Online Payment Fraud Detection

Data Science With Python Lab Project Report

Bachelor

in

Computer Science

By

SUVVARI JAGADEESWARI AND SHAIK TASLIM KOWSAR

S200180

S200824

Under the Guidance of Siva Rama Sastry Sir

Rajiv Gandhi University Of Knowledge And Technologies

S.M. Puram, Srikakulam - 532410

Andhra Pradesh, India

Abstract

Online Payment Fraud Detection

Online payment plays a major role in everyone's life. Over the past few years, there is rise in count of online payments. If we want to buy anything with out hand cash, then we will simply make an online payment. We can send money to anyone at anywhere easily even though they are far from us. It saves our time. Due to rapid increase in online payments as a result online payments fraud has been increased.

We are working on this project to predict *"Fraudulent Payments"*. Our goal is to create a model to prevent frauds in real time by using our dataset. As it ensures security, it is beneficial to both consumers and service providers. We are aiming to get accurate results through the Machine Learning which includes KNN, SVM and Random Forest. In this project we have used a "kaggle" dataset. In this project, we use some of the python libraries such as Pandas, Numpy, Seaborn, Matplotlib, Scikit-learn.

Contents

Chapter 1

Introduction

1.1 Introduction to the Project

Figure 1.1: Online payment fraud detection

Nowadays, every person buys whatever they want from online. As technology develops rapidly, people becomes smart. By the introduction of smart mobiles, every things comes with in our hands. And moreover an online payment platform is just like a digital wallet that allows consumers to make their payments fast. As demand increases for online payments the risk of fraudulent activites rises. The above Fig-1.1 shows that fraudulent activities that have been happening. So our approach is to detect this kind of activities and prevent from being implemented.

1.2 Applications

Online Payment fraud detection has various applications in today's world.

- **E-commerce:**

  Online payment fraud detection helps e-commerce websites to prevent the fraudulent transactions and protect both the merchants and customers.

- **Banking:**

  Banks use online payment fraud detection to identify and stop illegal transactions, protecting their customer's accounts and financial information.

- **Travel Industry:**

  Online payment fraud detection is used by travel agencies and other booking platforms to prevent payment frauds and protect customer's travel plans.

- **Government:**

  Online payment fraud detection keeps government transactions safe, protects money, and boosting public trust and efficiency.

1.3 Motivation towards the project

As more people make online payments, the number of fraud cases increases. However so many members losing their personal information and money to the hackers. At this stage, some people feel really bad and worried when they lose their details and money. This makes it important for us to be careful when using online services. The motivation behind **"online payment fraud detection"** is to catch and stop hackers who tries to steal money or personal information during online transactions. This helps businesses maintain their reputation and consumers can feel confident that their financial information is secure when they buy things in online.

1.4 Problem Statement

The problem is to identifying and preventing fraudulent transcations. There is a need for effective and efficient fraud detection systems that can identify and prevent fraudulent transactions. Online payment plat-form is just like a digital wallet that allows consumers to make their payments fast. By building strong fraud detection tools, consumers can make online shopping safely. This project will be able to meet real-world demands and increase the security of transactions for the customers.

Chapter 2

Approach To The Project

2.1 Explanation Of The Project

This Project is about to detect the online payment fruad. By detecting and preventing fraud, consumers can feel more secure when making payments in online. The aim is to develop a system that can automatically detect and prevent fraudulent transactions. This project can be more benificial to all customers and businesses.

2.2 Data Set

We have obtained an **Online Payment Fraud Detection dataset** from Kaggle, which contains 63,62,620 rows. To simplify the analysis, we took a random sample of 10,000 observations.The dataset provides valuable information about payment frauds of those transactions. The features of the dataset are,

1. **step**: The 'step' column contains an integer representing the number of hours passed since the recording of the dataset began.

2. **type**: Type of online transaction(CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.)

3. **amount**: The amount of the transaction

4. **nameOrig**: Customer starting the transaction

5. **oldbalanceOrg**: Balance before the transaction
6. **newbalanceOrg**: Balance after the transaction
7. **nameDest** Recipient of the transaction
8. **oldbalanceDest**: Initial balance of receipient before the transaction
9. **newbalanceDest**: The new balance of the receipient after the transaction
10. **isFraud**: Fraud transaction
11. **isFlaggedFraud**: It tells us if the system suspects a transaction could be fraud.

2.3 Prediction technique

Our prediction Techniques are Logistic regression, Decision Tree and Random Forest. Logistic regression is a process of modeling the probability of a binary outcome (fraudulent or not) based on input variables. It predicts the likelihood of a transaction being fraudulent by estimating probabilities based on transaction attributes.

Decision trees use a tree-like model of decisions to predict outcomes based on input features. We use Random Forest which can perform well with large datasets and complex relationships. It's easy to understand, can handle different kinds of data.

2.4 Graphs

Below are some graphs that describes the project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
```

1.BarGraph

```
#Bar Graph
plt.figure(figsize=(10, 6))
sns.countplot(x='type',data=df)
plt.title('Number of Transactions by Each Transaction Type')
plt.xlabel('Transaction Type')
plt.ylabel('Number of Transactions')
plt.show()
```

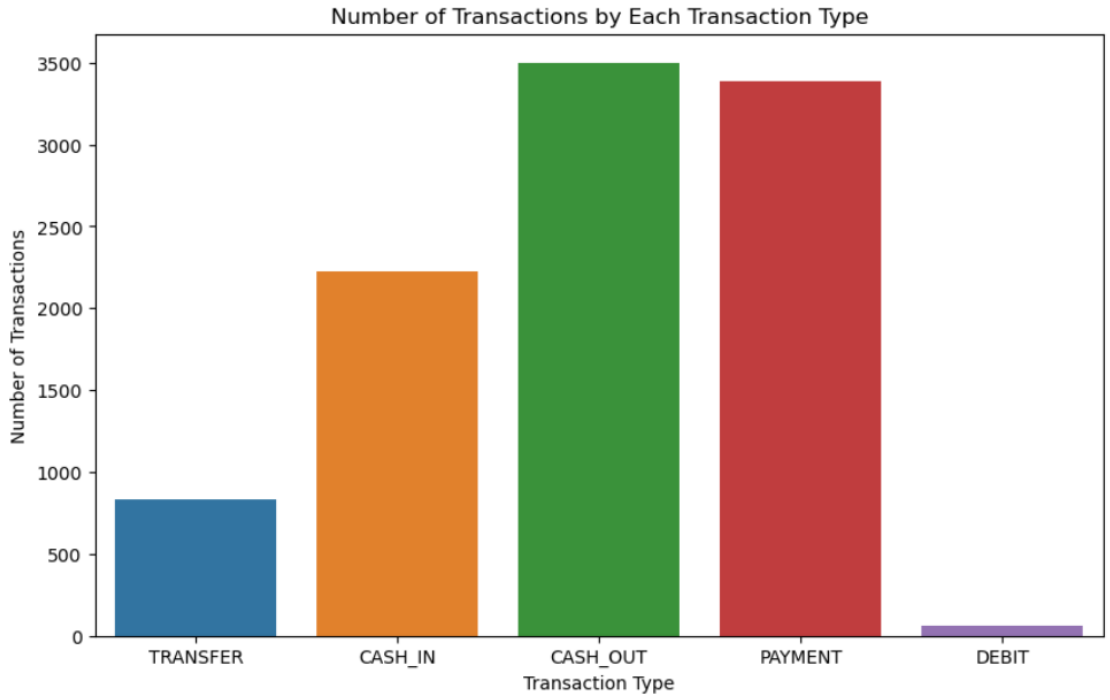

Figure 2.1: Number of Transactions by Each Transaction Type

from the above graph we can say that more money has been transacted mostly through cash out followed by payment type.

2.Histogram:

```
#Histogram
fraud_amount= df[df.isFraud==1]
fraud_amount=fraud_amount.sort_values(by=['amount'],ascending=False)
fraud_amount.amount.plot(kind='hist', bins=20, figsize=(12,6),
facecolor='orange',edgecolor='black')
plt.title('Distribution of Fraudulent Transaction Amounts')
plt.xlabel('Transaction Amount')
plt.ylabel('Frequency')
plt.show()
```

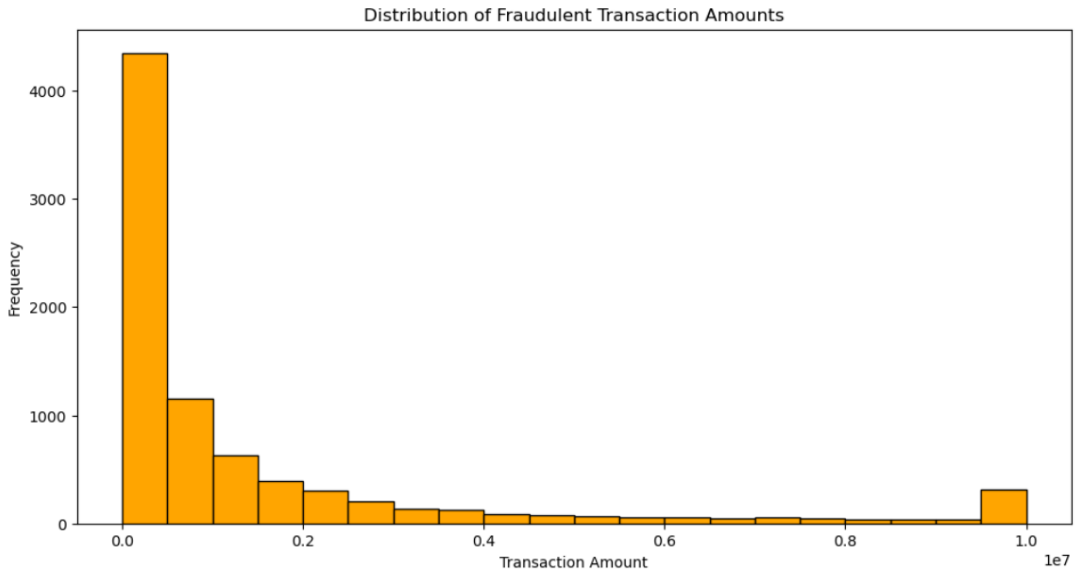

Figure 2.2: Distribution of Fraudulent Transaction Amounts

2.5 Visualization

1.Pie Chart:

```
# Pie chart
transaction_amounts = df.groupby('type')['amount'].sum()
plt.figure(figsize=(10, 6))
plt.pie(transaction_amounts,autopct='%1.1f%%',startangle=120)
plt.title('Total Transaction Amount by Each Transaction Type')
plt.show()
```

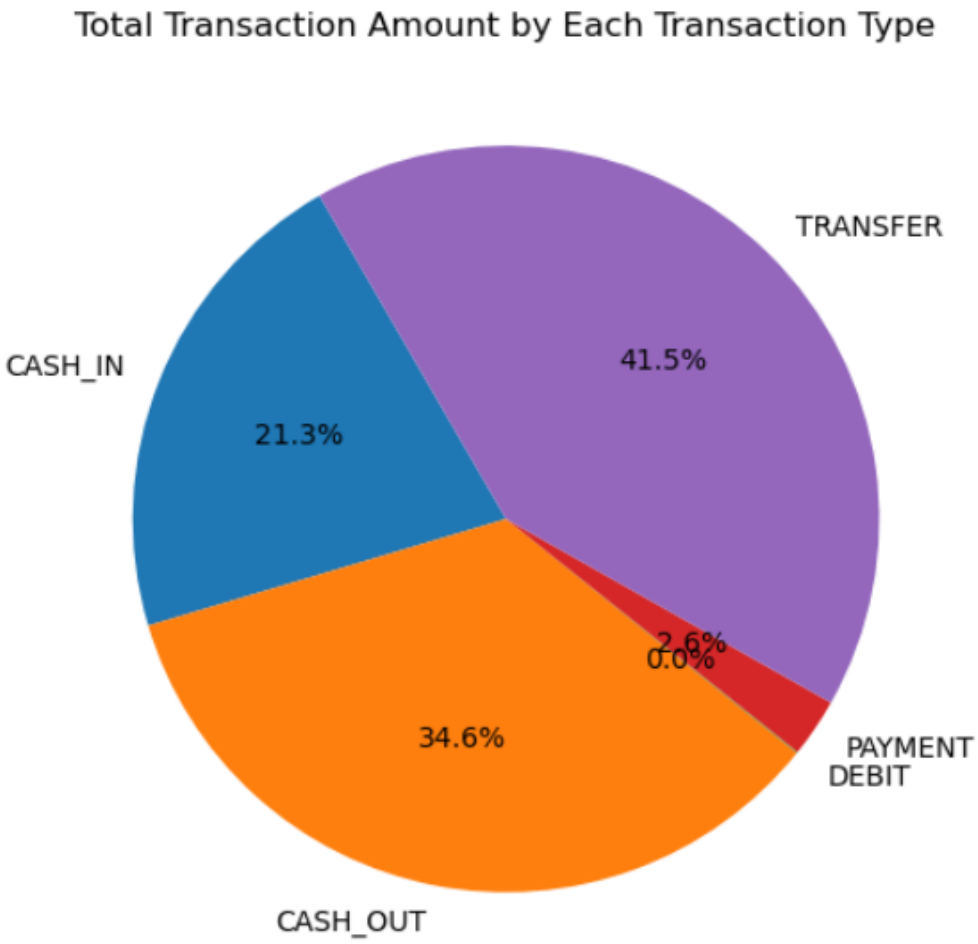

Figure 2.3: Total Transaction Amount by Each Transaction Type

2.Box Plot:

```
# Box Plot
df.boxplot(column='step', by='isFraud')
plt.title('Transaction Steps by Fraud Status')
plt.suptitle('')  # Suppress the default Boxplot grouped by title
plt.xlabel('Fraud Status (0: Non-Fraud, 1: Fraud)')
plt.ylabel('Step')
plt.show()
```

Figure 2.4: Transaction Steps by Fraud Status

3.Line Plot:

```
# Line Plot
plt.plot(df['step'], df['amount'], linestyle='-')
plt.xlabel('Time (Step)')
plt.ylabel('Transaction Amount')
plt.title('Trend of Transaction Amounts Over Time')
plt.show()
```

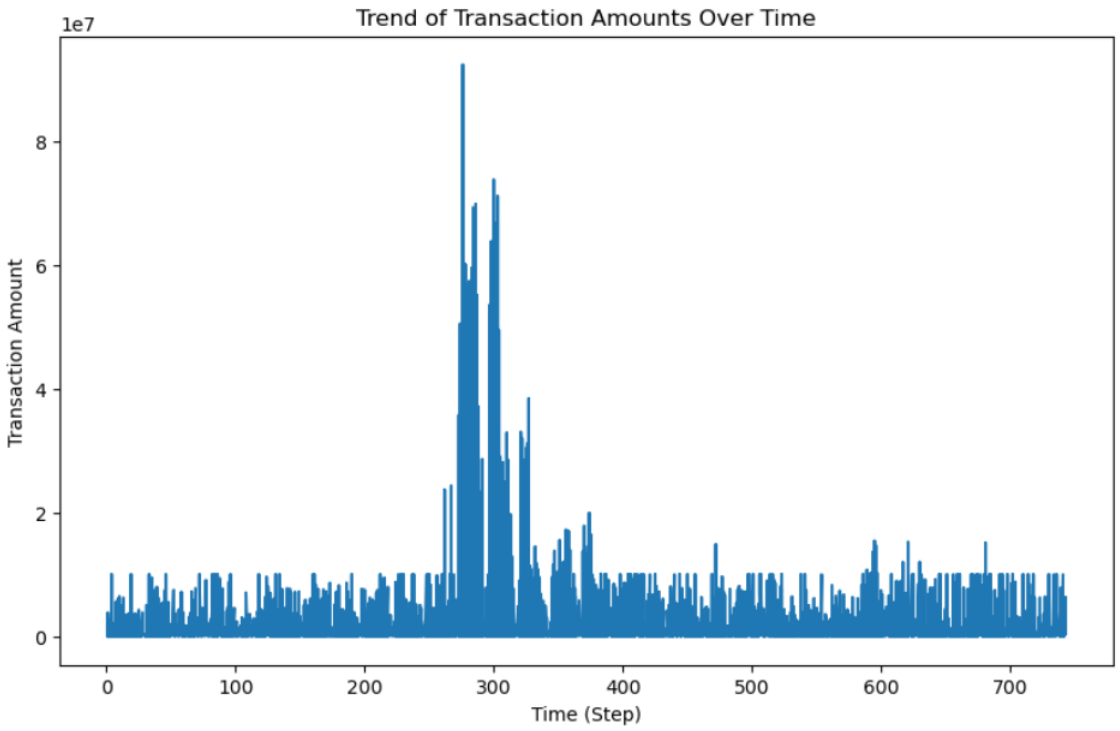

Figure 2.5: Trend of Transaction Amounts Over Time

The above graph indicates that a step value of approximately 300 corresponds to the maximum amount transferred to a recipient.

4.ScatterPlot:

```
# Scatter Plot
plt.scatter(x='type',y='isFraud',data=df)
plt.xlabel('type')
plt.ylabel('fraud_transaction_label')
plt.title('Fraudulent transactions occurs on each type')
plt.figure(figsize=(10, 6))
plt.show()
```

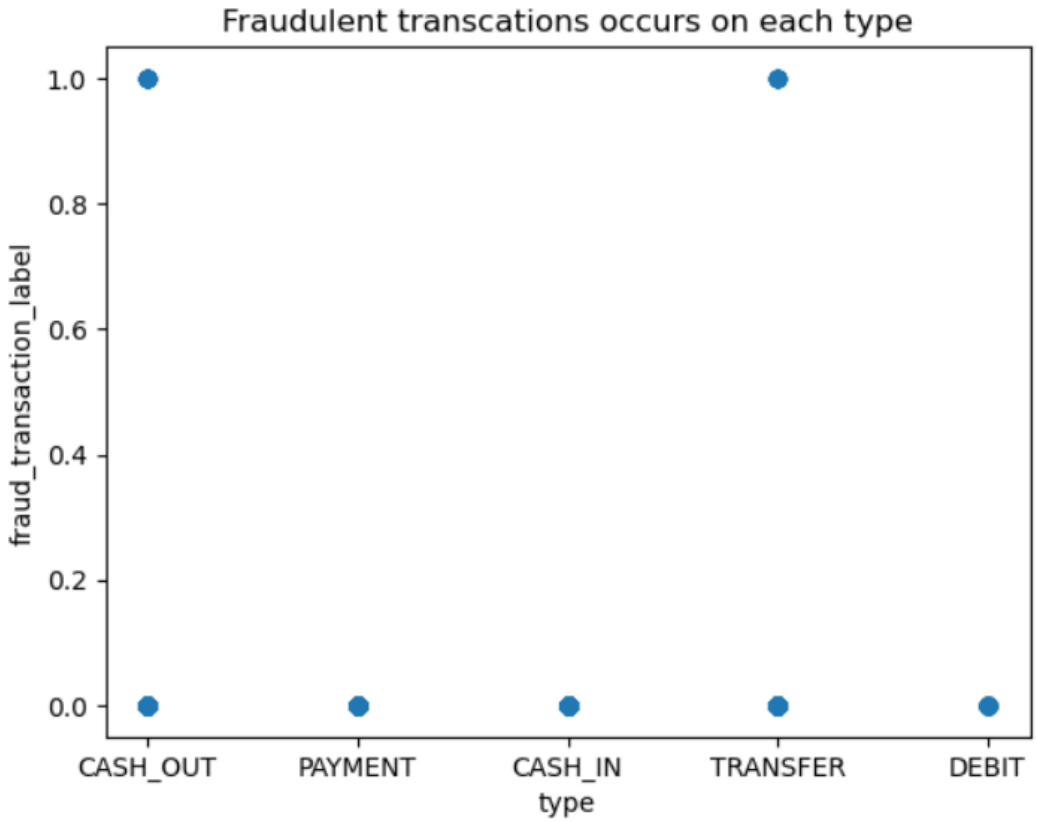

Figure 2.6: Fraudulent transactions occurs on each type

This Visualization shows that fraudulent transactions only occur in transfer and cashout types.

5.BarhGraph:

```
# Barh Graph
fraudster= df.nameDest.value_counts()
fraudster[:10].plot(kind='barh')
plt.title('Top 10 Recipients of Fraudulent Transactions')
plt.xlabel('Number of Transactions')
plt.ylabel('Recipient Account')
plt.figure(figsize=(10, 6))
plt.show()
```

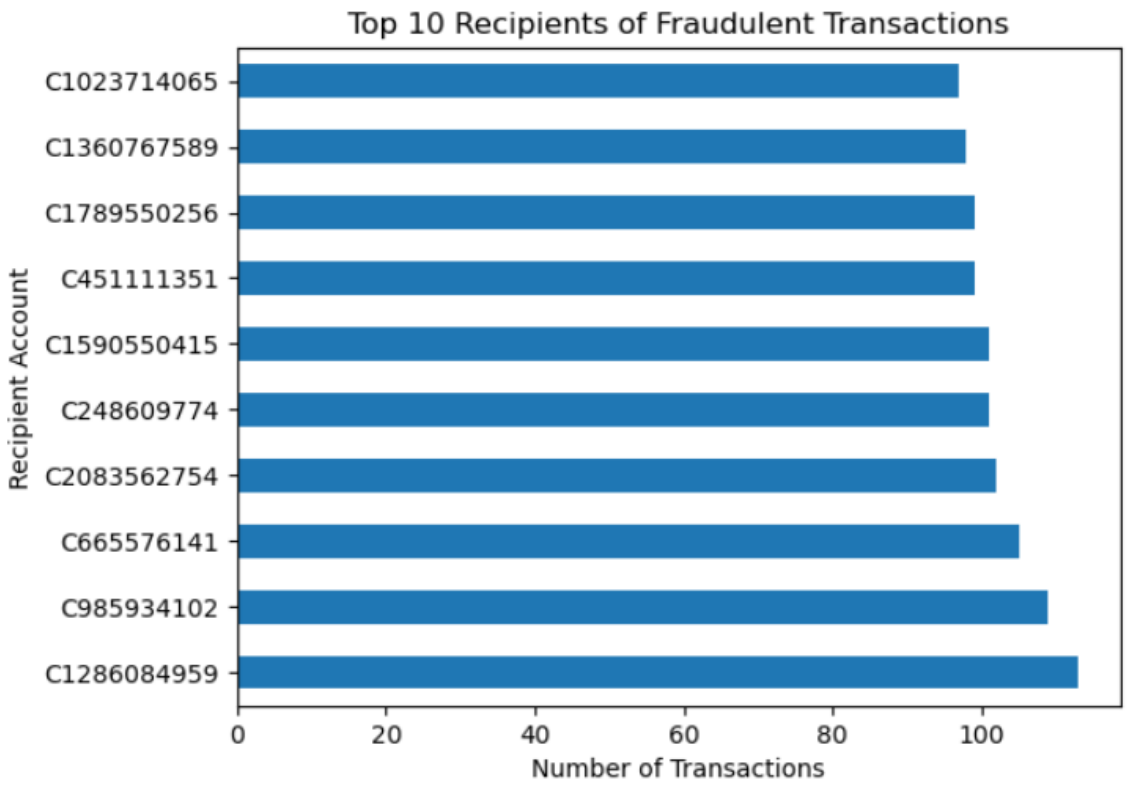

Figure 2.7: Top 10 Recipients of Fraudulent Transactions

6.AreaPlot:

```
# Area Plot
# Group the data by the 'step' column and count the number of
transactions for each step
transactions_count = df.groupby('step').size()
plt.figure(figsize=(10, 6))
transactions_count.plot.area()
plt.title('Transaction Count Over Time')
plt.xlabel('Time Step')
plt.ylabel('Transaction Count')
plt.grid(True)
plt.show()
```

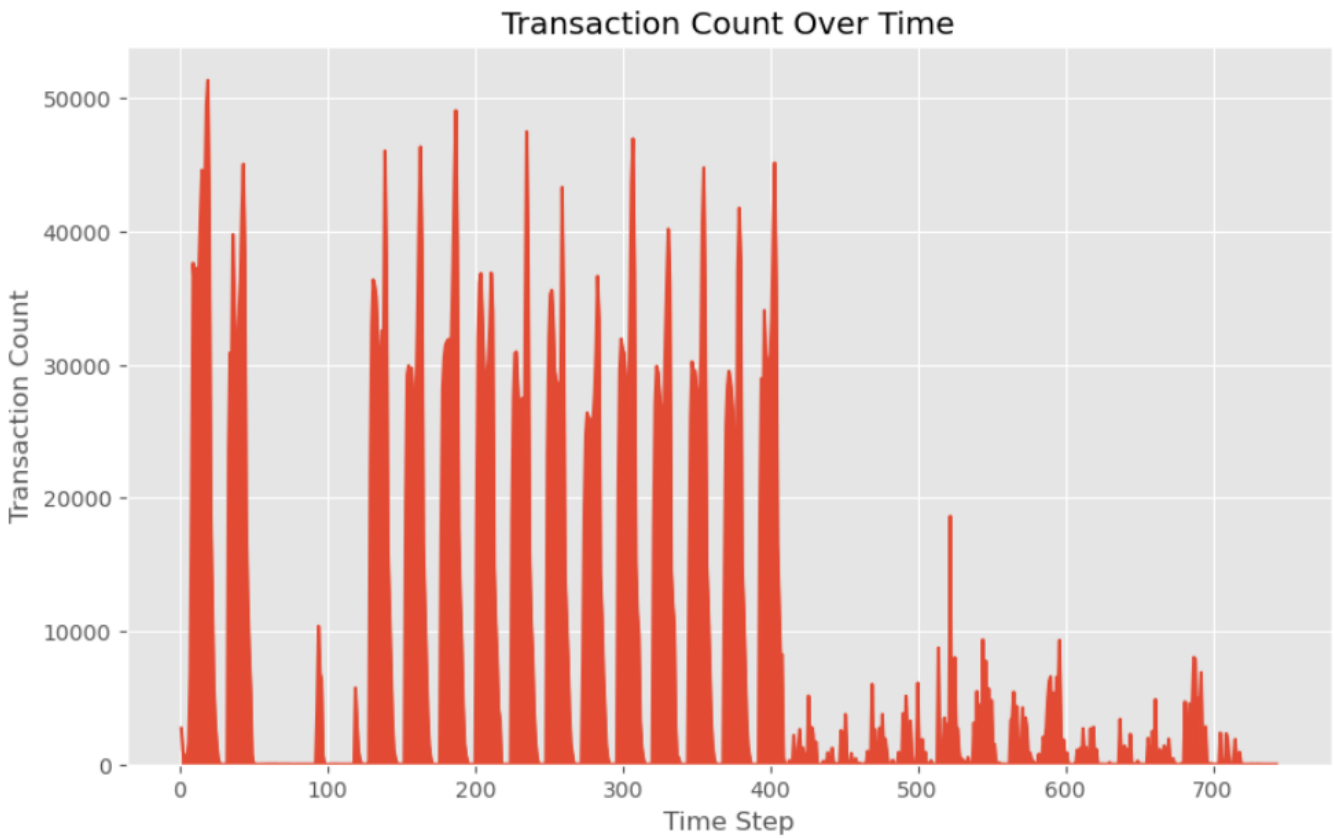

Figure 2.8: Transaction Count Over Time

This Visualization indicates higher transaction numbers during the initial 50 time steps followed by another time steps 100 to 400.

7.Word Cloud:

```
# Word Cloud
from wordcloud import WordCloud
unique_words =  set(df.columns.tolist()) | set(df['type'])
text_data = ' '.join(unique_words)
wordcloud = WordCloud(width=800, height=400).generate(text_data)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud for Online Payment fraud Detection')
plt.show()
```

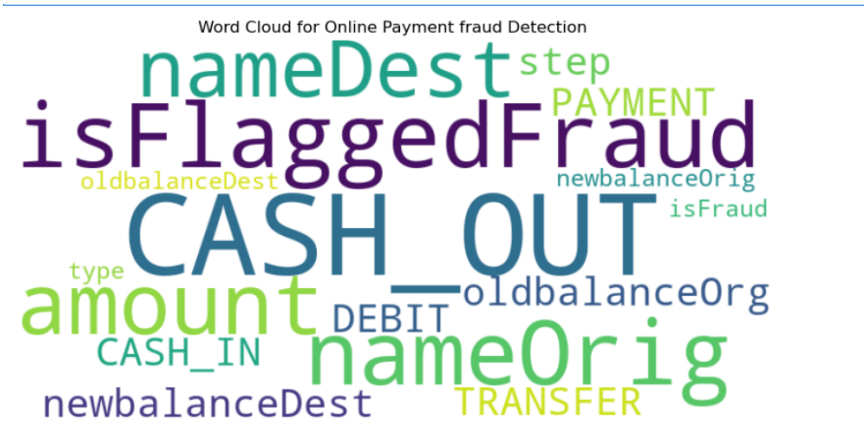

Figure 2.9: Word Cloud

8.Histogram:

```
# Histogrsm
sns.histplot(df['step'],bins=50,color='orange',edgecolor='black')
plt.title('Distribution of Transaction Steps')
plt.xlabel('Transaction Step')
plt.ylabel('Density')
plt.figure(figsize=(15, 6))
plt.show()
```

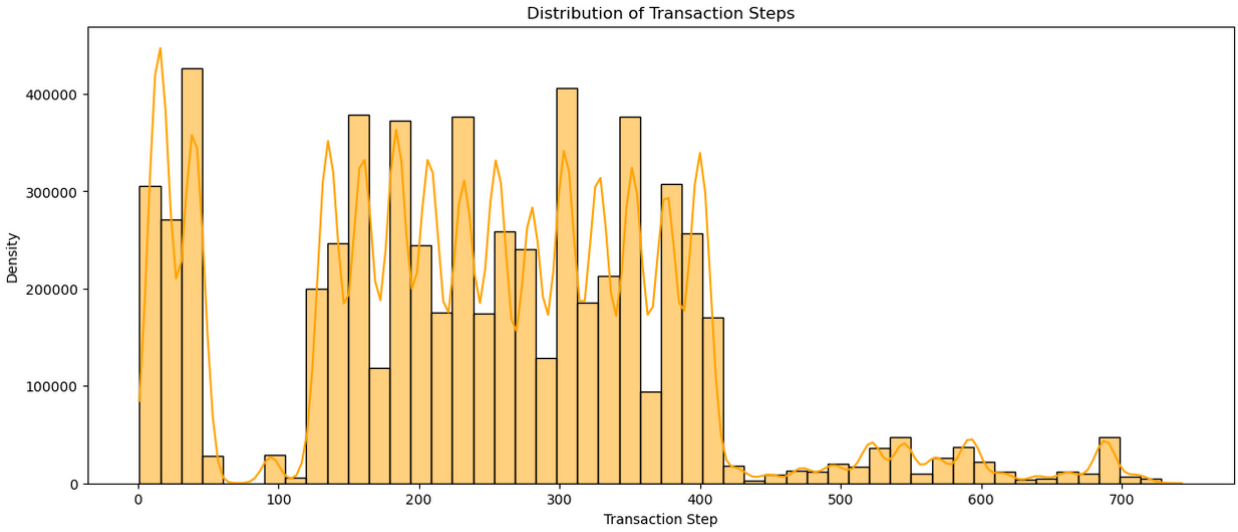

Figure 2.10: Distribution of Transaction Steps

This Visualization shows that there is a maximum distribution of 150 to 400 of step.

9.Heat Map :

```
# Plot the correlation heatmap
numeric_df = df.select_dtypes(include='number')
correlation = numeric_df.corr()
sns.heatmap(correlation, annot=True, cmap="Blues")
plt.title('Correlation Between Numeric Features')
plt.figure(figsize=(8, 6))
plt.show()
```

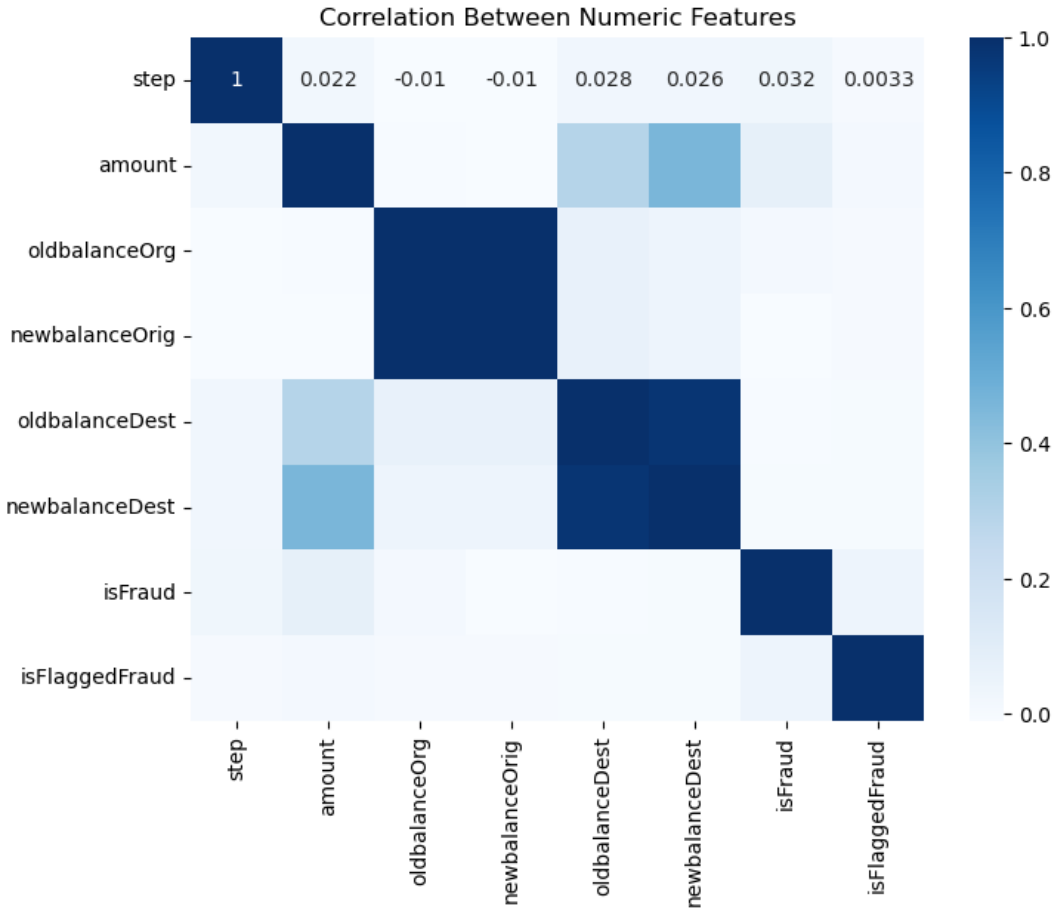

Figure 2.11: Heat Map

Chapter 3

Code

3.1 Explain Our Code With Outputs

3.1.1 Data Understanding:

```
df1 = pd.read_csv('onlinefraud.csv')
```

```
df1
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0	0
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065	0.00	0.00	1	0
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010	21182.00	0.00	1	0
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0	0
...	...	...	...	...	...	...	...	...	...	...	...
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776919290	0.00	339682.13	1	0
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881841831	0.00	0.00	1	0
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365125890	68488.84	6379898.11	1	0
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080388513	0.00	0.00	1	0
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873221189	6510099.11	7360101.63	1	0

6362620 rows × 11 columns

Figure 3.1: The dataset consists of 63,62,620 observations.

The dataset has 63,62,620 observations. To make it easier to work with, we shuffle the data and use only 10,000 observations.

```
# Shuffle the Data
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
# Select the first 10,000 rows
df = df.head(10000)
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	237	TRANSFER	35911.20	C1349435850	301885.00	265973.80	C2059262944	567731.26	603642.46	0	0
1	157	CASH_IN	34021.33	C2079545670	4347317.94	4381339.28	C1052878928	544497.20	510475.87	0	0
2	324	CASH_IN	89355.68	C369322363	782.00	90137.68	C397644820	133043.62	43687.94	0	0
3	11	CASH_OUT	27697.40	C2089692083	48.00	0.00	C286738540	674135.52	701832.91	0	0
4	254	CASH_OUT	389.27	C1262215312	13489.20	13099.93	C1827377386	202995.39	203384.66	0	0
...	...	...	...	...	...	...	...	...	...	...	...
9995	276	CASH_OUT	156619.25	C384169490	3076.00	0.00	C1123686650	0.00	156619.25	0	0
9996	254	TRANSFER	271659.67	C501533000	0.00	0.00	C1348815716	2988867.17	3260526.84	0	0
9997	308	PAYMENT	22663.44	C986403623	0.00	0.00	M1209482928	0.00	0.00	0	0
9998	43	CASH_IN	113962.43	C1762015941	20121.00	134083.43	C1389074037	28817.42	0.00	0	0
9999	138	CASH_OUT	160757.90	C435727829	0.00	0.00	C2137390232	387094.55	547852.45	0	0

10000 rows × 11 columns

Figure 3.2: Shuffled data

```
# View data (to give you first five rows)
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	237	TRANSFER	35911.20	C1349435850	301885.00	265973.80	C2059262944	567731.26	603642.46	0	0
1	157	CASH_IN	34021.33	C2079545670	4347317.94	4381339.28	C1052878928	544497.20	510475.87	0	0
2	324	CASH_IN	89355.68	C369322363	782.00	90137.68	C397644820	133043.62	43687.94	0	0
3	11	CASH_OUT	27697.40	C2089692083	48.00	0.00	C286738540	674135.52	701832.91	0	0
4	254	CASH_OUT	389.27	C1262215312	13489.20	13099.93	C1827377386	202995.39	203384.66	0	0

Figure 3.3: head

```
# View data (to give you last five rows)
df.tail()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
9995	276	CASH_OUT	156619.25	C384169490	3076.0	0.00	C1123686650	0.00	156619.25	0	0
9996	254	TRANSFER	271659.67	C501533000	0.0	0.00	C1348815716	2988867.17	3260526.84	0	0
9997	308	PAYMENT	22663.44	C986403623	0.0	0.00	M1209482928	0.00	0.00	0	0
9998	43	CASH_IN	113962.43	C1762015941	20121.0	134083.43	C1389074037	28817.42	0.00	0	0
9999	138	CASH_OUT	160757.90	C435727829	0.0	0.00	C2137390232	387094.55	547852.45	0	0

Figure 3.4: tail

```
# statistical analysis of the data
df.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	10000.000000	1.000000e+04	1.000000e+04	1.000000e+04	1.000000e+04	1.000000e+04	10000.00000	10000.0000
mean	243.990100	1.737107e+05	9.050636e+05	9.272651e+05	1.063047e+06	1.184138e+06	0.00190	0.0001
std	141.704463	4.662073e+05	3.086712e+06	3.122753e+06	2.840653e+06	3.034161e+06	0.04355	0.0100
min	1.000000	1.260000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00000	0.0000
25%	156.000000	1.341284e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00000	0.0000
50%	249.000000	7.420973e+04	1.409942e+04	0.000000e+00	1.345331e+05	2.118248e+05	0.00000	0.0000
75%	345.250000	2.058379e+05	1.088195e+05	1.573190e+05	9.483795e+05	1.114265e+06	0.00000	0.0000
max	717.000000	1.727101e+07	3.359321e+07	3.374855e+07	9.234964e+07	9.261398e+07	1.00000	1.0000

Figure 3.5: describe

```
#Data Verification
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   step            10000 non-null  int64
 1   type            10000 non-null  object
 2   amount          10000 non-null  float64
 3   nameOrig        10000 non-null  object
 4   oldbalanceOrg   10000 non-null  float64
 5   newbalanceOrig  10000 non-null  float64
 6   nameDest        10000 non-null  object
 7   oldbalanceDest  10000 non-null  float64
 8   newbalanceDest  10000 non-null  float64
 9   isFraud         10000 non-null  int64
 10  isFlaggedFraud  10000 non-null  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 859.5+ KB
```

Figure 3.6: info

```
# Columns of the dataset

df.columns
```

```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')
```

Figure 3.7: columns

```
# Shape of the data set

df.shape
```

```
(10000, 11)
```

Figure 3.8: shape

3.1.2 Data Preprocessing:

```
#  Checking how many missing values each column contains

df.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Figure 3.9: Checking missing values.

There are no duplicate values.

```
df.isFraud.value_counts()
```

```
isFraud
0    9981
1      19
Name: count, dtype: int64
```

Figure 3.10: Fraud counting

```
fraud_percentage = (8213/6362620)*100
nonfraud_percentage = (6354407/6362620)*100
print(f"""We can see that this dataset is very imbalanced with fraud
transaction only {fraud_percentage:.1f}% and non-fraud transaction {
nonfraud_percentage:.1f}%, This is expected of this kind of dataset
because there is no way majority of our transaction are to be fraud
else there is something wrong with the system.""")
```

```
We can see that this dataset is very imbalanced with fraud transaction only 0.1% and
    non-fraud transaction 99.9%, This is expected of this kind of dataset because there is
    no way majority of our transaction are to be fraud else there is something wrong with the system.
```

Figure 3.11: fraud vs non-fraud percentage

```
datapayment = df.loc[df['type'] == 'PAYMENT']
datapayment = pd.DataFrame(datapayment)
datapayment.head(10)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
5	234	PAYMENT	2398.16	C229923928	0.00	0.00	M482244351	0.0	0.0	0	0
7	402	PAYMENT	18367.37	C219164201	21187.00	2819.63	M936536987	0.0	0.0	0	0
10	182	PAYMENT	2588.54	C1173615080	41547.00	38958.46	M531725823	0.0	0.0	0	0
14	142	PAYMENT	8280.95	C1578120005	103981.13	95700.18	M1598671677	0.0	0.0	0	0
16	303	PAYMENT	40760.25	C1913346275	31147.00	0.00	M1005627161	0.0	0.0	0	0
22	393	PAYMENT	50737.82	C23552834	0.00	0.00	M1105201056	0.0	0.0	0	0
23	142	PAYMENT	4618.51	C10873201	2523.00	0.00	M2018218703	0.0	0.0	0	0
25	227	PAYMENT	2406.93	C2141398877	0.00	0.00	M166963975	0.0	0.0	0	0
28	209	PAYMENT	10534.61	C615389868	102512.00	91977.40	M426667936	0.0	0.0	0	0
29	140	PAYMENT	7920.52	C773890759	43558.00	35637.48	M2011283784	0.0	0.0	0	0

Figure 3.12: payment

Defining a dataset containing only payment type of transactions

3.1.3 Feature Engineering:

Creating a new features errorBalanceOrig and errorBalanceDest as follows:

errorBalanceOrig = newbalanceOrig + amount - oldbalanceOrg

errorBalanceDest = newbalanceDest + amount - oldbalanceDest

These features help to identify errors in transaction balances, improving fraud detection.

```
df['errorBalanceOrig'] = df.newbalanceOrig + df.amount - df.
oldbalanceOrg
df['errorBalanceDest'] = df.oldbalanceDest + df.amount - df.
newbalanceDest
df.head(5)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud	errorBalanceOrig
0	237	TRANSFER	35911.20	C1349435850	301885.00	265973.80	C2059262944	567731.26	603642.46	0	0	0.00
1	157	CASH_IN	34021.33	C2079545670	4347317.94	4381339.28	C1052878928	544497.20	510475.87	0	0	68042.67
2	324	CASH_IN	89355.68	C369322363	782.00	90137.68	C397644820	133043.62	43687.94	0	0	178711.36
3	11	CASH_OUT	27697.40	C2089692083	48.00	0.00	C286738540	674135.52	701832.91	0	0	27649.40
4	254	CASH_OUT	389.27	C1262215312	13489.20	13099.93	C1827377386	202995.39	203384.66	0	0	0.00

Figure 3.13: Error balance

3.1.4 Model Selection:

Here we stored the features which are used to predict in variable 'x'. Also we stored the target variable in 'y'. We split the data into to parts test data and train data by using train_test_split() function of

scikit-learn.

```
from sklearn.model_selection import train_test_split
X = df[['type','amount','oldbalanceOrg','newbalanceOrig']]
y = df['isFraud']
y
```

```
0       0
1       0
2       0
3       0
4       0
       ..
9995    0
9996    0
9997    0
9998    0
9999    0
Name: isFraud, Length: 10000, dtype: int64
```

Figure 3.14: Error balance

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,
random_state=101)
X_train.shape
```

```
(8000, 4)
```

```
X_test.shape
```

```
(2000, 4)
```

```
# Importing the Logistic Regression machine learning library
from sklearn.linear_model import LogisticRegression
# Creating a Logistic Regression instance
lr_model = LogisticRegression(max_iter=500)
# Training the Logistic Regression model on our test data
lr_model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=500)
```

```
# Importing the Random Forest machine learning library
from sklearn.ensemble import RandomForestClassifier
# Creating a Random Forest instance
rf_model = RandomForestClassifier()
# Training the Random Forest model on our test data
rf_model.fit(X_train, y_train.values.ravel())
```

```
RandomForestClassifier
RandomForestClassifier()
```

```
# spliting data into train_test_split
x=df.loc[:, df.columns!='isFraud']
y=df['isFraud']
x_train, x_test, y_train, y_test=train_test_split(x,y, test_size=0.2,
stratify=df['isFraud'], random_state=1)
print(len(df),len(x_train),len(y_test))
```

```
10000 8000 2000
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

```
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import OneHotEncoder
X = df.drop(['isFraud', 'nameOrig', 'nameDest'], axis='columns')
y = df['isFraud']

# One-Hot Encoding of the 'type' column
encoder = OneHotEncoder()
X_encoded = encoder.fit_transform(X[['type']]).toarray()
X_encoded_df = pd.DataFrame(X_encoded, columns=encoder.
get_feature_names_out(['type']))

# Combine encoded columns with original data
X = X.join(X_encoded_df).drop(['type'], axis=1)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.4, random_state=42, stratify=y)

# Oversample the minority class using SMOTE
smote = SMOTE(random_state=42, sampling_strategy='not majority')
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Initialize the models
dtc = DecisionTreeClassifier(random_state=42)
rfc = RandomForestClassifier(random_state=42)
knn = KNeighborsClassifier(n_neighbors=3)  # Reduced number of
neighbors
lr_model = LogisticRegression(random_state=42)

# Train the models
dtc.fit(X_train_smote, y_train_smote)
rfc.fit(X_train_smote, y_train_smote)
knn.fit(X_train_smote, y_train_smote)
lr_model.fit(X_train_smote, y_train_smote)
```

```
# Test the models
y_pred_rfc = rfc.predict(X_test)
y_pred_knn = knn.predict(X_test)
y_pred_lr = lr_model.predict(X_test)
y_pred_dt = model.predict(X_test)

# Print the accuracy scores and classification reports for each model
print("Decision Tree Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_dt)*100)
print(classification_report(y_test, y_pred_dt))
print(confusion_matrix(y_test, y_pred_dt))

print("Random Forest Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_rfc)*100)
print(classification_report(y_test, y_pred_rfc))
print(confusion_matrix(y_test, y_pred_rfc))
print()

print("K-Nearest Neighbors Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn)*100)
print(classification_report(y_test, y_pred_knn))
print(confusion_matrix(y_test, y_pred_knn))

print("Logistic Regression Results:")
print("Accuracy:", accuracy_score(y_test, y_pred_lr)*100)
# print("Recall:", recall_score(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
print(confusion_matrix(y_test, y_pred_lr))
print()
```

```
Decision Tree Results:
Accuracy: 99.52499999999999
              precision    recall  f1-score   support

       Fraud       0.24      0.62      0.34         8
    No Fraud       1.00      1.00      1.00      3992

    accuracy                           1.00      4000
   macro avg       0.62      0.81      0.67      4000
weighted avg       1.00      1.00      1.00      4000

[[   5    3]
 [  16 3976]]
Random Forest Results:
Accuracy: 99.65
              precision    recall  f1-score   support

       Fraud       0.31      0.62      0.42         8
    No Fraud       1.00      1.00      1.00      3992

    accuracy                           1.00      4000
   macro avg       0.66      0.81      0.71      4000
weighted avg       1.00      1.00      1.00      4000

[[   5    3]
 [  11 3981]]

K-Nearest Neighbors Results:
Accuracy: 98.825
              precision    recall  f1-score   support

       Fraud       0.12      0.75      0.20         8
    No Fraud       1.00      0.99      0.99      3992

    accuracy                           0.99      4000
   macro avg       0.56      0.87      0.60      4000
weighted avg       1.00      0.99      0.99      4000

[[   6    2]
 [  45 3947]]
Logistic Regression Results:
Accuracy: 93.15
              precision    recall  f1-score   support

       Fraud       0.03      0.88      0.05         8
    No Fraud       1.00      0.93      0.96      3992

    accuracy                           0.93      4000
   macro avg       0.51      0.90      0.51      4000
weighted avg       1.00      0.93      0.96      4000

[[   7    1]
 [ 273 3719]]
```

Chapter 4

Conclusion and Future Work

4.1 Conclusion:

In conclusion, our project aimed to predict whether a payment is fraudulent or not by using machine learning models like Decision Tree, Logistic Regression and Random Forest Classifier. This model is essential for protecting financial transactions and preventing losses. This project helps in demonstrating the use of machine learning in real world in Financial sectors, Banks, Funds etc. **Random Forest Classifier** is preferred for this task because it offers higher accuracy (99.65) compared to Decision Tree, K-Nearest Neighbour, and Logistic Regression. Its effectiveness in handling complex patterns makes it a reliable choice for identifying the fraudulent activities.