# MINORS Lab 2B: Logistic Regression

Varun Kamath

2019110023

13/11/2022

1. Upload csv files and import libraries

```
[1]  import warnings
     import pandas as pd
     from pandas.api.types import is_numeric_dtype
     import numpy as np

     import matplotlib.pyplot as plt
     import seaborn as sns

     from sklearn.preprocessing import LabelEncoder as le
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import RobustScaler as rbScaler
     from sklearn.linear_model import LogisticRegression as lgrClassifier
     from sklearn import metrics

     from statsmodels.stats.outliers_influence import variance_inflation_factor

     warnings.filterwarnings('ignore')
     %matplotlib inline
```

```
[3]  from google.colab import drive
     drive.mount('/content/drive')
     df = pd.read_csv('/content/drive/MyDrive/data/train.csv', low_memory=False)
     df.shape
```

```
Mounted at /content/drive
(100000, 28)
```

2. Analyse the data by finding out number of null values in each attribute and their data types

```
null_count = df.isnull().sum().sort_values(ascending=False)
null_count
```

```
df.info()
```

3. Classify the data set into continuous (integral) attributes and categorical columns. Remove the irrelevant columns

```
num_cols = ['Age', 'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Outstanding_Debt', 'Credit_Utilization_Ratio',
       'Total_EMI_per_month', 'Amount_invested_monthly',
       'Monthly_Balance', 'Credit_History_Age']

categorical_cols = ['Occupation', 'Credit_Mix', 'Payment_of_Min_Amount',
       'Payment_Behaviour', 'Credit_Score']
```

+ Code

```
[7] irrelavent_coulumns = ['ID', 'Customer_ID', 'Month', 'Name', 'SSN']
    df.drop(columns=irrelavent_coulumns, inplace=True, axis=1)
```

4. Clean the data by replacing invalid entries with NaN values. Change the type of the numerical values.

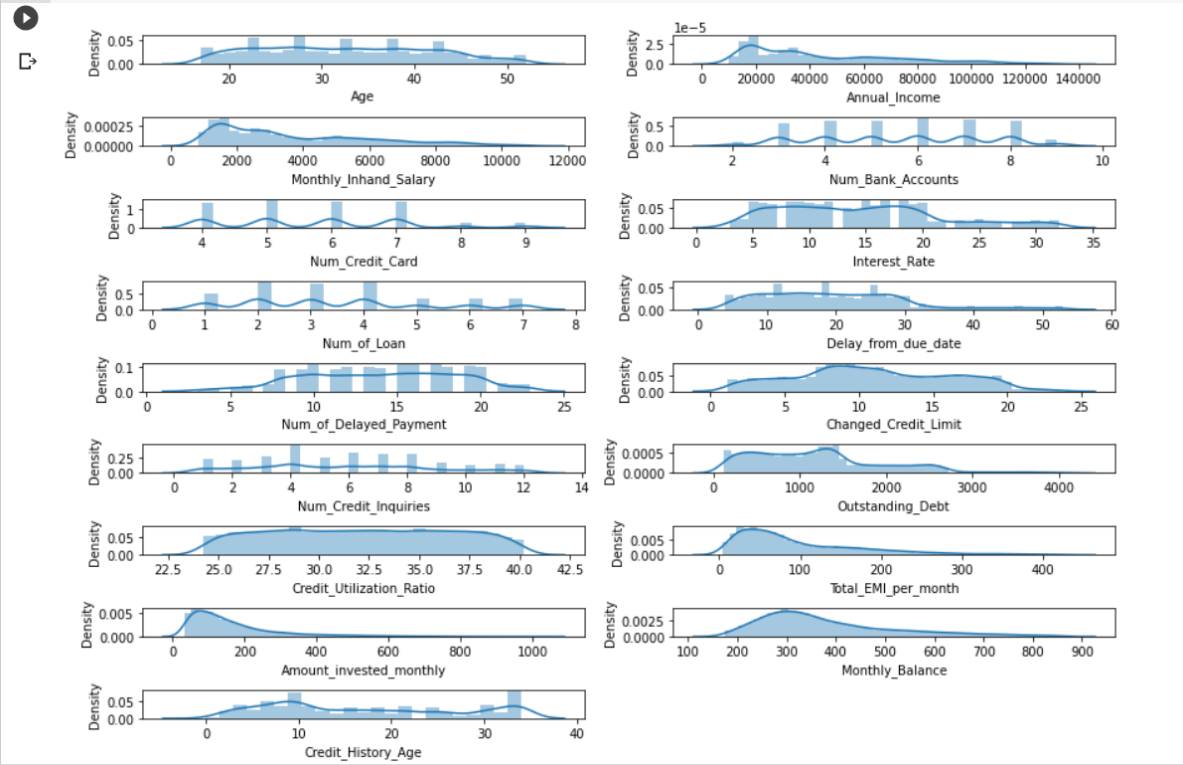
```
[8] df = df.applymap(
        lambda x: x if x is np.NaN or not \
            isinstance(x, str) else str(x).strip('_')).replace(
                ['', 'nan', '!@9#%8', '#F%$D@*&8'], np.NaN
            )
```

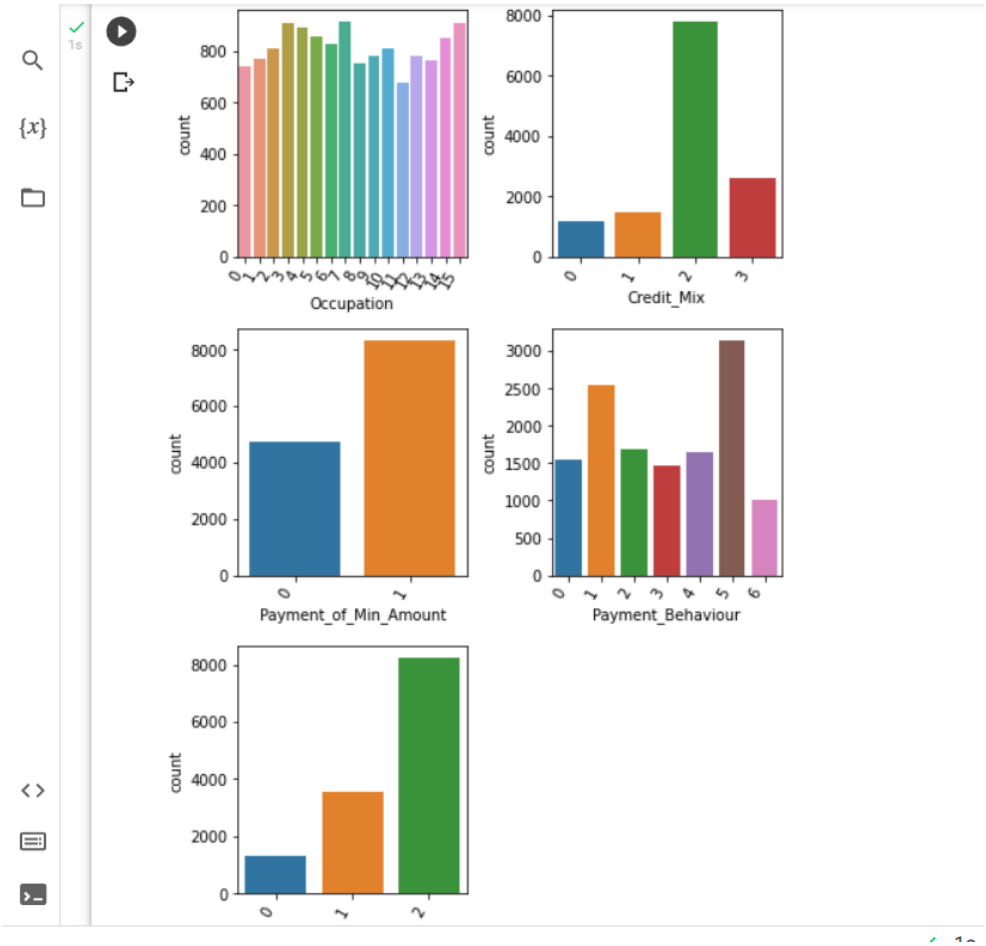
```
[10] def take_years(x):
         if x is not None:
             return str(x).strip()[0:2]

     df.Credit_History_Age=df.Credit_History_Age.apply(take_years)
     df['Credit_History_Age'] = df['Credit_History_Age'].replace({'na': np.NaN})
```

```
df.Age = df.Age.astype(int)
df.Annual_Income = df.Annual_Income.astype(float)
df.Num_of_Loan = df.Num_of_Loan.astype(int)
df.Num_of_Delayed_Payment = df.Num_of_Delayed_Payment.astype(float)
df.Changed_Credit_Limit = df.Changed_Credit_Limit.astype(float)
df.Outstanding_Debt = df.Outstanding_Debt.astype(float)
df.Amount_invested_monthly = df.Amount_invested_monthly.astype(float)
df.Monthly_Balance = df.Monthly_Balance.astype(float)
```

5. Observe the distribution of the numerical data, to realize the trend. Also observe the distribution of categorical data




6. Remove the outliers and generate a heatmap to observe correlation

```
[15] def remove_outlier(df):
         low = .05
         high = .95
         quant_df = df.quantile([low, high])
         print(quant_df)
         for name in list(df.columns):
             if is_numeric_dtype(df[name]):
                 df = df[(df[name] > quant_df.loc[low, name]) & (df[name] < quant_df.loc[high, name])]
         return df

     df = remove_outlier(df)
```

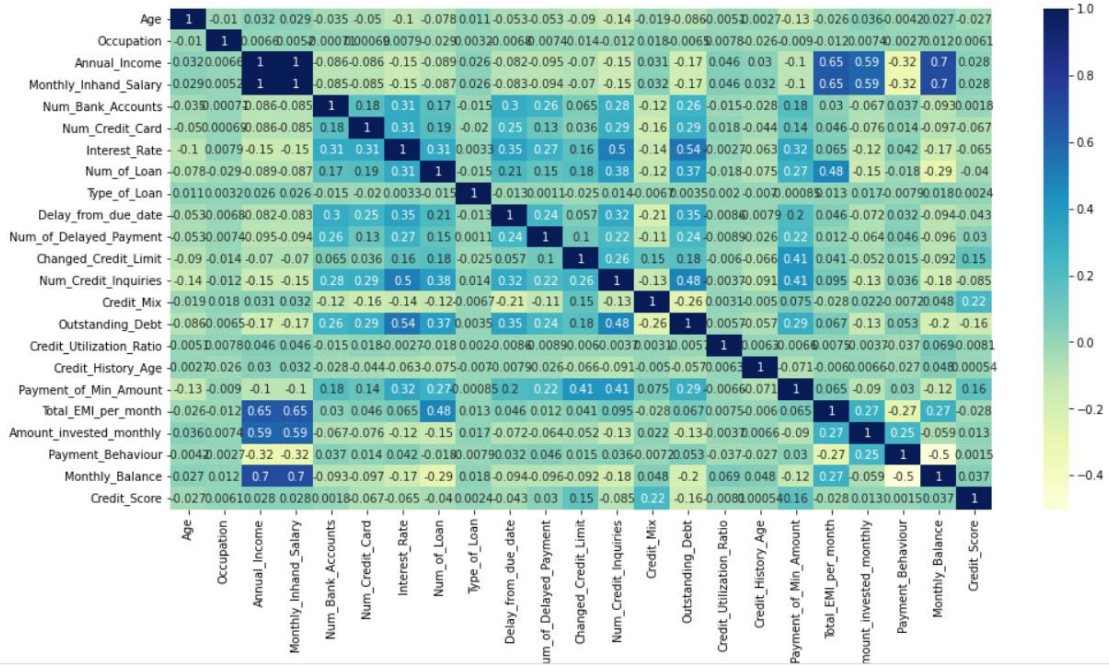
```
       Age  Annual_Income  Monthly_Inhand_Salary  Num_Bank_Accounts  \
0.05  16.0        9743.51             836.125833                1.0
0.95  53.0      134533.32           10828.226500               10.0

      Num_Credit_Card  Interest_Rate  Num_of_Loan  Delay_from_due_date  \
0.05              3.0            2.0          0.0                  3.0
0.95             10.0           33.0          8.0                 54.0

      Num_of_Delayed_Payment  Changed_Credit_Limit  Num_Credit_Inquiries  \
0.05                     2.0                  1.16                   0.0
0.95                    24.0                 23.60                  13.0

      Outstanding_Debt  Credit_Utilization_Ratio  Total_EMI_per_month  \
0.05          118.5465                 24.230834             0.000000
0.95         4073.7605                 40.220207           437.012753

      Amount_invested_monthly  Monthly_Balance
0.05                31.893067       174.599433
0.95              1149.405785       862.590861
```



7. Split the data, train the model and find out the test accuracy. It comes out be 68.6%

```
[27] x = mdf.drop(['Credit_Score'] , axis = 1).values
     y = mdf['Credit_Score' ].values
```

```
# Data Split
x_train , x_test , y_train , y_test = train_test_split(x,y , test_size= 0.2 , random_state=50)
print([x_train.shape, y_train.shape, x_test.shape, y_test.shape])

# Data Scaling using Robust Scaler
ro_scaler = rbScaler()
x_train = ro_scaler.fit_transform(x_train)
x_test = ro_scaler.fit_transform(x_test)
[x_train.shape, x_test.shape]

# logistic Regression
lgr = lgrClassifier(C = 100)
lgr.fit(x_train , y_train)

lgr_score = lgr.score(x_train , y_train)
lgr_score_t = lgr.score(x_test , y_test)

y_pred1 = lgr.predict(x_test)
dd = pd.DataFrame({"Y_test" : y_test , "y_pred1": y_pred1})
plt.figure(figsize=(10,8))
plt.plot(dd[:100])
plt.legend(["Actual" , "Predicted"])

print(f"Train Score: {lgr_score}")
print(f"Test Score: {lgr_score_t}")
```

```
[(10455, 11), (10455,), (2614, 11), (2614,)]
Train Score: 0.6983261597321856
Test Score: 0.6866870696250956
```

Conclusion:

- Successfully trained a model to predict the credit score using Regression
- Test accuracy can be improved using other techniques like regularization