Credit Card Fraud Detication

The idea of the project is to detict fraud transactions by train classification model that fit this huge data and apply them into multiple transaction to check weather it's fraud or not.

** NOTE ** THE CODE WORK FINE BUT SOME CHARTS WILL NOT WERK BY READING THIS NOTEBOOK BECAUSE I NEED TO RUN IT ALL AT THE SEAM TIME AND MY DEVICE IS NOT CAPABLE OF DOING THAT

DATASET

This dataset contains credit card transactions for fictitious users resident within the United States, but who travel the world in making purchases.

This dataset has a header line followed by transactions -- with one line per transaction. The header describes the fields of each transaction, which are similar to the fields in a monthly credit card statement, e.g. (1) date and time of purchase, (2) merchant name, and (3) merchant location (city, US state, US zipcode -- with US state replaced by country name for purchases made outside the United States). Another field is the MCC (Merchant Category Code) -- a number from 1 - 9999, which describes the broad area of the merchant, e.g. groceries, clothing, hair care, jewelry, etc. These MCC values are standard values from the credit card industry.

The final columns are "Errors" and "Is Fraud". The "Is Fraud" column is the text string Yes when transactions are fraudulent, i.e. made with a stolen card and the text string No when transactions are made by the legitimate owner of the card. This "Is Fraud" baniry values that hold only yes or no, whenever training an unsupervised model to detect fraud, and later when assessing the accuracy of fraud inference.

The "Errors" field enumerates whether a problem like "Technical Glitch", "Bad PIN", "Insufficient Funds", etc occurred. A blank value for "Errors" indicates no errors were present.

The "User" field in the first column is a numeric index from 0 - 1999 indicating which of the 2000 users generated each transactions. The "Card" field in the second column is similarly an index among all of the credit (and debit) cards owned by "User". The transactions in the file are ordered as follows: By User By Card of User

in this notebook after each line of code there are an explonation of it.

Import

Typesetting math: 0%

In [37]:
```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
import lightgbm as lgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import  confusion_matrix, accuracy_score
from sklearn.metrics import classification_report
import requests
```

Data Cleaning

In [2]:
```
df = pd.read_csv("Dataset/credit_card_transactions-ibm_v2.csv")
```

In [4]:
```
df.sample(5)
```

Out[4]:

	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name
10585157	885	1	2017	5	12	13:41	$37.24	Chip Transaction	-3739862438923451178
23720734	1934	3	2019	7	24	13:06	$80.00	Swipe Transaction	-4282466774399734331
13193150	1081	4	2003	1	12	09:21	$47.50	Swipe Transaction	6601217045817418951
8693804	751	1	2018	1	23	20:33	$-77.00	Chip Transaction	-1288082279022882052
9347404	801	2	2015	5	12	09:05	$20.61	Online Transaction	-2088492411650162548

Data type of the featuers

In [4]:
```
df.dtypes.to_frame(name='Type').T
```

Out[4]:

	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State
Type	int64	int64	int64	int64	int64	object	object	object	int64	object	object

Typesetting math: 0%

We can see that from the table above that the time is object and we want to extract the hour and the minuts form it to represent them in numbercal value and use it later in the model and in EDA

Similarly, User and Card features should also be converted to object type as unique User values represent different users, and card values correspond to an index of a card a particular user uses. The card values do not specify a unique card number.

Additionally Merchant Name, Zip and MCC are treated as numeric but should be processed as categorical.

However, since this dataset is large, to work with the kaggle kernel for visualization purposes we will not convert to object/category type. Also, we will downcast all numeric datatypes to reduce memory consumption.

So let's convert these features to the desired datatypes

In [5]:
```
#create new feature from time instade of that feature and remove the dolar si
df['Amount'] = df['Amount'].apply(lambda value: value.split("$")[1])
df['Hour'] = df['Time'].apply(lambda value: value.split(":")[0])
df['Minutes'] = df['Time'].apply(lambda value: value.split(":")[1])
# drop time feature
df.drop(['Time'],axis=1,inplace=True)

# convert the data type to decrease the number of data takes
convertion = {'Amount': np.float32, 'Minutes': np.uint8, 'Hour': np.uint8,
              'Year': np.int16, 'Month': np.uint8, 'Day': np.uint8, 'Zip': np
              'User':np.uint16, 'Card':np.uint8}

df = df.astype(convertion)
```

checking the data type after changing them

In [27]:
```
df.dtypes.to_frame(name='Type').T
```

Out[27]:

	User	Card	Year	Month	Day	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zi
Type	uint16	uint8	int16	uint8	uint8	float32	object	int64	object	object	float1

The data now looks like this:

We see that the extra decimals in ZipCode no longer exist as the type is converted to object, we have two more columns: Hour and Minute whereas Time doesn't exist

Typesetting math: 0%

In [26]:
```
df.sample(5)
```

Out[26]:

	User	Card	Year	Month	Day	Amount	Use Chip	Merchant Name	M
9712252	819	0	2016	8	9	9.280000	Chip Transaction	6698459923198770712	San
9411031	805	1	1998	3	21	57.299999	Swipe Transaction	112925206871091074	Phila
20405427	1662	3	2005	11	11	109.099998	Swipe Transaction	2268665024076934393	.
13929667	1135	0	2016	3	19	7.210000	Chip Transaction	-4693979874497918566	Jc
9500507	808	0	2018	5	14	75.639999	Chip Transaction	-5162038175624867091	

Typesetting math: 0%

In [7]:
```
df.describe(include='all').fillna("").T.style
```

```
C:\anaconda\lib\site-packages\numpy\lib\function_base.py:3961: RuntimeWarni
ng: invalid value encountered in subtract
  diff_b_a = subtract(b, a)
```

Out[7]:

	count	unique	top	freq	mean	
User	24386900.000000				1001.019335	
Card	24386900.000000				1.351366	
Year	24386900.000000				2011.955170	
Month	24386900.000000				6.525064	
Day	24386900.000000				15.718123	
Amount	24386900.000000				41.987522	
Use Chip	24386900	3	Swipe Transaction	15386082		
Merchant Name	24386900.000000				-4769229627664683 52.000000	47589
Merchant City	24386900	13429	ONLINE	2720821		
Merchant State	21666079	223	CA	2591830		
Zip	21508765.000000					
MCC	24386900.000000				5561.171253	
Errors?	388431	23	Insufficient Balance	242783		
Is Fraud?	24386900	2	No	24357143		
Hour	24386900.000000				12.414200	
Minutes	24386900.000000				29.585951	

From the basic statistics we can see that the dataset consists of **24386900 transactions, with 2000 unique users and a user owns at most 9 cards.**

The most common type of transactions are swipe transactions

The median amount is 30$ which is almost equal to one grocery shopping trip for a single person which coincides with the most common merchant category code: 5411 (Grocery and Supermarkets).

Missing Values and their proportion

Typesetting math: 0%

Missing Value Analysis

Based on how the data was generated, Merchant State and Zip are not present when a transaction is processed online. Additionally for tranactions which are not US based, Zipcode is missing.

For successful transactions, errors are absent and a mjaority of transactions in this dataset are processed without errors which explains the high missing ratio for the errors column

In [8]:
```
missing_count = df.isna().sum()
missing_df = (pd.concat([missing_count.rename('Missing Record'),
                         (missing_count.div(len(df)) * 100)
                          .rename('Percentage')],axis = 1)
             .loc[missing_count.ne(0)])
missing_df
```

Out[8]:

	Missing Record	Percentage
Merchant State	2720821	11.156896
Zip	2878135	11.801972
Errors?	23998469	98.407215

EDA

Distribution of transactions over the Months

In [10]:
```
fig = px.histogram(df, x="Month", color='Is Fraud?')
fig.update_layout(bargap=0.2, title="Transactions Distribution over the month
fig.show()
```

Distribution of transactions over the Hours in a Day

In [11]:
```
fig_hour = px.histogram(df, x="Hour")
fig.update_layout(bargap=0.09, title="Transactions Distribution over Hour")
fig_hour.show()
```

Fraudulent Transactions over the Years

Typesetting math: 0%

In [ ]:

```
df_year_fraud = df.loc[:,['Year','Is Fraud?']]

df_year_fraud = df_year_fraud.groupby(['Year'])['Is Fraud?'].value_counts().t

unique_year_vals = df.Year.unique()

to_plot_df = pd.DataFrame(columns=['Year', 'No','Yes'])

for year in unique_year_vals:
    try:
        no = df_year_fraud.loc[(year,'No')]['Count']
    except:
        no = 0
    try:
        yes = df_year_fraud.loc[(year,'Yes')]['Count']
    except:
        yes = 0
    to_plot_df = to_plot_df.append(pd.DataFrame([[year,no,yes]],columns=["Yea

to_plot_df['No'] = to_plot_df['No'].replace(0, np.nan)
to_plot_df['Yes'] = to_plot_df['Yes'].replace(0, np.nan)
to_plot_df['No'] = to_plot_df['No'].apply(lambda x: np.log10(x))
to_plot_df['Yes'] = to_plot_df['Yes'].apply(lambda x: np.log10(x))


fig = go.Figure(data=[
    go.Bar(name='Non-Fraud', x=to_plot_df.Year, y=to_plot_df.No),
    go.Bar(name='Fraud', x=to_plot_df.Year, y=to_plot_df.Yes)
])
fig.update_layout(barmode='group',title="Logartihmic Count of Fraud and Non-F
fig.show()
```

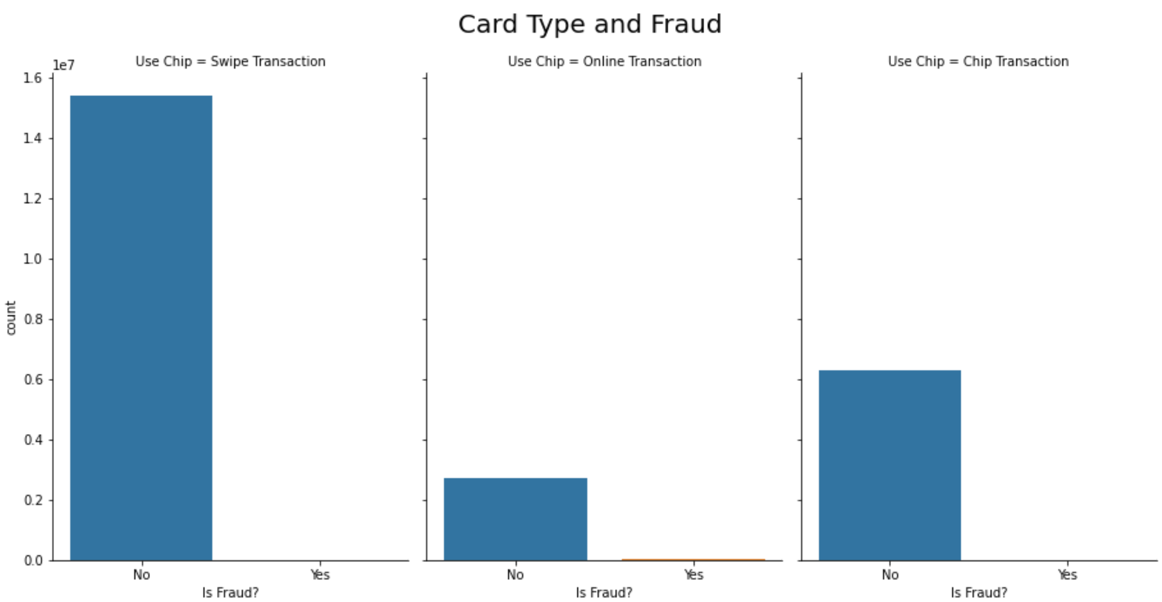
Fraudulent and Non fraudulent transactions based on type of card use

Typesetting math: 0%

In [ ]:
```
plot = sns.catplot("Is Fraud?", col="Use Chip",data=df,kind="count", height=6
plot.fig.suptitle("Card Type and Fraud", size = 20, y=1.05);
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWar
ning:

Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
```



Transactions per state in the US

Typesetting math: 0%

In [13]:

```
us_transactions = df[~df['Merchant State'].isna()]

transactions_per_state = us_transactions.groupby(['Merchant State'],as_index=

fraud_count_per_state = us_transactions.groupby(['Merchant State', 'Is Fraud?

merchant_state_fraud_dict = fraud_count_per_state.to_dict()

merchant_plot_df = pd.DataFrame(us_transactions['Merchant State'].value_count

merchant_plot_df.rename({'index':"State", 'Merchant State':"Total_Transactior

merchant_plot_df['FraudPercent'] = merchant_plot_df['State'].apply(lambda x:

merchant_plot_df['NonFraudPercent'] = merchant_plot_df['State'].apply(lambda

merchant_plot_df['FraudPercent'] = round(100 * (merchant_plot_df['FraudPercer

merchant_plot_df['NonFraudPercent'] = round(100 * (merchant_plot_df['NonFrauc

for col in merchant_plot_df.columns:
    merchant_plot_df[col] = merchant_plot_df[col].astype(str)


merchant_plot_df['text'] = "Fraudulent: " + merchant_plot_df['FraudPercent']


fig = go.Figure(data=go.Choropleth(
    locations=merchant_plot_df['State'],
    z=merchant_plot_df['Total_Transactions'].astype(float),
    locationmode='USA-states',
    colorscale='deep',
    autocolorscale=False,
    text=merchant_plot_df['text'], # hover text
    marker_line_color='white', # line markers between states
    colorbar_title="Credit Card Transactions"
))

fig.update_layout(
    title_text='Credit Card Transactions per US state',
    geo = dict(
        scope='usa',
        projection=go.layout.geo.Projection(type = 'albers usa'),
        showlakes=True, # lakes
        lakecolor='rgb(255, 255, 255)'),
)

fig.show()
```

Transactions Outside the US

Typesetting math: 0%

In [14]:
```
df_nonusa = df[(df.Zip.isnull()) & (df['Merchant City'] != 'ONLINE')]
print(f"Transactions not in the United States: {len(df_nonusa)}")
```

```
Transactions not in the United States: 157314
```

In [15]:
```
df_usa = df[(~df.Zip.isnull()) & (df['Merchant City'] != 'ONLINE')]
print(f"Transactions in the United States: {len(df_usa)}")
```

```
Transactions in the United States: 21508765
```

checks the correlation of the data to the feature is fraud? since it's the target

In [13]:
```
data = []
for i in range(len(df['Is Fraud?'])):
    if df['Is Fraud?'][i] == 'No':
        data.append(0)
    else:
        data.append(1)
```

In [14]:
```
data
```

Out[14]:
```
[0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
```

In [15]:
```
df['Is Fraud?'] = data
```

Typesetting math: 0%

In [16]:
```
df['Is Fraud?']
```

Out[16]:
```
0           0
1           0
2           0
3           0
4           0
           ..
24386895    0
24386896    0
24386897    0
24386898    0
24386899    0
Name: Is Fraud?, Length: 24386900, dtype: int64
```

In [22]:
```
data = []
for i in range(len(df['Is Fraud?'])):
    if df['Is Fraud?'][i] == 0:
        data.append('No')
    else:
        data.append('Yes')
df['Is Fraud??'] = data
```

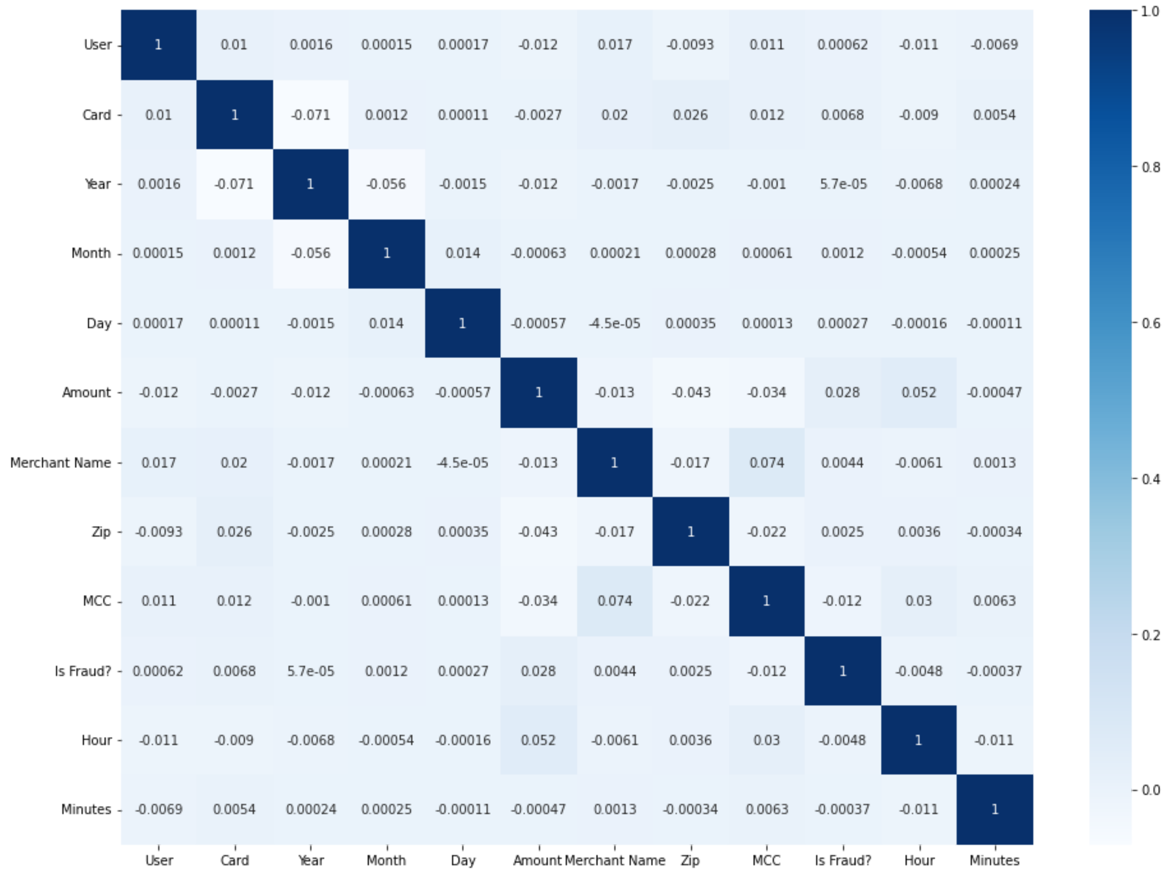
the code above is to convert the target feature from object to numbercal values

Typesetting math: 0%

In [17]:

```
plt.figure(figsize = (16, 12))
sns.heatmap(df.corr(), annot = True, cmap = 'Blues')
```

Out[17]: <AxesSubplot:>



In [18]:

```
corr = df.corr()['Is Fraud?'].abs().sort_values(ascending = False)#abs will t
corr
```

Out[18]:

```
Is Fraud?        1.000000
Amount           0.027681
MCC              0.012281
Card             0.006754
Hour             0.004801
Merchant Name    0.004400
Zip              0.002493
Month            0.001181
User             0.000615
Minutes          0.000374
Day              0.000267
Year             0.000057
Name: Is Fraud?, dtype: float64
```

Spliting the data which will be x and the target which will be y and apply the splitting method from sklearn

Typesetting math: 0%

In [31]:
```
for col in df.columns:
    col_type = df[col].dtype
    if col_type == 'object' or col_type.name == 'category':
        df[col] = df[col].astype('category')
```

In [39]:
```
y = df['Is Fraud?']
X = df.drop(['Is Fraud?', 'Is Fraud??'],axis=1)

#split the traning and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, r
```

the conf_matrix is to method applying the confusion matrix that take the actual data and the predication and gives the results

In [40]:
```
def conf_matrix(actual, predicted):
    cm = confusion_matrix(actual, predicted)
    sns.heatmap(cm, xticklabels=['predicted_negative', 'predicted_positive'],
                ytickslabels=['actual_negative', 'actual_positive'], annot=Tru
                fmt='d', annot_kws={'fontsize':20}, cmap='Blues')

    true_neg, false_pos = cm[0]
    false_neg, true_pos = cm[1]

    accuracy = round((true_pos + true_neg) / (true_pos + true_neg + false_pos
    precision = round((true_pos) / (true_pos + false_pos),3)
    recall = round((true_pos) / (true_pos + false_neg),3)
    f1 = round(2 * (precision * recall) / (precision + recall),3)

    cm_results = [accuracy, precision, recall, f1]
    return cm_results
```

use the LightGBM Classifier model for the dataset

In [41]:
```
model = lgb.LGBMClassifier()
model.fit(X_train, y_train, feature_name='auto', categorical_feature = 'auto'
y_pred=model.predict(X_test)
```

```
C:\anaconda\lib\site-packages\lightgbm\sklearn.py:736: UserWarning: 'verbos
e' argument is deprecated and will be removed in a future release of LightG
BM. Pass 'log_evaluation()' callback via 'callbacks' argument instead.
  _log_warning("'verbose' argument is deprecated and will be removed in a f
uture release of LightGBM. "
```

apply these methods to checks the model and how does it preform in the traning and the testing datasets

Typesetting math: 0%

In [42]:
```
acc = accuracy_score(y_test, y_pred)
print("\nThe Training Score: {} \n".format(model.score(X_train,y_train)*100))
print('The Accuracy is : {}\n\n'.format(acc) )
```

```
The Training Score: 99.88906449223443

The Accuracy is : 0.9988530851822209
```

apply the confusion matrix that defiened above

In [43]:
```
cm_LR = conf_matrix(y_test, y_pred)
```

finally apply the Classification Report

In [44]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   8037857
           1       0.53      0.47      0.50      9820

    accuracy                           1.00   8047677
   macro avg       0.77      0.74      0.75   8047677
weighted avg       1.00      1.00      1.00   8047677
```

From the report we can see that for the minority class, the classifier doesn't perform as well as the majority class in terms of F-1 score.

Typesetting math: 0%

A high F-1 score indicates a good balance of high precision as well as high recall. In the credit card transactions scenario, to ensure customers use the credit card and are satisfied with the service, it is necessary to detect fraud (have lower false negatives) but also prevent unecessary blocking since if the card is always blocked, then users may be frustated with the service (have lower false positives). Thus F-1 score is a good metric choice.

Additionally, since fraudulent transactions are rare, it would also be interesting to look at other metrics such as Precision-Recall curve and area under this curve along with Matthews Correlation Coefficient.

Conclusion

we train model that got incredible accuracy which is 99.3% and save the model as SAV extrntion for future use, working with huge number of data is so diffcult and hard, you should have a very strong device that can run it and the one that use here is one of the highest dataset that i have ever use approximalty 24.4M records.

Computer Security - CS 3801

Class 2108

Created by:

- Abdulaziz Alkhonain - 439051169

- Hamad Alhajlan - 439051043

- Moahmmed Banajh - 438050064

Typesetting math: 0%