Objective of the case

Situation

We are given with the credit score and their characteristics for 12.5k customers over 8 months, giving around 100k rows

Complication

Our dataset has many null values and outliers, and we must reformat certain categorical attributes for analysis

Question

How can one maintain a good credit score, and what mistakes impact a credit score?

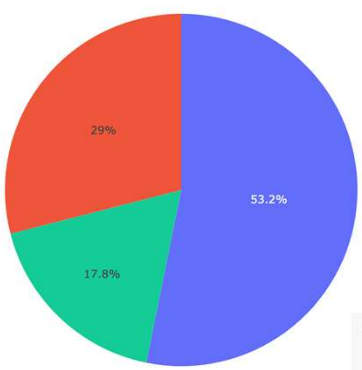
01

Understanding the dataset and the case

1. Understanding the data

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	...	Monthly_Balance	Credit_Score
0	0x1602	CUS_0xd40	January	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	...	312.494088679436	Good
1	0x1603	CUS_0xd40	February	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	...	284.62916249607184	Good
2	0x1604	CUS_0xd40	March	Aaron Maashoh	-500	821-00-0265	Scientist	19114.12	NaN	3	...	331.209862853791	Good
3	0x1605	CUS_0xd40	April	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	NaN	3	...	223.451309727367	Good
4	0x1606	CUS_0xd40	May	Aaron Maashoh	23	821-00-0265	Scientist	19114.12	1824.843333	3	...	341.489231032221	Good

df.shape = (100000, 28)



Split of customers under credit score segments

Majority of Credit Scores are in the standard segment whereas there are less than 20% customers having a good credit score

```
temp=df2.groupby('Credit_Score')['Customer_ID'].count().reset_index()
fig = px.pie(temp, values='Customer_ID', names='Credit_Score')
fig.show()
```

Data attributes

```
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)
```

02

Methodology

2.1 Data Cleaning and Processing
2.2 Handling Null Values
2.3 Handling Outliers

2.1 Data Cleaning and Processing

Checking data for:

- Strings in a numeric columns
- Special characters in categorical/numeric columns
- Unexpected negative values in natural columns

- *We deep dive into each feature in the dataset to check for unexpected values in the column. Some unexpected values, example, Age being mentioned as -500, etc. needs sanitization.*
- *Whereas other columns such as credit history age which have time-periods mentioned as '5 year and 6 months' needs to be changed to numeric columns as 5.5.*

Month

Month
January
February
March
April
May

```
months_key={'January':1,
            'February':2,
             'March':3,
             'April':4,
             'May':5,
             'June':6,
             'July':7,
             'August':8,
             'September':9,
             'October':10,
             'November':11,
             'December':12
            }
df1=df.replace({"Month": months_key})
```

Month
1
2
3
4
5

Age

```
df['Age']
```

```
0          23
1          23
2        -500
3          23
4          23
          ...
99995      25
99996      25
99997      25
99998      25
99999      25
```

```
df1['Age'] = df1['Age'].str.replace('_','')
df1['Age'] = df1['Age'].replace('-500',np.nan)

df1['Age'] = pd.to_numeric(df1['Age'])
df1.head()
```

```
df1['Age']
```

```
0        23.0
1        23.0
2         NaN
3        23.0
4        23.0
          ...
99995    25.0
99996    25.0
99997    25.0
99998    25.0
99999    25.0
```

2.1 Data Cleaning and Processing

Credit Mix

```
df1['Credit_Mix'].unique()
```

```
array(['_', 'Good', 'Standard', 'Bad'], dtype=object)
```

```
df1['Credit_Mix'] = df1['Credit_Mix'].str.replace('_','Unknown')
credit_mix_dummies = pd.get_dummies(df1['Credit_Mix'])
credit_mix_dummies = credit_mix_dummies.drop('Unknown',axis = 1)
credit_mix_dummies.head()
```

ID	Bad	Good	Standard
0x1602	0	0	0
0x1603	0	1	0
0x1604	0	1	0
0x1605	0	1	0
0x1606	0	1	0

Credit History Age

Credit_History_Age
5 Years and 9 Months
5 Years and 10 Months
0 Years and 0 Months
6 Years and 0 Months
0 Years and 0 Months

Converting text column to numeric:

- *First, we fill the nulls with 0 Years and 0 Months*
- *Then we split the elements into various columns and series*
- *Then we select the numeric part and convert it to float*

```
df1['Credit_History_Age']=df1['Credit_History_Age'].fillna('0 Years and 0 Months')
Credit_history_age1=df1['Credit_History_Age'].str.split(' Years and ', expand=True)
Credit_history_age2=Credit_history_age1[1].str.split(' ', expand=True)
Credit_history_age3= Credit_history_age1[[0]].merge(Credit_history_age2[[0]],  how='left',on ='ID')
df1['Credit_history_age_sanitized']=(Credit_history_age3['0_x']).astype(int)+(Credit_history_age3['0_y']).astype(int)/12
```

Credit_history_age_sanitized
5.750000
5.833333
0.000000
6.000000
0.000000

2.1 Data Cleaning and Processing

Type of Loan

Type_of_Loan
Debt Consolidation Loan, Personal Loan, Home E...
Debt Consolidation Loan, Personal Loan, Home E...
Debt Consolidation Loan, Personal Loan, Home E...
Debt Consolidation Loan, Personal Loan, Home E...

```
df1['Type_of_Loan']=df1['Type_of_Loan'].str.replace(', and ',', ')
df1.set_index('ID', inplace=True)
Loan_Types=df1['Type_of_Loan'].str.split(', ', expand=True)

Loan_Types.head()
ID_loan_type_series= pd.concat([Loan_Types[0],
            Loan_Types[1],
            Loan_Types[2],
            Loan_Types[3],
            Loan_Types[4],
            Loan_Types[5],
            Loan_Types[6],
            Loan_Types[7],
            Loan_Types[8]])
ID_loan_type_series.unique()
```

```
#Filling 'Type of Loan' with mode by grouping them within Customer ID. However, if the entire values are nul
df1['Type_of_Loan'] = df1.groupby('Customer_ID')['Type_of_Loan'].transform(lambda x:
                                                                            x.fillna(x.mode()[0]
                                                                                     if not x.mode().empty
                                                                                     else "Not Specified"))
```

```
df1['Auto_Loan']=df1['Type_of_Loan'].str.contains('Auto Loan').astype(int)
df1['Credit_Builder Loan']=df1['Type_of_Loan'].str.contains('Credit-Builder Loan').astype(int)
df1['Personal Loan']=df1['Type_of_Loan'].str.contains('Personal Loan').astype(int)
df1['Payday Loan']=df1['Type_of_Loan'].str.contains('Payday Loan').astype(int)
df1['Mortgage Loan']=df1['Type_of_Loan'].str.contains('Mortgage Loan').astype(int)
df1['Home Equity Loan']=df1['Type_of_Loan'].str.contains('Home Equity Loan').astype(int)
df1['Debt Consolidation Loan']=df1['Type_of_Loan'].str.contains('Debt Consolidation Loan').astype(int)
df1['Student Loan']=df1['Type_of_Loan'].str.contains('Student Loan').astype(int)
```

Auto_Loan	Credit_Builder Loan	Personal Loan	Payday Loan	Mortgage Loan	Home Equity Loan	Debt Consolidation Loan	Student Loan
1	1	1	0	0	1	0	0
1	1	1	0	0	1	0	0
1	1	1	0	0	1	0	0
1	1	1	0	0	1	0	0

2.2 Handling Null values

```
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)
```

Handling Null Values

Drop row ***Fill zero*** ***Fill Mean*** ***Fill Median*** ***Fill Mode***

Forward Fill ***Backward Fill***

❑ ***Fill zero***

Delayed Payments, changed credit limit, Credit Inquiries, Amount Invested monthly are filled with 0 assuming they are null in that month

```
#Filling the columns with 0 (Assuming there were no delayed payments, changes in credit limits, credit inquiries, amount invested as 0 if values are missing
df1[['Num_of_Delayed_Payment','Delay_from_due_date','Changed_Credit_Limit','Num_Credit_Inquiries','Amount_invested_monthly']] \
= df1[['Num_of_Delayed_Payment','Delay_from_due_date','Changed_Credit_Limit', 'Num_Credit_Inquiries','Amount_invested_monthly']].fillna(0)
```

2.2 Handling Null values

❑ *Fill mean*

Salary, Balances within same customer are filled with mean

```python
#Filling the columns with mean by grouping them within Customer_ID
df1[['Monthly_Inhand_Salary','Monthly_Balance','Credit_history_age_sanitized']] = df1.groupby('Customer_ID') \
[['Monthly_Inhand_Salary','Monthly_Balance','Credit_history_age_sanitized']].transform(lambda x: x.fillna(x.mean()))
```

❑ *Fill mode*

Categorical columns with null values are filled with mode within the same customer

```python
#Filling the columns with mode by grouping them within Customer_ID since these are categorical variables
df1['Occupation'] = df1.groupby('Customer_ID')['Occupation'].transform(lambda x: x.fillna(x.mode()[0]))
df1['Credit_Mix'] = df1.groupby('Customer_ID')['Credit_Mix'].transform(lambda x: x.fillna(x.mode()[0]))
df1['Payment_Behaviour'] = df1.groupby('Customer_ID')['Payment_Behaviour'].transform(lambda x: x.fillna(x.mode()[0]))
```

```python
#Filling 'Type of Loan' with mode by grouping them within Customer ID. However, if the entire values are null within customer id, filling them with mode
df1['Type_of_Loan'] = df1.groupby('Customer_ID')['Type_of_Loan'].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else "Not Specified"))
```

There are no null values remaining in the data after the processing

```
<class 'pandas.core.frame.DataFrame'>
Index: 100000 entries, 0x1602 to 0x25fed
Data columns (total 35 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   Customer_ID                   100000 non-null  object
 1   Month                         100000 non-null  int64
 2   Name                          90015 non-null   object
 3   Age                           100000 non-null  int64
 4   Occupation                    100000 non-null  object
 5   Annual_Income                 100000 non-null  float64
 6   Monthly_Inhand_Salary         100000 non-null  float64
 7   Num_Bank_Accounts             100000 non-null  int64
 8   Num_Credit_Card               100000 non-null  int64
 9   Interest_Rate                 100000 non-null  int64
 10  Num_of_Loan                   100000 non-null  int64
 11  Type_of_Loan                  100000 non-null  object
 12  Delay_from_due_date           100000 non-null  int64
 13  Num_of_Delayed_Payment        100000 non-null  float64
 14  Changed_Credit_Limit          100000 non-null  float64
 15  Num_Credit_Inquiries          100000 non-null  float64
 16  Credit_Mix                    100000 non-null  object
 17  Outstanding_Debt              100000 non-null  float64
 18  Credit_Utilization_Ratio      100000 non-null  float64
 19  Credit_History_Age            100000 non-null  object
 20  Payment_of_Min_Amount         100000 non-null  object
 21  Total_EMI_per_month           100000 non-null  float64
 22  Amount_invested_monthly       100000 non-null  float64
 23  Payment_Behaviour             100000 non-null  object
 24  Monthly_Balance               100000 non-null  float64
 25  Credit_Score                  100000 non-null  object
 26  Credit_history_age_sanitized  100000 non-null  float64
 27  Auto_Loan                     100000 non-null  int64
 28  Credit_Builder Loan           100000 non-null  int64
 29  Personal Loan                 100000 non-null  int64
 30  Payday Loan                   100000 non-null  int64
 31  Mortgage Loan                 100000 non-null  int64
 32  Home Equity Loan              100000 non-null  int64
 33  Debt Consolidation Loan       100000 non-null  int64
 34  Student Loan                  100000 non-null  int64
dtypes: float64(11), int64(15), object(9)
memory usage: 29.5+ MB
```

2.3 Handling Outliers - Process Overview

1

Review descriptive statistics summary

Use the DataFrame describe method to look at min, max, and std values for each variable

2

Visually analyze attributes

Plot box and whisker plots & histograms to identify which variables had large amounts of outliers

3

Decide on outlier methodology

Use statistical methods to create a uniform process for removing outliers

4

Logical updates to methodology

Update boundaries by variable based on business logic

```
[48] df1.describe(exclude=[object])
```

	Month	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan
count	100000.000000	100000.000000	1.000000e+05	100000.000000	100000.000000	100000.00000	100000.000000	100000.000000
mean	4.500000	110.649700	1.764157e+05	4198.468568	17.091280	22.47443	72.466040	3.009960
std	2.291299	686.244717	1.429618e+06	3187.369878	117.404834	129.05741	466.422621	62.647879
min	1.000000	-500.000000	7.005930e+03	303.645417	-1.000000	0.00000	1.000000	-100.000000
25%	2.750000	24.000000	1.945750e+04	1626.594167	3.000000	4.00000	8.000000	1.000000
50%	4.500000	33.000000	3.757861e+04	3096.378333	6.000000	5.00000	13.000000	3.000000
75%	6.250000	42.000000	7.279092e+04	5961.637500	7.000000	7.00000	20.000000	5.000000
max	8.000000	8698.000000	2.419806e+07	15204.633333	1798.000000	1499.00000	5797.000000	1496.000000

2.3 Handling Outliers - Process Overview

1 Review descriptive statistics summary

Use the DataFrame describe method to look at min, max, and std values for each variable

2 Visually analyze attributes

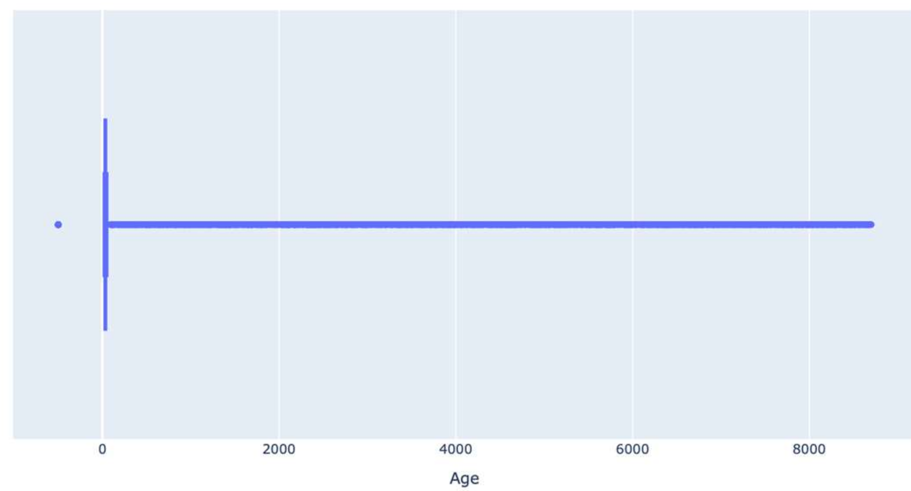
Plot box and whisker plots & histograms to identify which variables had large amounts of outliers

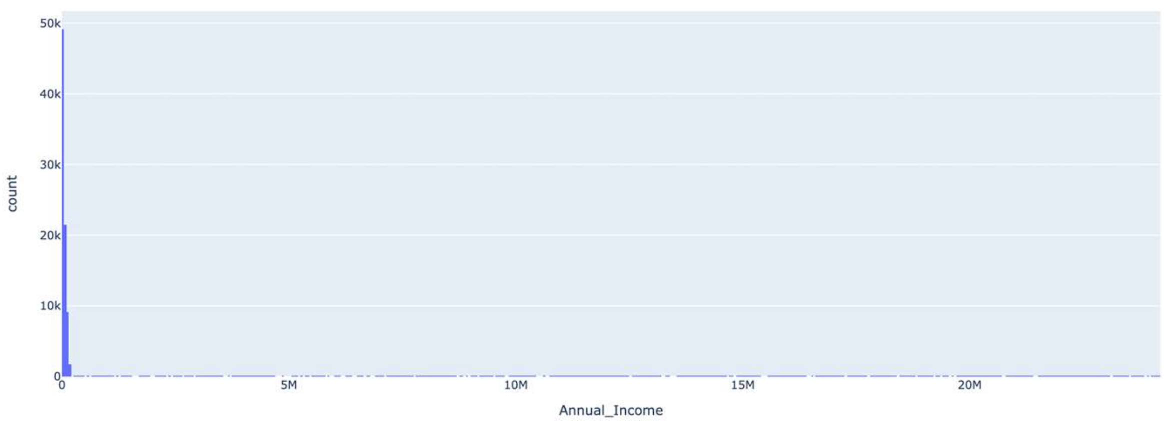
3 Decide on outlier methodology

Use statistical methods to create a uniform process for removing outliers

4 Logical updates to methodology

Update boundaries by variable based on business logic





2.3 Handling Outliers - Process Overview

1
Review descriptive statistics summary

Use the DataFrame describe method to look at min, max, and std values for each variable

2
Visually analyze attributes

Plot box and whisker plots & histograms to identify which variables had large amounts of outliers

3
Decide on outlier methodology

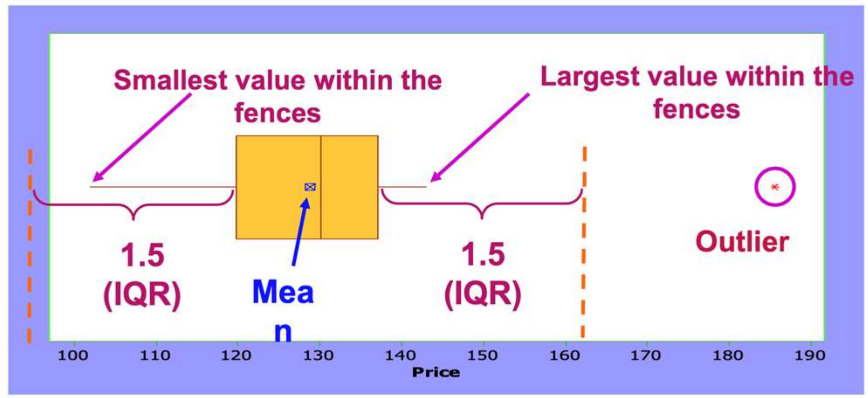
Use statistical methods to create a uniform process for removing outliers

4
Logical updates to methodology

Update boundaries by variable based on business logic

Box Plot Construction

- Box ends at Q_3 and Q_1, vertical line at median.
- Fences at $Q_1 - 1.5(IQR)$ and $Q_3 + 1.5(IQR)$
- Whiskers to the smallest and largest values inside fences
- The locations of outliers marked with *.



- $IQR = Q_3 - Q_1 = 137.0325 - 119.8125 = 17.22$
- Fence: $Q_1 - 1.5\ (IQR) = 119.8125 - 25.83 = 93.9825$
  $Q_3 + 1.5\ (IQR) = 137.0325 + 25.83 = 162.8625$

1

*From MGMT670: Business Analytics course

2.3 Handling Outliers - Process Overview

1 Review descriptive statistics summary

Use the DataFrame describe method to look at min, max, and std values for each variable

2 Visually analyze attributes

Plot box and whisker plots & histograms to identify which variables had large amounts of outliers

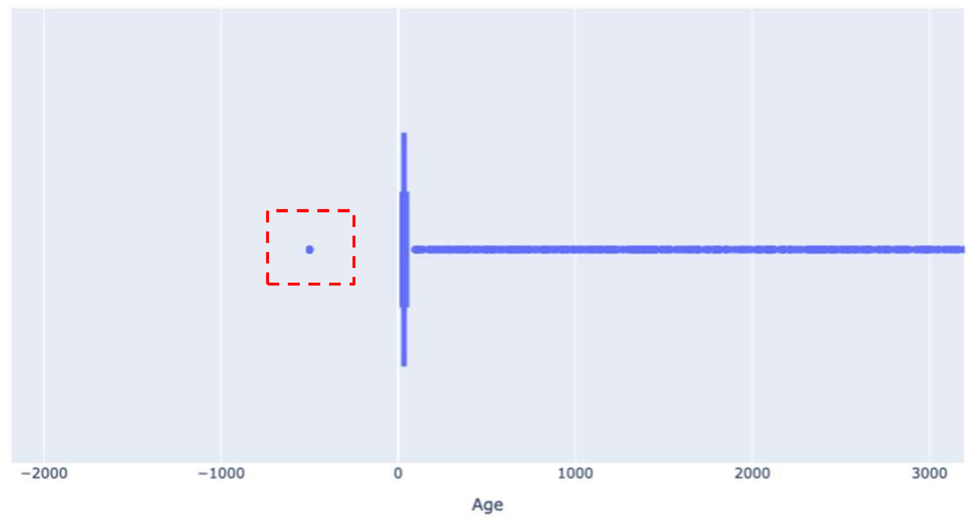
3 Decide on outlier methodology

Use statistical methods to create a uniform process for removing outliers

4 Logical updates to methodology

Update boundaries by variable based on business logic

```
fig = px.box(df, x="Age")
fig.show()
```




```
Age {'min': -500, 'lowerbound': -3.0, 'upperbound': 69.0, 'max': 8698}
```

2.3 Removing Outliers

```
RangeIndex: 100000 entries, 0 t
Data columns (total 28 columns)
 #   Column
---  ------
 0   ID
 1   Customer_ID
 2   Month
 3   Name
 4   Age
 5   SSN
 6   Occupation
 7   Annual_Income
 8   Monthly_Inhand_Salary
 9   Num_Bank_Accounts
 10  Num_Credit_Card
 11  Interest_Rate
 12  Num_of_Loan
 13  Type_of_Loan
 14  Delay_from_due_date
 15  Num_of_Delayed_Payment
 16  Changed_Credit_Limit
 17  Num_Credit_Inquiries
 18  Credit_Mix
 19  Outstanding_Debt
 20  Credit_Utilization_Ratio
 21  Credit_History_Age
 22  Payment_of_Min_Amount
 23  Total_EMI_per_month
 24  Amount_invested_monthly
 25  Payment_Behaviour
 26  Monthly_Balance
 27  Credit_Score
```

[1] *(25th percentile)-(IQR*1.5)*
[2] *(75th percentile)+(IQR*1.5)*

- The dataset was heavily right-skewed
- Age and Num_of_Loan had values outside of the lower bound[1] for outliers
- All attributes had outliers outside of the upper bound[2] for outliers

Spread of the data – Min, 25th Percentile, 75th Percentile, Max

```
Age {'min': -500, 'lowerbound': -3.0, 'upperbound': 69.0, 'max': 8698}
Num_Bank_Accounts {'min': -1, 'lowerbound': -3.0, 'upperbound': 13.0, 'max': 1798}
Num_Credit_Card {'min': 0, 'lowerbound': -0.5, 'upperbound': 11.5, 'max': 1499}
Interest_Rate {'min': 1, 'lowerbound': -10.0, 'upperbound': 38.0, 'max': 5797}
Num_of_Loan {'min': -100, 'lowerbound': -5.0, 'upperbound': 11.0, 'max': 1496}
Delay_from_due_date {'min': -5, 'lowerbound': -17.0, 'upperbound': 55.0, 'max': 67}
Num_of_Delayed_Payment {'min': -3.0, 'lowerbound': -7.0, 'upperbound': 33.0, 'max': 4397.0}
Num_Credit_Inquiries {'min': 0.0, 'lowerbound': -6.0, 'upperbound': 18.0, 'max': 2597.0}
```

2.3 Removing Outliers

1. Defining the upper and lower bounds
2. Creating boolean DataFrames to identify outliers
3. Filtering outliers from the DataFrame using an AND operator

```python
def mod_outlier(df,column):
    df = df._get_numeric_data()

    q1 = df[column].quantile(.25)
    q3 = df[column].quantile(.75)

    iqr = q3-q1

    lower_bound= q1 - (1.5*iqr)
    upper_bound= q3 + (1.5*iqr)

    return({'lowerbound':lower_bound,'upperbound':upper_bound})

#CREATE DICTIONARY FOR LOWER AND UPPER BOUNDS
databounds={}
columnlist=['Age','Num_Bank_Accounts','Num_Credit_Card','Interest_Rate',
            'Num_of_Loan','Delay_from_due_date','Num_of_Delayed_Payment',
            'Num_Credit_Inquiries']

for i in columnlist:
    databounds[i]=mod_outlier(df1,i)
```

- Created a function to calculate the lower and upper bounds using a dataframe and column as input variables
- The output was a dictionary with lower and upper bounds to filter outliers with
- Using pandas.dataframe.describe() was only gave us minimum, maximum, and quartiles – not outlier bounds

2.3 Removing Outliers

1 Defining the upper and lower bounds

2 Creating boolean DataFrames to identify outliers

3 Filtering outliers from the DataFrame using an AND operator

```
#UPDATING DF
dfage1=df1["Age"]<= 69
dfage2=df1["Age"]> 0

dfbank1=df1["Num_Bank_Accounts"]<= 13
dfbank2=df1["Num_Bank_Accounts"]>= 0

dfcc1=df1["Num_Credit_Card"]<= 11.5
dfcc2=df1["Num_Credit_Card"]>= 0

dfirate1=df1["Interest_Rate"]<= 38
dfirate2=df1["Interest_Rate"]>= 0

dfloan1=df1["Num_of_Loan"]<= 11
dfloan2=df1["Num_of_Loan"]>= 0

dfddate1=df1["Delay_from_due_date"]<= 55
dfddate2=df1["Delay_from_due_date"]>= -17

dfdpay1=df1["Num_of_Delayed_Payment"]<= 33
dfdpay2=df1["Num_of_Delayed_Payment"]>= 0

dfcred1=df1["Num_Credit_Inquiries"]<=18
dfcred2=df1["Num_Credit_Inquiries"]>0
```

- Using values from the upper and lower bound dictionary and list of columns we identified with outliers
- All highlighted lower bounds were modified because negative values did not make sense
- A negative Delay_from_due_date represents paying ahead of time

** Lower bound modified to be zero*

2.3 Removing Outliers

1. Defining the upper and lower bounds

2. Creating boolean DataFrames to identify outliers

3. Filtering outliers from the DataFrame using an AND operator

```
#UPDATING DF
dfage1=df1["Age"]<= 69
dfage2=df1["Age"]> 0

dfbank1=df1["Num_Bank_Accounts"]<= 13
dfbank2=df1["Num_Bank_Accounts"]>= 0

dfcc1=df1["Num_Credit_Card"]<= 11.5
dfcc2=df1["Num_Credit_Card"]>= 0

dfirate1=df1["Interest_Rate"]<= 38
dfirate2=df1["Interest_Rate"]>= 0

dfloan1=df1["Num_of_Loan"]<= 11
dfloan2=df1["Num_of_Loan"]>= 0

dfddate1=df1["Delay_from_due_date"]<= 55
dfddate2=df1["Delay_from_due_date"]>= -17

dfdpay1=df1["Num_of_Delayed_Payment"]<= 33
dfdpay2=df1["Num_of_Delayed_Payment"]>= 0

dfcred1=df1["Num_Credit_Inquiries"]<=18
dfcred2=df1["Num_Credit_Inquiries"]>0
```

```
#create new dataframe
outliersremoved=df1[
    dfage1&dfage2&
    dfbank1&dfbank2&
    dfcc1&dfcc2&
    dfirate1&dfirate2&
    dfloan1&dfloan2&
    dfddate1&dfddate2&
    dfdpay1&dfdpay2&
    dfcred1&dfcred2]
```

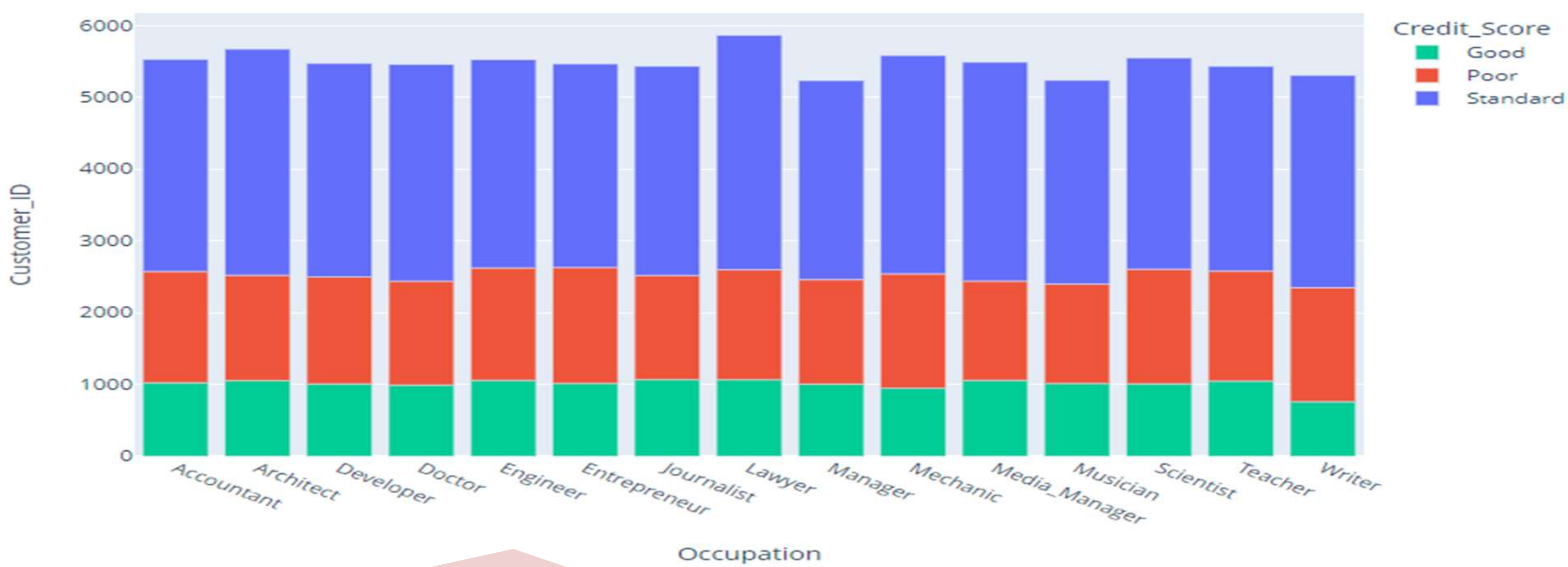
- Creating a new DataFrame using Boolean DataFrames from step 2 above
- Referencing the original DataFrame to use clean data

03

Trends in Data / EDA

3. Trends in data

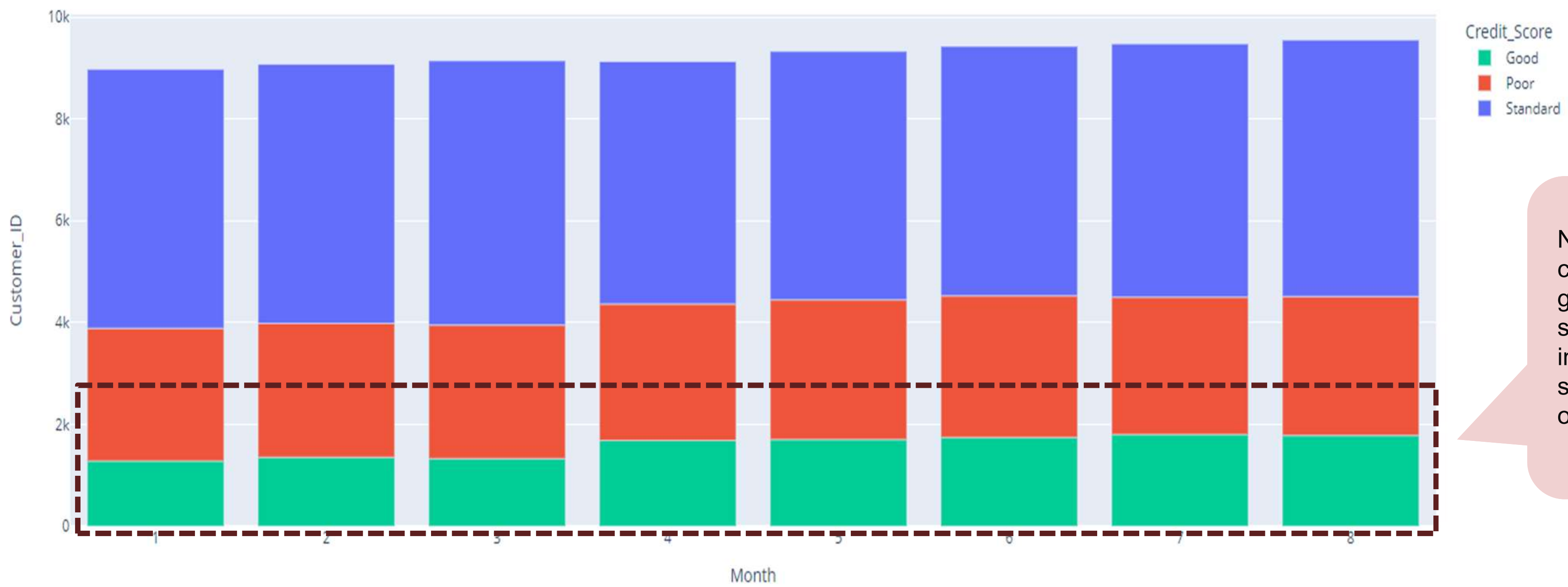
Occupation



Credit scores are almost equally distributed among all. So, it indicates that your credit score doesn't depend significantly on your profession

3. Trends in data

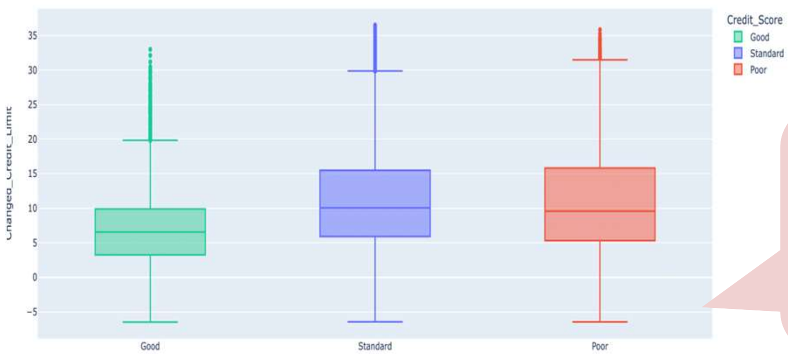
Distribution of credit score - month over month



Number of customers with good credit score tend to increase slightly month over month

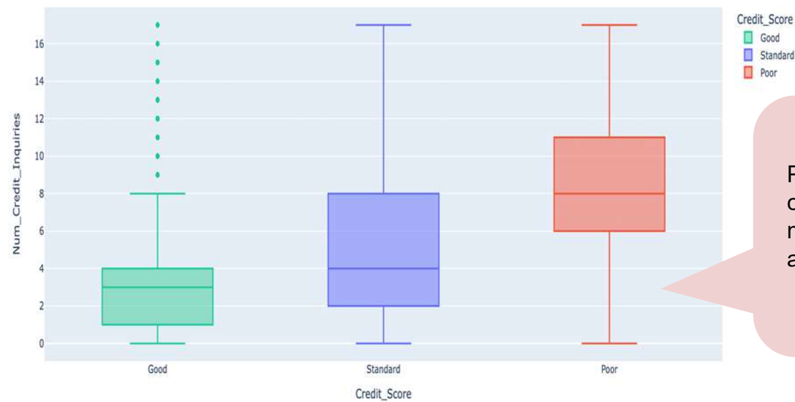
3. Trends in data

Changed Credit Limit



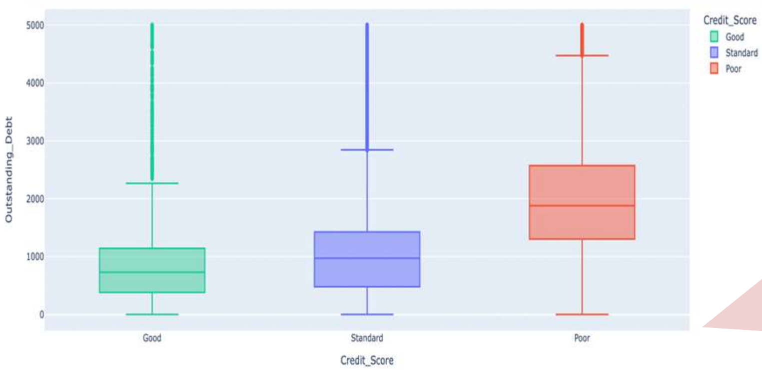
People with good credit score usually have lower variability in their credit limit.

Number of Credit Enquiries



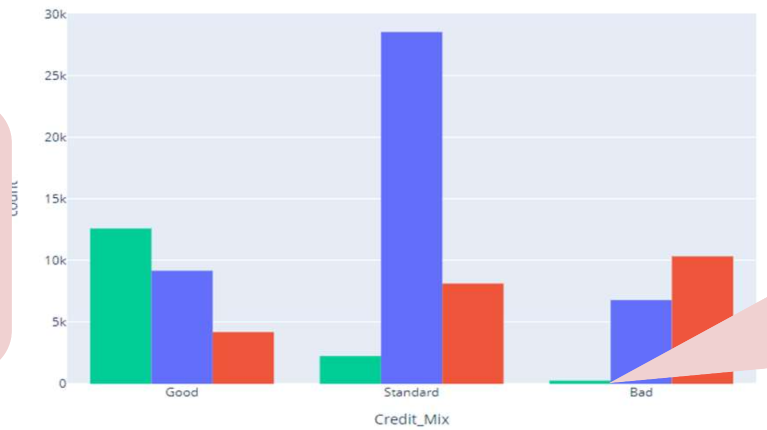
People with Poor credit score have more enquiries about the credit.

Outstanding Debt



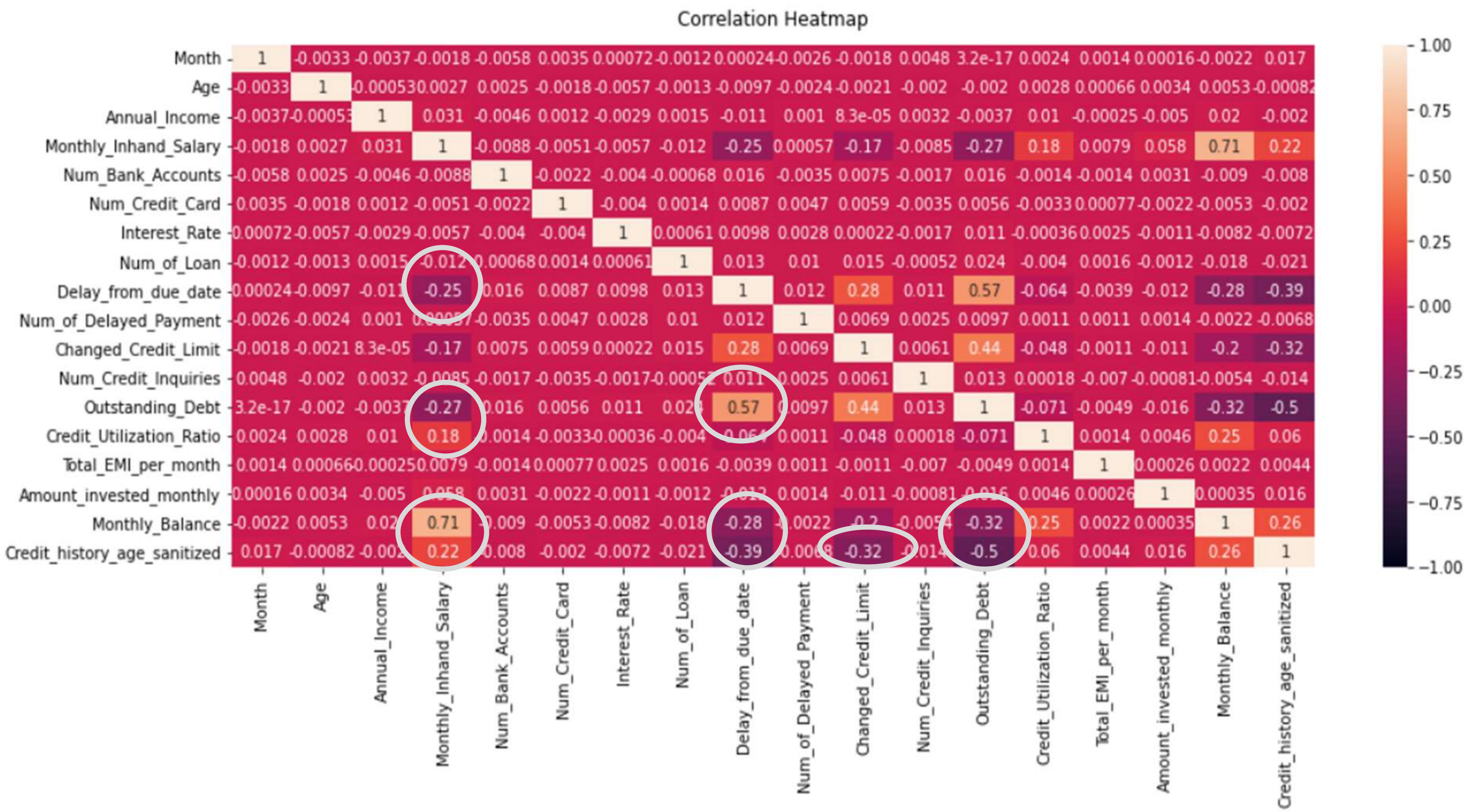
People having higher outstanding debt tend to have Poor credit score.

Distribution of credit mix



Only a few of customers with bad mix end up having good credit score maybe because of paying on time.

Factors affecting credit score



What features are correlated?

- ***Delay from due date vs monthly salary***
- ***Outstanding debt vs monthly salary***
- ***monthly balance vs monthly salary***
- ***credit history age vs monthly salary***
- ***outstanding debt vs delay from due date***
- ***monthly balance vs delay from due date***
- ***credit history age vs delay from due date***
- ***changed credit limit vs credit history age***
- ***outstanding debt vs monthly balance***
- ***outstanding debt vs credit history age***

3. Trends in data

In-hand monthly salary vs Delay from due date

As monthly salary increases people tend to pay earlier.

In-hand monthly salary vs Outstanding Debt

Most of the outstanding debt is owed by people who have lower monthly salary

In-hand monthly salary vs Monthly Balance

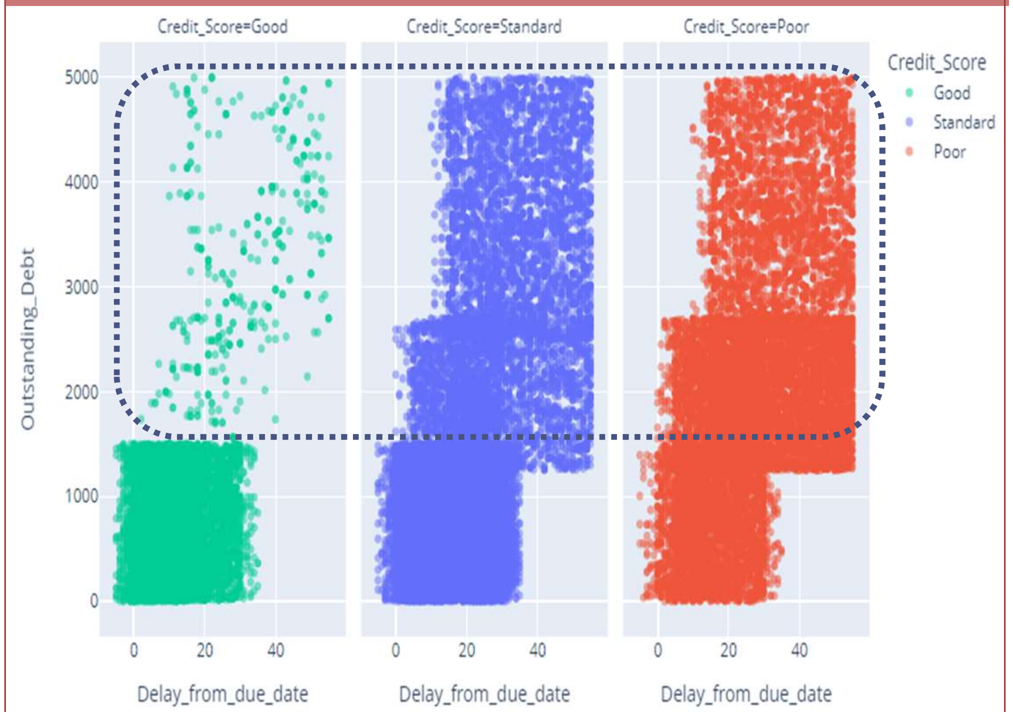
Monthly in-hand salary is linearly proportional to monthly balance.

In-hand monthly salary vs Credit History Age

As monthly salary increases, people tend to have a higher credit history age.

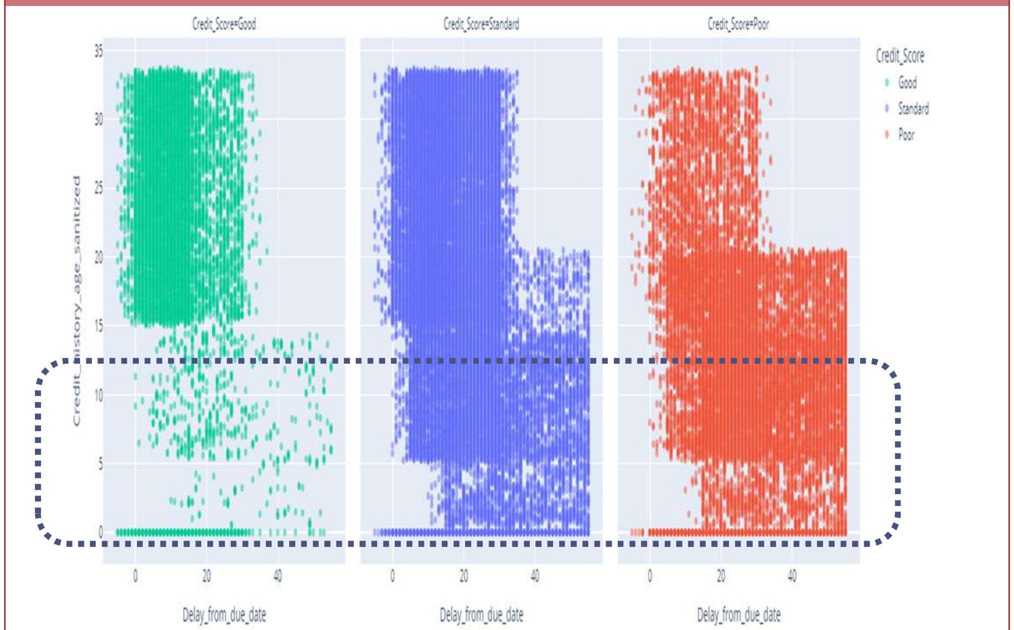
3. Trends in data

Delay from due date vs Outstanding Debt



- As the payment is delayed from the due date, outstanding debt increases.
- People with good credit score and high outstanding debt clear the dues early.

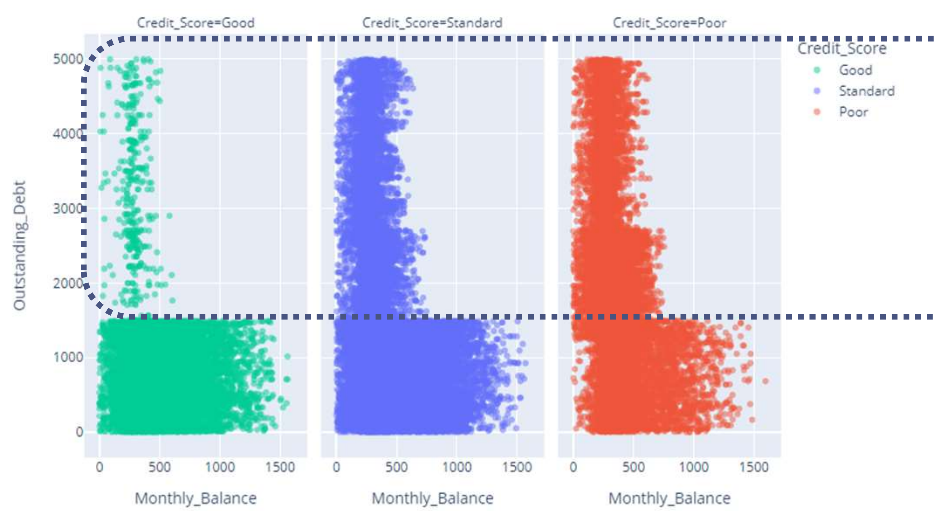
Delay from due date vs Credit History Age



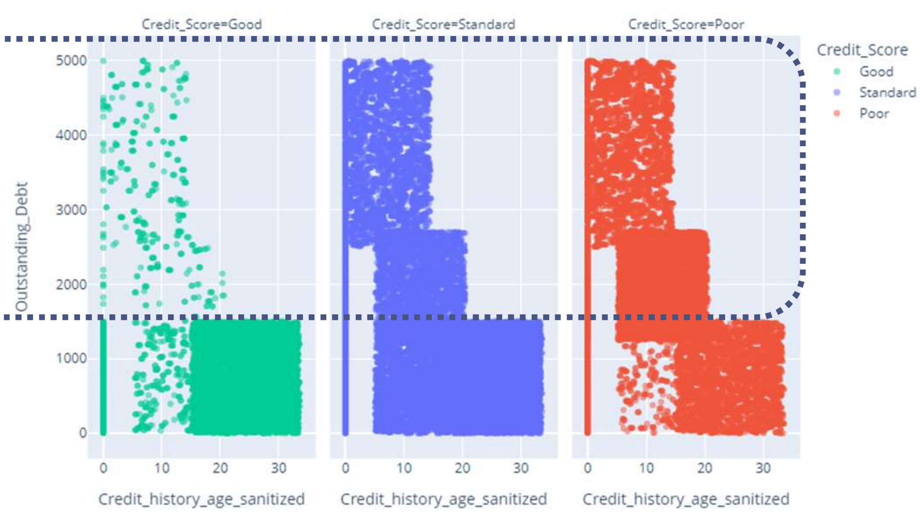
Credit History Age decreases with increase in payment delay

3. Trends in data

Monthly Balance vs Outstanding Debt



Credit History Age vs Outstanding Debt



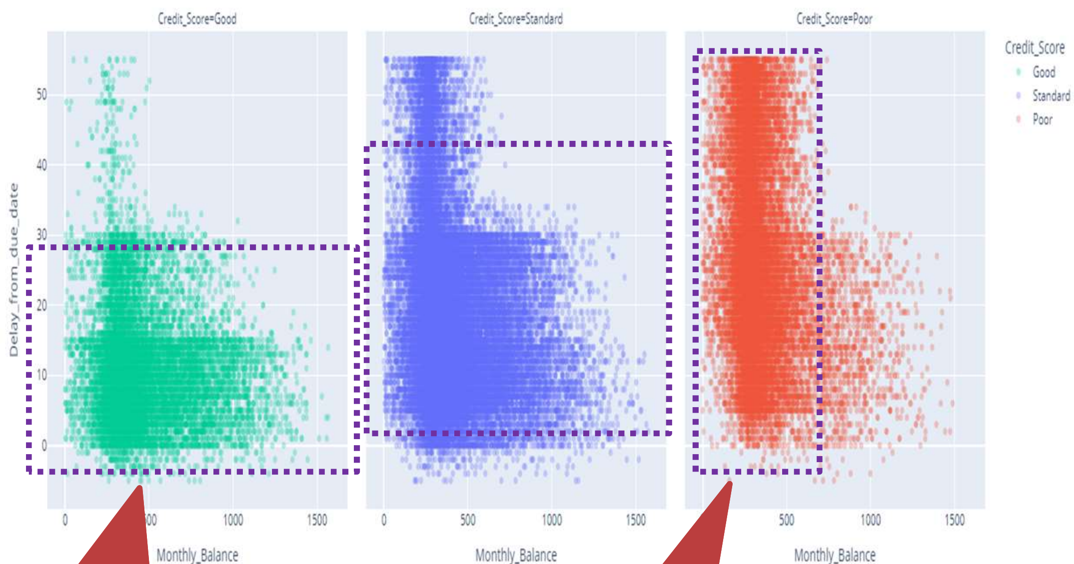
- *Customers in good credit score bucket, usually end up having lower outstanding debts and their monthly account balance is decently split between 0 to 1500 dollars*
- *Customers in the poor credit score bucket has high outstanding debts and have low monthly income*
- *Customers who have a longer credit history end up being good customers and they usually have less outstanding debts*

04

Recommendation

How can you maintain a good credit score?

Maintain a high account balance and avoid delayed payments

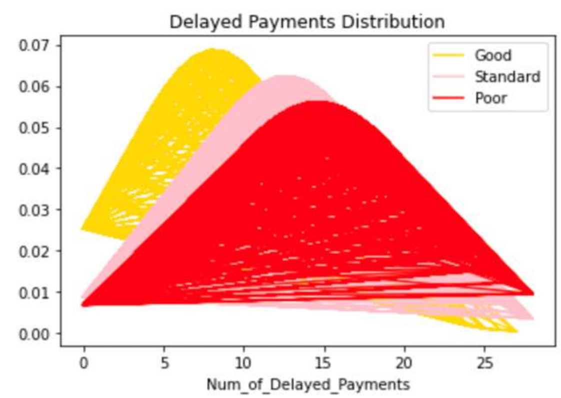


People with good credit score and high monthly balance clear the dues early

People with poor credit score and low monthly balance usually make more delays in payments

Avoid more than 8 delayed payments per month

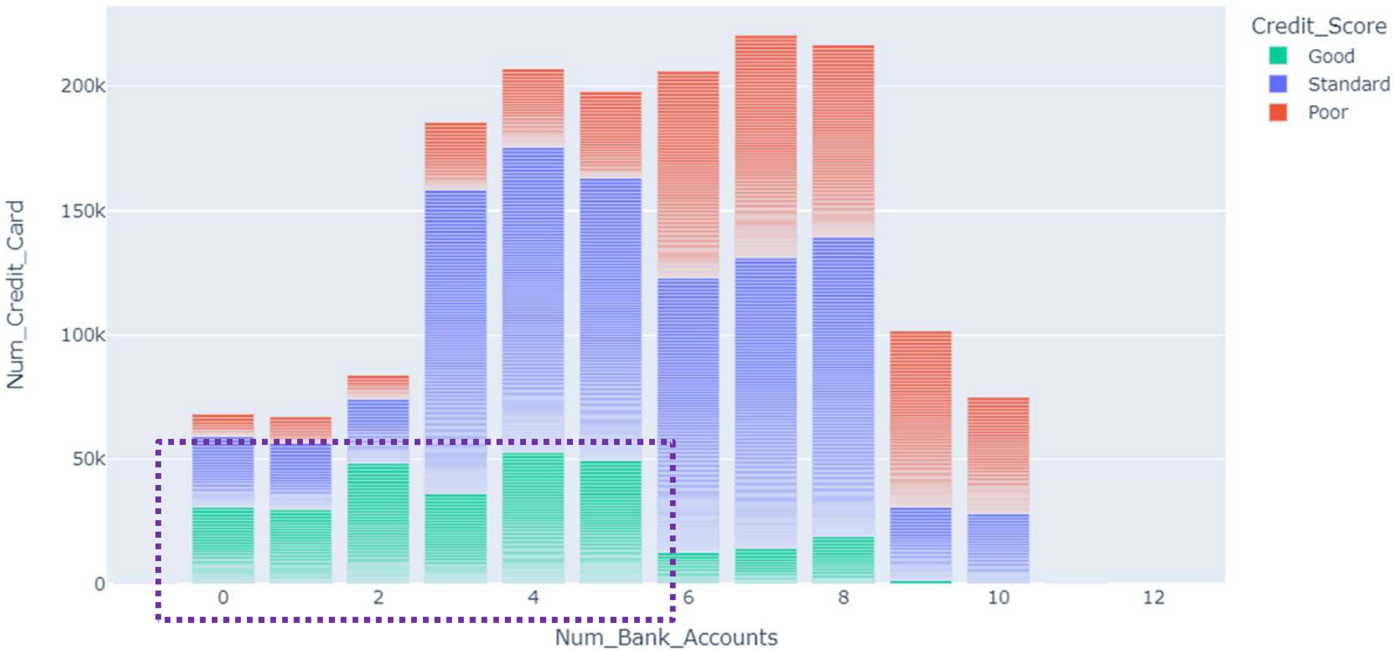
Credit Score	Avg. No. of Delayed Payments	Average Outstanding Debt
Good	8	792.97
Standard	12	1230.82
Poor	14	2021.45



The distribution of customers for good, standard and poor credit score shows that less than 8 defaults would be a safe spot

How can you maintain a good credit score?

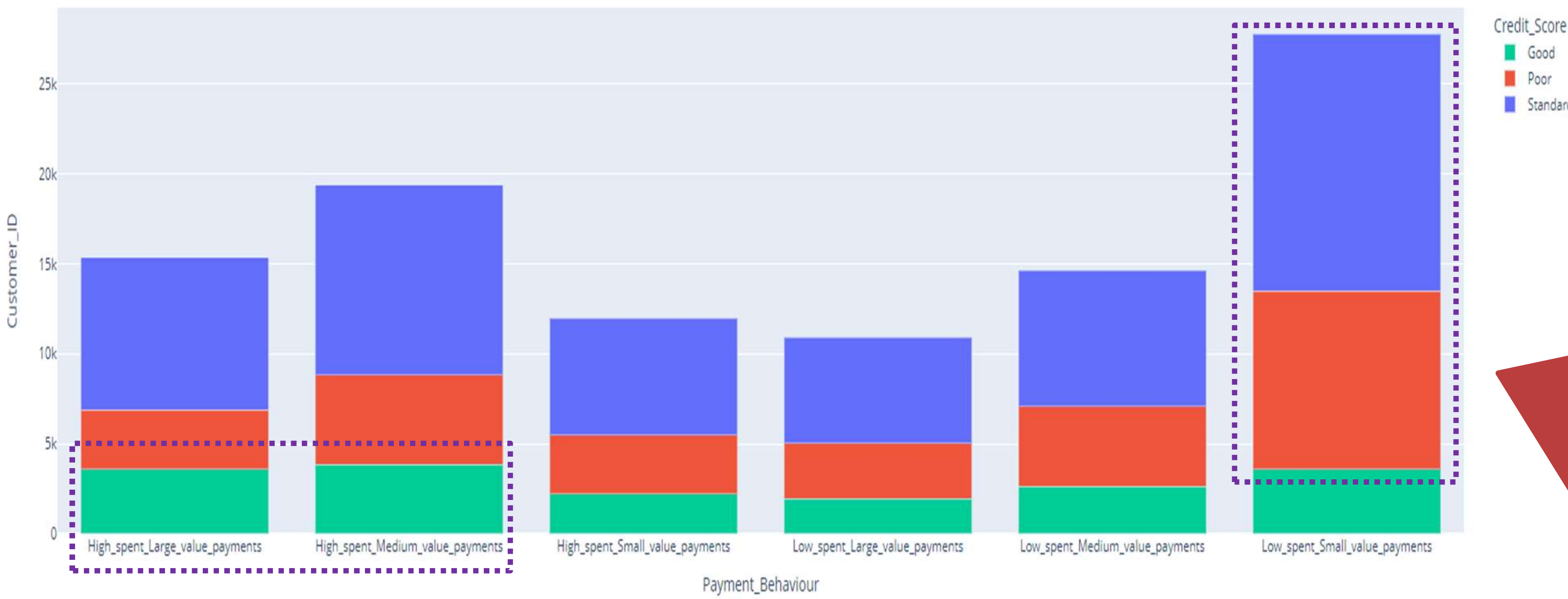
Avoid having multiple bank accounts.



Having multiple bank accounts is not a negative characteristic, however, one must ensure not to have more than 5 accounts. Most of the customers with good credit score have less than 5 bank accounts

How can you maintain a good credit score?

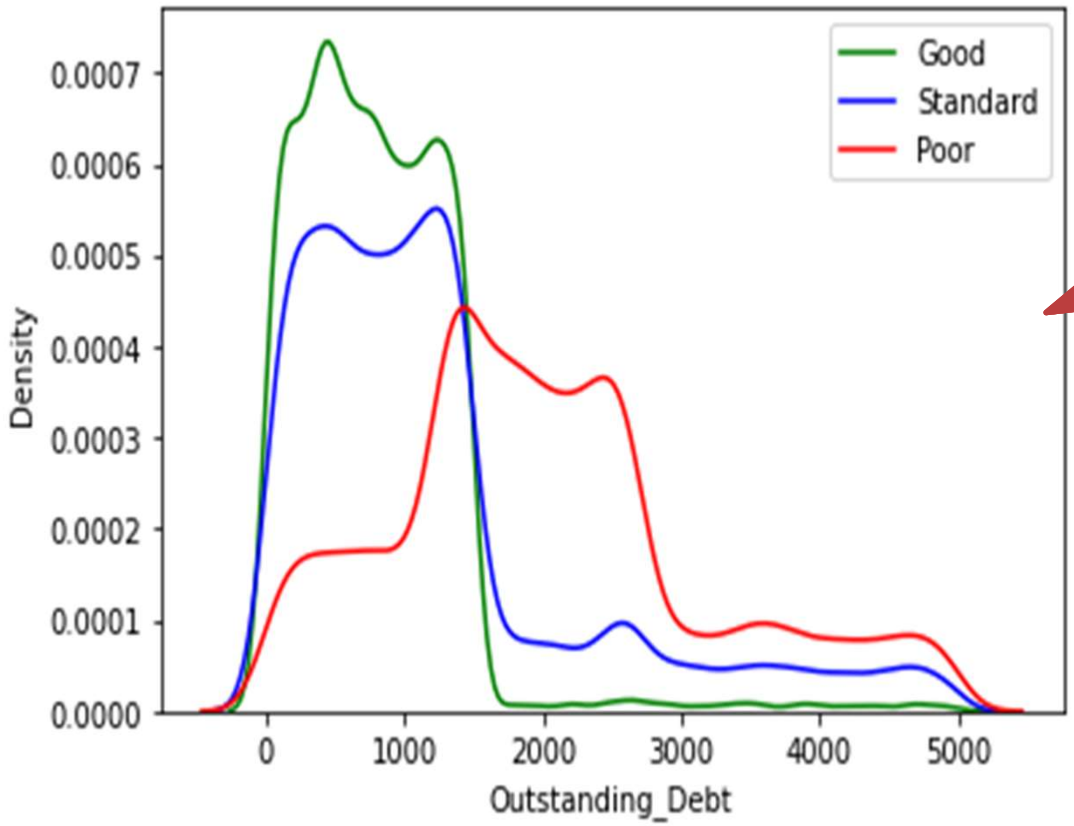
Prioritize increasing the number of spends with large or medium dollar values and avoid payments with low small dollar values



High-spent-large-value and High-spent-medium-value customers usually tend to have a good credit score. Whereas customers that make Low-spent-medium-value and Low-spent-small-value payments tend to have poorer credit scores

How can you maintain a good credit score?

Maintain less than $ 1,500 outstanding debts to secure a good spot in good and standard credit score bucket



From the graph we can infer – After an outstanding debt of $1500 there is a sharp decline in good and standard credit scores.

Thank You... And Remember!

- Maintain a high account balance
- Avoid delayed payments
- Avoid more than 5 bank accounts
- Prioritize large or medium dollar values spends
- Maintain less than $1,500 outstanding debt