Problem statement: Imagine you are working for a financial institution, and your task is to detect anomalies in financial transactions to identify potential fraudulent activities. You are provided with a dataset containing various parameters related to financial transactions. Your goal is to design an anomaly detection model to flag suspicious transactions.

Importing libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [13]:
```
df = pd.read_csv("C:/Users/91798/Desktop/step_change/Anomaly_detection - Sheet1.csv
```

Data summary

In [14]:
```
df.head(5)
```

Out[14]:

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location
0	01-01-2023 8:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	Tokyo
1	01-01-2023 8:01	TXN1639	ACC10	15607.89	MerchantH	Purchase	London
2	01-01-2023 8:02	TXN872	ACC8	65092.34	MerchantE	Withdrawal	London
3	01-01-2023 8:03	TXN1438	ACC6	87.87	MerchantE	Purchase	London
4	01-01-2023 8:04	TXN1338	ACC6	716.56	MerchantI	Purchase	Los Angeles

In [116…
```
df.isnull().sum()
```

Out[116]:
```
Timestamp          0
TransactionID      0
AccountID          0
Amount             0
Merchant           0
TransactionType    0
Location           0
dtype: int64
```

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 999 entries, 0 to 998
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Timestamp        999 non-null    object 
 1   TransactionID    999 non-null    object 
 2   AccountID        999 non-null    object 
 3   Amount           999 non-null    float64
 4   Merchant         999 non-null    object 
 5   TransactionType  999 non-null    object 
 6   Location         999 non-null    object 
dtypes: float64(1), object(6)
memory usage: 54.8+ KB
```

In [20]:
```
# converting Timestamp to datetime format
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
```

In [26]:
```
print("Number of rows and column in the data is: ",df.shape)
print('----------------------------------------')
print("Number of unique transaction_ID are:",df.TransactionID.nunique())
```

```
Number of rows and column in the data is:  (999, 7)
----------------------------------------
Number of unique transaction_ID are: 764
```

In [28]:
```
df.describe().T
```

Out[28]:

	count	mean	std	min	25%	50%	75%	max
Amount	999.0	49672.674494	28851.580582	11.16	25040.06	50598.11	74054.64	99486.68

In [39]:
```
print('----------------------------------------')
print("number of transactions country wise:\n",df.Location.value_counts())
print('----------------------------------------')
print("number of transactions TransactionType wise:\n",df.TransactionType.value_cou
print('----------------------------------------')
print("number of transactions Merchant wise:\n",df['Merchant'].value_counts().sort_
print('----------------------------------------')
```

```
----------------------------------------
number of transactions country wise:
 London           220
Los Angeles      201
Tokyo            196
New York         193
San Francisco    189
Name: Location, dtype: int64
----------------------------------------
number of transactions TransactionType wise:
 Purchase      351
Transfer      344
Withdrawal    304
Name: TransactionType, dtype: int64
----------------------------------------
number of transactions Merchant wise:
 MerchantA     94
MerchantB    102
MerchantC    108
MerchantD    102
MerchantE     88
MerchantF     90
MerchantG    101
MerchantH    117
MerchantI    103
MerchantJ     94
Name: Merchant, dtype: int64
----------------------------------------
```

In [72]:
```
# As noticed earlier there are duplicate TransactionID in the data, based on the ca
unique_id = pd.DataFrame(df.TransactionID.value_counts())
(unique_id['TransactionID'] == 1).value_counts()
```

Out[72]:
```
True     565
False    199
Name: TransactionID, dtype: int64
```

Out of 764 unique TransactionID's, 199 TransactionID's have some duplicates, therefore in fraud catagory

In [106…
```
# Nature of duplicate
df.TransactionID.value_counts().head(10)
```

Out[106]:
```
TXN1108    4
TXN1760    4
TXN1849    4
TXN188     4
TXN1425    4
TXN273     4
TXN726     3
TXN649     3
TXN759     3
TXN1137    3
Name: TransactionID, dtype: int64
```

In [114…
```
duplicate_ids = ['TXN1108', 'TXN649', 'TXN1760', 'TXN1137']

df[df['TransactionID'].isin(duplicate_ids)].sort_values(by = 'TransactionID')
```

Out[114]:

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location
321	2023-01-01 13:21:00	TXN1108	ACC2	9535.20	MerchantA	Transfer	New York
384	2023-01-01 14:24:00	TXN1108	ACC6	25712.16	MerchantH	Transfer	New York
703	2023-01-01 19:43:00	TXN1108	ACC6	73351.34	MerchantG	Withdrawal	Los Angeles
818	2023-01-01 21:38:00	TXN1108	ACC3	98530.53	MerchantI	Transfer	Los Angeles
32	2023-01-01 08:32:00	TXN1137	ACC14	97178.49	MerchantD	Withdrawal	San Francisco
201	2023-01-01 11:21:00	TXN1137	ACC2	49706.22	MerchantC	Transfer	London
226	2023-01-01 11:46:00	TXN1137	ACC11	29719.20	MerchantC	Withdrawal	San Francisco
35	2023-01-01 08:35:00	TXN1760	ACC1	63438.79	MerchantB	Purchase	London
198	2023-01-01 11:18:00	TXN1760	ACC13	39737.84	MerchantC	Transfer	Los Angeles
435	2023-01-01 15:15:00	TXN1760	ACC11	32012.78	MerchantF	Transfer	New York
568	2023-01-01 17:28:00	TXN1760	ACC6	54145.35	MerchantE	Purchase	New York
30	2023-01-01 08:30:00	TXN649	ACC14	22218.56	MerchantC	Withdrawal	London
732	2023-01-01 20:12:00	TXN649	ACC2	78740.83	MerchantE	Withdrawal	Tokyo
934	2023-01-01 23:34:00	TXN649	ACC10	50466.86	MerchantD	Transfer	San Francisco

__nature of duplicate is not duplicate entries, but might indicate fraud__

Exploratory data analysis

Location-wise Analysis:Transaction Type Analysis:Merchant-wise Analysis:Location and Transaction Type Interaction:Temporal Analysis:Anomaly Detection Feature Engineering:Outlier Detection:

1. Location-wise Analysis

In [91]:
```
print("number of transactions country wise:\n",df.Location.value_counts().sort_inde
print('-----------------------------------------')
print('Total transation amount based on location: \n',df.groupby('Location')['Amoun
print('-----------------------------------------')
print('Average transation amount based on location: \n',df.groupby('Location')['Amo
print('-----------------------------------------')
```

```
number of transactions country wise:
 London           220
Los Angeles      201
New York         193
San Francisco    189
Tokyo            196
Name: Location, dtype: int64
-----------------------------------------
Total transation amount based on location:
 Location
London           10729320.37
Los Angeles      10106776.42
New York          9294089.05
San Francisco     9336205.04
Tokyo            10156610.94
Name: Amount, dtype: float64
-----------------------------------------
Average transation amount based on location:
 Location
London           48769.638045
Los Angeles      50282.469751
New York         48155.901813
San Francisco    49397.910265
Tokyo            51819.443571
Name: Amount, dtype: float64
-----------------------------------------
```

In [86]:
```
## to check if there is any relation between the duplicate entries and the location
df_no_duplicates = df.drop_duplicates(subset='TransactionID', keep=False)
```

In [88]:
```
df_no_duplicates.shape
```

Out[88]:
```
(565, 7)
```

In [92]:
```
print("number of transactions country wise:\n",df_no_duplicates.Location.value_coun
print('-----------------------------------------')
print('Total transation amount based on location: \n',df_no_duplicates.groupby('Loc
print('-----------------------------------------')
print('Average transation amount based on location: \n',df_no_duplicates.groupby('L
print('-----------------------------------------')
```

```
number of transactions country wise:
 London           112
Los Angeles      117
New York         112
San Francisco    105
Tokyo            119
Name: Location, dtype: int64
----------------------------------------
Total transation amount based on location:
 Location
London           5460427.91
Los Angeles      5617590.98
New York         5454462.37
San Francisco    5235190.92
Tokyo            6303536.14
Name: Amount, dtype: float64
----------------------------------------
Average transation amount based on location:
 Location
London           48753.820625
Los Angeles      48013.598120
New York         48700.556875
San Francisco    49858.961143
Tokyo            52970.891933
Name: Amount, dtype: float64
----------------------------------------
```

there is no clear relation between the location and duplicate TransactionID, all locations are equally affected

2. Transaction Type Analysis

In [96]:
```
print("Transaction types:\n",df.TransactionType.value_counts().sort_index())
print('----------------------------------------')
print('Total transation amount based on transaction types: \n',df.groupby('Transact
print('----------------------------------------')
print('Average transation amount based on transaction types: \n',df.groupby('Transa
print('----------------------------------------')
```

```
Transaction types:
 Purchase      351
Transfer      344
Withdrawal    304
Name: TransactionType, dtype: int64
----------------------------------------
Total transation amount based on transaction types:
 TransactionType
Purchase      17441962.92
Transfer      17095256.27
Withdrawal    15085782.63
Name: Amount, dtype: float64
----------------------------------------
Average transation amount based on transaction types:
 TransactionType
Purchase      49692.202051
Transfer      49695.512413
Withdrawal    49624.284967
Name: Amount, dtype: float64
----------------------------------------
```

In [97]:
```
print("Transaction types:\n",df_no_duplicates.TransactionType.value_counts().sort_i
print('----------------------------------------')
print('Total transation amount based on transaction types: \n',df_no_duplicates.gro
print('----------------------------------------')
print('Average transation amount based on transaction types: \n',df_no_duplicates.g
print('----------------------------------------')
```

```
Transaction types:
 Purchase      199
Transfer      191
Withdrawal    175
Name: TransactionType, dtype: int64
----------------------------------------
Total transation amount based on transaction types:
 TransactionType
Purchase      9558033.03
Transfer      9830344.57
Withdrawal    8682830.72
Name: Amount, dtype: float64
----------------------------------------
Average transation amount based on transaction types:
 TransactionType
Purchase      48030.316734
Transfer      51467.772618
Withdrawal    49616.175543
Name: Amount, dtype: float64
----------------------------------------
```

3. Merchant-wise Analysis

In [98]:
```
print("Merchant types:\n",df.Merchant.value_counts().sort_index())
print('----------------------------------------')
print('Total transation amount based on Merchant types: \n',df.groupby('Merchant')[
print('----------------------------------------')
```

```
print('Average transation amount based on Merchant types: \n',df.groupby('Merchant'
print('----------------------------------------')
```

```
Merchant types:
 MerchantA     94
MerchantB    102
MerchantC    108
MerchantD    102
MerchantE     88
MerchantF     90
MerchantG    101
MerchantH    117
MerchantI    103
MerchantJ     94
Name: Merchant, dtype: int64
----------------------------------------
Total transation amount based on Merchant types:
 Merchant
MerchantA    4749490.18
MerchantB    5233276.03
MerchantC    4928139.63
MerchantD    5432823.53
MerchantE    4387177.03
MerchantF    4669130.53
MerchantG    5092845.41
MerchantH    5926359.58
MerchantI    5103138.82
MerchantJ    4100621.08
Name: Amount, dtype: float64
----------------------------------------
Average transation amount based on Merchant types:
 Merchant
MerchantA    50526.491277
MerchantB    51306.627745
MerchantC    45630.922500
MerchantD    53262.975784
MerchantE    49854.284432
MerchantF    51879.228111
MerchantG    50424.211980
MerchantH    50652.645983
MerchantI    49545.037087
MerchantJ    43623.628511
Name: Amount, dtype: float64
----------------------------------------
```

In [99]:
```
print("Merchant types:\n",df_no_duplicates.Merchant.value_counts().sort_index())
print('----------------------------------------')
print('Total transation amount based on Merchant types: \n',df_no_duplicates.groupb
print('----------------------------------------')
print('Average transation amount based on Merchant types: \n',df_no_duplicates.grou
print('----------------------------------------')
```

```
Merchant types:
 MerchantA    54
MerchantB    54
MerchantC    71
MerchantD    57
MerchantE    45
MerchantF    43
MerchantG    58
MerchantH    69
MerchantI    54
MerchantJ    60
Name: Merchant, dtype: int64
----------------------------------------
Total transation amount based on Merchant types:
 Merchant
MerchantA    2780624.39
MerchantB    3128872.29
MerchantC    3198325.07
MerchantD    3056113.54
MerchantE    2231685.63
MerchantF    2310181.93
MerchantG    2835190.45
MerchantH    3355084.42
MerchantI    2706961.18
MerchantJ    2468169.42
Name: Amount, dtype: float64
----------------------------------------
Average transation amount based on Merchant types:
 Merchant
MerchantA    51493.044259
MerchantB    57942.079444
MerchantC    45046.831972
MerchantD    53616.027018
MerchantE    49593.014000
MerchantF    53725.161163
MerchantG    48882.593966
MerchantH    48624.411884
MerchantI    50128.910741
MerchantJ    41136.157000
Name: Amount, dtype: float64
----------------------------------------
```

6. AcountID

In [136…
```
df.AccountID.nunique()
```

Out[136]:
```
15
```

In [139…
```
df.AccountID.value_counts().sort_index()
```

Out[139]:
```
ACC1     68
ACC10    63
ACC11    73
ACC12    62
ACC13    66
ACC14    68
ACC15    75
ACC2     62
ACC3     63
ACC4     63
ACC5     76
ACC6     75
ACC7     63
ACC8     59
ACC9     63
Name: AccountID, dtype: int64
```

5. Temporal Analysis

In [104…
```
print('starting date: ',df['Timestamp'].min())
print('----------------------------------------')
print('ending date: ',df['Timestamp'].max())
```

```
starting date:  2023-01-01 08:00:00
----------------------------------------
ending date:  2023-02-01 00:38:00
```

In [130…
```
print('----------------------------------------')
print('Number of year of data: ',df['Timestamp'].dt.year.nunique())
print('----------------------------------------')
print('Number of months of data: ',df['Timestamp'].dt.month.nunique())
print('----------------------------------------')
print('Number of days of data: ',df['Timestamp'].dt.strftime('%A').nunique())
```

```
----------------------------------------
Number of year of data:  1
----------------------------------------
Number of months of data:  2
----------------------------------------
Number of days of data:  2
```

In [131…
```
print('----------------------------------------')
print('Number of year of data: ',df_no_duplicates['Timestamp'].dt.year.nunique())
print('----------------------------------------')
print('Number of months of data: ',df_no_duplicates['Timestamp'].dt.month.nunique()
print('----------------------------------------')
print('Number of days of data: ',df_no_duplicates['Timestamp'].dt.strftime('%A').nu
```

```
----------------------------------------
Number of year of data:  1
----------------------------------------
Number of months of data:  2
----------------------------------------
Number of days of data:  2
```

In [134…
```
## Lets see if hourly there is any relation between the fraud activity
df['Timestamp'].dt.hour.value_counts().sort_index()
```

Out[134]:
```
0     39
8     60
9     60
10    60
11    60
12    60
13    60
14    60
15    60
16    60
17    60
18    60
19    60
20    60
21    60
22    60
23    60
Name: Timestamp, dtype: int64
```

In [135…
```
df_no_duplicates['Timestamp'].dt.hour.value_counts().sort_index()
```

Out[135]:
```
0     15
8     36
9     37
10    33
11    32
12    38
13    36
14    31
15    31
16    33
17    37
18    32
19    30
20    35
21    33
22    34
23    42
Name: Timestamp, dtype: int64
```

Feature addition

In [190…
```
df_ = df.copy()
df_.drop(['Timestamp'], axis=1, inplace=True)
df_ = pd.get_dummies(df_, columns=['Location', 'TransactionType','AccountID','Merch
```

In [191…
```
df_['Label'] = df_.duplicated(subset='TransactionID', keep=False).astype(int)
df_['Label'] = df_['Label'].apply(lambda x: 1 if x == 0 else -1)
```

In [192…
```
df_.drop(['TransactionID'], axis=1, inplace=True)
```

In [193…
```
df_.Label.value_counts()
```

Out[193]:
```
 1    565
-1    434
Name: Label, dtype: int64
```

-1 represents fraud activity, and 1 as normal

In [194…
```
df_
```

Out[194]:

	Amount	Location_Los Angeles	Location_New York	Location_San Francisco	Location_Tokyo	TransactionType_Transfe
0	95071.92	0	0	0	1	0
1	15607.89	0	0	0	0	0
2	65092.34	0	0	0	0	0
3	87.87	0	0	0	0	0
4	716.56	1	0	0	0	0
…	…	…	…	…	…	…
994	14355.51	0	1	0	0	0
995	75515.81	0	0	0	0	0
996	31507.89	0	0	0	0	0
997	27714.15	0	1	0	0	0
998	98775.08	0	0	0	0	0

999 rows × 31 columns

In [195…
```
## Get the Fraud and the normal dataset

Fraud = df_[df_['Label']==1]

Valid = df_[df_['Label']==-1]

outlier_fraction = len(Fraud)/float(len(Valid))
```

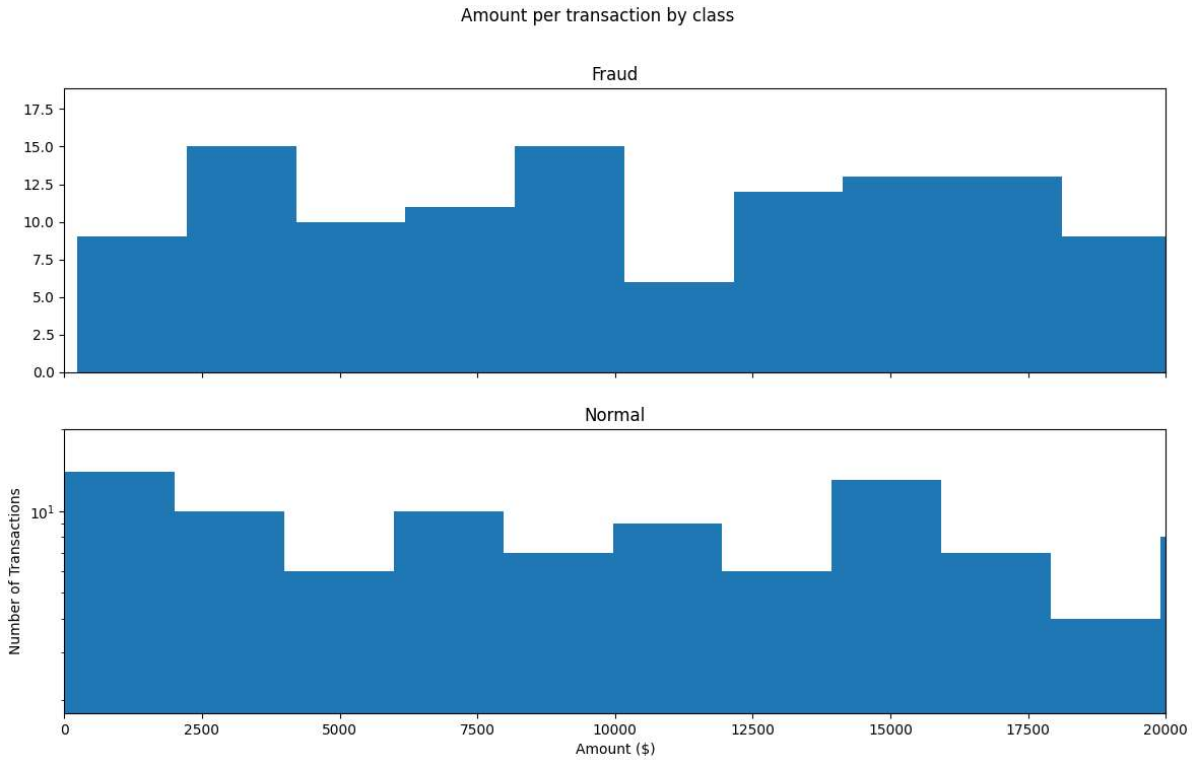
In [196…
```
fraud.Amount.describe()
```

Out[196]:
```
count      565.000000
mean     49683.554549
std      28649.238382
min        241.730000
25%      24984.720000
50%      50272.880000
75%      74427.520000
max      99486.680000
Name: Amount, dtype: float64
```

In [197…
```
Valid.Amount.describe()
```

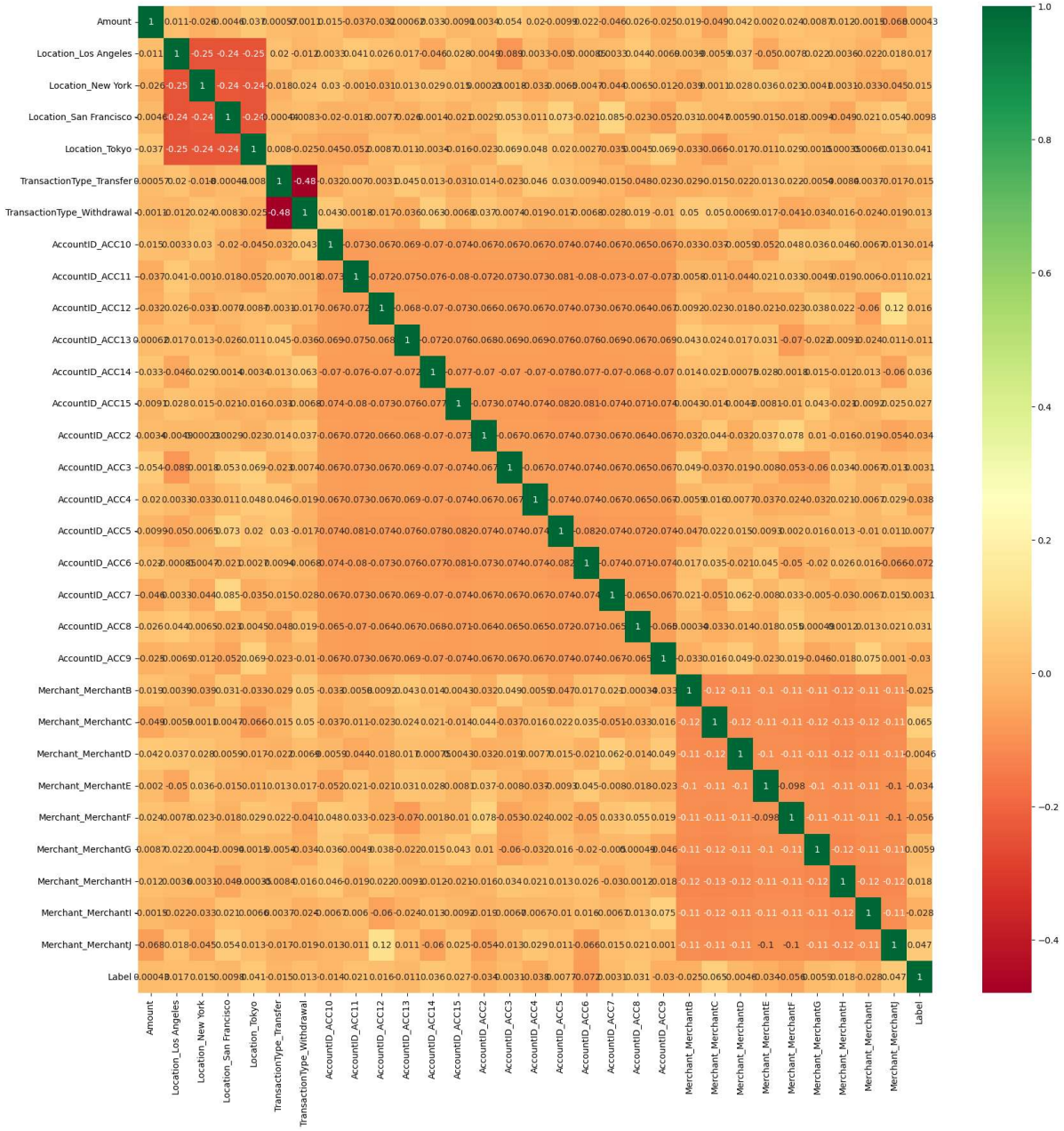
Out[197]:
```
count      434.000000
mean     49658.510369
std      29146.022580
min         11.160000
25%      25185.447500
50%      51330.140000
75%      73783.552500
max      99481.980000
Name: Amount, dtype: float64
```

In [198…
```
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```



In [199…

```
#get correlations of each features in dataset
corrmat = df_.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(df_[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [200…

```
#Create independent and Dependent Features
columns = df_.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Label"]]
# Store the variable we are predicting
target = "Label"
# Define a random state
state = np.random.RandomState(42)
X = df_[columns]
Y = df_[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))
# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(999, 30)
(999,)
```

Model predection

Using two models, and comparing the reasults:

1. Isolation Forest Algorithm
2. Local Outlier Factor(LOF) Algorithm

In [201…
```
import sklearn
import scipy

from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
```

In [208…
```
##Define the outlier detection methods

classifiers = {
    "Isolation Forest":IsolationForest(n_estimators=100, max_samples=len(X),
                                       contamination= 0.5,random_state=state, verbo
    "Local Outlier Factor":LocalOutlierFactor(n_neighbors=12, algorithm='auto',
                                              leaf_size=30, metric='minkowski',
                                              p=2, metric_params=None, contaminatio
    "Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.05,
                                         max_iter=-1)


}
```

In [209…
```
n_outliers = len(Fraud)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    #Fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_prediction = clf.negative_outlier_factor_
    else:
        clf.fit(X)
        scores_prediction = clf.decision_function(X)
        y_pred = clf.predict(X)
    #Reshape the prediction values to 0 for Valid transactions , 1 for Fraud transa
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1
    n_errors = (y_pred != Y).sum()
    # Run Classification Metrics
    print("{}: {}".format(clf_name,n_errors))
    print("Accuracy Score :")
    print(accuracy_score(Y,y_pred))
    print("Classification Report :")
    print(classification_report(Y,y_pred))
```

```
C:\Python310\lib\site-packages\sklearn\base.py:439: UserWarning: X does not have v
alid feature names, but IsolationForest was fitted with feature names
  warnings.warn(
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Isolation Forest: 721
Accuracy Score :
0.2782782782782783
Classification Report :
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00       434
           0       0.00      0.00      0.00         0
           1       0.56      0.49      0.52       565

    accuracy                           0.28       999
   macro avg       0.19      0.16      0.17       999
weighted avg       0.32      0.28      0.30       999

Local Outlier Factor: 717
Accuracy Score :
0.2822822822822823
Classification Report :
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00       434
           0       0.00      0.00      0.00         0
           1       0.57      0.50      0.53       565

    accuracy                           0.28       999
   macro avg       0.19      0.17      0.18       999
weighted avg       0.32      0.28      0.30       999

Support Vector Machine: 793
Accuracy Score :
0.2062062062062062
Classification Report :
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00       434
           0       0.00      0.00      0.00         0
           1       0.56      0.36      0.44       565

    accuracy                           0.21       999
   macro avg       0.19      0.12      0.15       999
weighted avg       0.32      0.21      0.25       999
```

```
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Python310\lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedM
etricWarning: Recall and F-score are ill-defined and being set to 0.0 in labels wi
th no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1. Demonstrate using code and explain how did would you identify potential fraudulent activities in financial transactions.
2. Why did you choose the given approach over other methods? Which other methods did you evaluate?
3. What features did you consider to find potential fraudulent activities? How did you perform feature engineering to improve the model?
4. Demonstrate using code and explain how would you predict the spend for all Transaction Types for the month of June.
5. How would you test the effectiveness of the model to unseen data?

In [ ]: