PACKAGES

Python Packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

Package Settings

```
import plotly.io as pio
pio.templates.default = "plotly_dark"

np.random.seed(31)
pd.set_option('display.max_columns', None)
sns.set_theme(context='notebook', style='darkgrid', palette='deep',
              font='sans-serif', font_scale=1, color_codes=True, rc=None)
```

Import Data

```
from google.colab import drive
drive.mount('/content/drive')
```

```
main = pd.read_csv('/content/drive/MyDrive/DS_Projects/Credit_Score_Jun23/train.csv')
```

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

Cleaned Data: https://www.kaggle.com/datasets/clkmuhammed/creditscoreclassification?select=train.csv

Task: Given a person's credit-related information, perform analysis, identify factors influencing credit score and build a machine learning model that can classify credit score.

OVERVIEW

```
main.head()
```

ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income
			Aaron				

- Age: Represents the age of the person
- Annual_Income: Represents the annual income of the person
- Monthly_Inhand_Salary: Represents the monthly base salary of a person
- Num_Bank_Accounts:Represents the number of bank accounts a person holds
- Num_Credit_Card: Represents the number of other credit cards held by a person
- Interest_Rate: Represents the interest rate on credit card (percent)
- Num_of_Loan: Represents the number of loans taken from the bank
- Delay_from_due_date: Represents the average number of days delayed from the payment date (days)
- Num_of_Delayed_Payment: Represents the average number of payments delayed by a person
- Changed_Credit_Limit: Represents the percentage change in credit card limit (percent)
- Num_Credit_Inquiries: Represents the number of credit card inquiries
- Credit_Mix: Represents the classification of the mix of credits (Bad, Standard, Good)
- Outstanding_Debt: Represents the remaining debt to be paid
- Credit_Utilization_Ratio: Represents the utilization ratio of credit card (percent)
- Credit_History_Age: Represents the age of credit history of the person (days)
- Payment_of_Min_Amount: Represents whether only the minimum amount was paid by the person
- Total_EMI_per_month: Represents the monthly EMI payments
- Amount_invested_monthly: Represents the monthly amount invested by the customer
- Monthly_Balance: Represents the monthly balance amount of the customer
- Credit_Score: Represents the bracket of credit score (Poor, Standard, Good)

```
# Function to create numerical and categorical tables
def describe_data(df):
  # Create numerical and categorical describe tables
  num_describe = df.describe()
  num_describe = num_describe.T.reset_index()
  cat_describe = df.describe(exclude=np.number)
  cat_describe = cat_describe.T.reset_index()
  # Create null and data types tables
  dtypes = df.dtypes.reset_index()
  dtypes.columns = ['index','dtype']
  null = pd.DataFrame(df.isna().sum()).reset_index()
  null.columns = ['index','null_values']
  # Merge null values and data types to describe tables
  num_describe = num_describe.merge(null, how='left', on='index')
  num_describe = num_describe.merge(dtypes, how='left', on='index')
  cat_describe = cat_describe.merge(null, how='left', on='index')
  cat_describe = cat_describe.merge(dtypes, how='left', on='index')

  return num_describe, cat_describe

num_describe, cat_describe = describe_data(main)
```

Describe Tables

```
num_describe
```

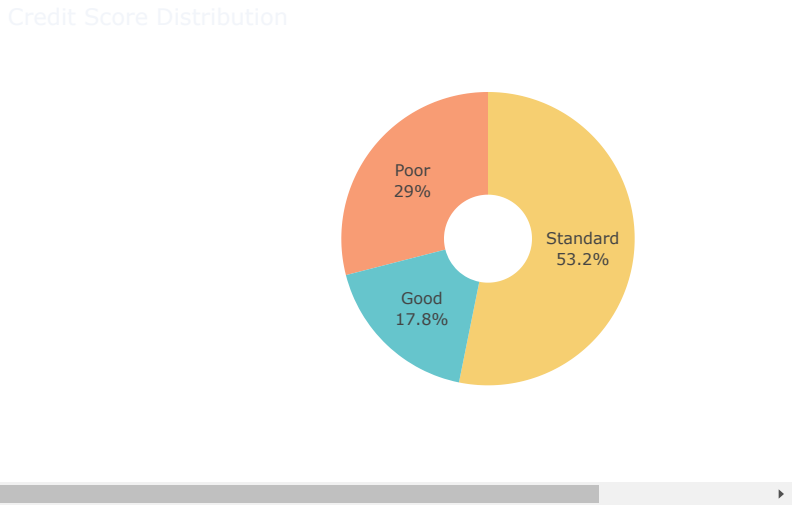
	index	count	mean	std	min	
0	ID	100000.0	8.063150e+04	4.330149e+04	5634.000000	4.3
1	Customer_ID	100000.0	2.598267e+04	1.434054e+04	1006.000000	1.3
2	Month	100000.0	4.500000e+00	2.291299e+00	1.000000	2.7
3	Age	100000.0	3.331634e+01	1.076481e+01	14.000000	2.4
4	SSN	100000.0	5.004617e+08	2.908267e+08	81349.000000	2.4
5	Annual_Income	100000.0	5.050512e+04	3.829942e+04	7005.930000	1.9
6	Monthly_Inhand_Salary	100000.0	4.197271e+03	3.186432e+03	303.645417	1.6
7	Num_Bank_Accounts	100000.0	5.368820e+00	2.593314e+00	0.000000	3.0
8	Num_Credit_Card	100000.0	5.533570e+00	2.067098e+00	0.000000	4.0
9	Interest_Rate	100000.0	1.453208e+01	8.741330e+00	1.000000	7.0
10	Num_of_Loan	100000.0	3.532880e+00	2.446356e+00	0.000000	2.0
11	Delay_from_due_date	100000.0	2.108141e+01	1.480456e+01	0.000000	1.0

```
cat_describe
```

	index	count	unique	top	freq	nu
0	Name	100000	10128	Jessicad	48	
1	Occupation	100000	15	Lawyer	7096	
2	Type_of_Loan	100000	6261	No Data	11408	
3	Credit_Mix	100000	3	Standard	45848	
4	Payment_of_Min_Amount	100000	3	Yes	52326	
5	Payment_Behaviour	100000	6	Low_spent_Small_value_payments	28585	
6	Credit_Score	100000	3	Standard	53174	

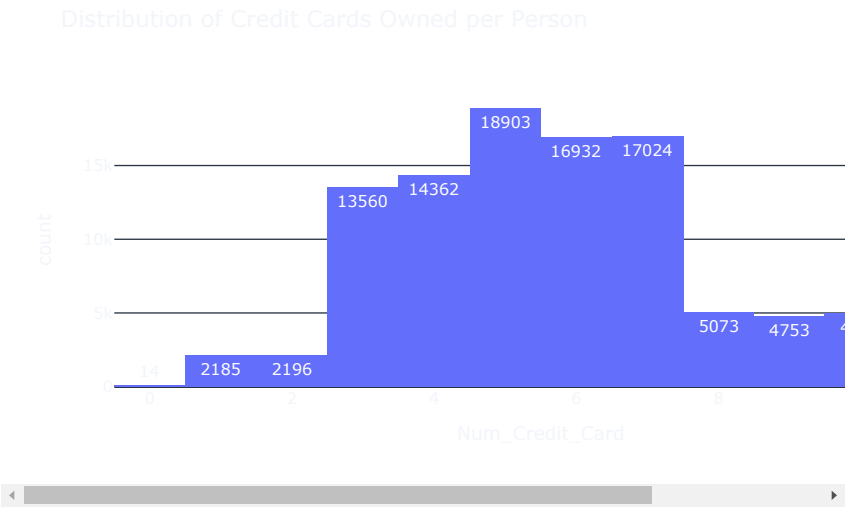
EDA

```
fig = px.pie(main, names='Credit_Score', color='Credit_Score',hole=.3,
       color_discrete_sequence=px.colors.qualitative.Pastel,
       width=800, height=400, title="Credit Score Distribution")
fig.update_traces(textposition='inside', textinfo='percent+label', showlegend=False)
```

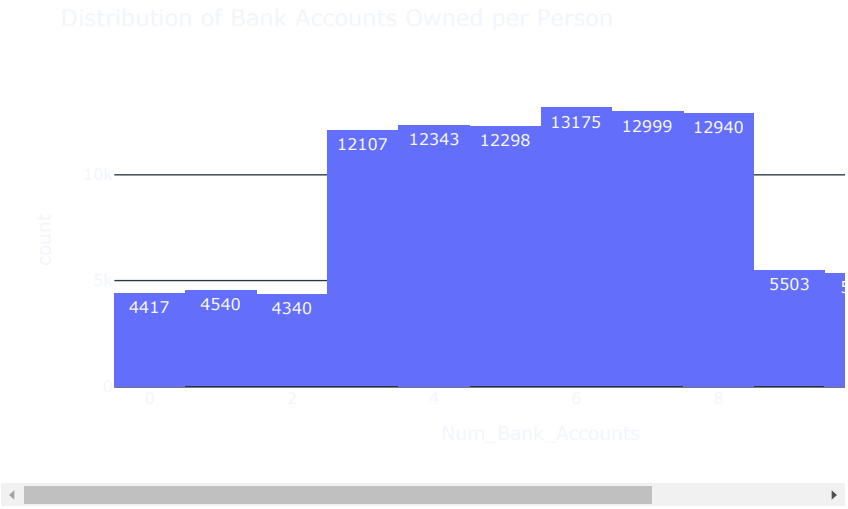


- The distribution of target feature is imbalanced with large number of entries fall on Standard Credit_Score.

```
px.histogram(main, x='Num_Credit_Card', text_auto='d',
             width=800, height=400, title="Distribution of Credit Cards Owned per Person")
```




```
px.histogram(main, x='Num_Bank_Accounts', text_auto='d',
             width=800, height=400, title="Distribution of Bank Accounts Owned per Person")
```



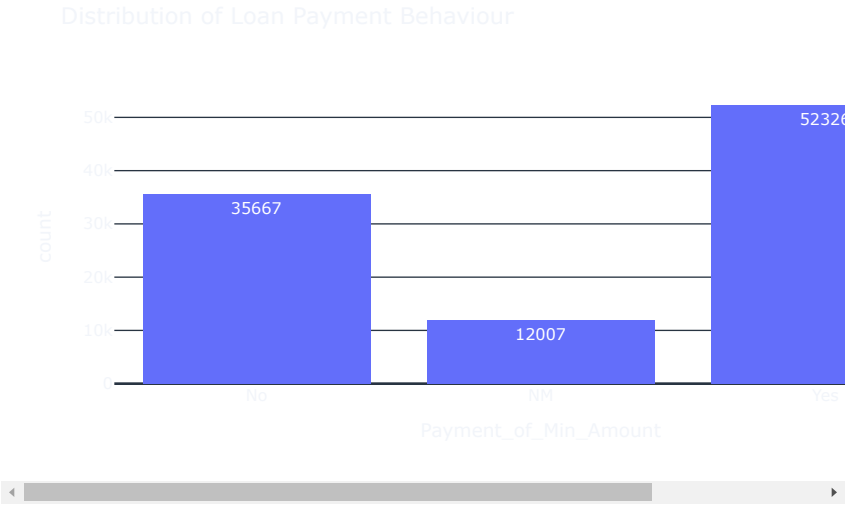
- There are records with zero Credit Cards and records with zero Bank Account.

```
main[main['Num_Bank_Accounts']==0].head()
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income
48	5706	10314	1	Nadiaq	33.0	411510676.0	Lawyer	131313.4
49	5707	10314	2	Nadiaq	34.0	411510676.0	Lawyer	131313.4
50	5708	10314	3	Nadiaq	34.0	411510676.0	Lawyer	131313.4
51	5709	10314	4	Nadiaq	34.0	411510676.0	Lawyer	131313.4
52	5710	10314	5	Nadiaq	34.0	411510676.0	Lawyer	131313.4

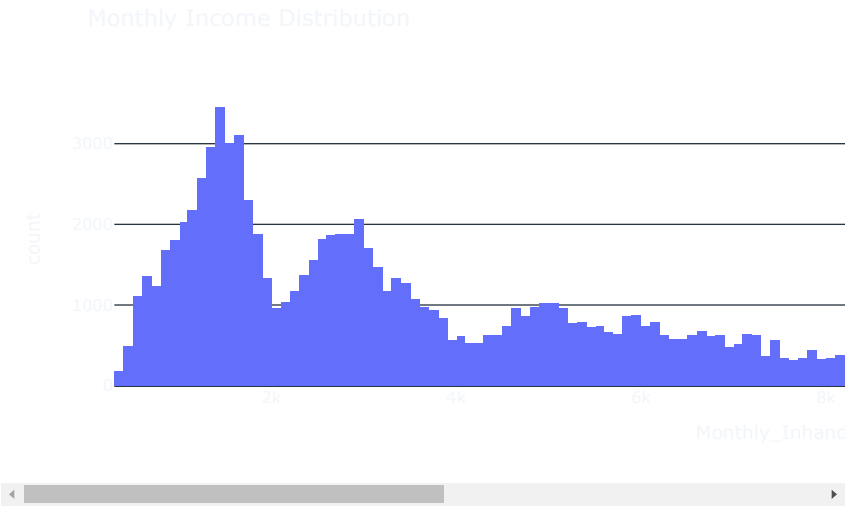
- There is a sizable number of records with zero bank accounts and Loan Type gave no concrete conclusion so it is assumed to be correct.

```
px.histogram(main, x='Payment_of_Min_Amount', text_auto='d',
             width=800, height=400, title="Distribution of Loan Payment Behaviour")
```

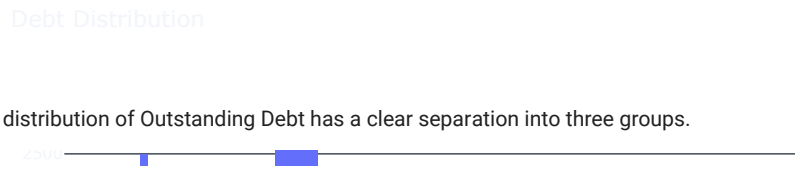


- There is an undefined value NM which we will consider as 'Not Minimum'.

```
px.histogram(main, x='Monthly_Inhand_Salary',
             width=1200, height=400, title="Monthly Income Distribution")
```

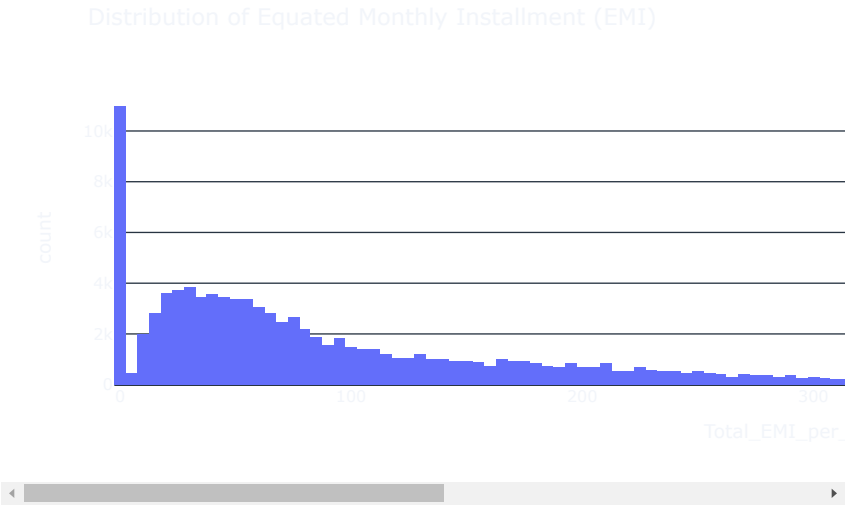



```
px.histogram(main, x='Outstanding_Debt',
             width=800, height=400, title='Debt Distribution')
```



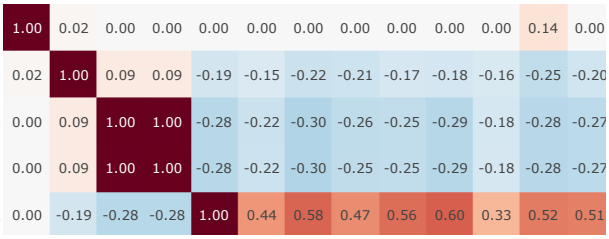
- The distribution of Outstanding Debt has a clear separation into three groups.


```
px.histogram(main[main['Total_EMI_per_month'] < 600], x='Total_EMI_per_month',
             width=1200, height=400, title='Distribution of Equated Monthly Installment (EMI)')
```



- This is the distribution of fixed payment to interest and principle each month. It has a very long and heavy tail.

```
temp = main.drop(columns=['ID','Customer_ID','SSN','Name']).copy()
temp['Credit_Score'] = temp['Credit_Score'].map({'Poor':0,'Standard':1,'Good':2})
px.imshow(temp.corr(numeric_only=True), text_auto='.2f',
                  width=950, height=950, zmin=-1, zmax=1, color_continuous_scale='RdBu_r')
```



CLEANING


```
# Drop unecessary columns
main.drop(columns=['ID','SSN','Name'], inplace=True)
# Replace '0' with '1' for Number of credit card values
main['Num_Credit_Card'].replace(0,1,inplace=True)
# Replace Minimum Payment value 'NM' with 'No'
main['Payment_of_Min_Amount'].replace('NM','No',inplace=True)
# Remap Month values
import calendar
main['Month'] = main['Month'].apply(lambda x: calendar.month_abbr[x])
```



```
# Make copy for detailed analysis
df = main.copy()
# Make target feature as Categorical for easy ordered sorting
df['Credit_Score'] = pd.Categorical(df['Credit_Score'], ordered=True, categories=['Poor', 'Standard', 'Good'])
df.sort_values('Credit_Score', inplace=True)
```


HYPOTHESIS TESTING

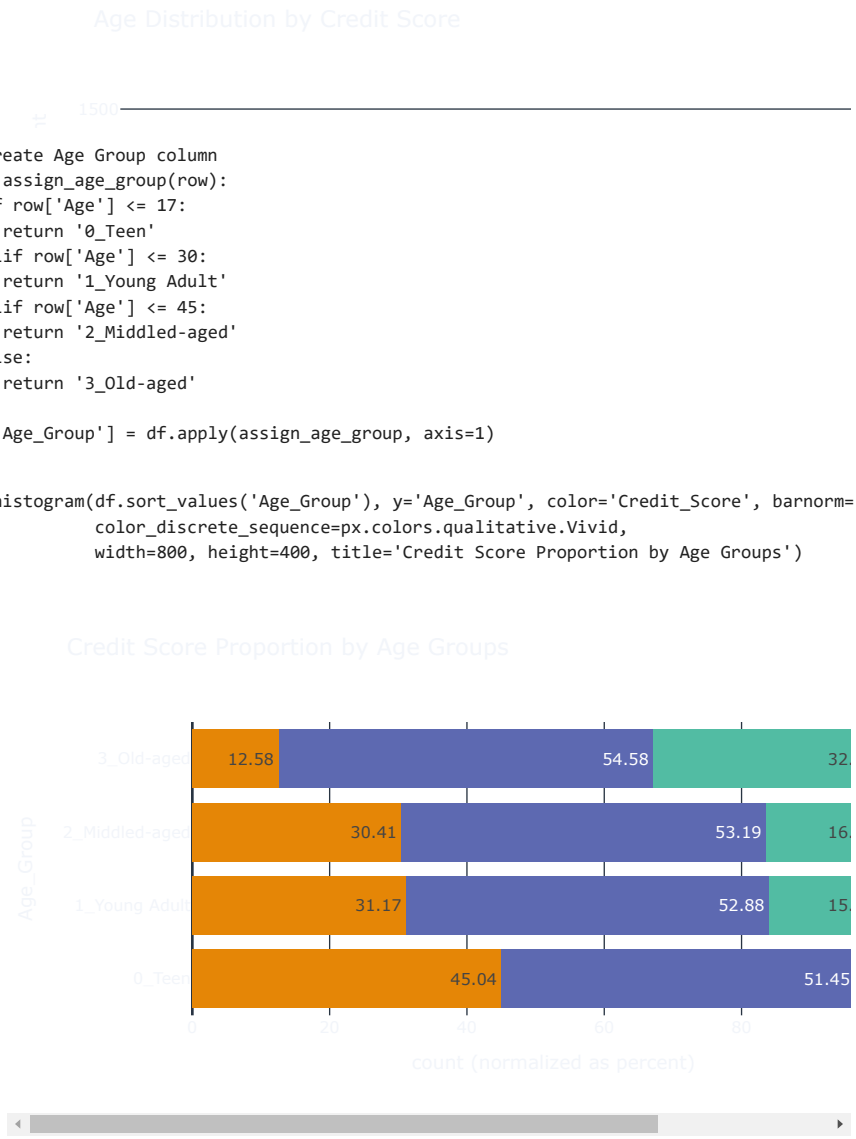
Demograph

What we are expecting to see:

1. Older people will have better credit scores as their career will be more stable with high income.
2. Higher skill and higher paying occupation will have better credit score as they can more reliably repay debts.
3. Individuals with lower Debt to Income ratio (DTI) will observe a better credit score as they will like be more responsible with finance.

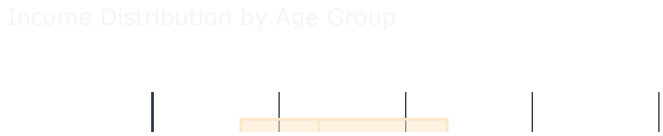
Age

```
px.histogram(df, x='Age', color='Credit_Score', facet_row='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1200, height=400, title='Age Distribution by Credit Score')
```



- By breaking up into Age Groups and normalize the distribution, we can clearly observe that as an individual gets older they will be more likely to have better Credit Score. There is correlation but we may not yet understand the cause.

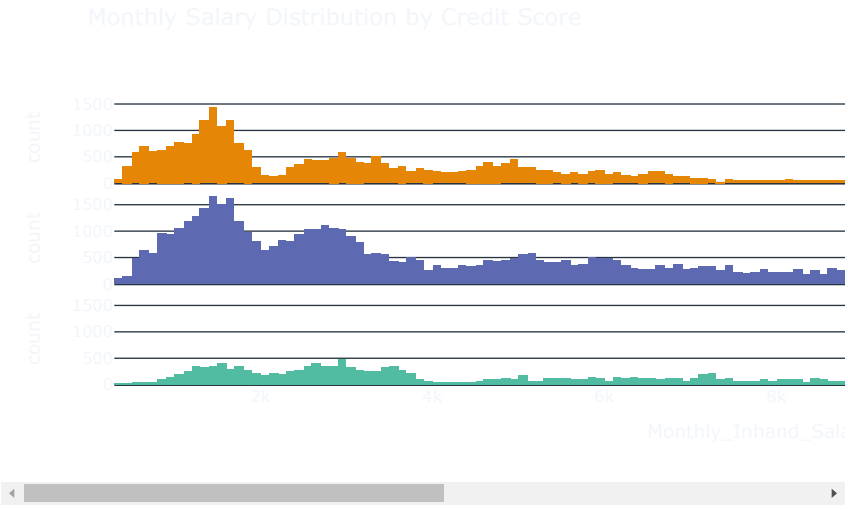
```
fig = px.box(df.sort_values('Age_Group'), x='Monthly_Inhand_Salary', y='Age_Group', color='Age_Group',
             color_discrete_sequence=px.colors.sequential.Burgyl,
             width=1000, height=500, title='Income Distribution by Age Group')
fig.update_layout(showlegend=False)
```



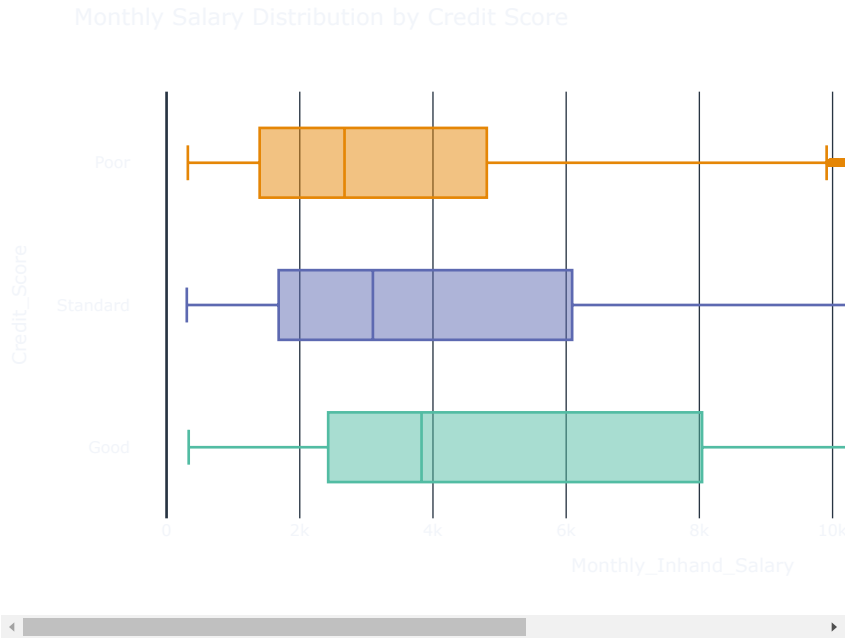
- The graph above shows what we expected. There is correlation between Age and Income. Older people will generally be higher in their career.

Income

```
px.histogram(df, x='Monthly_Inhand_Salary', color='Credit_Score', facet_row='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1200, height=400, title='Monthly Salary Distribution by Credit Score')
```



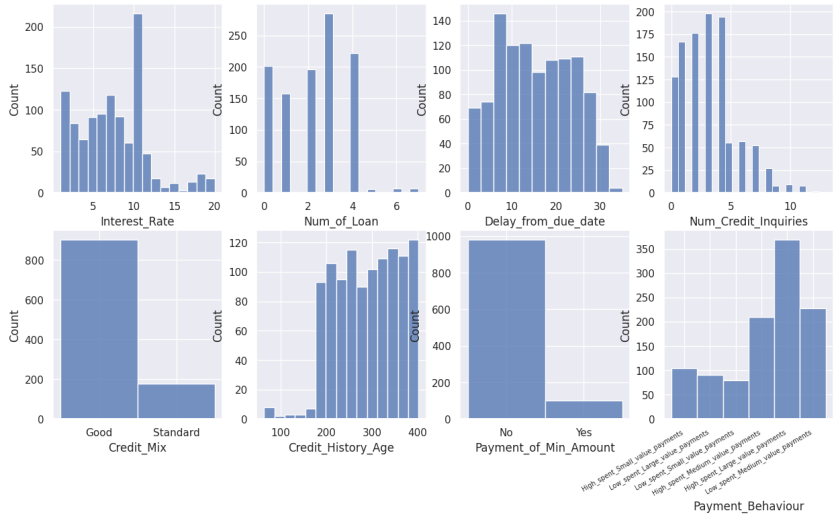

```
fig = px.box(df, x='Monthly_Inhand_Salary', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Monthly Salary Distribution by Credit Score')
fig.update_layout(showlegend=False)
```



- Although we can see that Income has a positive effect on Credit Score, the amount of outliers at the upper whiskers for Stanard and Poor Credit Score are quite substantial. There are alot of high income individuals who has low Credit Score.

```
# Get data for Poor Credit Scorer with very high income
poor_outliers = df[(df['Credit_Score']=='Poor') & (df['Monthly_Inhand_Salary']>=10000)]
# Plot distribution for quick overview of outliers
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15,8))
sns.histplot(poor_outliers['Interest_Rate'], ax=axes[0, 0])
sns.histplot(poor_outliers['Num_of_Loan'], ax=axes[0, 1])
sns.histplot(poor_outliers['Delay_from_due_date'], ax=axes[0, 2])
sns.histplot(poor_outliers['Num_Credit_Inquiries'], ax=axes[0, 3])
sns.histplot(poor_outliers['Credit_Mix'], ax=axes[1, 0])
sns.histplot(poor_outliers['Credit_History_Age'], ax=axes[1, 1])
sns.histplot(poor_outliers['Payment_of_Min_Amount'], ax=axes[1, 2])
sns.histplot(poor_outliers['Payment_Behaviour'], ax=axes[1, 3])
axes[1,3].set_xticklabels(axes[1,3].get_xticklabels(), rotation=30, fontsize=7, ha='right')
plt.show();
```

```
<ipython-input-26-007e97e7c0e0>:13: UserWarning:

FixedFormatter should only be used together with FixedLocator
```



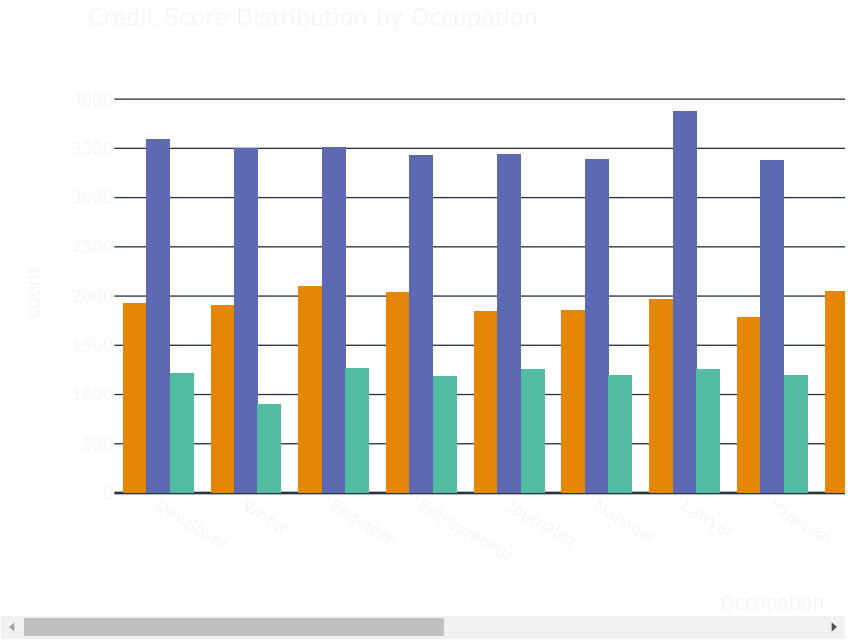

```
print(f"Percentage of Poor Outliers: {len(poor_outliers) / len(df) * 100}%")
```

```
Percentage of Poor Outliers: 1.082%
```

- These outliers has good Credit Mix, long Credit Age, not many loans and pay more than the minimum amount. But they are assigned as Poor Score by the bank. Removing these outliers may increase False Positive for Good Score. We are leaving these noise in to avoid overfitting and bias when model building.

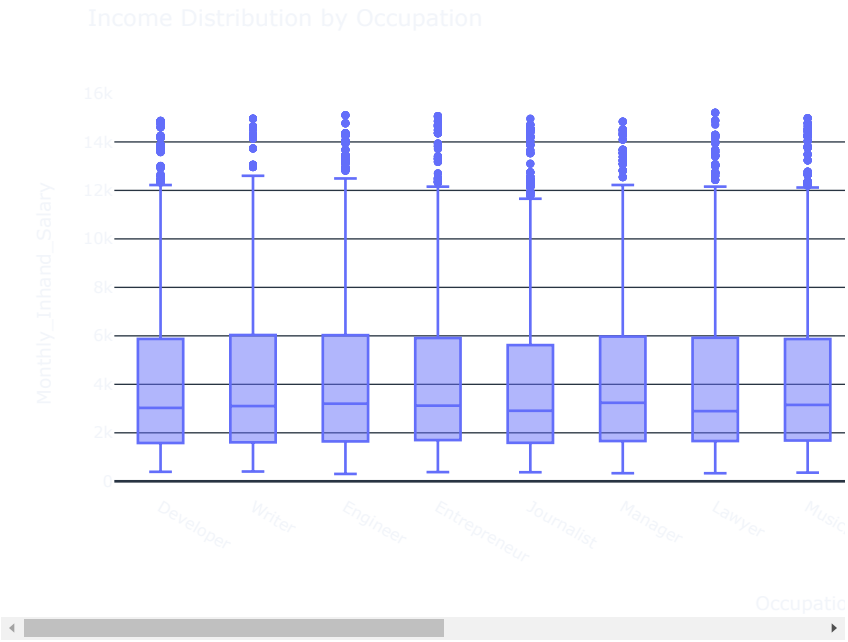
Occupation

```
px.histogram(df, x='Occupation', color='Credit_Score', barmode='group', histfunc='count',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1200, height=500, title='Credit Score Distribution by Occupation')
```



- From the graph, there is no clear difference in the distribition of Score between Occupation. We will look into income distribution between these careers before we can conclude that higher skilled jobs have an effect on Credit Score.

```
px.box(df, y='Monthly_Inhand_Salary', x='Occupation',
       width=1200, height=500, title='Income Distribution by Occupation')
```

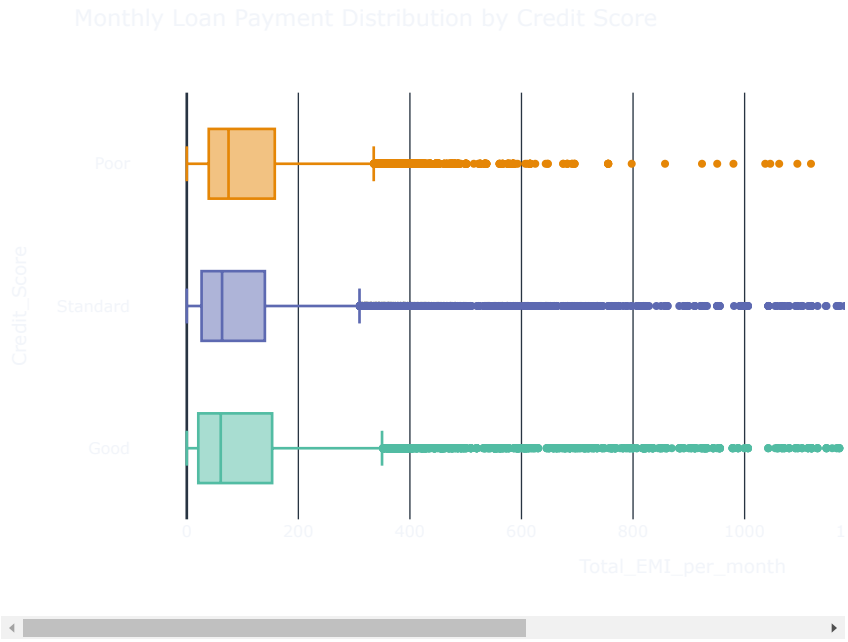


- Surprisingly there is almost no difference in Income between the jobs.

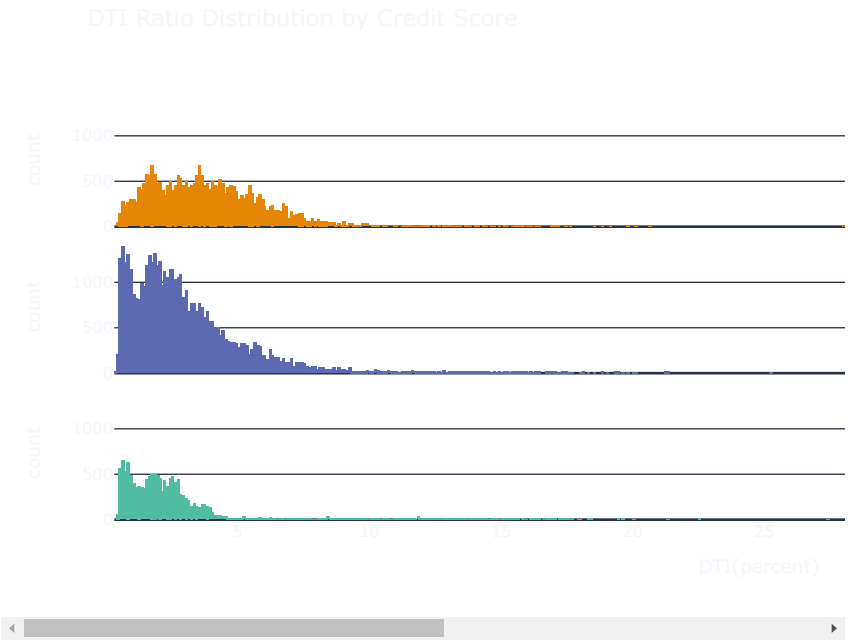
Debt to Income Ratio

"The debt-to-income (DTI) ratio is the percentage of your gross monthly income that goes to paying your monthly debt payments and is used by lenders to determine your borrowing risk." - https://www.investopedia.com/terms/d/dti.asp

```
fig = px.box(df, x='Total_EMI_per_month', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Monthly Loan Payment Distribution by Credit Score')
fig.update_layout(showlegend=False)
```



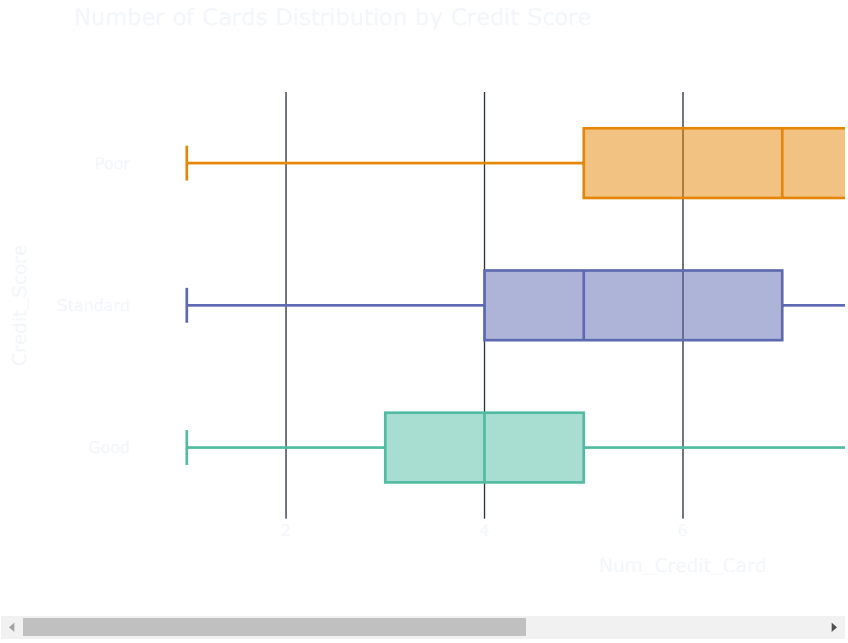

```
# Calculate the Debt to Income ratio onto a new column
df['DTI'] = df['Total_EMI_per_month'] / df['Monthly_Inhand_Salary'] * 100
# Shows only people who are paying debts
px.histogram(df[(df['DTI']!=0) & (df['DTI']<=50)], x='DTI', color='Credit_Score', facet_row='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1200, height=500, title='DTI Ratio Distribution by Credit Score', labels={'DTI':'DTI(percent)'})
```



- There is very minimal changes in the distribution of DTI. Overall everyone's DTI are quite low. This may explains that this Bank is safely financing clients. This could also be Survival Bias as the clients data have already been through the Bank screening process which eliminated high risk individuals with high DTI ratio.

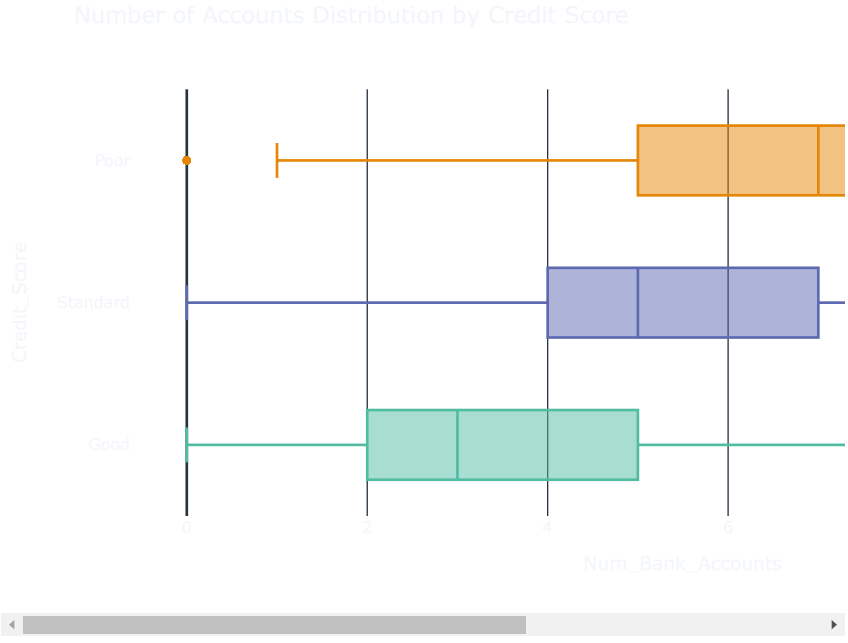
Accounts

```
fig = px.box(df, x='Num_Credit_Card', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Number of Cards Distribution by Credit Score')
fig.update_layout(showlegend=False)
```

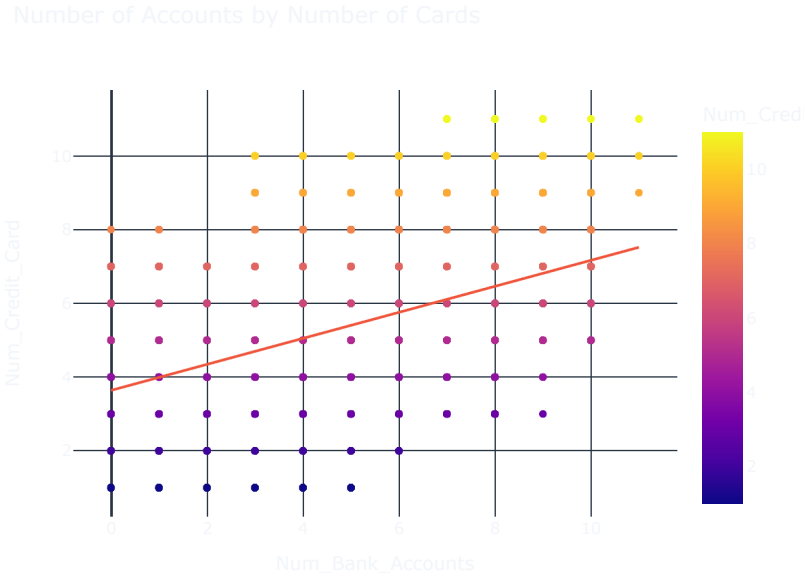


- High number of active credit cards will look riskier to lender as it is hard to manage and keep track of due date.

```
fig = px.box(df, x='Num_Bank_Accounts', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Number of Accounts Distribution by Credit Score')
fig.update_layout(showlegend=False)
```




```
px.scatter(df, x='Num_Bank_Accounts', y='Num_Credit_Card', color='Num_Credit_Card', trendline="ols",
           width=700, height=500, title='Number of Accounts by Number of Cards')
```



- Multiple bank accounts is not inherently bad for credit score.
- But it has positive correlation with other bad features.

Credit Behaviours

According to FICO Score 8, your Credit Score is based on five factors. 35% of your Score will weight on your Payment History. 30% of it is from the Amount you owed. Your Credit Age make up the next 15%. 10% is from your Credit Mix. And the last 10% is from Recent Activity which we will ignore since we do not have that information. - https://www.investopedia.com/articles/pf/10/credit-score-factors.asp

Payment History

Do you make payments on time? How often you miss payments? How many days past the due date you pay your bills? Payments made over 30 days late will negatively impact your credit scores.

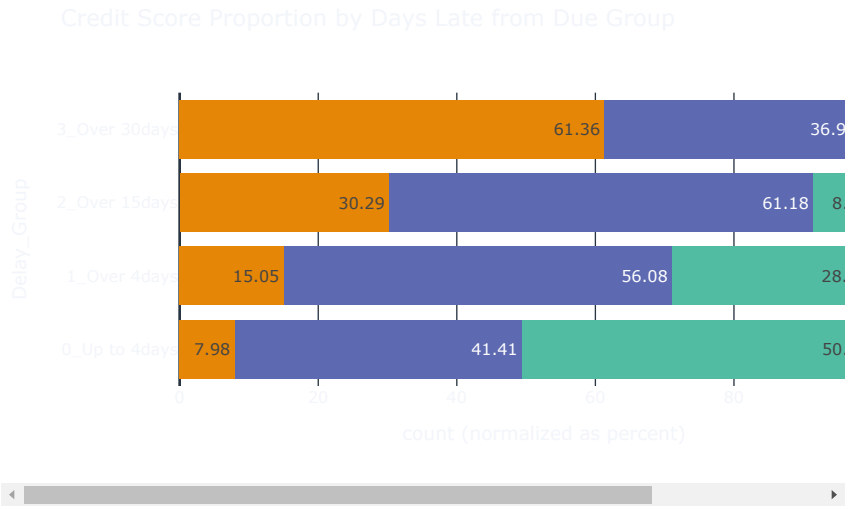
What we are expecting to see:

1. How often you missed your due date affect your Credit Score negatively.
2. Payments made 30 days late will greatly decrease your Credit Score.
3. A person with lots of loans will likely be late more often.

```
px.histogram(df, x='Delay_from_due_date', color='Credit_Score', facet_row='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1200, height=400, title='Days Late from Due Distribution by Credit Score', labels={'Delay_from_due_date':'Delay_from_due_d
```

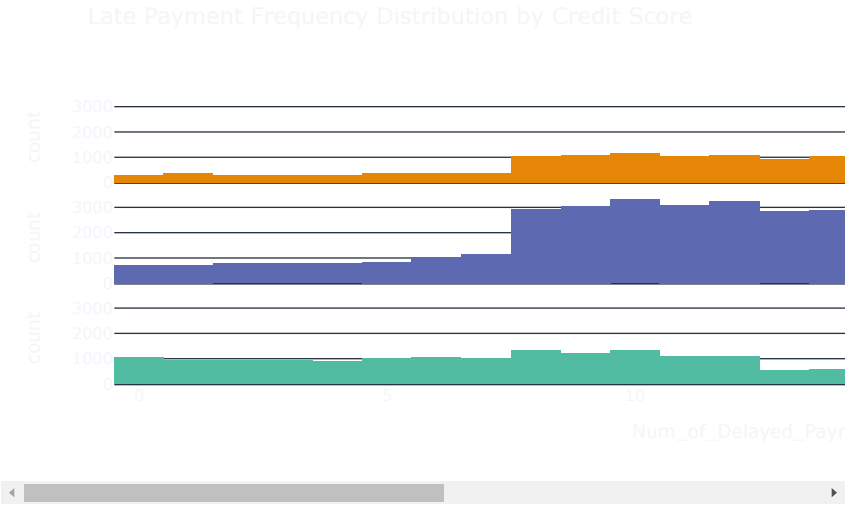
```
# Create Delay Group column
def assign_delay_group(row):
  if row['Delay_from_due_date'] <= 4:
    return '0_Up to 4days'
  elif row['Delay_from_due_date'] <= 15:
    return '1_Over 4days'
  elif row['Delay_from_due_date'] <= 30:
    return '2_Over 15days'
  else:
    return '3_Over 30days'

df['Delay_Group'] = df.apply(assign_delay_group, axis=1)
```

```
px.histogram(df.sort_values(['Delay_Group', 'Credit_Score']), y='Delay_Group', color='Credit_Score', barnorm='percent', text_auto='.2f',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=800, height=400, title='Credit Score Proportion by Days Late from Due Group')
```



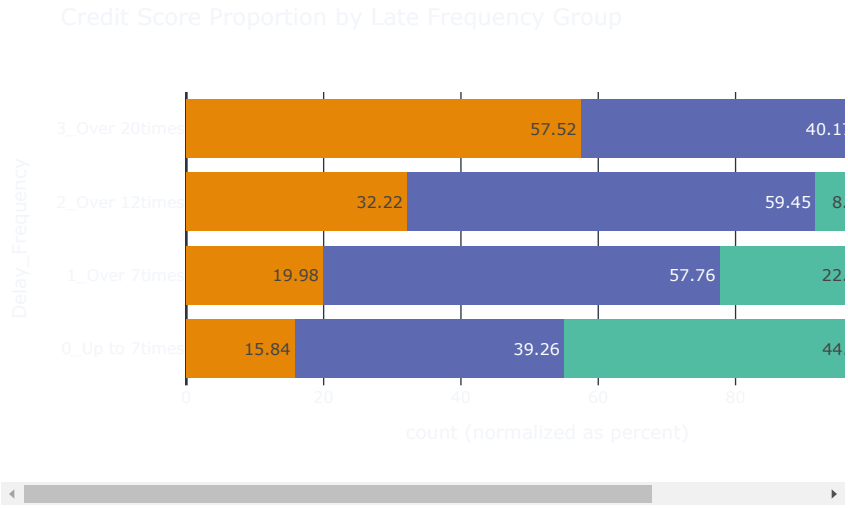
- We can see a clear gradual increase in Poor Scorer as the number of days over payment date increase and a big jump after 30 days. This is inline to what we assumed as it is a very important signifier for reliability.

```
px.histogram(df, x='Num_of_Delayed_Payment', color='Credit_Score', facet_row='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1200, height=400, title='Late Payment Frequency Distribution by Credit Score')
```



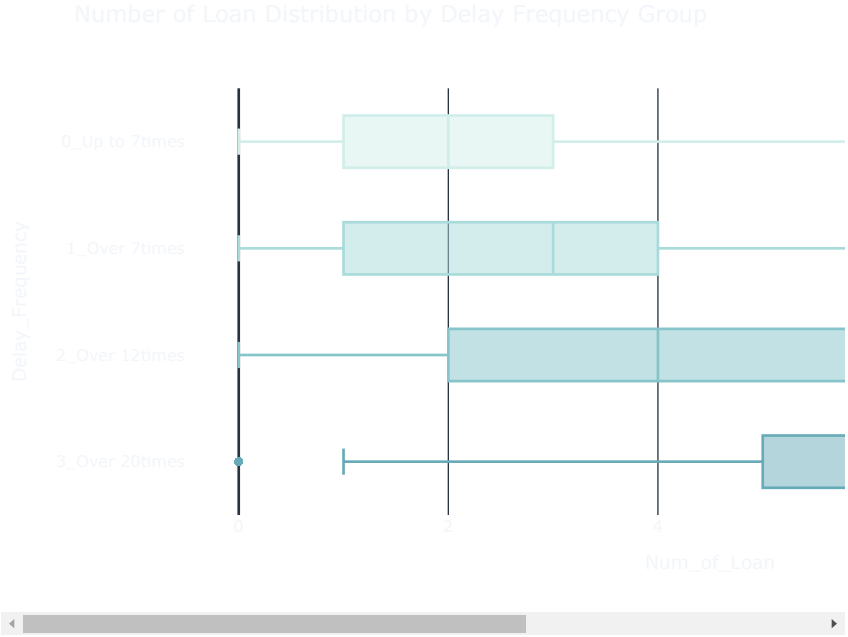

```
# Create Delay Habit column
def assign_delay_freq(row):
  if row['Num_of_Delayed_Payment'] <= 7:
    return '0_Up to 7times'
```

```
  elif row['Num_of_Delayed_Payment'] <= 12:
    return '1_Over 7times'
  elif row['Num_of_Delayed_Payment'] <= 20:
    return '2_Over 12times'
  else:
    return '3_Over 20times'

df['Delay_Frequency'] = df.apply(assign_delay_freq, axis=1)


px.histogram(df.sort_values(['Delay_Frequency','Credit_Score']), y='Delay_Frequency', color='Credit_Score', barnorm='percent', text_auto='.2f
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=800, height=400, title='Credit Score Proportion by Late Frequency Group')
```



- The findings here is very similar to our findings for Number of Days Late for loan payment. It shows individuals who frequently late for their payment is marked down in their score greatly.

```
fig = px.box(df.sort_values('Delay_Frequency'), x='Num_of_Loan', y='Delay_Frequency', color='Delay_Frequency',
             color_discrete_sequence=px.colors.sequential.Teal,
             width=1000, height=500, title='Number of Loan Distribution by Delay Frequency Group')
fig.update_layout(showlegend=False)
```



- From the above box plot, we find that a person holding many loans will more likely be late for their monthly payment. The bank should prevent people with 4 loans from getting more as they are generally more irresponsible with their finance which carries more risk.

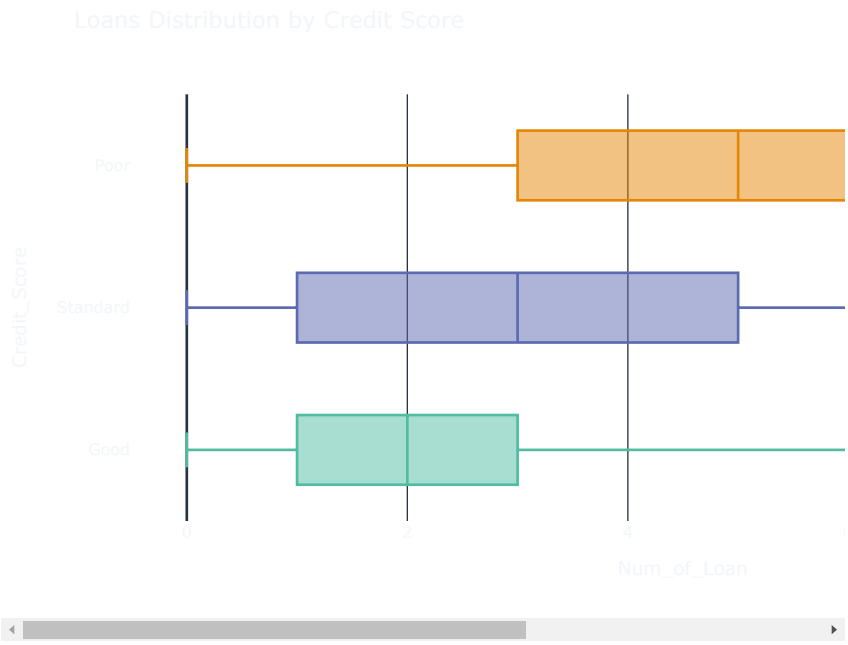
Debt

How much you owe? What type of loans do you have? What is your credit-utilization ratio?

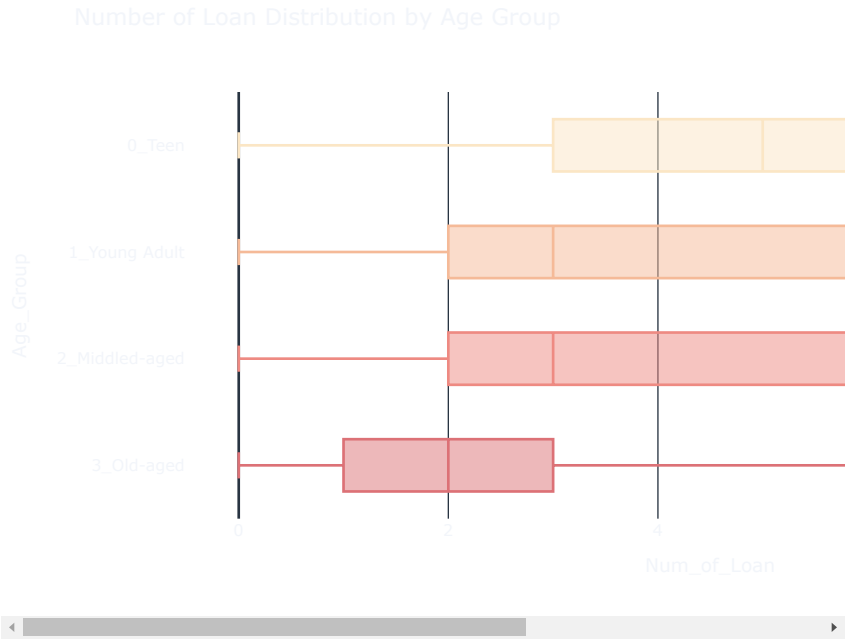
What we are expecting to see:

1. The more a person owes the worse their Credit Score is.
2. Personal loans will negatively affect Credit Score because a hard credit inquiry will be made.
3. Higher Credit-Utilization ratio will decrease Credit Score.

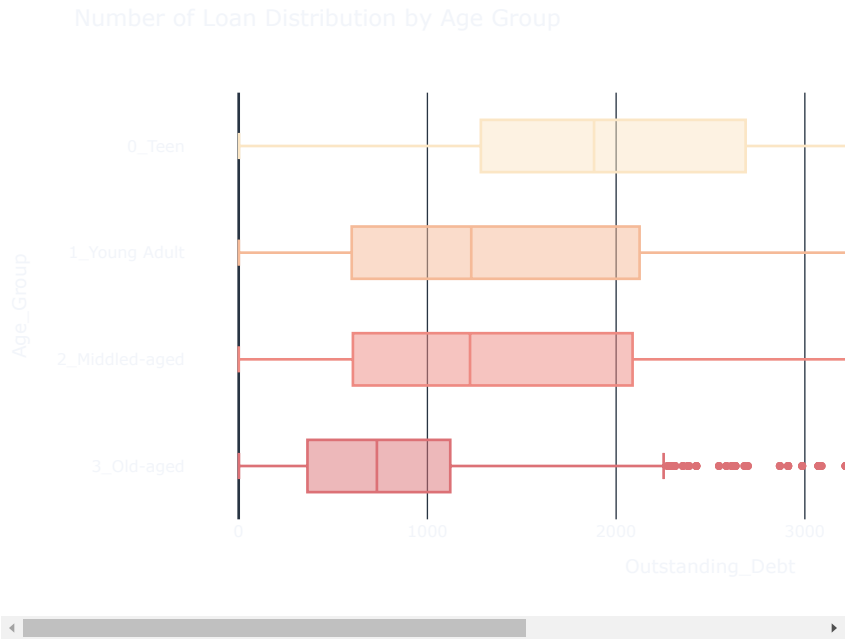
```
fig = px.box(df, x='Num_of_Loan', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Loans Distribution by Credit Score')
fig.update_layout(showlegend=False)
```




```
fig = px.box(df.sort_values('Age_Group'), x='Num_of_Loan', y='Age_Group', color='Age_Group',
             color_discrete_sequence=px.colors.sequential.Burgyl,
             width=1000, height=500, title='Number of Loan Distribution by Age Group')
fig.update_layout(showlegend=False)
```

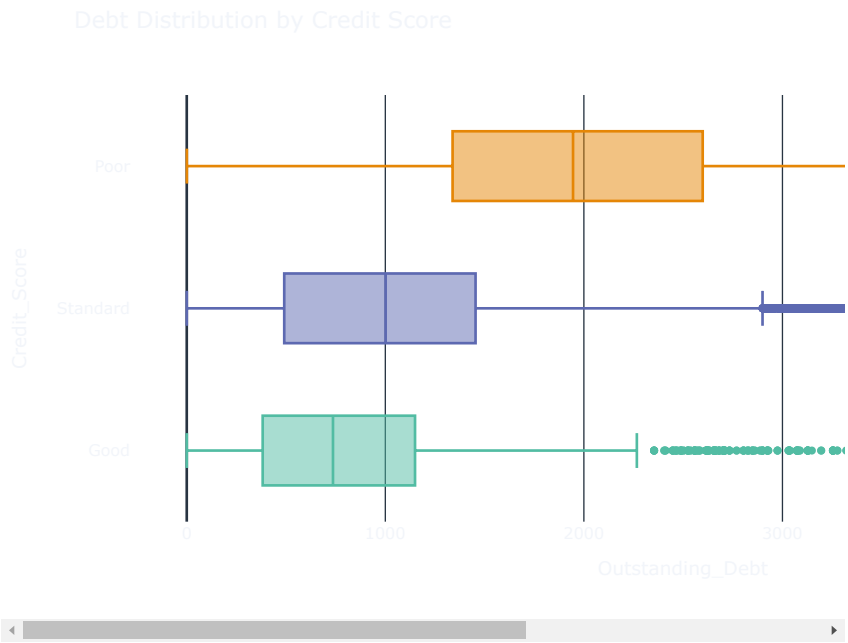


```
fig = px.box(df.sort_values('Age_Group'), x='Outstanding_Debt', y='Age_Group', color='Age_Group',
             color_discrete_sequence=px.colors.sequential.Burgyl,
             width=1000, height=500, title='Number of Loan Distribution by Age Group')
fig.update_layout(showlegend=False)
```



- Interestingly, eventhough younger people has less income, they take on more debts than older people. This is because younger people will likely have more ability to pay off their debts as they have longer to live. Also, older individuals will have already paid off most of their loans.
- Highest debt is only around 5000 which is very low if it is America (https://www.bankrate.com/personal-finance/debt/average-american-debt/#at-a-glance).

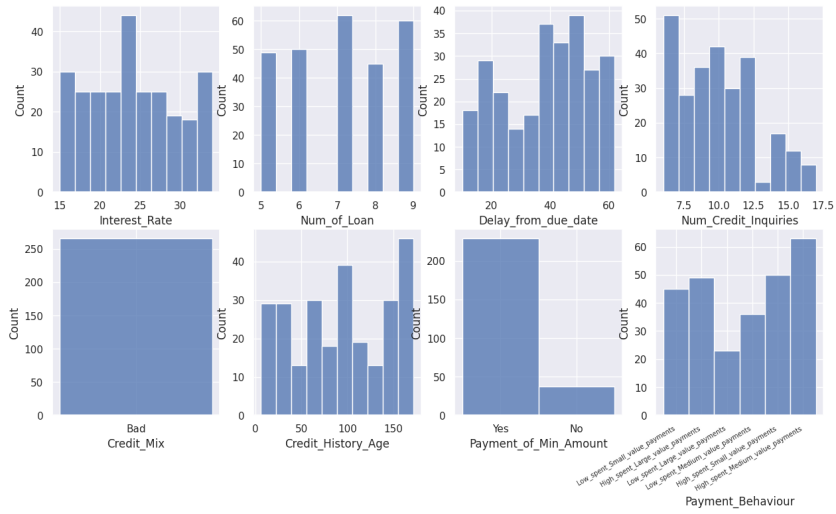
```
fig = px.box(df, x='Outstanding_Debt', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Debt Distribution by Credit Score')
fig.update_layout(showlegend=False)
```



- The distribution of Debt by Credit Score is as expected, there is a clear negative correlation. Although there are a significant number of outliers for all groups.

```
# Get outliers Good Scorer with very high Debt
good_outliers = df[(df['Credit_Score']=='Good') & (df['Outstanding_Debt']>=3000)]
# Plot distribution for quick overview of outliers
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(15,8))
sns.histplot(good_outliers['Interest_Rate'], ax=axes[0, 0])
sns.histplot(good_outliers['Num_of_Loan'], ax=axes[0, 1])
sns.histplot(good_outliers['Delay_from_due_date'], ax=axes[0, 2])
sns.histplot(good_outliers['Num_Credit_Inquiries'], ax=axes[0, 3])
sns.histplot(good_outliers['Credit_Mix'], ax=axes[1, 0])
sns.histplot(good_outliers['Credit_History_Age'], ax=axes[1, 1])
sns.histplot(good_outliers['Payment_of_Min_Amount'], ax=axes[1, 2])
sns.histplot(good_outliers['Payment_Behaviour'], ax=axes[1, 3])
axes[1,3].set_xticklabels(axes[1,3].get_xticklabels(), rotation=30, fontsize=7, ha='right')
plt.show();
```

```
<ipython-input-46-30c7c748ce1e>:13: UserWarning:

FixedFormatter should only be used together with FixedLocator
```




```
print(f"Percentage of Good Outliers: {len(good_outliers) / len(df) * 100}%")
```

```
Percentage of Good Outliers: 0.266%
```

- From the above overview, these Good Scorer seems to have the statistic of a Poor Scorer. As before we should leave them in to maintain non-bias for model building.
- However, if these records are removed, it is likely that we will increase precision score of Good label for our models(less likely for similar statistic of a Poor Scorer to be labeled as Good - decreasing in False Positive).

- As a Banker, we would prioritize protecting our money, therefore avoid incorrectly labeling a Poor Scorer as Good. So it may be good to remove these outliers when fine tuning for model training.

```
# Loan types are in one column separated by commas
df['Type_of_Loan'].unique()
```

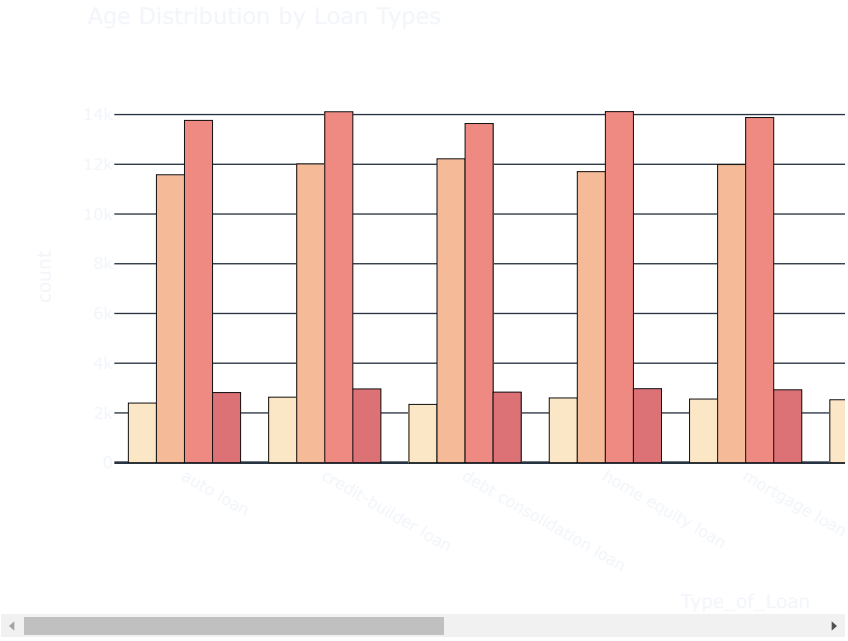
```
array(['auto loan,credit-builder loan,home equity loan,mortgage loan',
       'mortgage loan,personal loan,debt consolidation loan,auto loan,payday loan,credit-builder loan,credit-builder loan,personal
loan',
       'mortgage loan,student loan,not specified', ...,
       'debt consolidation loan,mortgage loan,credit-builder loan,debt consolidation loan',
       'mortgage loan,personal loan,not specified,payday loan',
       'payday loan,credit-builder loan,payday loan,mortgage loan'],
      dtype=object)
```

```
# Function for dummifying data with Melt Option
def dummify_string(df, long_col, drop_col=None, separator=",", melt=False):
  dummies = df[long_col].str.get_dummies(sep=separator)
  # Drop unwanted dummified data if any
  if drop_col != None:
    dummies.drop(columns=drop_col, inplace=True)
  # Drop original column and merge dummy df with main df
  temp_df = df.drop(columns=long_col)
  merge_df = temp_df.merge(dummies, left_index=True, right_index=True)
  # Return dummified df if not melt
  if not melt:
    return merge_df
  # Define id_vars and value_vars if melt
  keep_col = temp_df.columns
  melt_col = dummies.columns
  # Melt
  melt_df = merge_df.melt(id_vars=keep_col,
                          value_vars=melt_col,
                          var_name=long_col,
                          value_name=long_col + "_True")
  return melt_df
```

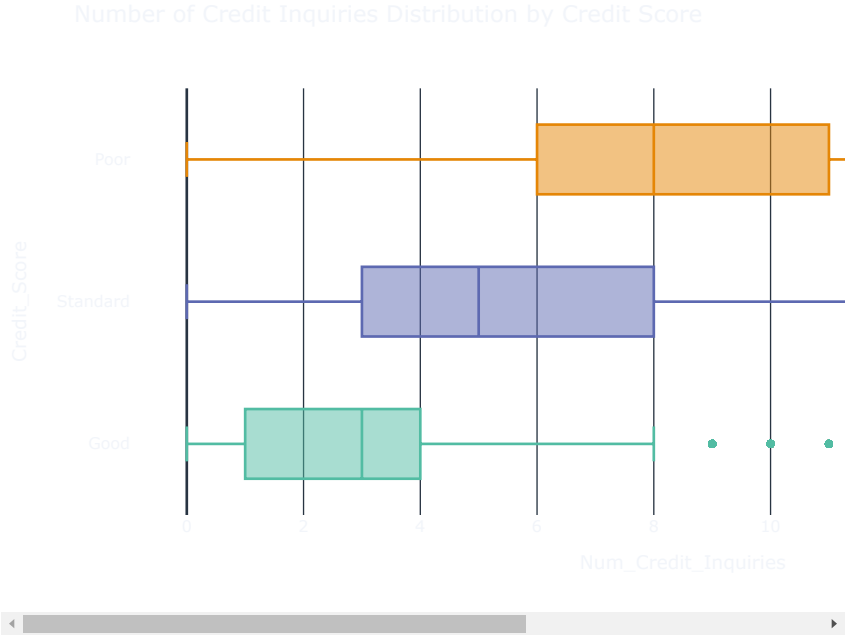
```
# Dummify and Melt Loan Type Column
melted_ltype_df = dummify_string(df,"Type_of_Loan",'No Data', ",", True)
ltype_df = melted_ltype_df.groupby(['Type_of_Loan','Credit_Score'])['Type_of_Loan_True'].sum().reset_index()
px.histogram(ltype_df, x='Type_of_Loan', y ='Type_of_Loan_True', color='Credit_Score',
             barmode='stack', histfunc='sum',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1200, height=500, title='Credit Score Distribution by Loan Types', labels={'Type_of_Loan_True':'count'})
```

- There doesn't seem to be any correlation between Loan Types and Credit Score. They are all relatively evenly distributed.

```
age_ltype_df = melted_ltype_df.groupby(['Age_Group','Type_of_Loan'])['Type_of_Loan_True'].sum().reset_index()
px.bar(age_ltype_df, x='Type_of_Loan', y='Type_of_Loan_True', color='Age_Group',
             barmode='group',
             color_discrete_sequence=px.colors.sequential.Burgyl,
             width=1200, height=500, title='Age Distribution by Loan Types', labels={'Type_of_Loan_True':'count'})
```

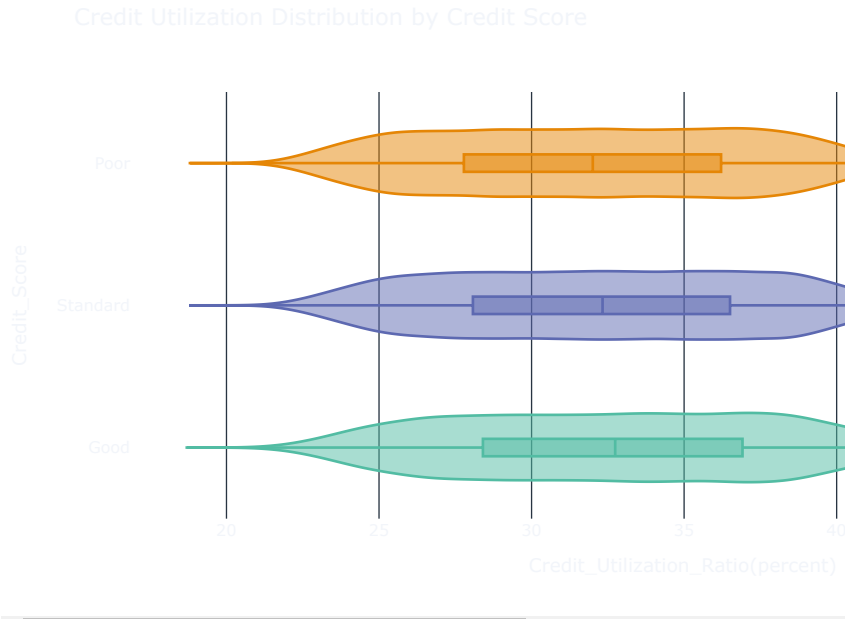



```
fig = px.box(df, x='Num_Credit_Inquiries', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Number of Credit Inquiries Distribution by Credit Score')
fig.update_layout(showlegend=False)
```



- But the negative effect of Credit Inquiries holds true. However, you can expect a person that take out lots of loan to have received many hard inquiries. Therefore, this may not be a direct causation.

```
fig = px.violin(df, x='Credit_Utilization_Ratio', y='Credit_Score', color='Credit_Score', box=True,
                color_discrete_sequence=px.colors.qualitative.Vivid,
                width=1000, height=500, title='Credit Utilization Distribution by Credit Score', labels={'Credit_Utilization_Ratio':'Credit_U
fig.update_layout(showlegend=False)
```



- Strangely, credit utilization ratio has an even distribution across all Score when we expected it to have a negative influence. Everybody in this dataset seems to be a very responsible spender.

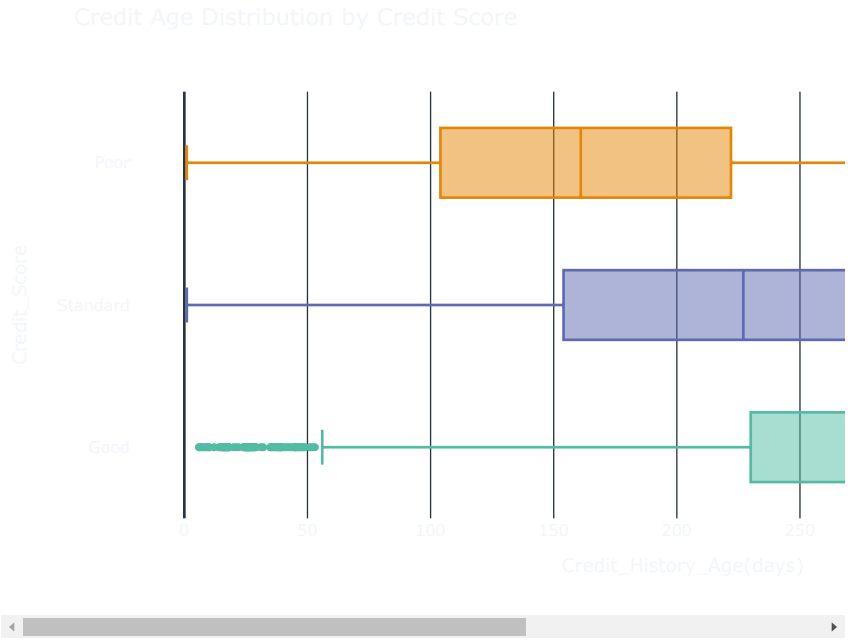
Credit Age

Do you have credit account? How long have you opened your credit account? Do you have a history of making timely payments?

What we are expecting to see:

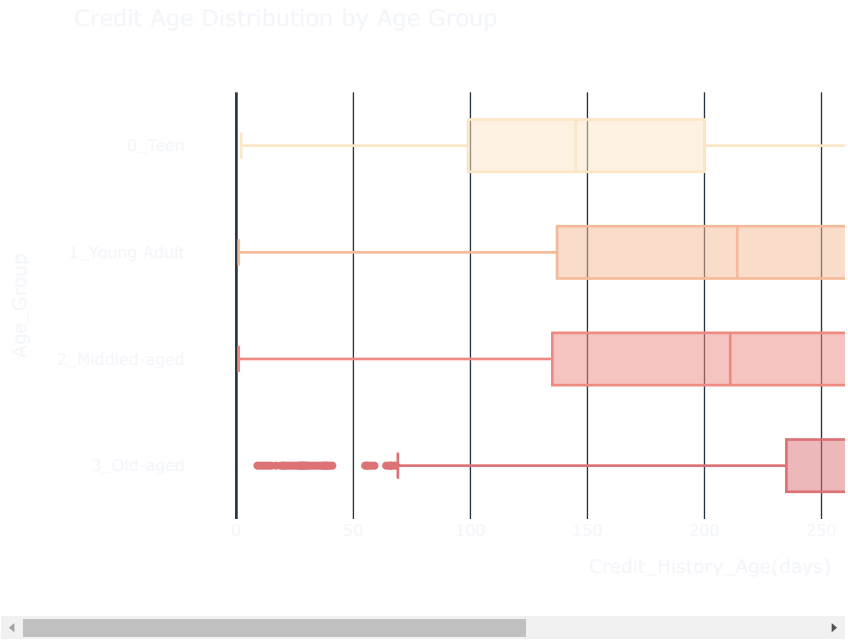
1. A person with longer Credit History will have better Score as they are generally more reliable, the Bank will have more information on them.
2. An individual who spends responsibly and make more than minimum payments will recieve a better Score.

```
fig = px.box(df, x='Credit_History_Age', y='Credit_Score', color='Credit_Score',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=1000, height=500, title='Credit Age Distribution by Credit Score', labels={'Credit_History_Age':'Credit_History_Age(days)'
fig.update_layout(showlegend=False)
```



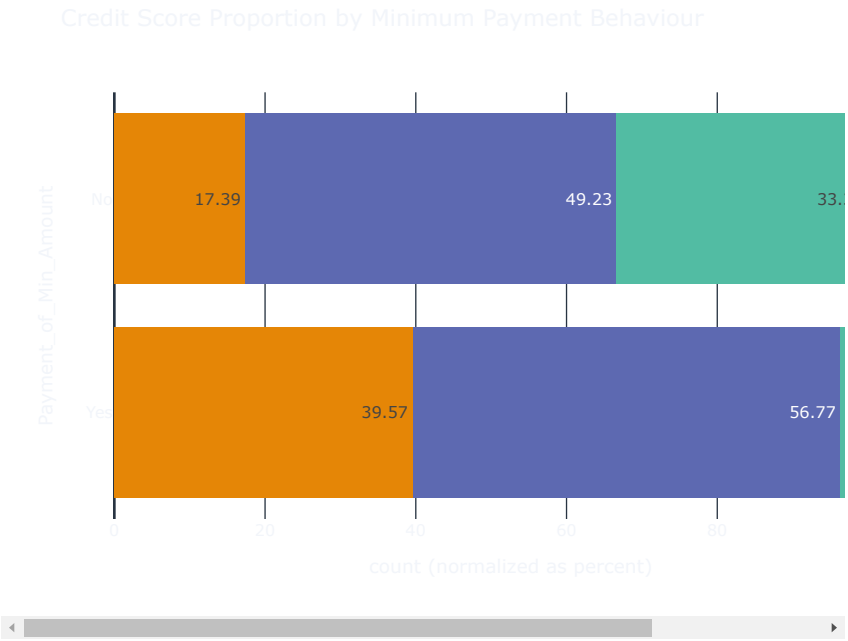
- As we expected, Credit History Age has a positive correlation with Credit Score. The more an individual uses their Credit Account, the more information the Bank has to categorize them. And long term customers are generally more reliable.

```
fig = px.box(df.sort_values('Age_Group'), x='Credit_History_Age', y='Age_Group', color='Age_Group',
             color_discrete_sequence=px.colors.sequential.Burgyl,
             width=1000, height=500, title='Credit Age Distribution by Age Group', labels={'Credit_History_Age':'Credit_History_Age(days)'})
fig.update_layout(showlegend=False)
```



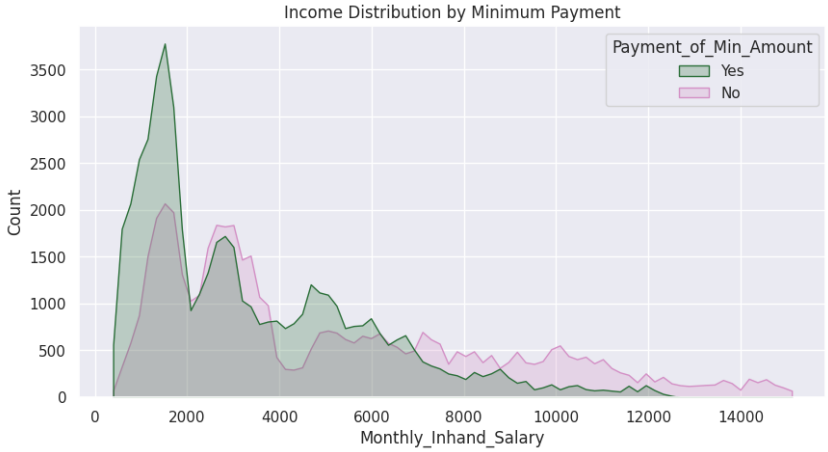
- Also as expected, younger people will unlikely to have a long Credit History. Here we can conclude that, age is not an inherent factor that negatively impact Credit Score but an indicator that generalized their position for other factors such as Income, Loan Amount and Credit History.

```
px.histogram(df, y='Payment_of_Min_Amount', color='Credit_Score', barnorm='percent', text_auto='.2f',
             color_discrete_sequence=px.colors.qualitative.Vivid,
             width=800, height=500, title='Credit Score Proportion by Minimum Payment Behaviour')
```

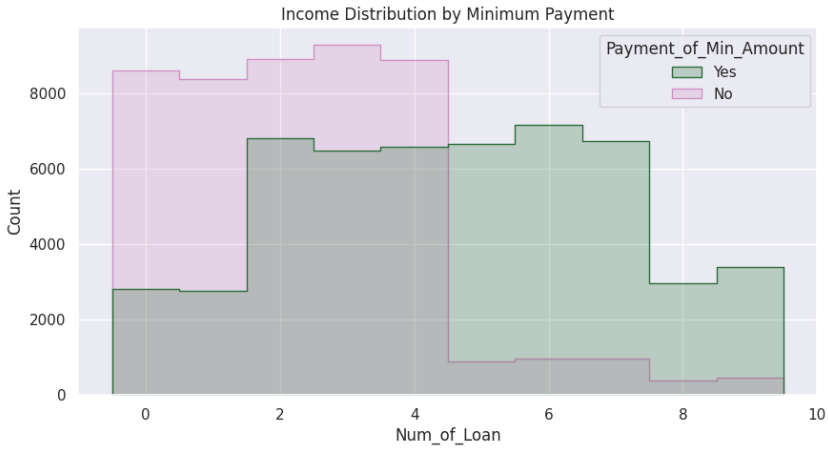


- People who only pays the minimum amount each month will not have a good Score.

```
plt.figure(figsize=(10,5))
plt.title('Income Distribution by Minimum Payment')
sns.histplot(df, x='Monthly_Inhand_Salary', hue='Payment_of_Min_Amount', element='poly',
             palette='cubehelix');
```

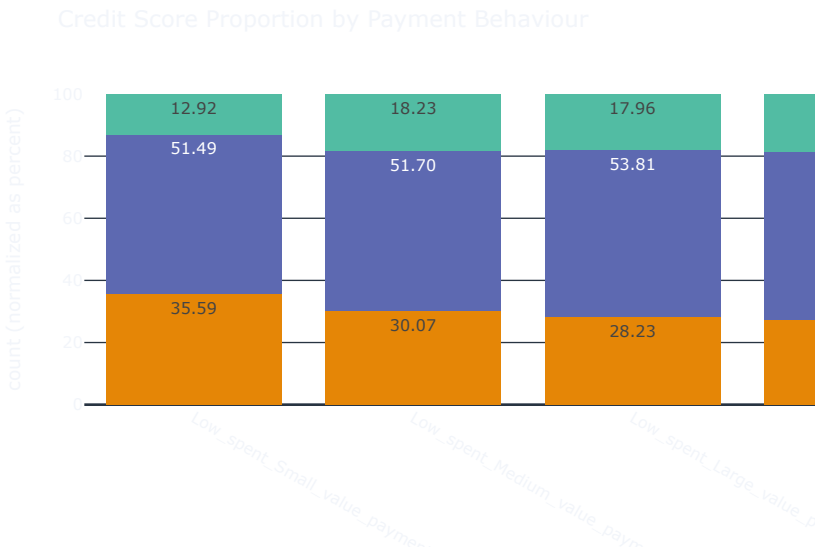



```
plt.figure(figsize=(10,5))
plt.title('Income Distribution by Minimum Payment')
sns.histplot(df, x='Num_of_Loan', hue='Payment_of_Min_Amount', element='step',
             palette='cubehelix', discrete=True);
```



- Minimum payment is an indicator for low income and high debt.

```
fig = px.histogram(df, x='Payment_Behaviour', color='Credit_Score', barmode='stack', barnorm='percent', text_auto='.2f',
                   color_discrete_sequence=px.colors.qualitative.Vivid,
                   width=1200, height=500, title='Credit Score Proportion by Payment Behaviour')
fig.update_xaxes(categoryorder='array',categoryarray=['Low_spent_Small_value_payments', 'Low_spent_Medium_value_payments','Low_spent_Large_va
                                                      'High_spent_Small_value_payments','High_spent_Medium_value_payments','High_spent_Large_
```



- From the graph above, there are small increase in better Credit Score as spending and payment behaviour grow from low to high. However, the changes are too minimal to be a good indicator for Credit Score.

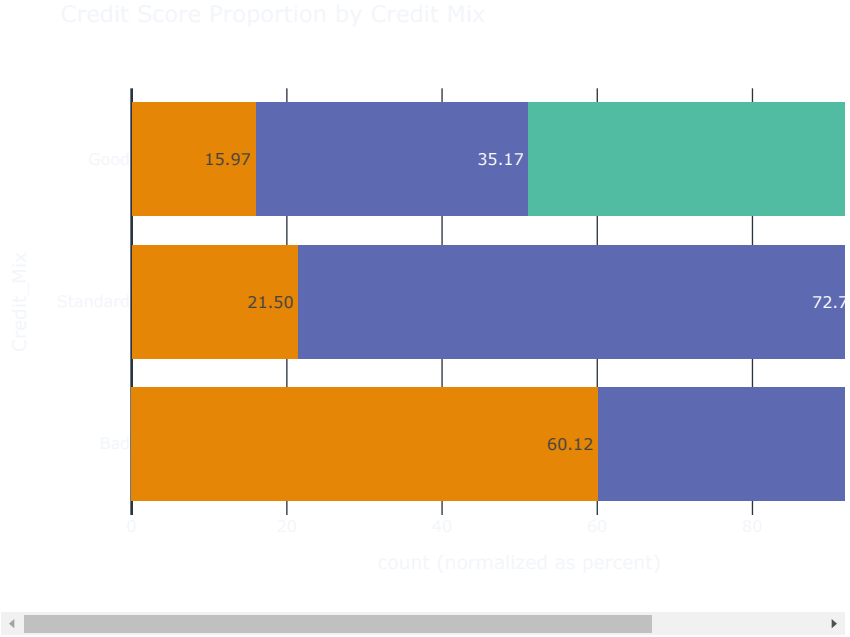
Credit Mix

Do you have mix of many types of loans?

What we are expecting to see:

1. Good mix of loan types will improve Credit Score.

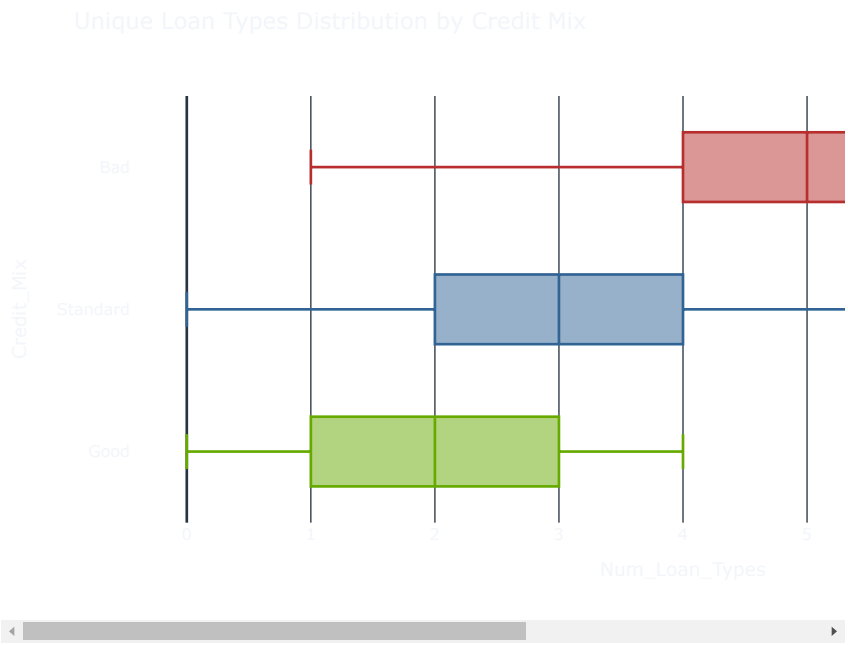
```
fig = px.histogram(df, y='Credit_Mix', color='Credit_Score', barmode='stack', barnorm='percent', text_auto='.2f',
                   color_discrete_sequence=px.colors.qualitative.Vivid,
                   width=800, height=500, title='Credit Score Proportion by Credit Mix')
fig.update_yaxes(categoryorder='array',categoryarray=['Bad','Standard','Good'])
```



- From the graph, it is clear that Good mix of accounts will indeed gives favorable Credit Score. We know this is true because loan types alone does not affect Score.

```
dummy_ltype_df = dummify_string(df,"Type_of_Loan",'No Data', ",", False)
dummy_ltype_df['Num_Loan_Types'] = dummy_ltype_df.iloc[:,-9:].sum(axis=1)
```

```
fig = px.box(dummy_ltype_df, y='Credit_Mix', x='Num_Loan_Types', color='Credit_Mix',
             color_discrete_sequence=px.colors.qualitative.G10_r,
             width=1000, height=500, title='Unique Loan Types Distribution by Credit Mix')
fig.update_layout(showlegend=False, yaxis={'categoryorder':'array', 'categoryarray':['Good','Standard','Bad']})
```



- We see that managing many loan types is not always a good thing. As we have found before, too many loans result in worse credit score.
- Similar distribution to Num Loans on Credit Scores.

MODELS BUILDING

- Target label is imbalanced.
- Data is mostly non Gaussian distribution.
- Data has a lot of overlapping between classes.
- Features are not independent.
- Linear models is ineffective because complex non-linear relationship. SVM is time consuming because of large dataset size.
- Features are not text data and independent of each other so Naive Bayes is ineffective.

Data Processing

```
# Make copy of cleaned dataframe and drop Customer_ID and Interest Rate because it is set based on Credit Score
processing_df = main.drop(columns=['Customer_ID','Interest_Rate'])
# Dummify Type_of_Loan
processing_df = dummify_string(processing_df, long_col="Type_of_Loan", drop_col='No Data', separator=",", melt=False)
# Remap target feature
processing_df['Credit_Score'] = processing_df['Credit_Score'].map({'Poor':0,
                                                                    'Standard':1,
                                                                    'Good':2})
```

```
processing_df.select_dtypes(include="object").head()
```

	Month	Occupation	Credit_Mix	Payment_of_Min_Amount	Payment_Beh
0	Jan	Scientist	Good	No	High_spent_Small_value_pa
1	Feb	Scientist	Good	No	Low_spent_Large_value_pa
2	Mar	Scientist	Good	No	Low_spent_Medium_value_pa
3	Apr	Scientist	Good	No	Low_spent_Small_value_pa
4	May	Scientist	Good	No	High_spent_Medium_value_pa

```
processing_df.select_dtypes(exclude="object").head()
```

	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card
0	23.0	19114.12	1824.843333	3.0	4.0
1	23.0	19114.12	1824.843333	3.0	4.0
2	23.0	19114.12	1824.843333	3.0	4.0
3	23.0	19114.12	1824.843333	3.0	4.0
4	23.0	19114.12	1824.843333	3.0	4.0

Split Data

```
# Split target label from features
X = processing_df.drop(columns='Credit_Score')
y = processing_df['Credit_Score']
# Split into train and test set 20%
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=31)
```

Transform Data

Encoding

```
cat_features = X.select_dtypes(include="object").columns
cat_features
```

```
Index(['Month', 'Occupation', 'Credit_Mix', 'Payment_of_Min_Amount',
       'Payment_Behaviour'],
      dtype='object')
```

```
from sklearn.preprocessing import OneHotEncoder
# Set encoder to get binary category as only one column
encoder = OneHotEncoder(handle_unknown="ignore", drop='if_binary')
# Fit and transform
encoder.fit(X_train[cat_features])
X_cat_train_encoded = encoder.transform(X_train[cat_features])
X_cat_test_encoded = encoder.transform(X_test[cat_features])
# Convert to smallest array type
X_cat_train_encoded = X_cat_train_encoded.toarray()
X_cat_test_encoded = X_cat_test_encoded.toarray()
```

```
# Get encoded columns' names
encoded_cat_features = encoder.get_feature_names_out()
```

Scaling

```
num_features = X.select_dtypes(exclude="object").columns[:16]
num_features
```

```
Index(['Age', 'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Num_of_Loan', 'Delay_from_due_date',
       'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Outstanding_Debt', 'Credit_Utilization_Ratio',
       'Credit_History_Age', 'Total_EMI_per_month', 'Amount_invested_monthly',
       'Monthly_Balance'],
      dtype='object')
```

```
from sklearn.preprocessing import RobustScaler
rscaler = RobustScaler()
# Fit and transform
rscaler.fit(X_train[num_features])
```

```
X_num_train_scaled = rscaler.transform(X_train[num_features])
X_num_test_scaled = rscaler.transform(X_test[num_features])
```

Final process

Recombine processed data

```
remaining_features = X.drop(columns=num_features.append(cat_features)).columns
remaining_features
```

```
Index(['auto loan', 'credit-builder loan', 'debt consolidation loan',
       'home equity loan', 'mortgage loan', 'not specified', 'payday loan',
       'personal loan', 'student loan'],
      dtype='object')
```

```
# Get values of remaining columns
X_remain_train = X_train[remaining_features].values
X_remain_test = X_test[remaining_features].values
# Concatenate processed data
X_train_final = np.concatenate([X_num_train_scaled, X_cat_train_encoded, X_remain_train], axis=1)
X_test_final = np.concatenate([X_num_test_scaled, X_cat_test_encoded, X_remain_test], axis=1)
```

```
# Names of all columns for processed dataframe
cols_name = np.concatenate([num_features, encoded_cat_features, remaining_features], axis=0)
X_train_final = pd.DataFrame(X_train_final, columns=cols_name)
X_test_final = pd.DataFrame(X_test_final, columns=cols_name)
```

Training

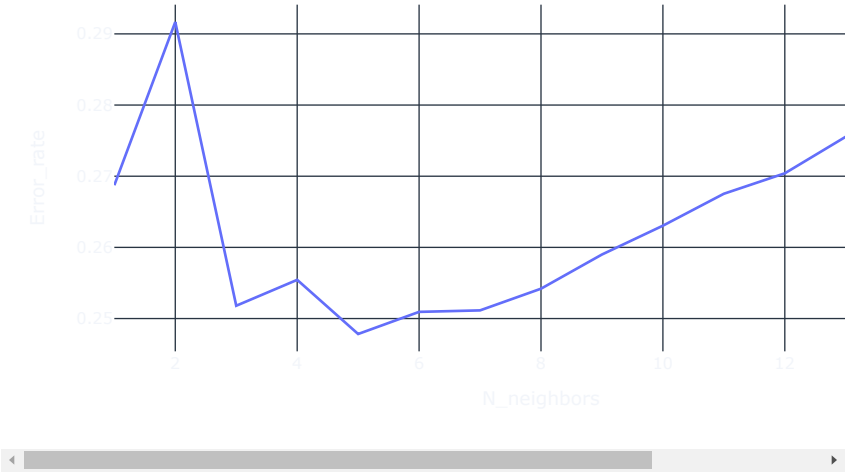
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.ensemble import VotingClassifier

from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV, cross_val_score
from sklearn.metrics import f1_score
```

```
def report_scores(model, show=True):
  # Calculate accuracy score
  train_ascore = model.score(X_train_final, y_train)
  test_ascore = model.score(X_test_final, y_test)
  # Get model prediction
  y_train_pred = model.predict(X_train_final)
  y_test_pred = model.predict(X_test_final)
  # Calculate f1 score
  train_fscore = f1_score(y_train, y_train_pred, average='macro')
  test_fscore = f1_score(y_test, y_test_pred, average='macro')
  if show:
    # Print train and test accuracy score
    print(f"Train Accuracy Score: {train_ascore}")
    print(f"Test Accuracy Score: {test_ascore}\n")
    # Print train and test macro f1 score
    print(f"Train Macro F1-Score: {train_fscore}")
    print(f"Test Macro F1-Score: {test_fscore}")
  else:
    return [train_ascore, test_ascore, train_fscore, test_fscore]
```

Nearest Neightbors Classifier

```
# Calculate the error rate from cross validation on n_neighbors
error_rate = []
for i in range(1,16):
    knc = KNeighborsClassifier(n_neighbors=i, n_jobs=-1)
    score = cross_val_score(knc, X_train_final, y_train, scoring='f1_macro', cv=5)
    error_rate.append(1-score.mean())
```

```
elbow_df = pd.DataFrame(error_rate,
            columns=['Error_rate'])
px.line(elbow_df, y='Error_rate', x=range(1,16),
        width=800, height=400, labels={'x':'N_neighbors'})
```



- Using the elbow method, we can choose the best value for parameter n_neighbors is 5.

```
# Build and fit
knc = KNeighborsClassifier(n_neighbors=5, n_jobs=-1)
knc.fit(X_train_final, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_jobs=-1)
```

```
report_scores(knc)
```

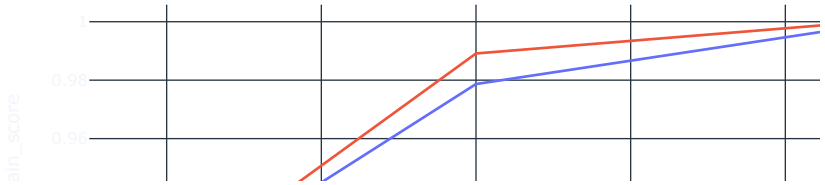
```
Train Accuracy Score: 0.8522125
Test Accuracy Score: 0.7829

Train Macro F1-Score: 0.8440282473336849
Test Macro F1-Score: 0.7670710177901173
```

- This model has quite high training score and relatively good test score.

Random Forest Classifier

```
# Using HalvingGridSearch to tune parameters
rfc = RandomForestClassifier(n_jobs=-1, random_state=31)
param = {'criterion':('gini','entropy'),
         'max_depth':[15,20,25]}
search = HalvingGridSearchCV(rfc, param, cv=5, scoring='f1_macro',
                             random_state=31, n_jobs=-1).fit(X_train_final, y_train)
```

```
cv_df = pd.DataFrame(search.cv_results_)
px.line(cv_df, y='mean_train_score', x='param_max_depth', color='param_criterion',
       width=800, height=400)
```



- From the graph, we can easily select entropy as criterion and a max depth of 20. There's no need to go high because it would cause overfitting.

```
# Build and fit
rfc = RandomForestClassifier(criterion='entropy', max_depth=20, n_jobs=-1, random_state=31)
rfc.fit(X_train_final, y_train)
```

```
RandomForestClassifier(criterion='entropy', max_depth=20, n_jobs=-1,
                       random_state=31)
```

```
report_scores(rfc)
```

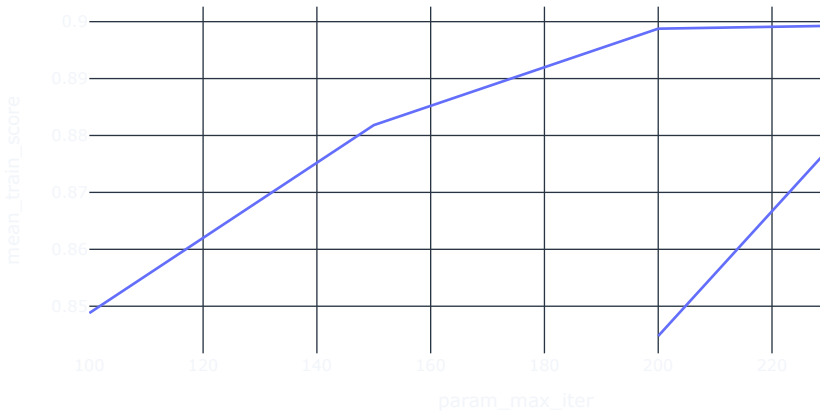
```
Train Accuracy Score: 0.9439375
Test Accuracy Score: 0.8032

Train Macro F1-Score: 0.9398307931158287
Test Macro F1-Score: 0.7934581827738872
```

- This model with tuned parameters returns relatively great results for both training and test sets.

Histogram Gradient Boosting Classifier

```
# Using HalvingGridSearch to tune parameters
hgbc = HistGradientBoostingClassifier(random_state=31)
param = {'max_iter':[100,150,200,250]}
search = HalvingGridSearchCV(hgbc, param, cv=5, scoring='f1_macro',
                             random_state=31, n_jobs=-1).fit(X_train_final, y_train)
```

```
cv_df = pd.DataFrame(search.cv_results_)
cv_df.drop(cv_df.tail(1).index, inplace=True)
px.line(cv_df, y='mean_train_score', x='param_max_iter',
       width=800, height=400)
```



- From the graph we can determine that 200 is the best max iteration for this model.

```
# Build and fit
hgbc = HistGradientBoostingClassifier(max_iter=200, random_state=31)
hgbc.fit(X_train_final, y_train)
```

```
HistGradientBoostingClassifier
HistGradientBoostingClassifier(max_iter=200, random_state=31)
```

```
report_scores(hgbc)
```

```
Train Accuracy Score: 0.84035
Test Accuracy Score: 0.78205

Train Macro F1-Score: 0.8356988477467943
Test Macro F1-Score: 0.7716962507988058
```

- This model has quite decent train and test score.

Voting Classifier

```
# Build & fit
vcf = VotingClassifier(estimators=[('NearestNeighbors', knc), ('RandomForest', rfc), ('HistGradientBoosting', hgbc)],
                       voting='soft', n_jobs=-1)
vcf.fit(X_train_final, y_train)
```

```
VotingClassifier
NearestNeighbors
KNeighborsClassifier
RandomForest
RandomForestClassifier
HistGradientBoosting
HistGradientBoostingClassifier
```

```
report_scores(vcf)
```

```
Train Accuracy Score: 0.89015
Test Accuracy Score: 0.8097

Train Macro F1-Score: 0.8862924904145874
Test Macro F1-Score: 0.800234207763597
```

- Test score improved by ~1% from Random Forest model. This is a very good score.

```
train_cv = cross_val_score(vcf, X_train_final, y_train, cv=5, scoring='f1_macro')
test_cv = cross_val_score(vcf, X_test_final, y_test, cv=5, scoring='f1_macro')
```

```
print(f"Train Cross-Val-Scores: {list(train_cv)}")
print(f"Test Cross-Val-Scores: {list(train_cv)}\n")
print(f"Train Cross-Val-Score Standard Deviation: {np.std(train_cv)}")
print(f"Test Cross-Val-Score Standard Deviation: {np.std(test_cv)}")
```

```
Train Cross-Val-Scores: [0.7894913944923537, 0.789944673479607, 0.7935883782391029, 0.7905056422565259, 0.7861834816943883]
Test Cross-Val-Scores: [0.7894913944923537, 0.789944673479607, 0.7935883782391029, 0.7905056422565259, 0.7861834816943883]

Train Cross-Val-Score Standard Deviation: 0.002364033698210143
Test Cross-Val-Score Standard Deviation: 0.00729006984015253
```

- There is a very tiny deviation in f1-score between samples.

```
models = {}
models['KNeighborsClassifer'] = report_scores(knc, show=False)
models['RandomForestClassifier'] = report_scores(rfc, show=False)
models['HistGradientBoostingClassifier'] = report_scores(hgbc, show=False)
models['VotingClassifier'] = report_scores(vcf, show=False)
```

```
models_df = pd.DataFrame.from_dict(models, orient='index').reset_index()
models_df.columns = ['Model','Train_Accuracy','Test_Accuracy','Train_F1','Test_F1']
models_df.round(2).sort_values('Test_F1', ascending=False).reset_index(drop=True)
```

	Model	Train_Accuracy	Test_Accuracy	Train_F1	Test_F1
0	VotingClassifier	0.89	0.81	0.89	0.80
1	RandomForestClassifier	0.94	0.80	0.94	0.79
2	KNeighborsClassifer	0.85	0.78	0.84	0.77

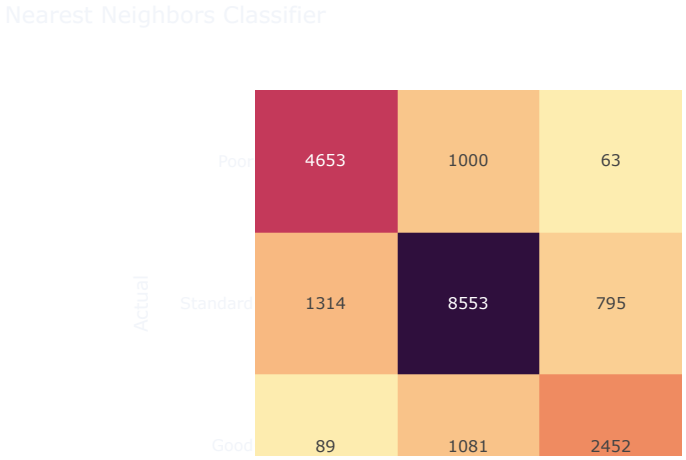
Evaluations

```
from sklearn.metrics import confusion_matrix, classification_report
!pip install shap
import shap
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting shap
  Downloading shap-0.41.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (572 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 572.6/572.6 kB 15.2 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from shap) (1.22.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from shap) (1.10.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from shap) (1.5.3)
Requirement already satisfied: tqdm>4.25.0 in /usr/local/lib/python3.10/dist-packages (from shap) (4.65.0)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-packages (from shap) (23.1)
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (from shap) (0.56.4)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python3.10/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from numba->shap) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->shap) (2022.7.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.41.0 slicer-0.0.7
```

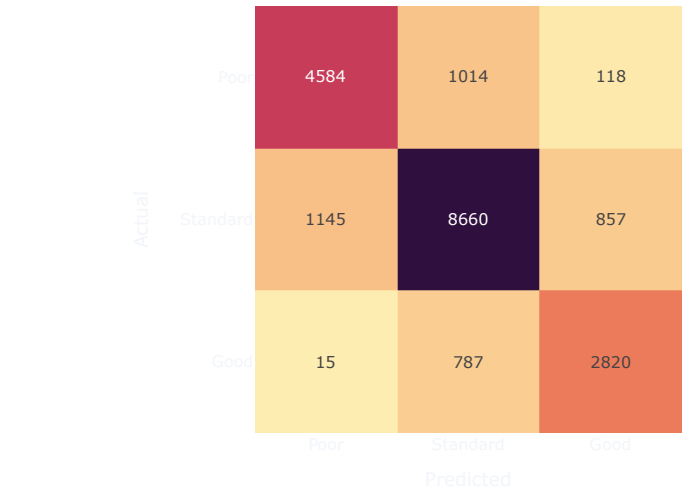
Models

```
def model_report(name, model, X, y):
  y_pred = model.predict(X)
  # Form confusion matrix dataframe
  cm_values = confusion_matrix(y, y_pred)
  df_cm = pd.DataFrame(cm_values,
                       columns=["Poor", "Standard", "Good"],
                       index=["Poor", "Standard", "Good"])
  df_cm.index.name = "Actual"
  df_cm.columns.name = "Predicted"
  # Plot confusion matrix
  fig = px.imshow(df_cm, text_auto=True, color_continuous_scale=px.colors.sequential.matter,
                  width=800, height=500, title=name)
  fig.show()
  # Print classification report
  print(classification_report(y, y_pred, target_names=['Poor','Standard','Good']))


model_report('Nearest Neighbors Classifier', knc, X_test_final, y_test)
```



- Overall, a 79% on test sample is decent. Recall score for Poor and Standard are quite high which is what we want. For Good, both recall and precision is quite low. There is a high confusion for Standard Scorers.
- Maybe we should reduce dimension to avoid the 'Curse of Dimensionality'. This means regions increase exponentially with dimension and each region might only contain one record which makes the target probability for that region 100%.

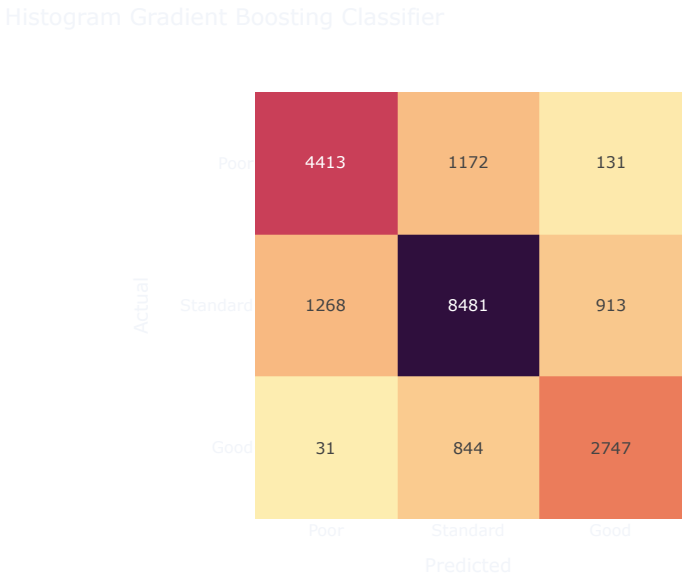
```
model_report('Random Forest Classifier', rfc, X_test_final, y_test)
```



	precision	recall	f1-score	support
Poor	0.80	0.80	0.80	5716
Standard	0.83	0.81	0.82	10662
Good	0.74	0.78	0.76	3622
accuracy			0.80	20000
macro avg	0.79	0.80	0.79	20000
weighted avg	0.80	0.80	0.80	20000

- Eventhough overall, accuracy score is good, the Good class f1_score is quite unimpressive at 76%. Maybe because of the imbalance classes, there are alot of confusion for the target Standard Score.

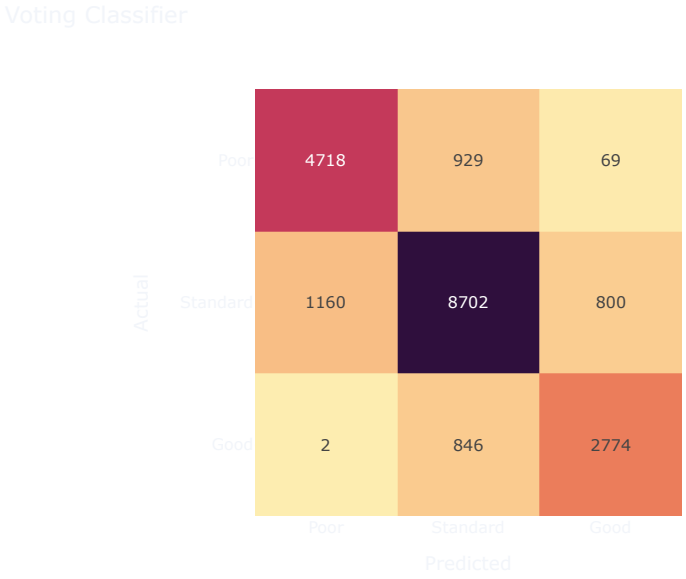
```
model_report('Histogram Gradient Boosting Classifier', hgbc, X_test_final, y_test)
```



	precision	recall	f1-score	support
Poor	0.77	0.77	0.77	5716
Standard	0.81	0.80	0.80	10662
Good	0.72	0.76	0.74	3622

- Overall f1-scores are a bit worse than Random Forest model.
- There is also a high confusion for Standard class.

```
model_report('Voting Classifier', vcf, X_test_final, y_test)
```



	precision	recall	f1-score	support
Poor	0.80	0.83	0.81	5716
Standard	0.83	0.82	0.82	10662
Good	0.76	0.77	0.76	3622
accuracy			0.81	20000
macro avg	0.80	0.80	0.80	20000
weighted avg	0.81	0.81	0.81	20000

- Slight better performance in all classes especially Poor recall score compare to Random Forest model.

- Clear improvement in the separation between Good and Poor Score which is what we want.

Features

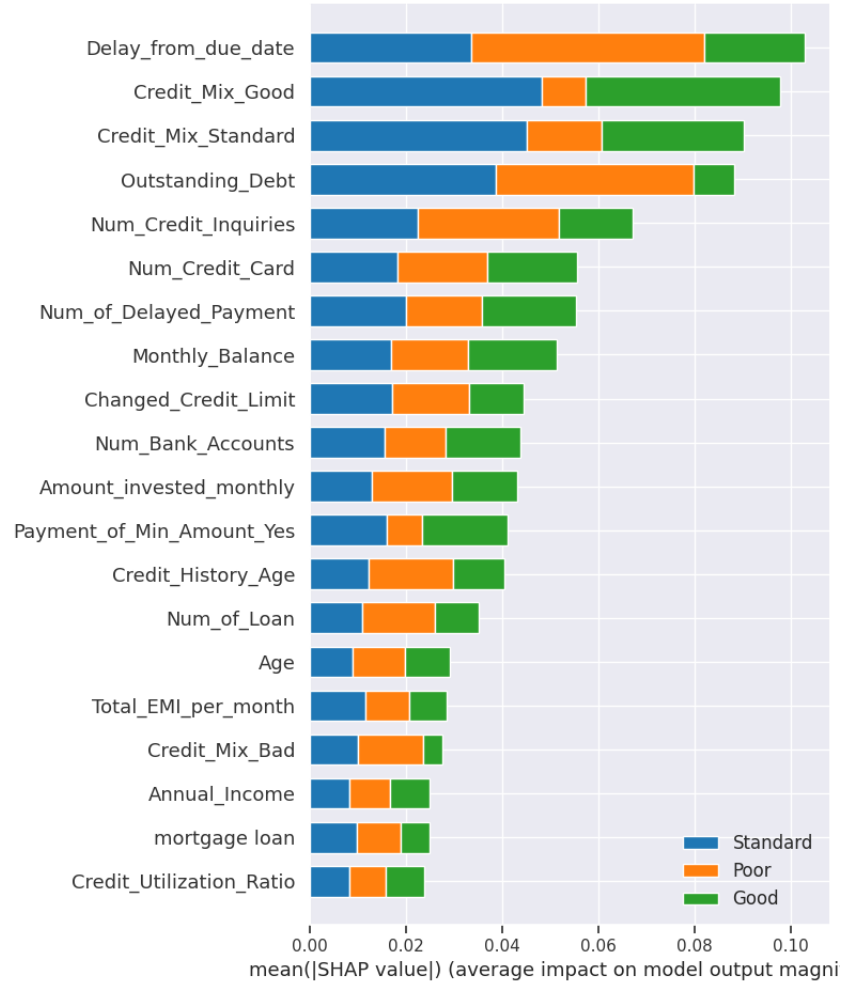
```
def plot_summary(shap_values, X):
  shap.summary_plot(shap_values, X.values, plot_type="bar", max_display=20,
                    class_names = ['Poor','Standard','Good'], feature_names = X.columns,
                    color=plt.get_cmap("tab10"), class_inds=[1,0,2])

def plot_bswarm(shap_values, X):
  plt.figure(figsize=(15,5))
  plt.subplot(131)
  plt.title('Poor Credit Score')
  shap.summary_plot(shap_values[0], X.values, feature_names=X.columns,
                    max_display=10, plot_type="violin", show=False, plot_size=None)
  plt.subplot(132)
  plt.title('Standard Credit Score')
  shap.summary_plot(shap_values[1], X.values, feature_names=X.columns,
                    max_display=10, plot_type="violin", show=False, plot_size=None)
  plt.subplot(133)
  plt.title('Good Credit Score')
  shap.summary_plot(shap_values[2], X.values, feature_names=X.columns,
                    max_display=10, plot_type="violin", show=False, plot_size=None)
  plt.tight_layout()
  plt.show();


# Reduce sample because Explainer has long runtime
X_50samples = X_train_final.sample(n=50, random_state=31)
X_100samples = X_train_final.sample(n=100, random_state=31)
# Build SHAP value explainer for NearestNeighbors
explainer = shap.KernelExplainer(knc.predict_proba, X_50samples)
knc_shap_values = explainer.shap_values(X_50samples)
# Build SHAP value explainer for RandomForest
explainer = shap.TreeExplainer(rfc)
rfc_shap_values = explainer.shap_values(X_100samples)
# Build SHAP value explainer for HistGradientBoost
explainer = shap.TreeExplainer(hgbc)
hgbc_shap_values = explainer.shap_values(X_100samples)
# Build SHAP value explainer for VotingClassifier
explainer = shap.KernelExplainer(vcf.predict_proba, X_50samples)
vcf_shap_values = explainer.shap_values(X_50samples)
```

```
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
  0%|          | 0/50 [00:00<?, ?it/s]
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
```

NearestNeighborsClassifier

```
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
```

```
plot_summary(knc_shap_values, X_50samples)
```



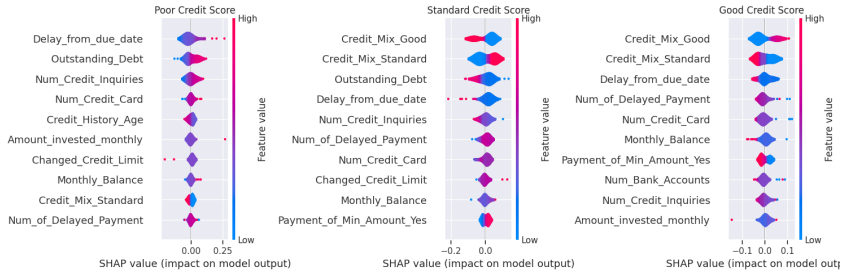

```
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
```

- Here we can see overall, Credit Score is explained mostly by Late payment, Debt, Credit Mix and Credit Inquiries. The class Standard is affected relatively equally with other classes in many features. This explains the high confusion for Standard Score class.

```
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
```

```
plot_bswarm(knc_shap_values, X_50samples)
```

```
No data for colormapping provided via 'c'. Parameters 'vmin', 'vmax' will be ignor
```




```
X does not have valid feature names, but KNeighborsClassifier was fitted with feat
```

The above graph shows the top explainer for each class. The feature relationship strength and direction.

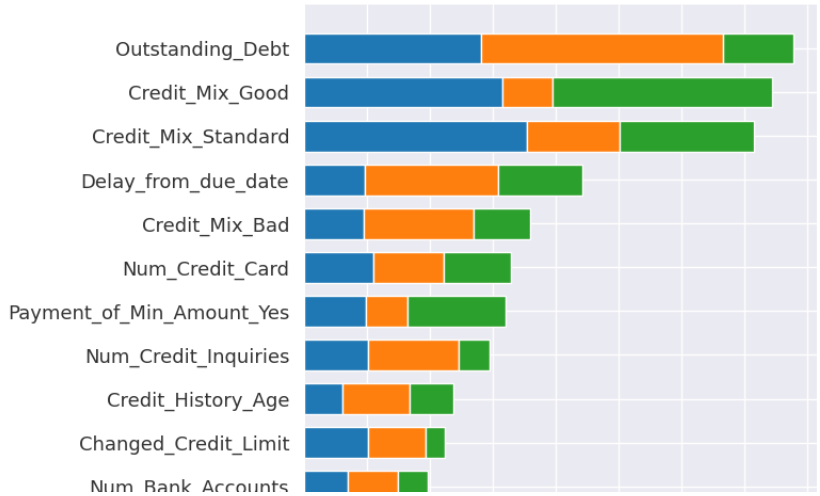
- Poor class: Top explainer are Late payment, Debt and Credit Inquiries.
- Standard class: Top explainer are Debt, Good and Standard Credit Mix.
- Good class: Top explainer are Number of Credit cards, Good and Standard Credit Mix.

```
X does not have valid feature names, but HistGradientBoostingClassifier was fitted
```

RandomForestClassifier

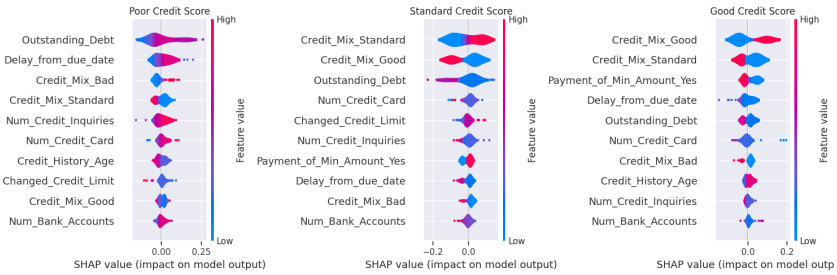
```
X does not have valid feature names, but HistGradientBoostingClassifier was fitted
```

```
plot_summary(rfc_shap_values, X_100samples)
```



- Here we can see overall, Credit Score is explained mostly by Late payment, Debt, Credit Mix and Delay from due date. The class Standard is affected relatively equally with other classes in many features. This explains the high confusion for Standard Score class.

```
plot_bswarm(rfc_shap_values, X_100samples)
```

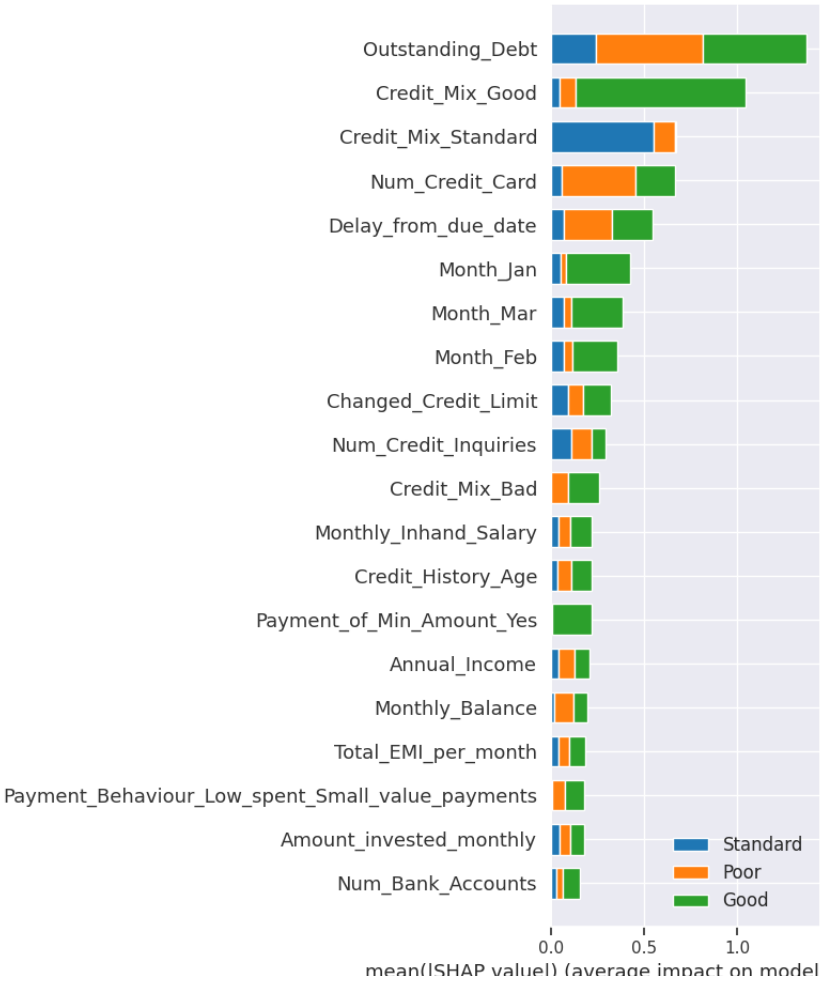


The above graph shows the top explainer for each class. The feature relationship strength and direction.

- Poor class: Top explainer are Debt, Credit mix and Late payment.
- Standard class: Top explainer are Debt, Good and Standard Credit Mix.
- Good class: Top explainer are Delay from due date, Good and Standard Credit Mix.

HistGradientBoosting

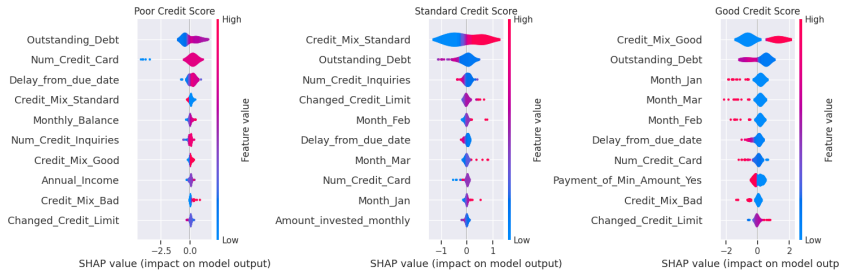
```
plot_summary(hgbc_shap_values, X_100samples)
```



- Overall, Credit Score is explained mostly by Debt, Credit Mix and Number of Credit Cards. Each features exaplain each classes relatively differently.

```
X does not have valid feature names, but HistGradientBoostingClassifier was fitted
```

```
plot_bswarm(hgbc_shap_values, X_100samples)
```




```
X does not have valid feature names, but RandomForestClassifier was fitted with fe
```

The above graph shows the top explainer for each class. The feature relationship strength and direction.

- Poor class: Top explainer are Debt, Late payment and Number of credit cards.

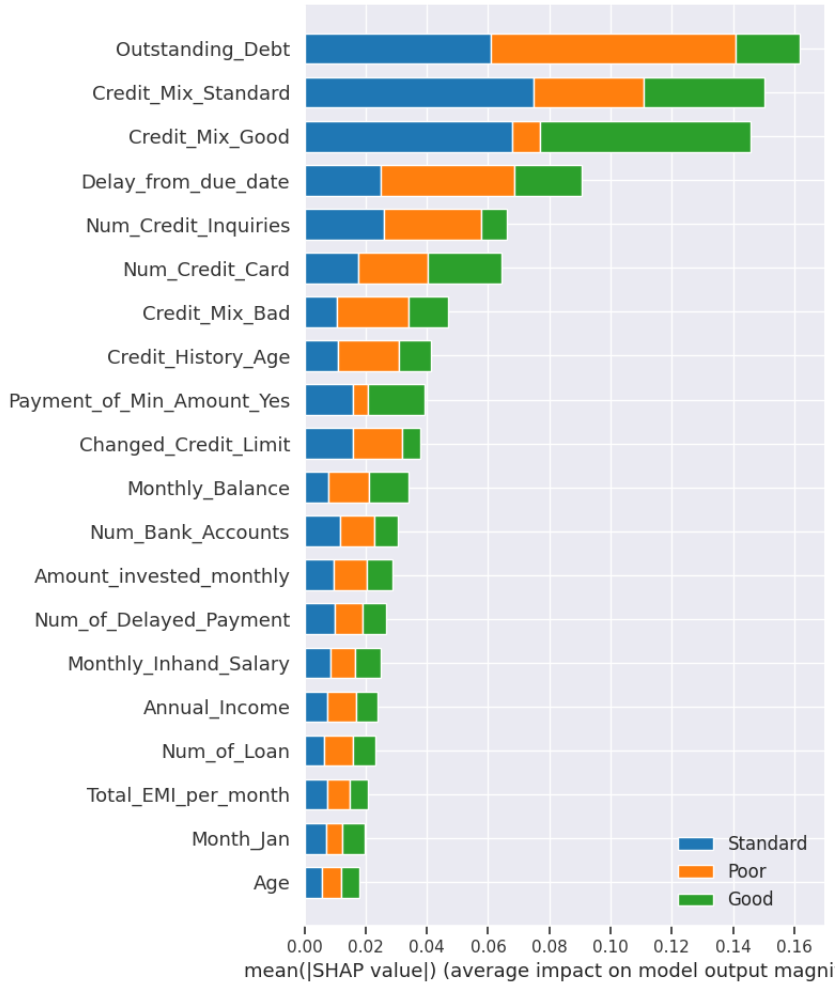
- Standard class: Top explainer are Debt, Standard Credit Mix and Credit inquiries.
- Good class: Top explainer are Debt, Good Credit Mix and Month of January.

```
X does not have valid feature names, but RandomForestClassifier was fitted with fe
```

VotingClassifier

```
X does not have valid feature names, but RandomForestClassifier was fitted with fe
X does not have valid feature names, but HistGradientBoostingClassifier was fitted
```

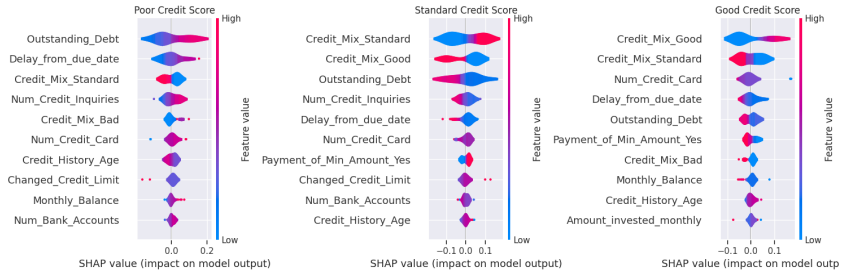
```
plot_summary(vcf_shap_values, X_50samples)
```




```
X does not have valid feature names, but HistGradientBoostingClassifier was fitted
```

- Overall, Credit Score is explained mostly by Debt, Credit Mix and Late payment. We can see that for Good Credit Mix feature, it explained Standard class and Good class equally. And for most features, there are large difference in SHAP values for Good and Poor class.

```
plot_bswarm(vcf_shap_values, X_50samples)
```



The above graph shows the top explainer for each class. The feature relationship strength and direction.

- Poor class: Top explainer are Debt, Late payment and Standard credit mix.
- Standard class: Top explainer are Debt, Good and Standard Credit Mix.
- Good class: Top explainer are Number of credit cards, Good and Standard Credit Mix.

Conclusion

- Standard class has many overlap features with other classes.
- Age explains for some of Credit Score indirectly.
- SHAP assume features are independent, and it explains the feature importance to the model not to the real world.

Future fixes

- Identify or create signature feature for Standard Credit Scorers. Then put more weight on that feature for Standard Class in future models.
- Maybe we can calculate actual Credit Score base on features which will provide a wider range of continuous values rather than 3 discreet Score values.