Credit Card Fraud Detection

Importing necessary libraries

In [46]:
```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.utils import resample
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score, confusion
from sklearn.model_selection import cross_val_score
```

Importing the dataset

In [2]:
```
df_creditcard = pd.read_csv("C:/Users/c2108436/OneDrive - Teesside Universit
df_creditcard.head()
```

Out[2]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt
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
2	2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28
3	3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19

5 rows × 23 columns

Summary of the dataframe

In [3]:
```
df_creditcard.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
```

Data Exploration/cleaning

In [4]:
```
df_creditcard.shape # To check the dimension
```

Out[4]:
```
(555719, 23)
```

In [5]:
```
# statistical summary
df_creditcard.describe()
```

Out[5]:

	Unnamed: 0	cc_num	amt	zip	lat	l
count	555719.000000	5.557190e+05	555719.000000	555719.000000	555719.000000	555719.00
mean	277859.000000	4.178387e+17	69.392810	48842.628015	38.543253	-90.23
std	160422.401459	1.309837e+18	156.745941	26855.283328	5.061336	13.72
min	0.000000	6.041621e+10	1.000000	1257.000000	20.027100	-165.67
25%	138929.500000	1.800429e+14	9.630000	26292.000000	34.668900	-96.79
50%	277859.000000	3.521417e+15	47.290000	48174.000000	39.371600	-87.47
75%	416788.500000	4.635331e+15	83.010000	72011.000000	41.894800	-80.17
max	555718.000000	4.992346e+18	22768.110000	99921.000000	65.689900	-67.95

To view the data type of each features

In [6]:
```
df_creditcard.dtypes
```

Out[6]:
```
Unnamed: 0                 int64
trans_date_trans_time     object
cc_num                     int64
merchant                  object
category                  object
amt                      float64
first                     object
last                      object
gender                    object
street                    object
city                      object
state                     object
zip                        int64
lat                      float64
long                     float64
city_pop                   int64
job                       object
dob                       object
trans_num                 object
unix_time                  int64
merch_lat                float64
merch_long               float64
is_fraud                   int64
dtype: object
```

Checking if any missing values present in the dataset

In [7]:
```
df_creditcard.isnull().sum()
```

Out[7]:
```
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 [8]:
```
df_creditcard.duplicated().sum() # check for duplicates
```

Out[8]:
```
0
```

Determine class distribution of the target variable

In [9]:
```
y = df_creditcard['is_fraud']

print(f'Percentage of fraudulent transactions: % {round(y.value_counts(norma
```

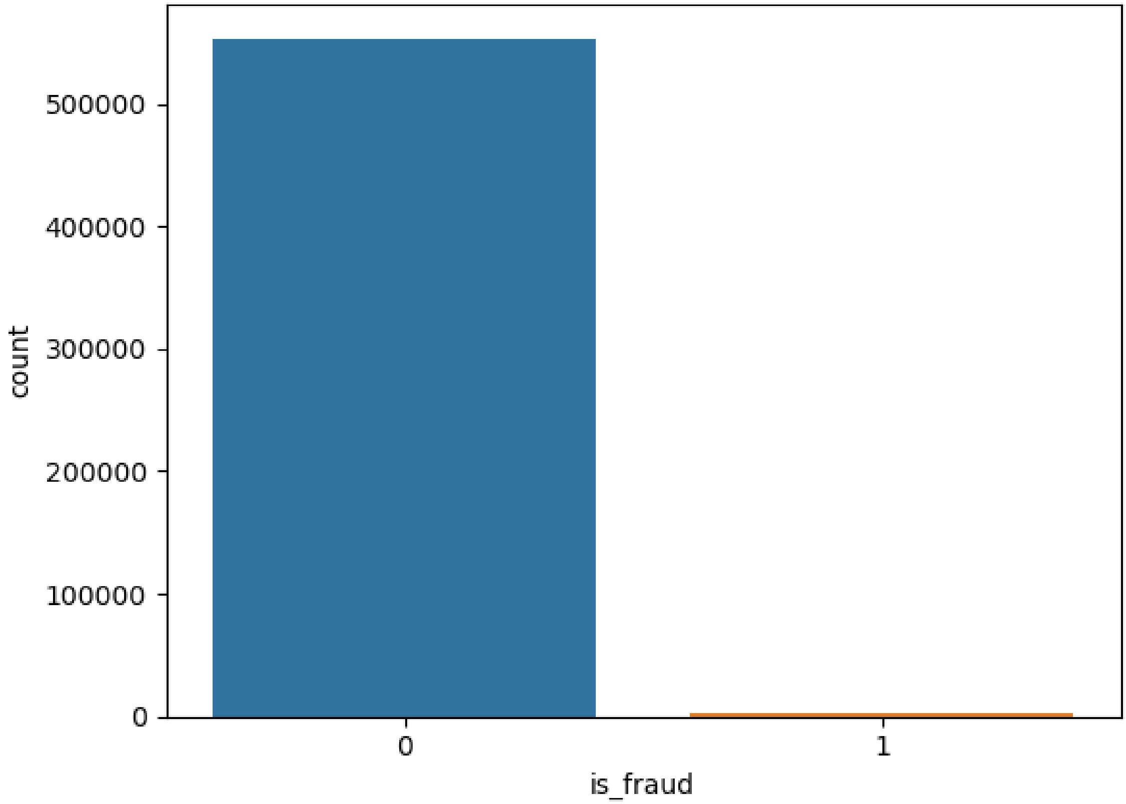
```
Percentage of fraudulent transactions: % 0.39 --> (2145 transactions)
Percentage of genuine transactions: % 99.61 --> (553574 transactions)
```

We have an unbalanced data

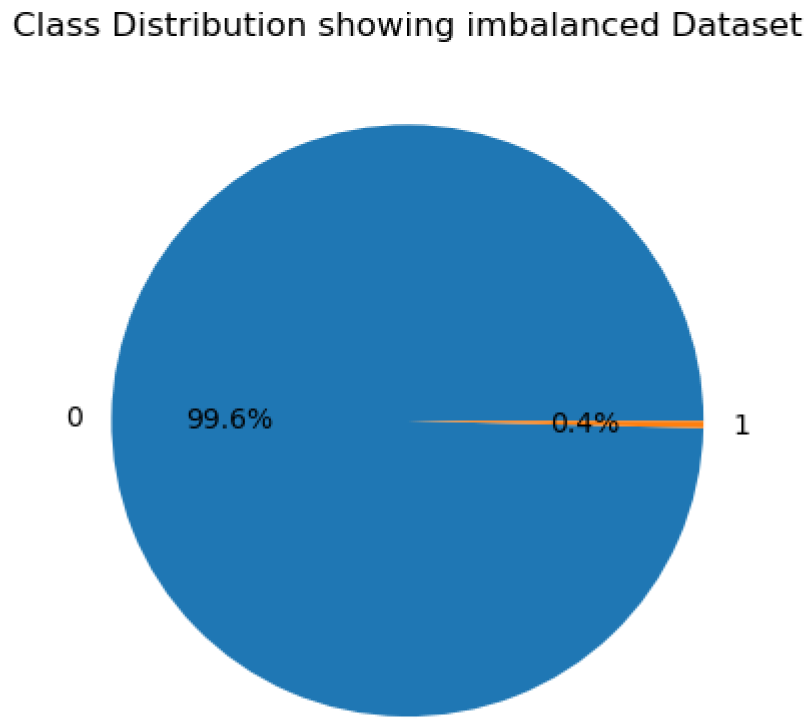
In [10]:
```
import seaborn as sns

sns.countplot(x= df_creditcard["is_fraud"])
```

Out[10]: <Axes: xlabel='is_fraud', ylabel='count'>



In [11]:
```
class_counts = df_creditcard['is_fraud'].value_counts()
plt.pie(class_counts, labels=class_counts.index, autopct='%1.1f%%')
plt.title('Class Distribution showing imbalanced Dataset')
plt.show()
```



In [12]:
```
# Seperate the fraudulent transactions from the genuine ones
df_creditcard_majority = df_creditcard[df_creditcard.is_fraud==0]
df_creditcard_minority = df_creditcard[df_creditcard.is_fraud==1]
```

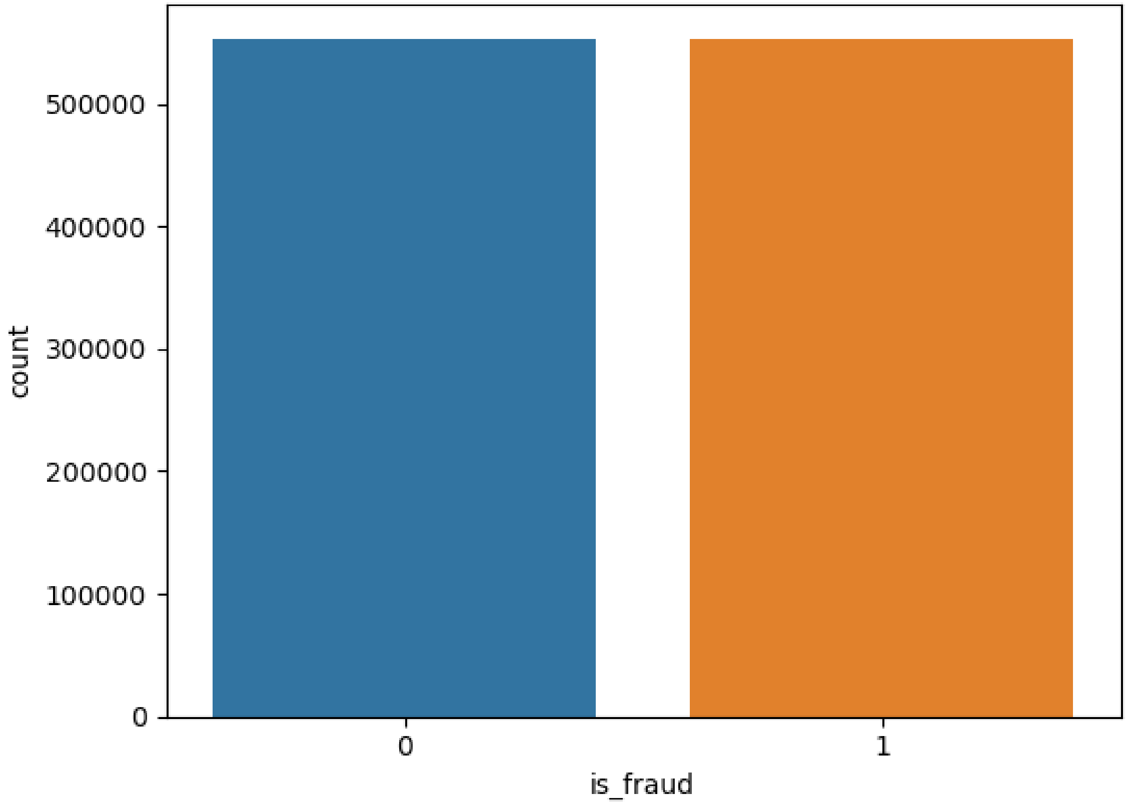
Let's resample the minority class to balance the dataset using resample()

In [13]:
```
df_creditcard_minority_upsampled = resample(df_creditcard_minority, replace=
                                             n_samples=len(df_creditcard_majority),
                                             random_state=42)    # reproducible results
```

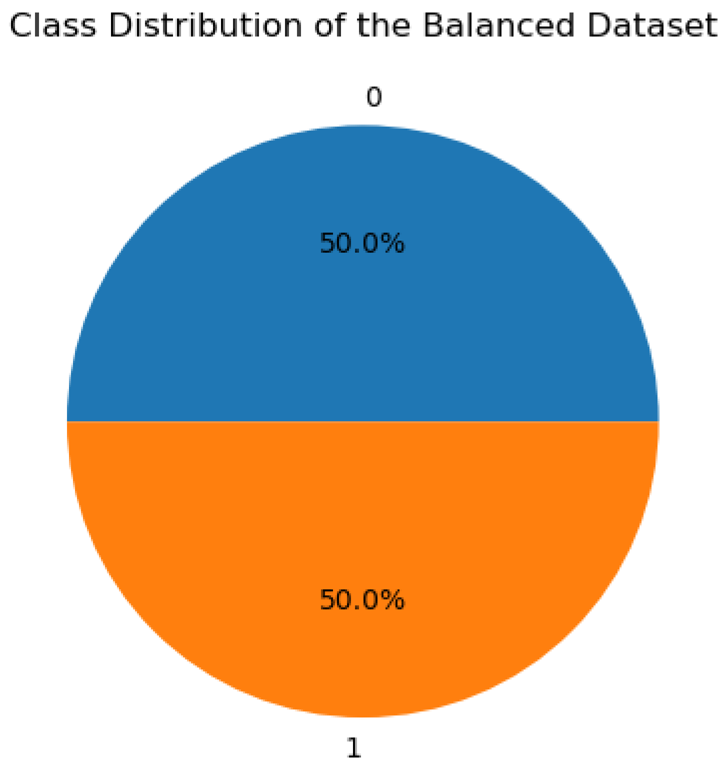
In [14]:
```
#Let's combine the upsampled minority class with the majority class
df_creditcard_balanced = pd.concat([df_creditcard_majority, df_creditcard_mi
```

In [15]:
```
#lets verify and see if the dataset is now balanced
sns.countplot(x= df_creditcard_balanced["is_fraud"])
```

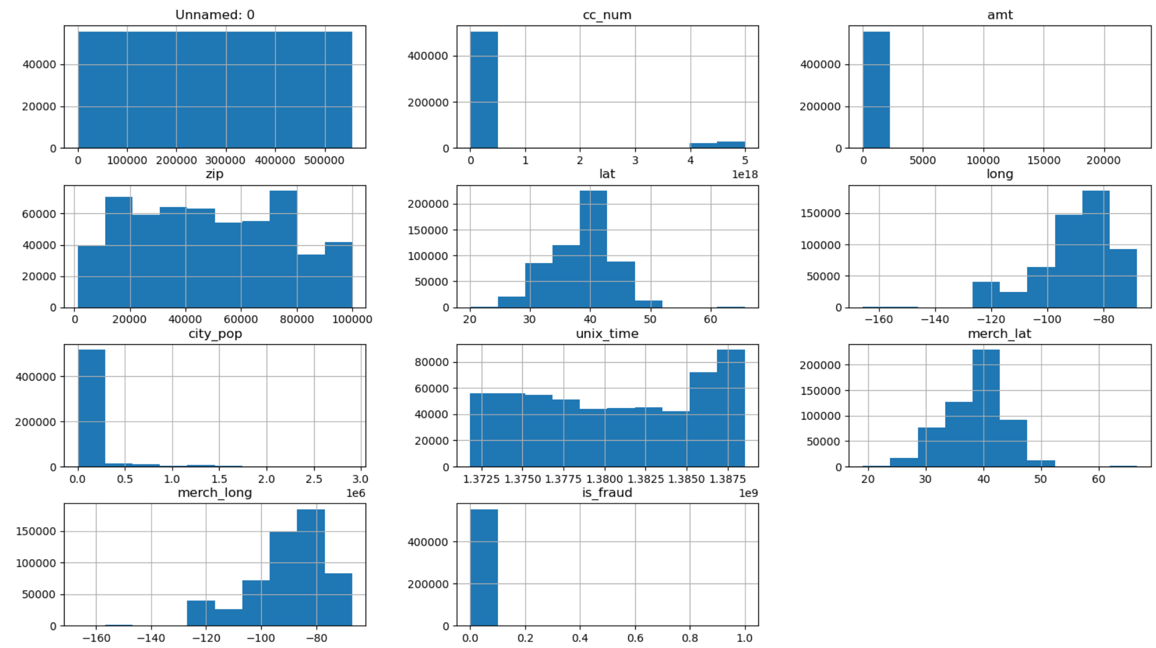
Out[15]: <Axes: xlabel='is_fraud', ylabel='count'>



In [16]:
```
# lets visualize the balanced dataset using a pie chart
class_counts_balanced = df_creditcard_balanced['is_fraud'].value_counts()
plt.pie(class_counts_balanced, labels=class_counts_balanced.index, autopct='
plt.title('Class Distribution of the Balanced Dataset')
plt.show()
```



In [17]:
```
df_creditcard.hist(figsize=(18,  10));
# Histogram to show the skewness in the dataset
```



Feature engineering

The columns like 'cc_num', 'first', 'last', 'trans_num' don't provide significant relevant information related to fraud detection. So, we drop it.

In [18]:
```
print(df_creditcard.columns)
df_creditcard.drop(['Unnamed: 0','first','last','trans_num','job'],axis = 1,
print(df_creditcard.columns)
```

```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'categ
ory',
       'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip',
       'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud'],
      dtype='object')
Index(['trans_date_trans_time', 'cc_num', 'merchant', 'category', 'amt',
       'gender', 'street', 'city', 'state', 'zip', 'lat', 'long', 'city_po
p',
       'dob', 'unix_time', 'merch_lat', 'merch_long', 'is_fraud'],
      dtype='object')
```

Let's convert the transaction date and time to separate columns which includes hour, day of a week and month

In [19]:
```
df_creditcard['trans_date_trans_time'] = pd.to_datetime(df_creditcard['trans
df_creditcard['Hour'] = df_creditcard['trans_date_trans_time'].dt.hour
df_creditcard['Day_of_week']= df_creditcard['trans_date_trans_time'].dt.dayo
df_creditcard['Month']=df_creditcard['trans_date_trans_time'].dt.month
```

In [20]:
```
df_creditcard['dob']=pd.to_datetime(df_creditcard['dob'])
df_creditcard['dob']
```

Out[20]:
```
0         1968-03-19
1         1990-01-17
2         1970-10-21
3         1987-07-25
4         1955-07-06
             ...
555714    1966-02-13
555715    1999-12-27
555716    1981-11-29
555717    1965-12-15
555718    1993-05-10
Name: dob, Length: 555719, dtype: datetime64[ns]
```

Let's look at the Frequency of Transactions

In [21]:
```
def last1dayTransCnt(df_creditcard):
    temp = pd.Series(df_creditcard.index,index=df_creditcard.trans_date_trar
    #data (parameter) is df_creditcard.index
    #temp is a series whose index is time stamp and value is row indices of
    In_a_Day = temp.rolling('1d').count()-1
#in a day is a series with timestamp as index and frequency as its value
    In_a_Day.index= temp.values
#in a day 's index is just 0 1 2; row indices of df_creditcard or x
    df_creditcard['In_a_Day'] = In_a_Day.reindex(df_creditcard.index)
#df_creditcard
    return df_creditcard
```

In [22]:
```
def last1weekTransCnt(x):
    temp = pd.Series(x.index,index=x.trans_date_trans_time,name="In_a_Week")
    In_a_Week = temp.rolling('7d').count()-1
    In_a_Week.index = temp.values
    x['In_a_Week'] = In_a_Week.reindex(x.index)
    return x
```

In [23]:
```
def last1monthTransCnt(x):
    temp = pd.Series(x.index,index=x.trans_date_trans_time,name="In_a_Month'
    In_a_Month = temp.rolling('30d').count()-1
    In_a_Month.index = temp.values
    x['In_a_Month'] = In_a_Month.reindex(x.index)
    return x
```

In [24]:
```
df1d = df_creditcard.groupby('cc_num').apply(last1dayTransCnt)
#drop = true ; we don't want to add new column
df1w = df1d.reset_index(drop=True).groupby('cc_num').apply(last1weekTransCnt
df1dm = df1w.reset_index(drop=True).groupby('cc_num').apply(last1monthTrans(
df1dm
```

```
C:\Users\c2108436\AppData\Local\Temp\ipykernel_5912\3752136529.py:1: Futur
eWarning: Not prepending group keys to the result index of transform-like
apply. In the future, the group keys will be included in the index, regard
less of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  df1d = df_creditcard.groupby('cc_num').apply(last1dayTransCnt)
C:\Users\c2108436\AppData\Local\Temp\ipykernel_5912\3752136529.py:3: Futur
eWarning: Not prepending group keys to the result index of transform-like
apply. In the future, the group keys will be included in the index, regard
less of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  df1w = df1d.reset_index(drop=True).groupby('cc_num').apply(last1weekTran
sCnt)
C:\Users\c2108436\AppData\Local\Temp\ipykernel_5912\3752136529.py:4: Futur
eWarning: Not prepending group keys to the result index of transform-like
apply. In the future, the group keys will be included in the index, regard
less of whether the applied function returns a like-indexed object.
To preserve the previous behavior, use

        >>> .groupby(..., group_keys=False)

To adopt the future behavior and silence this warning, use

        >>> .groupby(..., group_keys=True)
  df1dm = df1w.reset_index(drop=True).groupby('cc_num').apply(last1monthTr
ansCnt)
```

Out[24]:

	trans_date_trans_time	cc_num	merchant	category	amt	ger
0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	
1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	
2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	
3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	
4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	
...	...	...	...	...	...	
555714	2020-12-31 23:59:07	30560609640617	fraud_Reilly and Sons	health_fitness	43.77	
555715	2020-12-31 23:59:09	3556613125071656	fraud_Hoppe-Parisian	kids_pets	111.84	
555716	2020-12-31 23:59:15	6011724471098086	fraud_Rau-Robel	kids_pets	86.88	
555717	2020-12-31 23:59:24	4079773899158	fraud_Breitenberg LLC	travel	7.99	
555718	2020-12-31 23:59:34	4170689372027579	fraud_Dare-Marvin	entertainment	38.13	

555719 rows × 24 columns

As per the frequency of transactions lets guess the fraudulent transactions

Threshold for a certain transaction to be fraudulent is estimated if no. of day, week or month is more that 90% of the data.

In [25]:
```
threshold_day = df1dm['In_a_Day'].quantile(0.9)
threshold_week = df1dm['In_a_Week'].quantile(0.9)
threshold_month = df1dm['In_a_Month'].quantile(0.9)
df1dm['prolly_fraud'] = ((df1dm['In_a_Day']>threshold_day)|
                        (df1dm['In_a_Week']>threshold_week)|
                        (df1dm['In_a_Month']>threshold_month))
df1dm[(df1dm['prolly_fraud'] == True) & (df1dm['is_fraud'] == True)]
```

Out[25]:

	trans_date_trans_time	cc_num	merchant	category	amt	genc
11799	2020-06-24 23:24:22	180098888332620	fraud_Pfeffer and Sons	shopping_pos	1047.30	
26607	2020-06-30 03:16:01	3586008444788268	fraud_Wilkinson Ltd	entertainment	483.28	
26696	2020-06-30 03:59:50	3586008444788268	fraud_Effertz, Welch and Schowalter	entertainment	520.02	
28628	2020-06-30 16:42:15	3586008444788268	fraud_Schaefer Ltd	kids_pets	19.68	
30010	2020-06-30 23:46:30	3528231451607350	fraud_Torphy-Goyette	shopping_pos	727.32	
...	...	...	...	...	...	
505774	2020-12-21 02:21:41	4716561796955522	fraud_Murray-Smitham	grocery_pos	358.24	
505826	2020-12-21 02:36:03	4716561796955522	fraud_Schmidt and Sons	shopping_net	859.12	
511244	2020-12-21 22:38:38	4716561796955522	fraud_Quitzon-Goyette	home	209.84	
511272	2020-12-21 22:42:11	4716561796955522	fraud_Schulist Ltd	food_dining	123.58	
511374	2020-12-21 22:59:22	4716561796955522	fraud_Botsford and Sons	home	219.11	

184 rows × 25 columns

Merged this dataframe's columns to original one

In [26]:
```
"""df1dm has cc_num as index but we're trying to merge on basis of cc_num cc
so we make cc_num a regular column before merging it"""
df1dm.reset_index(drop =True,inplace=True)
df_creditcard = df_creditcard.merge(df1dm[['trans_date_trans_time','prolly_f
# df.drop(['prolly_fraud_x','prolly_fraud_y'], axis=1, inplace=True)
# df_creditcard.head(3)
```

In [27]:
```
df_creditcard.head()
```

Out[27]:

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

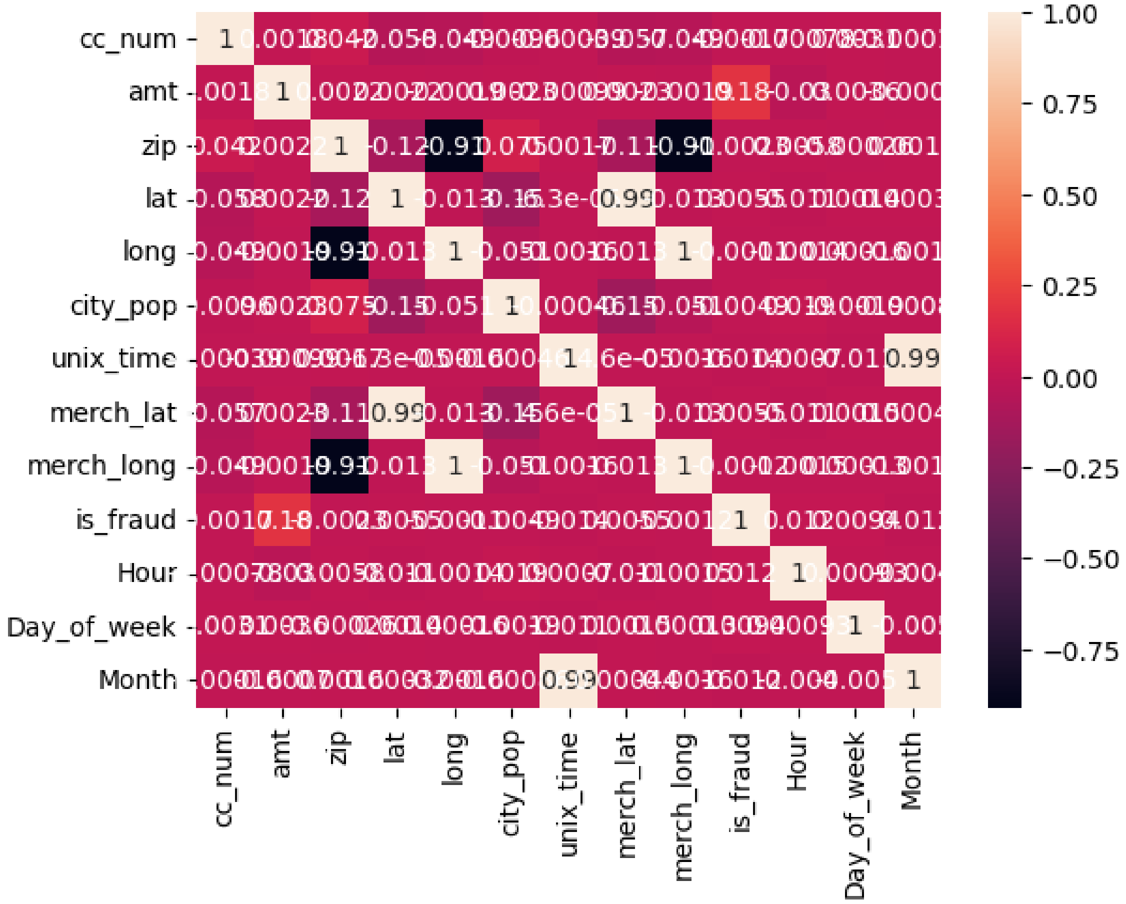
5 rows × 22 columns

Correlation Heatmaps

Correlation heatmap of original data

In [28]:
```
num_cols = df_creditcard.select_dtypes(include = ['float64','int64'])
sns.heatmap(num_cols.corr(),annot = True)
```

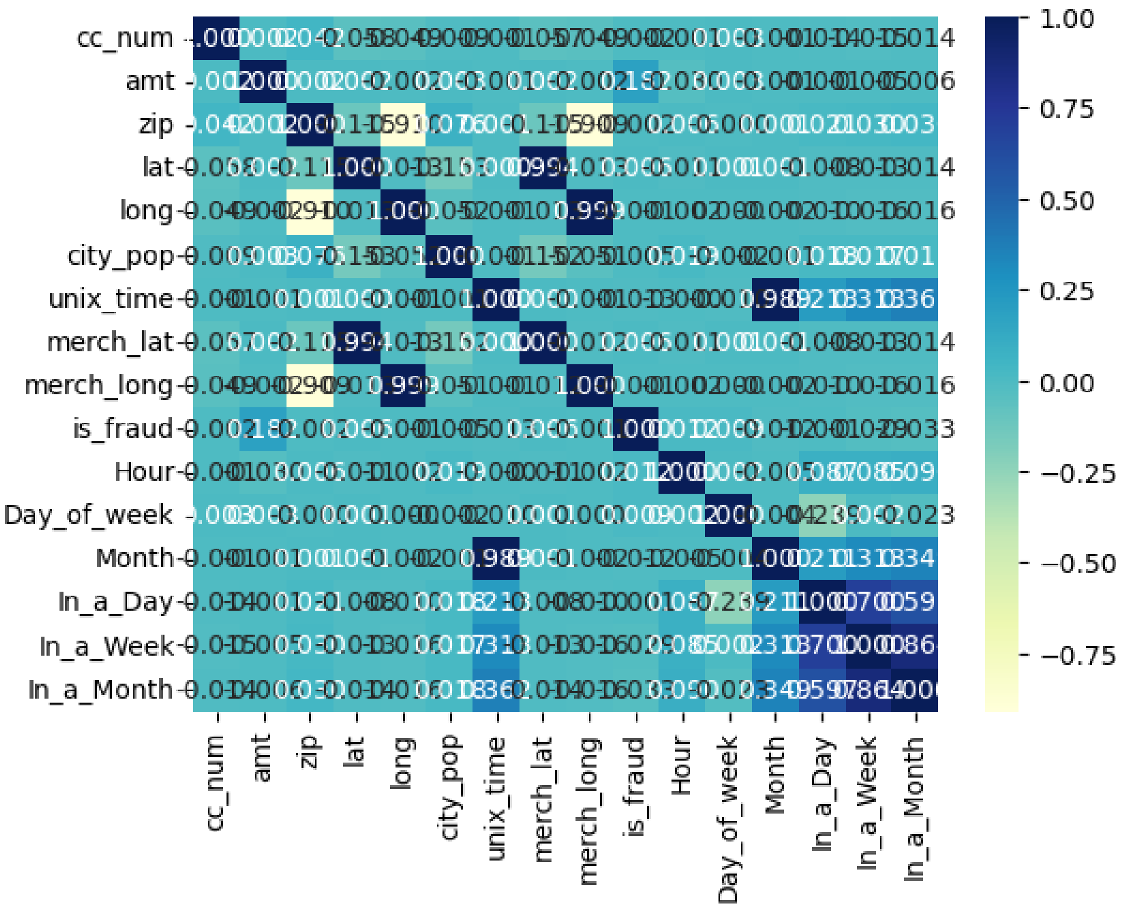
Out[28]: <Axes: >



Correlation heatmap of data with added features

In [29]:
```
numeric_columns = df1dm.select_dtypes(include=['float64', 'int64'])
corr_matrix = numeric_columns.corr()
sns.heatmap(corr_matrix, annot=True,fmt='.3f',cmap="YlGnBu")
```

Out[29]: <Axes: >



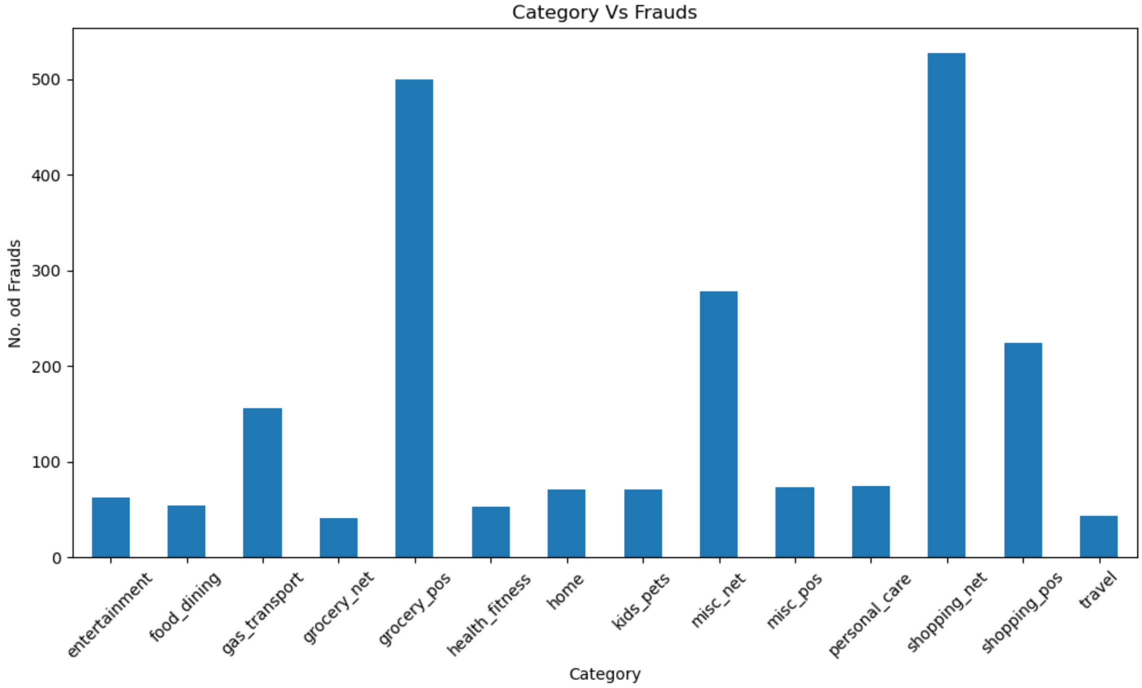
Frauds and Categories

In [30]:
```
""""
split-apply-combine
first it groups category and is_fraud then it selects is_fraud and counts it
results in
        is_fraud
cat1    0       2
        1       1
cat2    0       1
        1       1
unstacked so the 'is_fraud' gets converted into a single column
is_fraud    0    1
category
cat1        2    1
cat2        1    1
"""
cat_counts = df_creditcard.groupby(['category','is_fraud'])['is_fraud'].cour
cat_counts
```

Out[30]:

is_fraud	0	1
category		
entertainment	41688	63
food_dining	41055	54
gas_transport	57894	156
grocery_net	19953	41
grocery_pos	53645	500
health_fitness	38377	53
home	54701	71
kids_pets	50896	71
misc_net	27983	278
misc_pos	35769	73
personal_care	41125	74
shopping_net	42933	527
shopping_pos	51585	224
travel	18211	43

In [31]:
```
cat_counts_fraud = cat_counts[1]
ccc = cat_counts_fraud.plot(kind='bar', figsize=(12, 6))
ccc.set_ylabel('No. od Frauds')
ccc.set_xlabel('Category')
ccc.set_title('Category Vs Frauds')
plt.xticks(rotation=45)
plt.show()
```

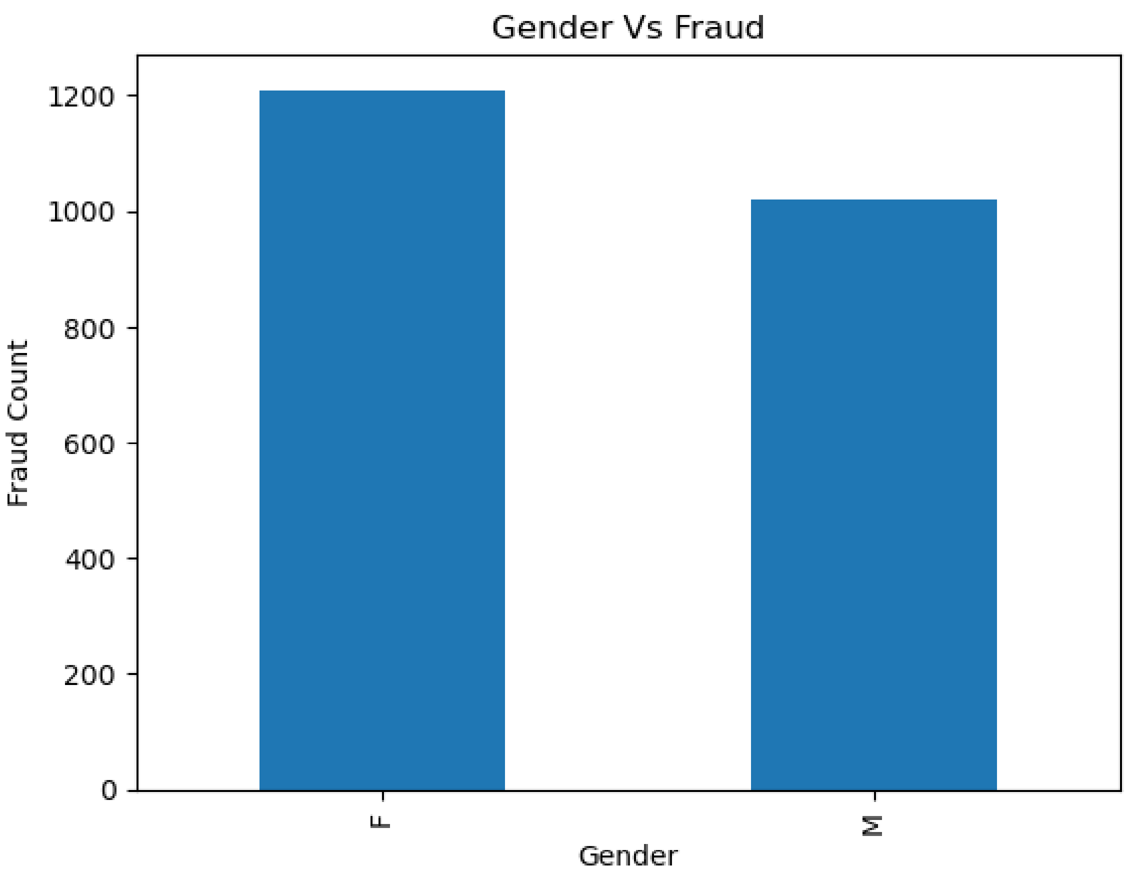


Grocery and Shopping are the categories with the most frauds.

Fraud by Gender

In [32]:
```
gen_counts=df_creditcard.groupby(['gender','is_fraud'])['is_fraud'].count()
gen_counts = df_creditcard[df_creditcard['is_fraud'] == 1].groupby('gender')
ax = gen_counts.plot(kind='bar')
ax.set_xlabel('Gender')
ax.set_ylabel('Fraud Count')
ax.set_title('Gender Vs Fraud')
gen_counts
```

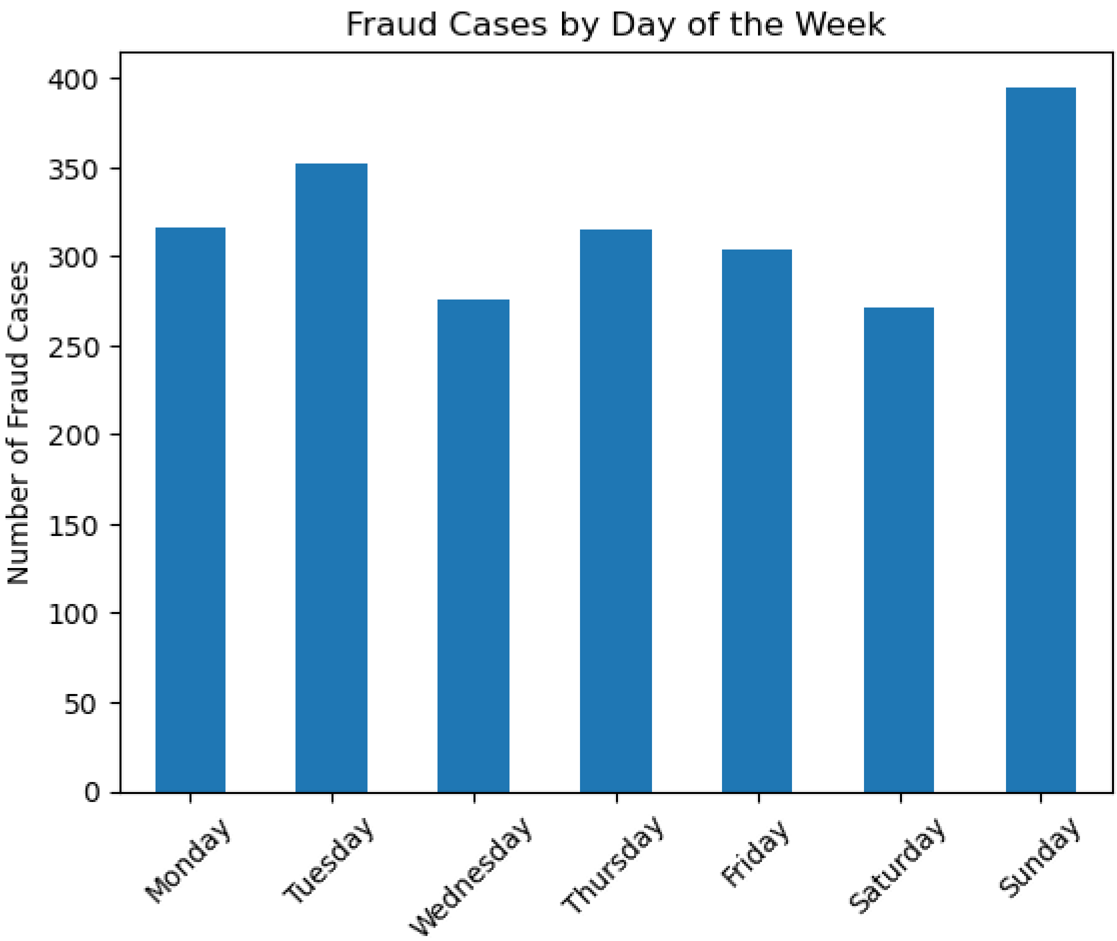
Out[32]:
```
gender
F    1209
M    1019
Name: is_fraud, dtype: int64
```



The bar chart above shows that females are more involved in credit card fraud

Most common day of the week for fraud

In [33]:
```
fraud_by_day = df_creditcard[df_creditcard['is_fraud']==1].groupby('Day_of_w
day_labels = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Satur
fraud_by_day.index=(day_labels)
fraud_by_day.plot(kind='bar')
plt.ylabel('Number of Fraud Cases')
plt.title('Fraud Cases by Day of the Week')
plt.xticks(rotation = 45)
plt.show()
```



Most fraud transactions occur during the weekend, especially on Sunday

Zip codes based fraud frequency

In [34]:
```
df_zip = df_creditcard[df_creditcard['is_fraud'] == 1].groupby('zip')['is_fr
top_10_zip= df_zip.sort_values(ascending=False).head(10)
top_10_zip
```

Out[34]:
```
zip
67020    19
16114    18
29819    17
12037    17
58275    16
69165    16
19007    16
61454    16
6365     16
29127    16
Name: is_fraud, dtype: int64
```

City Vs Fraud

In [35]:
```
df_city = df_creditcard[df_creditcard['is_fraud'] == 1].groupby('city')['is_
top_10_city= df_city.sort_values(ascending=False).head(10)
top_10_city
```

Out[35]:
```
city
Camden          29
Birmingham      26
Burrton         19
Clarks Mills    18
Chatham         17
Bradley         17
Preston         16
Heislerville    16
Bristol         16
Jay             16
Name: is_fraud, dtype: int64
```

Camden has the highest number of frauds occurence.

Merchants Vs Fraud

In [36]:
```
df_mer = df_creditcard[df_creditcard['is_fraud'] == 1].groupby('merchant')['
top_10_mer= df_mer.sort_values(ascending=False).head(10)
top_10_mer
```

Out[36]:
```
merchant
fraud_Romaguera, Cruickshank and Greenholt    19
fraud_Lemke-Gutmann                           19
fraud_Mosciski, Ziemann and Farrell           19
fraud_Medhurst PLC                            18
fraud_Schultz, Simonis and Little             17
fraud_Heathcote, Yost and Kertzmann           17
fraud_Miller-Hauck                            16
fraud_Heathcote LLC                           16
fraud_Kilback LLC                             15
fraud_Wolf Inc                                15
Name: is_fraud, dtype: int64
```

Dropping some more features, and encoding categorical features and scaling with numeric values

In [37]:
```
X = df_creditcard.drop(['zip','lat','long','unix_time','merch_lat','merch_lo
X= pd.get_dummies(X,columns=['merchant', 'category', 'gender', 'street', 'ci
```

In [38]:
```
from sklearn.preprocessing import MinMaxScaler
num_colss= ['amt', 'city_pop', 'Hour', 'Day_of_week', 'Month']
scaler = MinMaxScaler()
X[num_colss] = scaler.fit_transform(X[num_colss])
```

In [39]:
```
y = X["is_fraud"]
X = X.drop("is_fraud",axis=1)
```

Splitting into test and train data

In [40]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.4,ranc
```

Decision Tree Classifier

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
```

In [55]:
```
dt.fit(X_train,y_train)
```

Out[55]:
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [56]:
```
from sklearn.metrics import accuracy_score, mean_absolute_error ,mean_square

print("Score of X-train with Y-train is : ", dt.score(X_train,y_train))
print("Score of X-test  with Y-test  is : ", dt.score(X_test,y_test))

y_pred=dt.predict(X_test)

print("Accuracy score " , accuracy_score(y_test,y_pred))

print("F1 score: ", round(f1_score(y_test, y_pred, average='weighted')*100,2

print('Decision Tree:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
print('ROC AUC Score:')
y_prob = dt.predict_proba(X_test)[:, 1]
print(roc_auc_score(y_test, y_prob))
```

```
Score of X-train with Y-train is :  1.0
Score of X-test  with Y-test  is :  0.9978894376735375
Accuracy score  0.9978894376735375
F1 score:  99.78 %
Decision Tree:
Accuracy: 0.9978894376735375
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    230360
           1       0.74      0.67      0.70       858

    accuracy                           1.00    231218
   macro avg       0.87      0.83      0.85    231218
weighted avg       1.00      1.00      1.00    231218

Confusion Matrix:
[[230157    203]
 [   285    573]]
ROC AUC Score:
0.8334754692260337
```

Logistic Regression

In [44]:

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print('Logistic Regression:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
print('ROC AUC Score:')
y_prob = logreg.predict_proba(X_test)[:, 1]
print(roc_auc_score(y_test, y_prob))
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.
py:444: ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sc
ikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-reg
ression (https://scikit-learn.org/stable/modules/linear_model.html#logisti
c-regression)
  n_iter_i = _check_optimize_result(

Logistic Regression:
Accuracy: 0.9962892162374901
Classification Report:

C:\ProgramData\anaconda3\lib\site-packages\sklearn\metrics\_classificatio
n.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\lib\site-packages\sklearn\metrics\_classificatio
n.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\ProgramData\anaconda3\lib\site-packages\sklearn\metrics\_classificatio
n.py:1334: UndefinedMetricWarning: Precision and F-score are ill-defined a
nd being set to 0.0 in labels with no predicted samples. Use `zero_divisio
n` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    230360
           1       0.00      0.00      0.00       858

    accuracy                           1.00    231218
   macro avg       0.50      0.50      0.50    231218
weighted avg       0.99      1.00      0.99    231218

Confusion Matrix:
[[230360      0]
 [   858      0]]
ROC AUC Score:
0.5
```

In [47]:
```
# Random Forest
rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print('Random Forest:')
print('Accuracy:', accuracy_score(y_test, y_pred))
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
print('ROC AUC Score:')
y_prob = rf.predict_proba(X_test)[:, 1]
print(roc_auc_score(y_test, y_prob))
```

```
Random Forest:
Accuracy: 0.998209481960747
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    230360
           1       0.98      0.53      0.69       858

    accuracy                           1.00    231218
   macro avg       0.99      0.76      0.84    231218
weighted avg       1.00      1.00      1.00    231218

Confusion Matrix:
[[230350     10]
 [   404    454]]
ROC AUC Score:
0.9826292919039056
```