Credit Card Fraud Detection

Google Drive Import

In [124…
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
e.mount("/content/drive", force_remount=True).
```

Step1:Importing Dataset

In [125…
```
import numpy as np
import pandas as pd
from scipy.stats import skew
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
import warnings
warnings.filterwarnings('ignore')
```

In [126…
```
train_data=pd.read_csv('/content/drive/MyDrive/Capstone Project/fraudTrain.csv', en
train_data.head()
```

Out[126]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jen
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Steph
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edv
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	Jer
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	

5 rows × 23 columns

In [127…
```
test_data=pd.read_csv('/content/drive/MyDrive/Capstone Project/fraudTest.csv', enc
test_data.head()
```

Out[127]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	f
0	0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joa
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ash
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	B
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nat

5 rows × 23 columns

Step2:Exploring Data

In [128…
```
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype  
---  ------                 --------------    -----  
 0   Unnamed: 0             1296675 non-null  int64  
 1   trans_date_trans_time  1296675 non-null  object 
 2   cc_num                 1296675 non-null  int64  
 3   merchant               1296675 non-null  object 
 4   category               1296675 non-null  object 
 5   amt                    1296675 non-null  float64
 6   first                  1296675 non-null  object 
 7   last                   1296675 non-null  object 
 8   gender                 1296675 non-null  object 
 9   street                 1296675 non-null  object 
 10  city                   1296675 non-null  object 
 11  state                  1296675 non-null  object 
 12  zip                    1296675 non-null  int64  
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64  
 16  job                    1296675 non-null  object 
 17  dob                    1296675 non-null  object 
 18  trans_num              1296675 non-null  object 
 19  unix_time              1296675 non-null  int64  
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

In [129…

```
test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   Unnamed: 0             555719 non-null  int64  
 1   trans_date_trans_time  555719 non-null  object 
 2   cc_num                 555719 non-null  int64  
 3   merchant               555719 non-null  object 
 4   category               555719 non-null  object 
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object 
 7   last                   555719 non-null  object 
 8   gender                 555719 non-null  object 
 9   street                 555719 non-null  object 
 10  city                   555719 non-null  object 
 11  state                  555719 non-null  object 
 12  zip                    555719 non-null  int64  
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64  
 16  job                    555719 non-null  object 
 17  dob                    555719 non-null  object 
 18  trans_num              555719 non-null  object 
 19  unix_time              555719 non-null  int64  
 20  merch_lat              555719 non-null  float64
 21  merch_long             555719 non-null  float64
 22  is_fraud               555719 non-null  int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 97.5+ MB
```

In [130…
```
train_data.describe()
```

Out[130]:

	Unnamed: 0	cc_num	amt	zip	lat	long	
count	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.2
mean	6.483370e+05	4.171920e+17	7.035104e+01	4.880067e+04	3.853762e+01	-9.022634e+01	8.8
std	3.743180e+05	1.308806e+18	1.603160e+02	2.689322e+04	5.075808e+00	1.375908e+01	3.0
min	0.000000e+00	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.3
25%	3.241685e+05	1.800429e+14	9.650000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.4
50%	6.483370e+05	3.521417e+15	4.752000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.4
75%	9.725055e+05	4.642255e+15	8.314000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.0
max	1.296674e+06	4.992346e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.9

In [131…
```
test_data.describe()
```

Out[131]:

	Unnamed: 0	cc_num	amt	zip	lat	long
count	555719.000000	5.557190e+05	555719.000000	555719.000000	555719.000000	555719.000000
mean	277859.000000	4.178387e+17	69.392810	48842.628015	38.543253	-90.231325
std	160422.401459	1.309837e+18	156.745941	26855.283328	5.061336	13.721780
min	0.000000	6.041621e+10	1.000000	1257.000000	20.027100	-165.672300
25%	138929.500000	1.800429e+14	9.630000	26292.000000	34.668900	-96.798000
50%	277859.000000	3.521417e+15	47.290000	48174.000000	39.371600	-87.476900
75%	416788.500000	4.635331e+15	83.010000	72011.000000	41.894800	-80.175200
max	555718.000000	4.992346e+18	22768.110000	99921.000000	65.689900	-67.950300

In [132…
```
print(train_data.shape)
print(test_data.shape)
```

```
(1296675, 23)
(555719, 23)
```

Step 3:Cleaning Data

In [133…
```
train_data.isnull().sum()
```

Out[133]:
```
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

In [134…
```
test_data.isnull().sum()
```

Out[134]:
```
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

In [135…
```
train_data.duplicated().sum()
test_data.duplicated().sum()
```

Out[135]:
```
0
```

In [136…
```
train_data['is_fraud'].value_counts() #checking for fraud values
```

Out[136]:
```
0    1289169
1       7506
Name: is_fraud, dtype: int64
```

In [137…
```
test_data['is_fraud'].value_counts()
```

Out[137]:
```
0    553574
1      2145
Name: is_fraud, dtype: int64
```

In [138…
```
# Converting date columns to datetime format

train_data['trans_date_trans_time']=pd.to_datetime(train_data['trans_date_trans_tir
train_data['trans_date']=train_data['trans_date_trans_time'].dt.strftime('%Y-%m-%d
train_data['trans_date']=pd.to_datetime(train_data['trans_date'])
train_data['dob']=pd.to_datetime(train_data['dob'])

test_data['trans_date_trans_time']=pd.to_datetime(test_data['trans_date_trans_time
test_data['trans_date']=test_data['trans_date_trans_time'].dt.strftime('%Y-%m-%d')
test_data['trans_date']=pd.to_datetime(test_data['trans_date'])
test_data['dob']=pd.to_datetime(test_data['dob'])
```

In [139…
```
#Drop unnecessary colums
drop_columns=['Unnamed: 0','zip','city_pop','unix_time',"merchant","first","last",'
train_data.drop(columns=drop_columns,inplace=True)
test_data.drop(columns=drop_columns,inplace=True)
```

In [140…
```
# Convert categorical column gender into numerical
train_data.gender=train_data.gender.apply(lambda x: 1 if x=="M" else 0)
test_data.gender=test_data.gender.apply(lambda x: 1 if x=="M" else 0)
```

In [141…
```
train_data.head()
```

Out[141]:

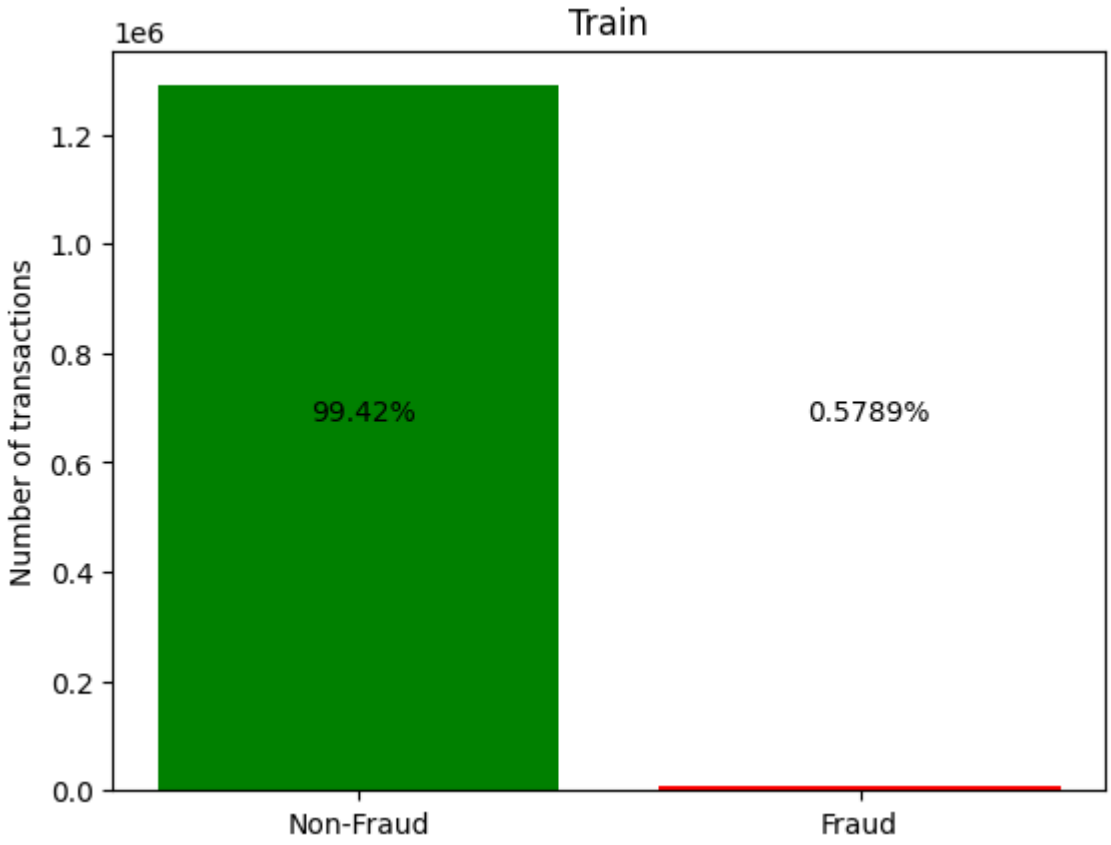
	trans_date_trans_time	cc_num	category	amt	gender	city	state	la
0	2019-01-01 00:00:18	2703186189652095	misc_net	4.97	0	Moravian Falls	NC	36.078
1	2019-01-01 00:00:44	630423337322	grocery_pos	107.23	0	Orient	WA	48.887
2	2019-01-01 00:00:51	38859492057661	entertainment	220.11	1	Malad City	ID	42.180
3	2019-01-01 00:01:16	3534093764340240	gas_transport	45.00	1	Boulder	MT	46.230
4	2019-01-01 00:03:06	375534208663984	misc_pos	41.96	1	Doe Hill	VA	38.420

Step 4: CHECKING CLASS IMBALANCE

In [142…
```
#class imbalance for train data set
classes= train_data['is_fraud'].value_counts()
normal_share=classes[0]/train_data['is_fraud'].count()*100
fraud_share=classes[1]/train_data['is_fraud'].count()*100
```

In [143…
```
#plotting bar graph for more clear understanding
plt.bar(['Non-Fraud','Fraud'], classes, color=['g','r'])
plt.ylabel('Number of transactions')
plt.title('Train')
plt.annotate("{0:.4}%".format(normal_share),(0.2, 0.5), xycoords='axes fraction')
```

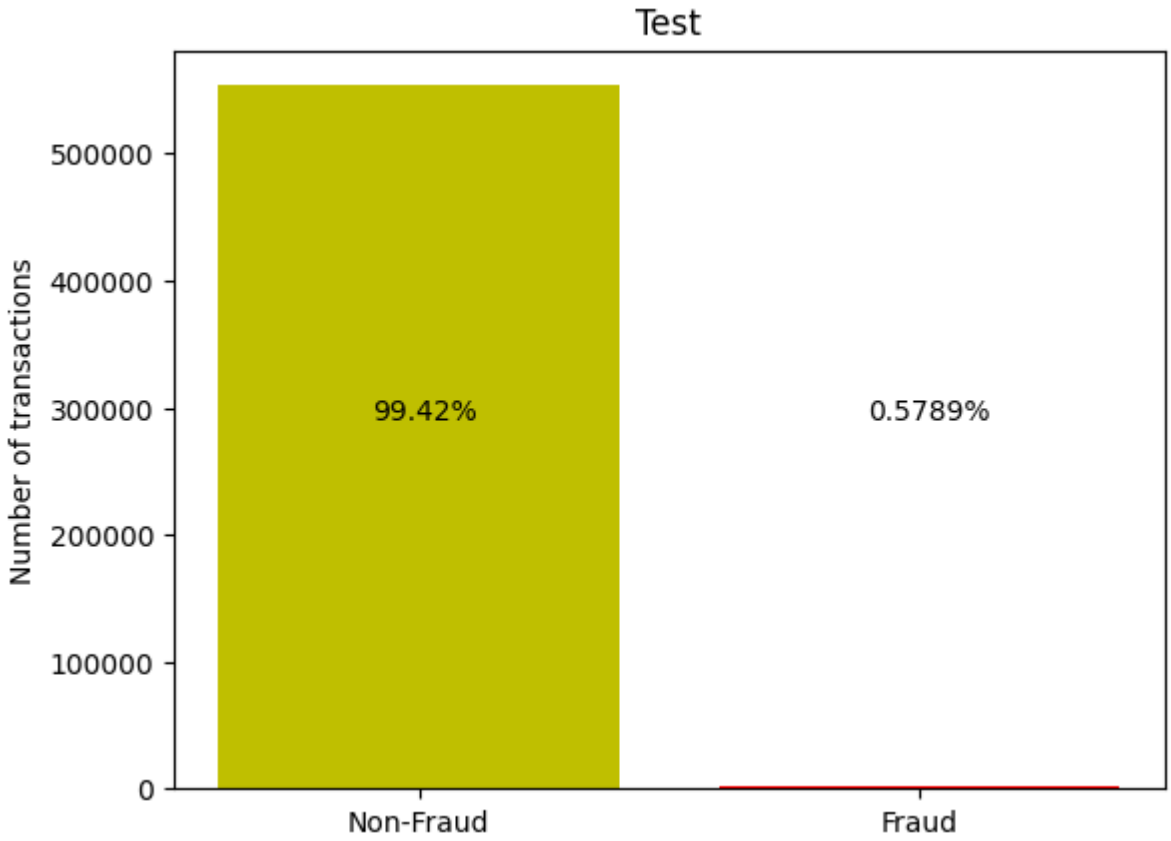
```
plt.annotate("{0:.4}%".format(fraud_share),(0.7, 0.5), xycoords='axes fraction')
plt.show()
```



OBSERVATION: 99.42% of transactions are not fraudulent and 0.58% of transactions are fraudulent for train_data

In [144…

```
#checking class imbalance for test_data set
classes_test=test_data['is_fraud'].value_counts()
normal_share_test=classes_test[0]/test_data['is_fraud'].count()*100
fraud_share_test=classes_test[1]/test_data['is_fraud'].count()*100

#plotting bar graph for more clear understanding
plt.bar(['Non-Fraud','Fraud'], classes_test, color=['y','r'])
plt.ylabel('Number of transactions')
plt.title('Test')
plt.annotate("{0:.4}%".format(normal_share),(0.2, 0.5), xycoords='axes fraction')
plt.annotate("{0:.4}%".format(fraud_share),(0.7, 0.5), xycoords='axes fraction')
plt.show()
```



OBSERVATION: 99.42% of transactions are not fraudulent and 0.58% of transactions are fraudulent for test_data

In [145…
```
train_data['category'].unique() #checking for different categories
```

Out[145]:
```
array(['misc_net', 'grocery_pos', 'entertainment', 'gas_transport',
       'misc_pos', 'grocery_net', 'shopping_net', 'shopping_pos',
       'food_dining', 'personal_care', 'health_fitness', 'travel',
       'kids_pets', 'home'], dtype=object)
```

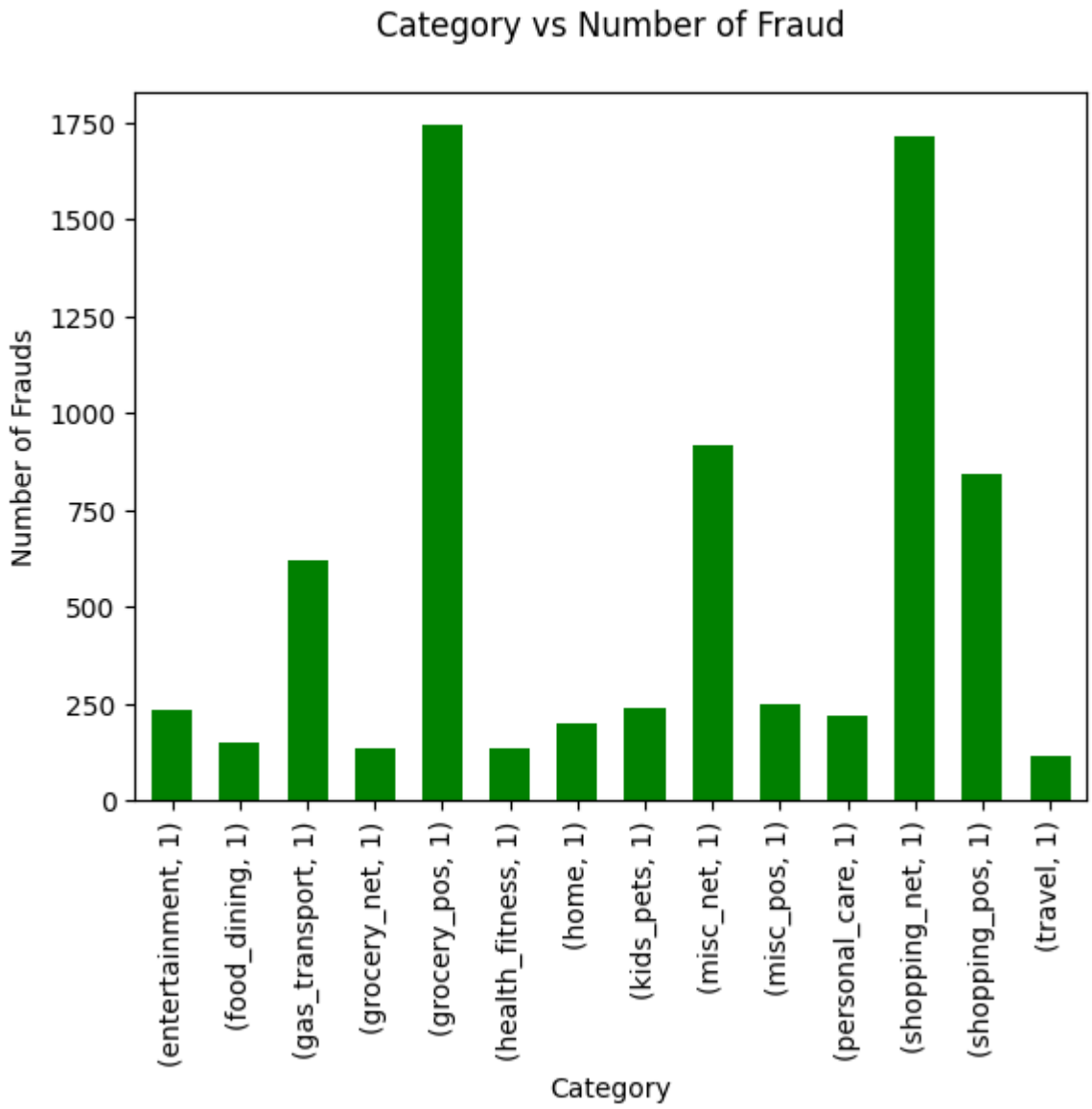
Train dataset containing only fraud transaction data for visualization purpose

In [146…
```
viz_fraud = train_data[train_data['is_fraud'] == 1]
```

In [147…
```
#plot the graph between gender and no.of frauds done
viz_fraud.groupby(['gender'])['is_fraud'].value_counts().plot.bar(color='g')
plt.title('Gender VS Fraud')
plt.show()
```

In [148…

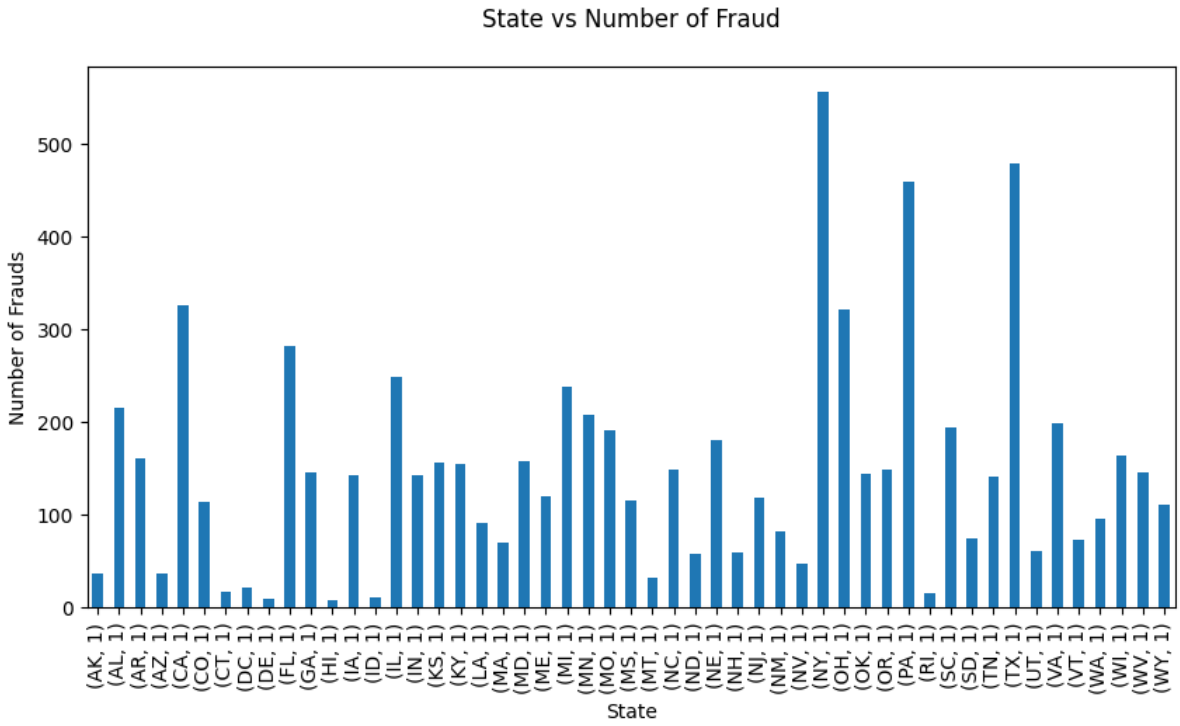
```
#plot a bar graph to show the category-wise fraud
viz_fraud.groupby(['category'])['is_fraud'].value_counts().plot.bar(color='g',widt
plt.title('\nCategory vs Number of Fraud\n')
plt.ylabel('Number of Frauds')
plt.xlabel('Category')
plt.figure(figsize=(8,8))
plt.show()
```




```
<Figure size 800x800 with 0 Axes>
```

In [149…

```
plt.figure(figsize=[10,5])
viz_fraud.groupby(['state'])['is_fraud'].value_counts().plot.bar()
plt.title('\nState vs Number of Fraud\n')
plt.ylabel('Number of Frauds')
plt.xlabel('State')
plt.show()
```



In [150…
```
train_data.columns
```

Out[150]:
```
Index(['trans_date_trans_time', 'cc_num', 'category', 'amt', 'gender', 'city',
       'state', 'lat', 'long', 'job', 'dob', 'merch_lat', 'merch_long',
       'is_fraud', 'trans_date'],
      dtype='object')
```

In [151…
```
#One Hot Encoding of Category column
train_data = pd.get_dummies(train_data, columns=['category'], prefix='category')
test_data = pd.get_dummies(test_data, columns=['category'], prefix='category')

test_data = test_data.reindex(columns=train_data.columns, fill_value=0)
```

In [152…
```
# Creating field for age on transaction date

train_data['trans_age'] = train_data['trans_date'] - train_data['dob']
train_data['trans_age'] = train_data['trans_age'].astype('timedelta64[Y]')

test_data['trans_age'] = test_data['trans_date'] - test_data['dob']
test_data['trans_age'] = test_data['trans_age'].astype('timedelta64[Y]')
```

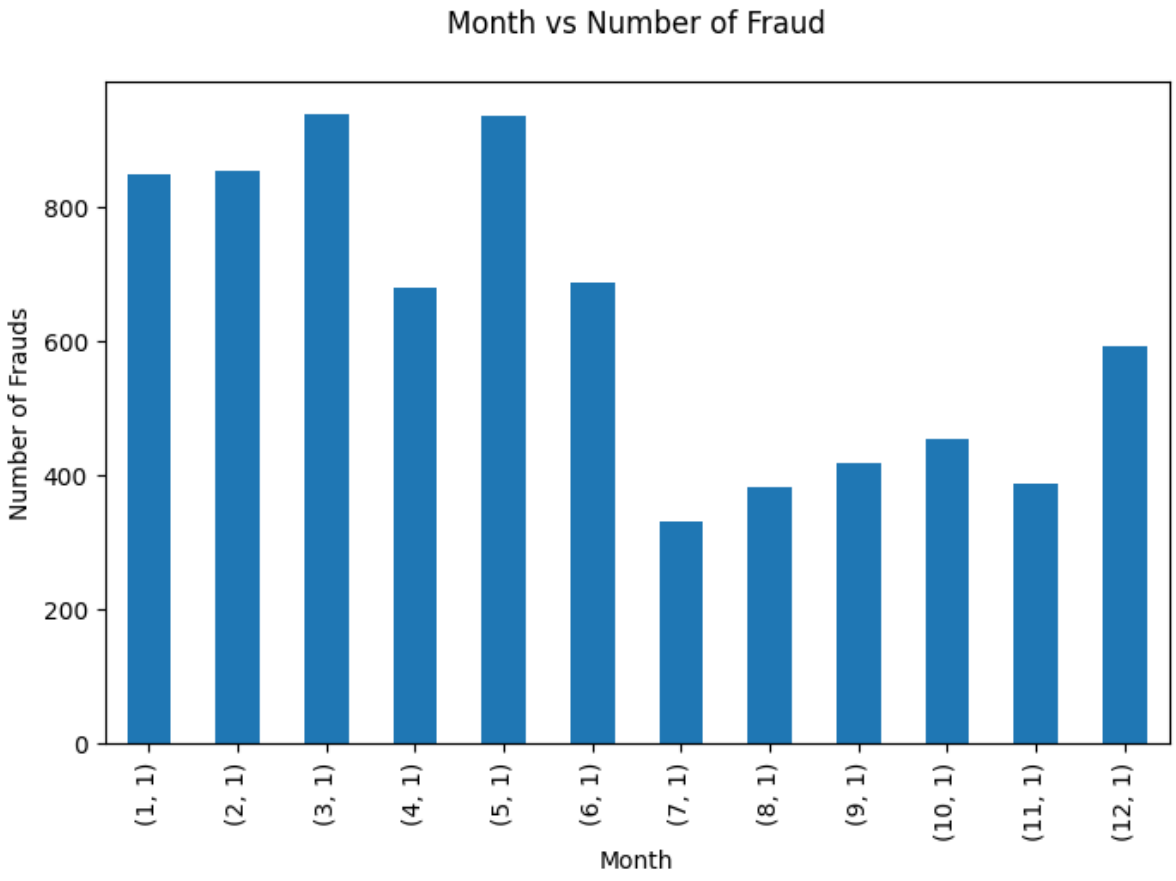
In [153…
```
# Creating month column

train_data['trans_month'] = pd.DatetimeIndex(train_data['trans_date']).month
test_data['trans_month']= pd.DatetimeIndex(test_data['trans_date']).month


train_data[['trans_date_trans_time','trans_month']].head()
```

Out[153]:

	trans_date_trans_time	trans_month
0	2019-01-01 00:00:18	1
1	2019-01-01 00:00:44	1
2	2019-01-01 00:00:51	1
3	2019-01-01 00:01:16	1
4	2019-01-01 00:03:06	1

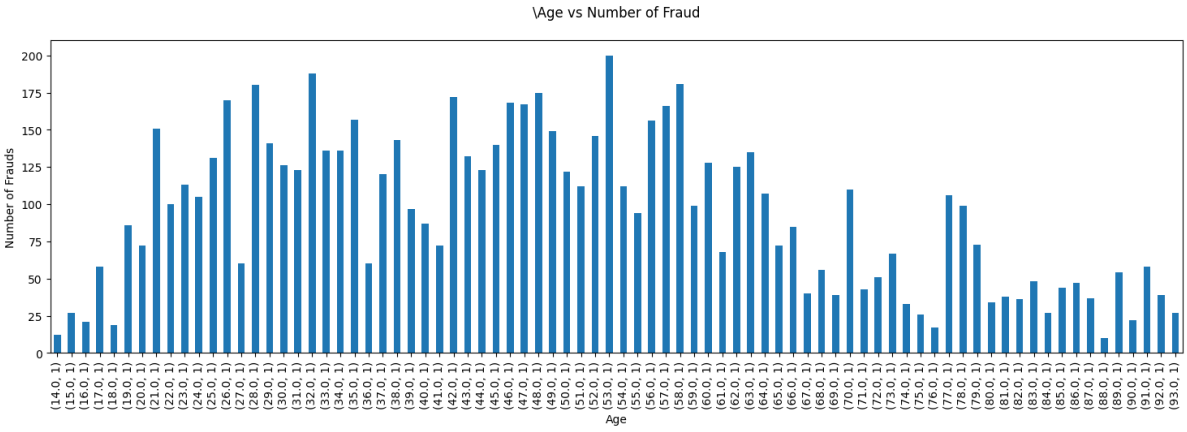
In [154…

```
viz_fraud = train_data[train_data['is_fraud'] == 1]

plt.figure(figsize=[8,5])
viz_fraud.groupby(['trans_month'])['is_fraud'].value_counts().plot.bar()
plt.title('\nMonth vs Number of Fraud\n')
plt.ylabel('Number of Frauds')
plt.xlabel('Month')
plt.show()
```



In [155…

```
plt.figure(figsize=[18,5])
viz_fraud.groupby(['trans_age'])['is_fraud'].value_counts().plot.bar()
plt.title('\Age vs Number of Fraud\n')
plt.ylabel('Number of Frauds')
plt.xlabel('Age')
plt.show()
```



Cost Benefit Analysis before Fraud Detection by Model

In [156...
```
number_tran_per_month = train_data.groupby(['trans_month'])['cc_num'].count()
print(number_tran_per_month)
```

```
trans_month
1     104727
2      97657
3     143789
4     134970
5     146875
6     143811
7      86596
8      87359
9      70652
10     68758
11     70421
12    141060
Name: cc_num, dtype: int64
```

Average Number of Monthly Transactions

In [157...
```
average_number_of_monthly_transaction = (round(sum(number_tran_per_month)/number_tr
print(average_number_of_monthly_transaction)
```

```
108056.25
```

Average Number of Monthly Fraudulent Transactions

In [158...
```
fraudulent_transaction = train_data[train_data['is_fraud'] == 1]
```

In [159...
```
fraudulent_transactions_per_month = fraudulent_transaction.groupby('trans_month')[
print(fraudulent_transactions_per_month)
```

```
trans_month
1     849
2     853
3     938
4     678
5     935
6     688
7     331
8     382
9     418
10    454
11    388
12    592
Name: cc_num, dtype: int64
```

In [160…
```
average_monthly_fraudulent_transactions = fraudulent_transactions_per_month.median(

print(average_monthly_fraudulent_transactions)
```

```
635.0
```

Average Amount of Fraudulent Transaction

In [161…
```
average_amount_of_fraud_transaction = fraudulent_transaction.groupby(['trans_month
average_amount_of_fraud_transaction = round(sum(average_amount_of_fraud_transactior
print(round(average_amount_of_fraud_transaction, 2))
```

```
461.56
```

Cost Incurred before model deployment

In [162…
```
cost_incurred_before = average_monthly_fraudulent_transactions * average_amount_of_
print(round(cost_incurred_before, 2))
```

```
293091.13
```

In [163…
```
#Dropping unnecessary columns
train_data.drop(['trans_date_trans_time','city','lat','long','job','state','dob','r
test_data.drop(['trans_date_trans_time','city','lat','long','job','dob','state','me
```

In [164…
```
train_data.shape
```

Out[164]:
```
(1296675, 20)
```

In [165…
```
test_data.shape
```

Out[165]:
```
(555719, 20)
```

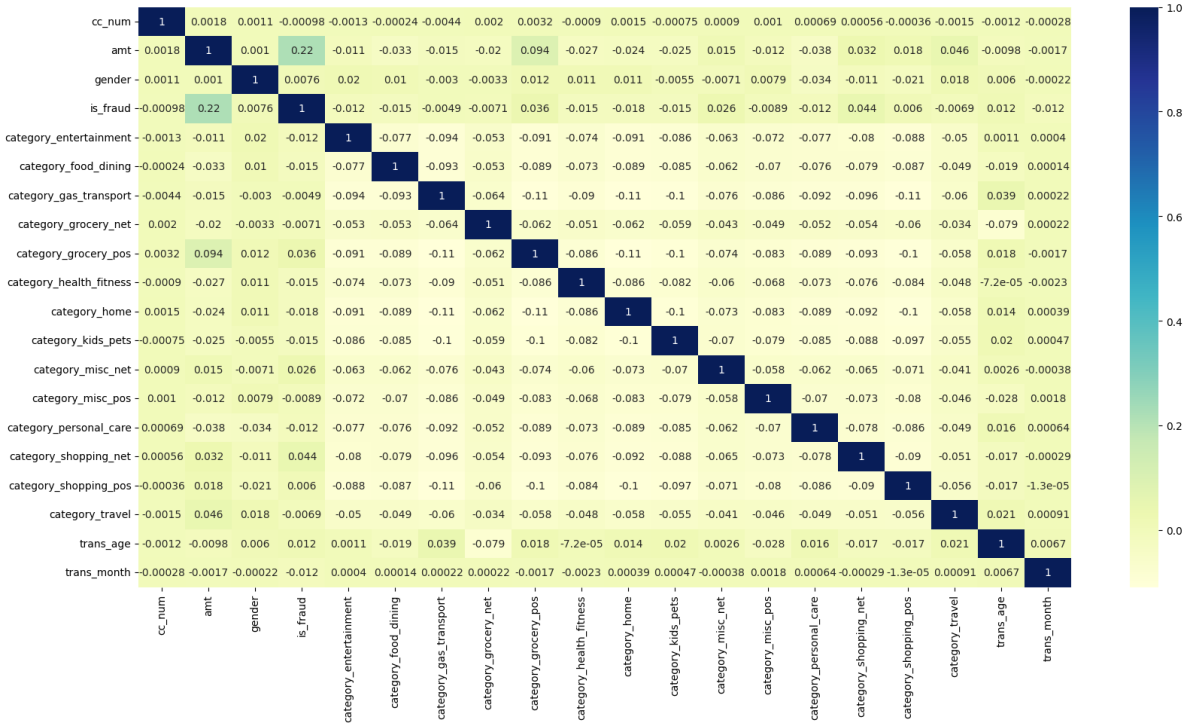
In [166…
```
train_data.corr()  #finding the correlation
```

Out[166]:

	cc_num	amt	gender	is_fraud	category_entertainment	categc
cc_num	1.000000	0.001769	0.001112	-0.000981	-0.001274	
amt	0.001769	1.000000	0.001034	0.219404	-0.010709	
gender	0.001112	0.001034	1.000000	0.007642	0.019591	
is_fraud	-0.000981	0.219404	0.007642	1.000000	-0.012200	
category_entertainment	-0.001274	-0.010709	0.019591	-0.012200	1.000000	
category_food_dining	-0.000240	-0.033102	0.010173	-0.015025	-0.077021	
category_gas_transport	-0.004438	-0.014503	-0.003029	-0.004851	-0.093991	
category_grocery_net	0.002040	-0.019831	-0.003297	-0.007136	-0.053289	
category_grocery_pos	0.003191	0.094389	0.011994	0.035558	-0.090771	
category_health_fitness	-0.000901	-0.026860	0.011030	-0.014885	-0.074462	
category_home	0.001548	-0.024408	0.011206	-0.017848	-0.090558	
category_kids_pets	-0.000746	-0.024701	-0.005471	-0.014967	-0.086402	
category_misc_net	0.000898	0.014856	-0.007138	0.025886	-0.063333	
category_misc_pos	0.001006	-0.011905	0.007930	-0.008937	-0.071529	
category_personal_care	0.000685	-0.038303	-0.033654	-0.012167	-0.076702	
category_shopping_net	0.000561	0.032153	-0.011368	0.044261	-0.079742	
category_shopping_pos	-0.000364	0.018492	-0.021437	0.005955	-0.087916	
category_travel	-0.001458	0.046097	0.018289	-0.006924	-0.050207	
trans_age	-0.001224	-0.009754	0.006037	0.012250	0.001056	
trans_month	-0.000281	-0.001748	-0.000215	-0.012409	0.000403	

In [167…

```
#plot a heatmap to show correlation
plt.figure(figsize = (20, 10))
sns.heatmap(train_data.corr(), annot = True, cmap="YlGnBu")
plt.show()
```



Step 5: Checking for Skewness

We use skewness analysis in the dataset to gain insights and potentially address skewness issues

In [168…

```
numerical_features = train_data.select_dtypes(include=[np.number])
skewness = numerical_features.apply(lambda x: skew(x))

# Display skewness for all numerical features
print(skewness)
```

```
cc_num                      2.851876
amt                        42.277825
gender                      0.190655
is_fraud                   13.029107
category_entertainment      3.297049
category_food_dining        3.354585
category_gas_transport      2.638513
category_grocery_net        5.056162
category_grocery_pos        2.755553
category_health_fitness     3.488522
category_home               2.763536
category_kids_pets          2.926934
category_misc_net           4.188092
category_misc_pos           3.652956
category_personal_care      3.370821
category_shopping_net       3.220980
category_shopping_pos       2.865785
category_travel             5.389191
trans_age                   0.610883
trans_month                 0.298515
dtype: float64
```

Here,A skewness value of 42.28 for the 'amt' (transaction amount) feature

indicates that the distribution of transaction amounts is highly positively skewed (right-skewed).

This skewness value suggests that there may be a small number of transactions with very high amounts that are causing the right-skew in the distribution.

Other features do not have such high skewness value,so it does not required much things do with it ,except 'is_fraud' skewness whose value might be high due to class imbalance

Dealing with skewness

To address the skewness, we might consider applying a data transformation to the 'amt' feature. One common approach is to use a log transformation, which can help make the distribution more symmetric by compressing the range of higher values.

In [169…
```
train_data['amt'] = np.log1p(train_data['amt'])
```

By taking the logarithm of the transaction amounts using np.log1p(), we can reduce the impact of extreme high values and potentially make the distribution closer to a normal distribution. After the transformation, we can check the skewness of the 'amt_log' feature to see if it has reduced.

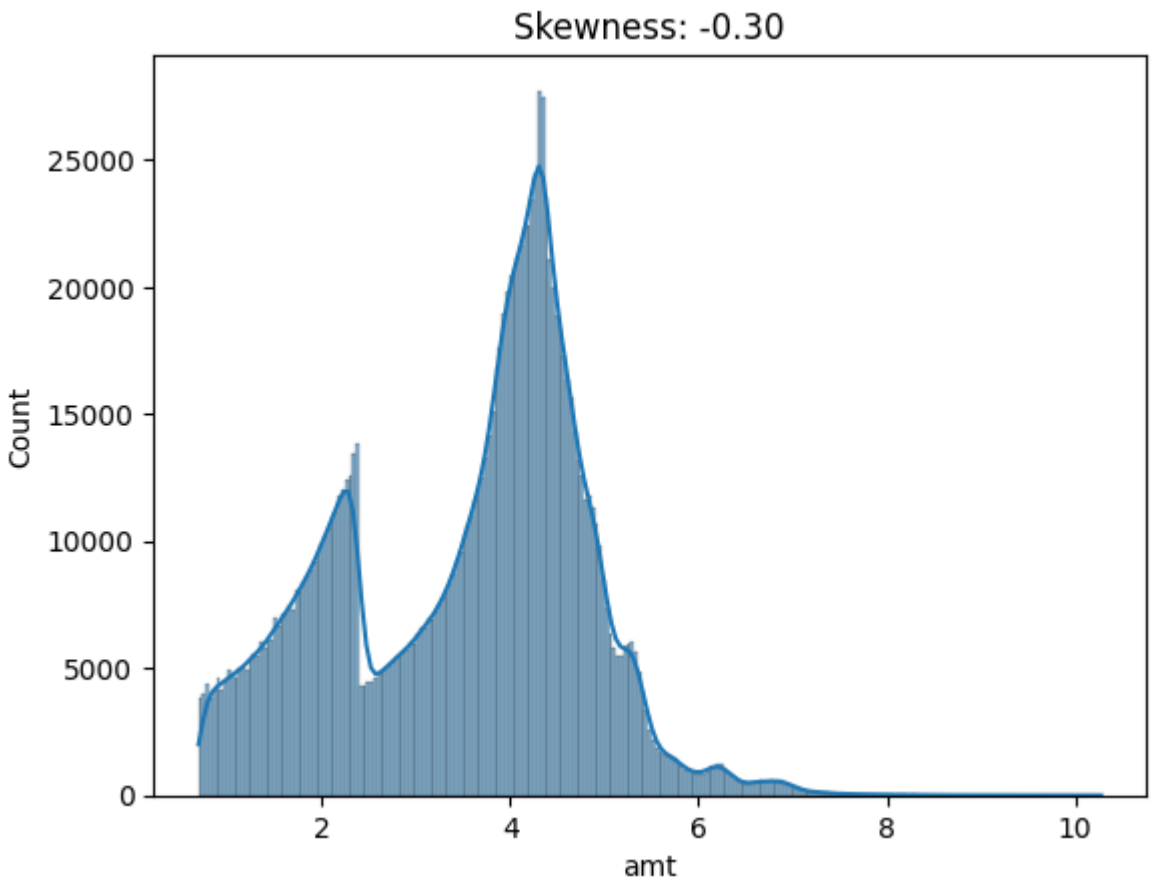
In [170…
```
#finding the skewness of 'amt' features
amt = skew(train_data['amt'])
print(f"Skewness of 'amt' feature: {amt:.2f}")
```

```
Skewness of 'amt' feature: -0.30
```

A skewness of -0.30 for the 'amt' feature indicates that the logarithmic transformation has successfully reduced the skewness and made the distribution of transaction amounts more symmetric.

A skewness value close to zero (-0.30 in this case) suggests that the distribution is approximately symmetric.

In [171…
```
#Understanding skewness by plotting a graph
sns.histplot(train_data['amt'], kde=True)
plt.title(f'Skewness: {amt:.2f}')
plt.show()
```



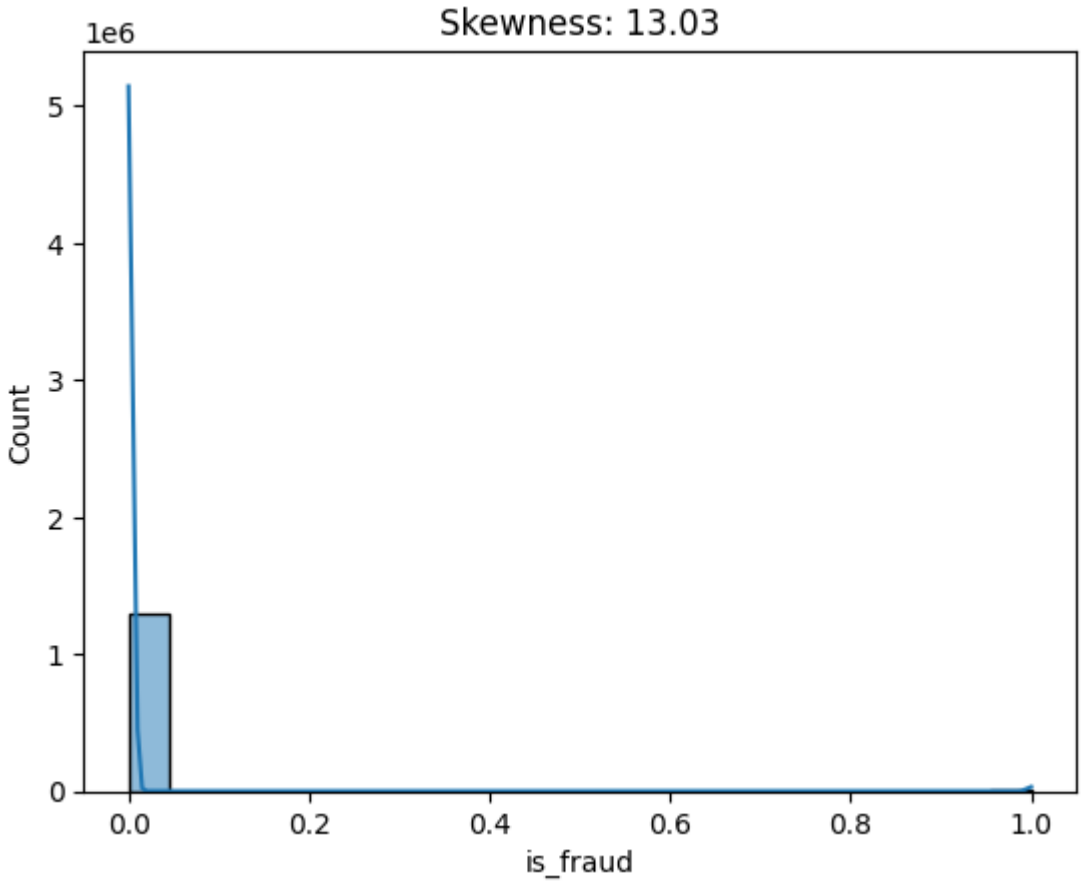
In [172…
```
from scipy.stats import skew

# Calculate the skewness of the 'is_fraud' feature
skewness_is_fraud = skew(train_data['is_fraud'])
```

```
# Print the skewness
print(f"Skewness of 'is_fraud' feature: {skewness_is_fraud:.2f}")
```

```
Skewness of 'is_fraud' feature: 13.03
```

In [173…

```
sns.histplot(train_data['is_fraud'], kde=True)
plt.title(f'Skewness: {skewness["is_fraud"]:.2f}')
plt.show()
```



Step 6:Splitting Data into Train and Test Set

In [174…

```
X_train = train_data.drop('is_fraud', axis=1)
y_train = train_data['is_fraud']
X_test = test_data.drop('is_fraud', axis=1)
y_test = test_data['is_fraud']
```

In [175…

```
#Standardizing the dataset
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

Step 7 : Handling Data imbalance

In [176…

```
# Using ADASYN or SMOTE to oversample

#ADASYN
from sklearn.linear_model import LogisticRegression
from imblearn.over_sampling import  SMOTE,ADASYN  #import SMOTE or ADASYN libraries
X_resampled, y_resampled=ADASYN().fit_resample(X_train, y_train) #fit train dataset
from collections import Counter
```

```
print(sorted(Counter(y_resampled).items()))
clf_adasyn = LogisticRegression().fit(X_resampled, y_resampled)

#SMOTE
X_resampled, y_resampled=SMOTE().fit_resample(X_train, y_train)
print(sorted(Counter(y_resampled).items())) #fit train dataset into SMOTE to overs

clf_smote = LogisticRegression().fit(X_resampled, y_resampled)
```

```
[(0, 1289169), (1, 1288586)]
[(0, 1289169), (1, 1289169)]
```

In [177…

```
from sklearn.metrics import accuracy_score


# Using ADASYN

y_pred_adasyn = clf_adasyn.predict(X_test)
accuracy_adasyn = accuracy_score(y_test, y_pred_adasyn)

# Using SMOTE

y_pred_smote = clf_smote.predict(X_test)
accuracy_smote = accuracy_score(y_test, y_pred_smote)

# Print the accuracy scores
print("Accuracy for ADASYN:", accuracy_adasyn)
print("Accuracy for SMOTE:", accuracy_smote)
```

```
Accuracy for ADASYN: 0.7815514675582443
Accuracy for SMOTE: 0.9210284334348835
```

In [178…

```
X_test.shape,X_train.shape
```

Out[178]:

```
((555719, 19), (1296675, 19))
```

Step 8: Model Evaluation

Model 1:Simple Logistic Regression

In [179…

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Define the number of folds
n_splits = 5  # You can adjust this number as needed

# Initialize the StratifiedKFold object
stratified_kfold = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=42


# Initialize lists to store evaluation metrics for Logistic Regression
lr_accuracy_scores = []
lr_precision_scores = []
lr_recall_scores = []
lr_f1_scores = []

# Perform cross-validation for Logistic Regression
for train_index, test_index in stratified_kfold.split(X_train, y_train):
    X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]
```

```
    # Initialize and train the Logistic Regression classifier
    lr_classifier = LogisticRegression(random_state=42)
    lr_classifier.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred_lr = lr_classifier.predict(X_test_fold)

    # Calculate evaluation metrics for Logistic Regression
    accuracy_lr = accuracy_score(y_test_fold, y_pred_lr)
    precision_lr = precision_score(y_test_fold, y_pred_lr)
    recall_lr = recall_score(y_test_fold, y_pred_lr)
    f1_lr = f1_score(y_test_fold, y_pred_lr)

    # Append the scores to the lists
    lr_accuracy_scores.append(accuracy_lr)
    lr_precision_scores.append(precision_lr)
    lr_recall_scores.append(recall_lr)
    lr_f1_scores.append(f1_lr)

# Calculate the average scores across all folds for Logistic Regression
avg_accuracy_lr = np.mean(lr_accuracy_scores)
avg_precision_lr = np.mean(lr_precision_scores)
avg_recall_lr = np.mean(lr_recall_scores)
avg_f1_lr = np.mean(lr_f1_scores)

# Print the average scores for Logistic Regression
print("Logistic Regression Classifier:")
print(f'Average Accuracy: {avg_accuracy_lr:.2f}')
print(f'Average Precision: {avg_precision_lr:.2f}')
print(f'Average Recall: {avg_recall_lr:.2f}')
print(f'Average F1 Score: {avg_f1_lr:.2f}')
```

```
Logistic Regression Classifier:
Average Accuracy: 0.99
Average Precision: 0.01
Average Recall: 0.00
Average F1 Score: 0.00
```

Model 2: RandomForestClassifier

In [180…

```
from sklearn.ensemble import RandomForestClassifier

# Perform cross-validation for RandomForestClassifier
for train_index, test_index in stratified_kfold.split(X_train, y_train):
    X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

# Initialize lists to store evaluation metrics
avg_accuracy_scores = []
avg_precision_scores = []
avg_recall_scores = []
avg_f1_scores = []
fraud_accuracy_scores = []
fraud_precision_scores = []
fraud_recall_scores = []
fraud_f1_scores = []
non_fraud_accuracy_scores = []
non_fraud_precision_scores = []
non_fraud_recall_scores = []
non_fraud_f1_scores = []

 # Initialize and train the Random Forest classifier
rf = RandomForestClassifier(random_state=42)
```

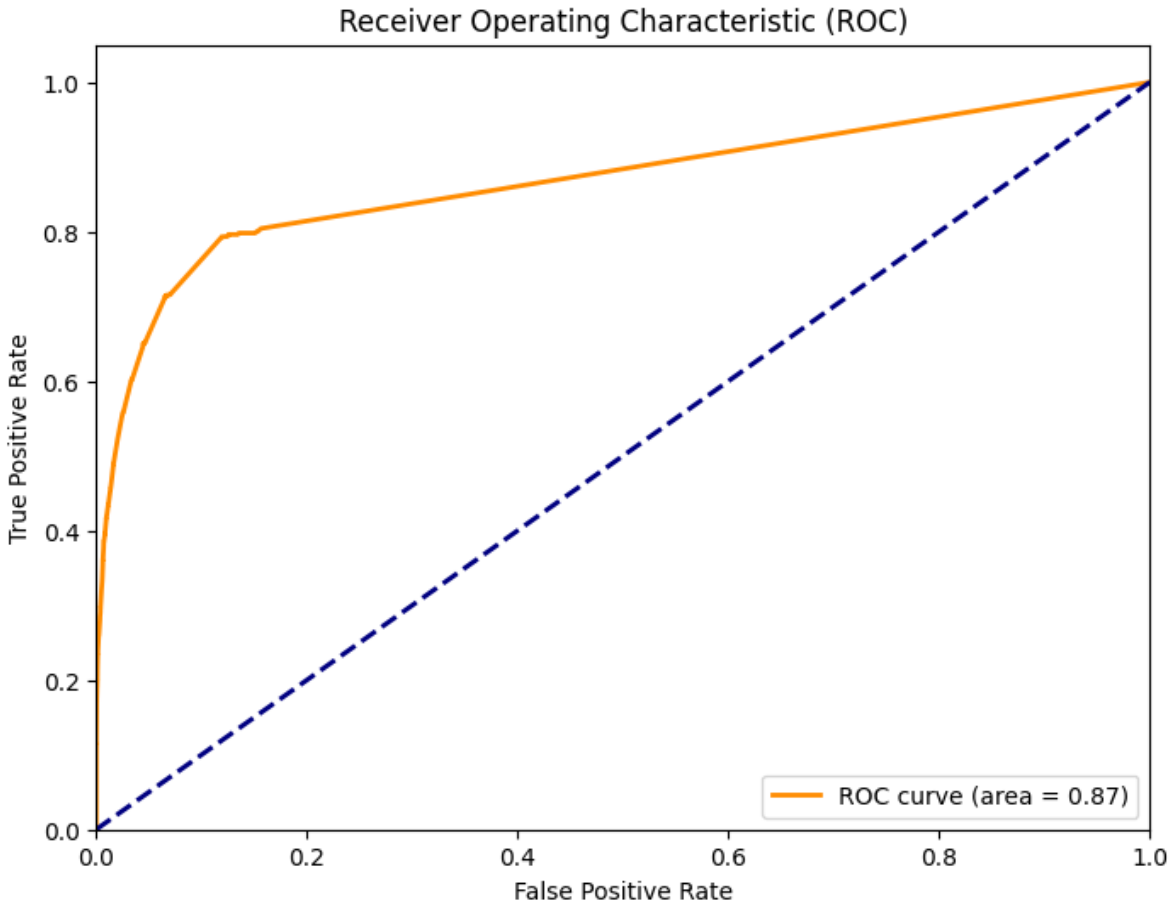
```
rf.fit(X_train, y_train)

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

 # Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Append the scores to the lists
avg_accuracy_scores.append(accuracy)
avg_precision_scores.append(precision)
avg_recall_scores.append(recall)
avg_f1_scores.append(f1)

 # Calculate individual scores for fraud and non-fraud classes
fraud_case = y_test == 1
non_fraud_case = y_test == 0

fraud_accuracy = accuracy_score(y_test[fraud_case], y_pred[fraud_case])
fraud_precision = precision_score(y_test[fraud_case], y_pred[fraud_case])
fraud_recall = recall_score(y_test[fraud_case], y_pred[fraud_case])
fraud_f1 = f1_score(y_test[fraud_case], y_pred[fraud_case])



non_fraud_accuracy = accuracy_score(y_test[non_fraud_case], y_pred[non_fraud_case])
non_fraud_precision = precision_score(y_test[non_fraud_case], y_pred[non_fraud_case
non_fraud_recall = recall_score(y_test[non_fraud_case], y_pred[non_fraud_case])
non_fraud_f1 = f1_score(y_test[non_fraud_case], y_pred[non_fraud_case])

fraud_accuracy_scores.append(fraud_accuracy)
fraud_precision_scores.append(fraud_precision)
fraud_recall_scores.append(fraud_recall)
fraud_f1_scores.append(fraud_f1)

non_fraud_accuracy_scores.append(non_fraud_accuracy)
non_fraud_precision_scores.append(non_fraud_precision)
non_fraud_recall_scores.append(non_fraud_recall)
non_fraud_f1_scores.append(non_fraud_f1)

# Calculate the average scores across all folds
avg_accuracy = np.mean(avg_accuracy_scores)
avg_precision = np.mean(avg_precision_scores)
avg_recall = np.mean(avg_recall_scores)
avg_f1 = np.mean(avg_f1_scores)

# Calculate the average scores for fraud and non-fraud classes
avg_fraud_accuracy = np.mean(fraud_accuracy_scores)
avg_fraud_precision = np.mean(fraud_precision_scores)
avg_fraud_recall = np.mean(fraud_recall_scores)
avg_fraud_f1 = np.mean(fraud_f1_scores)

avg_non_fraud_accuracy = np.mean(non_fraud_accuracy_scores)
avg_non_fraud_precision = np.mean(non_fraud_precision_scores)
avg_non_fraud_recall = np.mean(non_fraud_recall_scores)
avg_non_fraud_f1 = np.mean(non_fraud_f1_scores)

# Print the average scores and individual scores
print("Random Forest Classifier - Average Scores:")
print(f'Average Accuracy: {avg_accuracy:.2f}')
print(f'Average Precision: {avg_precision:.2f}')
```

```
print(f'Average Recall: {avg_recall:.2f}')
print(f'Average F1 Score: {avg_f1:.2f}')

print("Random Forest Classifier - Fraud Class Scores:")
print(f'Average Accuracy for Fraud Class: {avg_fraud_accuracy:.2f}')
print(f'Average Precision for Fraud Class: {avg_fraud_precision:.2f}')
print(f'Average Recall for Fraud Class: {avg_fraud_recall:.2f}')
print(f'Average F1 Score for Fraud Class: {avg_fraud_f1:.2f}')

print("Random Forest Classifier - Non-Fraud Class Scores:")
print(f'Average Accuracy for Non-Fraud Class: {avg_non_fraud_accuracy:.2f}')
print(f'Average Precision for Non-Fraud Class: {avg_non_fraud_precision:.2f}')
print(f'Average Recall for Non-Fraud Class: {avg_non_fraud_recall:.2f}')
print(f'Average F1 Score for Non-Fraud Class: {avg_non_fraud_f1:.2f}')
```

```
Random Forest Classifier - Average Scores:
Average Accuracy: 1.00
Average Precision: 0.58
Average Recall: 0.17
Average F1 Score: 0.26
Random Forest Classifier - Fraud Class Scores:
Average Accuracy for Fraud Class: 0.17
Average Precision for Fraud Class: 1.00
Average Recall for Fraud Class: 0.17
Average F1 Score for Fraud Class: 0.29
Random Forest Classifier - Non-Fraud Class Scores:
Average Accuracy for Non-Fraud Class: 1.00
Average Precision for Non-Fraud Class: 0.00
Average Recall for Non-Fraud Class: 0.00
Average F1 Score for Non-Fraud Class: 0.00
```

In [181…

```
from sklearn.metrics import roc_curve, auc


# Predict the probabilities for the positive class (class 1)
y_proba = rf.predict_proba(X_test)[:, 1]

# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

# Calculate the AUC (Area Under the Curve)
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```



Model Performance: The closer the ROC curve is to the top-left corner and the larger the AUC, the better your model's performance. It signifies that the model can differentiate between the positive and negative classes effectively across various threshold settings.

The y-axis of the ROC curve represents the True Positive Rate (TPR), also known as Sensitivity or Recall. It measures the proportion of actual positive cases that the model correctly identifies as positive.

The x-axis of the ROC curve represents the False Positive Rate (FPR). It measures the proportion of actual negative cases that the model incorrectly classifies as positive

As you move along the curve from left to right, you are changing the threshold from lenient (classifying more instances as positive) to strict (classifying fewer instances as positive).

Model 3: Using DecisionTreeClassifier

In [182…

```
from sklearn.tree import DecisionTreeClassifier

# ... (similar code as the Random Forest section to load data, initialize Stratifie

# Perform cross-validation for DecisionTreeClassifier
for train_index, test_index in stratified_kfold.split(X_train, y_train):
    X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

# Initialize and train the Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
```

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

# ... (similar code as the Random Forest section to calculate and append scores)
# Calculate evaluation metrics

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

 # Append the scores to the lists
avg_accuracy_scores.append(accuracy)
avg_precision_scores.append(precision)
avg_recall_scores.append(recall)
avg_f1_scores.append(f1)

 # Calculate individual scores for fraud and non-fraud classes
fraud_case = y_test == 1
non_fraud_case = y_test == 0

fraud_accuracy = accuracy_score(y_test[fraud_case], y_pred[fraud_case])
fraud_precision = precision_score(y_test[fraud_case], y_pred[fraud_case])
fraud_recall = recall_score(y_test[fraud_case], y_pred[fraud_case])
fraud_f1 = f1_score(y_test[fraud_case], y_pred[fraud_case])



non_fraud_accuracy = accuracy_score(y_test[non_fraud_case], y_pred[non_fraud_case])
non_fraud_precision = precision_score(y_test[non_fraud_case], y_pred[non_fraud_case
non_fraud_recall = recall_score(y_test[non_fraud_case], y_pred[non_fraud_case])
non_fraud_f1 = f1_score(y_test[non_fraud_case], y_pred[non_fraud_case])

fraud_accuracy_scores.append(fraud_accuracy)
fraud_precision_scores.append(fraud_precision)
fraud_recall_scores.append(fraud_recall)
fraud_f1_scores.append(fraud_f1)

non_fraud_accuracy_scores.append(non_fraud_accuracy)
non_fraud_precision_scores.append(non_fraud_precision)
non_fraud_recall_scores.append(non_fraud_recall)
non_fraud_f1_scores.append(non_fraud_f1)

# Calculate the average scores across all folds
avg_accuracy = np.mean(avg_accuracy_scores)
avg_precision = np.mean(avg_precision_scores)
avg_recall = np.mean(avg_recall_scores)
avg_f1 = np.mean(avg_f1_scores)

# Calculate the average scores for fraud and non-fraud classes
avg_fraud_accuracy = np.mean(fraud_accuracy_scores)
avg_fraud_precision = np.mean(fraud_precision_scores)
avg_fraud_recall = np.mean(fraud_recall_scores)
avg_fraud_f1 = np.mean(fraud_f1_scores)

avg_non_fraud_accuracy = np.mean(non_fraud_accuracy_scores)
avg_non_fraud_precision = np.mean(non_fraud_precision_scores)
avg_non_fraud_recall = np.mean(non_fraud_recall_scores)
avg_non_fraud_f1 = np.mean(non_fraud_f1_scores)

# Print the average scores and individual scores for Decision Tree
print("Decision Tree Classifier - Average Scores:")
# ... (similar code as the Random Forest section to print average and individual sc
print(f'Average Accuracy: {avg_accuracy:.2f}')
```

```
print(f'Average Precision: {avg_precision:.2f}')
print(f'Average Recall: {avg_recall:.2f}')
print(f'Average F1 Score: {avg_f1:.2f}')

print("Decision Tree Classifier - Fraud Class Scores:")
print(f'Average Accuracy for Fraud Class: {avg_fraud_accuracy:.2f}')
print(f'Average Precision for Fraud Class: {avg_fraud_precision:.2f}')
print(f'Average Recall for Fraud Class: {avg_fraud_recall:.2f}')
print(f'Average F1 Score for Fraud Class: {avg_fraud_f1:.2f}')

print("Decision Tree Classifier - Non-Fraud Class Scores:")
print(f'Average Accuracy for Non-Fraud Class: {avg_non_fraud_accuracy:.2f}')
print(f'Average Precision for Non-Fraud Class: {avg_non_fraud_precision:.2f}')
print(f'Average Recall for Non-Fraud Class: {avg_non_fraud_recall:.2f}')
print(f'Average F1 Score for Non-Fraud Class: {avg_non_fraud_f1:.2f}')
```

```
Decision Tree Classifier - Average Scores:
Average Accuracy: 0.99
Average Precision: 0.34
Average Recall: 0.17
Average F1 Score: 0.19
Decision Tree Classifier - Fraud Class Scores:
Average Accuracy for Fraud Class: 0.17
Average Precision for Fraud Class: 1.00
Average Recall for Fraud Class: 0.17
Average F1 Score for Fraud Class: 0.29
Decision Tree Classifier - Non-Fraud Class Scores:
Average Accuracy for Non-Fraud Class: 1.00
Average Precision for Non-Fraud Class: 0.00
Average Recall for Non-Fraud Class: 0.00
Average F1 Score for Non-Fraud Class: 0.00
```

Model 4:Using Xgboost

In [183…

```
from xgboost import XGBClassifier

# ... (similar code as the Random Forest section to load data, initialize Stratifie

# Perform cross-validation for Xgboost
for train_index, test_index in stratified_kfold.split(X_train, y_train):
    X_train_fold, X_test_fold = X_train[train_index], X_train[test_index]
    y_train_fold, y_test_fold = y_train[train_index], y_train[test_index]

 # Initialize and train the XGBoost classifier
xgb_classifier = XGBClassifier(random_state=42)
xgb_classifier.fit(X_train, y_train)

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

 # ... (similar code as the Random Forest section to calculate and append scores)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Append the scores to the lists
avg_accuracy_scores.append(accuracy)
avg_precision_scores.append(precision)
avg_recall_scores.append(recall)
avg_f1_scores.append(f1)

# Calculate individual scores for fraud and non-fraud classes
fraud_case = y_test == 1
```

```
non_fraud_case = y_test == 0

fraud_accuracy = accuracy_score(y_test[fraud_case], y_pred[fraud_case])
fraud_precision = precision_score(y_test[fraud_case], y_pred[fraud_case])
fraud_recall = recall_score(y_test[fraud_case], y_pred[fraud_case])
fraud_f1 = f1_score(y_test[fraud_case], y_pred[fraud_case])



non_fraud_accuracy = accuracy_score(y_test[non_fraud_case], y_pred[non_fraud_case])
non_fraud_precision = precision_score(y_test[non_fraud_case], y_pred[non_fraud_case
non_fraud_recall = recall_score(y_test[non_fraud_case], y_pred[non_fraud_case])
non_fraud_f1 = f1_score(y_test[non_fraud_case], y_pred[non_fraud_case])

fraud_accuracy_scores.append(fraud_accuracy)
fraud_precision_scores.append(fraud_precision)
fraud_recall_scores.append(fraud_recall)
fraud_f1_scores.append(fraud_f1)

non_fraud_accuracy_scores.append(non_fraud_accuracy)
non_fraud_precision_scores.append(non_fraud_precision)
non_fraud_recall_scores.append(non_fraud_recall)
non_fraud_f1_scores.append(non_fraud_f1)

# Calculate the average scores across all folds
avg_accuracy = np.mean(avg_accuracy_scores)
avg_precision = np.mean(avg_precision_scores)
avg_recall = np.mean(avg_recall_scores)
avg_f1 = np.mean(avg_f1_scores)

# Calculate the average scores for fraud and non-fraud classes
avg_fraud_accuracy = np.mean(fraud_accuracy_scores)
avg_fraud_precision = np.mean(fraud_precision_scores)
avg_fraud_recall = np.mean(fraud_recall_scores)
avg_fraud_f1 = np.mean(fraud_f1_scores)

avg_non_fraud_accuracy = np.mean(non_fraud_accuracy_scores)
avg_non_fraud_precision = np.mean(non_fraud_precision_scores)
avg_non_fraud_recall = np.mean(non_fraud_recall_scores)
avg_non_fraud_f1 = np.mean(non_fraud_f1_scores)


# Print the average scores and individual scores for XGBoost
print("XGBoost Classifier - Average Scores:")
# ... (similar code as the Random Forest section to print average and individual sc
print(f'Average Accuracy: {avg_accuracy:.2f}')
print(f'Average Precision: {avg_precision:.2f}')
print(f'Average Recall: {avg_recall:.2f}')
print(f'Average F1 Score: {avg_f1:.2f}')

#Print the result for fraud and non-fraud class:
print("XGBoost Classifier - Fraud Class Scores:")
print(f'Average Accuracy for Fraud Class: {avg_fraud_accuracy:.2f}')
print(f'Average Precision for Fraud Class: {avg_fraud_precision:.2f}')
print(f'Average Recall for Fraud Class: {avg_fraud_recall:.2f}')
print(f'Average F1 Score for Fraud Class: {avg_fraud_f1:.2f}')

print("XGBoost Classifier - Non-Fraud Class Scores:")
print(f'Average Accuracy for Non-Fraud Class: {avg_non_fraud_accuracy:.2f}')
print(f'Average Precision for Non-Fraud Class: {avg_non_fraud_precision:.2f}')
print(f'Average Recall for Non-Fraud Class: {avg_non_fraud_recall:.2f}')
print(f'Average F1 Score for Non-Fraud Class: {avg_non_fraud_f1:.2f}')
```

```
XGBoost Classifier - Average Scores:
Average Accuracy: 0.99
Average Precision: 0.40
Average Recall: 0.16
Average F1 Score: 0.20
XGBoost Classifier - Fraud Class Scores:
Average Accuracy for Fraud Class: 0.16
Average Precision for Fraud Class: 1.00
Average Recall for Fraud Class: 0.16
Average F1 Score for Fraud Class: 0.28
XGBoost Classifier - Non-Fraud Class Scores:
Average Accuracy for Non-Fraud Class: 1.00
Average Precision for Non-Fraud Class: 0.00
Average Recall for Non-Fraud Class: 0.00
Average F1 Score for Non-Fraud Class: 0.00
```

Step 9:Prediction on test set

In [184…
```
y_test_pred = rf.predict(X_test)
y_test_pred[:10]
```

Out[184]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Testing model on original data without oversampling

In [185…
```
y_train_pred = rf.predict(X_train)
y_test_pred = rf.predict(X_test)
```

In [186…
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_test_pred)
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[553312    262]
 [  1785    360]]
```

In [187…
```
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_test_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

In [188…
```
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_test, y_test_pred)
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_test_pred)
print(f"F1-Score: {f1:.2f}")
```

```
F1-Score: 0.26
```

Converting y_test_pred to a Data Frame

In [189…
```
y_test_pred1 = pd.DataFrame(y_test_pred)
y_test_pred1.head()
```

Out[189]:

	0
0	0
1	0
2	0
3	0
4	0

Adding 'trans_num' as index

In [190…

```
y_test_df = pd.DataFrame(y_test)
y_test_df['trans_num'] = y_test_df.index
```

Merging y_test_df and y_test_pred_df

In [191…

```
y_test_pred1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)

final_y_pred = pd.concat([y_test_df, y_test_pred1],axis=1)

final_y_pred.head()
```

Out[191]:

	is_fraud	trans_num	0
0	0	0	0
1	0	1	0
2	0	2	0
3	0	3	0
4	0	4	0

Renaming column name '0' to 'Predictable_Value'

In [192…

```
final_y_pred = final_y_pred.rename(columns={ 0 : 'Predictable_Value'})

final_y_pred.head()
```

Out[192]:

	is_fraud	trans_num	Predictable_Value
0	0	0	0
1	0	1	0
2	0	2	0
3	0	3	0
4	0	4	0

Predicting Test Set

In [193…
```
final_y_pred['fraud_prediction'] = final_y_pred.Predictable_Value.map(lambda x: 1 i
final_y_pred.head()
```

Out[193]:

	is_fraud	trans_num	Predictable_Value	fraud_prediction
0	0	0	0	0
1	0	1	0	0
2	0	2	0	0
3	0	3	0	0
4	0	4	0	0

In [194…
```
#Predicting Probabilities on 'fraud_prediction'
final_prediction = final_y_pred['fraud_prediction']
final_prediction
```

Out[194]:
```
0         0
1         0
2         0
3         0
4         0
         ..
555714    0
555715    0
555716    0
555717    0
555718    0
Name: fraud_prediction, Length: 555719, dtype: int64
```

In [195…
```
# Assuming 'final_y_pred' is a DataFrame containing 'fraud_prediction' and has an i
test_set = test_data.merge(final_y_pred[['fraud_prediction']], left_index=True, rig
test_set.head()
```

Out[195]:

	cc_num	amt	gender	is_fraud	category_entertainment	category_food_dining	cate
0	2291163933867244	2.86	1	0	0	0	
1	3573030041201292	29.84	0	0	0	0	
2	3598215285024754	41.28	0	0	0	0	
3	3591919803438423	60.05	1	0	0	0	
4	3526826139003047	3.19	1	0	0	0	

5 rows × 21 columns

Cost Benefit Analysis after Fraud Detection by Model

In [196…
```
test_set.groupby(['trans_month'])['is_fraud'].value_counts()
```

Out[196]:
```
trans_month  is_fraud
6            0            29925
             1              133
7            0            85527
             1              321
8            0            88344
             1              415
9            0            69193
             1              340
10           0            68964
             1              384
11           0            72341
             1              294
12           0           139280
             1              258
Name: is_fraud, dtype: int64
```

In [197…
```
test_set.groupby(['trans_month'])['fraud_prediction'].value_counts()
```

Out[197]:
```
trans_month  fraud_prediction
6            0                  30022
             1                     36
7            0                  85748
             1                    100
8            0                  88659
             1                    100
9            0                  69444
             1                     89
10           0                  69258
             1                     90
11           0                  72548
             1                     87
12           0                 139418
             1                    120
Name: fraud_prediction, dtype: int64
```

In [198…
```
fraud_data = test_set[test_set['fraud_prediction'] == 1]
fraud_data.head()
```

Out[198]:

	cc_num	amt	gender	is_fraud	category_entertainment	category_food_dining
767	2610529083834453	420.98	0	0	1	0
2195	4005676619255478	326.94	1	1	0	0
2258	3524574586339330	353.08	0	1	0	0
2270	180094608895855	449.25	0	0	0	0
6143	3596357274378601	461.23	1	0	0	0

5 rows × 21 columns

Cost of providing customer support per month

In [199…
```
number_of_fraud_per_month = fraud_data.groupby(['trans_month'])['fraud_prediction']
```

In [200…
```
TF = round(sum(number_of_fraud_per_month), 2)/number_of_fraud_per_month.count() #av
print(round(TF, 2))
```

```
88.86
```

In [201…
```
total_service_cost = TF * 1.5
print(round(total_service_cost, 2))
```

```
133.29
```

Cost incurred due to these fraudulent transactions left undetected by the model

In [202…
```
undetected_fraud =test_set[(test_set['is_fraud'] == 1) & (test_set['fraud_predictio

undetected_fraud.head()
```

Out[202]:

	cc_num	amt	gender	is_fraud	category_entertainment	category_food_dining
1685	3560725013359375	24.84	0	1	0	0
1767	6564459919350820	780.52	1	1	0	0
1781	6564459919350820	620.33	1	1	1	0
1784	4005676619255478	1077.69	1	1	0	0
1857	3560725013359375	842.65	0	1	0	0

5 rows × 21 columns

In [203…
```
number_of_tran_per_month = undetected_fraud.groupby(['trans_month']).size()
```

In [204…
```
FN = sum(number_of_tran_per_month)/number_of_tran_per_month.count() #average numbe
print(round(FN, 2))
```

```
255.0
```

In [209…
```
# Calculate the incurred loss for the test data
incurred_loss = round((FN * average_amount_of_fraud_transaction), 2)

# Print the incurred loss
print(round(incurred_loss))
```

```
117698
```

Total Cost Incurred after model deployment

In [210…
```
cost_incurred_after = total_service_cost + incurred_loss
print(round(cost_incurred_after))
```

```
117831
```

Savings after model deployment

In [211…
```
savings = round((cost_incurred_before - cost_incurred_after), 2)
print(round(savings))
```

```
175260
```

In [208…
```
#Saving percentage
savings_percent = (savings/cost_incurred_before)*100
print(round(savings_percent, 2))
```

```
59.8
```