GROUP 2

AML PROJECT DELIVERABLE 2

CREDIT SCORE CLASSIFICATION

GOKUL NATESAN, MADHULIKA BALAKUMAR, SHUMAIL SAJJAD, YASH DANGE

CONTENT

1. INITIAL DATA EXPLORATION

At a glance:

By simply taking a look at some basic summary information about the data, we can identify various issues that need to be addressed about the data, before it is ready to be trained using a model.

NUMBER OF ROWS: 100,000

NUMBER OF COLUMNS: 28

TARGET COLUMN: CREDIT SCORE

```
credit_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   ID                        100000 non-null  object
 1   Customer_ID               100000 non-null  object
 2   Month                     100000 non-null  object
 3   Name                      90015 non-null   object
 4   Age                       100000 non-null  object
 5   SSN                       100000 non-null  object
 6   Occupation                100000 non-null  object
 7   Annual_Income             100000 non-null  object
 8   Monthly_Inhand_Salary     84998 non-null   float64
 9   Num_Bank_Accounts         100000 non-null  int64
 10  Num_Credit_Card           100000 non-null  int64
 11  Interest_Rate             100000 non-null  int64
 12  Num_of_Loan               100000 non-null  object
 13  Type_of_Loan              88592 non-null   object
 14  Delay_from_due_date       100000 non-null  int64
 15  Num_of_Delayed_Payment    92998 non-null   object
 16  Changed_Credit_Limit      100000 non-null  object
 17  Num_Credit_Inquiries      98035 non-null   float64
 18  Credit_Mix                100000 non-null  object
 19  Outstanding_Debt          100000 non-null  object
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        90970 non-null   object
 22  Payment_of_Min_Amount     100000 non-null  object
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   95521 non-null   object
 25  Payment_Behaviour         100000 non-null  object
 26  Monthly_Balance           98800 non-null   object
 27  Credit_Score              100000 non-null  object
dtypes: float64(4), int64(4), object(20)
memory usage: 21.4+ MB
```

As highlighted, there are various issues about the data at hand such as:

a. Many columns have certain missing values. The dataset has 1,00,000 total rows but the number of non-null counts in columns like Name, Monthly salary, Loan Type, etc. are less than that. We need to address that.

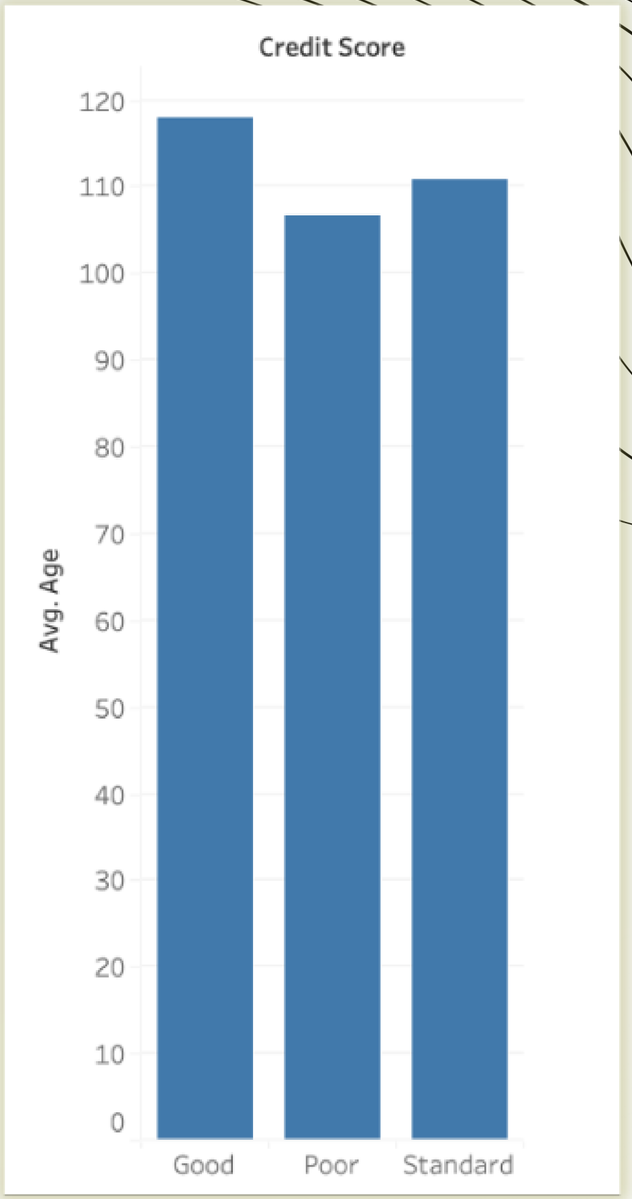
b. The datatype for numerical columns like ‘Age’, ‘Annual Income’ should not be object. We need to convert it to a numerical datatype like floar or int, whichever is more suitable.

There are various issues that could arise if we don’t do this. For instance, if we calculate the summary statistics of the dataframe using the ‘describe’ function in pandas, we will not be able to see the stats for columns like Age, Income etc, despite the fact that they are numerical.

AGE VS CREDIT SCORE

In the graph (drawn using Tableau), we can see that the average age for all three classes of credit score is well over 100 years. This is certainly not consistent with expected values and indicates that outliers exist within the ‘Age’ column.

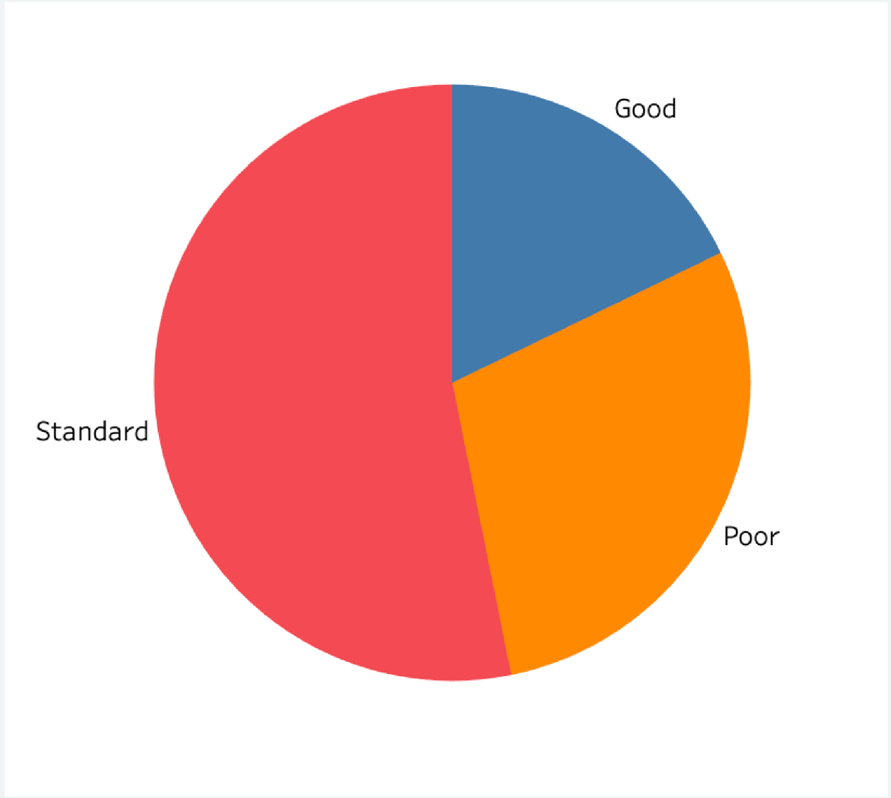
Note: This graph was made possible by Tableau which could identify Age as a numerical column and ignored the ‘_’ (underscore) actually present in some instances in the data.



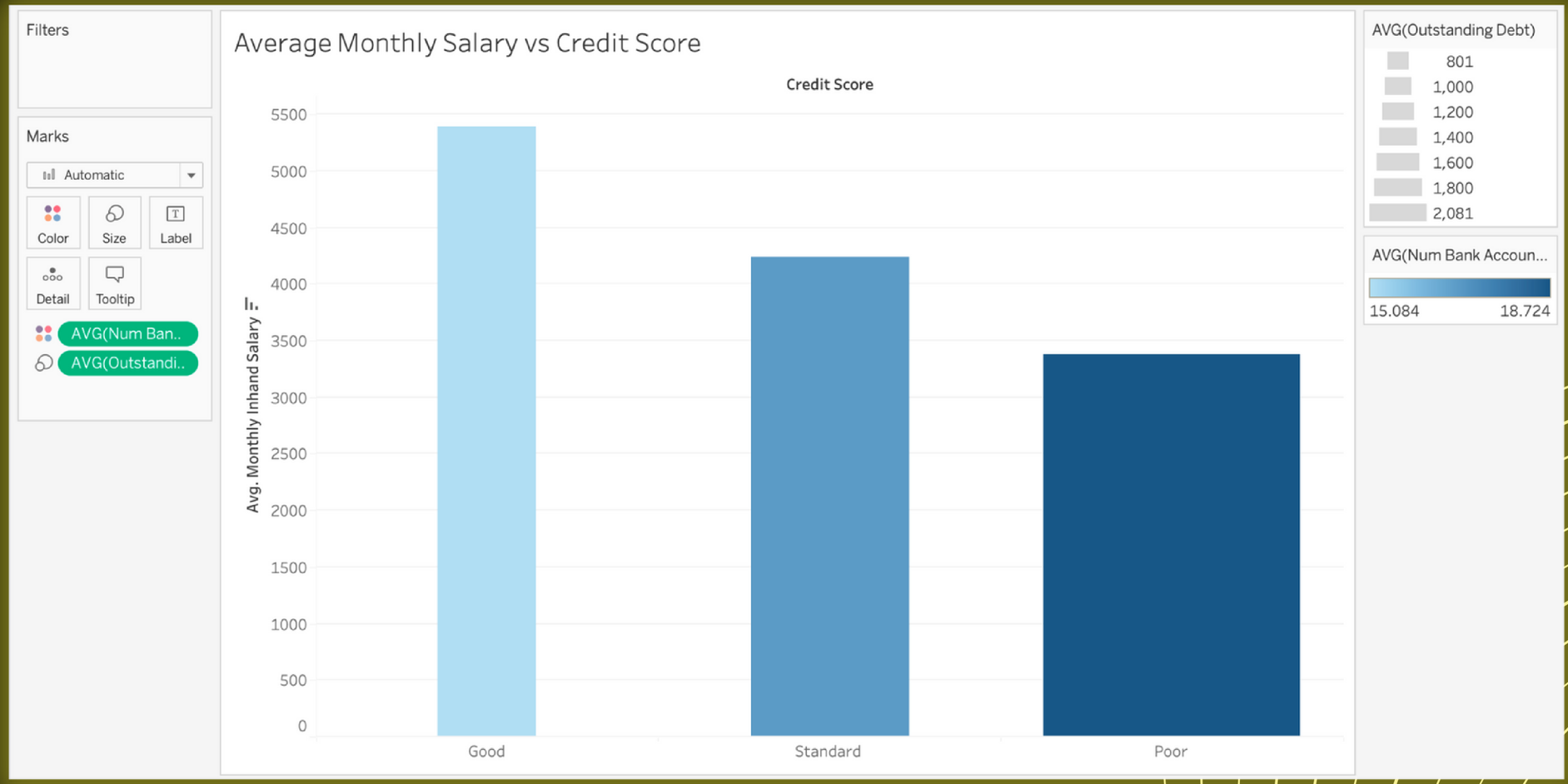
CLASS BALANCE: CREDIT SCORE

As shown in the figure, there are three possible classes for the target Column, viz. Standard, Poor, and Good. It can be seen that a majority of the class values are Standard, and Good or Poor labels are comparatively less. Since there are relatively good number of representations of each class interval, we can say the dataset is balanced.

Note: Since we have three classes in the target column, we will be doing either of 'One vs One' or 'One vs Many' approach while model training.



AVG. MONTHLY SALARY VS CREDIT SCORE



Note: Plot drawn on data *before* preprocessing

In the previous graph, which a bit more advanced, packs a lot more information about the target column by using attributes such as size, and colour to portray information.

Apart from the basic bar graph which shows that the people with the highest montly income have the most representation in ‘Good’ credit score class, it can be noted that the size of all bars is not the same, We have displayed additional information about the average outstanding debt of all customers in each class using this attribute. Lesser width of the bar indicates higher debt, which is naturally seen among people with a poor credit score.

Finally using colour, we have displayed the average number of bank accounts the customer holds. Again, smaller the better. The same trend is reflected here as well, with people in the poor category showing the darkest shade of blue, indicating the most accounts.

2. CLEANING AND SAMPLING

Age Column

Upon a brief observation of the data, we notice that there are misplaced underscores '_' occasionally, the datatype is 'object' and some absurdly high values of age (5000) are present. Hence, we will fix this by

1. Eliminating the underscore
2. Converting the datatype to integer
3. Replacing 'abnormal' age values (here assumed to be under 10 and over 70) by corresponding age values of other columns with same customer ID and a 'normal' age value.

Before

Customer_ID	Month	Name	Age
CUS_0xd40	January	Aaron Maashoh	23
CUS_0xd40	February	Aaron Maashoh	23
CUS_0xd40	March	Aaron Maashoh	-500
CUS_0xd40	April	Aaron Maashoh	23
CUS_0xd40	May	Aaron Maashoh	23
CUS_0xd40	June	Aaron Maashoh	23
CUS_0xd40	July	Aaron Maashoh	23
CUS_0xd40	August		23

After

```
credit_df[credit_df['Customer_ID'] == 'CUS_0xd40']['Age']
```

```
0    23
1    23
2    23
3    23
4    23
5    23
6    23
7    23
Name: Age, dtype: int64
```

Before

A	B	C	D
ID	Customer_ID	Month	Name
0x1602	CUS_0xd40	January	Aaron Maashoh
0x1603	CUS_0xd40	February	Aaron Maashoh
0x1604	CUS_0xd40	March	Aaron Maashoh
0x1605	CUS_0xd40	April	Aaron Maashoh
0x1606	CUS_0xd40	May	Aaron Maashoh
0x1607	CUS_0xd40	June	Aaron Maashoh
0x1608	CUS_0xd40	July	Aaron Maashoh
0x1609	CUS_0xd40	August	
0x160e	CUS_0x21b1	January	Rick Rothackerj
0x160f	CUS_0x21b1	February	Rick Rothackerj
0x1610	CUS_0x21b1	March	Rick Rothackerj

After

```
credit_df[credit_df['Customer_ID'] == 'CUS_0xd40']['Name']
```

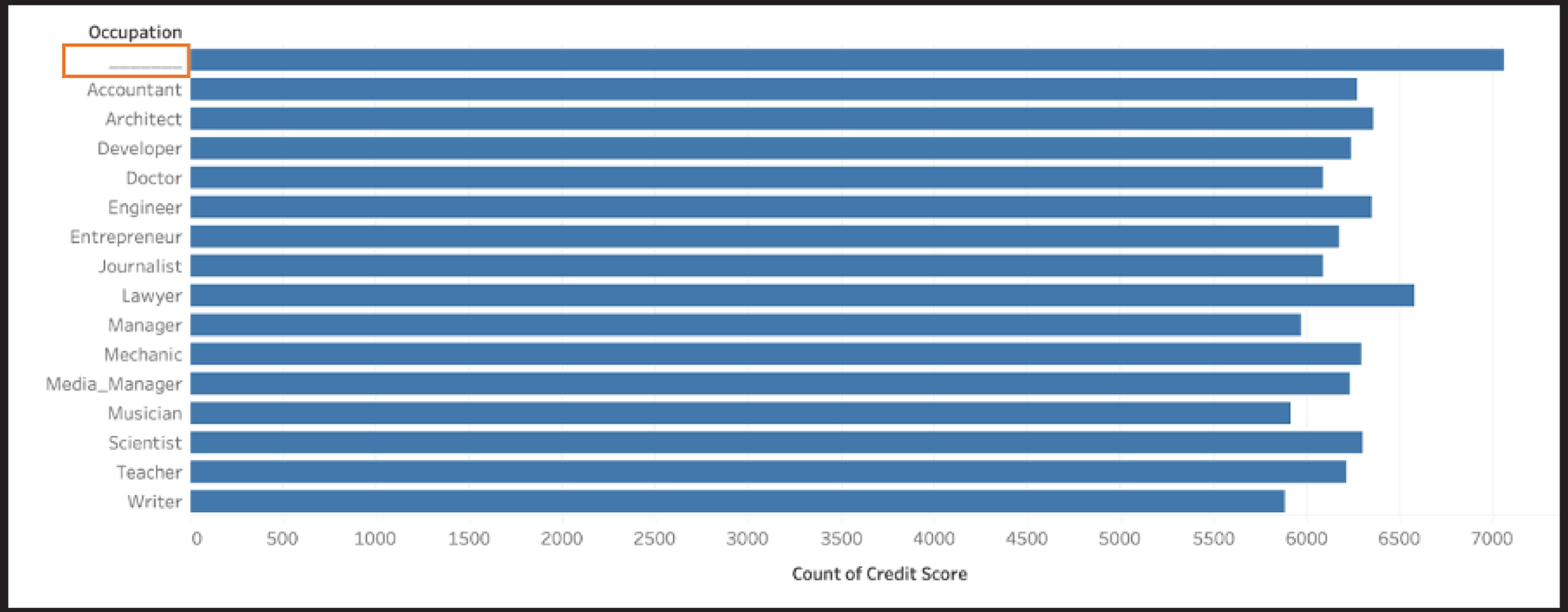
```
0    Aaron Maashoh
1    Aaron Maashoh
2    Aaron Maashoh
3    Aaron Maashoh
4    Aaron Maashoh
5    Aaron Maashoh
6    Aaron Maashoh
7    Aaron Maashoh
Name: Name, dtype: object
```

Name Column

Note: There's actually no need to handle missing values in columns such as 'Name' since we will eventually drop it during the model training stages. However, this data presents us with a unique opportunity to handle these values by using the corresponding customer-id associated with every name, and hence we have implemented it. Using such techniques can help us learn different ways in which missing values can be handled depending on the data - apart from traditional methods like dropping values of interpolation.

OCCUPATION COLUMN

Upon observing various plots in Tableau, we noticed that in the occupation vs target column (credit score) graph, there was an anomaly. An occupation that just had '_____' mentioned in it.



Similar to above techniques, we can simply replace occupation of '_____' with the correct occupation of the applicant by looking up the corresponding customer_id

```
credit_df[credit_df['Customer_ID'] == 'CUS_0x4d43']['Occupation']
```

```
200    Entrepreneur
201         _______
202         _______
203    Entrepreneur
204    Entrepreneur
205    Entrepreneur
206    Entrepreneur
207    Entrepreneur
Name: Occupation, dtype: object
```

Before

```
credit_df[credit_df['Customer_ID'] == 'CUS_0x4d43']['Occupation']
```

```
200    Entrepreneur
201    Entrepreneur
202    Entrepreneur
203    Entrepreneur
204    Entrepreneur
205    Entrepreneur
206    Entrepreneur
207    Entrepreneur
Name: Occupation, dtype: object
```

After

Important thing to note here is that: even though all seemed fine about the occupation column in the initial data.info part (no missing values, datatype was categorical as expected), there were certain issues with the data. Unlike the Name column which couldn't have any influence on our target column, the occupation might be an important feature and thereby needs to be cleaned properly. This goes to show that data cleaning should be done carefully and scrupulously.

Annual Income

Similar to previous cases, the annual income column has two issues. A. ‘_’ present on certain intervals, B. Object datatype, C. Presence of outliers.

After handling underscores and converting the datatype, we handled the outliers by using the correct annual income from the corresponding customer_id, as shown in previous cases.

Before

34	411-51-0676	Lawyer	131313.4
34	411-51-0676	Lawyer	131313.4
34	#F%$D@*&8	Lawyer	131313.4
34	411-51-0676	Lawyer	131313.4
34	411-51-0676	Lawyer	131313.4
34	#F%$D@*&8	Lawyer	10909427
34	411-51-0676	Lawyer	131313.4

After

```
credit_df[credit_df['Customer_ID'] == 'CUS_0x284a']['Annual_Income']
```

```
48    131313.4
49    131313.4
50    131313.4
51    131313.4
52    131313.4
53    131313.4
54    131313.4
55    131313.4
Name: Annual_Income, dtype: float64
```

Monhtly Inhand Salary

```
print('Null monthly salary count = ',credit_df['Monthly_Inhand_Salary'].isnull().sum())
credit_df['Monthly_Inhand_Salary'].replace(np.nan, -1.0, inplace=True)
null_msalary = credit_df[credit_df['Monthly_Inhand_Salary']==-1.0]['Customer_ID'].values
for id in null_msalary:
    salary = credit_df[credit_df['Customer_ID'] == id]['Monthly_Inhand_Salary'].drop_duplicates().values[0]
  #salary=credit_df[credit_df['Customer_ID'] ==
   #               id]['Monthly_Inhand_Salary'].drop_duplicates().sort_values().values[-1]
    if salary == -1.0:
        salary = credit_df[credit_df['Customer_ID'] == id]['Monthly_Inhand_Salary'].drop_duplicates().values[1]
    credit_df.loc[(credit_df['Customer_ID'] == id) & (credit_df['Monthly_Inhand_Salary'] == -1.0), ['Monthly_Inhand_
```

```
Null monthly salary count =  15002
```

As we saw in the data.info( ) section, there are missing values in this columns. Precisely, 15002 values are missing. We first assign a dummy value, '-1' to all these missing values and then replace it with corresponding salary values of its customer id. The same process has been shown below:

Before

	SSN	Occupation	Annual_Income	Monthly_Inh
23	821-00-0265	Scientist	19114.12	1824.84333
23	821-00-0265	Scientist	19114.12	
00	821-00-0265	Scientist	19114.12	
23	821-00-0265	Scientist	19114.12	
23	821-00-0265	Scientist	19114.12	1824.84333
23	821-00-0265	Scientist	19114.12	
23	821-00-0265	Scientist	19114.12	1824.84333

After

```
credit_df[credit_df['Customer_ID'] == 'CUS_0xd40']['Monthly_Inhand_Sala
```

```
0    1824.843333
1    1824.843333
2    1824.843333
3    1824.843333
4    1824.843333
5    1824.843333
6    1824.843333
7    1824.843333
Name: Monthly_Inhand_Salary, dtype: float64
```

Number of bank Accounts:

Entries in this dataset consist of people with an usual amount of bank accounts.
For example, a person in the list below holds 1414 according to one entry and 8 according to all ither entries. This is clearly an error which would massively impact the model if not treated.

CUS_0x95b5	August	Lisa Baertleinu	22	602-55-1355	Engineer	12986.745	959.22875	4
CUS_0x4004	January	Carlosj	43	679-26-6464	Writer	58317	4664.75	8
CUS_0x4004	February	Carlosj	43	679-26-6464	Writer	58317	4664.75	8
CUS_0x4004	March	Carlosj	44	679-26-6464	Writer	58317		8
CUS_0x4004	April	Carlosj	44	679-26-6464	Writer	58317		1414
CUS_0x4004	May		44	679-26-6464	Writer	58317	4664.75	8
CUS_0x4004	June	Carlosj	44	679-26-6464	Writer	58317	4664.75	8
CUS_0x4004	July	Carlosj	44	679-26-6464	Writer	58317	4664.75	8
CUS_0x4004	August	Carlosj	44	679-26-6464	Writer	58317	4664.75	8

We will thus adjust all entries where bank accounts are more than 11 or less than 0 (summary data shows there are negative values too).

```
In [64]: len(credit_df[(credit_df['Num_Bank_Accounts'] > 7)])
Out[64]: 24779

In [60]: len(credit_df[(credit_df['Num_Bank_Accounts'] < 0)])
Out[60]: 21
```

After this, the values of the previous customer have now been adjusted as shown:

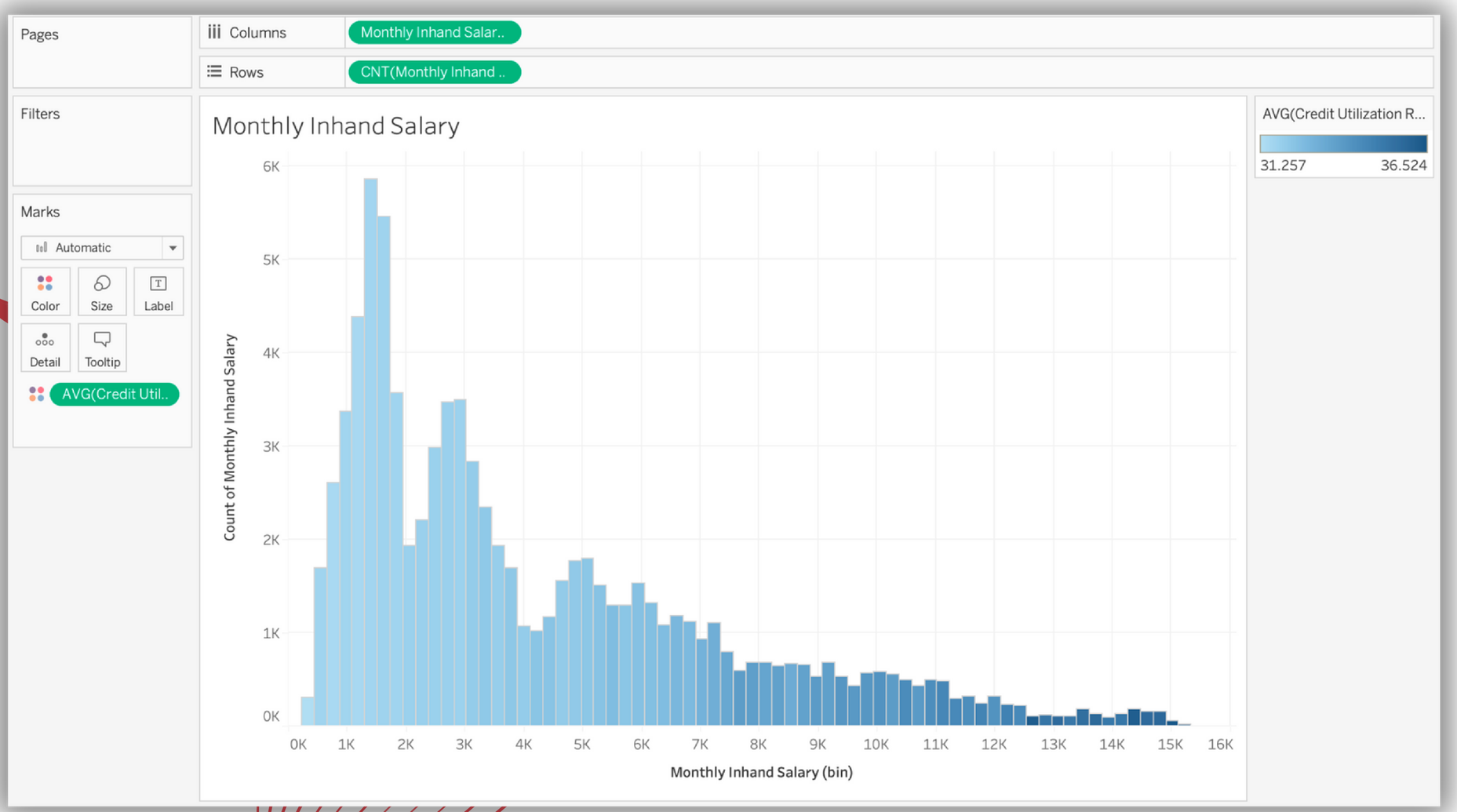
```
credit_df[credit_df['Customer_ID'] == 'CUS_0x4004']['Num_Bank_Accounts']
```

```
264    8
265    8
266    8
267    8
268    8
269    8
270    8
271    8
Name: Num_Bank_Accounts, dtype: int64
```

Similar processing techniques were applied to remaining columns like Number of Credit cards, Interest rates, Number of loans, etc.

3. INSIGHTS FROM DATA EXPLORATION

In this section, we will try to understand dependance or general trend of individual columns and the target column, and also check how these columns are correlated with each other.



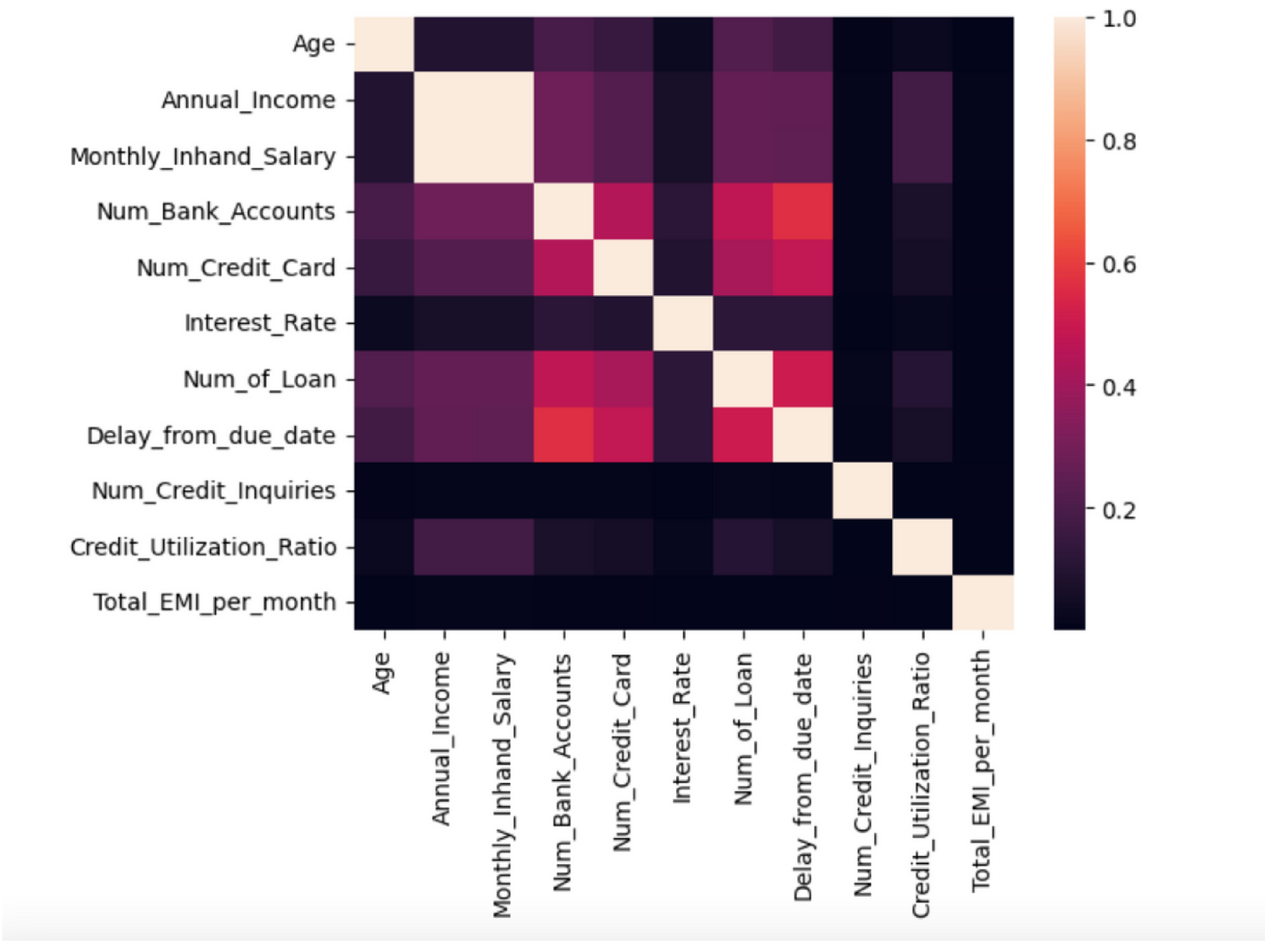
In the first graph, we plotted the frequency plot of ‘monthly in hand’ salaries of the people in the dataset. We have added the average credit utilisation ratio (in %) as a colour attribute. We can see that the data is skewed towards the left, suggesting that most people make about ~10K a month or less. People with higher income also seem to have a higher utilisation ratio of their credit line.

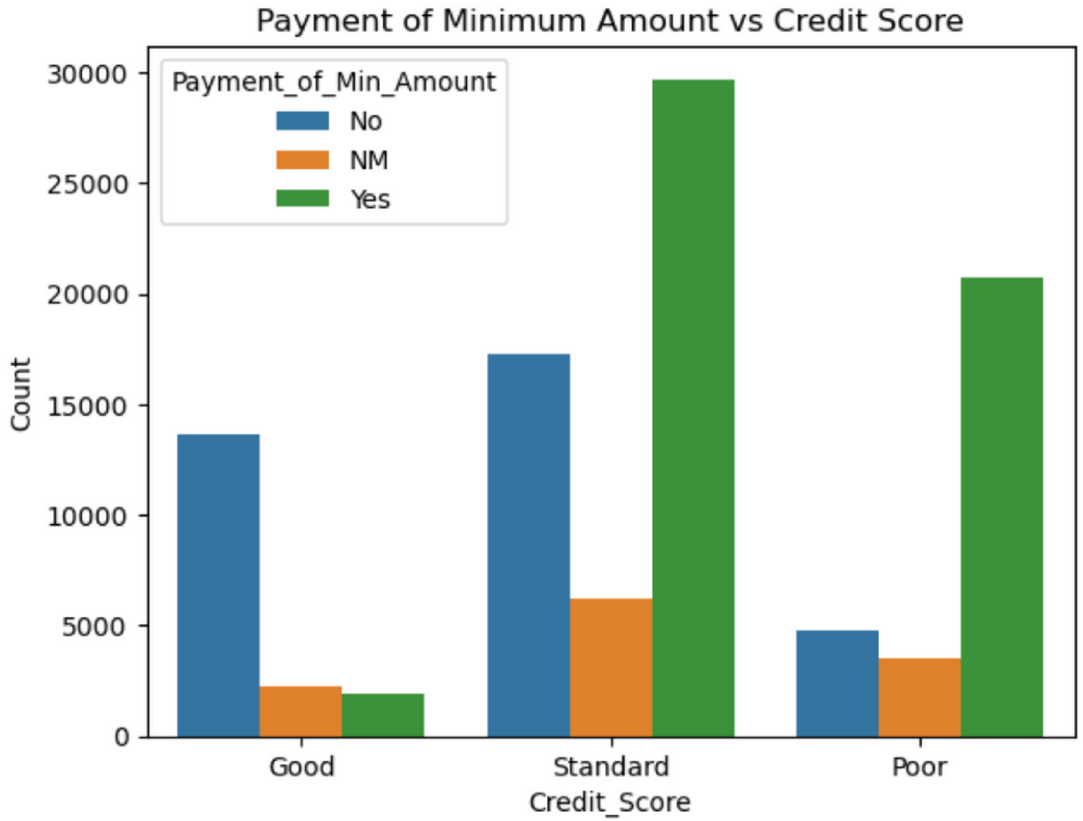
- In this correlation plot, we can see how the independent variables are associated with each other.

- As expected, Annual Income and Monthly inhand salary are almost perfectly positively correlated.

- Credit Utilization Ratio is also stronly correlated with the income value.

- Number of loans taken, Number of credit cards and number of bank accounts are also correlated, which is no surprise since having multiple loans at the same bank is unlikely.





Here, we have seen the distribution of people who only paid the minimum amount due in each class of the target column. As seen, in the ‘Good’ class, majority of people did not pay just the minimum amount due, but the full amount. In the poor class, this number is significantly higher and most people only paid the minimum amount. For standard, again, almost twice as many people only paid the minimum amount than the ones who paid potentially more than that. This goes to show that it is preferred to pay the entire amount for a good score rating, but it is not the only matric that matters.

4. MACHINE LEARNING TECHNIQUES

Since our dataset had multiple entries for each customer ID and almost the same demographic and income information across entries for customers, we used a custom splitter to group data by ‘Customer_ID’ and did a stratified split based on the target class.

We have applied a standard scaler to the dataset as models like Logistic Regression and SVM require this preprocessing step

Multiclass One-vs-One Logistic Regression model with default parameters:

```
Model - One-vs-One Logistic Regression
Accuracy: 0.6346
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.48      0.54      5906
           1       0.66      0.73      0.69     10482
           2       0.56      0.62      0.59      3612

    accuracy                           0.63     20000
   macro avg       0.62      0.61      0.61     20000
weighted avg       0.63      0.63      0.63     20000
```

Multiclass One-vs-One SVM Primal model:

```
Model - One-vs-One Linear SVM
Classification Report SVM Test:
              precision    recall  f1-score   support

           0       0.65      0.45      0.53      5804
           1       0.65      0.76      0.70     10603
           2       0.52      0.51      0.51      3593

    accuracy                           0.63     20000
   macro avg       0.61      0.57      0.58     20000
weighted avg       0.63      0.63      0.62     20000
```

Multiclass One-vs-One SVM Random Forest model:

```
Model - One-vs-One Random Forest
Accuracy: 0.70375
Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.66      0.69      5804
           1       0.74      0.75      0.75     10603
           2       0.57      0.63      0.60      3593

    accuracy                           0.70     20000
   macro avg       0.68      0.68      0.68     20000
weighted avg       0.71      0.70      0.70     20000
```

CONCLUSION

This presentation has taken us through initial data exploration, data cleaning, and insightful discoveries. Our path forward includes rigorous model testing, hyperparameter tuning, and model calibration. In coming weeks, we aim to find the optimal model, enhancing decision-making and uncovering opportunities. Thank you!