Milestone 1 - Credit Score Classification

CSE 4/587 Data Intensive Computing

Lakshya Rawal, lakshyar, 50459636

Mamatha Yarramaneni, mamathay, 50466676

1. a) Discuss the background of the problem leading to your objectives. Why is it a significant problem?

Credit score classification is the process of assigning a numerical value to an individual's creditworthiness based on their financial history. The problem with credit score classification is that it relies on various factors that are not always within the control of the individual, such as economic conditions, unexpected life events, and inaccuracies in credit reports. Additionally, the classification system can be biased, as it may favor certain groups of people over others based on socioeconomic or demographic factors. These issues can lead to unfair credit decisions and limited access to credit, which can have negative impacts on individuals' financial stability and opportunities.

In this project we are working a dataset from a global finance company. Over the years, the company has collected basic bank details and gathered a lot of credit-related information. This model can be used by financial institutions who want to build an intelligent system to segregate the people into credit score brackets.

b. Explain the potential of your project to contribute to your problem domain. Discuss why this contribution is crucial?

Our project can contribute significantly to the problem of credit score classification by helping to reduce bias and improve accuracy in credit decisions. By training models on large datasets of historical credit information, classification algorithms can identify patterns and insights that may not be immediately apparent to human analysts. These models can then be used to predict credit risk more accurately and objectively, taking into account a wider range of factors that may impact an individual's creditworthiness.

One important way classification algorithm can improve credit score classification is by reducing bias. Traditional credit scoring methods may incorporate biases based on factors such as race, gender, or zip code, which can lead to unfair and discriminatory credit decisions. Machine learning models, however, can be designed to identify and remove these biases, resulting in more equitable credit decisions.

In addition to reducing bias, classification algorithms can also improve the accuracy of credit scoring models. By analyzing large volumes of data and identifying correlations and patterns that may not be immediately apparent to humans, algorithms can create more robust and predictive models. This can result in more accurate credit decisions, which can benefit both lenders and borrowers.

By doing so, it can promote more equitable access to credit and help individuals achieve financial stability and opportunities that might otherwise be limited.

2. Data Sources: Collect your data. Your data can come from multiple sources.

Data Source: https://www.kaggle.com/datasets/parisrohan/credit-score-classification

Data Shape: 100,000 Rows and 28 Columns

Data Definition

Column Number	Column Name	Column Definition
1	ID	Represents a unique identification of an entry
2	Customer_ID	Represents a unique identification of a person
3	Month	Represents the month of the year
4	Name	Represents the name of a person
5	Age	Represents the age of the person
6	SSN	Represents the social security number of a person
7	Occupation	Represents the occupation of the person
8	Annual_Income	Represents the annual income of the person
9	Monthly_Inhand_Salary	Represents the monthly base salary of a person
10	Num_Bank_Accounts	Represents the number of bank accounts a person holds
11	Num_Credit_Card	Represents the number of other credit cards held by a person
12	Interest_Rate	Represents the interest rate on credit card
13	Num_of_Loan	Represents the number of loans taken from the bank
14	Type_of_Loan	Represents the types of loan taken by a person
15	Delay_from_due_date	Represents the average number of days delayed from the payment date
16	Num_of_Delayed_Payment	Represents the average number of payments delayed by a person
17	Changed_Credit_Limit	Represents the percentage change in credit card limit
18	Num_Credit_Inquiries	Represents the number of credit card inquiries

Column Number	Column Name	Column Definition
19	Credit_Mix	Represents the classification of the mix of credits
20	Outstanding_Debt	Represents the remaining debt to be paid (in USD)
21	Credit_Utilization_Ratio	Represents the utilization ratio of credit card
22	Credit_History_Age	Represents the age of credit history of the person
23	Payment_of_Min_Amount	Represents whether only the minimum amount was paid by the person
24	Total_EMI_per_month	Represents the monthly EMI payments (in USD)
25	Amount_invested_monthly	Represents the monthly amount invested by the customer (in USD)
26	Payment_Behaviour	Represents the payment behavior of the customer (in USD)
27	Monthly_Balance	Represents the monthly balance amount of the customer (in USD)
28	Credit_Score	Represents the bracket of credit score (Poor, Standard, Good)

3. Data Cleaning/Processing and 4. Exploratory Data Analysis (EDA):

Importing Necessary Libraries

In [154…
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import missingno
from sklearn.impute import KNNImputer
```

Taking Input Data

In [154…
```
df=pd.read_csv('credit-score.csv')
```

```
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:332
6: DtypeWarning: Columns (26) have mixed types.Specify dtype option on impo
rt or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Doing Basic Exploration on Data

In [154…
```
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
```

Checking for null values

In [154…

```
df.isna().sum()
```

Out[1543]:
```
ID                             0
Customer_ID                    0
Month                          0
Name                        9985
Age                            0
SSN                            0
Occupation                     0
Annual_Income                  0
Monthly_Inhand_Salary      15002
Num_Bank_Accounts              0
Num_Credit_Card                0
Interest_Rate                  0
Num_of_Loan                    0
Type_of_Loan               11408
Delay_from_due_date            0
Num_of_Delayed_Payment      7002
Changed_Credit_Limit           0
Num_Credit_Inquiries        1965
Credit_Mix                     0
Outstanding_Debt               0
Credit_Utilization_Ratio       0
Credit_History_Age          9030
Payment_of_Min_Amount          0
Total_EMI_per_month            0
Amount_invested_monthly     4479
Payment_Behaviour              0
Monthly_Balance             1200
Credit_Score                   0
dtype: int64
```

Seeing If we have a lot of Variance in our data

In [154…
```python
df.var()
```

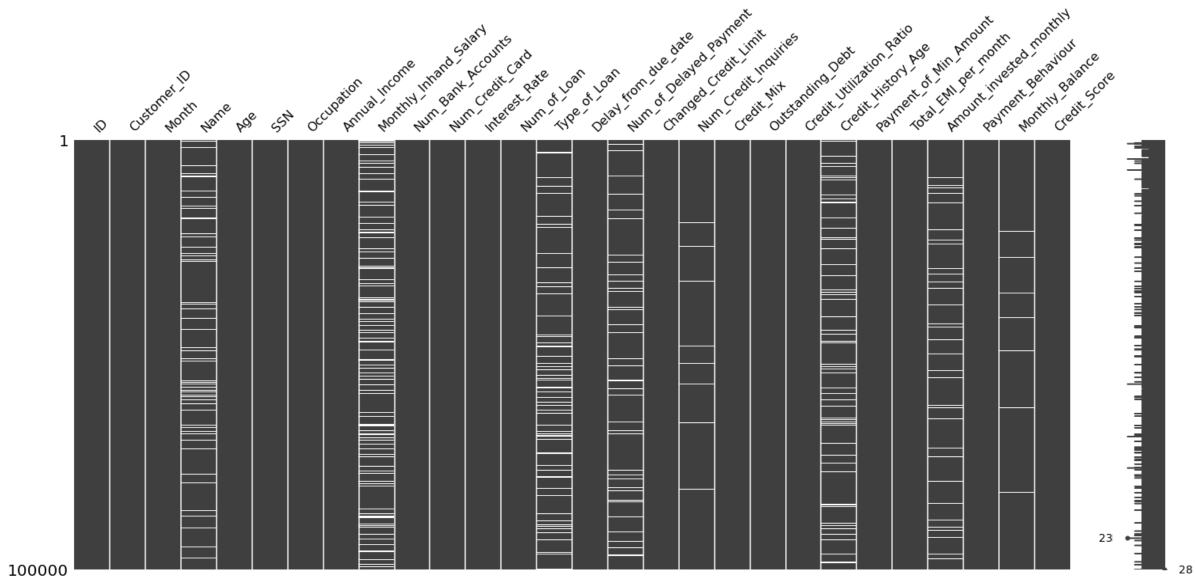
```
<ipython-input-1544-28ded241fd7c>:1: FutureWarning: Dropping of nuisance co
lumns in DataFrame reductions (with 'numeric_only=None') is deprecated; in
a future version this will raise TypeError.  Select only valid columns befo
re calling the reduction.
  df.var()
```

Out[1544]:
```
Monthly_Inhand_Salary       1.013586e+07
Num_Bank_Accounts           1.378390e+04
Num_Credit_Card             1.665582e+04
Interest_Rate               2.175501e+05
Delay_from_due_date         2.208227e+02
Num_Credit_Inquiries        3.731748e+04
Credit_Utilization_Ratio    2.618241e+01
Total_EMI_per_month         6.899032e+07
dtype: float64
```

Plotting the missing values

In [154…
```python
missingno.matrix(df)
```

Out[1545]: <AxesSubplot:>



Helper functions

In [154…
```
def fill_median(df, column):
  median_value_1 = df[column].median()
  df[column].fillna(value=median_value_1, inplace=True)
```

In [154…
```
def fill_mean(df, column):
  mean_value_1 = df[column].mean()
  df[column].fillna(value=mean_value_1, inplace=True)
```

In [154…
```
def box_plot(df, x, y, title_):
  plt.figure(figsize= (12,8))
  sns.boxplot(x= df[x], y= df[y])
  plt.title(title_, size = 15)
  plt.show()
```

In [154…
```
imputer = KNNImputer(n_neighbors=3)

def fill_na(df, column, type_=None):
    if type_ == "num":
        df[column] = imputer.fit_transform(df[column].values.reshape(-1, 1))
    else:
      if df[column][0] == None:
        df[column].fillna(method='bfill', inplace=True)
      else:
        df[column].fillna(method='ffill', inplace=True)
    return df[column]
```

In [155…
```
def handle_outliers_numericals(df, numerical_cols):
  for x in list(numerical_cols):
    q75,q25 = np.percentile(df.loc[:,x],[75,25])
    intr_qr = q75-q25

    max = q75+(1.5*intr_qr)
    min = q25-(1.5*intr_qr)
```

```
    df.loc[df[x] < min,x] = np.nan
    df.loc[df[x] > max,x] = np.nan
```

Drop unnecessary columns like SSN, Name that do not impact the model

In [155…
```
df.drop(columns=['SSN', 'Name'], axis=1, inplace=True)
```

Dropping columns that have too many categorical values that will not impact our model

In [155…
```
df.drop(columns=['Interest_Rate', 'Type_of_Loan'], axis=1, inplace=True)
```

Preprocessing each column

Column: Age

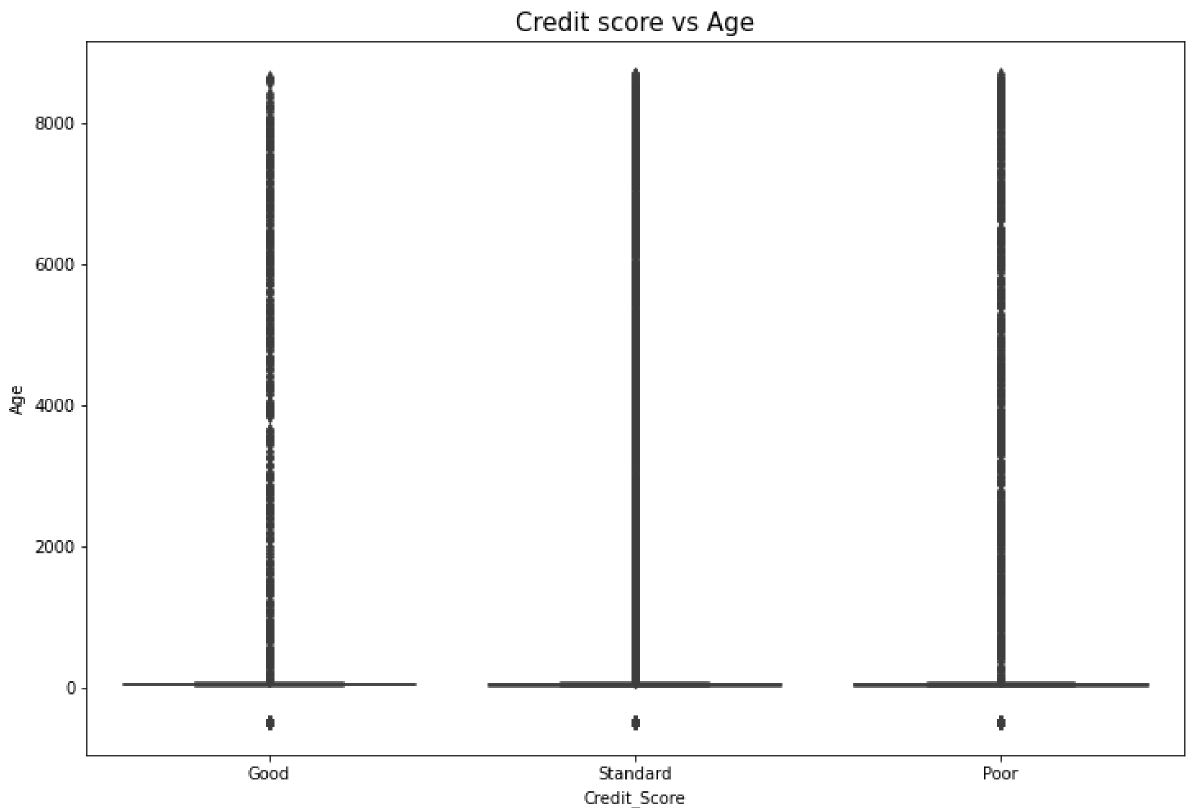
In [155…
```
df['Age'].dtype
```

Out[1553]:
```
dtype('O')
```

Column Age as an object data type does not make sense, so converting into numeric

In [155…
```
#remove unnecessary underscores
df['Age'] = df['Age'].str.strip('_')

#convert object to float data type
df['Age'] = df['Age'].astype('int64')
```

In [155…
```
#box plot
box_plot(df, 'Credit_Score', 'Age', 'Credit score vs Age')
```



Since the data has Outliers we will remove them for all numerical variables at once below

Column: Occupation

In [155…
```
df['Occupation'].unique()
```

Out[1556]:
```
array(['Scientist', '_______', 'Teacher', 'Engineer', 'Entrepreneur',
       'Developer', 'Lawyer', 'Media_Manager', 'Doctor', 'Journalist',
       'Manager', 'Accountant', 'Musician', 'Mechanic', 'Writer',
       'Architect'], dtype=object)
```

In [155…
```
df['Occupation'].value_counts()
```

Out[1557]:
```
_______          7062
Lawyer           6575
Architect        6355
Engineer         6350
Scientist        6299
Mechanic         6291
Accountant       6271
Developer        6235
Media_Manager    6232
Teacher          6215
Entrepreneur     6174
Doctor           6087
Journalist       6085
Manager          5973
Musician         5911
Writer           5885
Name: Occupation, dtype: int64
```

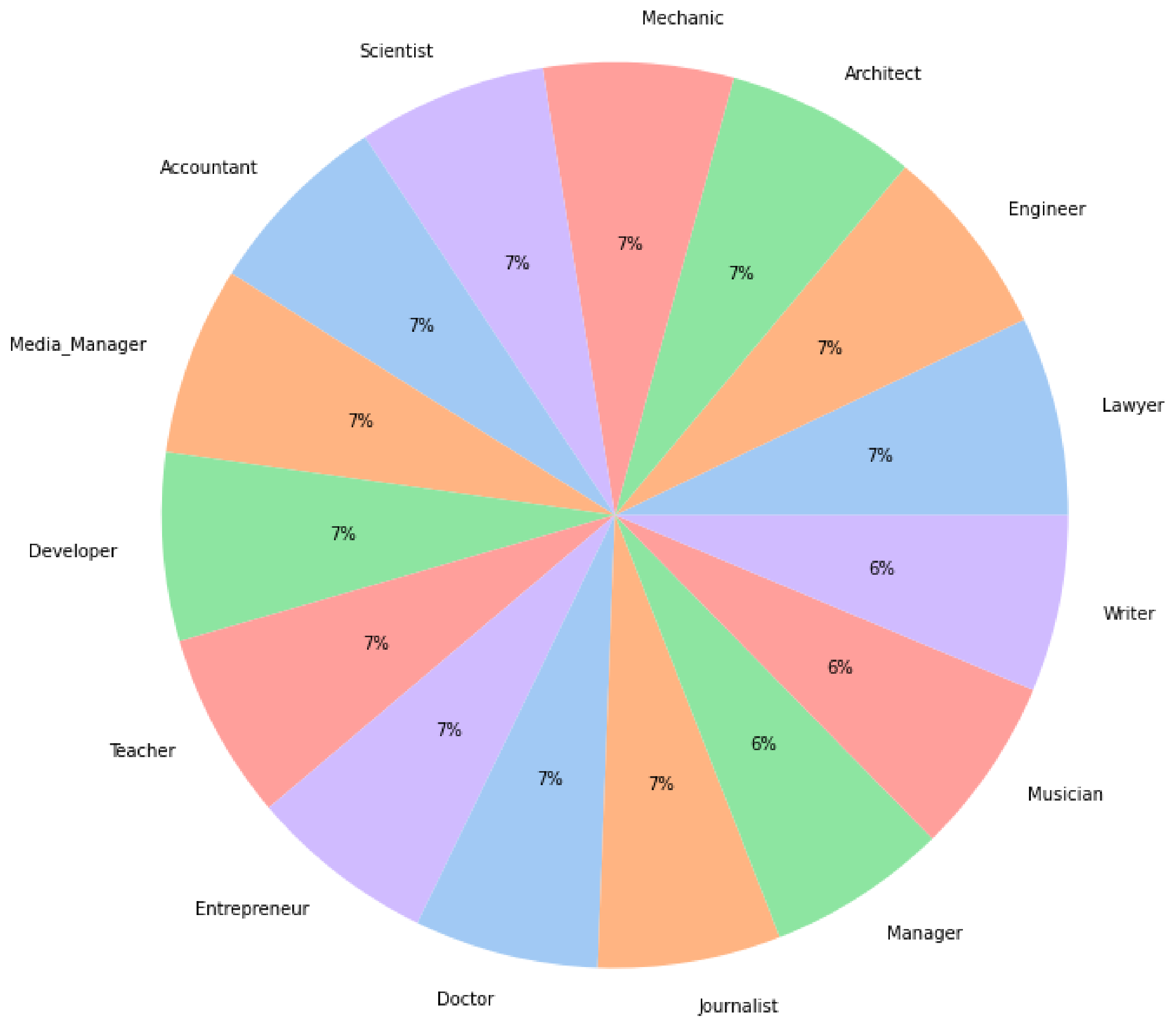
Filling '_______' value with None

In [155…
```
df.loc[df['Occupation'] == '_______', 'Occupation'] = None
```

In [155…
```
fill_na(df, 'Occupation')
```

Out[1559]:
```
0         Scientist
1         Scientist
2         Scientist
3         Scientist
4         Scientist
            ...
99995      Mechanic
99996      Mechanic
99997      Mechanic
99998      Mechanic
99999      Mechanic
Name: Occupation, Length: 100000, dtype: object
```

In [156…
```
#intializing seaborn color palette
colors = sns.color_palette('pastel')[0:5]

#create pie chart
plt.figure(figsize= (12,12))
plt.pie(df['Occupation'].value_counts(dropna = False).values, labels = df['O
plt.show()
```



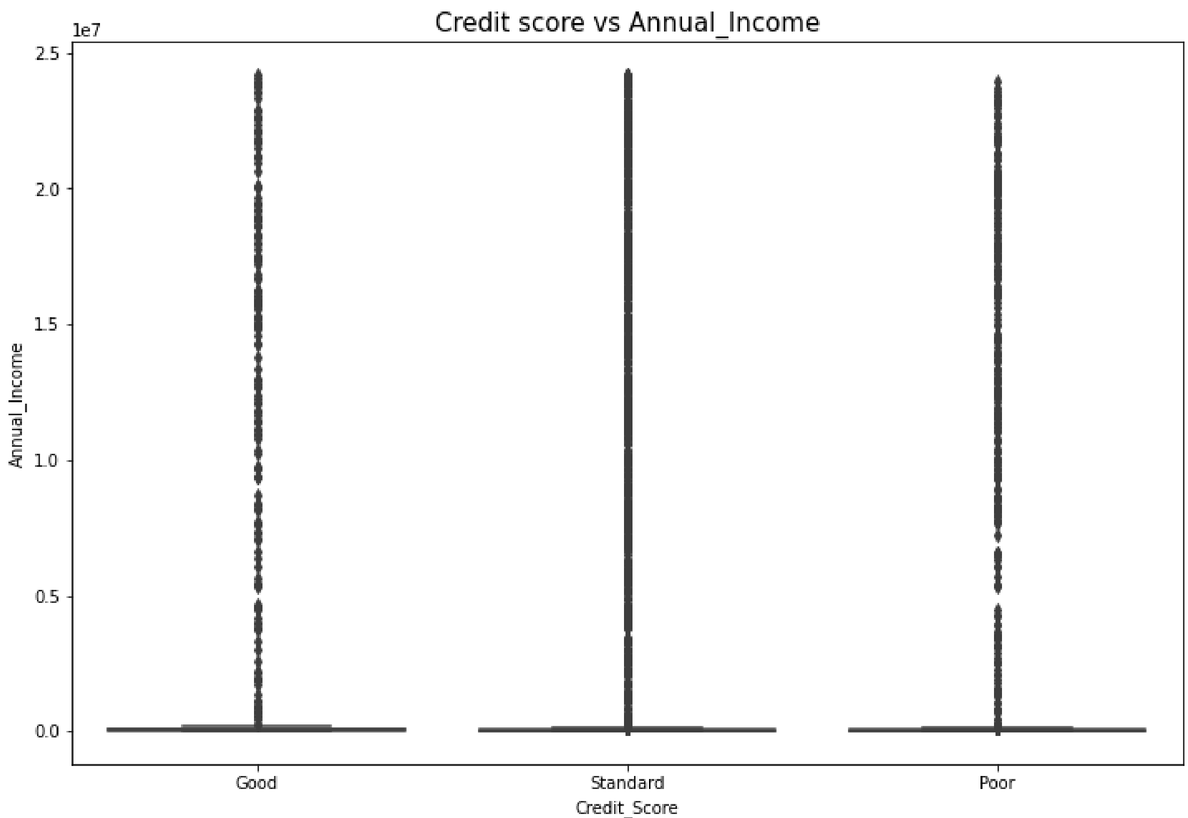
Column: Annual_Income

In [156…
```
df['Annual_Income'].dtype
```

Out[1561]:
```
dtype('O')
```

Annual_Income as an object data type does not make sense, so converting it into numeric

In [156…
```
#remove unnecessary underscores
df['Annual_Income'] = df['Annual_Income'].str.strip('_')

#convert object to float data type
df['Annual_Income'] = df['Annual_Income'].astype('float64')

#box plot
box_plot(df, 'Credit_Score', 'Annual_Income', 'Credit score vs Annual_Income
```



Since the data has Outliers we will remove them for all numerical variables at once below

In [156…
```
(df['Annual_Income']> 0).all()
```

Out[1563]: True

Column: Monthly Inhand Salary

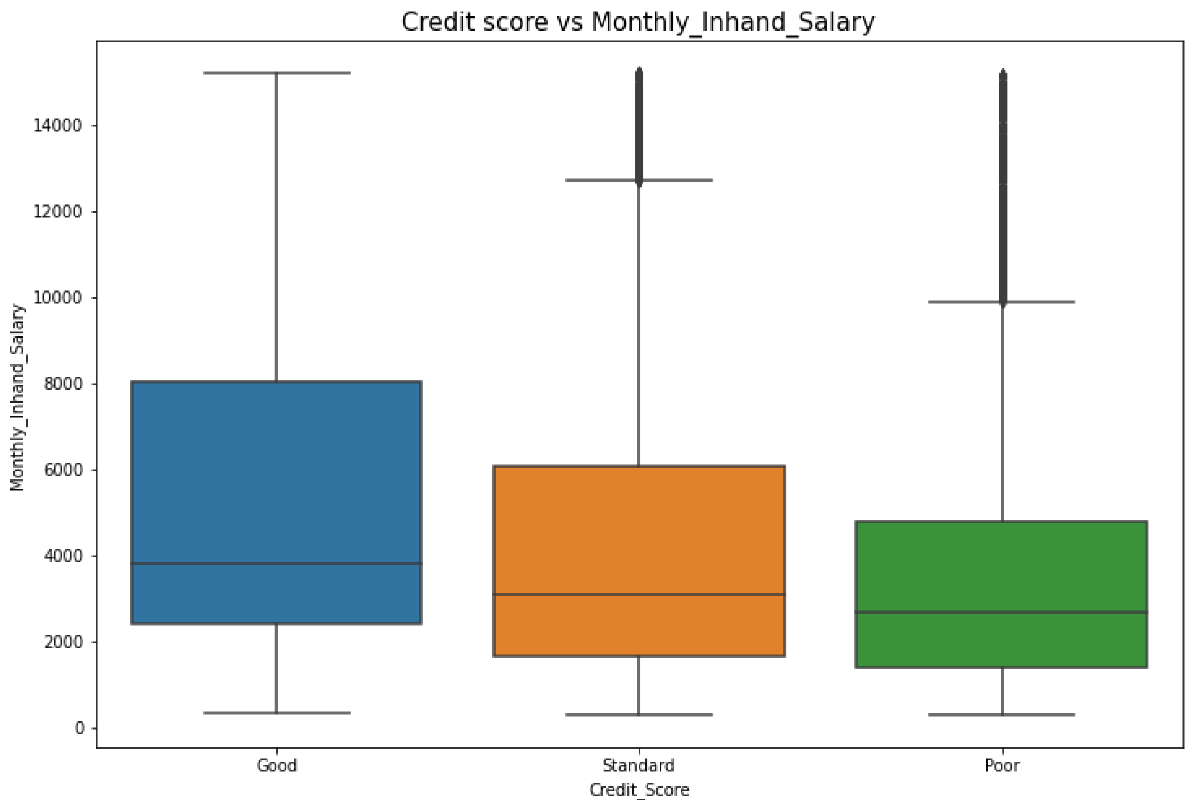
In [156…
```
df['Monthly_Inhand_Salary'].dtype
```

Out[1564]: dtype('float64')

In [156…
```
df['Monthly_Inhand_Salary'].isna().sum()
```

Out[1565]: 15002

In [156…
```
#box plot
box_plot(df, 'Credit_Score', 'Monthly_Inhand_Salary', 'Credit score vs Month
```



Since the data has Outliers we will remove them for all numerical variables at once below

We will also address the null values of all columns through the helper function at once below

Column: Num_Bank_Accounts

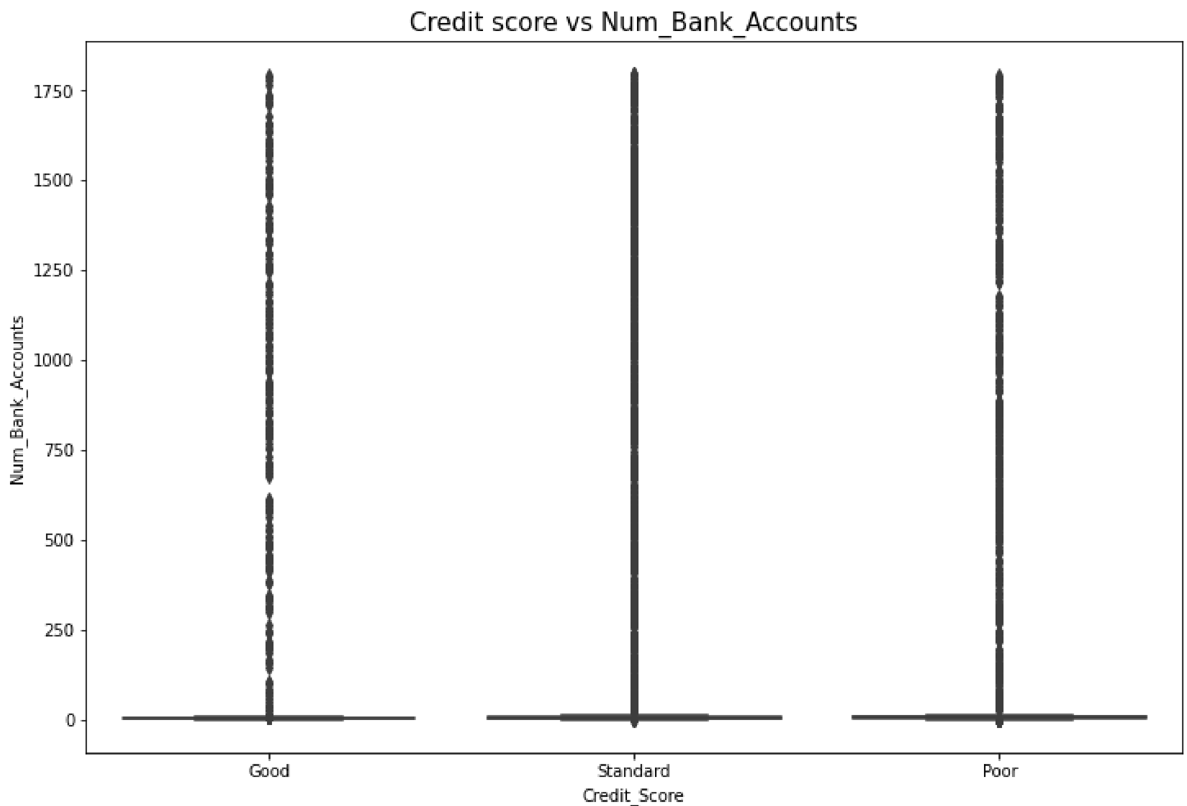
In [156…

```
df['Num_Bank_Accounts'].dtype
```

Out[1567]: dtype('int64')

In [156…

```
box_plot(df, 'Credit_Score', 'Num_Bank_Accounts', 'Credit score vs Num_Bank_
```



In [156…
```
df['Num_Bank_Accounts'].isna().sum()
```

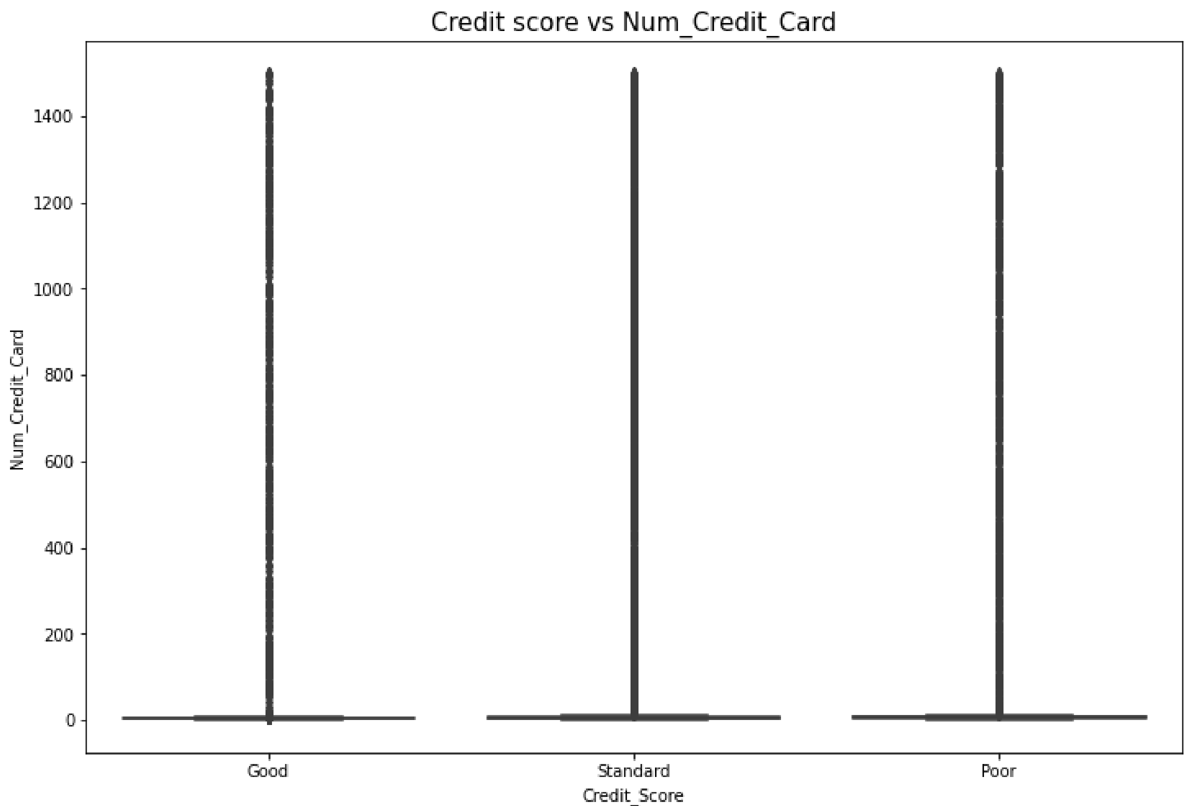
Out[1569]:
```
0
```

Column: Num_Credit_Card

In [157…
```
df['Num_Credit_Card'].dtype
```

Out[1570]:
```
dtype('int64')
```

In [157…
```
box_plot(df, 'Credit_Score', 'Num_Credit_Card', 'Credit score vs Num_Credit_
```



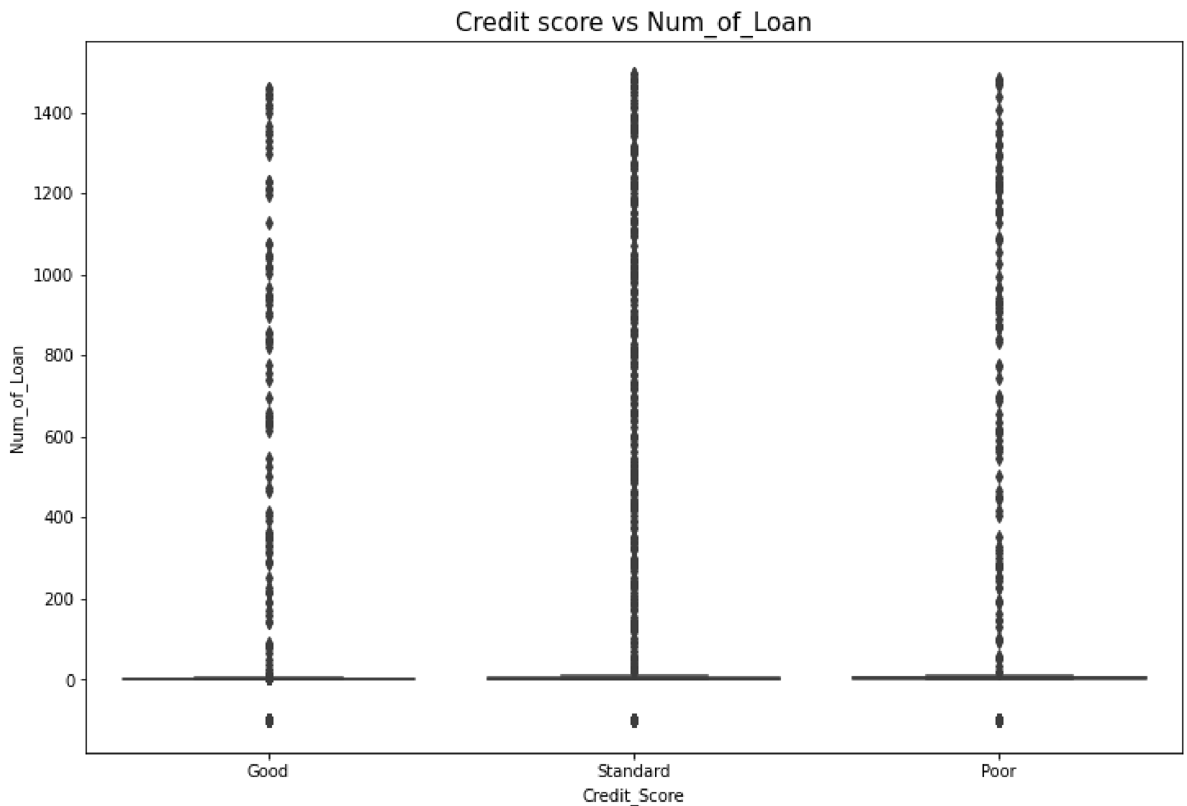
Column: Num_of_Loan

In [157…
```
df['Num_of_Loan'].isna().sum()
```

Out[1572]: 0

In [157…
```
#remove unnecessary underscores
df['Num_of_Loan'] = df['Num_of_Loan'].str.strip('_')

#convert object to float data type
df['Num_of_Loan'] = df['Num_of_Loan'].astype('int64')
```

In [157…
```
box_plot(df, 'Credit_Score', 'Num_of_Loan', 'Credit score vs Num_of_Loan')
```

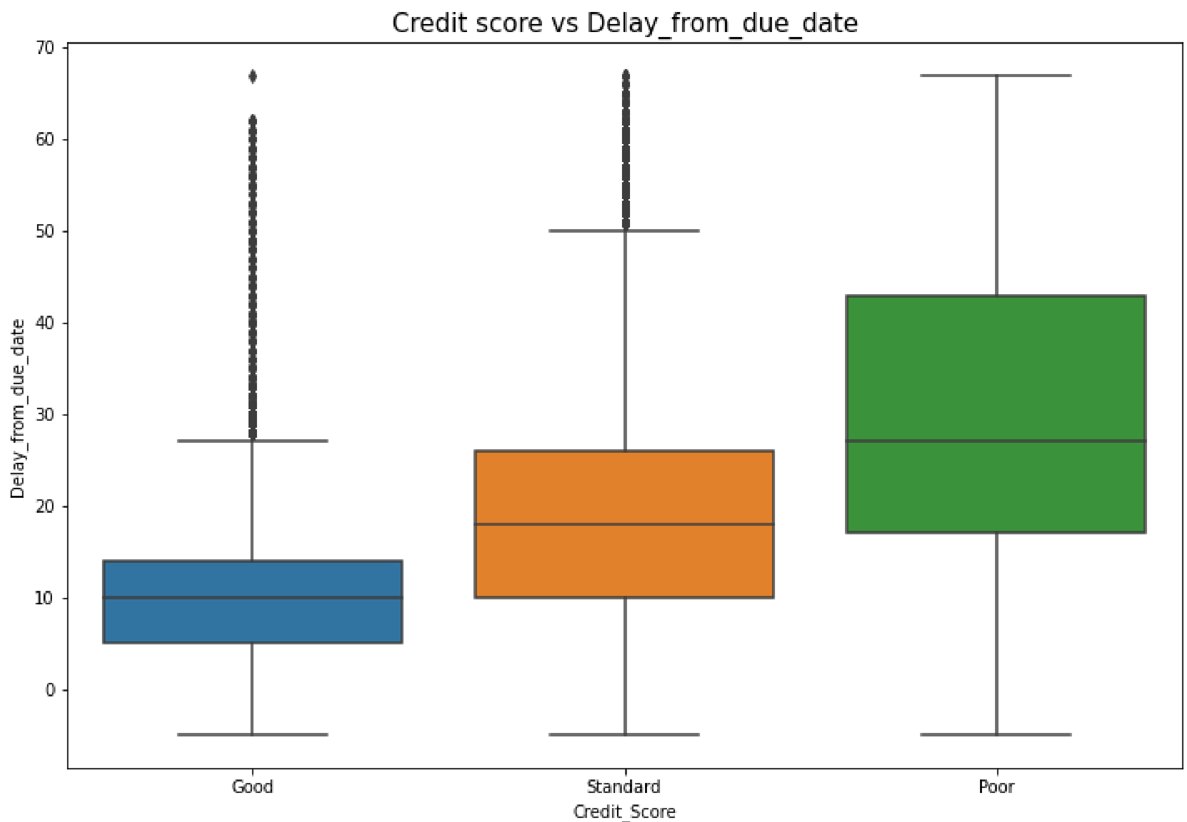


Column: Delay_from_due_date

In [157…
```
df['Delay_from_due_date'].isna().sum()
```

Out[1575]: 0

In [157…
```
box_plot(df, 'Credit_Score', 'Delay_from_due_date', 'Credit score vs Delay_f
```



Column: Num_of_Delayed_Payment

In [157…
```
df['Num_of_Delayed_Payment'].dtype
```

Out[1577]: dtype('O')

In [157…
```
#remove unnecessary underscores
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].str.strip('_')
```

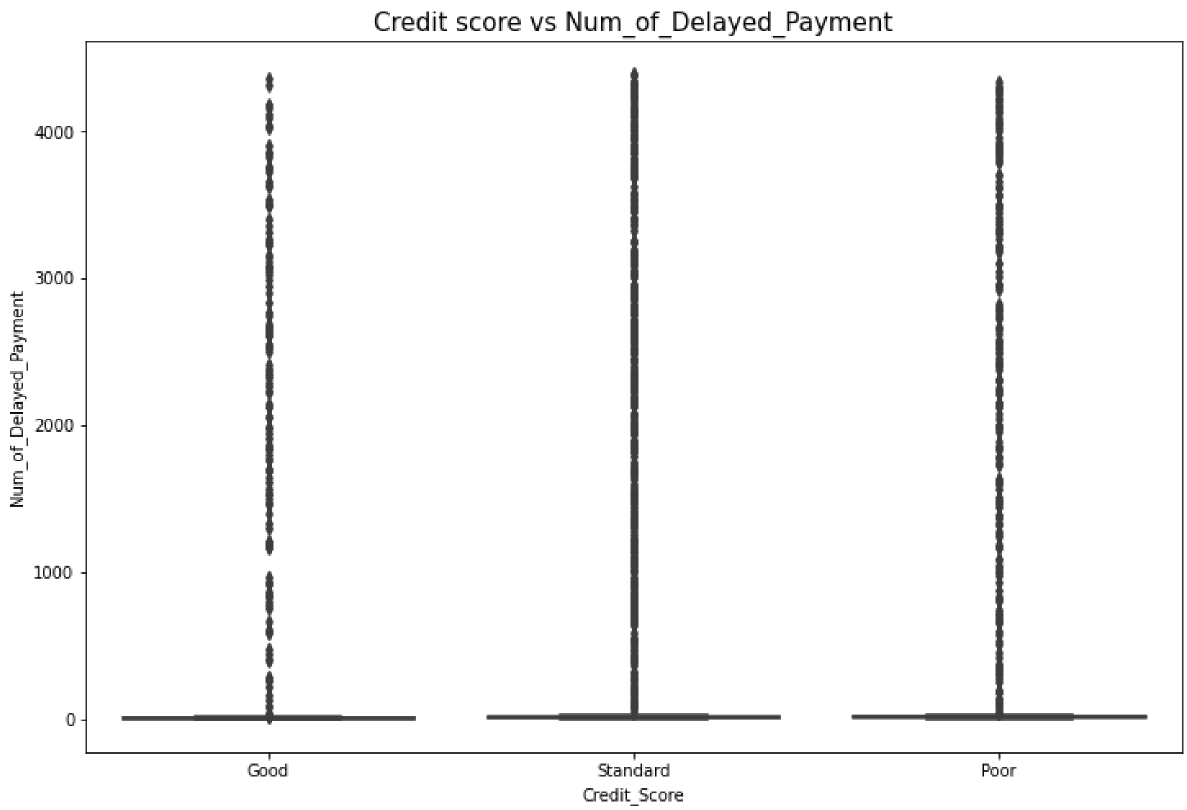
In [157…
```
#Considering that bank has no data for delayed payment by the user and hence
df['Num_of_Delayed_Payment'].fillna(value='0', inplace=True)
```

In [158…
```
df['Num_of_Delayed_Payment'].isna().sum()
```

Out[1580]: 0

In [158…
```
#convert object to float data type
df['Num_of_Delayed_Payment'] = df['Num_of_Delayed_Payment'].astype('int64')
```

In [158…
```
box_plot(df, 'Credit_Score', 'Num_of_Delayed_Payment', 'Credit score vs Num_
```



Column: Changed_Credit_Limit

In [158…
```
df['Changed_Credit_Limit'].dtype
```

Out[1583]:
```
dtype('O')
```

Checking for the most common text values in the numerical data

In [158…
```
df.describe(include="O").T
```

Out[1584]:

	count	unique	top	f
ID	100000	100000	0x1602	
Customer_ID	100000	12500	CUS_0xd40	
Month	100000	8	January	12
Occupation	100000	15	Lawyer	7
Changed_Credit_Limit	100000	4384	_	2
Credit_Mix	100000	4	Standard	36
Outstanding_Debt	100000	13178	1360.45	
Credit_History_Age	90970	404	15 Years and 11 Months	
Payment_of_Min_Amount	100000	3	Yes	52
Amount_invested_monthly	95521	91049	__10000__	4
Payment_Behaviour	100000	7	Low_spent_Small_value_payments	25
Monthly_Balance	98800	98792	__-333333333333333333333333333__	
Credit_Score	100000	3	Standard	53

In [158…

```
#removing '_' and converting into float
df.loc[df['Changed_Credit_Limit'] == '_', 'Changed_Credit_Limit'] = None

df.loc[df['Changed_Credit_Limit'].notnull(), 'Changed_Credit_Limit'] = df.lo

fill_na(df, 'Changed_Credit_Limit')
```

Out[1585]:

```
0        11.27
1        11.27
2        11.27
3         6.27
4        11.27
         ...
99995    11.50
99996    11.50
99997    11.50
99998    11.50
99999    11.50
Name: Changed_Credit_Limit, Length: 100000, dtype: float64
```

In [158…

```
df['Changed_Credit_Limit'].dtype
```

Out[1586]: dtype('float64')

Column: Num_Credit_Inquiries

In [158…

```
df['Num_Credit_Inquiries'].isna().sum()
```

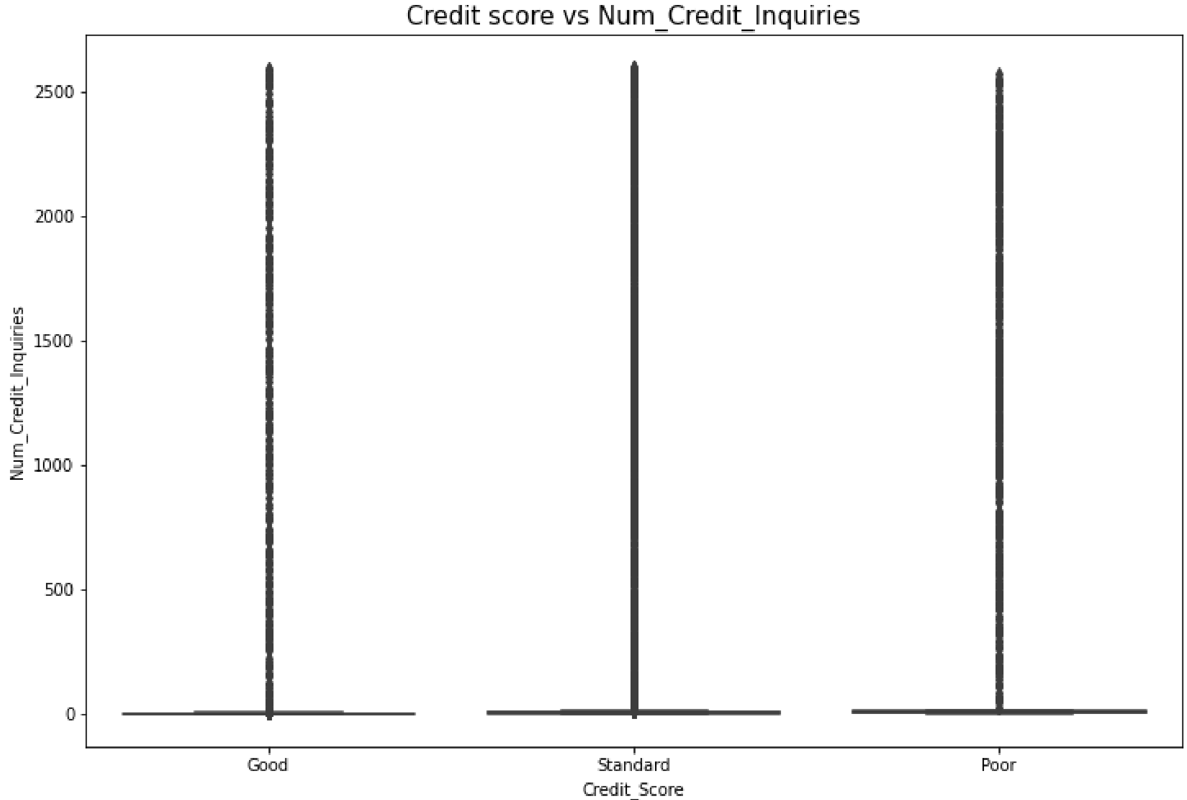
Out[1587]: 1965

In [158…

```
df['Num_Credit_Inquiries'].isna().sum()
```

Out[1588]: 1965

In [158…
```
box_plot(df, 'Credit_Score', 'Num_Credit_Inquiries', 'Credit score vs Num_Cr
```



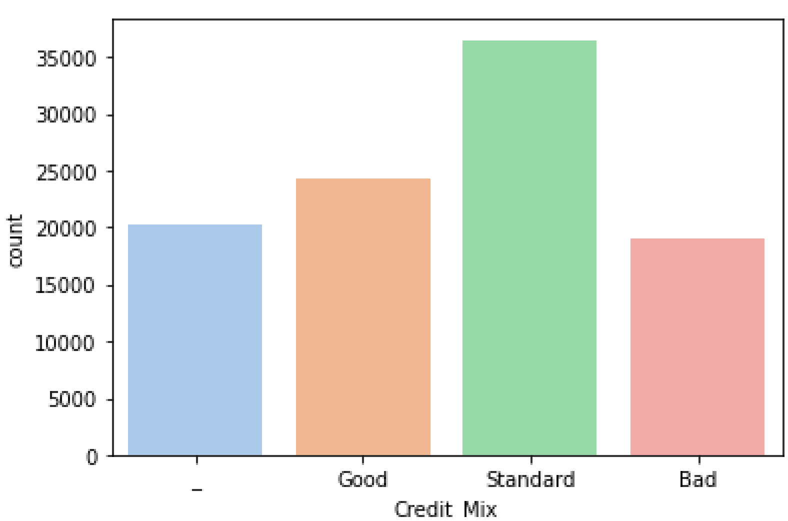
Column: Credit_Mix

In [159…
```
df['Credit_Mix'].unique()
```

Out[1590]: array(['_', 'Good', 'Standard', 'Bad'], dtype=object)

In [159…
```
sns.countplot(data=df, x='Credit_Mix', palette='pastel')
```

Out[1591]: <AxesSubplot:xlabel='Credit_Mix', ylabel='count'>



In [159…
```
# replace '_' with None
df.loc[df['Credit_Mix'] == '_', 'Credit_Mix'] = None
```

In [159…
```
df['Credit_Mix'].unique()
```

Out[1593]:
```
array([None, 'Good', 'Standard', 'Bad'], dtype=object)
```

Column: Outstanding_Debt

In [159…
```
df['Outstanding_Debt'].dtype
```

Out[1594]:
```
dtype('O')
```

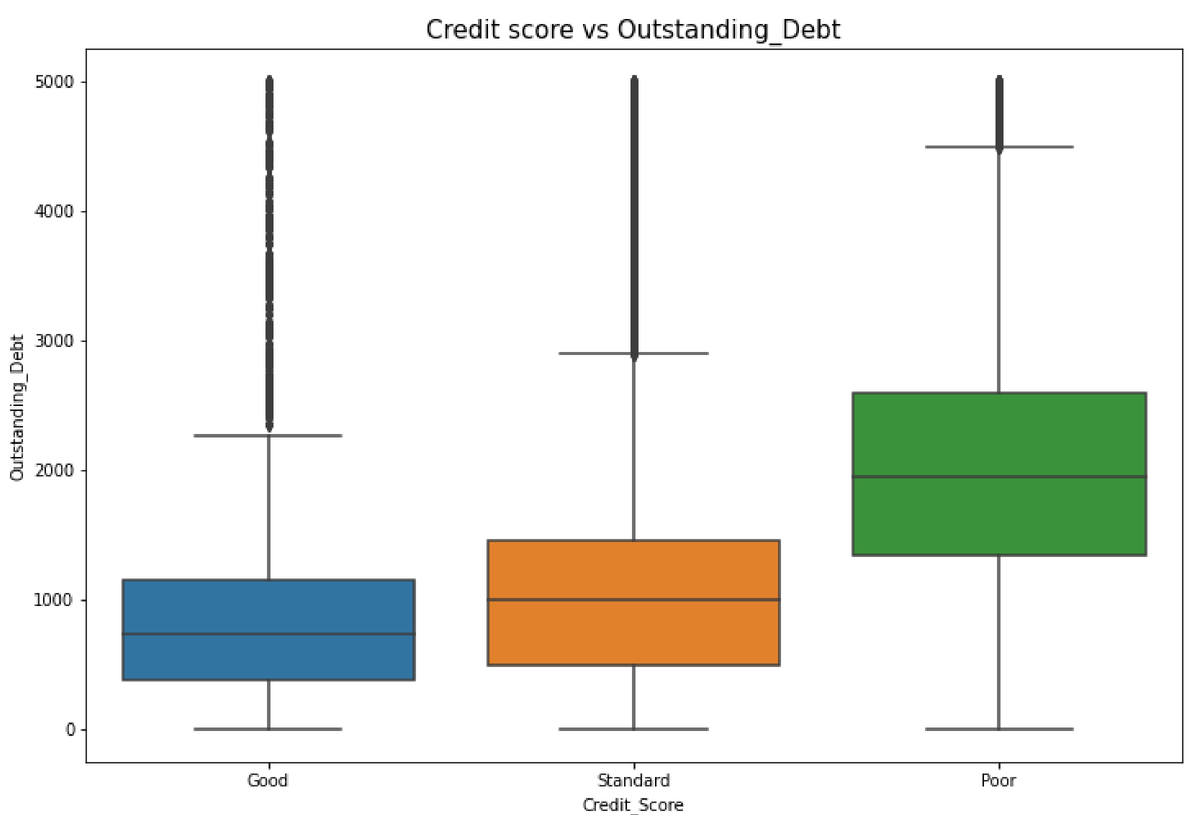
In [159…
```
#remove unnecessary underscores
df['Outstanding_Debt'] = df['Outstanding_Debt'].str.strip('_')
```

In [159…
```
df['Outstanding_Debt'].isna().sum()
```

Out[1596]:
```
0
```

In [159…
```
#convert object to float data type
df['Outstanding_Debt'] = df['Outstanding_Debt'].astype('float64')
```

In [159…
```
box_plot(df, 'Credit_Score', 'Outstanding_Debt', 'Credit score vs Outstandin
```



Column: Credit_Utilization_Ratio

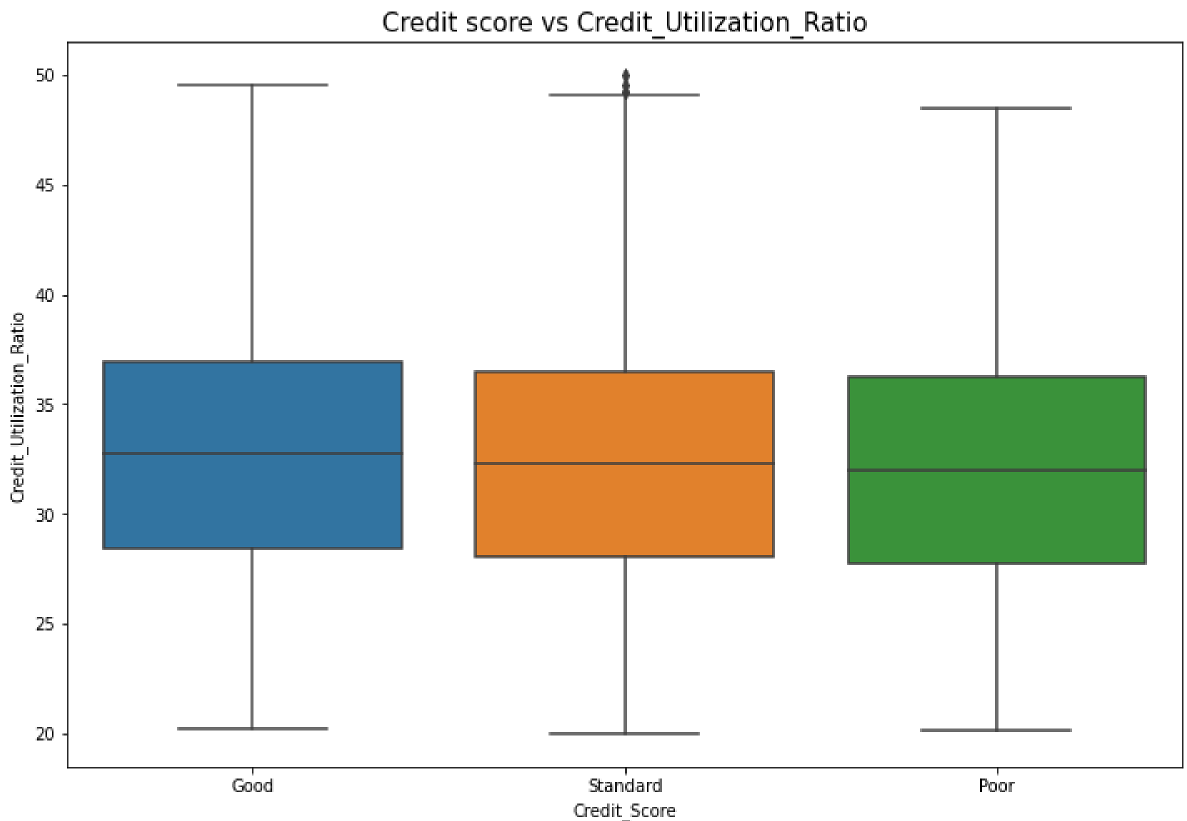
In [159…
```
df['Credit_Utilization_Ratio'].dtype
```

Out[1599]: dtype('float64')

In [160…
```
df['Credit_Utilization_Ratio'].isna().sum()
```

Out[1600]: 0

In [160…
```
box_plot(df, 'Credit_Score', 'Credit_Utilization_Ratio', 'Credit score vs Cr
```



Column: Credit_History_Age

In [160…
```
df['Credit_History_Age'].head()
```

Out[1602]:
```
0    22 Years and 1 Months
1                      NaN
2    22 Years and 3 Months
3    22 Years and 4 Months
4    22 Years and 5 Months
Name: Credit_History_Age, dtype: object
```

Converting the column to integer by stripping the first value in the string and ignoring the monthly value

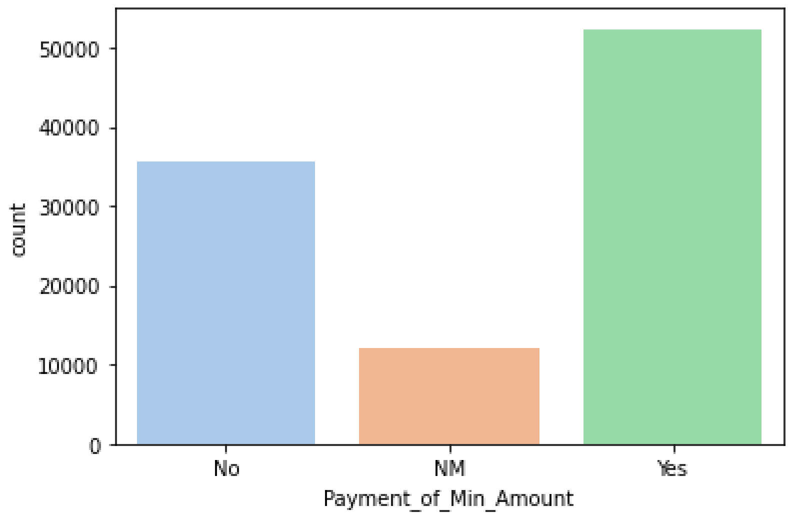
In [160…
```
new_credit = []
for i in df['Credit_History_Age']:
    new_credit.append(str(i).split(' ')[0])

df['Credit_History_Age'] = new_credit
```

```
df['Credit_History_Age'] = df['Credit_History_Age'].replace({'nan':np.nan})
df['Credit_History_Age'] = df['Credit_History_Age'].astype('float64')
```

Column: Payment_of_Min_Amount

In [160…
```
sns.countplot(data=df, x='Payment_of_Min_Amount', palette='pastel')
```

Out[1604]: <AxesSubplot:xlabel='Payment_of_Min_Amount', ylabel='count'>



In [160…
```
df.loc[df['Payment_of_Min_Amount'] == 'NM', 'Payment_of_Min_Amount'] = None
```

In [160…
```
df['Payment_of_Min_Amount'].fillna(value='No', inplace=True)
```

In [160…
```
df['Payment_of_Min_Amount'].unique()
```

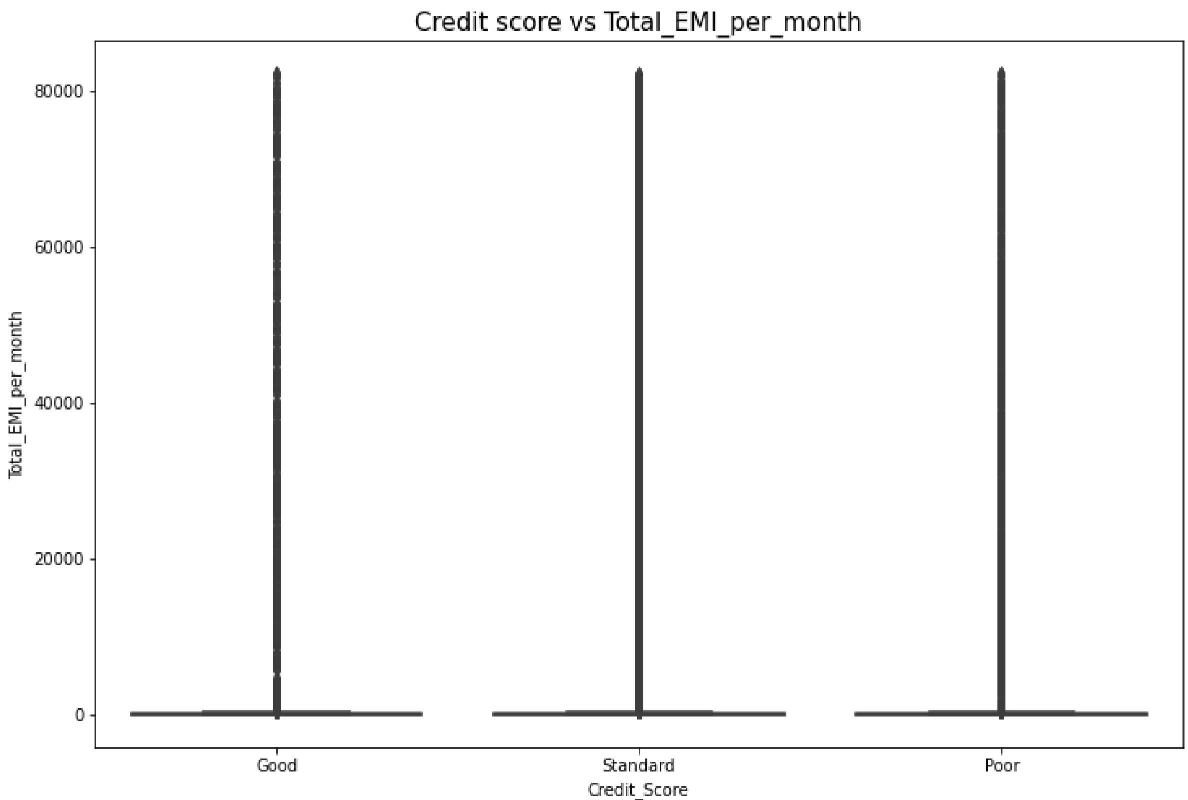
Out[1607]: array(['No', 'Yes'], dtype=object)

Column: Total_EMI_per_month

In [160…
```
df['Total_EMI_per_month'].dtype
```

Out[1608]: dtype('float64')

In [160…
```
box_plot(df, 'Credit_Score', 'Total_EMI_per_month', 'Credit score vs Total_E
```



Since there are no NA values and the data type is float we will handle the outliers in the end along with all the other numerical columns

Column: Amount_invested_monthly

In [161…
```
df['Amount_invested_monthly'].dtype
```

Out[1610]:
```
dtype('O')
```

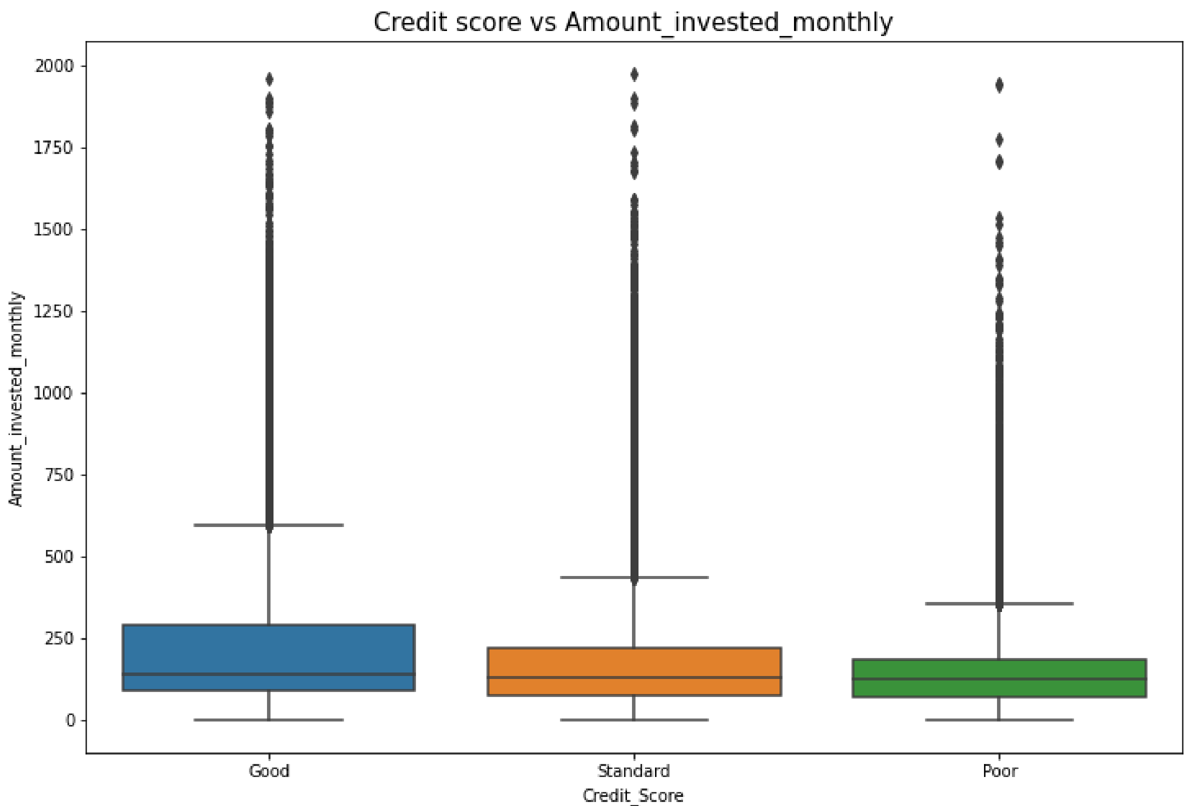
In [161…
```
df.loc[df['Amount_invested_monthly'] == '__10000__', 'Amount_invested_monthl
```

In [161…
```
#convert object to float data type
df['Amount_invested_monthly'] = df['Amount_invested_monthly'].astype('float6
```

Handling the Null Values as Median

In [161…
```
fill_median(df, 'Amount_invested_monthly')
```

In [161…
```
box_plot(df, 'Credit_Score', 'Amount_invested_monthly', 'Credit score vs Amo
```



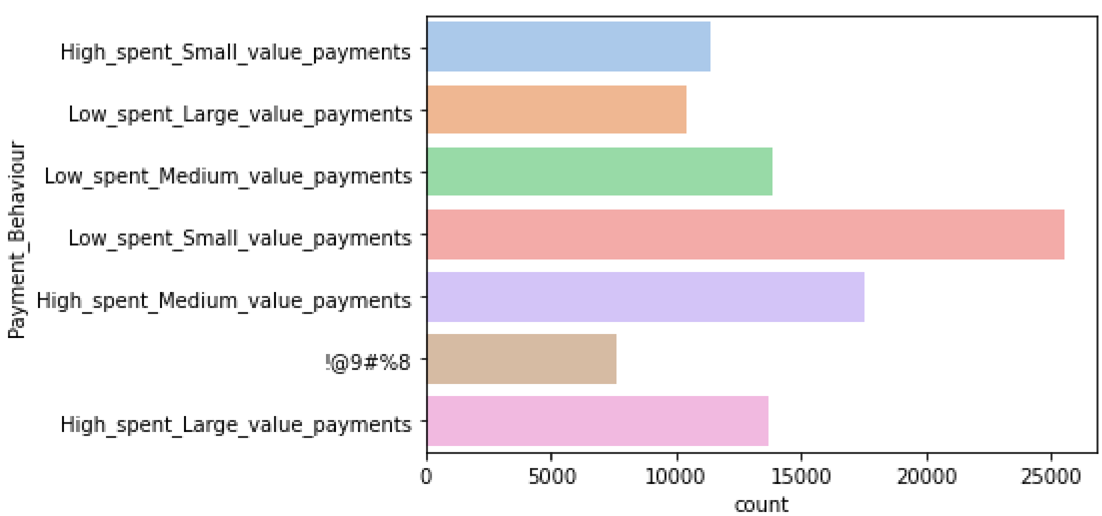
Column: Payment_Behaviour

In [161…
```
df['Payment_Behaviour'].dtype
```

Out[1615]: dtype('O')

In [161…
```
sns.countplot(data=df, y='Payment_Behaviour', palette='pastel')
```

Out[1616]: <AxesSubplot:xlabel='count', ylabel='Payment_Behaviour'>

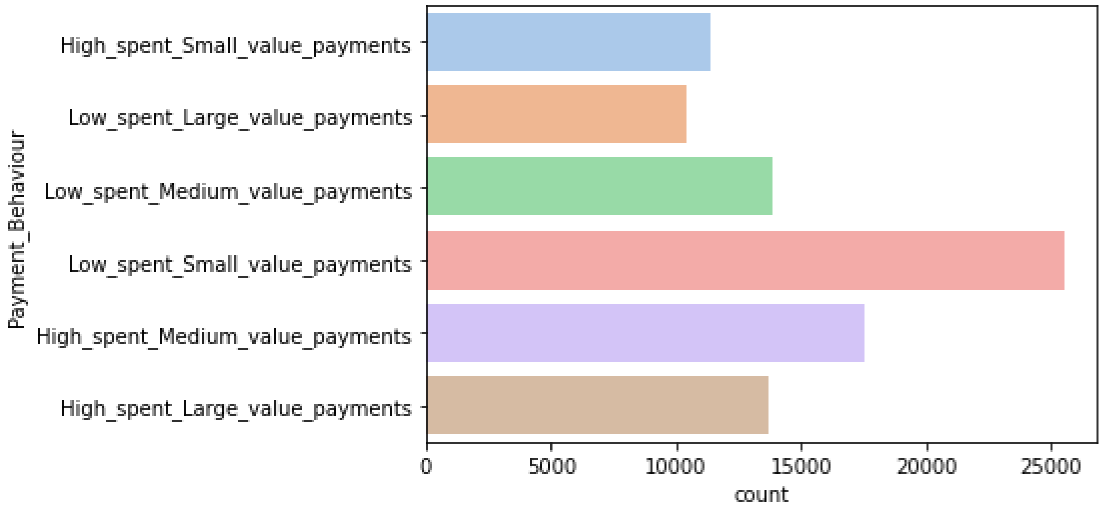


There is a grabage value !@9#%8 which we are removing manually

In [161…
```
df.loc[df['Payment_Behaviour'] == '!@9#%8', 'Payment_Behaviour'] = None
```

In [161…
```
sns.countplot(data=df, y='Payment_Behaviour', palette='pastel')
```

Out[1618]: <AxesSubplot:xlabel='count', ylabel='Payment_Behaviour'>



Column: Monthly_Balance

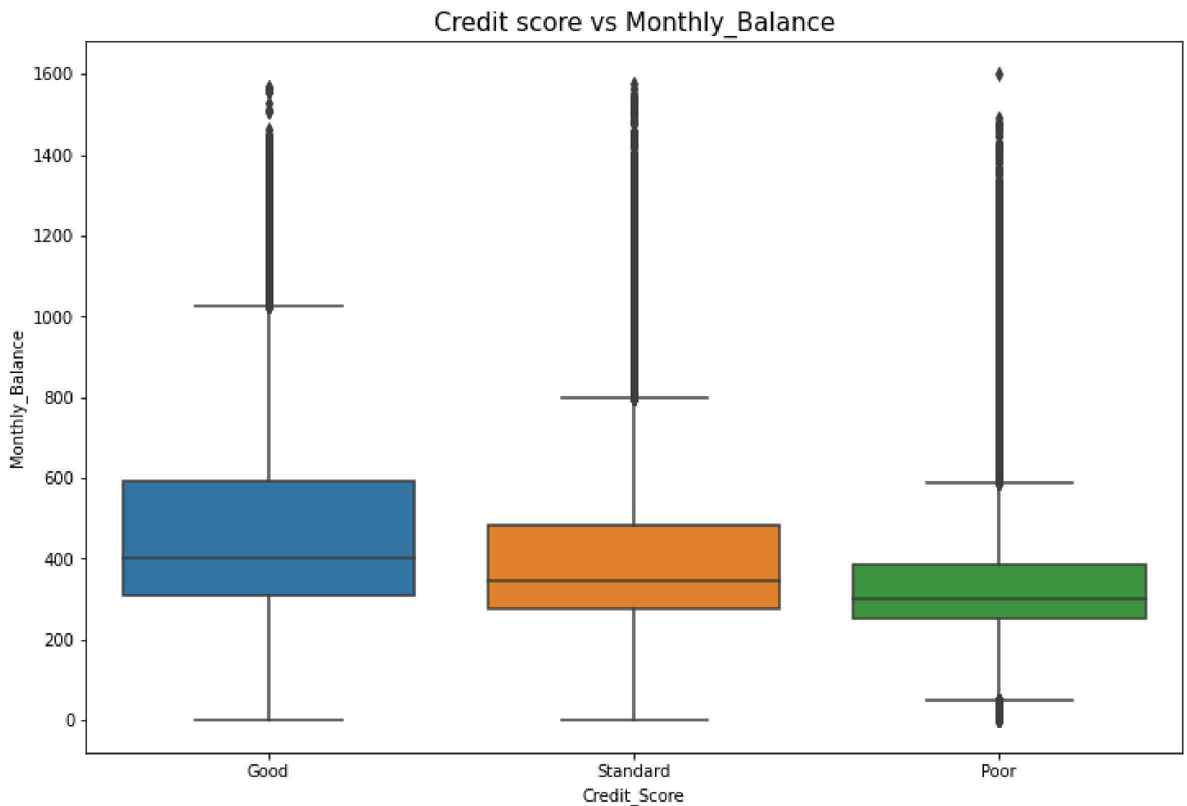
In [161…
```
df['Monthly_Balance'].dtype
```

Out[1619]: dtype('O')

In [162…
```
#remove '__-333333333333333333333333333__' from Monthly_Balance
df.loc[df['Monthly_Balance'] == '__-333333333333333333333333333__', 'Monthly
```

In [162…
```
#convert object to float data type
df['Monthly_Balance'] = df['Monthly_Balance'].astype('float64')
```

In [162…
```
box_plot(df, 'Credit_Score', 'Monthly_Balance', 'Credit score vs Monthly_Bal
```

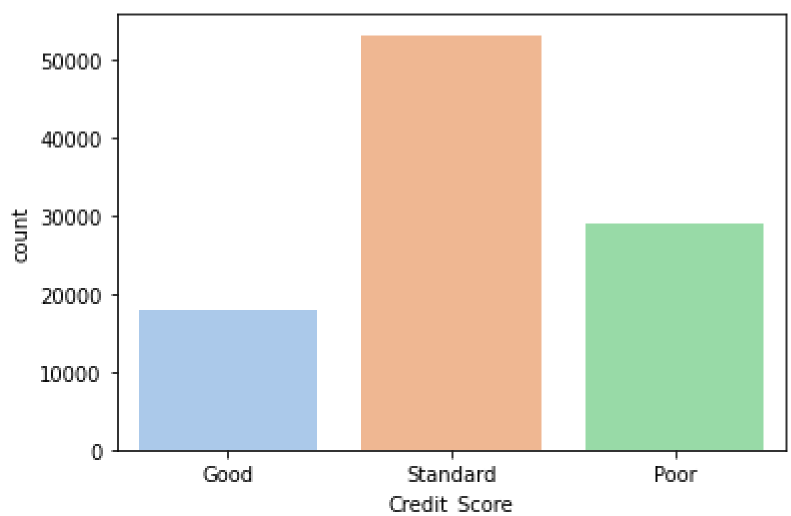


Column: Credit_Score

In [162…

```
sns.countplot(data=df, x='Credit_Score', palette='pastel')
```

Out[1623]: <AxesSubplot:xlabel='Credit_Score', ylabel='count'>



Handling outliers: We handle outliers for all numerical variables through this function

In [164…

```
numerical_cols = [col for col in df.columns if (df[col].dtype == 'int64') |
handle_outliers_numericals(df, numerical_cols)
```

There are NaN values for Inhand salary, so use the helper function fill_na() to

impute missing data

In [162…

```
for col in df.columns:
  fill_na(df, col)
```

Confirming that we have no NULL values remaining after our operations

In [162…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 24 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   Age                       100000 non-null  float64
 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  float64
 8   Num_Credit_Card           100000 non-null  float64
 9   Num_of_Loan               100000 non-null  float64
 10  Delay_from_due_date       100000 non-null  float64
 11  Num_of_Delayed_Payment    100000 non-null  float64
 12  Changed_Credit_Limit      100000 non-null  float64
 13  Num_Credit_Inquiries      100000 non-null  float64
 14  Credit_Mix                100000 non-null  object 
 15  Outstanding_Debt          100000 non-null  float64
 16  Credit_Utilization_Ratio  100000 non-null  float64
 17  Credit_History_Age        100000 non-null  float64
 18  Payment_of_Min_Amount     100000 non-null  object 
 19  Total_EMI_per_month       100000 non-null  float64
 20  Amount_invested_monthly   100000 non-null  float64
 21  Payment_Behaviour         100000 non-null  object 
 22  Monthly_Balance           100000 non-null  float64
 23  Credit_Score              100000 non-null  object 
dtypes: float64(16), object(8)
memory usage: 18.3+ MB
```

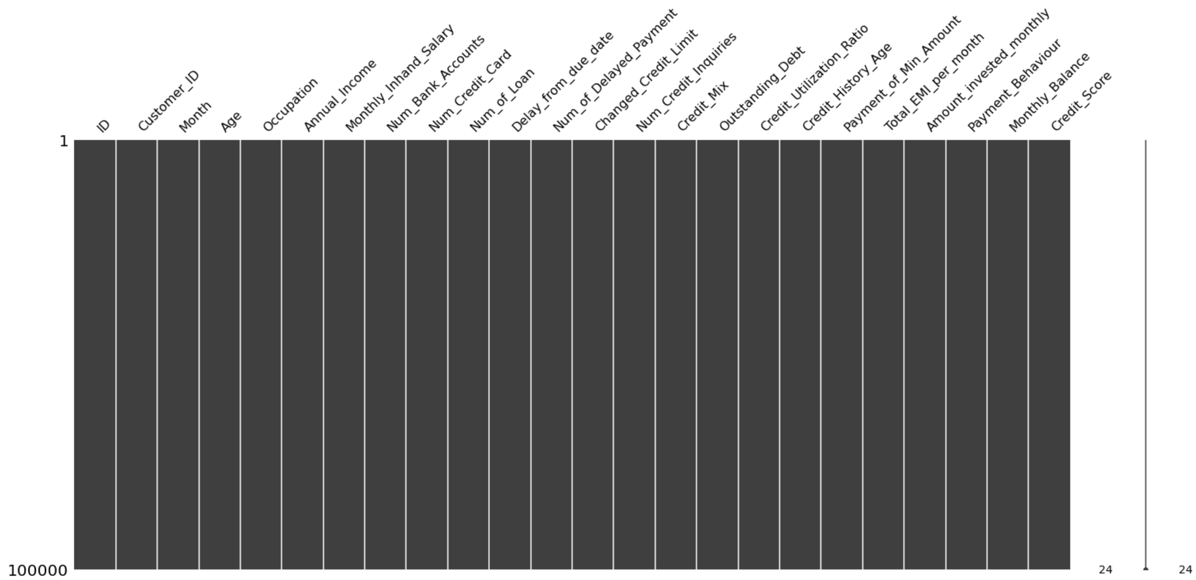
Data visualization after feature engineering

We can see that all null values have been removed

In [162…

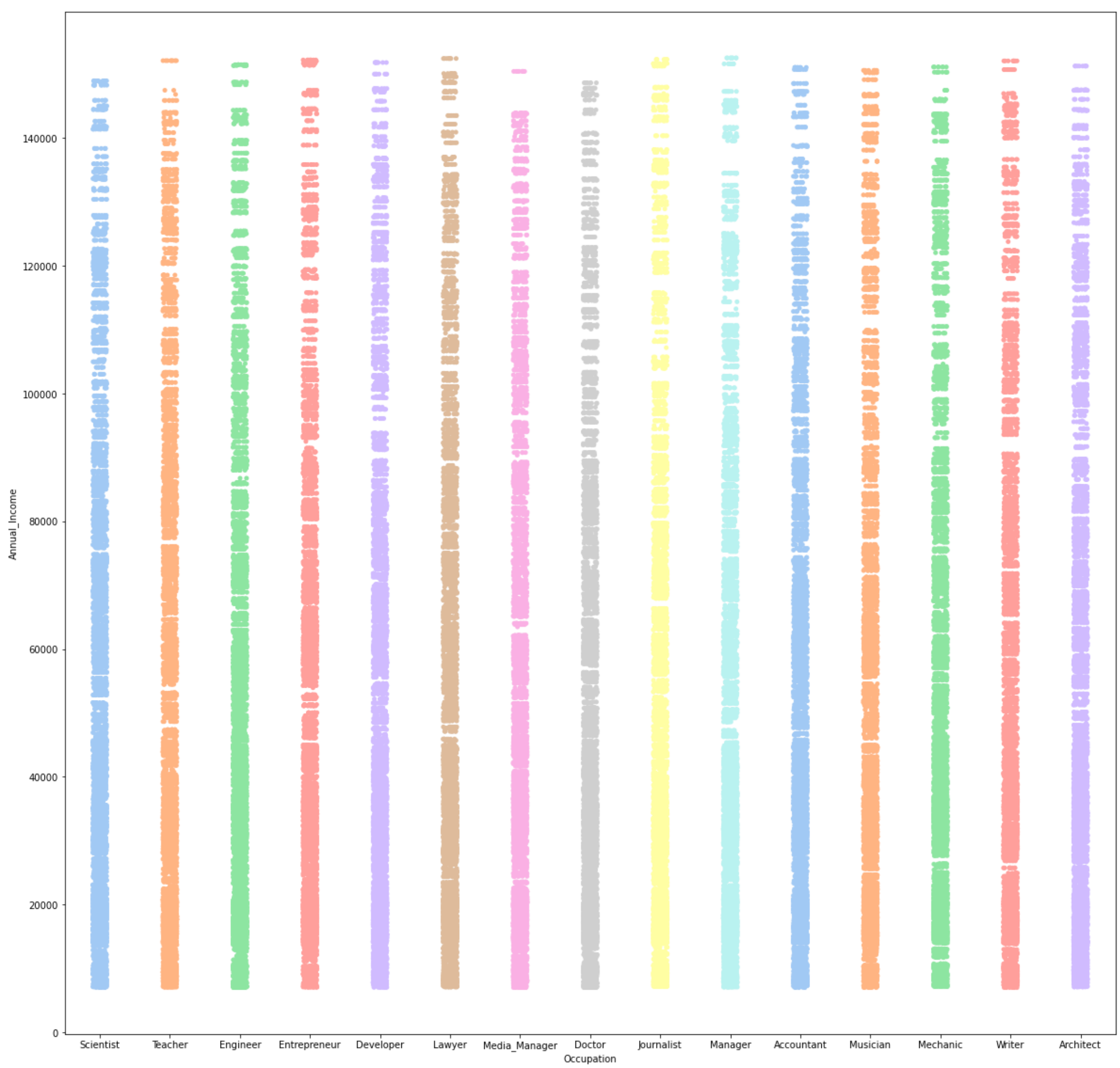
```
missingno.matrix(df)
```

Out[1627]: <AxesSubplot:>



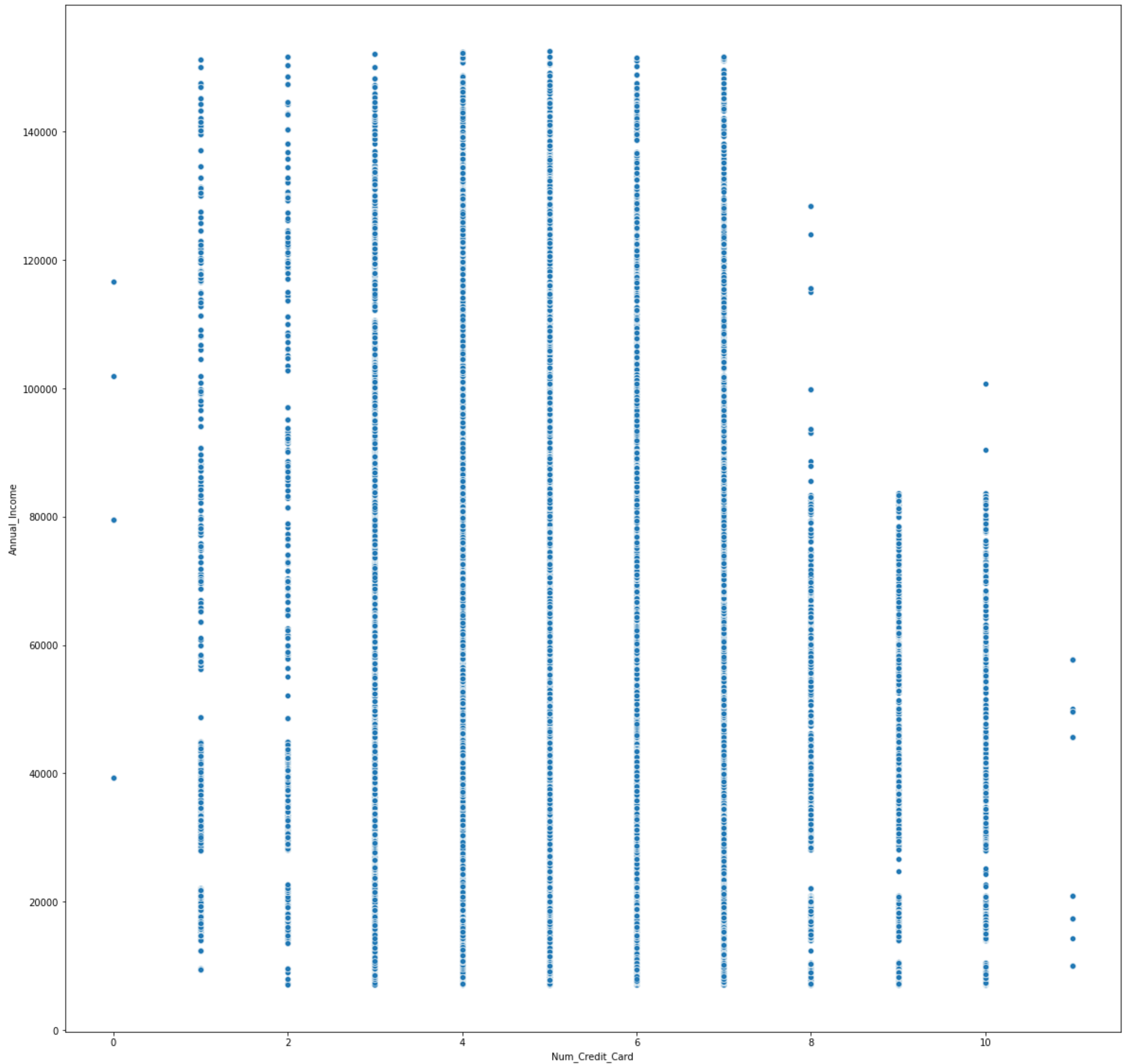
In [162…

```
plt.figure(figsize=[20, 20])
sns.stripplot(data=df, x='Occupation', y='Annual_Income', palette='pastel')
plt.show()
```



In [162…

```
plt.figure(figsize=[20, 20])
sns.scatterplot(data=df, x='Num_Credit_Card', y='Annual_Income', palette='pa
plt.show()
```



The above scatter plot shows that higher number of credit cards do not usually indicate more income as they could be in use for benfits, and 0 credit card also a indicator of low salary

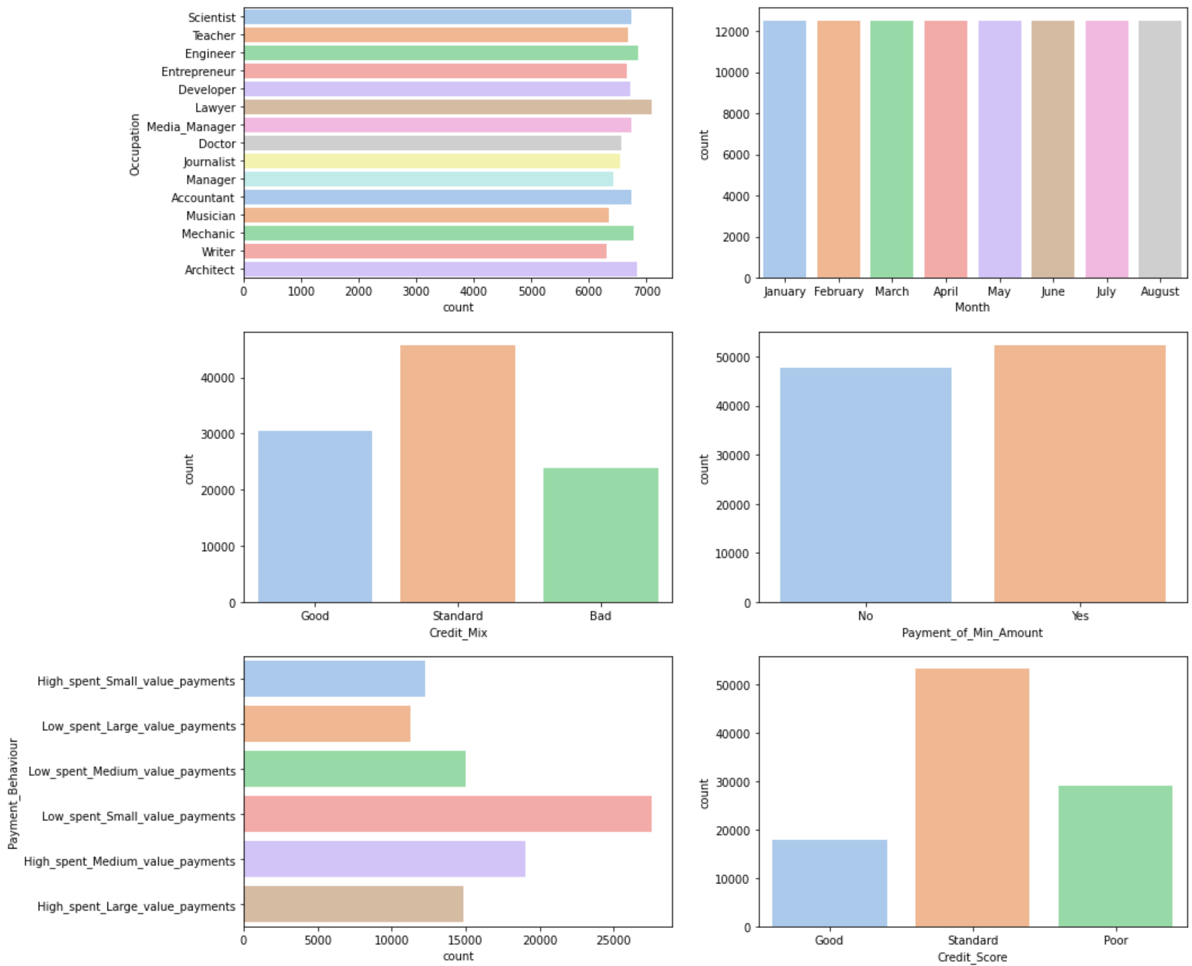
Plotting countplot of categrorical variables to find their counts in the data

In [163…

```
import seaborn as sns
fig, ax =plt.subplots(3,2, figsize=(15,15))
sns.countplot(data=df, y='Occupation', ax=ax[0,0], palette='pastel')
sns.countplot(data=df, x='Month', ax=ax[0,1], palette='pastel')
sns.countplot(data=df, x='Credit_Mix', ax=ax[1,0], palette='pastel')
sns.countplot(data=df, x='Payment_of_Min_Amount', ax=ax[1,1], palette='paste
sns.countplot(data=df, y='Payment_Behaviour', ax=ax[2,0], palette='pastel')
sns.countplot(data=df, x='Credit_Score', ax=ax[2,1], palette='pastel')
```

Out[1630]:

```
<AxesSubplot:xlabel='Credit_Score', ylabel='count'>
```

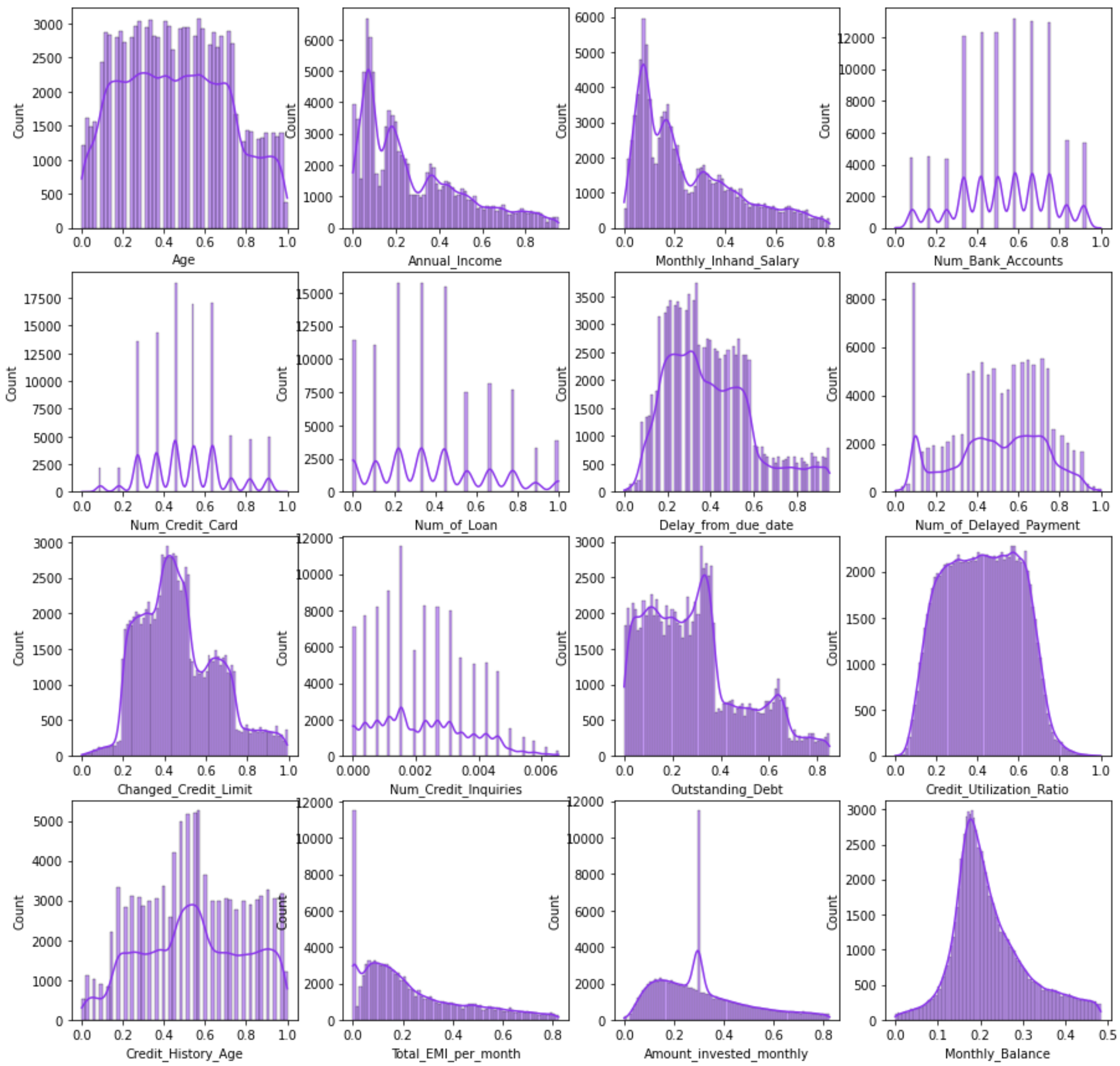


observation: We can see that most people have low spenditure and small value payments, which is the median and high spending can indicate a good credit score

Plotting histogram of numerical variables to find their distributions

In [164…

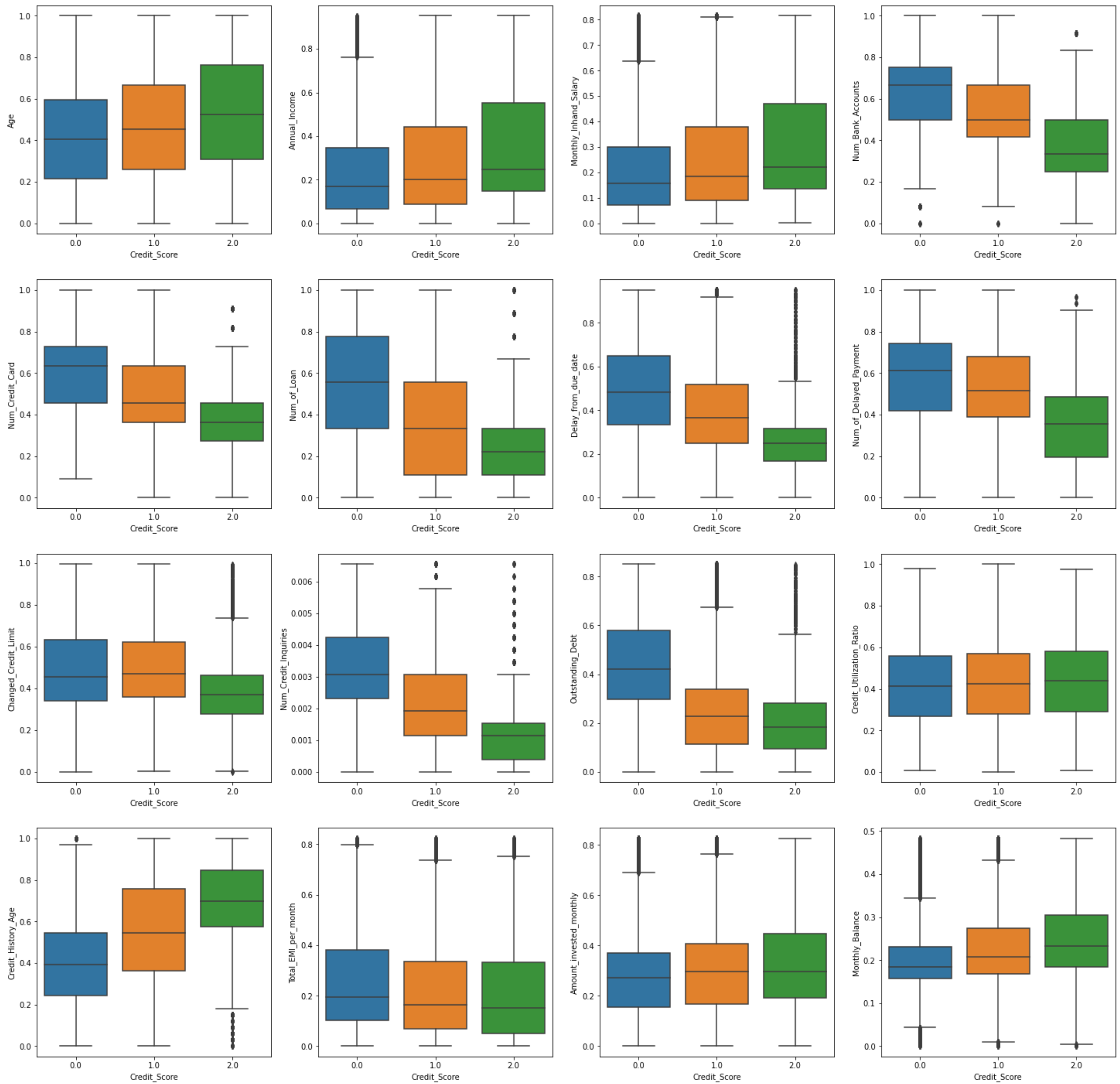
```
import seaborn as sns
fig, ax =plt.subplots(4,4, figsize=(15,15))
numCols = df.select_dtypes([np.number]).columns

i=0;
for j in range(4):
  for k in range(4):
    sns.histplot(data = df, x = numCols[i], kde = True,color='#8934eb', ax=a
    i += 1
```



Plotting a box plot for all numerical columns against target variable. To understand the scale of data and if there is a need for scaling

In [164…

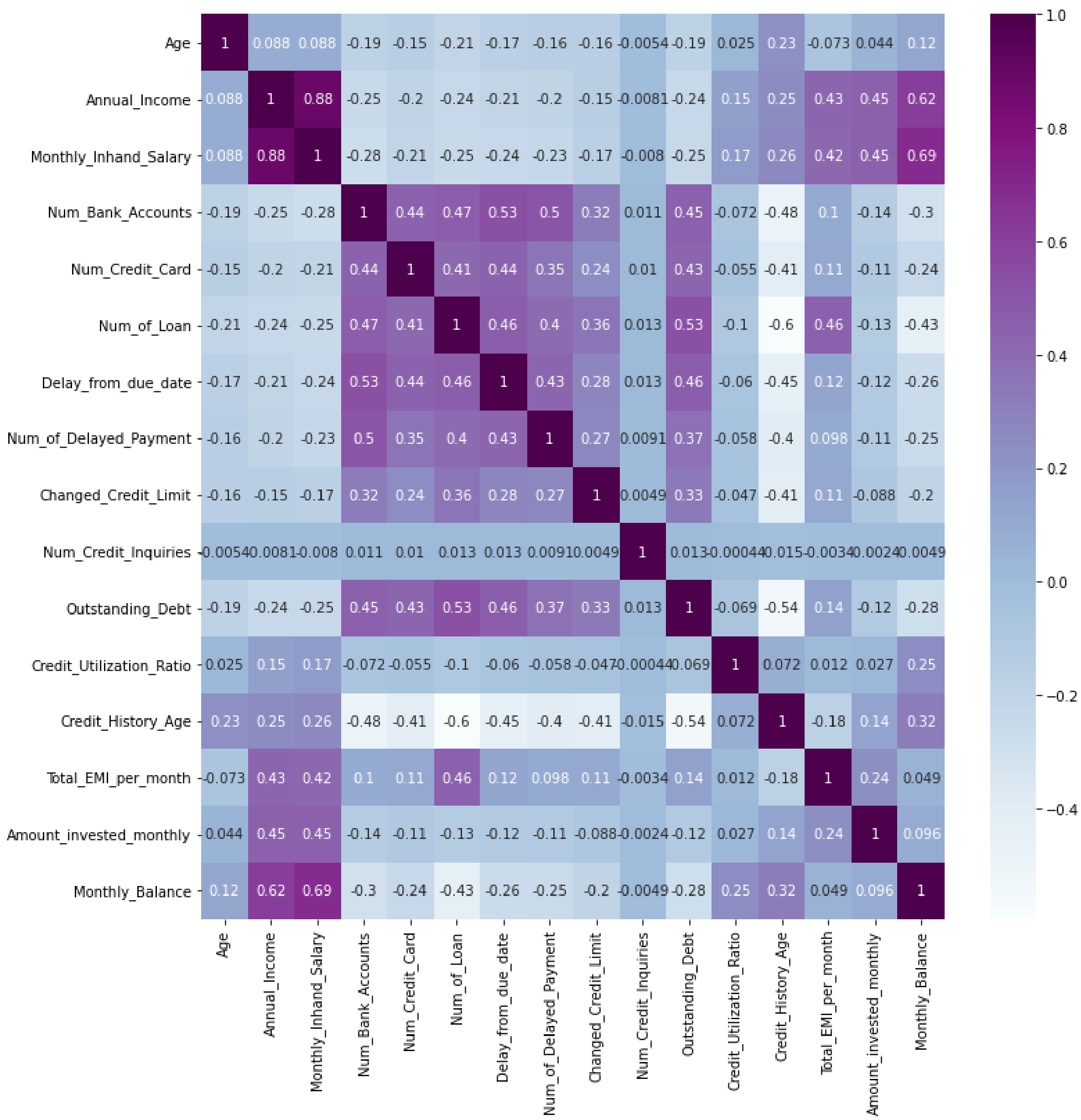
```
import seaborn as sns
fig, ax =plt.subplots(4,4, figsize=(25,25))
numCols = df.select_dtypes([np.number]).columns

i=0;
for j in range(4):
  for k in range(4):
    sns.boxplot(data = df, x = df['Credit_Score'], y = numCols[i], ax=ax[j,
    i += 1
```



Creating a HeatMap to understand the correlation between different columns and we can combine/remove some columns

In [163…
```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr(),annot=True,cmap="BuPu")
```

Out[1633]: <AxesSubplot:>



As expected there is a high correlation between Annual Income and Monthly Inhand Salary, which makes sense. We can identify other variables that also might make sense to a domain expert

Scaling of numeric columns: We will split out the numerical data and scale them as values have different scale

In [163…
```
numerical_df_columns = [col for col in df.columns if (df[col].dtype == 'int6

numerical_df = df[numerical_df_columns].copy()

onehot_object_df = df[['Occupation', 'Payment_of_Min_Amount']].copy()
```

In [163…
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
numerical_df = pd.DataFrame(scaler.fit_transform(df[numerical_df_columns]),
```

```
                              index=df[numerical_df_columns].index,
                              columns=df[numerical_df_columns].columns)
```

In [163…

```
numerical_df
```

Out[1636]:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Cr
0	0.214286	0.083178	0.102087	0.333333	
1	0.214286	0.083178	0.102087	0.333333	
2	0.214286	0.083178	0.102087	0.333333	
3	0.214286	0.083178	0.102087	0.333333	
4	0.214286	0.083178	0.102087	0.333333	
...	...	...	...	...	
99995	0.261905	0.224107	0.205072	0.416667	
99996	0.261905	0.224107	0.205072	0.416667	
99997	0.261905	0.224107	0.205072	0.416667	
99998	0.261905	0.224107	0.205072	0.416667	
99999	0.261905	0.224107	0.205072	0.416667	

100000 rows × 16 columns

Encoding Categorical Values: We will convert categorical variables to numerical using mapping and Onehot encoding to make sure we are able to feed them into our model.

In [163…

```
onehot_object_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 2 columns):
 #   Column                Non-Null Count   Dtype 
---  ------                --------------   ----- 
 0   Occupation            100000 non-null  object
 1   Payment_of_Min_Amount  100000 non-null  object
dtypes: object(2)
memory usage: 1.5+ MB
```

In [163…

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(handle_unknown='ignore')

onehot_encoder_df_1 = pd.DataFrame(encoder.fit_transform(df[['Occupation']])
onehot_final_1 = pd.concat([pd.get_dummies(df["Occupation"],prefix="Occupati

onehot_encoder_df_2 = pd.DataFrame(encoder.fit_transform(df[['Payment_of_Min
onehot_final_2 = pd.concat([pd.get_dummies(df["Payment_of_Min_Amount"],prefi

onehot_encoder_df = pd.concat([onehot_final_1, onehot_final_2], axis=1, join
onehot_encoder_df
```

Out[1638]:

	Occupation_Accountant	Occupation_Architect	Occupation_Developer	Occupation
0	0	0	0	
1	0	0	0	
2	0	0	0	
3	0	0	0	
4	0	0	0	
...	...	...	...	
99995	0	0	0	
99996	0	0	0	
99997	0	0	0	
99998	0	0	0	
99999	0	0	0	

100000 rows × 17 columns

Label Mapping: We will map ordinal variables accordingly as they indicate a increasing positive behaviour about our user. We will also do the same for our target variable as we know that Good > Standard > Bad

In [163…

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

Out[1639]:

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

In [164…

```
df['Payment_Behaviour'].unique()
```

Out[1640]:

```
array(['High_spent_Small_value_payments',
       'Low_spent_Large_value_payments',
       'Low_spent_Medium_value_payments',
       'Low_spent_Small_value_payments',
       'High_spent_Medium_value_payments',
       'High_spent_Large_value_payments'], dtype=object)
```

In [164…

```
df['Credit_Score'].unique()
```

Out[1641]:

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

In [164…

```
le_cols = ['Credit_Mix', 'Payment_Behaviour', 'Credit_Score']

Credit_Mix_mapper = {"Good":2 ,"Standard":1, "Bad":0}
Payment_Behaviour_mapper = {'High_spent_Small_value_payments': 6,
       'Low_spent_Large_value_payments': 5,
       'Low_spent_Medium_value_payments': 4,
       'Low_spent_Small_value_payments': 3,
       'High_spent_Medium_value_payments': 2,
       'High_spent_Large_value_payments': 1}
Credit_Score_mapper = {"Good":2 ,"Standard":1, "Poor":0}
df["Credit_Mix"] = df["Credit_Mix"].replace(Credit_Mix_mapper)
```

```
df["Payment_Behaviour"] = df["Payment_Behaviour"].replace(Payment_Behaviour_
df["Credit_Score"] = df["Credit_Score"].replace(Credit_Score_mapper)
```

Concatenating columns after scaling, one hot encoding and label encoding

In [164…

```
df_final = df[['Credit_Mix', 'Payment_Behaviour', 'Credit_Score']].copy()
df_final = pd.concat([numerical_df, onehot_encoder_df, df_final], axis=1, jo
df_final #has numeric, one hot encoded columns, label encoded columns
```

Out[1643]:

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Cr
0	0.214286	0.083178	0.102087	0.333333	
1	0.214286	0.083178	0.102087	0.333333	
2	0.214286	0.083178	0.102087	0.333333	
3	0.214286	0.083178	0.102087	0.333333	
4	0.214286	0.083178	0.102087	0.333333	
...	...	...	...	...	
99995	0.261905	0.224107	0.205072	0.416667	
99996	0.261905	0.224107	0.205072	0.416667	
99997	0.261905	0.224107	0.205072	0.416667	
99998	0.261905	0.224107	0.205072	0.416667	
99999	0.261905	0.224107	0.205072	0.416667	

100000 rows × 36 columns

In [164…

```
df = df_final

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 36 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   Age                       100000 non-null  float64
 1   Annual_Income             100000 non-null  float64
 2   Monthly_Inhand_Salary     100000 non-null  float64
 3   Num_Bank_Accounts         100000 non-null  float64
 4   Num_Credit_Card           100000 non-null  float64
 5   Num_of_Loan               100000 non-null  float64
 6   Delay_from_due_date       100000 non-null  float64
 7   Num_of_Delayed_Payment    100000 non-null  float64
 8   Changed_Credit_Limit      100000 non-null  float64
 9   Num_Credit_Inquiries      100000 non-null  float64
 10  Outstanding_Debt          100000 non-null  float64
 11  Credit_Utilization_Ratio  100000 non-null  float64
 12  Credit_History_Age        100000 non-null  float64
 13  Total_EMI_per_month       100000 non-null  float64
 14  Amount_invested_monthly   100000 non-null  float64
 15  Monthly_Balance           100000 non-null  float64
 16  Occupation_Accountant     100000 non-null  uint8  
 17  Occupation_Architect      100000 non-null  uint8  
 18  Occupation_Developer      100000 non-null  uint8  
 19  Occupation_Doctor         100000 non-null  uint8  
 20  Occupation_Engineer       100000 non-null  uint8  
 21  Occupation_Entrepreneur   100000 non-null  uint8  
 22  Occupation_Journalist     100000 non-null  uint8  
 23  Occupation_Lawyer         100000 non-null  uint8  
 24  Occupation_Manager        100000 non-null  uint8  
 25  Occupation_Mechanic       100000 non-null  uint8  
 26  Occupation_Media_Manager  100000 non-null  uint8  
 27  Occupation_Musician       100000 non-null  uint8  
 28  Occupation_Scientist      100000 non-null  uint8  
 29  Occupation_Teacher        100000 non-null  uint8  
 30  Occupation_Writer         100000 non-null  uint8  
 31  Payment_of_Min_Amount_No  100000 non-null  uint8  
 32  Payment_of_Min_Amount_Yes 100000 non-null  uint8  
 33  Credit_Mix                100000 non-null  int64  
 34  Payment_Behaviour         100000 non-null  int64  
 35  Credit_Score              100000 non-null  int64  
dtypes: float64(16), int64(3), uint8(17)
memory usage: 16.1 MB
```

Split data into training and testing set

In [164…

```
from sklearn.model_selection import train_test_split
X = df.iloc[:,:-1].values
Y = df.iloc[:,-1].values
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, ran
```