Codsoft Task 2 - Credit Card Fraud Detection

For this task, we'll be using Logistic Regression, Decision Trees or Random Forests to classify transactions as fraudulent or legitimate

Importing required libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
train = pd.read_csv('./Dataset/fraudTrain.csv',index_col=0)
test = pd.read_csv('./Dataset/fraudTest.csv',index_col=0)
```

Lets Analyze Training data

In [3]:
```
train.head()
```

Out[3]:

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

5 rows × 22 columns

In [4]:

```
train.info()
```

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

In [5]:
```
train.shape
```

Out[5]:
```
(1296675, 22)
```

In [6]:
```
train.describe()
```

Out[6]:

	cc_num	amt	zip	lat	long	city_p
count	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+06	1.296675e+
mean	4.171920e+17	7.035104e+01	4.880067e+04	3.853762e+01	-9.022634e+01	8.882444e+
std	1.308806e+18	1.603160e+02	2.689322e+04	5.075808e+00	1.375908e+01	3.019564e+
min	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+
25%	1.800429e+14	9.650000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.430000e+
50%	3.521417e+15	4.752000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.456000e+
75%	4.642255e+15	8.314000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+
max	4.992346e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.906700e+

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

Out[7]:
```
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]:
```
train.duplicated().sum()
```

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

From the above we can clearly see that there are no duplicates and null values in the training dataset.

In [9]:
```
train['is_fraud'].value_counts()
```

Out[9]:
```
is_fraud
0    1289169
1       7506
Name: count, dtype: int64
```

We can see that the training data is imbalanced

Lets Analyze the test data

In [10]:
```
test.head()
```

Out[10]:

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

5 rows × 22 columns

In [11]:

```
test.info()
```

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

In [12]:
```
test.shape
```

Out[12]: (555719, 22)

In [13]:
```
test.describe()
```

Out[13]:

	cc_num	amt	zip	lat	long	city
count	5.557190e+05	555719.000000	555719.000000	555719.000000	555719.000000	5.557190
mean	4.178387e+17	69.392810	48842.628015	38.543253	-90.231325	8.822189
std	1.309837e+18	156.745941	26855.283328	5.061336	13.721780	3.003909
min	6.041621e+10	1.000000	1257.000000	20.027100	-165.672300	2.300000
25%	1.800429e+14	9.630000	26292.000000	34.668900	-96.798000	7.410000
50%	3.521417e+15	47.290000	48174.000000	39.371600	-87.476900	2.408000
75%	4.635331e+15	83.010000	72011.000000	41.894800	-80.175200	1.968500
max	4.992346e+18	22768.110000	99921.000000	65.689900	-67.950300	2.906700

In [14]:
```
test.isnull().sum()
```

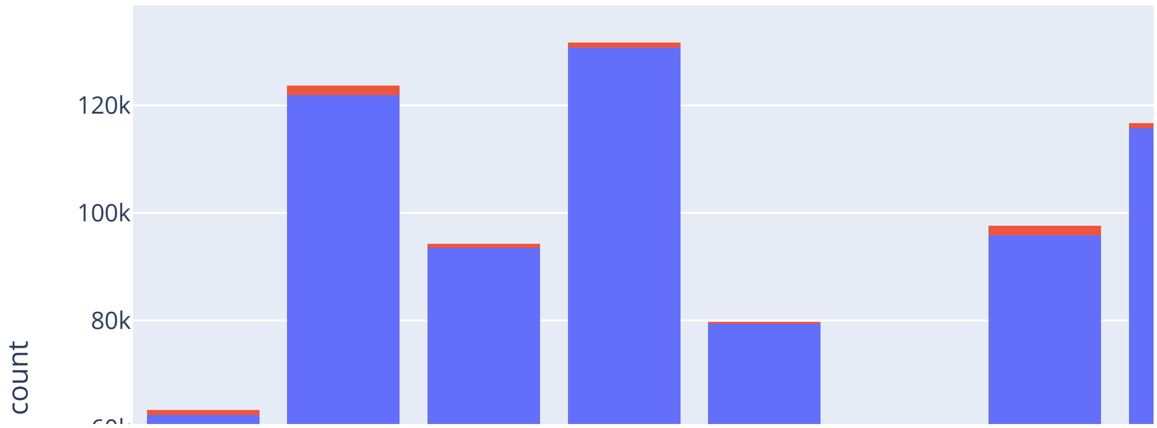
Out[14]:
```
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    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 [15]:
```
test.duplicated().sum()
```

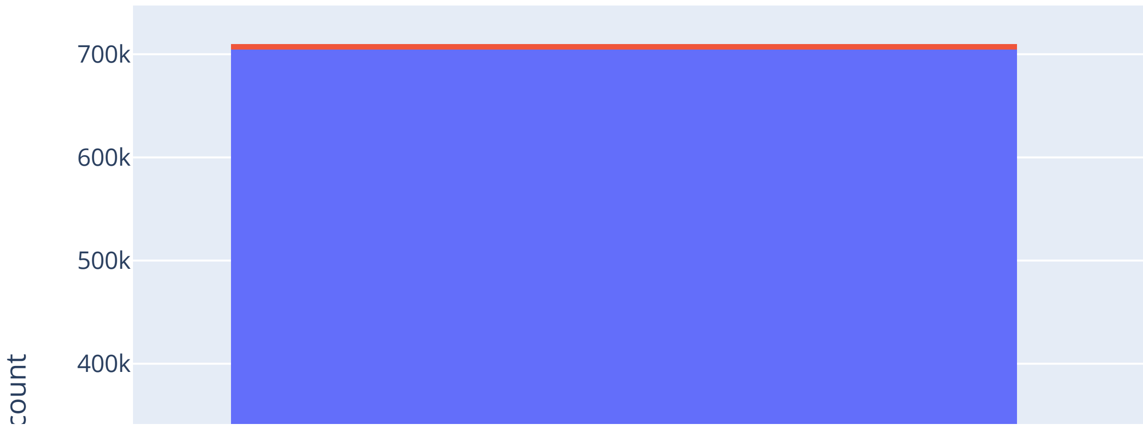
Out[15]:
```
0
```

Visualization

In [16]:
```
cat_his = px.histogram(data_frame=train,x='category',color='is_fraud');
cat_his.show();
```



In [17]:
```
gen_his = px.histogram(data_frame=train,x='gender',color='is_fraud');
gen_his.show()
```



In [18]:
```
train.columns
```

Out[18]:
```
Index(['trans_date_trans_time', 'cc_num', 'merchant', 'category', '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')
```

In [19]:
```
test.columns
```

Out[19]:
```
Index(['trans_date_trans_time', 'cc_num', 'merchant', 'category', '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')
```

Data Processing & Cleaning

In [20]:
```
#Date of birth --> Age of customer

train['dob'] = pd.to_datetime(train['dob'],format='mixed')
test['dob'] = pd.to_datetime(test['dob'],format='mixed')
```

```
train['trans_date_trans_time'] = pd.to_datetime(train['trans_date_trans_time'],form
test['trans_date_trans_time'] = pd.to_datetime(test['trans_date_trans_time'],format

train['age'] = (train['trans_date_trans_time'].dt.year - train['dob'].dt.year).asty
test['age'] = (test['trans_date_trans_time'].dt.year - test['dob'].dt.year).astype(

train.drop(columns='dob',inplace=True)
test.drop(columns='dob',inplace=True)
```

In [21]:
```
#cleaning merchant column
train['merchant'] = train['merchant'].apply(lambda x : x.replace('fraud_',''))
test['merchant'] = test['merchant'].apply(lambda x : x.replace('fraud_',''))
```

In [22]:
```
#Converting gender to binary classification
train = pd.get_dummies(train,columns=['gender'],drop_first=True)
test = pd.get_dummies(test,columns=['gender'],drop_first=True)
```

In [23]:
```
train_numerical_columns = train.select_dtypes(include=['int64', 'float64']).columns
train_numerical_columns
```

Out[23]:
```
Index(['cc_num', 'amt', 'zip', 'lat', 'long', 'city_pop', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud'],
      dtype='object')
```

In [24]:
```
test_numerical_columns = test.select_dtypes(include=['int64', 'float64']).columns
test_numerical_columns
```

Out[24]:
```
Index(['cc_num', 'amt', 'zip', 'lat', 'long', 'city_pop', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud'],
      dtype='object')
```

In [25]:
```
train.drop(
    columns = ['cc_num','trans_date_trans_time','first', 'last','street','state',
               'zip', 'lat','long', 'city_pop','unix_time', 'merch_lat','merch_long
test.drop(
    columns = ['cc_num','trans_date_trans_time','first', 'last','street','state',
               'zip', 'lat','long', 'city_pop','unix_time', 'merch_lat','merch_long
```

In [26]:
```
train = train[['merchant','category','city','job','gender_M','age','amt','is_fraud'
train.head()
```

Out[26]:

	merchant	category	city	job	gender_M	age	amt	is_fraud
0	Rippin, Kub and Mann	misc_net	Moravian Falls	Psychologist, counselling	False	31	4.97	0
1	Heller, Gutmann and Zieme	grocery_pos	Orient	Special educational needs teacher	False	41	107.23	0
2	Lind-Buckridge	entertainment	Malad City	Nature conservation officer	True	57	220.11	0
3	Kutch, Hermiston and Farrell	gas_transport	Boulder	Patent attorney	True	52	45.00	0
4	Keeling-Crist	misc_pos	Doe Hill	Dance movement psychotherapist	True	33	41.96	0

In [27]:

```
test = test[['merchant','category','city','job','gender_M','age','amt','is_fraud']]
test.head()
```

Out[27]:

	merchant	category	city	job	gender_M	age	amt	is_fraud
0	Kirlin and Sons	personal_care	Columbia	Mechanical engineer	True	52	2.86	0
1	Sporer-Keebler	personal_care	Altonah	Sales professional, IT	False	30	29.84	0
2	Swaniawski, Nitzsche and Welch	health_fitness	Bellmore	Librarian, public	False	50	41.28	0
3	Haley Group	misc_pos	Titusville	Set designer	True	33	60.05	0
4	Johnston-Casper	travel	Falmouth	Furniture designer	True	65	3.19	0

Now we wil encode the String columns

In [28]:

```
# We will use Weight of Evidence encoder for this task
from category_encoders import WOEEncoder
```

In [29]:

```
encoder = WOEEncoder(cols=['city', 'job', 'merchant', 'category'])
train_transformed = encoder.fit_transform(train[['city', 'job', 'merchant', 'catego
test_transformed = encoder.transform(test[['city', 'job', 'merchant', 'category']])
```

```
# Replace the original columns with the transformed values
train[['city', 'job', 'merchant', 'category']] = train_transformed
test[['city', 'job', 'merchant', 'category']] = test_transformed
```

In [30]:
```
train.head(3)
```

Out[30]:

	merchant	category	city	job	gender_M	age	amt	is_fraud
0	0.959326	0.924914	-2.469513	-1.080186	False	31	4.97	0
1	0.663187	0.898799	-3.027790	-0.904144	False	41	107.23	0
2	-0.790166	-0.847622	-1.076791	1.120434	True	57	220.11	0

In [31]:
```
test.head(3)
```

Out[31]:

	merchant	category	city	job	gender_M	age	amt	is_fraud
0	-1.259443	-0.869588	0.364725	0.430148	True	52	2.86	0
1	-0.569596	-0.869588	0.215637	0.200487	False	30	29.84	0
2	-1.202771	-1.315531	-0.626823	-0.637068	False	50	41.28	0

We can see that the data is not standardized. Lets Standardize

In [32]:
```
from sklearn.preprocessing import StandardScaler
```

In [33]:
```
scaler = StandardScaler()
```

In [34]:
```
X_train = train.drop('is_fraud',axis=1)
y_train = train['is_fraud']

X_test = test.drop('is_fraud',axis=1)
y_test = test['is_fraud']
```

In [35]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Now we have Standardized the Train & Test Data

Model Building

First we will proceed with Models without sampling the data. Then we will sample the data and check the accuracies before and after Sampling

In [36]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.metrics import *
```

In [37]:
```python
models = [LogisticRegression(),DecisionTreeClassifier()]
# Removed RandomForestClassifier() model as it is taking very long time for predict
```

In [38]:
```python
def model_prediction(X_train,y_train,X_test,y_test):
    for model in models:
        model.fit(X_train,y_train)
        y_prediction = model.predict(X_test)

        print(f"{model} Model")
        print("\nAccuracy Score :",accuracy_score(y_test,y_prediction))
        print("Precision :",precision_score(y_test,y_prediction))
        print("Recall Score :",recall_score(y_test,y_prediction))
        print("F1 Score :",f1_score(y_test,y_prediction))
        print("...............................................\n")
```

In [39]:
```python
model_prediction(X_train,y_train,X_test,y_test)
```

```
LogisticRegression() Model

Accuracy Score : 0.9956704737466238
Precision : 0.03225806451612903
Recall Score : 0.004195804195804196
F1 Score : 0.007425742574257425
...............................................

DecisionTreeClassifier() Model

Accuracy Score : 0.9951612235680263
Precision : 0.3353510895883777
Recall Score : 0.25827505827505826
F1 Score : 0.29180932314985514
...............................................
```

Lets Apply Sampling and check the Accuracies

In [40]:
```python
from imblearn.over_sampling import SMOTE
```

In [41]:
```python
smote = SMOTE()
X_train,y_train = smote.fit_resample(X_train,y_train)
```

In [42]:
```python
model_prediction(X_train,y_train,X_test,y_test)
```

```
LogisticRegression() Model

Accuracy Score : 0.9121498455154493
Precision : 0.02229136388758111
Recall Score : 0.5076923076923077
F1 Score : 0.04270755715910428
...............................................

DecisionTreeClassifier() Model

Accuracy Score : 0.9938026232682344
Precision : 0.19776640297812936
Recall Score : 0.19813519813519814
F1 Score : 0.19795062878435024
...............................................
```

From the above, We can clearly see that the accuracies, precisions, recall & F1 score are very low. Now we try with dropping String columns from the train and test dataset.

In [43]:
```
fraudTrain_df = pd.read_csv('./Dataset/fraudTrain.csv',index_col=0)
fraudTest_df = pd.read_csv('./Dataset/fraudTrain.csv',index_col=0)
```

In [44]:
```
# Remove string columns
numeric_fraudTrain_df = fraudTrain_df.select_dtypes(exclude=['object'])
numeric_fraudTest_df = fraudTest_df.select_dtypes(exclude=['object'])
```

In [45]:
```
X_train = train.drop('is_fraud',axis=1)
y_train = train['is_fraud']

X_test = test.drop('is_fraud',axis=1)
y_test = test['is_fraud']
```

In [46]:
```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [47]:
```
# Initialize classifiers
log_reg = LogisticRegression()
dec_tree = DecisionTreeClassifier()
```

In [48]:
```
# Train classifiers
log_reg.fit(X_train_scaled, y_train)
dec_tree.fit(X_train, y_train)
```

Out[48]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [49]:
```
# Make predictions
y_pred_log_reg = log_reg.predict(X_test_scaled)
```

```python
y_pred_dec_tree = dec_tree.predict(X_test)
```

In [50]:
```python
# Calculate metrics
metrics = {}
metrics['Logistic Regression'] = {
    'Precision': precision_score(y_test, y_pred_log_reg),
    'Recall': recall_score(y_test, y_pred_log_reg),
    'Accuracy': accuracy_score(y_test, y_pred_log_reg),
    'F1 Score': f1_score(y_test, y_pred_log_reg)
}
metrics['Decision Tree'] = {
    'Precision': precision_score(y_test, y_pred_dec_tree),
    'Recall': recall_score(y_test, y_pred_dec_tree),
    'Accuracy': accuracy_score(y_test, y_pred_dec_tree),
    'F1 Score': f1_score(y_test, y_pred_dec_tree)
}
```

In [51]:
```python
# Print the metrics for each model
for model, scores in metrics.items():
    print(f'{model} metrics:')
    print('\n')
    for score_name, score_value in scores.items():
        print(f'{score_name}: {score_value:.4f}')
    print('.......................................')
```

```
Logistic Regression metrics:


Precision: 0.0323
Recall: 0.0042
Accuracy: 0.9957
F1 Score: 0.0074
.......................................
Decision Tree metrics:


Precision: 0.3370
Recall: 0.2555
Accuracy: 0.9952
F1 Score: 0.2906
.......................................
```