Double-click (or enter) to edit

1. Problem Statement

With the growth of e-commerce websites, people and financial companies rely on online services to carry out their transactions that have led to an exponential increase in the credit card frauds

1. Fraudulent credit card transactions lead to a loss of huge amount of money. The design of an effective fraud detection system is necessary in order to reduce the losses incurred by the customers and financial companies
2. Research has been done on many models and methods to prevent and detect credit card frauds. Some credit card fraud transaction datasets contain the problem of imbalance in datasets. A good fraud detection system should be able to identify the fraud transaction accurately and should make the detection possible in real-time transactions.

About the Dataset

This is a simulated credit card transaction dataset containing legitimate and fraud transactions from the duration 1st Jan 2019 - 31st Dec 2020. It covers credit cards of 1000 customers doing transactions with a pool of 800 merchants.

Data Dictionary

trans_date_trans_time -> Transaction time stamp
cc_num -> Credit card number
merchant -> merchant name
category -> transaction category
amt -> Transaction amount
first -> First name of card holder
last -> Last name of card holder
gender -> Sex of card holder
street -> transaction address
city -> transaction city
state -> transaction state
zip -> transaction zipcode
lat -> transaction lattitude
long -> transaction longitude
city_pop -> Population of the city
job -> job of the card holder
dob -> date of birth of card holder
trans_num -> transaction number of transaction
unix_time -> time in unix format
merch_lat -> lattitude of the merchant
merch_long -> longitude of merchant
is_fraud -> nature of transaction (fraud or not fraud)

Our Goals:

1. Understand the little distribution of the "little" data that was provided to us.
2. Create a 50/50 sub-dataframe ratio of "Fraud" and "Non-Fraud" transactions. (NearMiss Algorithm)
3. Determine the Classifiers we are going to use and decide which one has a higher accuracy.

1. Import libraries

Double-click (or enter) to edit

```
# Data preprocessing libraries
import numpy as np
import pandas as pd
from pandas.plotting import parallel_coordinates

import os
import sqlite3
import math
from collections import Counter
from pathlib import Path
from tqdm import tqdm

# Visualization
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import plotly
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.io as pio

# Model
from scipy.stats import skew
import yellowbrick
import sklearn
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

# Config
mpl.rcParams['font.family'] = 'monospace'
sns.set_theme(style="white", palette=None)
plotly.offline.init_notebook_mode()
plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300
```

```
%matplotlib inline
```

Data preprocessing

```
# Reading csv files and drop the first column
fraud = pd.read_csv("fraudTrain.csv")
```

```
fraud_test = pd.read_csv("fraudTest.csv")
```

```
# First view 10 rows
fraud.head(10)
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	a
0	0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4
1	1	2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107
2	2	2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220
3	3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45
4	4	2019-01-01 00:03:06	375534208663984	fraud_Keeling-Crist	misc_pos	41
5	5	2019-01-01 00:04:08	4767265376804500	fraud_Stroman, Hudson and Erdman	gas_transport	94
6	6	2019-01-01 00:04:42	30074693890476	fraud_Rowe-Vandervort	grocery_net	44
7	7	2019-01-01 00:05:08	6011360759745864	fraud_Corwin-Collins	gas_transport	71
8	8	2019-01-01 00:05:18	4922710831011201	fraud_Herzog Ltd	misc_pos	4
9	9	2019-01-01 00:06:01	2720830304681674	fraud_Schoen, Kuphal and Nitzsche	grocery_pos	198

10 rows × 23 columns

```
fraud.columns
```

```
Index(['Unnamed: 0', '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')
```

Exploratory Data Analysis

1. Univariate Analysis
2. Bivariate Analysis
3. Data Cleaning
4. Outlier Treatment
5. Variable Transformation

```
# checking for various columns and nulls in the dataset
fraud.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7815 entries, 0 to 7814
Data columns (total 23 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             7815 non-null   int64
```

```
 1   trans_date_trans_time  7815 non-null   object
 2   cc_num                 7815 non-null   int64
 3   merchant               7815 non-null   object
 4   category               7815 non-null   object
 5   amt                    7815 non-null   float64
 6   first                  7815 non-null   object
 7   last                   7815 non-null   object
 8   gender                 7814 non-null   object
 9   street                 7814 non-null   object
 10  city                   7814 non-null   object
 11  state                  7814 non-null   object
 12  zip                    7814 non-null   float64
 13  lat                    7814 non-null   float64
 14  long                   7814 non-null   float64
 15  city_pop               7814 non-null   float64
 16  job                    7814 non-null   object
 17  dob                    7814 non-null   object
 18  trans_num              7814 non-null   object
 19  unix_time              7814 non-null   float64
 20  merch_lat              7814 non-null   float64
 21  merch_long             7814 non-null   float64
 22  is_fraud               7814 non-null   float64
dtypes: float64(9), int64(2), object(12)
memory usage: 1.4+ MB
```

```
# checking % of data provided by Kaggle in the train & test
1296675 * 100 / (1296675 + 555719)
```

```
69.99995681264353
```

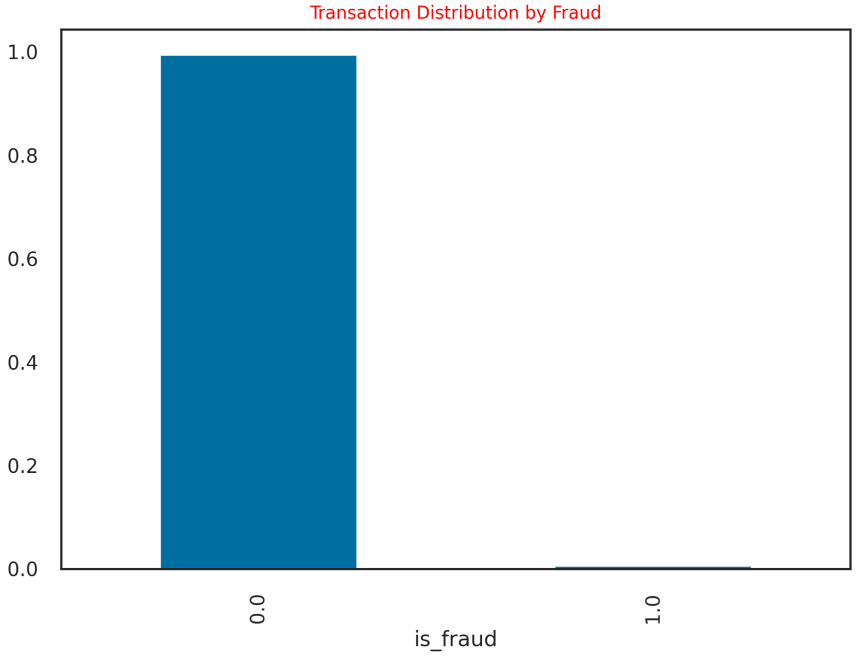
- 70% data is present in the train dataset and remaining 30% in the test dataset.
- No null values in either of the files

```
# Check for imbalance on target variable in the train dataset
fraud.is_fraud.value_counts(normalize=True)
```

```
is_fraud
0.0    0.994241
1.0    0.005759
Name: proportion, dtype: float64
```

```
# Check for imbalance on target variable in the main dataset
print ('Fraud Distribution')
print (fraud.is_fraud.value_counts(normalize=True))
plt.title('Transaction Distribution by Fraud', fontsize= 10, color = 'Red', fontweight = 100)
fraud.is_fraud.value_counts(normalize=True).plot.bar()
plt.show()
```

```
Fraud Distribution
is_fraud
0.0    0.994241
1.0    0.005759
Name: proportion, dtype: float64
```




```
# Check for imbalance on target variable in the test dataset
fraud_test.is_fraud.value_counts(normalize=True)
```

```
is_fraud
0.0    0.996984
1.0    0.003016
Name: proportion, dtype: float64
```

Both the datasets have high imbalnce of the target variable with the test dataset having slightly higher imbalance. At this point, lets keep the test data seperate. We will be building the model on the train dataset. If required, a validation dataset will be carved from it. The final evaluation will be done on the test dataset.

Univariate Analysis

The following columns seems of very less/ no significance in determining a fraud case. Primary reason being no model can be created based on person's name or his PII or some unique ID/ S.no. assigned. Hence, dropping them:-

1. cc_num
2. first
3. last
4. street

5. trans_num

```
# Dropping the unwanted columns from both datasets
fraud.drop(['cc_num', 'first', 'last', 'street', 'trans_num'], axis=1, inplace=True)
fraud.drop(fraud.iloc[:,[0]], axis=1, inplace=True)
fraud_test.drop(['cc_num', 'first', 'last', 'street', 'trans_num'], axis=1, inplace=True)
fraud_test.drop(fraud_test.iloc[:,[0]], axis=1, inplace=True)
```

```
# Inspecting the fraud dataset
fraud.head()
```

	trans_date_trans_time	merchant	category	amt	gender	city	state
0	2019-01-01 00:00:18	fraud_Rippin, Kub and Mann	misc_net	4.97	F	Moravian Falls	NC
1	2019-01-01 00:00:44	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	Orient	WA
2	2019-01-01 00:00:51	fraud_Lind-Buckridge	entertainment	220.11	M	Malad City	ID
3	2019-01-01 00:01:16	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	M	Boulder	MT
4	2019-01-01 00:03:06	fraud_Keeling-Crist	misc_pos	41.96	M	Doe Hill	VA

```
# Inspecting the fraud test dataset
fraud_test.head()
```

	trans_date_trans_time	merchant	category	amt	gender	city	stat
0	2020-06-21 12:14:25	fraud_Kirlin and Sons	personal_care	2.86	M	Columbia	S
1	2020-06-21 12:14:33	fraud_Sporer-Keebler	personal_care	29.84	F	Altonah	U
2	2020-06-21 12:14:53	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	F	Bellmore	N
3	2020-06-21 12:15:15	fraud_Haley Group	misc_pos	60.05	M	Titusville	F
4	2020-06-21 12:15:17	fraud_Johnston-Casper	travel	3.19	M	Falmouth	M

```
# Converting dob to age
from datetime import date
import pandas as pd
import numpy as np

fraud['dob'] = pd.to_datetime(fraud['dob'])
fraud['age'] = (pd.to_datetime('now') - fraud['dob'])/ np.timedelta64(1, 'Y')

# Fill or drop NaN values in 'age' before converting to integer
fraud['age'] = fraud['age'].fillna(-1) # Replace NaN with -1, or any suitable placeholder
fraud['age'] = fraud['age'].astype(int) # Now convert to integer

fraud.drop(['dob'], axis=1, inplace=True)
fraud.head()
```

	trans_date_trans_time	merchant	category	amt	gender	city	state
0	2019-01-01 00:00:18	fraud_Rippin, Kub and Mann	misc_net	4.97	F	Moravian Falls	NC
1	2019-01-01 00:00:44	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	Orient	WA
2	2019-01-01 00:00:51	fraud_Lind-Buckridge	entertainment	220.11	M	Malad City	ID
3	2019-01-01 00:01:16	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	M	Boulder	MT
4	2019-01-01 00:03:06	fraud_Keeling-Crist	misc_pos	41.96	M	Doe Hill	VA

```
# Same change on the test dataset
fraud_test['dob'] = pd.to_datetime(fraud_test['dob'])
fraud_test['age'] = (pd.to_datetime('now') - fraud_test['dob'])/ np.timedelta64(1, 'Y')

# Fill or drop NaN values in 'age' before converting to integer
fraud_test['age'] = fraud_test['age'].fillna(-1) # Replace NaN with -1
fraud_test['age'] = fraud_test['age'].astype(int)

fraud_test.drop(['dob'], axis=1, inplace=True)
fraud_test.head()
```

	trans_date_trans_time	merchant	category	amt	gender	city	stat
0	2020-06-21 12:14:25	fraud_Kirlin and Sons	personal_care	2.86	M	Columbia	S
1	2020-06-21 12:14:33	fraud_Sporer-Keebler	personal_care	29.84	F	Altonah	U
2	2020-06-21 12:14:53	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	F	Bellmore	N
3	2020-06-21 12:15:15	fraud_Haley Group	misc_pos	60.05	M	Titusville	F
4	2020-06-21 12:15:17	fraud_Johnston-Casper	travel	3.19	M	Falmouth	M

```
# Seggregating data and time from trans_date_trans_time field
fraud['trans_date'] = pd.DatetimeIndex(fraud['trans_date_trans_time']).date
fraud['trans_time'] = pd.DatetimeIndex(fraud['trans_date_trans_time']).time
fraud.drop(['trans_date_trans_time'], axis=1, inplace=True)
fraud.head()
```

	merchant	category	amt	gender	city	state	zip	lat	lo
0	fraud_Rippin, Kub and Mann	misc_net	4.97	F	Moravian Falls	NC	28654.0	36.0788	-81.17
1	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	F	Orient	WA	99160.0	48.8878	-118.21
2	fraud_Lind-Buckridge	entertainment	220.11	M	Malad City	ID	83252.0	42.1808	-112.26
3	fraud_Kutch, Hermiston and Farrell	gas_transport	45.00	M	Boulder	MT	59632.0	46.2306	-112.11
4	fraud_Keeling-Crist	misc_pos	41.96	M	Doe Hill	VA	24433.0	38.4207	-79.46

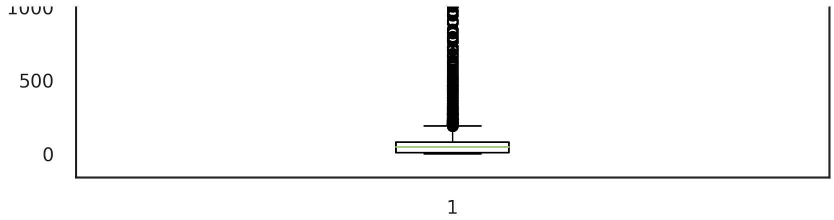
```
# Same changes on test dataset
fraud_test['trans_date'] = pd.DatetimeIndex(fraud_test['trans_date_trans_time']).date
fraud_test['trans_time'] = pd.DatetimeIndex(fraud_test['trans_date_trans_time']).time
fraud_test.drop(['trans_date_trans_time'], axis=1, inplace=True)
fraud_test.head()
```

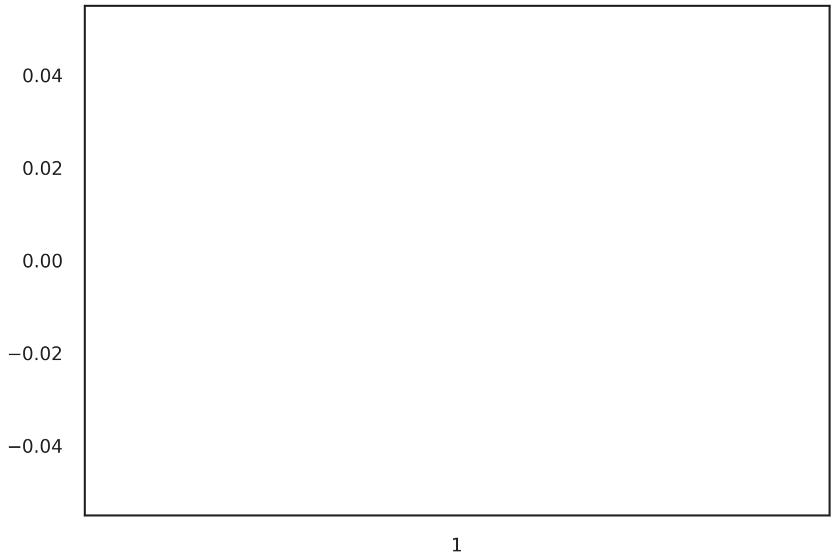
	merchant	category	amt	gender	city	state	zip	lat	l
0	fraud_Kirlin and Sons	personal_care	2.86	M	Columbia	SC	29209	33.9659	-80.9
1	fraud_Sporer-Keebler	personal_care	29.84	F	Altonah	UT	84002	40.3207	-110.4
2	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	F	Bellmore	NY	11710	40.6729	-73.5
3	fraud_Haley Group	misc_pos	60.05	M	Titusville	FL	32780	28.5697	-80.8
4	fraud_Johnston-Casper	travel	3.19	M	Falmouth	MI	49632	44.2529	-85.0

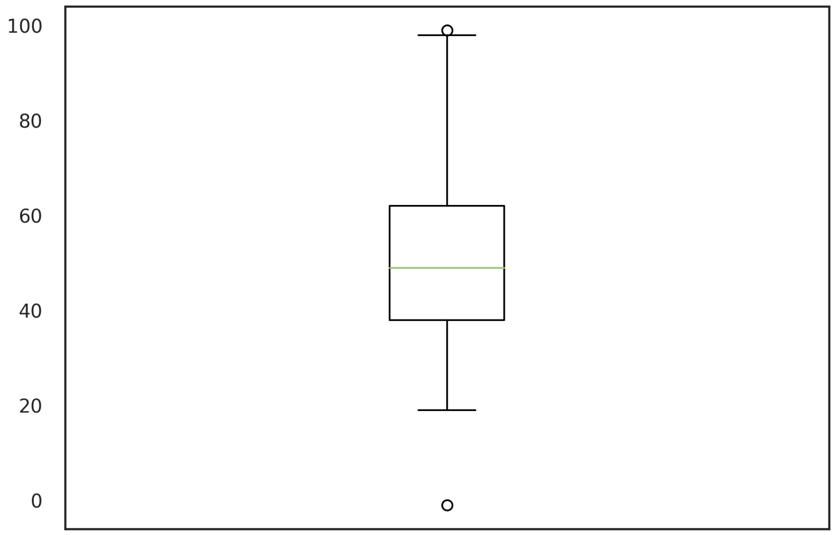
```
# Check on numeric columns for outliers
fraud.describe()
```

	amt	zip	lat	long	city_pop	unix_time
count	7815.000000	7814.000000	7814.000000	7814.000000	7.814000e+03	7.814000e+03
mean	68.661363	49479.396340	38.595821	-90.688408	8.823451e+04	1.325583e+09
std	115.530027	27184.827244	5.198199	14.512405	2.944470e+05	1.359720e+05
min	1.010000	1257.000000	20.027100	-165.672300	2.300000e+01	1.325376e+09
25%	9.785000	26041.000000	34.852700	-97.235100	7.410000e+02	1.325448e+09
50%	48.490000	49259.000000	39.376400	-87.764400	2.395000e+03	1.325583e+09
75%	82.220000	72476.000000	41.846700	-80.143075	1.905400e+04	1.325704e+09
max	3178.510000	99783.000000	65.689900	-67.950300	2.906700e+06	1.325815e+09

```
# Further checking distribution of continuous variables - amt, city_pop and age columns to see if there are any valid outliers
plt.boxplot(fraud.amt)
plt.show()
plt.boxplot(fraud.city_pop)
plt.show()
plt.boxplot(fraud.age)
plt.show()
```


1000
500
0
1


0.04
0.02
0.00
−0.02
−0.04
1


100
80
60
40
20
0

The age column has no outliers while amt and city_pop stastically shows outliers. However, both amount and city population can vary drastically and none of them seems very high or very low. Hence, we will consider it as valid data.

```
# Identifying all the Numeric and non numeric columns
num = []
obj = []
for i in range (0,13):
    if fraud.iloc[:,i].dtype != 'O':
        num.append(i)
    else:
        obj.append(i)
print(num)
print(obj)
col_names = fraud.columns
print(col_names)
```

```
[2, 6, 7, 8, 9, 11, 12]
[0, 1, 3, 4, 5, 10]
Index(['merchant', 'category', 'amt', 'gender', 'city', 'state', 'zip', 'lat',
       'long', 'city_pop', 'job', 'unix_time', 'merch_lat', 'merch_long',
       'is_fraud', 'age', 'trans_date', 'trans_time'],
      dtype='object')
```

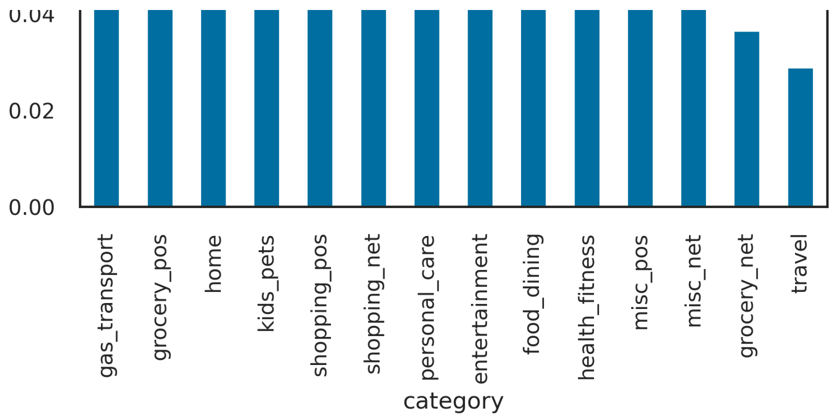
```
# Checking the distribution of object variables
for i in obj:
    print (col_names[i])
    print (fraud.iloc[:,i].value_counts(normalize=True))
    print ('*' * 50)
```

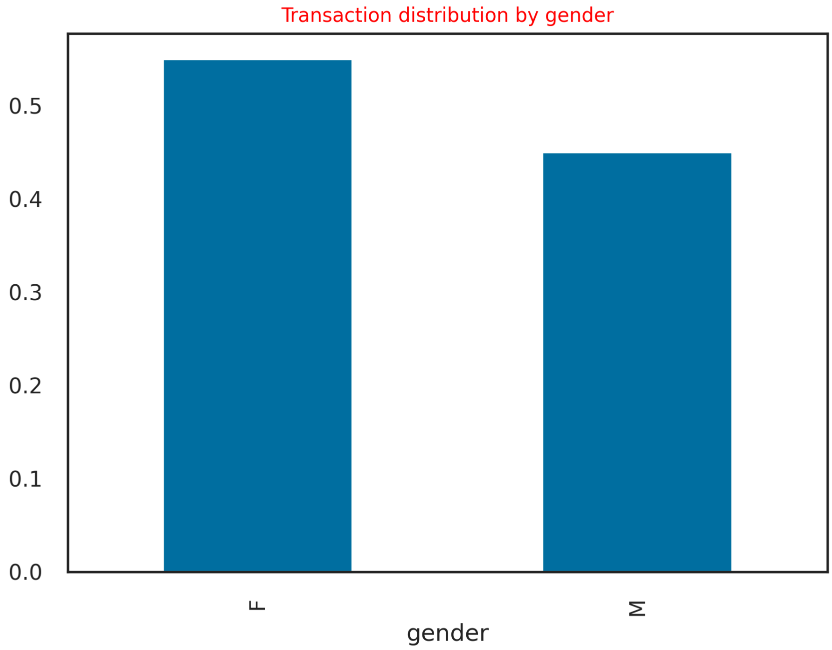
```
AL    0.031354
AR    0.025339
VA    0.025339
IA    0.022396
MN    0.022268
MD    0.022268
OK    0.022140
NC    0.022012
WI    0.021756
WV    0.021244
SC    0.020988
KY    0.019964
KS    0.019324
NE    0.019068
NJ    0.018300
IN    0.018300
OR    0.017149
LA    0.016637
WA    0.015997
GA    0.015869
WY    0.015485
TN    0.014461
MS    0.013949
NM    0.013181
ME    0.012414
CO    0.009726
MT    0.009470
AZ    0.009342
ND    0.009214
VT    0.008830
MA    0.008574
UT    0.008062
CT    0.007807
SD    0.007679
NH    0.004607
ID    0.004479
AK    0.003711
NV    0.003583
DC    0.003327
HI    0.003327
RI    0.000256
Name: proportion, dtype: float64
**************************************************
job
job
Designer, ceramics/pottery          0.007934
Exhibition designer                 0.007679
Systems developer                   0.006911
IT trainer                          0.006783
Financial adviser                   0.006399
                                      ...
Investment banker, operational      0.000128
Buyer, retail                       0.000128
Minerals surveyor                   0.000128
Production assistant, television    0.000128
Media planner                       0.000128
Name: proportion, Length: 473, dtype: float64
**************************************************
```

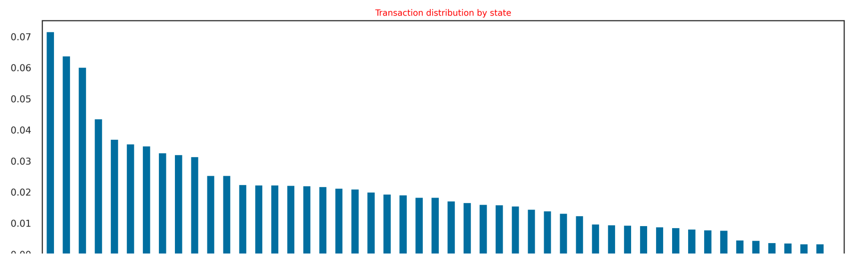
```
# Lets check the transaction distribution by Category, Gender and State variables
plt.figure(figsize = (7,5))
plt.title('Transaction distribution by Category', fontsize= 10, color = 'Red', fontweight = 100)
fraud.category.value_counts(normalize=True).plot.bar()
plt.show()

plt.figure(figsize = (7,5))
plt.title('Transaction distribution by gender', fontsize= 10, color = 'Red', fontweight = 100)
fraud.gender.value_counts(normalize=True).plot.bar()
plt.show()

plt.figure(figsize = (17,5))
plt.title('Transaction distribution by state', fontsize= 10, color = 'Red', fontweight = 100)
fraud.state.value_counts(normalize=True).plot.bar()
plt.show()
```


0.04
0.02
0.00
gas_transport
grocery_pos
home
kids_pets
shopping_pos
shopping_net
personal_care
entertainment
food_dining
health_fitness
misc_pos
misc_net
grocery_net
travel
category


Transaction distribution by gender
0.5
0.4
0.3
0.2
0.1
0.0
F
M
gender


Transaction distribution by state
0.07
0.06
0.05
0.04
0.03
0.02
0.01

Bi-Variate Analysis

Check for the behaviour of various columns against the is_fraud column

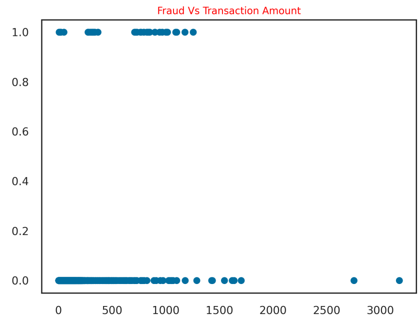
```
fraud.info()
```

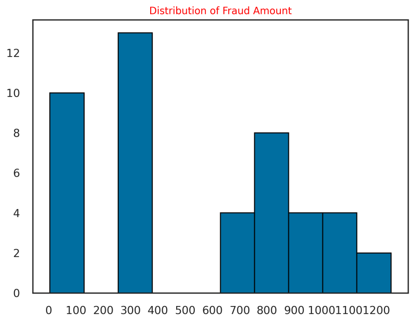
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7815 entries, 0 to 7814
Data columns (total 18 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   merchant    7815 non-null   object 
 1   category    7815 non-null   object 
 2   amt         7815 non-null   float64
 3   gender      7814 non-null   object 
 4   city        7814 non-null   object 
 5   state       7814 non-null   object 
 6   zip         7814 non-null   float64
 7   lat         7814 non-null   float64
 8   long        7814 non-null   float64
 9   city_pop    7814 non-null   float64
 10  job         7814 non-null   object 
 11  unix_time   7814 non-null   float64
 12  merch_lat   7814 non-null   float64
 13  merch_long  7814 non-null   float64
 14  is_fraud    7814 non-null   float64
 15  age         7815 non-null   int64  
 16  trans_date  7815 non-null   object 
 17  trans_time  7815 non-null   object 
```

```
dtypes: float64(9), int64(1), object(8)
memory usage: 1.1+ MB
```

```
# Fraud Vs Amount
plt.figure(figsize=[15,5])
plt.subplot(1,2,1)
plt.title('Fraud Vs Transaction Amount', fontsize= 10, color = 'Red', fontweight = 100)
plt.scatter(fraud.amt, fraud.is_fraud)
plt.subplot(1,2,2)
#fraud.groupby('is_fraud')['amt'].mean().plot.bar()
#plt.xticks((0,1),['Not Fraud', 'Fraud'])
#plt.xticks(rotation=0)
temp = fraud[fraud.is_fraud == 1]
plt.title('Distribution of Fraud Amount', fontsize= 10, color = 'Red', fontweight = 100)
plt.hist(temp.amt, edgecolor='Black')
plt.xticks(np.arange(0, 1300, step=100))

plt.show()
```





As can be seen from above, frauds are happening in transactions with lower amount hence indicating there is a relation in them.

```
# Fraud transactions Vs merchant
# Total number of transactions per merchant
merch_tran_total = fraud.sort_values('merchant').groupby('merchant').count()['is_fraud']
merch_tran_total.head()
```

```
merchant
fraud_Abbott-Rogahn               11
fraud_Abbott-Steuber              12
fraud_Abernathy and Sons           6
fraud_Abshire PLC                 16
fraud_Adams, Kovacek and Kuhlman   8
Name: is_fraud, dtype: int64
```

```
# Total fraud transactions per merchant
merch_tran_fraud = fraud[fraud.is_fraud == 1]['merchant'].value_counts()
merch_tran_fraud.head()
```

```
merchant
fraud_Padberg-Welch              2
fraud_Koepp-Parker               2
fraud_Moen, Reinger and Murphy   2
fraud_Rau and Sons               2
fraud_Rutherford-Mertz           1
Name: count, dtype: int64
```

```
# Percent of fraud transactions per merchant
fraud_perc = merch_tran_fraud/ merch_tran_total * 100
fraud_perc.sort_values(ascending=False)
```

```
merchant
fraud_Stokes, Christiansen and Sipes    33.333333
fraud_Block-Parisian                    20.000000
fraud_Ankunding LLC                     16.666667
fraud_Mosciski Group                    16.666667
```

```
fraud_Goyette Inc                          12.500000
                                              ...
fraud_Zemlak Group                               NaN
fraud_Zemlak, Tillman and Cremin                 NaN
fraud_Ziemann-Waters                             NaN
fraud_Zieme, Bode and Dooley                     NaN
fraud_Zulauf LLC                                 NaN
Length: 693, dtype: float64
```

Baring a few merchants, most of them have equal distribution of transactions and hence this field may play important role in the model. Changing the alphabetic values to numeric as models expects numeric data.

```
# variable transformation - merchant
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
fraud.merchant = label_encoder.fit_transform(fraud.merchant)
fraud_test.merchant = label_encoder.fit_transform(fraud_test.merchant)
```

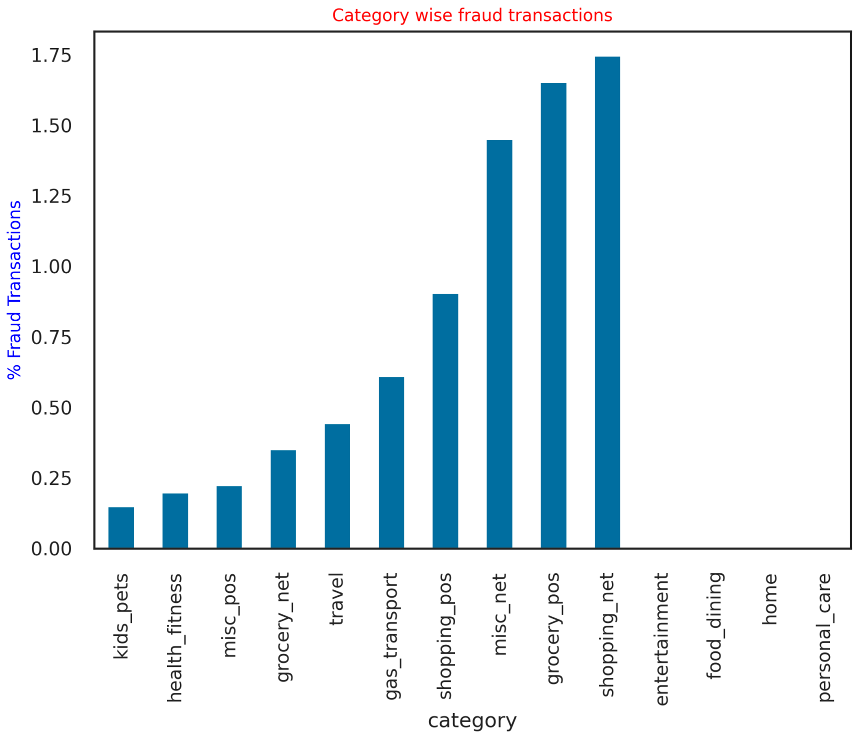
```
# Fraud transactions Vs City
# Percent distribution of fraud based on city
city_tran_total = fraud.sort_values('city').groupby('city').count()['is_fraud']
city_tran_fraud = fraud[fraud.is_fraud == 1]['city'].value_counts()
fraud_perc = city_tran_fraud/ city_tran_total * 100
fraud_perc.sort_values(ascending=False).head()
```

```
city
Collettsville    72.727273
Manor            70.588235
Wales            62.500000
Browning         20.000000
San Antonio      17.948718
dtype: float64
```

As can be seen, few cities have all transactions as fraud. All these cities have low transaction rate. There are 58 such cities.

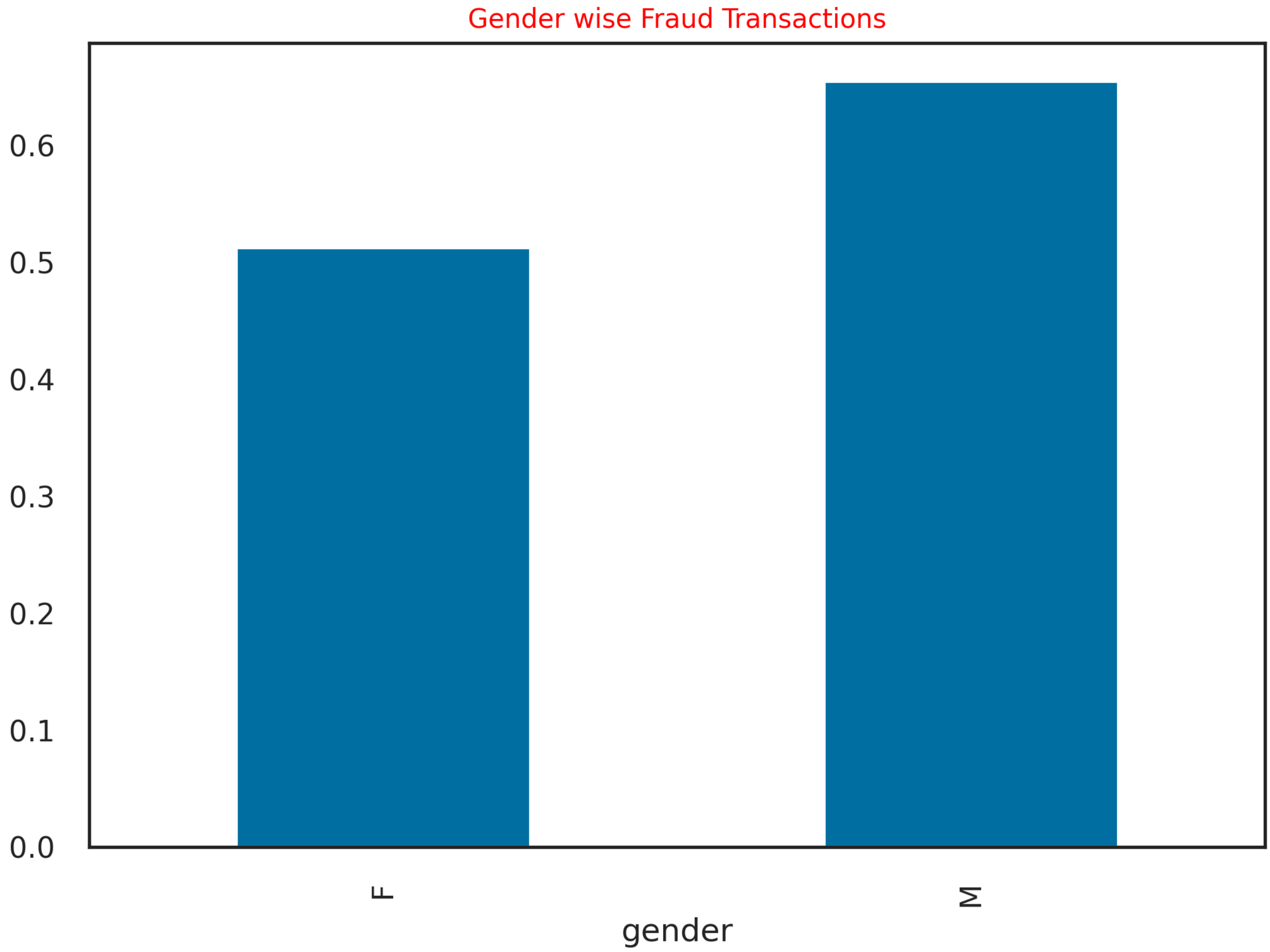
```
# Transforming alphabetic city data into numeric to be processed by the model
fraud.city = label_encoder.fit_transform(fraud.city)
fraud_test.city = label_encoder.fit_transform(fraud_test.city)
```

```
# category Vs fraud
# Percent distribution of fraud based on transaction category
cat_tran_total = fraud.sort_values('category').groupby('category').count()['is_fraud']
cat_tran_fraud = fraud[fraud.is_fraud == 1]['category'].value_counts()
fraud_perc = cat_tran_fraud/ cat_tran_total * 100
plt.title('Category wise fraud transactions', fontsize= 10, color = 'Red', fontweight = 100)
plt.ylabel('% Fraud Transactions', fontdict = {'fontsize': 10, 'color': 'Blue', 'fontweight' : '300'})
fraud_perc.sort_values().plot.bar()
plt.show()
```



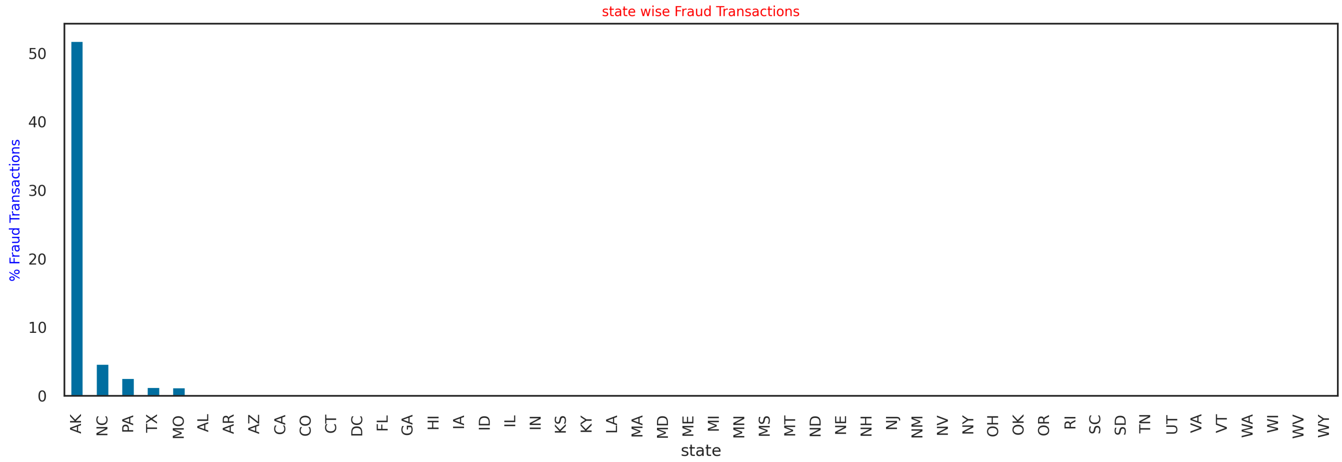

```
# Transforming alphabetic category data into numeric to be processed by the model
fraud.category = label_encoder.fit_transform(fraud.category)
fraud_test.category = label_encoder.fit_transform(fraud_test.category)
```

```
# Gender Vs Fraud
# Percent distribution of fraud based on Gender
gen_tran_total = fraud.sort_values('gender').groupby('gender').count()['is_fraud']
gen_tran_fraud = fraud[fraud.is_fraud == 1]['gender'].value_counts()
fraud_perc = gen_tran_fraud/ gen_tran_total * 100
plt.title('Gender wise Fraud Transactions', fontsize= 10, color = 'Red', fontweight = 100)
fraud_perc.sort_values().plot.bar()
plt.show()
```




```
# Transforming alphabetic gender data into numeric to be processed by the model
fraud.gender = fraud.gender.map({'M': 1, "F": 0})
fraud_test.gender = fraud_test.gender.map({'M': 1, "F": 0})
```

```
# state Vs fraud
# Percent distribution of fraud based on State
plt.figure(figsize = (17,5))
state_tran_total = fraud.sort_values('state').groupby('state').count()['is_fraud']
state_tran_fraud = fraud[fraud.is_fraud == 1]['state'].value_counts()
fraud_perc = state_tran_fraud/ state_tran_total * 100
plt.title('state wise Fraud Transactions', fontsize= 10, color = 'Red', fontweight = 100)
plt.ylabel('% Fraud Transactions', fontdict = {'fontsize': 10, 'color': 'Blue', 'fontweight' : '300'})
fraud_perc.sort_values(ascending=False).plot.bar()
plt.show()
```




```
fraud_perc.sort_values(ascending=False).head()
```

```
state
AK    51.724138
NC     4.651163
PA     2.553191
TX     1.252236
MO     1.176471
dtype: float64
```

This is very significant. While the number of transactions in DE is very less, all of them are fraud transaction. Rest all the states have very low fraud transaction.

```
# Transforming alphabetic state data into numeric to be processed by the model
fraud.state = label_encoder.fit_transform(fraud.state)
fraud_test.state = label_encoder.fit_transform(fraud_test.state)
```

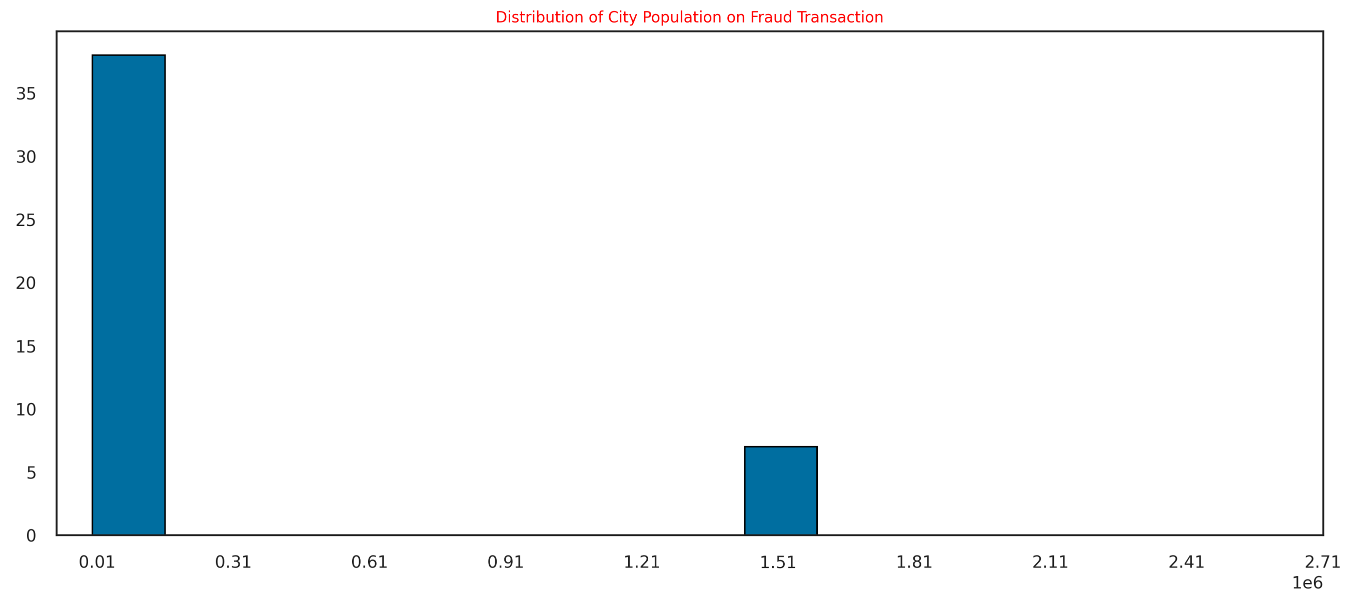
```
# Job Vs Fraud
# Percent distribution of fraud based on Job
job_tran_total = fraud.sort_values('job').groupby('job').count()['is_fraud']
job_tran_fraud = fraud[fraud.is_fraud == 1]['job'].value_counts()
fraud_perc = job_tran_fraud/ job_tran_total * 100
fraud_perc.sort_values(ascending=False).head(20)
```

```
job
Horticultural consultant                             70.000000
Public affairs consultant                            60.000000
Administrator, education                             46.875000
Soil scientist                                       44.444444
Cytogeneticist                                        8.823529
Academic librarian                                         NaN
Accountant, chartered certified                            NaN
Accountant, chartered public finance                       NaN
Accounting technician                                      NaN
Acupuncturist                                              NaN
Administrator                                              NaN
Administrator, arts                                        NaN
Administrator, charities/voluntary organisations           NaN
Administrator, local government                            NaN
Advertising account executive                              NaN
Advertising account planner                                NaN
Advertising copywriter                                     NaN
Advice worker                                              NaN
Aeronautical engineer                                      NaN
Agricultural consultant                                    NaN
dtype: float64
```

There seems certain jobs that have real high % of fraud transactions.

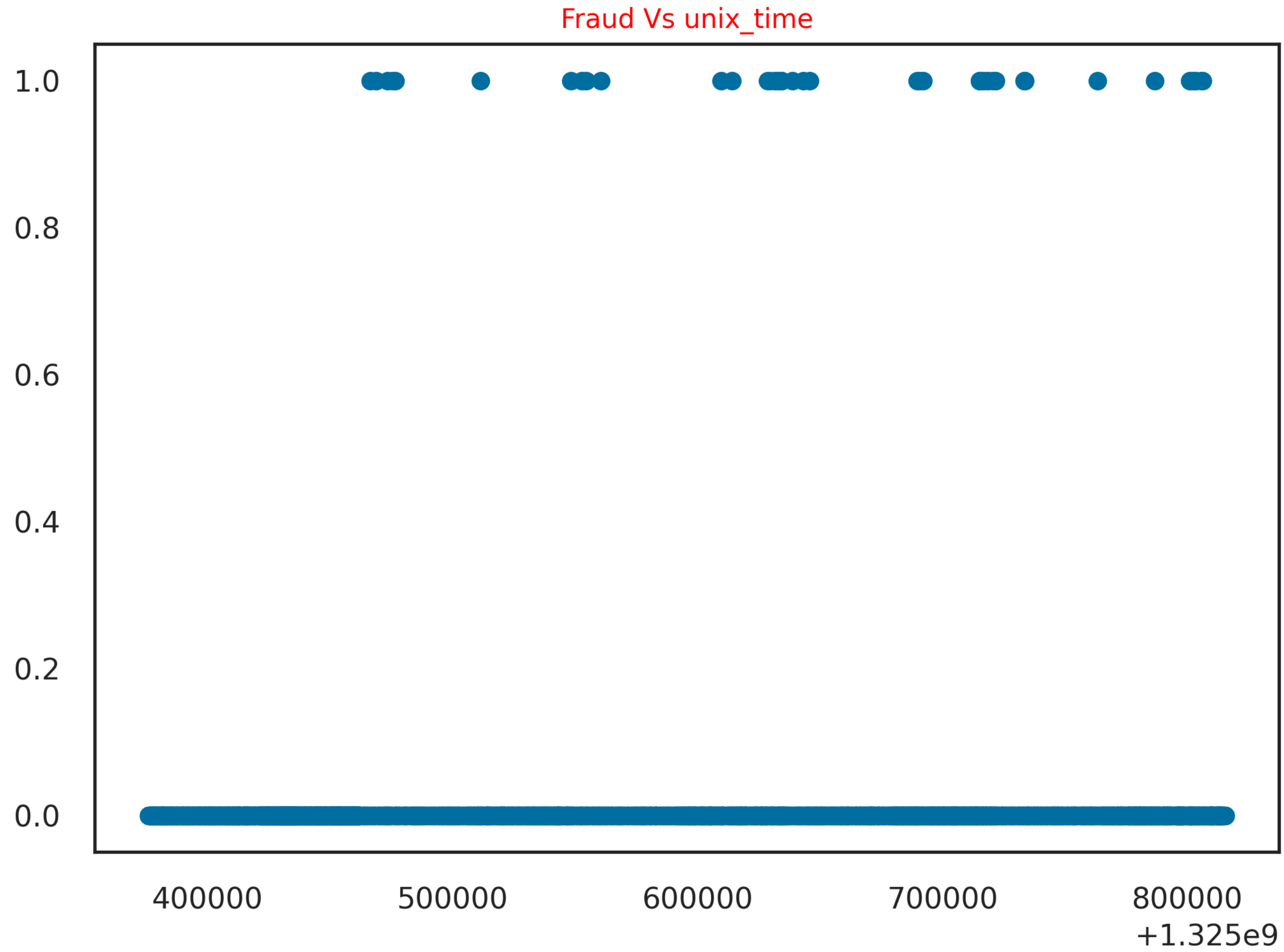
```
# Transforming alphabetic job data into numeric to be processed by the model
fraud.job = label_encoder.fit_transform(fraud.job)
fraud_test.job = label_encoder.fit_transform(fraud_test.job)
```

```
# Fraud Vs City Population
plt.figure(figsize=[15,6])
temp = fraud[fraud.is_fraud == 1]
plt.title('Distribution of City Population on Fraud Transaction', fontsize= 10, color = 'Red', fontweight = 100)
plt.hist(temp.city_pop, edgecolor='Black')
plt.xticks(np.arange(10000, 3000000, step=300000))
plt.show()
```

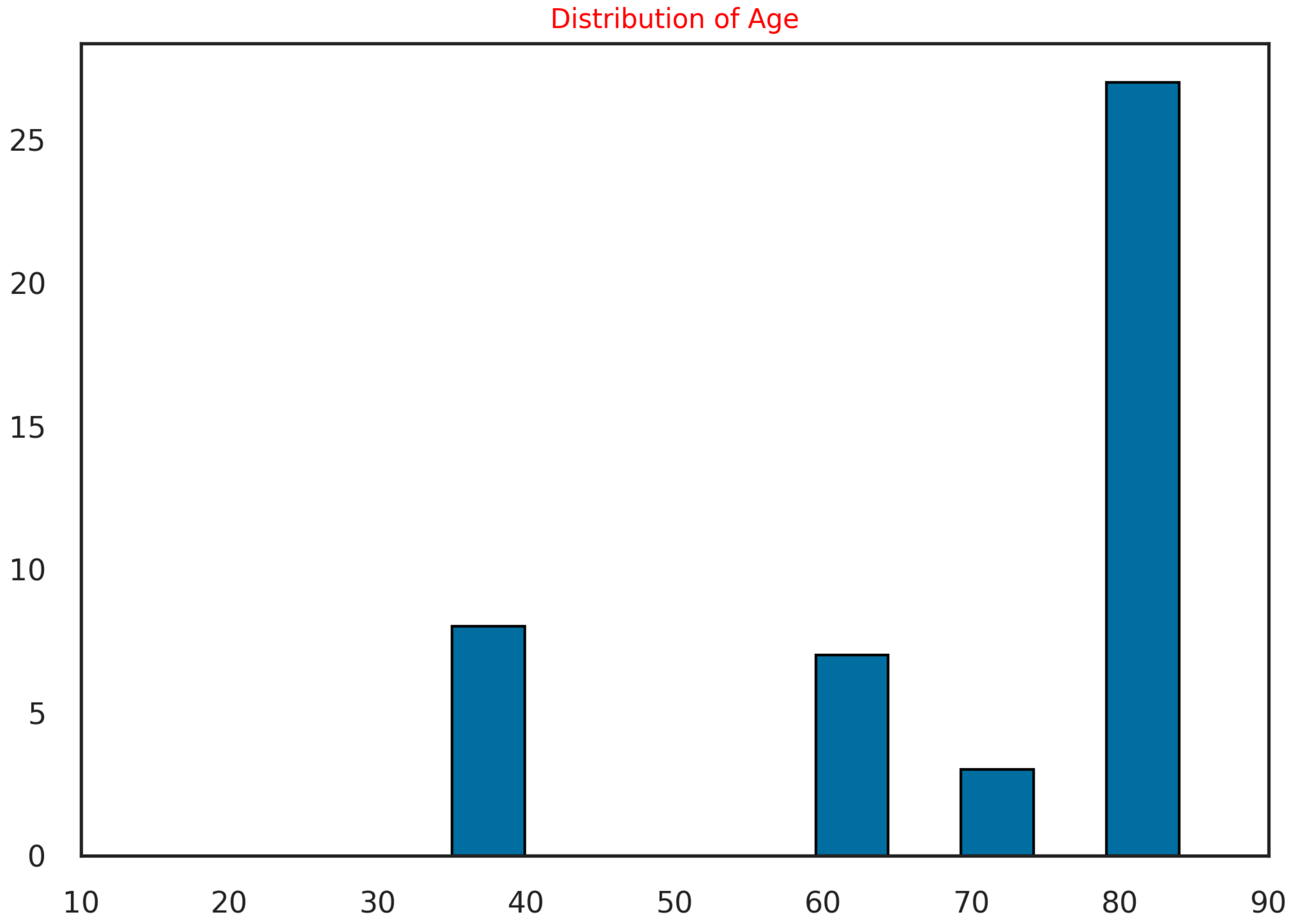


Cities with less population, tends to have more fraud cases.

```
# Fraud Vs Unix Time
plt.title('Fraud Vs unix_time', fontsize= 10, color = 'Red', fontweight = 100)
plt.scatter(fraud.unix_time, fraud.is_fraud)
plt.show()
```




```
# Fraud Vs Age
temp = fraud[fraud.is_fraud == 1]
plt.title('Distribution of Age', fontsize= 10, color = 'Red', fontweight = 100)
plt.hist(temp.age, edgecolor='Black')
plt.xticks(np.arange(10, 100, step=10))
plt.show()
```



So, people in age group 50 to 60 tends to be slightly more victims of fraud.

```
# Fraud Vs Zip
zip_tran_total = fraud.sort_values('zip').groupby('zip').count()['is_fraud']
zip_tran_fraud = fraud[fraud.is_fraud == 1]['zip'].value_counts()
fraud_perc = zip_tran_fraud/ zip_tran_total * 100
fraud_perc.sort_values(ascending=False).head(25)
```

```
zip
78208.0    100.000000
28611.0     72.727273
15665.0     70.588235
99783.0     62.500000
64630.0     20.000000
1257.0            NaN
1330.0            NaN
1535.0            NaN
1545.0            NaN
1612.0            NaN
1843.0            NaN
1844.0            NaN
2180.0            NaN
2630.0            NaN
2908.0            NaN
3220.0            NaN
3452.0            NaN
3601.0            NaN
3753.0            NaN
3774.0            NaN
3816.0            NaN
3818.0            NaN
3858.0            NaN
3905.0            NaN
```

```
4047.0          NaN
dtype: float64
```

As is evident from above stats, there are perticular ZIP codes that have 100% frauds.

```
# Fraud Vs lat
lat_tran_total = fraud.sort_values('lat').groupby('lat').count()['is_fraud']
lat_tran_fraud = fraud[fraud.is_fraud == 1]['lat'].value_counts()
fraud_perc = lat_tran_fraud/ lat_tran_total * 100
fraud_perc.sort_values(ascending=False).head()
```

```
lat
29.4400    100.000000
35.9946     72.727273
40.3359     70.588235
64.7556     62.500000
40.0290     20.000000
dtype: float64
```

As is evident from above stats, there are perticular latitudes codes that have 100% frauds.

```
# Fraud Vs long
long_tran_total = fraud.sort_values('long').groupby('long').count()['is_fraud']
long_tran_fraud = fraud[fraud.is_fraud == 1]['long'].value_counts()
fraud_perc = long_tran_fraud/ long_tran_total * 100
fraud_perc.sort_values(ascending=False).head()
```

```
long
-98.4590     100.000000
-81.7266      72.727273
-79.6607      70.588235
-165.6723     62.500000
-93.1607      20.000000
dtype: float64
```

```
# Fraud Vs merch_lat
lat_tran_total = fraud.sort_values('merch_lat').groupby('merch_lat').count()['is_fraud']
lat_tran_fraud = fraud[fraud.is_fraud == 1]['merch_lat'].value_counts()
fraud_perc = lat_tran_fraud/ lat_tran_total * 100
fraud_perc.sort_values(ascending=False).head()
```

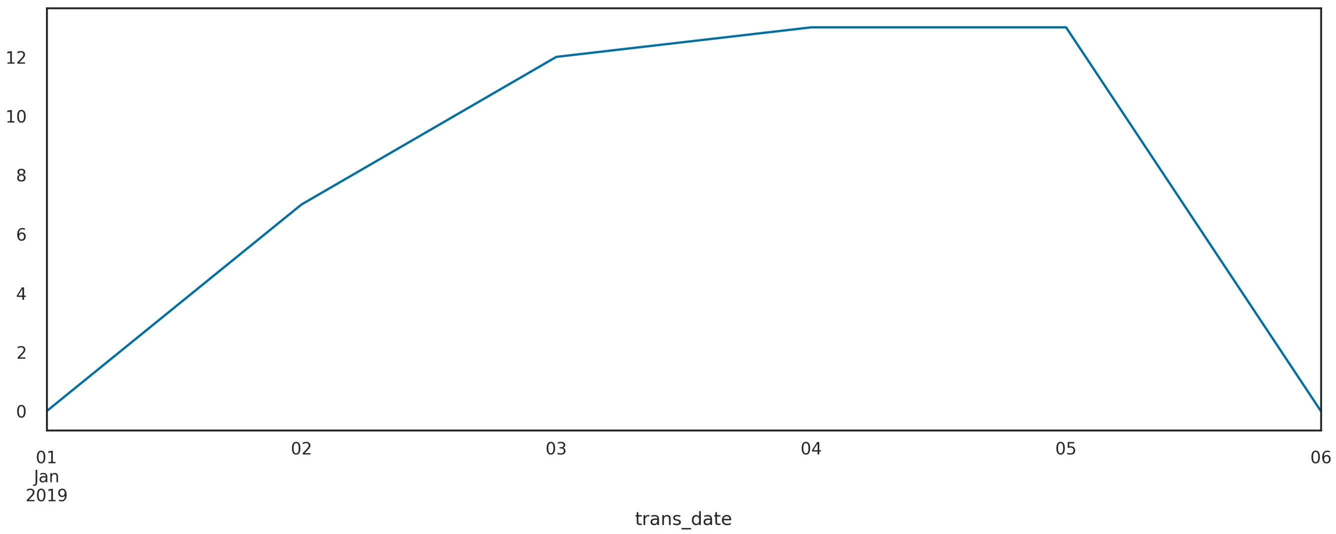
```
merch_lat
28.856712    100.0
40.346282    100.0
40.578351    100.0
40.601968    100.0
41.126312    100.0
dtype: float64
```

```
# Fraud Vs merch_long
long_tran_total = fraud.sort_values('merch_long').groupby('merch_long').count()['is_fraud']
long_tran_fraud = fraud[fraud.is_fraud == 1]['merch_long'].value_counts()
fraud_perc = long_tran_fraud/ long_tran_total * 100
fraud_perc.sort_values(ascending=False).head()
```

```
merch_long
-166.550779    100.0
-93.499211     100.0
-82.091010     100.0
-81.951839     100.0
-81.593183     100.0
dtype: float64
```

1. There are multiple demographies - Zip, City, States, Latitudes, Longitudes and Job types that have only Fraud transactions.
2. Even though they have 100% frauds, the number of transactions is very low. For Example State DE had only 9 transactions in 2 years. Hence, it is very less likely to impact the model.

```
# Fraud Vs trans_date
fraud['trans_date'] =  pd.to_datetime(fraud['trans_date'])
plt.figure(figsize=[15,5])
fraud.groupby(['trans_date'])['is_fraud'].sum().plot()
plt.show()
```



Start coding or generate with AI.

Now its time to change date and time to a format more acceptable for modelling. Before that, lets pull some stats required for Cost sheet. Also, it may be noticed that the train data is for 1.5 years (full 2019 till mid of 2020) and test data is for last 6 months of 2020. This way we will be able to build model on 1.5 year of data and test it on future data and hence check model performance in future. We will do the Cost Benifit analysis on the entire data.

```
# Total number of months
date_fraud = fraud.trans_date
date_fraud_test = pd.to_datetime(fraud_test.trans_date)
date_fraud = date_fraud.dt.to_period('M')
date_fraud_test = date_fraud_test.dt.to_period('M')
date = pd.concat([date_fraud, date_fraud_test])
print ('total number of records in file: ', date.size)
```

```
total number of records in file:  23397
```

```
print ('Total number of months: ', date.value_counts().size)
```

```
Total number of months:  2
```

```
print ('Average transactions per month: ', round(date.size/date.value_counts().size,0) )
```

```
Average transactions per month:  11698.0
```

```
# Extracting fraud data
temp1 = fraud[['amt', 'is_fraud']]
temp2 = fraud_test[['amt', 'is_fraud']]
temp = pd.concat([temp1, temp2])
temp.shape
```

```
(23397, 2)
```

```
# Average frauds per month
fraud_temp = temp[temp.is_fraud == 1]
print ('Average fraud transactions per month: ', round(fraud_temp.shape[0]/ date.value_counts().size,0))
```

```
Average fraud transactions per month:  46.0
```

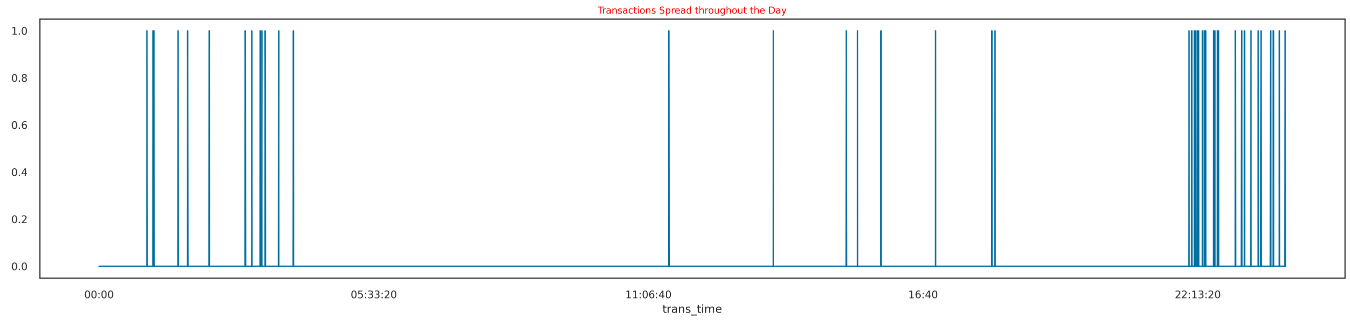
```
# Average amount per fraud transaction
print ('Average amount per fraud transaction: ', round(sum(fraud_temp.amt)/ fraud_temp.shape[0], 2))
```

```
Average amount per fraud transaction:  529.93
```

```
# Average amount per fraud transaction
print ('max fraud amount : ' , max(fraud_temp.amt))
```

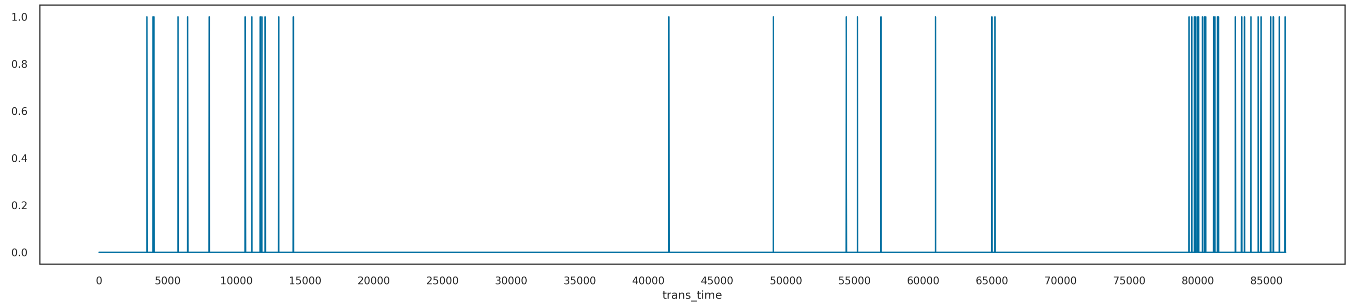
```
max fraud amount :  1254.27
```

```
# Fraud Vs trans_time
import datetime as dt
fraud.trans_date = fraud.trans_date.map(dt.datetime.toordinal)
plt.figure(figsize=[25,5])
plt.title('Transactions Spread throughout the Day', fontsize= 10, color = 'Red', fontweight = 100)
fraud.groupby(['trans_time'])['is_fraud'].sum().plot()
plt.show()
```



So, late nights and early mornings are the most prone time for frauds. Highest frequency of frauds is between 10 pm to 12 am. 12 am to 4:00 am also shows very high frequency of fraud transactions.

```
# Converting trans_time into seconds & ploting the above graph again
fraud.trans_time = pd.to_datetime(fraud.trans_time,format='%H:%M:%S')
fraud.trans_time = 3600 * pd.DatetimeIndex(fraud.trans_time).hour + 60 * pd.DatetimeIndex(fraud.trans_time).minute + pd.DatetimeIndex(fr
plt.figure(figsize=[25,5])
plt.xticks(np.arange(0,90000,5000))
fraud.groupby(['trans_time'])['is_fraud'].sum().plot()
plt.show()
```




```
# Similar data-time changes in test dataset
fraud_test['trans_date'] =  pd.to_datetime(fraud_test['trans_date'])
fraud_test.trans_date = fraud_test.trans_date.map(dt.datetime.toordinal)
fraud_test.trans_time = pd.to_datetime(fraud_test.trans_time,format='%H:%M:%S')
fraud_test.trans_time = 3600 * pd.DatetimeIndex(fraud_test.trans_time).hour + 60 * pd.DatetimeIndex(fraud_test.trans_time).minute + pd.I
```

```
print ('train : ', fraud.shape)
print ('test : ', fraud_test.shape)
```

```
train :  (7815, 18)
test :  (15582, 18)
```

```
fraud
```

	merchant	category	amt	gender	city	state	zip	lat	long	city_pop	job	unix_time	merch_lat	merch_long
0	514	8	4.97	0.0	489	26	28654.0	36.0788	-81.1781	3495.0	354	1.325376e+09	36.011293	-82.048315
1	241	4	107.23	0.0	563	46	99160.0	48.8878	-118.2105	149.0	409	1.325376e+09	49.159047	-118.186462
2	390	0	220.11	1.0	436	12	83252.0	42.1808	-112.2620	4154.0	293	1.325376e+09	43.150704	-112.154481
3	360	2	45.00	1.0	81	25	59632.0	46.2306	-112.1138	1939.0	314	1.325376e+09	47.034331	-112.561071
4	297	9	41.96	1.0	202	44	24433.0	38.4207	-79.4629	99.0	110	1.325376e+09	38.674999	-78.632459
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7810	447	4	132.27	0.0	591	1	35581.0	34.3470	-87.7154	5778.0	146	1.325815e+09	34.011938	-88.080823
7811	217	4	83.51	1.0	302	37	16421.0	42.1767	-79.9416	2518.0	177	1.325815e+09	41.328228	-79.226936
7812	329	4	64.04	0.0	12	9	30009.0	34.0770	-84.3033	165556.0	336	1.325815e+09	33.376781	-83.592263
7813	518	2	66.86	1.0	557	11	52576.0	41.2001	-92.1354	568.0	88	1.325815e+09	40.955293	-92.395676
7814	437	0	64.98	NaN	827	50	NaN	NaN	NaN	NaN	473	NaN	NaN	NaN

7815 rows × 18 columns

```
corr = fraud.corr()
corr.style.background_gradient(cmap='coolwarm')
```

	merchant	category	amt	gender	city	state	zip	lat	long	city_pop	job	unix_tim
merchant	1.000000	0.018463	-0.013524	0.008124	-0.002060	0.002619	0.000346	-0.015625	-0.000590	-0.010285	0.007849	-0.00854
category	0.018463	1.000000	0.025268	-0.030596	0.014957	0.004953	-0.004188	0.005723	0.000181	0.014303	0.008802	0.00113
amt	-0.013524	0.025268	1.000000	0.008920	0.000675	-0.012178	0.005582	0.057857	-0.037101	0.004502	-0.014733	0.02078
gender	0.008124	-0.030596	0.008920	1.000000	0.012364	-0.038091	-0.070584	0.057593	0.049783	-0.014348	-0.080383	0.00805
city	-0.002060	0.014957	0.000675	0.012364	1.000000	-0.043472	0.057344	-0.045346	-0.054509	0.034533	0.025156	0.01312
state	0.002619	0.004953	-0.012178	-0.038091	-0.043472	1.000000	-0.118987	0.183797	0.142097	0.001101	0.050927	0.00091
zip	0.000346	-0.004188	0.005582	-0.070584	0.057344	-0.118987	1.000000	-0.087483	-0.898469	0.061986	-0.000131	-0.01120
lat	-0.015625	0.005723	0.057857	0.057593	-0.045346	0.183797	-0.087483	1.000000	-0.057590	-0.156403	-0.051465	0.00728
long	-0.000590	0.000181	-0.037101	0.049783	-0.054509	0.142097	-0.898469	-0.057590	1.000000	-0.030264	-0.003195	-0.00037
city_pop	-0.010285	0.014303	0.004502	-0.014348	0.034533	0.001101	0.061986	-0.156403	-0.030264	1.000000	-0.030347	0.00858
job	0.007849	0.008802	-0.014733	-0.080383	0.025156	0.050927	-0.000131	-0.051465	-0.003195	-0.030347	1.000000	-0.03297
unix_time	-0.008543	0.001131	0.020781	0.008050	0.013124	0.000912	-0.011201	0.007285	-0.000371	0.008582	-0.032978	1.00000
merch_lat	-0.013532	0.006353	0.058029	0.056617	-0.043000	0.181684	-0.086826	0.993910	-0.057369	-0.156128	-0.050427	0.00758
merch_long	0.000116	0.000090	-0.036639	0.049713	-0.055034	0.141988	-0.897820	-0.057604	0.999199	-0.030422	-0.002915	-0.00079
is_fraud	-0.000416	0.022512	0.306440	0.009375	0.032716	-0.017157	0.026653	0.108245	-0.114512	0.041524	-0.015587	0.04667
age	0.003909	-0.001879	0.024025	-0.001692	-0.007470	-0.063495	0.036377	0.060088	-0.060806	-0.094439	-0.058437	-0.02235
trans_date	-0.009518	-0.029511	0.029904	0.011175	0.012901	-0.001762	-0.011345	0.009277	-0.001156	0.007524	-0.034568	0.98298
trans_time	0.005646	0.163827	-0.048890	-0.016702	0.003550	0.016493	0.000502	-0.010591	0.004236	0.005929	0.009916	0.11634

Start coding or generate with AI.

Advance EDA (optional)

```
df_train = fraud
df_test = fraud_test
```

```
df_train.rename(columns={"trans_date_trans_time":"transaction_time",
                         "cc_num":"credit_card_number",
                         "amt":"amount(usd)",
                         "trans_num":"transaction_id"},
                inplace=True)
```

> **Convert datetime columns**
> *transaction_time* and *dob* should be in pd.datetime format and we also convert unix_time to exact timestamp

```
df_train["trans_date_trans_time"] = pd.to_datetime(df_train["trans_date_trans_time"], infer_datetime_format=True)
df_train["dob"] = pd.to_datetime(df_train["dob"], infer_datetime_format=True)
```

```
from datetime import datetime
df_train['timestamp'] = pd.to_datetime(df_train['unix_time'], unit='s')

# Apply function utcfromtimestamp and drop column unix_time

df_train.drop('unix_time', axis=1, inplace=True)
df_train['timestamp'] = pd.to_datetime(df_train['unix_time'], unit='s')
# Add cloumn hour of day
df_train['hour_of_day'] = df_train.time.dt.hour
```

```
df_train[['time','hour_of_day']]
```

> **Convert dtypes**
> Credit card number should be integer, let's change.

```
# Change dtypes
df_train.cc_num = df_train.cc_number.astype('category')
df_train.is_fraud = df_train.is_fraud.astype('category')
df_train.hour_of_day = df_train.hour_of_day.astype('category')

# Check
df_train.info()
```

```
np.round(df_train.describe(), 2)
```

	merchant	category	amount(usd)	gender	city	state	zip	lat	long	city_pop	job	merch_lat	merch_lon
count	7815.00	7815.00	7815.00	7814.00	7815.00	7815.00	7814.00	7814.00	7814.00	7814.00	7815.00	7814.00	7814.0
mean	341.21	6.17	68.66	0.45	410.35	25.77	49479.40	38.60	-90.69	88234.51	239.75	38.60	-90.6
min	0.00	0.00	1.01	0.00	0.00	0.00	1257.00	20.03	-165.67	23.00	0.00	19.17	-166.5
25%	165.00	3.00	9.78	0.00	201.00	14.00	26041.00	34.85	-97.24	741.00	123.00	34.91	-97.3
50%	344.00	6.00	48.49	0.00	406.00	26.00	49259.00	39.38	-87.76	2395.00	238.00	39.39	-87.7
75%	512.00	10.00	82.22	1.00	625.00	37.00	72476.00	41.85	-80.14	19054.00	360.00	41.92	-80.1
max	692.00	13.00	3178.51	1.00	827.00	50.00	99783.00	65.69	-67.95	2906700.00	473.00	66.65	-66.9
std	199.63	3.87	115.53	0.50	240.41	14.24	27184.83	5.20	14.51	294446.99	135.67	5.23	14.5

> Quick Summarize using pandas_profiling

```
groups = [pd.Grouper(key="transaction_time", freq="1W"), "is_fraud"]
df_ = df_train.groupby(by=groups).agg({"amount(usd)":'mean',"transaction_id":"count"}).reset_index()
```

```
def add_traces(df, x, y,hue, mode, cmap, showlegend=None):
    name_map = {1:"Yes", 0:"No"}
    traces = []
    for flag in df[hue].unique():
        traces.append(
            go.Scatter(
                x=df[df[hue]==flag][x],
                y=df[df[hue]==flag][y],
                mode=mode,
                marker=dict(color=cmap[flag]),
                showlegend=showlegend,
                name=name_map[flag]
            )
        )
    return traces
```

```
fig = make_subplots(rows=2, cols=2,
                    specs=[
                        [{}, {}],
                        [{"colspan":2}, None]
                    ],
                    subplot_titles=("Amount(usd) over time", "Number of transactions overtime",
                                    "Number of transaction by amount(usd)")
                   )

ntraces = add_traces(df=df_,x='transaction_time',y='amount(usd)',hue='is_fraud',mode='lines',
                    showlegend=True, cmap=['#61E50F','#D93C1D'])

for trace in ntraces:
    fig.add_trace(
        trace,
        row=1,col=1
    )

ntraces = add_traces(df=df_,x='transaction_time',y='transaction_id',hue='is_fraud',mode='lines',
                    showlegend=False, cmap=['#61E50F','#D93C1D'])
for trace in ntraces:
    fig.add_trace(
        trace,
        row=1,col=2
    )

ntraces = add_traces(df=df_,x='transaction_id',y='amount(usd)',hue='is_fraud',mode='markers',
                    showlegend=True, cmap=['#61E50F','#D93C1D'])
for trace in ntraces:
    fig.add_trace(
        trace,
        row=2,col=1
    )

fig.update_layout(height=780,
                  width=960,
                  legend=dict(title='Is fraud?'),
                  plot_bgcolor='#fafafa',
                  title='Overview'
                 )

fig.show()


df_ = df_train.groupby(by=[pd.Grouper(key="transaction_time", freq="1W"),
                           'is_fraud','category']).agg({"amount(usd)":'mean',"transaction_id":"count"}).reset_index()

fig = px.scatter(df_,
        x='transaction_time',
        y='amount(usd)',
        color='is_fraud',
        facet_col ='category',
        facet_col_wrap=3,
        facet_col_spacing=.04,
        color_discrete_map={0:'#61E50F', 1:'#D93C1D'}
)

fig.update_layout(height=1400,
                  width=960,
                  legend=dict(title='Is fraud?'),
                  plot_bgcolor='#fafafa'
                 )

fig.update_yaxes(matches=None)
fig.for_each_yaxis(lambda yaxis: yaxis.update(showticklabels=True))
fig.for_each_xaxis(lambda xaxis: xaxis.update(showticklabels=True, title=''))

fig.show();
```

```
df_ = df_train.groupby(by=[pd.Grouper(key="transaction_time", freq="1M"),
                           'is_fraud','category']).agg({"amount(usd)":'sum',"transaction_id":"count"}).reset_index()

fig = px.area(
    df_[df_.is_fraud==1],
    x='transaction_time',
    y='amount(usd)',
    color='category',
    color_discrete_sequence=px.colors.qualitative.Dark24
)

fig.update_layout(height=600,
                  width=960,
                  legend=dict(title='Categories'),
                  plot_bgcolor='#fafafa'
                 )

fig.show();
```

```
# Specified list of 12 merchants with the highest number of transactions.
top12_merchants = df_train.merchant.value_counts()[:12]

df_ = df_train.groupby(by=[pd.Grouper(key="transaction_time", freq="1W"),'is_fraud',
                           'merchant']).agg({"amount(usd)":'mean',"transaction_id":"count"}).reset_index()

df_ = df_[df_.merchant.isin(top12_merchants.index)]


fig = px.scatter(df_,
        x='transaction_time',
        y='amount(usd)',
        color='is_fraud',
        facet_col ='merchant',
        facet_col_wrap=3,
        facet_col_spacing=.06,
        category_orders={'merchant': top12_merchants.index}, # order the subplots
        color_discrete_map={1:'#61E50F', 0:'#D93C1D'}
)

fig.update_layout(height=1200,
                  width=960,
                  title='Top 12 merchants with highest number of transactions per week',
                  legend=dict(title='Is fraud?'),
                  plot_bgcolor='#fafafa'
                 )

fig.update_yaxes(matches=None)
fig.for_each_yaxis(lambda yaxis: yaxis.update(showticklabels=True))
fig.for_each_xaxis(lambda xaxis: xaxis.update(showticklabels=True, title=''))

fig.show();
```

```
# df_ = df_train[df_train.is_fraud==1].groupby(by='hour_of_day').agg({'transaction_id':'count'}).reset_index()

# fig = px.bar(data_frame=df_,
#        x='hour_of_day',
#        y='transaction_id',
#        labels={'transaction_id':'Number of transaction'})

# fig.update_layout(
#     title=dict(
#         text='Number of FRAUD transactions by hours of day'
#     ),
#     plot_bgcolor='#fafafa'
# )

# fig.update_xaxes(type='category')
```

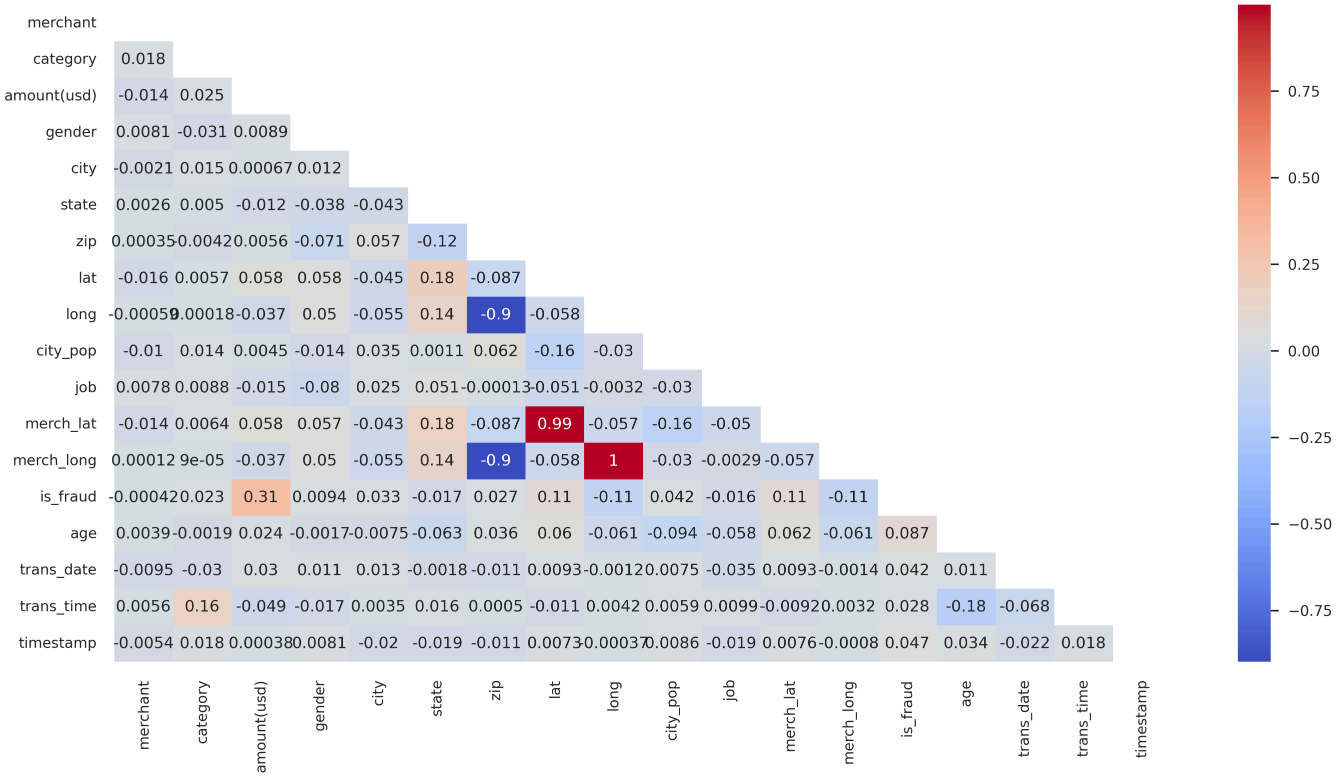
```
df_train.dtypes
```

```
merchant             int64
category             int64
amount(usd)        float64
gender             float64
city                 int64
state                int64
zip                float64
lat                float64
long               float64
city_pop           float64
job                  int64
merch_lat          float64
```

```
merch_long           float64
is_fraud             float64
age                    int64
trans_date             int64
trans_time             int32
timestamp     datetime64[ns]
dtype: object
```

```
%matplotlib inline
fig = plt.figure(figsize=(18,9))
mask = np.triu(np.ones_like(df_train.corr()))
sns.heatmap(df_train.corr(), mask=mask, cmap='coolwarm', annot=True)
```

```
<Axes: >
```



Next, build the model to predict Fraud Transactions(label "1")
Target: The higher **F1-Score** for label 1, the better the model!

Double-click (or enter) to edit

Start coding or generate with AI.

Model Building

```
fraud
```