```
!pip install PyDrive
```

```
!pip install shap
```

```
!pip install keras
```

```
!pip install sklearn
```

```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import seaborn as sns
import numpy as np
from datetime import datetime as dt
import matplotlib.pyplot as plt
from sklearn.preprocessing import OneHotEncoder
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import xgboost as xgb
from xgboost import XGBClassifier
```

Saved successfully!

```
from keras.layers import Dense
import tensorflow.keras as keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
import geopy.distance
```

```
# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
#2. Get the file
downloaded = drive.CreateFile({'id':"17TlCXfs4WKyo8nshLrTGoVTLOvP6CIyr"})   # replace the id
downloaded.GetContentFile('fraudTrain.csv')


#3. Read file as panda dataframe
import pandas as pd
data = pd.read_csv('fraudTrain.csv')
```

Exploration of Data

```
data.shape
```

    (1296675, 23)

```
data.isna().sum()
```

```
fraud = data.loc[data['is_fraud']==1]
non_fraud = data.loc[data['is_fraud']==0]
```

```
fraud.describe().apply(lambda s: s.apply('{0:.2f}'.format))
```

1 to 8 of 8 entries Filter ?

index	Unnamed: 0	cc_num	amt	zip	lat	long	city_pop	uni
count	7506.00	7506.00	7506.00	7506.00	7506.00	7506.00	7506.00	7506.0
mean	624949.72	400357683719606400.00	531.32	48038.71	38.66	-89.92	97276.76	13483
std	401056.01	1276871477748873216.00	390.56	27265.56	5.17	14.28	326581.47	13830
min	2449.00	60416207185.00	1.06	1330.00	20.03	-165.67	23.00	13254
[illegible]	[illegible]	[illegible]50.00	245.66	24927.00	35.06	-96.70	746.50	13357
[illegible]	[illegible]	[illegible]998.00	396.50	46290.00	39.43	-86.69	2623.00	13488
75%	984921.50	4651007077623147.00	900.88	71107.00	42.07	-79.94	21437.00	13600
max	1295733.00	4992346398065154048.00	1376.04	99783.00	66.69	-68.56	2906700.00	13717

Show 25 per page

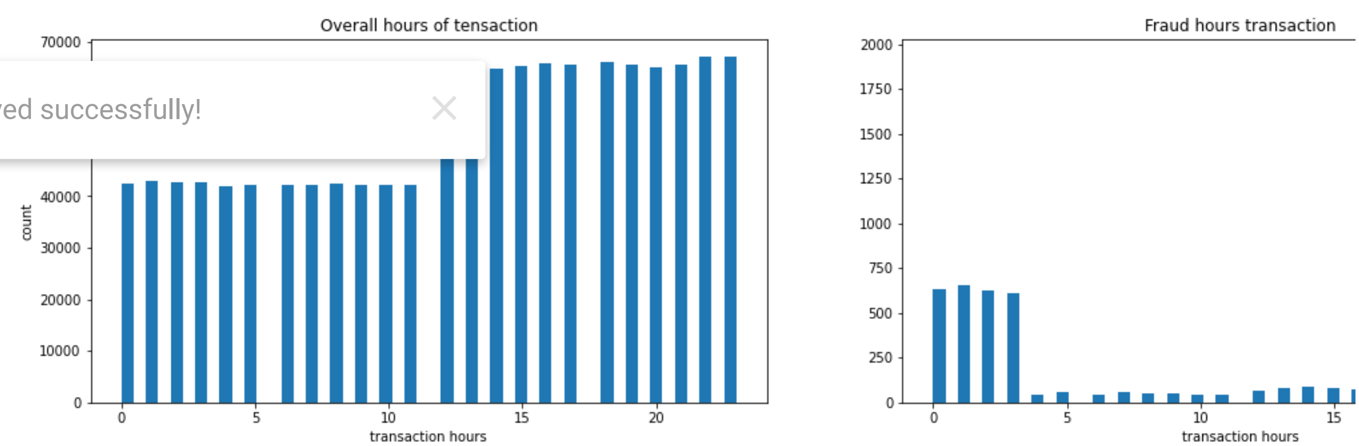
Saved successfully!

```
non_fraud.describe().apply(lambda s: s.apply('{0:.2f}'.format))
```

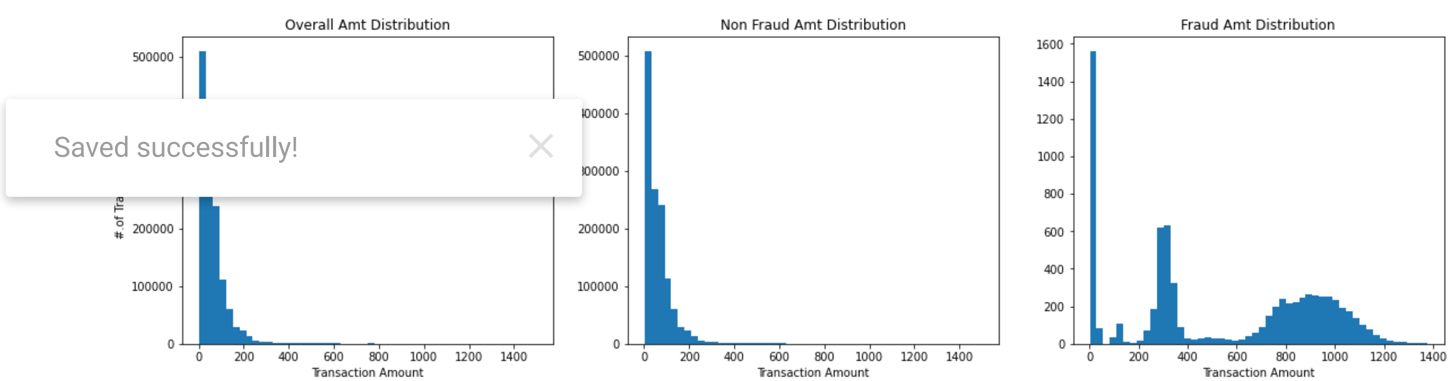
1 to 8 of 8 entries Filter ?

index	Unnamed: 0	cc_num	amt	zip	lat	long	city_
count	1289169.00	1289169.00	1289169.00	1289169.00	1289169.00	1289169.00	12891
mean	648473.17	417290057695786368.00	67.67	48805.11	38.54	-90.23	88775.
std	374152.59	1308989951701786880.00	154.01	26890.99	5.08	13.76	30180
min	0.00	60416207185.00	1.00	1257.00	20.03	-165.67	23.00

```
fraud['trans_date_trans_time'] = pd.to_datetime(fraud['trans_date_trans_time'])
fraud.dtypes['trans_date_trans_time']

fraud['transaction_hour'] = fraud['trans_date_trans_time'].dt.hour

data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time'])
data.dtypes['trans_date_trans_time']

data['transaction_hour'] = data['trans_date_trans_time'].dt.hour


fig, ax = plt.subplots(1,2,figsize=(20,5))
ax[0].hist(data['transaction_hour'], bins=50)
ax[1].hist(fraud['transaction_hour'], bins=50)

ax[0].set_title('Overall hours of tensaction')
ax[1].set_title('Fraud hours transaction')

ax[0].set_xlabel('transaction hours')
ax[0].set_ylabel('count')

ax[1].set_xlabel('transaction hours')

plt.show()
```



```
fig, ax = plt.subplots(1,2,figsize=(20,5))
ax[0].hist(data['distance'], bins=100)
ax[1].hist(fraud['distance'], bins=100)

ax[0].set_title('distance distribution for overall trasaction')
ax[1].set_title('distance distribution for fradulent trasaction')

ax[0].set_xlabel('distance between customer and merchant')
ax[0].set_ylabel('count')

ax[1].set_xlabel('distance between customer and merchant')

plt.show()



fig, ax = plt.subplots(1,3,figsize=(20,5))
ax[0].hist(data[data['amt']<=1500]['amt'], bins=50)
ax[1].hist(data[(data['is_fraud']==0) & (data['amt']<=1500)]['amt'], bins=50)
ax[2].hist(data[(data['is_fraud']==1) & (data['amt']<=1500)]['amt'], bins=50)

ax[0].set_title('Overall Amt Distribution')
ax[1].set_title('Non Fraud Amt Distribution')
ax[2].set_title('Fraud Amt Distribution')

ax[0].set_xlabel('Transaction Amount')
ax[0].set_ylabel('#.of Transactions')

ax[1].set_xlabel('Transaction Amount')
ax[2].set_xlabel('Transaction Amount')
plt.show()
```



Feature Engineering

```
# calculating % of fraud happened in each hour
data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time'])
data.dtypes['trans_date_trans_time']

data['transaction_hour'] = data['trans_date_trans_time'].dt.hour

t = data.groupby('transaction_hour').count()
t = t['trans_num']
t = pd.DataFrame(t)
t = t.rename(columns={'trans_num':'Actual_transaction'})

fraud_t = data.loc[data['is_fraud']==1]
tf = fraud_t.groupby('transaction_hour').count()
tf = tf['trans_num']

time = pd.concat([t, tf], axis=1)
time[['Actual_transaction','trans_num']]

time = time.rename(columns={'trans_num':'Fradulent_transaction'})

time['Fradulent_transaction'] = time['Fradulent_transaction'].fillna(0)

time['fraud_rate%_by_hour'] = (time['Fradulent_transaction'] / time['Actual_transaction']) *

time = time.sort_values(['fraud_rate%_by_hour'], ascending=False)
fr_time = pd.DataFrame(time['fraud_rate%_by_hour'])

act_time = data['transaction_hour']
act_time = pd.DataFrame(act_time)
```

Saved successfully!

```
e, how='left', on='transaction_hour')
```

1 to 25 of 20000 en

index	transaction_hour	fraud_rate%_by_hour
0	0	1.4
1	0	1.4
2	0	1.4
3	0	1.4
4	0	1.4
5	0	1.4
6	0	1.4
7	0	1.4
8	0	1.4
9	0	1.4
10	0	1.4
11	0	1.4
12	0	1.4
13	0	1.4
14	0	1.4
15	0	1.4
16	0	1.4
18	0	1.4

```
data['transaction_hour'] = new_time['fraud_rate%_by_hour']
```

```
# calculating % of fraud happened profession wise - assigning high risk and low risk to the p
t_categ_anal = data.groupby('category')[['trans_num']].count().reset_index()
t_categ_anal.rename({'trans_num':'total_count_of_trasactions'}, axis=1)

f_categ_anal = data[data['is_fraud']==1].groupby('category')[['trans_num']].count().reset_ind
f_categ_anal.rename({'trans_num':'count_of_fraud_trasactions'}, axis=1)

categ_anal = pd.merge(t_categ_anal, f_categ_anal, how='left', on='category')
categ_anal['fraud_perc'] = (categ_anal['trans_num_y'] / categ_anal['trans_num_x']) * 100
['fraud_perc'], ascending=False)

t_job_anal = data.groupby('job')[['trans_num']].count().reset_index()
t_job_anal = t_job_anal.sort_values(['trans_num'], ascending=False)

f_job_anal = data[data['is_fraud']==1].groupby('job')[['trans_num']].count().reset_index()
f_job_anal = f_job_anal.sort_values(['trans_num'], ascending=False)

job_anal = pd.merge(t_job_anal, f_job_anal, how='inner', on='job')
job_anal['fraud_perc'] = (job_anal['trans_num_y'] / job_anal['trans_num_x']) * 100
job_anal = job_anal.sort_values(['fraud_perc'], ascending=False)
high_fraud = job_anal[job_anal['fraud_perc'] > 90]
#high_fraud = pd.DataFrame(high_fraud['job'])
high_fraud_job = high_fraud['job'].tolist()

new_job = []
```

Saved successfully!

```
for job in data['job']:
  for fraud_job in high_fraud_job:
    if fraud_job == job:
      new_job.append('high risk')
      break
  else:
    new_job.append('low risk')

data['job'] = new_job


enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(data[['job']]).toarray())
enc_df
data = data.join(enc_df)


data.rename({0:'low risk job', 1:'high risk job'}, axis=1, inplace=True)


#calculating % fraud zip code wise
z = data.groupby('zip').count()
z = z['city']
z = pd.DataFrame(z)
z = z.rename(columns={'city':'Actual_transaction'})

fraud = data.loc[data['is_fraud']==1]
zf = fraud.groupby('zip').count()
zf = zf['city']

zip = pd.concat([z, zf], axis=1)
zip[['Actual_transaction','city']]

zip = zip.rename(columns={'city':'Fradulent_transaction'})

zip['Fradulent_transaction'] = zip['Fradulent_transaction'].fillna(0)
```

Saved successfully!

```
_transaction'] / zip['Actual_transaction']) * 100

zip = zip.sort_values(['fraud_rate%'], ascending=False)
fr_zip = pd.DataFrame(zip['fraud_rate%'])

# Replacing Zip codes with fraud rates
act_zip = data['zip']
act_zip = pd.DataFrame(act_zip)

new_zip = pd.merge(act_zip, fr_zip, how='left', on='zip')

data['zip'] = new_zip['fraud_rate%']
data['zip']


# normalising amount column
```

```
data['amt'] = np.log(data['amt'])
data['amt'].skew()
```

```
enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(data[['gender']]).toarray())
enc_df
data = data.join(enc_df)
```

```
data.rename({0:'female', 1:'male'}, axis=1, inplace=True)
```

```
data['category'].replace({'misc_net':1, 'grocery_pos':2, 'entertainment':3, 'gas_transport':4
```

```
enc = OneHotEncoder()
enc_df = pd.DataFrame(enc.fit_transform(data[['category']]).toarray())
enc_df
data = data.join(enc_df)
```

```
data.rename({0:'misc_net', 1:'grocery_pos', 2:'entertainment', 3:'gas_transport', 4:'misc_pos
```

```
data['day_of_week'] = data['trans_date_trans_time'].dt.day_name()
data['day_of_week']
```

```
label_encoder = preprocessing.LabelEncoder()
data['day_of_week'] = label_encoder.fit_transform(data['day_of_week'])
data['day_of_week'].unique()
```

```
#Target encoding of transaction hour
temp = data['transaction_hour'].value_counts().to_dict()
data['Fraud_by_hour'] = data['transaction_hour'].map(temp)
```



```
#target encoding of address = street + city + state
data['address'] = data['street'] + data['city'] + data['state']
temp = data['address'].value_counts().to_dict()
data['address_fe'] = data['address'].map(temp)
data['address_fe']
```

```
data.columns
```

```
Y = data['is_fraud']
```

```
#data.drop(columns=['Unnamed: 0','is_fraud','trans_date_trans_time','dob','trans_num','unix_t
```

```
X = data.copy()
```

Train test split

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.30)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(907672, 46) (389003, 46) (907672,) (389003,)
```

```
X_train.columns
```

Aggregate encoding done after train test split - as the encoding is done based on the statistical properties like avg, std

```
X_train['cust_det'] = X_train['first'] + X_train['last'] + X_train['cc_num'].map(str)
```

```
X_test['cust_det'] = X_test['first'] + X_test['last'] + X_test['cc_num'].map(str)
```

```
#AE
temp = X_train.groupby('cust_det')['distance'].agg(['std']).rename({'std':'cust_std_dist'}, a
X_train = pd.merge(X_train,temp,on='cust_det',how='left')
```

```
#AE
temp = X_train.groupby('cust_det')['amt'].agg(['mean']).rename({'mean':'tran_mean_cust'}, axi
X_train = pd.merge(X_train,temp,on='cust_det',how='left')
X_train['tran_mean_cust']
```

Saved successfully!

```
mt'].agg(['mean']).rename({'mean':'tran_mean_cust'}, axis
st_det',how='left')
X_test['tran_mean_cust']
```

```
#AE
temp = X_train.groupby('cust_det')['amt'].agg(['std']).rename({'std':'tran_std_cust'}, axis=1
X_train = pd.merge(X_train,temp,on='cust_det',how='left')
X_train['tran_std_cust']
```

```
#AE
temp = X_test.groupby('cust_det')['amt'].agg(['std']).rename({'std':'tran_std_cust'}, axis=1)
X_test = pd.merge(X_test,temp,on='cust_det',how='left')
X_test['tran_std_cust']
```

```
X_train.drop(columns=['Unnamed: 0','cust_det','address','is_fraud','trans_date_trans_time','d
```

```
X_test.drop(columns=['Unnamed: 0','cust_det','address','is_fraud','trans_date_trans_time','do
```

```
X_train.head(5)
```

	amt	zip	city_pop	transaction_hour	high risk job	male	misc_net	grocery_pos	en
0	4.429506	0.000000	4056	0.132542	1.0	0.0	1.0	0.0	
1	2.615204	1.353965	91	0.142278	1.0	0.0	0.0	0.0	
2	2.695978	0.591716	2036	0.123649	1.0	0.0	0.0	0.0	
3	2.900872	0.000000	207	0.120812	1.0	0.0	0.0	1.0	
4	2.284421	0.000000	3096	0.122633	1.0	1.0	0.0	0.0	

```
X_train['city_pop'] = (X_train['city_pop'] - X_train['city_pop'].min())/(X_train['city_pop'].
X_train['Fraud_by_hour'] = (X_train['Fraud_by_hour'] - X_train['Fraud_by_hour'].min())/(X_tra
X_train['address_fe'] = (X_train['address_fe'] - X_train['address_fe'].min())/(X_train['addre
```

```
X_test['city_pop'] = (X_test['city_pop'] - X_test['city_pop'].min())/(X_test['city_pop'].max(
X_test['Fraud_by_hour'] = (X_test['Fraud_by_hour'] - X_test['Fraud_by_hour'].min())/(X_test['
X_test['address_fe'] = (X_train['address_fe'] - X_train['address_fe'].min())/(X_train['addres
```

```
xgb_instance = xgb.XGBClassifier()
model_for_feature_selection = xgb_instance.fit(X_train, y_train)
```

Saved successfully!

```
so use SHAP values
feature_importance = {'Feature':X_train.columns,'Importance':model_for_feature_selection.feat
feature_importance = pd.DataFrame(feature_importance)
feature_importance.sort_values("Importance", inplace=True,ascending=False)
feature_importance
```

```
shap_values = shap.TreeExplainer(xgb_instance, feature_perturbation='interventional').shap_va
```

```
explainer = shap.TreeExplainer(xgb_instance)
shap_values = explainer.shap_values(X_train)
```

```
shap_values = shap.TreeExplainer(xgb_instance).shap_values(X_train)
shap.summary_plot(shap_values, X_train, plot_type='bar')
```

MODEL

```
lr_model = LogisticRegression(solver='saga', max_iter=600)
reult = lr_model.fit(X_train, y_train)
```

    The max_iter was reached which means the coef_ did not converge

```
y_pred = lr_model.predict(X_test)
y_pred
```

```
matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(matrix, annot=True, fmt="d")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
print(classification_report(y_test, y_pred))
```

Saved successfully!

✓ 4s completed at 8:53 PM