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. Based on your approach answer the following questions:

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? Your submission should be a PDF export of your Jupyter Notebook with your code and answers to the above questions.

```
import pandas as pd
data=pd.read_csv("/content/financial_anomaly_data.csv")
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# droppping all null values
data=data.dropna()
```

```
data
```

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Locat
0	01-01-2023 08:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	To
1	01-01-2023 08:01	TXN1639	ACC10	15607.89	MerchantH	Purchase	Lor
2	01-01-2023 08:02	TXN872	ACC8	65092.34	MerchantE	Withdrawal	Lor
3	01-01-2023 08:03	TXN1438	ACC6	87.87	MerchantE	Purchase	Lor
4	01-01-2023 08:04	TXN1338	ACC6	716.56	MerchantI	Purchase	Ang
...	...	...	...	...	...	...	

1)Identifying fradulent activities

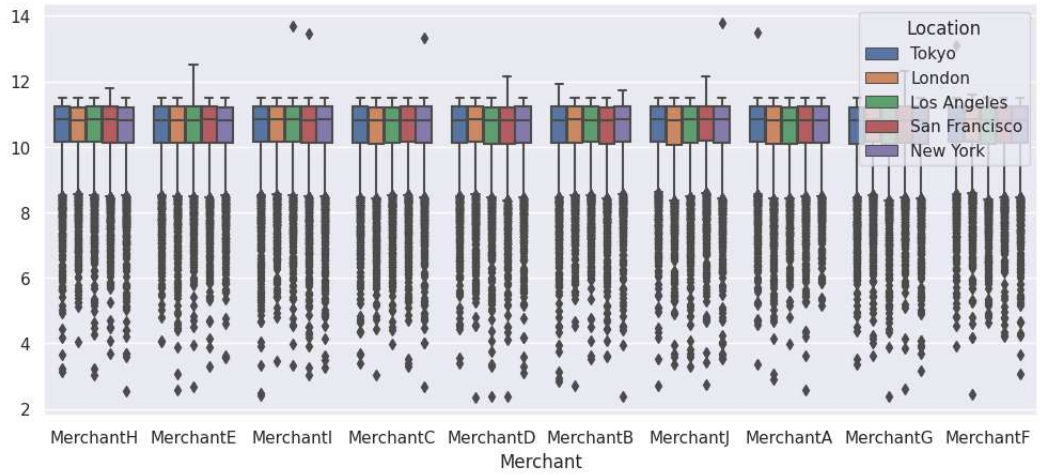
There are two methods proposed in the neotebook the first one includes the visualization of the data. On visulaizing there were some things that was noticed the amount values were right skewed to correct them log was apllied on the values and then all the procedures were performed.

- The method that we go with here in the notebook is the isolation forest method.

Visulization on the basis of country for amount columns

```
import seaborn as sns
sns.set(rc={"figure.figsize":(12,5)}) #width=3, #height=4
sns.boxplot( x=data["Merchant"], y=np.log(np.array(data.Amount)),hue=data["Location"])
```

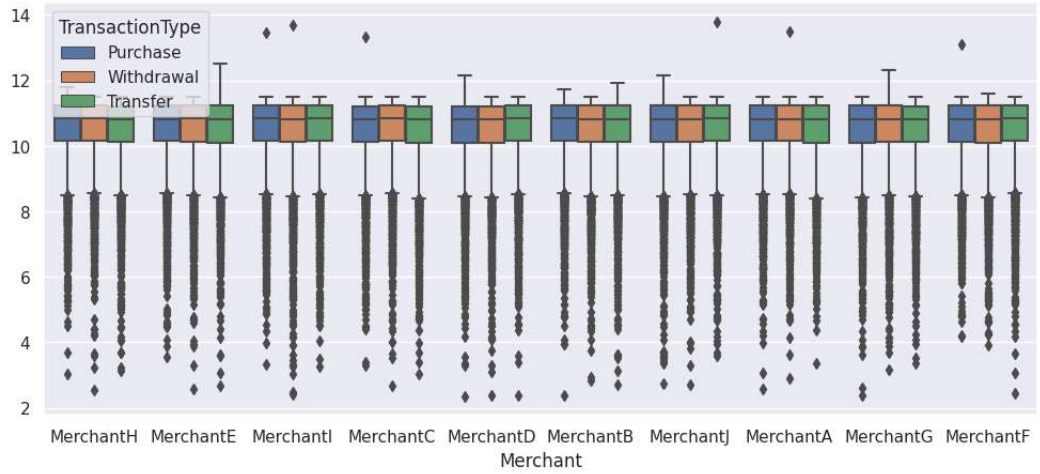
```
<Axes: xlabel='Merchant'>
```



Visulaisation on basis of transaction type on amount

```
sns.set(rc={"figure.figsize":(12,5)}) #width=3, #height=4
sns.boxplot( x=data["Merchant"], y=np.log(np.array(data.Amount)),hue=data["TransactionType"])
```

```
<Axes: xlabel='Merchant'>
```



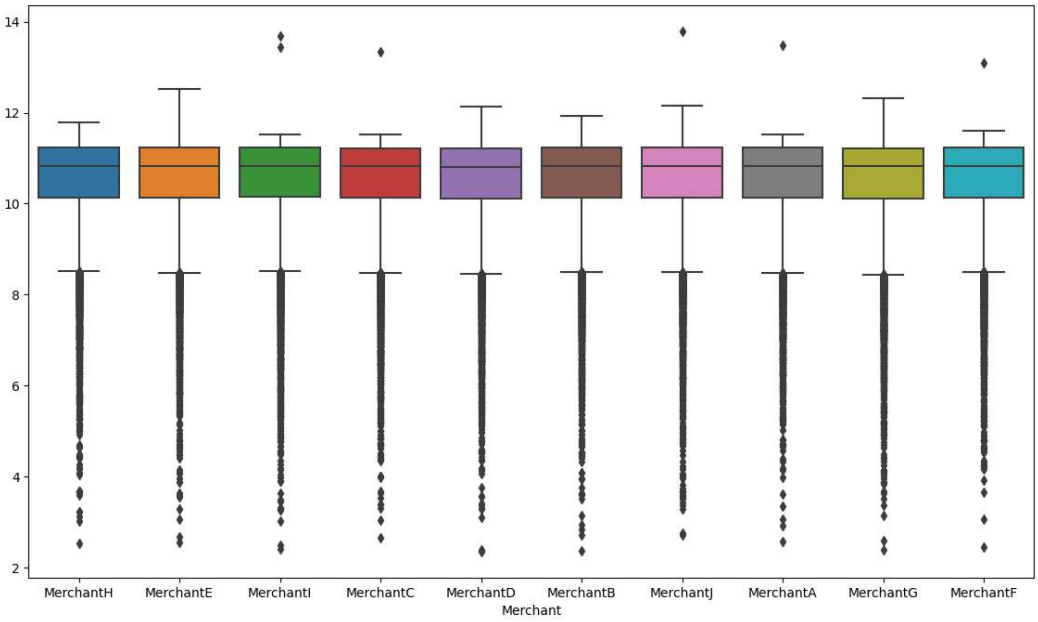

```
data[data["Merchant"]=="MerchantI"]["Amount"]
```

```
50455.82534617506
```

Visulaisation on basis of different merchants

```
sns.boxplot( x=data["Merchant"], y=np.log(np.array(data.Amount)))
```

```
<Axes: xlabel='Merchant'>
```




```
sns.kdeplot(x=np.log(np.array(data.Amount)),hue=data["TransactionType"])
```

```
<Axes: ylabel='Density'>
```

```
# data["Timestamp"]
data.Timestamp=pd.to_datetime(data.Timestamp)
```

```
days=[]
months=[]
times=[]
```

```
for i in range(0,len(data)):
  days.append(data["Timestamp"][i].day)
  months.append(data["Timestamp"][i].month)
  times.append(data["Timestamp"][i].time())
```

```
data['Day']=days
data['Month']=months
```

```
merchs=data["Merchant"].unique()
```

```
0.00
```

```
locs=data["TransactionType"].unique()
```

```
import numpy as np
import warnings

# To ignore all warnings
warnings.filterwarnings("ignore")
```

```
new_df=[]
```

```
import numpy as np
```

```
# Finding outliers in the dataset using IQR range based on merchant and transaction type

for i in merchs :
  for j in locs:
    # print(i)
    # print(j)
    ds=data[(data["Merchant"]==i) & (data["TransactionType"]==j)]
    dt=np.log(np.array(data[(data["Merchant"]==i) & (data["TransactionType"]==j)]["Amount"]))
    Q1 = np.percentile(dt, 25, interpolation = 'midpoint')
    Q2 = np.percentile(dt, 50, interpolation = 'midpoint')
    Q3 = np.percentile(dt, 75, interpolation = 'midpoint')
    IQR = Q3 - Q1
    low_lim = Q1 - 1.5 * IQR
    up_lim = Q3 + 1.5 * IQR
    new=ds[(np.log(np.array(ds["Amount"]))<low_lim) | ((np.log(np.array(ds["Amount"])))>up_lim)]
    if(new.empty==False):
      new_df.append(new)
```

```
len(new_df)
```

```
30
```

```
# Finding outliers in the dataset using IQR range based on merchant and location

for i in merchs :
  for j in locs:
    # print(i)
    # print(j)
    ds=data[(data["Merchant"]==i) & (data["Location"]==j)]
    dt=np.log(np.array(data[(data["Merchant"]==i) & (data["Location"]==j)]["Amount"]))
    Q1 = np.percentile(dt, 25, interpolation = 'midpoint')
    Q2 = np.percentile(dt, 50, interpolation = 'midpoint')
    Q3 = np.percentile(dt, 75, interpolation = 'midpoint')
    IQR = Q3 - Q1
    low_lim = Q1 - 1.5 * IQR
    up_lim = Q3 + 1.5 * IQR
    new=ds[(np.log((ds["Amount"]))<low_lim) | ((np.log((ds["Amount"])))>up_lim)]
    if(new.empty==False):
      new_df.append(new)


len(new_df)
```

```
80
```

```
data["Fraud"]=0


for i in range(len(new_df)):
  for j in range(len(new_df[i])):
    index_to_update = new_df[i].index[j]
    data.at[index_to_update, "Fraud"] = 1


# our prediction of fraud points
data["Fraud"].value_counts()
```

```
0    206260
1     10700
Name: Fraud, dtype: int64
```

```
data["Amount"]
```

```
0         95071.92
1         15607.89
2         65092.34
3            87.87
4           716.56
             ...
216955    62536.88
216956    68629.69
216957     8203.57
216958    77800.36
216959    65004.99
Name: Amount, Length: 216960, dtype: float64
```

```
# isolation forest appraoch
import numpy as np
import pandas as pd
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
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"]
iso=IsolationForest(n_estimators=100, max_samples=len(data),random_state=42, verbose=0)
```

```
data
```

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location	Fraud	Day	Month
0	2023-01-01 08:00:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	Tokyo	0	1	1
1	2023-01-01 08:01:00	TXN1639	ACC10	15607.89	MerchantH	Purchase	London	0	1	1
2	2023-01-01 08:02:00	TXN872	ACC8	65092.34	MerchantE	Withdrawal	London	0	1	1
3	2023-01-01 08:03:00	TXN1438	ACC6	87.87	MerchantE	Purchase	London	1	1	1
4	2023-01-01 08:04:00	TXN1338	ACC6	716.56	MerchantI	Purchase	Los Angeles	1	1	1
...	...	...	...	...	...	...	...	...	...	...
216955	2023-05-31 23:55:00	TXN1286	ACC6	62536.88	MerchantA	Withdrawal	San Francisco	0	31	5
216956	2023-05-31 23:56:00	TXN1015	ACC5	68629.69	MerchantG	Transfer	London	0	31	5
216957	2023-05-31 23:57:00	TXN1979	ACC15	8203.57	MerchantF	Purchase	London	0	31	5
216958	2023-05-31 23:58:00	TXN1845	ACC14	77800.36	MerchantF	Purchase	New York	0	31	5
216959	2023-05-31 23:59:00	TXN1807	ACC3	65004.99	MerchantG	Withdrawal	Los Angeles	0	31	5

216960 rows × 10 columns

```
data['Merchant'].unique()
```

```
array(['MerchantH', 'MerchantE', 'MerchantI', 'MerchantC', 'MerchantD',
       'MerchantB', 'MerchantJ', 'MerchantA', 'MerchantG', 'MerchantF',
       nan], dtype=object)
```

```
vals=[]
```

```
data['Timestamp'].diff()[1].seconds/60
for i in data['Merchant'].unique():
  df=data[data["Merchant"]==i]
  df = df.sort_values(by='Timestamp')
  vals.append(df['Timestamp'].diff())


data['diff']=0


for i in range(len(vals)):
  for j in range(len(vals[i])):
    print(i)
    print(j)
    data.loc[vals[i].index[j],'diff']=vals[i][vals[i].index[j]]


data['New_var'] = 0  # Assuming you want to initialize with 0
# data.loc[0, 'New_var'] = 1


all=[]
vals=[]


data['Amount']=np.log(np.array(data['Amount']))


# converting categorical values to numerical
for i in data["TransactionType"].unique():
  for j in data["Merchant"].unique():
    for k in data["Location"].unique():
      s=i+'_'+j+'_'+k
      all.append(s)
      vals.append(len(data[(data["Merchant"]==j) & (data["TransactionType"]==i) & (data["Location"]==k)])/len(data))


for i in range(len(data)):
  m=data.loc[i]["TransactionType"]+'_'+data.loc[i]["Merchant"]+'_'+data.loc[i]["Location"]
  for j in range(0,len(all)):
    if(all[j]==m):
      data.loc[i,'New_var']=vals[j]


import numpy as np


data["Amount"]=np.log(data["Amount"])


# reshaping data to 2d array
n=np.array(data[["Amount","Day","Month","New_var","diff"]])


n.shape
```

```
(216960, 4)
```

```
y_pred = iso.fit_predict(n)


# prediction using isolation forest
np.unique(y_pred,return_counts=True)
```

```
(array([-1,  1]), array([ 50719, 166241]))
```

2)Other methods:

- Analyzed oultliers on the basis of only merchants does not take into account the location and type of transaction many semantic and necessary information lost.
- Trained isolation forest on identfying outliers only on the basis of amount and no other features. Does not even consider the merchant type and looses information that must have been necessary.

Reason for choosing our algorithm 2

- Takes into account merchant type, the location and transaction type. Converts all of the categorical data to numerical data and does not loose the seamnatics as well.Takes into account the time and day of the transaction also in consideration.

3)Features used

- Categorical data like merchant type, location and transaction type all are combined to one numerical column .
- Features used include date and time and als time differnece based on the merchant and the location.
- Other features which could be included later correspond to rolling mean, montly average, standard deviation.

Feature engineering

- The numerical column is created such that the length of all these features occuring together denote the final value of the column.
- The datetime column is converted to date and time and is included in the list of features for training.
- The final model does it prediction based on 4 values which include our created feature and date and day columns as well.

5)Effectivness on unseen data

- To test the model on unseen data we can mimic the older data as training dataset and predict for the next month using our model.The values predicted from the test dataset and the original values will confirm how well the model has performed on