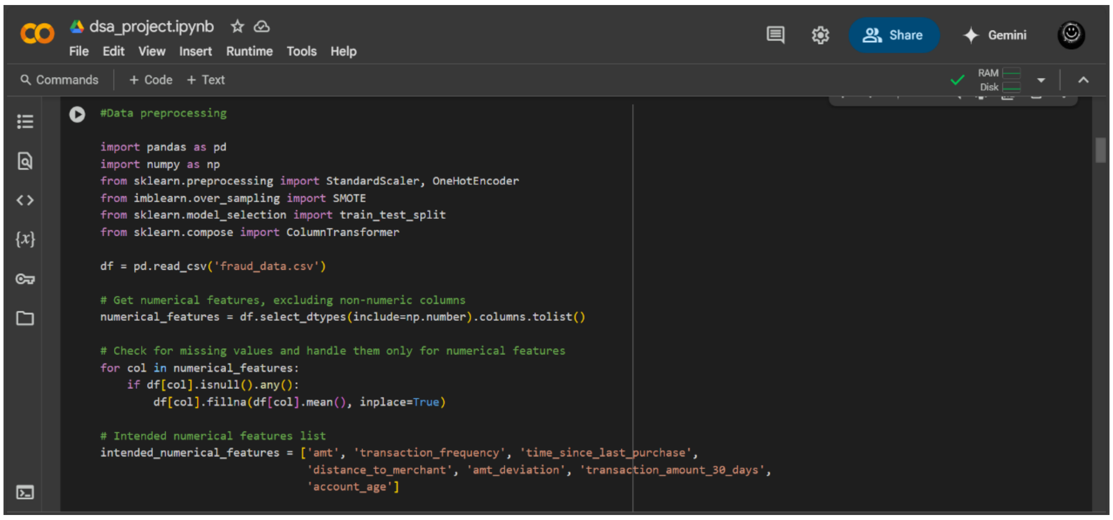
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	trans_date_trans_time	merchant	category	amt	city	state	lat	long	city_pop	job	dob	trans_num	merch_lat	merch_long	fraud	
2	04-01-2019 00:58	"Stokes, C	grocery_n	14.37	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	a3806e984cec6	65.654142	-164.722603	1	
3	04-01-2019 15:06	Predovic I	shopping_	966.11	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	a59185fe1b9cc	65.468863	-165.473127	1	
4	04-01-2019 22:37	Wisozk an	misc_pos	49.61	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	86ba3a888b42	65.347667	-165.914542	1	
5	04-01-2019 23:06	Murray-Sr	grocery_p	295.26	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	3a068fe1d856f	64.445035	-166.080207	1	
6	04-01-2019 23:59	Friesen Lt	health_fit	18.17	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	891cdd119102	65.447094	-165.446843	1	
7	05-01-2019 03:15	"Raynor, F	gas_trans	20.45	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	ef010a5f4f570	64.088838	-165.104078	1	
8	05-01-2019 03:21	Heller-Lan	gas_trans	18.19	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	8e2d2fae5319c	63.917785	-165.827621	1	
9	05-01-2019 11:31	Padberg-V	grocery_p	367.29	Browning	MO	40.029	-93.1607	602	Cytogenet	14-07-1954	5fbe827807ec9	39.167065	-93.705245	1	
10	05-01-2019 18:03	McGlynn-I	misc_net	768.15	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	fba83e0a3adb5	64.623325	-166.403973	1	
11	05-01-2019 22:02	Dooley-Th	misc_net	849.49	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	b87c92d48247	65.266065	-164.865352	1	
12	05-01-2019 22:05	"Gottlieb,	shopping_	1177.79	Browning	MO	40.029	-93.1607	602	Cytogenet	14-07-1954	f1c51701d8b5d	39.288305	-92.476947	1	
13	05-01-2019 22:12	"Moen, Re	grocery_p	307.09	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	755e4e8350ec4	64.909145	-164.712087	1	
14	05-01-2019 22:18	"Hauck, Di	kids_pets	4.58	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	8fa7880cf01e6	65.052892	-166.067029	1	
15	05-01-2019 22:32	Pouros-Ha	shopping_	730.78	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	2396a5b8e277	65.233866	-166.550779	1	
16	05-01-2019 22:33	Goyette In	shopping_	1006.4	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	4d7e567247b6	65.220316	-165.005725	1	
17	05-01-2019 22:38	"Baumbac	shopping_	830.72	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	773a3305db09	65.710538	-165.986117	1	
18	05-01-2019 23:17	Pacocha-C	grocery_p	311.92	Wales	AK	64.7556	-165.672	145	"Administ	09-11-1939	191b3dcec7a6	64.79501	-165.670735	1	
19	05-01-2019 23:26	Barrows P	shopping_	762.93	Browning	MO	40.029	-93.1607	602	Cytogenet	14-07-1954	19b126ecf4c79	40.205262	-93.499211	1	
20	06-01-2019 18:39	Fisher-Sch	shopping	855.88	Browning	MO	40.029	-93.1607	602	Cytogenet	14-07-1954	bbae703c3794	40.786018	-93.301092	1	

4. Data Preprocessing

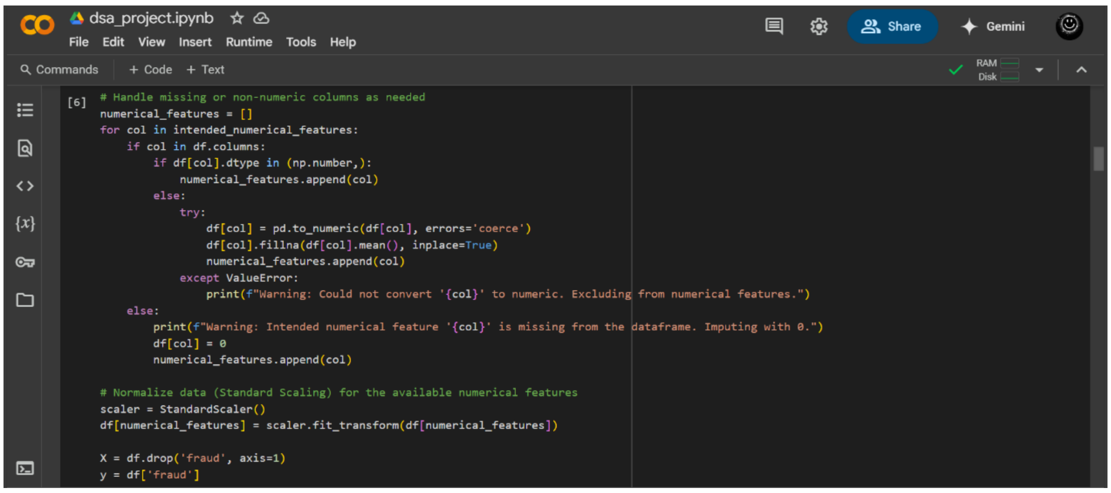
Data preprocessing ensures that the data is clean, normalized, and ready for machine learning. The key preprocessing steps include:

- **Handling Missing Values**: Missing data was imputed using statistical methods like filling with the mean for numerical features or the most frequent value for categorical features.
- **Normalization**: Numerical features, including the transaction amount, were normalized to ensure they are on a similar scale.
- **Class Imbalance**: To address the class imbalance, **SMOTE (Synthetic Minority Over-sampling Technique)** was used to generate synthetic examples of fraudulent transactions.

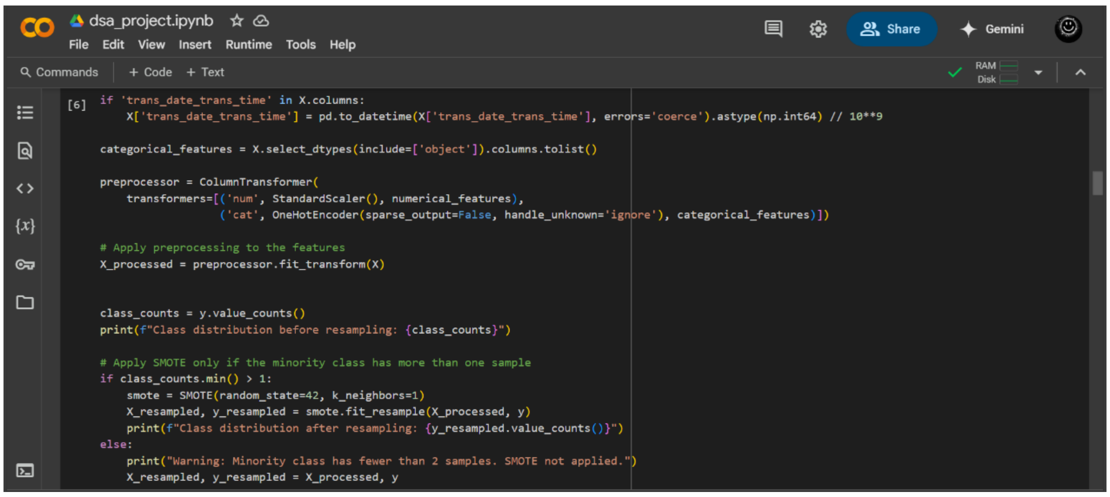
Code:



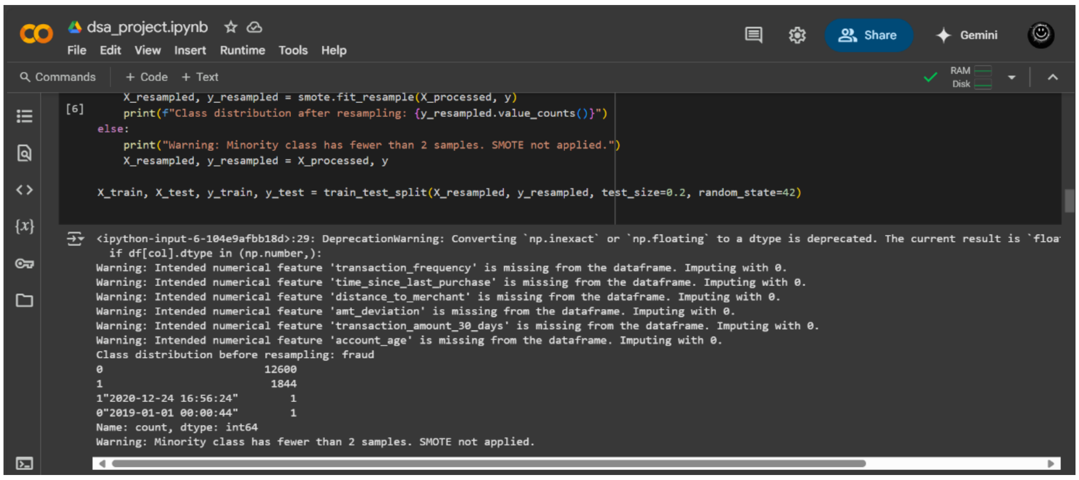
```python
#Data preprocessing

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer

df = pd.read_csv('fraud_data.csv')

# Get numerical features, excluding non-numeric columns
numerical_features = df.select_dtypes(include=np.number).columns.tolist()

# Check for missing values and handle them only for numerical features
for col in numerical_features:
    if df[col].isnull().any():
        df[col].fillna(df[col].mean(), inplace=True)

# Intended numerical features list
intended_numerical_features = ['amt', 'transaction_frequency', 'time_since_last_purchase',
                               'distance_to_merchant', 'amt_deviation', 'transaction_amount_30_days',
                               'account_age']
```



```python
[6] # Handle missing or non-numeric columns as needed
    numerical_features = []
    for col in intended_numerical_features:
        if col in df.columns:
            if df[col].dtype in (np.number,):
                numerical_features.append(col)
            else:
                try:
                    df[col] = pd.to_numeric(df[col], errors='coerce')
                    df[col].fillna(df[col].mean(), inplace=True)
                    numerical_features.append(col)
                except ValueError:
                    print(f"Warning: Could not convert '{col}' to numeric. Excluding from numerical features.")
        else:
            print(f"Warning: Intended numerical feature '{col}' is missing from the dataframe. Imputing with 0.")
            df[col] = 0
            numerical_features.append(col)

    # Normalize data (Standard Scaling) for the available numerical features
    scaler = StandardScaler()
    df[numerical_features] = scaler.fit_transform(df[numerical_features])

    X = df.drop('fraud', axis=1)
    y = df['fraud']
```



```python
[6] if 'trans_date_trans_time' in X.columns:
        X['trans_date_trans_time'] = pd.to_datetime(X['trans_date_trans_time'], errors='coerce').astype(np.int64) // 10**9

    categorical_features = X.select_dtypes(include=['object']).columns.tolist()

    preprocessor = ColumnTransformer(
        transformers=[('num', StandardScaler(), numerical_features),
                      ('cat', OneHotEncoder(sparse_output=False, handle_unknown='ignore'), categorical_features)])

    # Apply preprocessing to the features
    X_processed = preprocessor.fit_transform(X)


    class_counts = y.value_counts()
    print(f"Class distribution before resampling: {class_counts}")

    # Apply SMOTE only if the minority class has more than one sample
    if class_counts.min() > 1:
        smote = SMOTE(random_state=42, k_neighbors=1)
        X_resampled, y_resampled = smote.fit_resample(X_processed, y)
        print(f"Class distribution after resampling: {y_resampled.value_counts()}")
    else:
        print("Warning: Minority class has fewer than 2 samples. SMOTE not applied.")
        X_resampled, y_resampled = X_processed, y
```



```
[6]     X_resampled, y_resampled = smote.fit_resample(X_processed, y)
        print(f"Class distribution after resampling: {y_resampled.value_counts()}")
    else:
        print("Warning: Minority class has fewer than 2 samples. SMOTE not applied.")
        X_resampled, y_resampled = X_processed, y

    X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

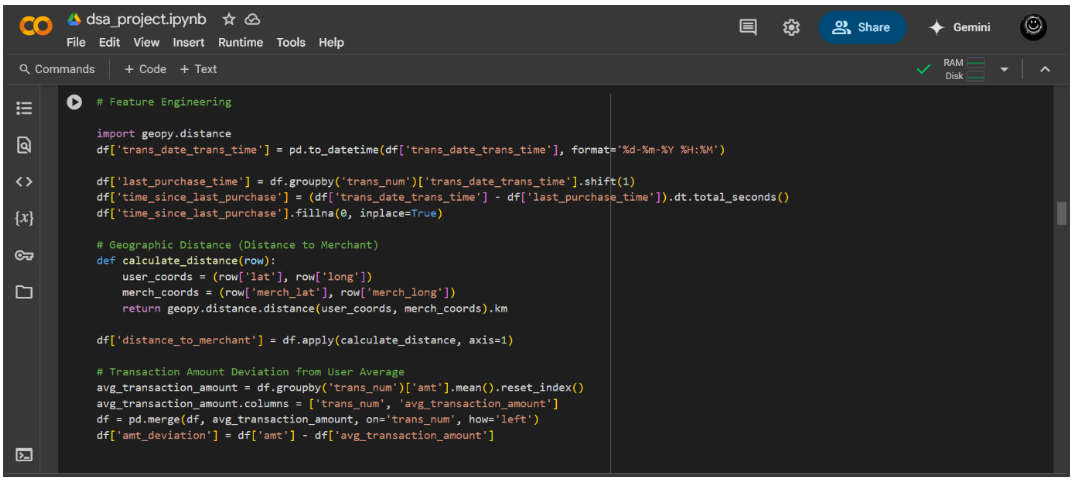
```
<ipython-input-6-104e9afbb18d>:29: DeprecationWarning: Converting `np.inexact` or `np.floating` to a dtype is deprecated. The current result is `floa
  if df[col].dtype in (np.number,):
Warning: Intended numerical feature 'transaction_frequency' is missing from the dataframe. Imputing with 0.
Warning: Intended numerical feature 'time_since_last_purchase' is missing from the dataframe. Imputing with 0.
Warning: Intended numerical feature 'distance_to_merchant' is missing from the dataframe. Imputing with 0.
Warning: Intended numerical feature 'amt_deviation' is missing from the dataframe. Imputing with 0.
Warning: Intended numerical feature 'transaction_amount_30_days' is missing from the dataframe. Imputing with 0.
Warning: Intended numerical feature 'account_age' is missing from the dataframe. Imputing with 0.
Class distribution before resampling: fraud
0                        12600
1                         1844
1"2020-12-24 16:56:24"       1
0"2019-01-01 00:00:44"       1
Name: count, dtype: int64
Warning: Minority class has fewer than 2 samples. SMOTE not applied.
```

5. Feature Engineering

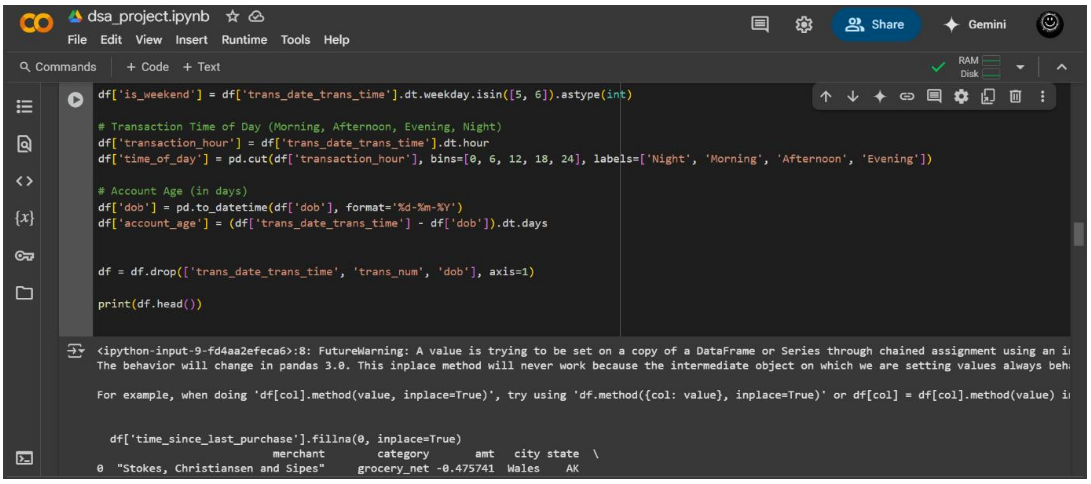
Feature engineering plays a pivotal role in improving the model's accuracy. Here are the key features derived from the raw transaction data:

- **Transaction Amount**: The monetary value of the transaction.
- **Time Since Last Purchase**: The time gap between the current and previous transaction for a user.
- **Transaction Frequency**: The frequency of transactions made by the user within a specific time frame.
- **Geographic Distance**: The distance between the user's location and the merchant's location.
- **Device Information**: Whether the transaction was made from a known or new device.

Code:

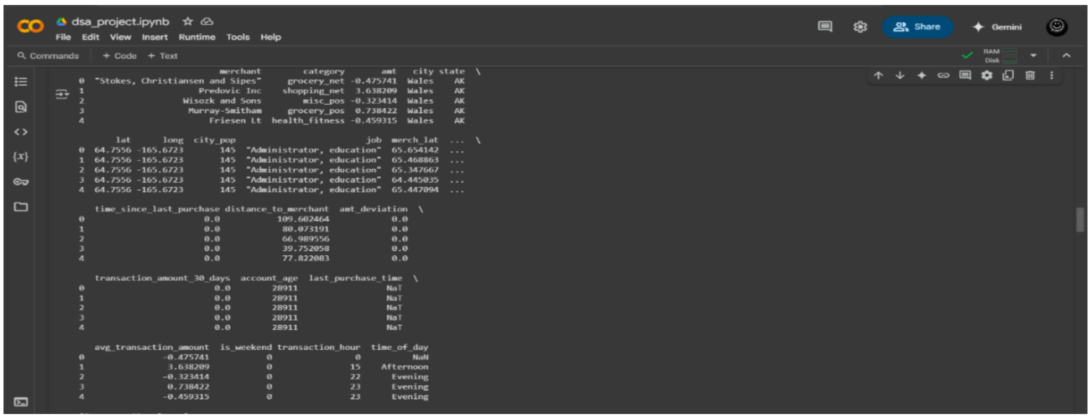


```
# Feature Engineering

import geopy.distance
df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'], format='%d-%m-%Y %H:%M')

df['last_purchase_time'] = df.groupby('trans_num')['trans_date_trans_time'].shift(1)
df['time_since_last_purchase'] = (df['trans_date_trans_time'] - df['last_purchase_time']).dt.total_seconds()
df['time_since_last_purchase'].fillna(0, inplace=True)

# Geographic Distance (Distance to Merchant)
def calculate_distance(row):
    user_coords = (row['lat'], row['long'])
    merch_coords = (row['merch_lat'], row['merch_long'])
    return geopy.distance.distance(user_coords, merch_coords).km

df['distance_to_merchant'] = df.apply(calculate_distance, axis=1)

# Transaction Amount Deviation from User Average
avg_transaction_amount = df.groupby('trans_num')['amt'].mean().reset_index()
avg_transaction_amount.columns = ['trans_num', 'avg_transaction_amount']
df = pd.merge(df, avg_transaction_amount, on='trans_num', how='left')
df['amt_deviation'] = df['amt'] - df['avg_transaction_amount']
```



```python
df['is_weekend'] = df['trans_date_trans_time'].dt.weekday.isin([5, 6]).astype(int)

# Transaction Time of Day (Morning, Afternoon, Evening, Night)
df['transaction_hour'] = df['trans_date_trans_time'].dt.hour
df['time_of_day'] = pd.cut(df['transaction_hour'], bins=[0, 6, 12, 18, 24], labels=['Night', 'Morning', 'Afternoon', 'Evening'])

# Account Age (in days)
df['dob'] = pd.to_datetime(df['dob'], format='%d-%m-%Y')
df['account_age'] = (df['trans_date_trans_time'] - df['dob']).dt.days


df = df.drop(['trans_date_trans_time', 'trans_num', 'dob'], axis=1)

print(df.head())
```

```
<ipython-input-9-fd4aa2efeca6>:8: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an i
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always beh

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) i


  df['time_since_last_purchase'].fillna(0, inplace=True)
                        merchant       category       amt   city state  \
0  "Stokes, Christiansen and Sipes"    grocery_net -0.475741  Wales    AK
```



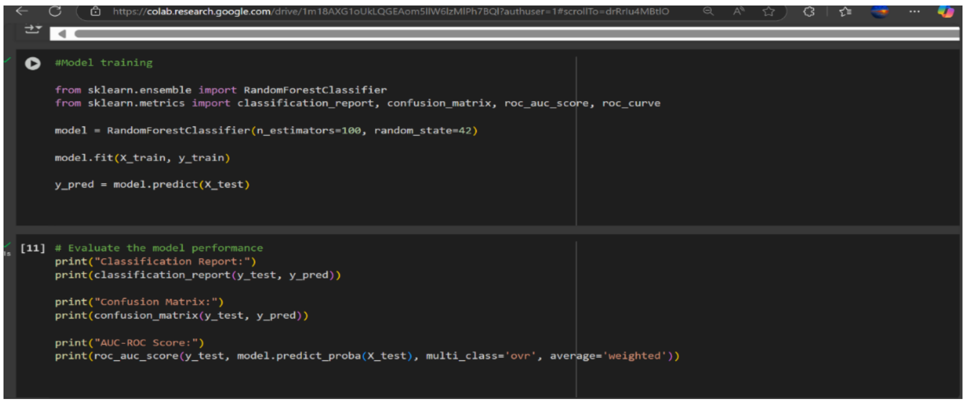
```
                        merchant       category       amt   city state  \
0  "Stokes, Christiansen and Sipes"    grocery_net -0.475741  Wales    AK
1                   Predovic Inc    shopping_net  3.638209  Wales    AK
2                Wisozk and Sons        misc_pos -0.323414  Wales    AK
3                 Murray-Smitham     grocery_pos  0.738422  Wales    AK
4                     Friesen Lt  health_fitness -0.459315  Wales    AK

       lat      long  city_pop                             job  merch_lat  ...  \
0  64.7556 -165.6723       145  "Administrator, education"  65.654142  ...
1  64.7556 -165.6723       145  "Administrator, education"  65.468863  ...
2  64.7556 -165.6723       145  "Administrator, education"  65.347667  ...
3  64.7556 -165.6723       145  "Administrator, education"  64.445035  ...
4  64.7556 -165.6723       145  "Administrator, education"  65.447094  ...

   time_since_last_purchase distance_to_merchant  amt_deviation  \
0                       0.0           109.602464            0.0
1                       0.0            80.073191            0.0
2                       0.0            66.989556            0.0
3                       0.0            39.752058            0.0
4                       0.0            77.822083            0.0

   transaction_amount_30_days  account_age  last_purchase_time  \
0                         0.0        28911                 NaT
1                         0.0        28911                 NaT
2                         0.0        28911                 NaT
3                         0.0        28911                 NaT
4                         0.0        28911                 NaT

   avg_transaction_amount  is_weekend transaction_hour  time_of_day
0               -0.475741           0                0          NaN
1                3.638209           0               15    Afternoon
2               -0.323414           0               22      Evening
3                0.738422           0               23      Evening
4               -0.459315           0               23      Evening
```

6. Model Development And Model Evaluation

To evaluate the models, metrics such as precision, recall, F1-score, and AUC-ROC were used. These metrics are essential in evaluating how well the model performs, especially in imbalanced datasets.

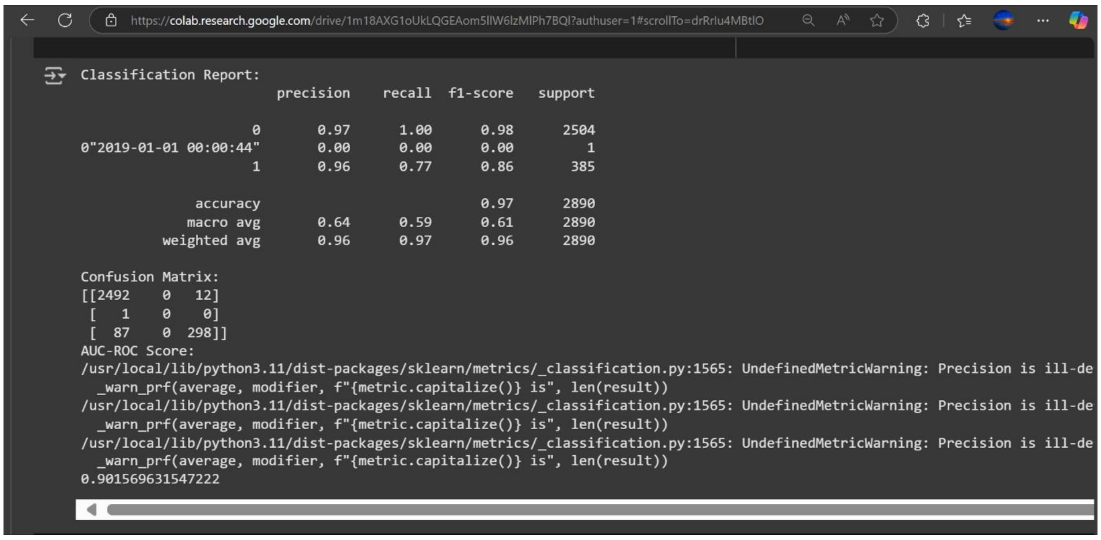
Based on the evaluation, **Random Forest** was chosen .

Code:

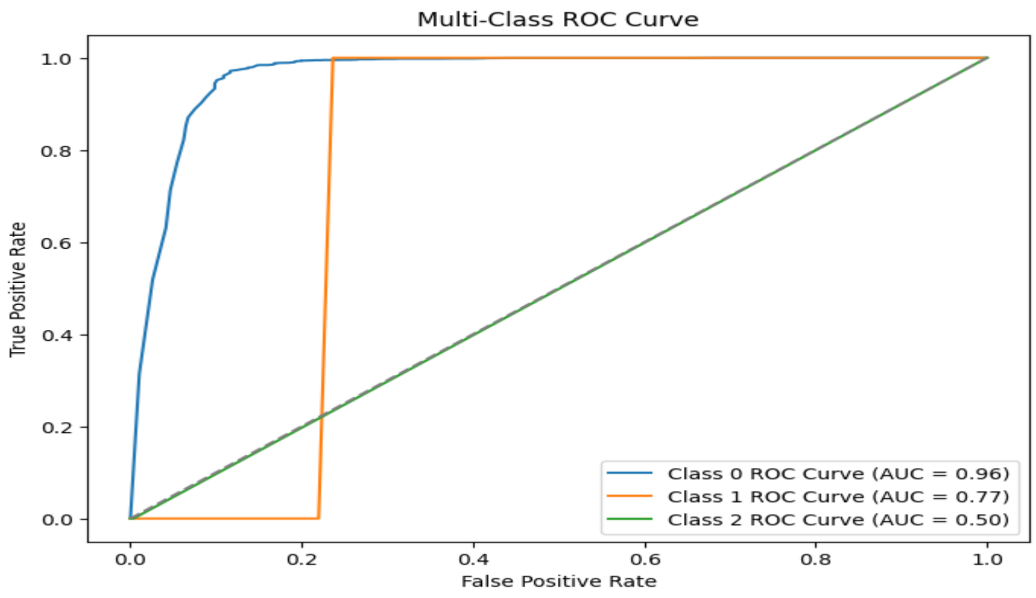


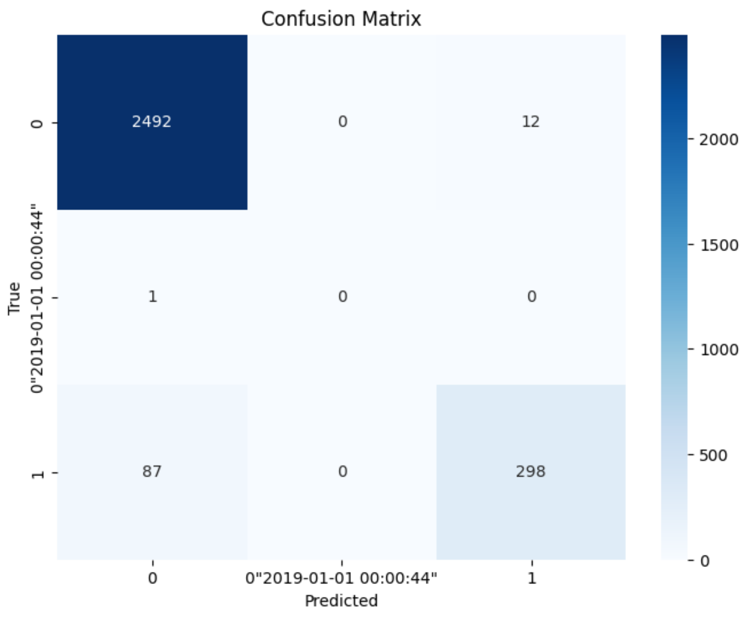
```python
#Model training

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

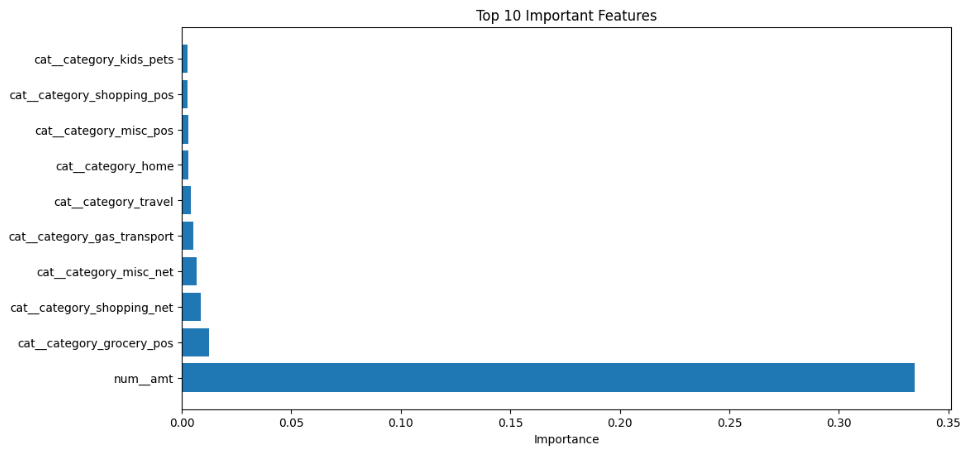
```python
[11] # Evaluate the model performance
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("AUC-ROC Score:")
print(roc_auc_score(y_test, model.predict_proba(X_test), multi_class='ovr', average='weighted'))
```


https://colab.research.google.com/drive/1m18AXG1oUkLQGEAom5lIW6lzMlPh7BQl?authuser=1#scrollTo=drRrlu4MBtlO

```
Classification Report:
                       precision    recall  f1-score   support

                   0       0.97      1.00      0.98      2504
0"2019-01-01 00:00:44"       0.00      0.00      0.00         1
                   1       0.96      0.77      0.86       385

            accuracy                           0.97      2890
           macro avg       0.64      0.59      0.61      2890
        weighted avg       0.96      0.97      0.96      2890

Confusion Matrix:
[[2492    0   12]
 [   1    0    0]
 [  87    0  298]]
AUC-ROC Score:
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
0.901569631547222
```







7. Deployment in Streamlit

For the deployment, **Streamlit** was chosen to build an interactive user interface for real-time fraud detection. The app allows users to input transaction details and receive immediate feedback on whether the transaction is fraudulent or safe.

Steps for Deployment:

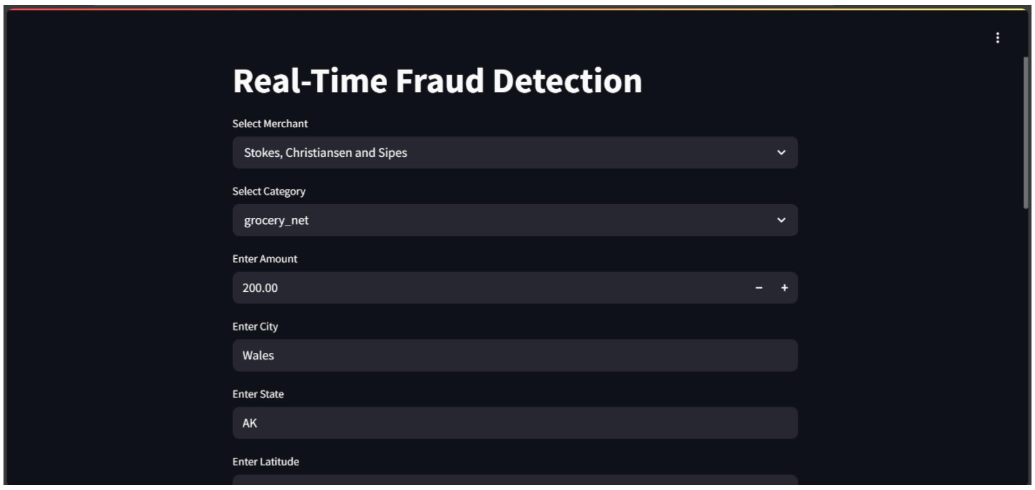
1. **Building the Interface**: Streamlit's simple and intuitive layout allows for easy construction of forms to input transaction data, including:
   - Merchant details
   - Transaction amount
   - User location
   - Device and merchant information
   - Date and time of transaction
2. **Model Integration**: The trained Random Forest model was integrated directly into the Streamlit app. Once a user inputs the transaction details, the model classifies the transaction as either fraudulent (1) or legitimate (0).
3. **Real-Time Prediction**: The app takes the input data, preprocesses it, and passes it to the model for prediction. The result is then displayed to the user in a user-friendly format.
4. **Result Display**: The prediction is shown as either:
   - **Fraudulent** (if the model predicts a fraud)

- **Safe** (if the model predicts a legitimate transaction)

Code:

```
import streamlit as st
import joblib
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder

model = joblib.load('fraud_detection_model.pkl')

label_encoder = LabelEncoder()

def make_prediction(input_data):
    input_data['merchant'] = label_encoder.fit_transform(input_data['merchant'])
    input_data['category'] = label_encoder.fit_transform(input_data['category'])
    input_data['job'] = label_encoder.fit_transform(input_data['job'])
    input_data['city'] = label_encoder.fit_transform(input_data['city'])
    input_data['state'] = label_encoder.fit_transform(input_data['state'])

    input_data = input_data.drop(columns=['dob'])

    expected_features = 15636
    current_features = input_data.shape[1]


    if current_features < expected_features:
        padding = np.zeros((input_data.shape[0], expected_features - current_features))
        input_data_padded = np.hstack([input_data.values, padding])
    else:
        input_data_padded = input_data.values
    prediction = model.predict(input_data_padded)
    return prediction[0]

st.title('Real-Time Fraud Detection')

merchant = st.selectbox('Select Merchant', ['Stokes, Christiansen and Sipes', 'Merchant A',
'Merchant B'])
category = st.selectbox('Select Category', ['grocery_net', 'ecommerce', 'retail'])
amount = st.number_input('Enter Amount', min_value=0.0, step=0.01)
city = st.text_input('Enter City', value='Wales')
state = st.text_input('Enter State', value='AK')
latitude = st.number_input('Enter Latitude', min_value=-90.0, max_value=90.0, step=0.01)
longitude = st.number_input('Enter Longitude', min_value=-180.0, max_value=180.0,
step=0.01)
city_population = st.number_input('Enter City Population', min_value=0, step=1)
job = st.selectbox('Select Job', ['Administrator', 'Engineer', 'Manager', 'Clerk'])
dob = st.date_input('Enter Date of Birth', value=pd.to_datetime('1939-09-11'))
transaction_number = st.text_input('Enter Transaction Number')
merchant_latitude = st.number_input('Enter Merchant Latitude', min_value=-90.0,
max_value=90.0, step=0.01)
merchant_longitude = st.number_input('Enter Merchant Longitude', min_value=-180.0,
max_value=180.0, step=0.01)

input_data = {
    'merchant': [merchant],
    'category': [category],
    'amt': [amount],
    'city': [city],
    'state': [state],
    'lat': [latitude],
    'long': [longitude],
    'city_pop': [city_population],
    'job': [job],
    'dob': [dob],
    'trans_num': [transaction_number],
    'merch_lat': [merchant_latitude],
    'merch_long': [merchant_longitude]
}

input_df = pd.DataFrame(input_data)

if st.button('Predict'):
    try:
        result = make_prediction(input_df)

        if result == 0:
            st.success("Prediction Result: Safe Transaction")
        else:
            st.error("Prediction Result: Fraud Transaction")
    except Exception as e:
        st.error(f"Error: {str(e)}")
```

Output:



Real-Time Fraud Detection

Select Merchant: Stokes, Christiansen and Sipes

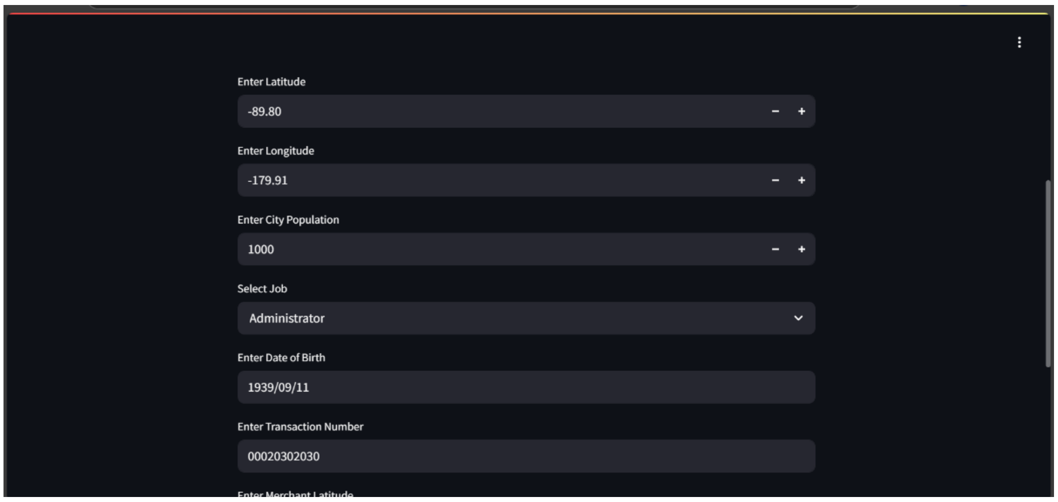
Select Category: grocery_net

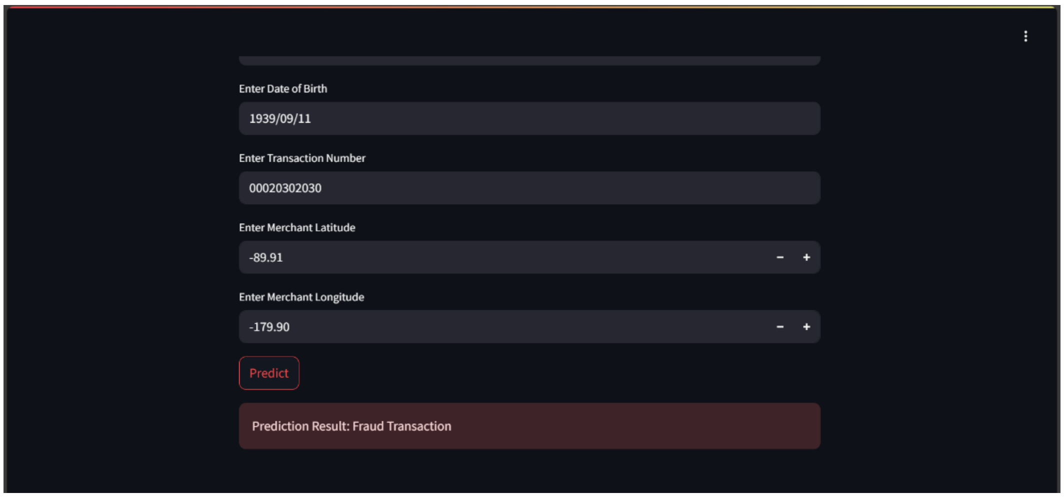
Enter Amount: 200.00

Enter City: Wales

Enter State: AK

Enter Latitude


Enter Latitude
-89.80
Enter Longitude
-179.91
Enter City Population
1000
Select Job
Administrator
Enter Date of Birth
1939/09/11
Enter Transaction Number
00020302030


Enter Date of Birth
1939/09/11
Enter Transaction Number
00020302030
Enter Merchant Latitude
-89.91
Enter Merchant Longitude
-179.90

Prediction Result: Fraud Transaction

8. Conclusion

The project successfully developed and deployed a real-time fraud detection system using **Streamlit**. The model, a **Random Forest classifier**, achieved high accuracy and is capable of providing predictions in real-time based on transaction data. The application is easy to use, allowing users to quickly assess the legitimacy of transactions.

Future Improvements:

- **Advanced Features**: Including network analysis or anomaly detection could further enhance the system's performance.
- **User Interface Enhancements**: Adding more interactive visualizations and alerts for detected fraud could improve the user experience.