Table of Contents

QUESTION 1 & 2 - LOADING & PLOTTING

1) Load the transactions.txt
2) Describe the structure of the Data
3) Additional basic summary for each field
4) Plot a histogram of the transactionAmount column
5) Provide Insights

1) Loading the dataset

```
import json
#loading our dataset into 'data'
with open('/kaggle/input/transactions/transactions.txt') as f:
    data = [json.loads(line) for line in f]

#Converting our data to a pandas df called 'transactions'
import pandas as pd
transactions = pd.DataFrame(data)
```

We have saved our dataset finally in the transactions data frame and will be using this to do our analysis

2) Describing the basic structure of the dataset

```
print(f"In our Dataset the number of rows is {transactions.shape[0]}
```

```
In our Dataset the number of rows is 786363 and the number of columns is 29
```

As seen we can see that we initially have *786363 Rows & 29 Columns*

3) Additional basic summary for different fields

We can use .info() to see all our columns and the null count as well as the datatype

```
RangeIndex: 786363 entries, 0 to 786362
Data columns (total 29 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   accountNumber             786363 non-null  object
 1   customerId                786363 non-null  object
 2   creditLimit               786363 non-null  float64
 3   availableMoney            786363 non-null  float64
 4   transactionDateTime       786363 non-null  datetime
 5   transactionAmount         786363 non-null  float64
 6   merchantName              786363 non-null  object
 7   acqCountry                786363 non-null  object
 8   merchantCountryCode       786363 non-null  object
 9   posEntryMode              786363 non-null  object
 10  posConditionCode          786363 non-null  object
 11  merchantCategoryCode      786363 non-null  object
 12  currentExpDate            786363 non-null  object
 13  accountOpenDate           786363 non-null  object
 14  dateOfLastAddressChange   786363 non-null  object
 15  cardCVV                   786363 non-null  object
 16  enteredCVV                786363 non-null  object
 17  cardLast4Digits           786363 non-null  object
 18  transactionType           786363 non-null  object
```

- .info() does not help with identifying whitespaces or other non standard missing values
- Some of the columns are numeric but are of datatype - object. We need to convert this to float
- We will also change the boolean dtype to integers for easy interpretation

```
 19  echoBuffer                786363 non-null  object
 20  currentBalance            786363 non-null  float64
 21  merchantCity              786363 non-null  object
 22  merchantState             786363 non-null  object
 23  merchantZip               786363 non-null  object
 24  cardPresent               786363 non-null  bool
 25  posOnPremises             786363 non-null  object
 26  recurringAuthInd          786363 non-null  object
 27  expirationDateKeyInMatch  786363 non-null  bool
 28  isFraud                   786363 non-null  bool
dtypes: bool(3), datetime64[ns](1), float64(4), object(21)
```

3) Additional basic summary for different fields

Checking for NA vlaues

```
#we use the .replace fuction to find more than just single blank
transactions.replace(r'^\s*$', pd.NA, regex=True, inplace=True)

#We count the number of missing values in each column
missing_count = transactions.isnull().sum()

#We then save all columns and the count where we have missing va
missing_columns = missing_count[missing_count > 0]

#Printing out the columns so that we can deal with them
print("These are the columns which have missing values",missing
```

```
These are the columns which have missing values acqCountry
merchantCountryCode       724
posEntryMode             4054
posConditionCode          409
transactionType           698
echoBuffer             786363
merchantCity           786363
merchantState          786363
merchantZip            786363
posOnPremises          786363
recurringAuthInd       786363
```

From this image we notice:

- A few columns are completley empty so we go ahead and remove them.
- When we go through our data we notice *merchantCountryCode* and *acqCountry* are 99.99% of the time the same value, so we make an assumption when either vaue is missing and the other column has a vlaue we fill the na value with that.
- In *transactionType* we see that when the *transactionAmount* is 0 then 90% of the times the *transactionType* is ADDRESS_VERIFICATION, so we full na values with adress verification in the row where transaction amount is 0 and if not we fill the na value with the mode
- Rest we fill with the mode of that column

3) Additional basic summary for different fields

From our dataset we can use .describe() to get Min, Max, Mean and other numeric insights from a few columns where it makes sense:

	creditLimit	availableMoney	transactionAmount	currentBalance
count	786363.000000	786363.000000	786363.000000	786363.000000
mean	10759.464459	6250.725369	136.985791	4508.739089
std	11636.174890	8880.783989	147.725569	6457.442068
min	250.000000	-1005.630000	0.000000	0.000000
25%	5000.000000	1077.420000	33.650000	689.910000
50%	7500.000000	3184.860000	87.900000	2451.760000
75%	15000.000000	7500.000000	191.480000	5291.095000
max	50000.000000	50000.000000	2011.540000	47498.810000

This table shows values based on all the rows in the dataset

	creditLimit	availableMoney	transactionAmount	currentBalance
count	5000.000000	5000.000000	5000.000000	5000.000000
mean	10149.150000	7700.381550	99.355932	2448.768450
std	10816.765564	9958.518614	129.929476	3993.820145
min	250.000000	-375.230000	0.000000	0.000000
25%	5000.000000	1383.407500	16.425000	197.777500
50%	7500.000000	4598.520000	50.965000	974.820000
75%	15000.000000	9937.012500	130.555000	3193.157500
max	50000.000000	50000.000000	1091.130000	45963.380000

This table shows values based on the latest transaction of the customer

3) Additional basic summary for different fields

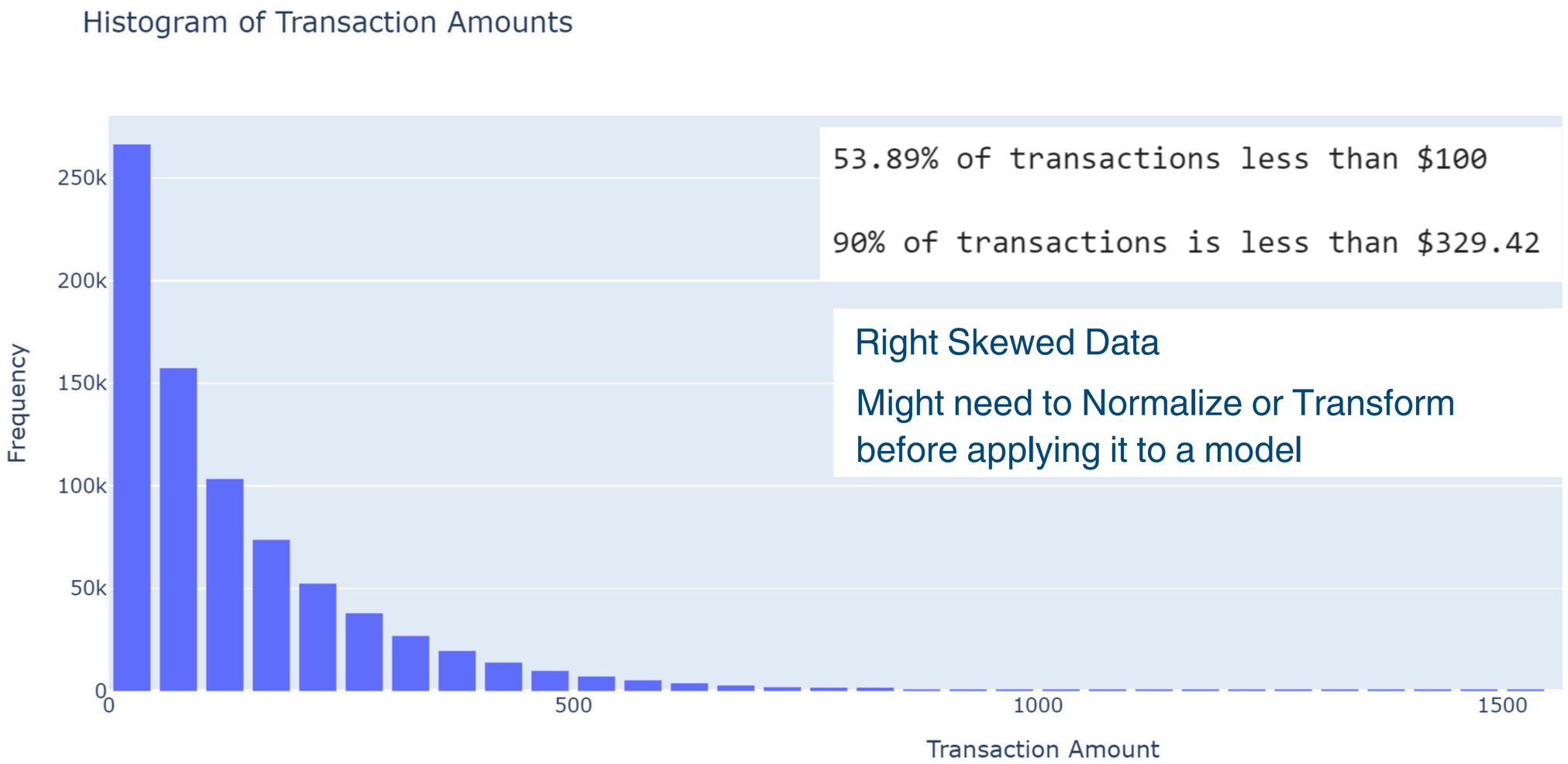
- Now we can look into unique vales and the mode for all our columns
- Number of unique values will helps identifying categorical vs numerical data, choosing our encoding and more

	Unique Values	Mode
accountNumber	5000	380680241.0
customerId	5000	380680241.0
creditLimit	10	5000.0
availableMoney	521916	250.0
transactionDateTime	776637	2016-05-28 14:24:41
transactionAmount	66038	0.0
merchantName	2490	Uber
acqCountry	4	US
merchantCountryCode	4	US
posEntryMode	5	5.0
posConditionCode	3	1.0

	Unique Values	Mode
merchantCategoryCode	19	online_retail
currentExpDate	165	03/2029
accountOpenDate	1820	2014-06-21
dateOfLastAddressChange	2184	2016-03-15
cardCVV	899	869.0
enteredCVV	976	869.0
cardLast4Digits	5245	593.0
transactionType	3	PURCHASE
currentBalance	487318	0.0
cardPresent	2	0.0
expirationDateKeyInMatch	2	0.0
isFraud	2	0.0

4) Plot a histogram of the transactionAmount column

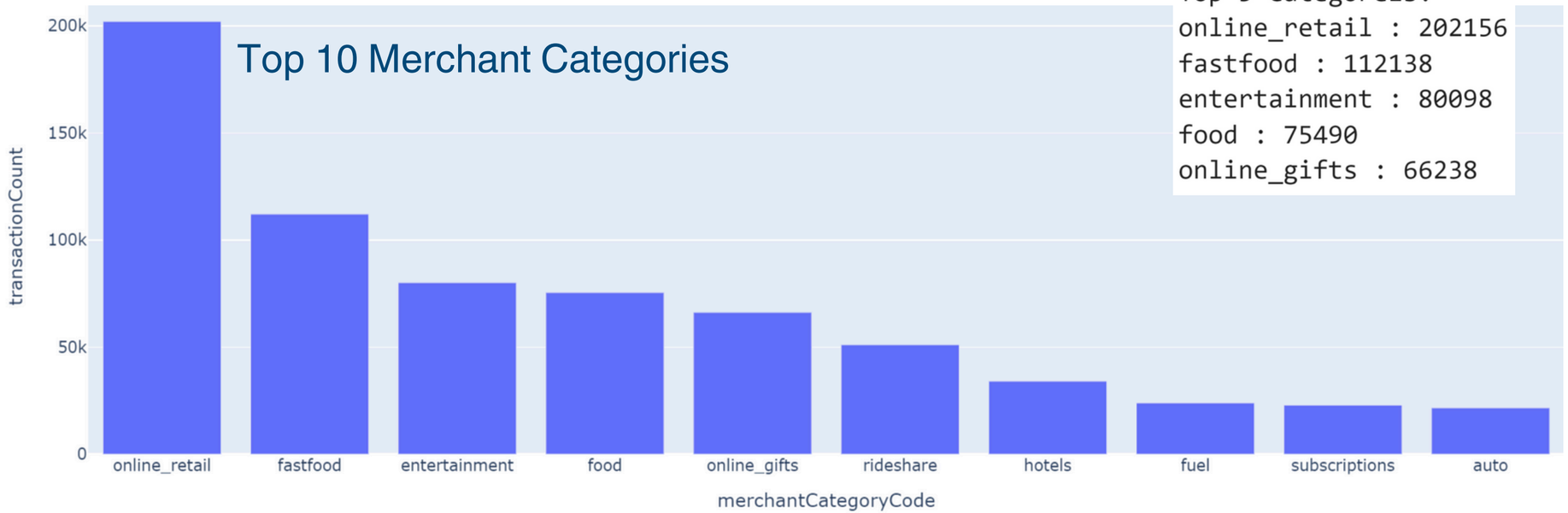
- Use plotly over matplotlib to give interactive plots
- Each bin is a 50$ increment and we can see that most transactions are less than 50$ and keeps decreasing with the next bin



5) Exploratory Data Analysis (EDA)

Top 5 Merchants:
Uber : 25613
Lyft : 25523
oldnavy.com : 16992
staples.com : 16980
alibaba.com : 16959

Top 5 Categoreis:
online_retail : 202156
fastfood : 112138
entertainment : 80098
food : 75490
online_gifts : 66238



5) Exploratory Data Analysis (EDA)

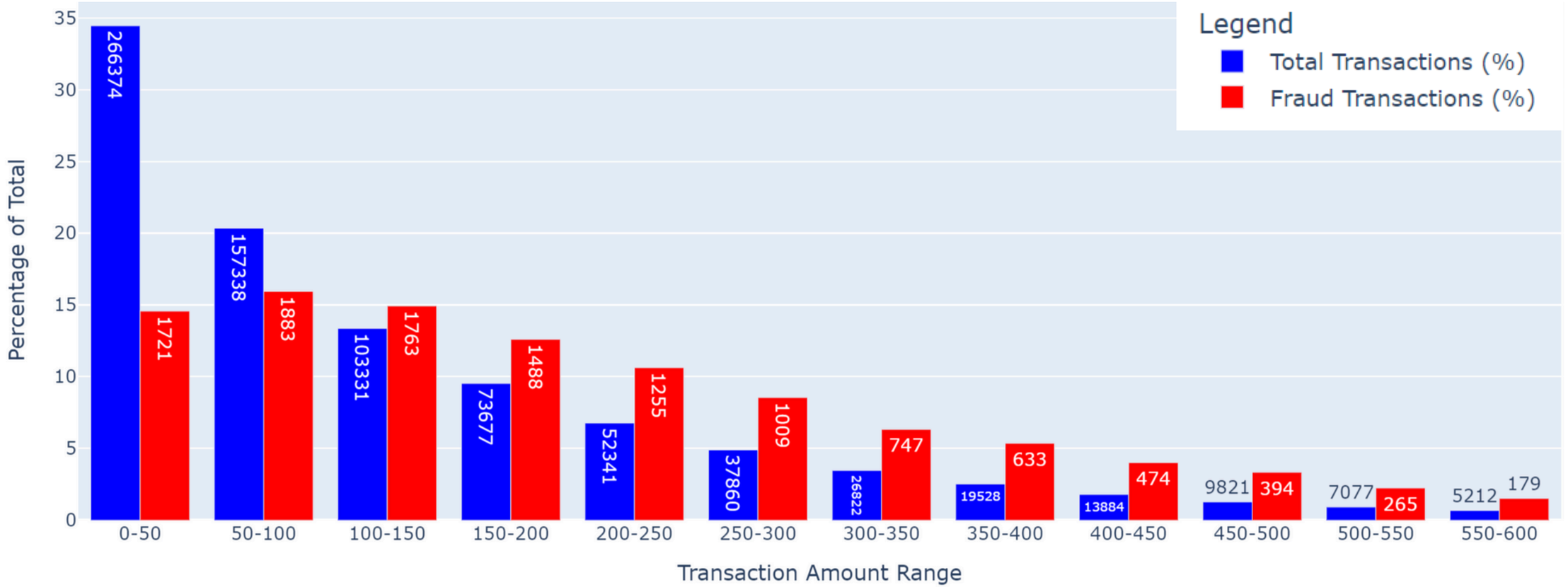
- Interesting to know people **spend the most in October**
- Normally one would have assumed Summer months(due to travel), Thanksgiving and Christmas time would have the higher expendature total

```
Monthly Transactions
2016-01 : 61572 transactions
2016-02 : 59042 transactions
2016-03 : 63927 transactions
2016-04 : 62633 transactions
2016-05 : 65689 transactions
2016-06 : 64735 transactions
2016-07 : 67159 transactions
2016-08 : 68129 transactions
2016-09 : 66777 transactions
2016-10 : 69627 transactions
2016-11 : 68097 transactions
2016-12 : 68976 transactions
```

```
Monthly Expenditure
2016-01 : 8878330.98$
2016-02 : 8440251.77$
2016-03 : 9090817.48$
2016-04 : 8778160.68$
2016-05 : 9129931.86$
2016-06 : 8917017.85$
2016-07 : 9126021.42$
2016-08 : 9164121.72$
2016-09 : 8921727.17$
2016-10 : 9298821.84$
2016-11 : 9012658.87$
2016-12 : 8962695.89$
```

5) Exploratory Data Analysis (EDA)

Proportion of Total Transactions vs Fraud Transactions in $50 Increments from 0-600



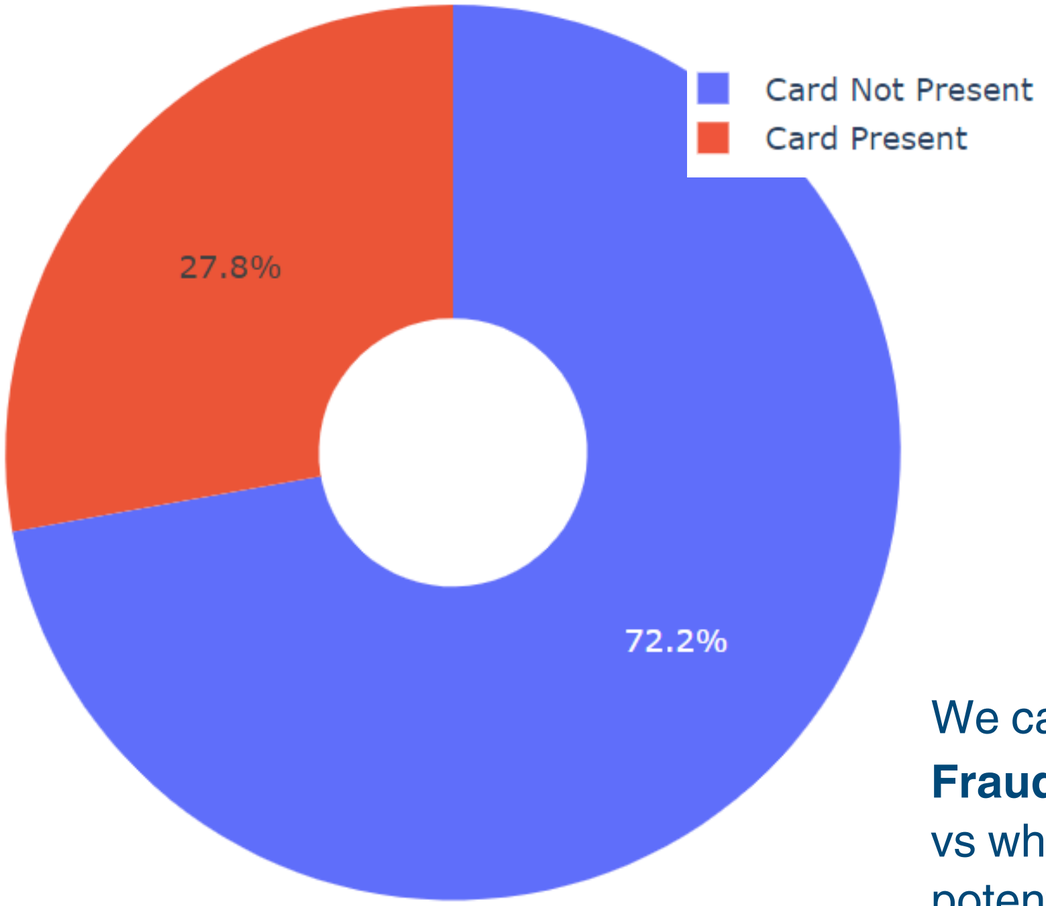
5) Exploratory Data Analysis (EDA)

```
Top 5 Ranges with Highest Fraud Percentages:
Range: 1600-1700
Total Transactions: 3
Total Fraud: 1
Fraud Percentage: 33.33%

Range: 1400-1500
Total Transactions: 33
Total Fraud: 4
Fraud Percentage: 12.12%

Range: 1500-1600
Total Transactions: 13
Total Fraud: 1
Fraud Percentage: 7.69%

Range: 1000-1100
Total Transactions: 457
Total Fraud: 32
Fraud Percentage: 7.00%

Range: 800-900
Total Transactions: 1625
Total Fraud: 93
Fraud Percentage: 5.72%
```

- We notice that the chances that a transaction is a fraudulant transaction is higher where the TransactionAmount is greater than 800$
- We notice that as the amount increases the chances of it being a fraud also increases
- It's important to consider both the fraud percentage and the total volume of transactions when assessing risk. High fraud percentages with small sample sizes might be misleading, whereas transaction ranges with a larger number of total transactions and fraud cases (Eg:800-900) present more tangible risks

5) Exploratory Data Analysis (EDA)

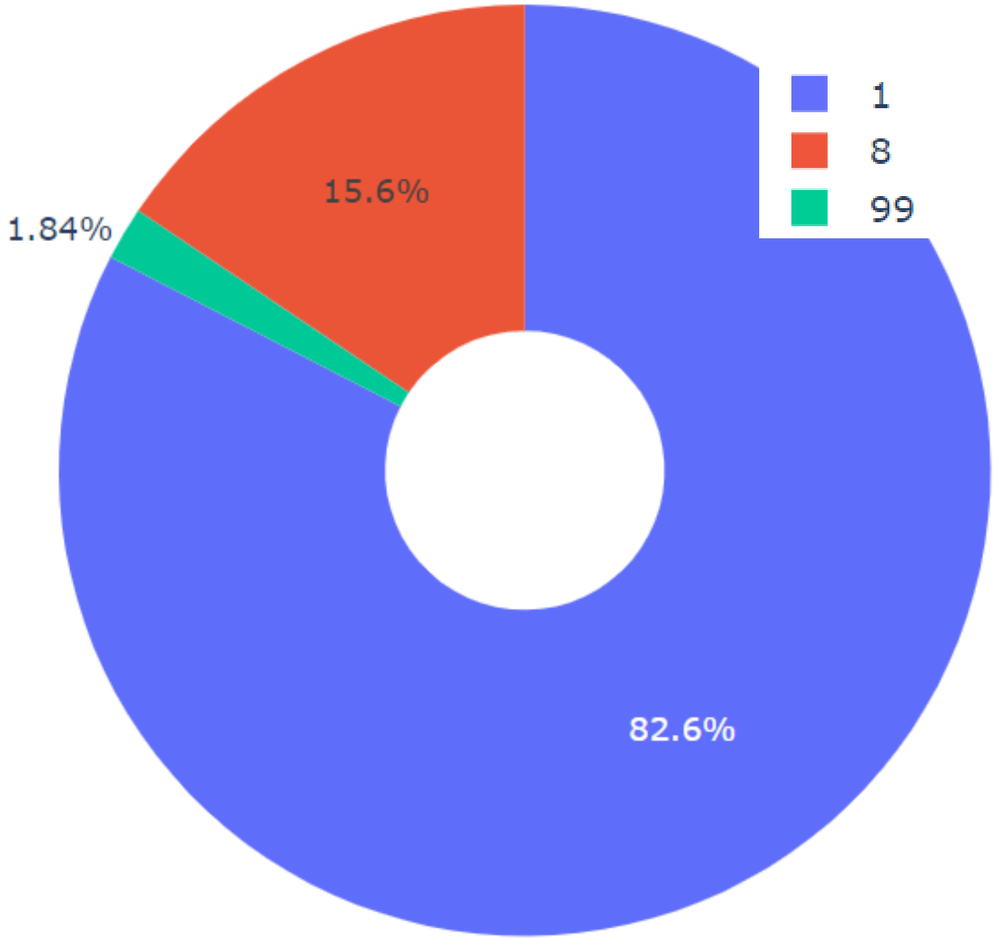
Fraud Transactions: Card Present vs Card Not Present



We can see there is a **huge spike in Fraud when the card is NOT present** vs when the card is present, it can potentially be an important feature

5) Exploratory Data Analysis (EDA)

Fraud Transactions: Based on type of POS Condition Code

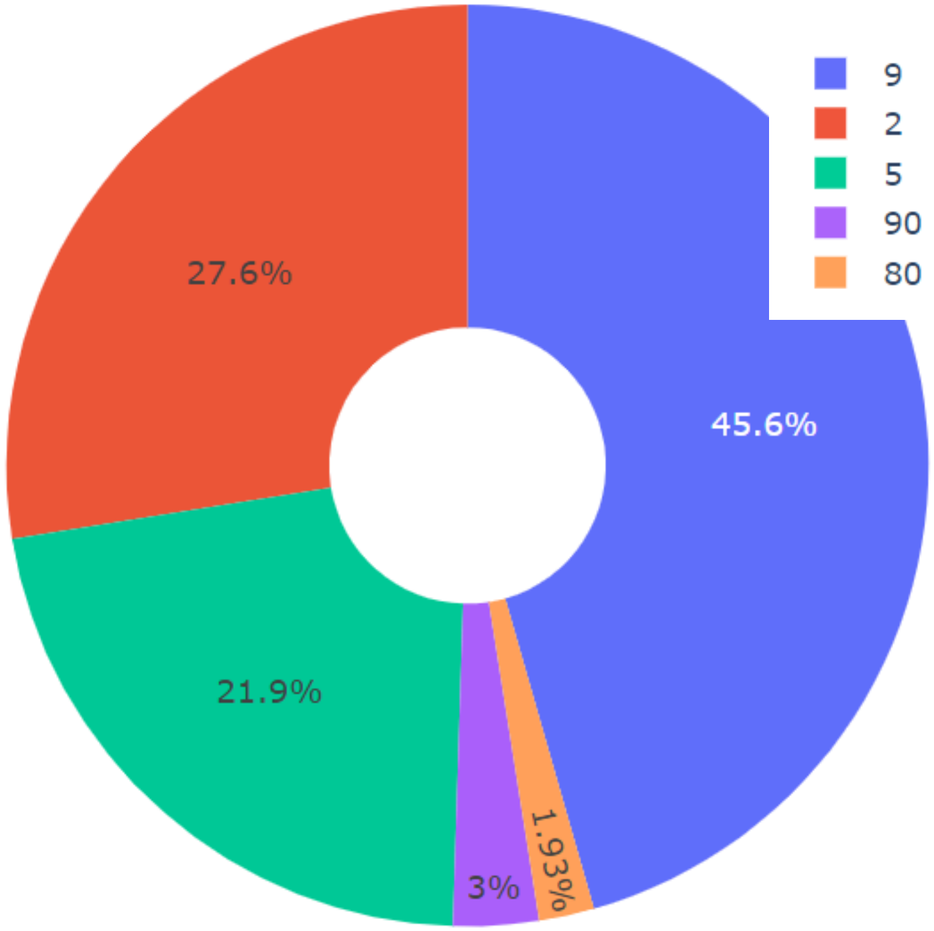


We can see there is a huge spike in Fraud when the POS Condition Code is 1.

Almost 83% of fraud cases have a POS Condition of 1

5) Exploratory Data Analysis (EDA)

Fraud Transactions: Based on type of POS Entry



We can see close to half the fraud cases have a POS entry value of 9

Very few cases are 80 or 90

5) Exploratory Data Analysis (EDA)

```
Top 5 Accounts with the most Transactions:
380680241 : 32850
882815134 : 13189
570884863 : 10867
246251253 : 10172
369308035 : 7229
```

We noticed that the account ending in 0241 has close to 3x the number of transactions compared to the second highest account.

```
Top 5 Accounts with the most Expenditure:
380680241 : 4765004.34$
882815134 : 1917310.66$
570884863 : 1569178.94$
246251253 : 1476262.36$
369308035 : 1055003.36$
```

But at the same time we see that 0241 has also spent the most amount of money.

```
Top 5 Total Fraud Amounts with Account Number:
380680241 : $200599.19
700725639 : $49950.15
782081187 : $47907.74
472288969 : $44764.62
246251253 : $44250.04
```

We see that 0241 has a whopping 200599$ in fraud amount, this is more than 4x the second highest account. Hence why they have the most expenditure

```
Top 5 Accounts with the most Fraud:
380680241 : 783
782081187 : 307
246251253 : 278
700725639 : 272
472288969 : 266
```

We notice the number of Fraud amounts goes hand in hand with the total fraud amount with account 0241 having 783 fraud cases!

QUESTION 3 - DATA WRANGLING - DUPLICATE TRANSACTIONS

1) Programmatically identify Reversed and Multi-Swipe transactions
2) Total number and total dollar amount for Reversed and Multi-Swipe
3) Interesting findings

1) Programmatically identify Reversed and Multi-Swipe transactions

First let us deal with Reversal transactions
These are the rules I have set based on my intuition to figure out which transactions are Reversed transactions

- **Same Account, Merchant and Amount:** The reversal transaction and the ‘normal’ transaction must have the same account number, merchant name and transaction amount.

- **Exactly Two Transactions:** There must be exactly 2 transactions in this combination of account number, merchant name and transaction amount. This ensures that we aren’t taking multiswipes into account.

- **At least one Transaction is Marked as 'REVERSAL':** One of the two transactions must have its transactionType marked as 'REVERSAL'.

- **Time Difference Between Transactions:** The time difference between the two transactions must be greater than 10 minutes (time_diff > 10 minutes). If its less than 10 then it might be a mistake so multi-swipe.

1) Programmatically identify Reversed and Multi-Swipe transactions

This code follows the same rules we have discussed earlier and we save the reversal transactions in the *reversal_df*

```
#calculate time differences within each group of (accountNumber, merchantName, transactionAmount)
transactions_copy['time_diff'] = transactions_copy.groupby(['accountNumber', 'merchantName', 'transactionAmount'])['transactionDateTime'].diff()

#make a new column to find rows where the transactionType is 'REVERSAL'
transactions_copy['is_reversal'] = transactions_copy['transactionType'] == 'REVERSAL'

#find duplicates with exactly 2 transactions
group_sizes = transactions_copy.groupby(['accountNumber', 'merchantName', 'transactionAmount']).size()
two_transaction_groups = group_sizes[group_sizes == 2].index
transactions_copy['in_two_transaction_group'] = transactions_copy.set_index(['accountNumber', 'merchantName', 'transactionAmount']).index.isin(two_transa

#Now let us detect Reversal transactions (two identical transactions, one marked as 'REVERSAL', and >10 min apart)
#Filter for groups with exactly 2 transactions and check for reversals, one is 'REVERSAL' and time diff greater than 10 mins
reversal_mask = (transactions_copy['in_two_transaction_group'] & transactions_copy['is_reversal'] & (transactions_copy['time_diff'] > pd.Timedelta(minute

#lwts make a reversal df with both transaction times
reversal_df = transactions_copy[transactions_copy['in_two_transaction_group']].groupby(['accountNumber', 'merchantName', 'transactionAmount']).agg(transa

#filter out only the groups where one transaction is 'REVERSAL' and the time_diff > 10 minutes
reversal_df = reversal_df[reversal_mask]
```

1) Programmatically identify Reversed and Multi-Swipe transactions

This is our reversal df and you can see the times of the first and last transaction along with other values

```
Reversal Transactions:
       accountNumber              merchantName  transactionAmount  \
142        106905774              discount.com              47.40
268        114896048                amazon.com               6.93
285        116227988                   gap.com              15.33
296        116866974  South Steakhouse #481929              65.23
546        128520413                  ebay.com               0.00
...              ...                       ...                ...
27328      982040165              staples.com             178.19
27446      987967388          Washington Times             499.13
27469      990304907         Franks Restaurant               0.00
27534      993824572                      Uber              98.83
27686      999257059          Wall Street News              47.58

      Transaction Type          first_time           last_time  count
142           Reversal 2016-04-25 19:19:35 2016-04-25 19:21:14      2
268           Reversal 2016-12-05 16:58:37 2016-12-07 05:20:36      2
285           Reversal 2016-09-19 18:05:39 2016-10-10 18:19:41      2
296           Reversal 2016-05-31 05:39:36 2016-06-27 03:06:54      2
546           Reversal 2016-09-13 09:48:50 2016-10-15 15:57:10      2
...                ...                 ...                 ...    ...
27328         Reversal 2016-03-27 15:39:15 2016-03-31 13:26:00      2
27446         Reversal 2016-10-08 05:47:33 2016-11-02 00:17:21      2
27469         Reversal 2016-04-02 07:58:18 2016-09-22 01:30:39      2
27534         Reversal 2016-11-15 05:34:11 2016-11-26 06:04:06      2
27686         Reversal 2016-01-14 16:48:41 2016-01-14 16:51:22      2
```

1) Programmatically identify Reversed and Multi-Swipe transactions

Now let us deal with Multi-Swipe transactions
These are the rules I have set to figure out which transactions are Multi- Swipe transactions

- **Same Account, Merchant and Amount:** The multiswipes and the ‘normal’ transaction must have the same account number, merchant name and transaction amount

- **Time Difference of 10 Minutes or Less:** The time difference between two or more transactions within the same account number, merchant name and transaction amount must be less than or equal to 10 minutes.

- **Not in the Two-Transaction Group (No Reversal):** This is done because the focus here is on multiple transactions happening in quick succession, rather than identifying reversals.

- **Filter for Groups with More Than One Transaction:** The final step filters for groups that have more than one transaction *(multi_swipe_summary['count'] > 1)*, as we are only interested in cases where multiple transactions occurred in a short span, indicating a possible multi-swipe event.

I’ve ensured that the 10 minute interval calculation is continuous, meaning if there are 10 multi-swipe transactions within 2 minutes of each other, we don’t just capture the first 5 within a 10-minute window. Instead, every consecutive transaction with a time difference of less than 10 minutes is included, so no transactions are missed.

1) Programmatically identify Reversed and Multi-Swipe transactions

This code follows the same rules we have discussed earlier and we save the reversal transactions in the *multi_swipe_df*

```
#lets assume a 10min window for continous transactions
#we need to also ensure we do not have overlaps between the multi-swipe and reversals
multi_swipe_mask = (transactions_copy['time_diff'] <= pd.Timedelta(minutes=10)) & (~transactions_copy['in_two_transaction_group'])

#create the multi-swipe summary DataFrame
multi_swipe_summary = transactions_copy[multi_swipe_mask].groupby(['accountNumber', 'merchantName', 'transactionAmount']).agg(
    count=('transactionAmount', 'size'),first_time=('transactionDateTime', 'min'),last_time=('transactionDateTime', 'max')).reset_index()

#filter for multi-swipe groups more than 1 transaction
multi_swipe_df = multi_swipe_summary[multi_swipe_summary['count'] > 1]
```

1) Programmatically identify Reversed and Multi-Swipe transactions

This is our multi swipe df and you can see the times of the first and last transaction along with other values

Multi-Swipe Transactions:

	accountNumber	merchantName	transactionAmount	\
0	100737756	Franks Deli	693.50	
1	101380713	amazon.com	33.74	
2	101876201	alibaba.com	214.45	
3	101876201	cheapfast.com	191.05	
4	102980467	Washington Post	333.92	
...	...	...	...	
1069	987594719	discount.com	15.12	
1070	988393082	Walgreens #646490	1.74	
1071	992532997	Renaissance Hotel #973058	191.52	
1072	993237077	In-N-Out #710537	37.53	
1073	998801944	Quizno's #784044	176.61	

	Transaction Type	first_time	last_time	count
0	Multi-Swipe	2016-01-18 01:55:28	2016-01-18 01:58:26	2
1	Multi-Swipe	2016-07-23 06:56:15	2016-07-23 06:57:42	2
2	Multi-Swipe	2016-11-29 14:02:26	2016-11-29 14:04:03	2
3	Multi-Swipe	2016-02-11 09:57:40	2016-02-11 09:59:28	2
4	Multi-Swipe	2016-02-27 06:27:14	2016-02-27 06:27:52	2
...	...	...	...	...
1069	Multi-Swipe	2016-02-01 18:09:08	2016-02-01 18:12:05	2
1070	Multi-Swipe	2016-02-19 00:10:36	2016-02-19 00:10:42	2
1071	Multi-Swipe	2016-01-09 10:51:02	2016-01-09 10:53:50	2
1072	Multi-Swipe	2016-01-27 10:24:55	2016-01-27 10:25:03	2
1073	Multi-Swipe	2016-07-11 03:15:49	2016-07-11 03:16:35	2

	count
2	774
3	11
5	1
4	1

We can see the distribution of our multiswipe df where the left column shows the number of swipes and the right column shows the number of times the multiple swipes happened

2) Total number and total dollar amount for Reversed and Multi-Swipe

We just need to take the length of the df to find the count since we have a count of 2 per row and the length of the dataframe is the number of total reveral transactions

We then just add the TransactionAmount each row *without* multiplying it woth the count to get the reversal transaction amount. We exclude the ‘proper’ transactions

```
reversal_total = len(reversal_df)
reversal_total_amount = reversal_df['transactionAmount'].sum()
```

```
The number of reversal transactions is:  429
The total amount of money deemed as reversal is: $56783.95
```

Similarly, since we have a count column, we need to calculate all the multi-swipe transactions while excluding the first transaction. We can achieve this by either creating a new column with count - 1 or simply subtracting the DataFrame's length, as each group will have one less multi-swipe transaction than the total count.

We now calculate the total amount by taking the transactionAmount and multiplying it by count-1 to leave out the ‘proper’ transaction

```
multiswipe_total = multi_swipe_df['count'].sum() - len(multi_swipe_df)
multi_swipe_df['totalAmount'] = multi_swipe_df.apply(lambda row: row['transactionAmount']*(row['count']-1),axis=1)
```

```
The number of multi swipe transactions is:  803
The total amount of money deemed as multi swipe is: $114632.90
```

3) Interesting findings

Top 5 Accounts with the Most Multi-Swipe Transactions:

	accountNumber	multi_swipe_count	multi_swipe_total_amount
208	380680241	48.0	5699.81
584	882815134	13.0	1156.08
119	246251253	13.0	1236.22
231	410523603	11.0	733.25
342	570884863	10.0	1069.42

Top 5 Accounts with the Most Reversal Transactions:

	accountNumber	reversal_count	reversal_total_amount
208	380680241	38.0	5012.56
584	882815134	10.0	1879.94
119	246251253	7.0	714.95
231	410523603	5.0	525.06
342	570884863	8.0	876.75

Total Unique Accounts in Transactions: 5000
Accounts involved in Multi-Swipe Transactions: 499 (9.98%)
Accounts involved in Reversal Transactions: 298 (5.96%)

Percentage of Multi-Swipe Transaction Amount: 0.11%
Percentage of Reversal Transaction Amount: 0.05%

- Top Account Activity: Account 0241 has the most multi-swipe (48) and reversal (38) transactions, indicating potential irregular behavior
- Involvement: 9.98% of accounts are involved in multi-swipes, and 5.96% in reversals
- Monetary Impact: Multi-swipe transactions account for 0.11% of total transaction value, while reversals contribute 0.05%
- Key Insight: Accounts like 380680241 and 882815134 are worth further investigation due to their high activity in both categories.

QUESTION 4 - MODEL BUILDING

1) Please build a predictive model to determine whether a given transaction will be fraudulent or not.

2) Provide an estimate of performance using an appropriate sample

3) Please explain your methodology(modeling algorithm/method used and why, what features/data you found useful)

Before we get into model building lets manipulate our data

New columns made:

- isCorrectCVV - gave 1 if ‘cardCVV’ and ‘enteredCVV’ is the same; else 0
- timeOfDay - gave 1,2,3,4 based on whether the transaction was in the morning, afternoon, evening or night
- isSameCountry - gave 1 if ‘acqCountry’ and ‘MerchantCountryCode’ was the same; else 0
- monthsToExpiry - made a column to figure out how many months were left to expiry

Dropped columns:

- WIll not affect the model: 'accountNumber', 'customerId', 'cardLast4Digits', ‘timeOfDay’,'transactionDateTime'
- Made Simpler Columns: 'cardCVV', 'enteredCVV', 'acqCountry','currentExpDate', 'merchantCountryCode'

	Unique Values	Mode
creditLimit	10	5000.0
availableMoney	521916	250.0
transactionAmount	66038	0.0
merchantName	2490	Uber
posEntryMode	5	5.0
posConditionCode	3	1.0
merchantCategoryCode	19	online_retail
accountOpenDate	1820	2014-06-21

	Unique Values	Mode
dateOfLastAddressChange	2184	2016-03-15
transactionType	3	PURCHASE
currentBalance	487318	0.0
cardPresent	2	0
expirationDateKeyInMatch	2	0
isFraud	2	0
isCorrectCVV	2	1
isSameCountry	2	1
monthsToExpiry	176	135

Before we get into model building lets manipulate our data

Earlier, we saw our TransactionAmount was right skewed and might affect our model

```
Skewness of features:
currentBalance       3.362137
transactionAmount    2.092246
availableMoney       2.999324
creditLimit          2.280312
```

Based on this we now apply Log Transformation to reduce skweness,
this helps makes the distribution more symmetric

For the rest of the data we either go with

- **Label Encoding:** on ‘merchantName’ since it contains many unique values
- **One Hot Encoding:** on 'posEntryMode', 'posConditionCode', 'merchantCategoryCode', 'transactionType'. Helps to avoid numeric bias on the categorical data

Before we get into model building lets manipulate our data

We can now sample our dataset, since running models with 780K rows will take time and might run into memory issues.
We will sample our data into a sample size of 100K with a 10% ratio having isFraud ==1

```
print(sampled_transactions['isFraud'].value_counts())
```

```
isFraud
0    90000
1    10000
```

Our new dataset has 90K non fraud values and 10K fraud values

We now need to deal with this imbalance in data.

- There are several ways we can go about it, but since we do have different types of data SMOTENC ie SMOTE NUMERIC CATEGORICAL is the technique I chose to deal with our dataset
- By using SMOTENC we avoid the problem of having incorrect synthetic categorical data, which can otherwise reduce the performance of your model.

```
Class distribution after SMOTENC:
isFraud
0    62971
1    31485
```

We have a sampling strategy of 0.5 as anything more leads to over sampling and lesser leads to under sampling affecting the recall and precision respectivley

```
smote_nc = SMOTENC(categorical_features=categorical_features_indices, random_state=42, sampling_strategy=0.5)
```

Before we get into model building lets manipulate our data

Lets get into some of the feature extraction techniques:

1. Correlation: *log_transactionAmount, posEntryMode_9.0,* and *merchantCategoryCode_online_retail* have the highest positive correlations with fraud detection.
Correlation is useful here to identify features like *log_transactionAmount* that have a strong linear relationship with the target (fraud). However, it might not capture non-linear relationships, and some lower correlation features could still be important for prediction.

2. Chi-Squared (Chi-sq) Test: *posEntryMode_5.0, posEntryMode_9.0, merchantCategoryCode_online_retail,* and *log_transactionAmount* have shown strong associations with the target, as indicated by high Chi-sq values and significant p-values.
This method highlights categorical features that are strongly linked to fraud

3. XGBoost and RandomForest-based Feature Extraction: From the feature importance scores in your Random Forest and XGBoost models, key features like *log_transactionAmount, merchantName* and *posEntryMode_9.0* rank highly. These models consider both linear and non-linear relationships and features like *merchantName* and l*og_currentBalance* have significant contributions according to these models. XGBoost's ability to handle non-linear relationships further highlights the importance of these features.

Combining these techniques, features such as *log_transactionAmount, posEntryMode_9.0,* and *merchantCategoryCode_online_retail consistently* appear as important across methods.

Before we get into model building lets manipulate our data

These are a few features and the scores we got

Feature Evaluation Table

	Feature	Correlation	Chi-squared	Random Forest Impor	XGBoost Importance
1	log_transactionAmount	0.277231	398.2	0.1743	0.0254
2	posEntryMode_9.0	0.17407	1891.45	0.0197	0.0063
3	merchantCategoryCode_online_retail	0.172499	1981.14	0.0137	0.0375
4	log_creditLimit	0.011788	1.07	0.1079	0.0644
5	posConditionCode_99.0	-0.030117	87.91	0.0018	0.0038
6	merchantCategoryCode_gym	-0.032753	104.5	0.0004	0.0088
7	merchantCategoryCode_furniture	-0.04824	225.54	0.0043	0.0291

Correlation: Measures the linear relationship with fraud. Higher absolute values (+,-) show stronger relationships.

Chi-Square: Tests feature dependency on fraud. Higher values indicate stronger importance for fraud prediction.

Random Forest Importance: Shows feature contribution to the model's predictions. Higher values mean the feature is more impactful

XGBoost Importance: Measures how much features reduce model error. Higher values mean more influence.

Before we get into model building lets manipulate our data

The most important features in our model based on feature selection techniques (correlation, Chi-squared test, Random Forest, and XGBoost) were:

- log_transactionAmount – Highly correlated with fraud and consistently ranked high in importance across models.
- posEntryMode_9.0 – Captured significant patterns related to fraud detection.
- merchantCategoryCode_online_retail – Showed strong relevance to fraudulent transactions.
- log_currentBalance – Influential in determining fraud likelihood.
- posConditionCode_8.0 – Important for detecting abnormal transaction conditions.
- merchantCategoryCode_rideshare – Relevant to distinguishing between legitimate and fraudulent transactions.

1) Please build a predictive model to determine whether a given transaction will be fraudulent or not.

The 4 models we are focusing on in this assignment is

- Random Forest: It is robust to overfitting (especially since we use SMOTENC). It works well with imbalanced data using class weights and can handle a variety of feature types.

- XGBoost: Known for its strong predictive performance. XGBoost is effective with imbalanced data due to built-in regularization and tree boosting which helps in capturing complex patterns in fraud detection.

- LightGBM: It is faster and more memory-efficient especially for large datasets. It can handle imbalanced datasets making it a good choice for fraud detection.

- CatBoost: This model is designed to handle categorical features natively and performs well with imbalanced datasets. It's especially useful when working with both numeric and categorical features which we have in our dataset

1) Please build a predictive model to determine whether a given transaction will be fraudulent or not.

After running the 4 models we get these results:

```
Random Forest Metrics
Accuracy: 0.7522
F1 Score: 0.2863
Precision: 0.2002
Recall: 0.5019
ROC AUC: 0.7259
```

```
LightGBM Metrics(Regu
Accuracy: 0.8099
F1 Score: 0.3357
Precision: 0.2567
Recall: 0.4850
ROC AUC: 0.7590
```

```
XGBoost Metrics(Regu
Accuracy: 0.7025
F1 Score: 0.2851
Precision: 0.1870
Recall: 0.5988
ROC AUC: 0.7341
```

```
CatBoost Metrics(Reg
Accuracy: 0.7022
F1 Score: 0.2745
Precision: 0.1809
Recall: 0.5688
ROC AUC: 0.7241
```

LightGBM performs best in terms of accuracy, F1 score and ROC. Its precision and recall are relatively balanced making it the most reliable model for this dataset.

Random Forest has good accuracy but low precision which means it identifies fewer true fraud cases. Its recall is higher, meaning it's better at detecting fraud but with many false positives.

XGBoost shows a good balance between precision and recall but its overall F1 score and accuracy are lower than LightGBM, suggesting that it may not be the optimal choice for this dataset.

CatBoost has similar performance to XGBoost, but its precision and recall are slightly lower, with a comparable ROC AUC

2) Provide an estimate of performance using an appropriate sample

After running cros validation on the 4 models (for ROC) :

```
Evaluating Random Forest with Cross-Validation...
Random Forest Cross-Validated ROC AUC: 0.9146 ± 0.0022

Evaluating XGBoost with Cross-Validation...
XGBoost Cross-Validated ROC AUC: 0.9245 ± 0.0024

Evaluating LightGBM with Cross-Validation...
LightGBM Cross-Validated ROC AUC: 0.9267 ± 0.0026

Evaluating CatBoost with Cross-Validation...
CatBoost Cross-Validated ROC AUC: 0.9252 ± 0.0030
```

LightGBM, CatBoost and XGBoost all perform similarly well on the imbalanced fraud detection problem, with **LightGBM having a slight edge**.

Random Forest, while still good, is outperformed by the gradient boosting models.

```
Evaluating Random Forest with Cross-Validation...
Random Forest Cross-Validated F1: 0.7733 ± 0.0039

Evaluating XGBoost with Cross-Validation...
XGBoost Cross-Validated F1: 0.7885 ± 0.0042

Evaluating LightGBM with Cross-Validation...
LightGBM Cross-Validated F1: 0.7886 ± 0.0049

Evaluating CatBoost with Cross-Validation...
CatBoost Cross-Validated F1: 0.7840 ± 0.0038
```

LightGBM and XGBoost provide the best performance in terms of the F1 score, with minimal difference between them. CatBoost performs well but slightly trails.

Random Forest is outperformed by all three gradient-boosting models for this fraud detection task.

2) Provide an estimate of performance using an appropriate sample

After running cros validation on the 4 models (for F1) :

```
Evaluating Random Forest with Cross-Validation...
Random Forest Cross-Validated ROC AUC: 0.9146 ± 0.0022

Evaluating XGBoost with Cross-Validation...
XGBoost Cross-Validated ROC AUC: 0.9245 ± 0.0024

Evaluating LightGBM with Cross-Validation...
LightGBM Cross-Validated ROC AUC: 0.9267 ± 0.0026

Evaluating CatBoost with Cross-Validation...
CatBoost Cross-Validated ROC AUC: 0.9252 ± 0.0030
```

LightGBM, CatBoost and XGBoost **all perform similarly** well on the imbalanced fraud detection problem, with LightGBM having a slight edge.

Random Forest, while still good, is outperformed by the gradient boosting models.

```
Evaluating Random Forest with Cross-Validation...
Random Forest Cross-Validated F1: 0.7733 ± 0.0039

Evaluating XGBoost with Cross-Validation...
XGBoost Cross-Validated F1: 0.7885 ± 0.0042

Evaluating LightGBM with Cross-Validation...
LightGBM Cross-Validated F1: 0.7886 ± 0.0049

Evaluating CatBoost with Cross-Validation...
CatBoost Cross-Validated F1: 0.7840 ± 0.0038
```

LightGBM and XGBoost provide the best performance in terms of the F1 score, with minimal difference between them. CatBoost performs well but slightly trails.

Random Forest is outperformed by all three gradient-boosting models for this fraud detection task.

2) Provide an estimate of performance using an appropriate sample

In order to further improve our model we can run some Hyper parameter tuning:

```
#hyperparameter grid for XGBoost
xgboost_param_grid_extended = {
    'n_estimators': [200, 500, 1000],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [4, 6, 9, 12],
    'min_child_weight': [1, 5, 10],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0],
    'gamma': [0, 0.1, 0.5, 1],
    'scale_pos_weight': [1, 1.5, 2],
    'reg_lambda': [1, 5, 10],
    'reg_alpha': [0, 0.1, 1],
    'tree_method': ['hist'],
    'eval_metric': ['logloss']
}
```

Best parameters for XGBoost:
'tree_method': 'hist'
'subsample': 0.8
'scale_pos_weight': 1
'reg_lambda': 10
'reg_alpha': 0.1
'n_estimators': 1000
'min_child_weight': 1
'max_depth': 12
'learning_rate': 0.05
'gamma': 0
'eval_metric': 'logloss'
'colsample_bytree': 0.7

We go ahead and save these parameters and apply it in our model

2) Provide an estimate of performance using an appropriate sample

In order to further improve our model we can run some Hyper parameter tuning (Focused on improving F1 score):

```
Final Model Evaluation Metrics for XGBoost:
Accuracy: 0.7124
F1 Score: 0.3060
Precision: 0.2010
Recall: 0.6402
ROC AUC: 0.7525
```

We can see across the board our metrics have improved

We can now apply some threshold tuning to further improve our results

```
Best Threshold: 0.6639042496681213
Best F1 Score at this threshold: 0.31988322385569806

Final Model Evaluation with Threshold 0.6639042496681213:
Accuracy: 0.7826
F1 Score: 0.3199
Precision: 0.2317
Recall: 0.5163
ROC AUC: 0.7525
```

We can see we have now increased our precision and F1 score

2) Provide an estimate of performance using an appropriate sample

We now apply the same to **LightGBM - Hyper parameter tuning + cross validation**

```
Final Model Evaluation Metrics LightGBM (Hyper Parameter tuning):
Accuracy: 0.8579
F1 Score: 0.3477
Precision: 0.3188
Recall: 0.3824
ROC AUC: 0.7690
```

We can see a massive improvement in all the metrics other than Recall which fell from 0.48 to 0.38

We can now apply some threshold tuning to improve the recall and further improve our results

```
Best Threshold: 0.39501304160266776
Best F1 Score at this threshold: 0.3540983606557377

Final Model Evaluation with Threshold 0.39501304160266776:
Accuracy: 0.8293
F1 Score: 0.3541
Precision: 0.2831
Recall: 0.4726
ROC AUC: 0.7690
```

We can see we have now increased our recall to 0.47 and have good metrics

2) Provide an estimate of performance using an appropriate sample

We now apply the same to **CatBoost - Hyper parameter tuning + cross validation**

```
Final Model Evaluation Metrics:
Accuracy: 0.7785
F1 Score: 0.3026
Precision: 0.2199
Recall: 0.4854
ROC AUC: 0.7340
```

We can see a good improvement in all the metrics other than Recall which fell from 0.56 to 0.48

We can now apply some threshold tuning to improve our results

```
Best Threshold: 0.559881590267627
Best F1 Score at this threshold: 0.3145980707395498

Final Model Evaluation with Threshold 0.559881590267627:
Accuracy: 0.8224
F1 Score: 0.3146
Precision: 0.2546
Recall: 0.4116
ROC AUC: 0.7340
```

We can see we have now increased our F1 and precision

2) Provide an estimate of performance using an appropriate sample

Now lets build a ensemble model with higher weights to LightGBM since it gave the best results:

```
ensemble_model = VotingCla
    estimators=[
        ('catboost', CatBo
        ('xgboost', XGBCla
        ('lightgbm', LGBMC
    ],
    voting='soft',
    weights=[0.4, 0.2, 1]
```

We can see we have given LightGBM a higher weight as compared to XGBoost and CastBoost

We can now apply some threshold tuning to improve the recall and further improve our results

```
Best Threshold: 0.378266627643147
Best F1 Score at this threshold: 0.2844004244782455

Final Model Evaluation with Threshold 0.378266627643147:
Accuracy: 0.6628
F1 Score: 0.2844
Precision: 0.1800
Recall: 0.6765
ROC AUC: 0.7288
```

3) Please explain your methodology

Data Preprocessing:

- SMOTENC used to handle imbalanced data by generating synthetic minority class samples.
- Feature Encoding: Label encoding for ordinal features and one-hot encoding

Feature Selection: Correlation, Chi-squared Test, Random Forest, and XGBoost feature importance used to rank features.

Model Selection: Random Forest, XGBoost, LightGBM, and CatBoost were chosen for their robustness with imbalanced data

Hyperparameter Tuning: RandomizedSearchCV used for optimizing model parameters for each algorithm.

Evaluation Metrics:

- F1 Score to balance precision and recall.
- ROC AUC to assess overall performance, precision and recall to focus on minimizing FP and FN

Ensemble Learning:

- VotingClassifier and StackingClassifier to combine model strengths.
- Weights assigned to models based on individual performance, prioritizing LightGBM for higher precision.

Early Stopping: Used in LightGBM and CatBoost to prevent overfitting.

Threshold Tuning: Optimized decision threshold using precision-recall curves for better F1 score.

3) Please explain your methodology

In our case, having a **high recall** is important because we want to detect as much fraud as possible, even if it means flagging a few legitimate transactions as fraud (which lowers precision). Missing fraudulent transactions could lead to bigger issues, so it's better to have a model with higher recall.

- **Best Recall:** The ensemble model had the highest recall at 0.6765, meaning it successfully flagged a significant portion of the fraud cases.
- **Best Precision:** LightGBM was the best performer in terms of precision, with 0.2831, indicating it was more accurate in identifying true frauds while minimizing false positives.
- **Best F1 Score:** LightGBM also had the highest F1 score, which balances both precision and recall, at 0.3541. This shows it struck the best overall balance between identifying true frauds and minimizing false positives and false negatives.
- **Best ROC AUC:** LightGBM had the best ROC AUC at 0.7690, meaning it was the most effective at distinguishing between fraudulent and non-fraudulent transactions.

Given these results, LightGBM stood out for its balanced performance, while the ensemble model excelled in recall. High recall is crucial in fraud detection to minimize missed fraud cases, but balancing that with improved precision (to reduce false positives) would make the model even more effective.

QUESTION 5 - CONCLUSION & QUESTIONS

1) What questions you have
2) What you would do next with more time
3) Conclusion

1) What questions you have

1) Model Selection: How do you typically approach model selection for imbalanced datasets like this one? Are there any other models you've found effective beyond the ones I used (CatBoost, XGBoost, LightGBM)?

2) Hyperparameter Tuning: How do you balance improving precision without sacrificing recall when tuning hyperparameters? Any specific strategies for managing this trade-off?

3) Ensemble Methods: In my assignment, I experimented with ensemble model, what is a good way to balance out the models to be more effective?

4) Class Imbalance: What other techniques have you used to handle severe class imbalance, apart from SMOTE?

2) What you would do next with more time

If I had more time, I would focus on:

1) I would **explore more complex versions of tree-based models** like XGBoost, LightGBM and CatBoost, increasing model depth, number of trees and refining hyperparameters. Additionally, I would experiment with advanced ensemble methods to capture more complex data patterns and improve performance

2) **Experimenting with SMOTE techniques** like hybrid sampling or fine-tuning SMOTENC for better data balancing

3) Exploring **feature engineering** to derive new interaction features or domain-specific insights

4) Optimizing hyperparameters with **Bayesian optimization** to find better model configurations

5) Stacking **more diverse models and meta-models** for improved ensemble predictions

6) I also had code written for other feature extraction methods like VIF and RFE, so I would explore that and see how that affects the outcome

3) Conclusion

Based on the results with hyperparameter tuning, threshold optimization and cross-validation LightGBM was the best-performing model in terms of both F1 score and precision:

LightGBM:

- F1 Score: 0.3541 (Best F1 score)
- Precision: 0.2831 (Best precision)
- Recall: 0.4726
- Accuracy: 0.8293 (Best accuracy)
- ROC AUC: 0.7690 (Best ROC AUC)

Why LightGBM performed best:

- LightGBM had the best **balance between precision and recall**, leading to the highest F1 score.
- It also maintained the **highest accuracy** and ROC AUC among all models, meaning it better captured the differences between the classes, even in an imbalanced dataset.
- Despite not having the highest recall, its precision and F1 score made it the strongest model overall.

THANK YOU!