Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

BE-ETRX **UID:**2019110039 **Sub-Minor ML**

NAME: Devansh Palliyath

Exp 2B

Aim: To build and predict the model using logistic regression on the given Credit score dataset.

Code:

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

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

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

```
[ ] 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']
```

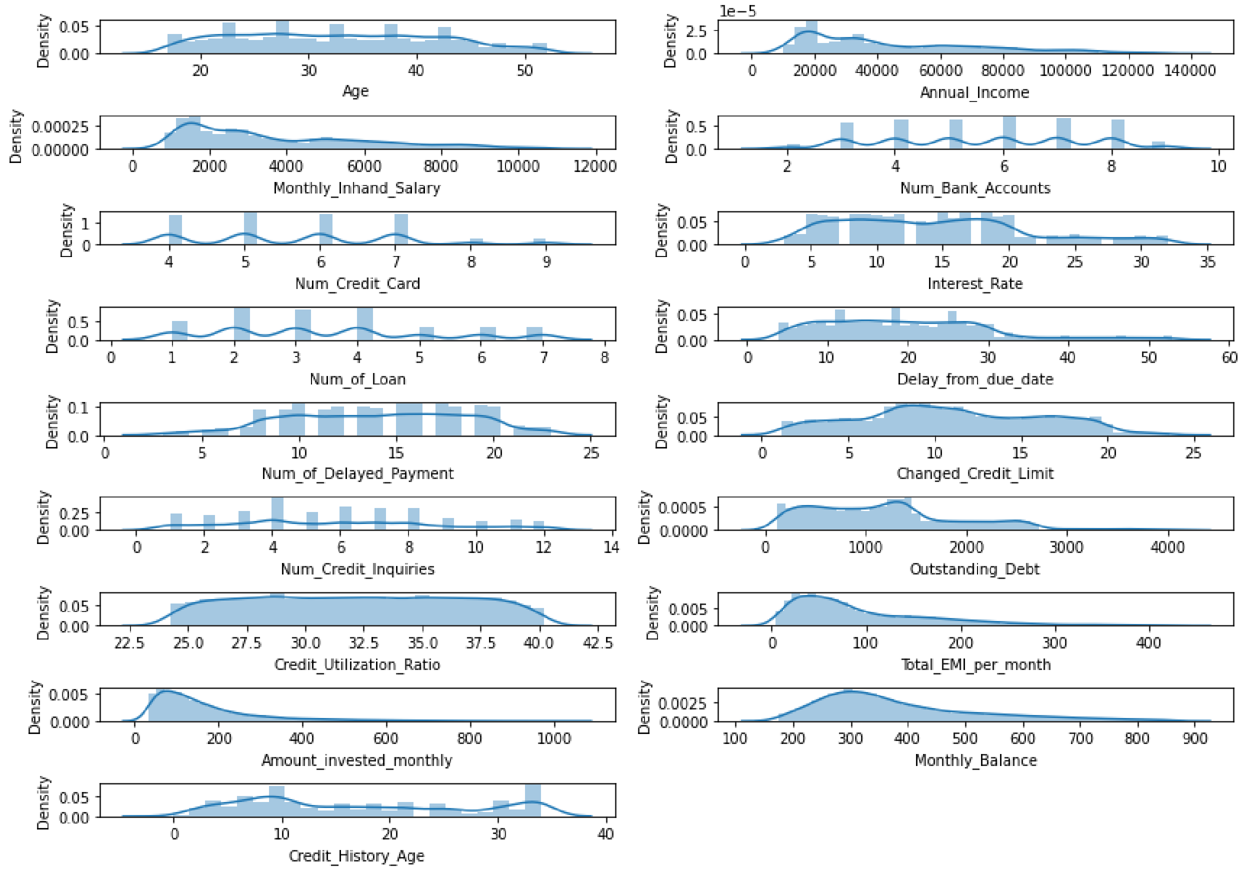
```
irrelavent_coulumns = ['ID', 'Customer_ID', 'Month', 'Name', 'SSN']
df.drop(columns=irrelavent_coulumns, inplace=True, axis=1)
```

```
[ ] 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
            )
```

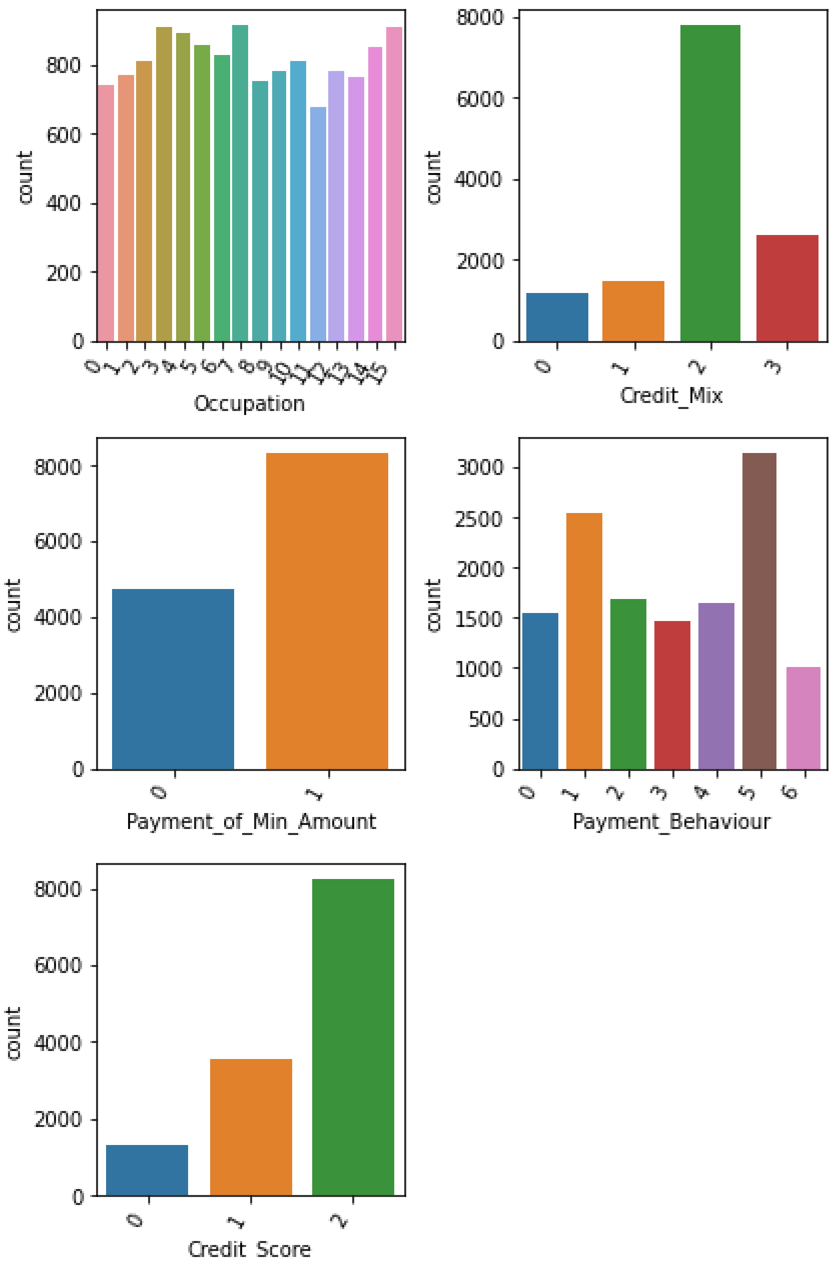
```
def take_years(x):
    if x is not None:
        return str(x).strip()[0:2]

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)
```

```
rows=10
cols=2
counter=1
plt.rcParams['figure.figsize']=[12, 9]
for i in num_cols:
    plt.subplot(rows, cols, counter)
    sns.distplot(df[i])
    counter+=1
plt.tight_layout()
plt.show()
```




```
rows=3
cols=2
counter=1
plt.rcParams['figure.figsize']=[6,9]
for i in categorical_cols:
    plt.subplot(rows,cols,counter)
    sns.countplot(x=i,data=df)
    plt.xticks(rotation=60,ha='right')
    counter+=1
plt.tight_layout()
plt.show()
```


count
Occupation
count
Credit_Mix
count
Payment_of_Min_Amount
count
Payment_Behaviour
count
Credit_Score

```
df['Payment_of_Min_Amount'] = df['Payment_of_Min_Amount'].replace({'NM': 'No'})
```

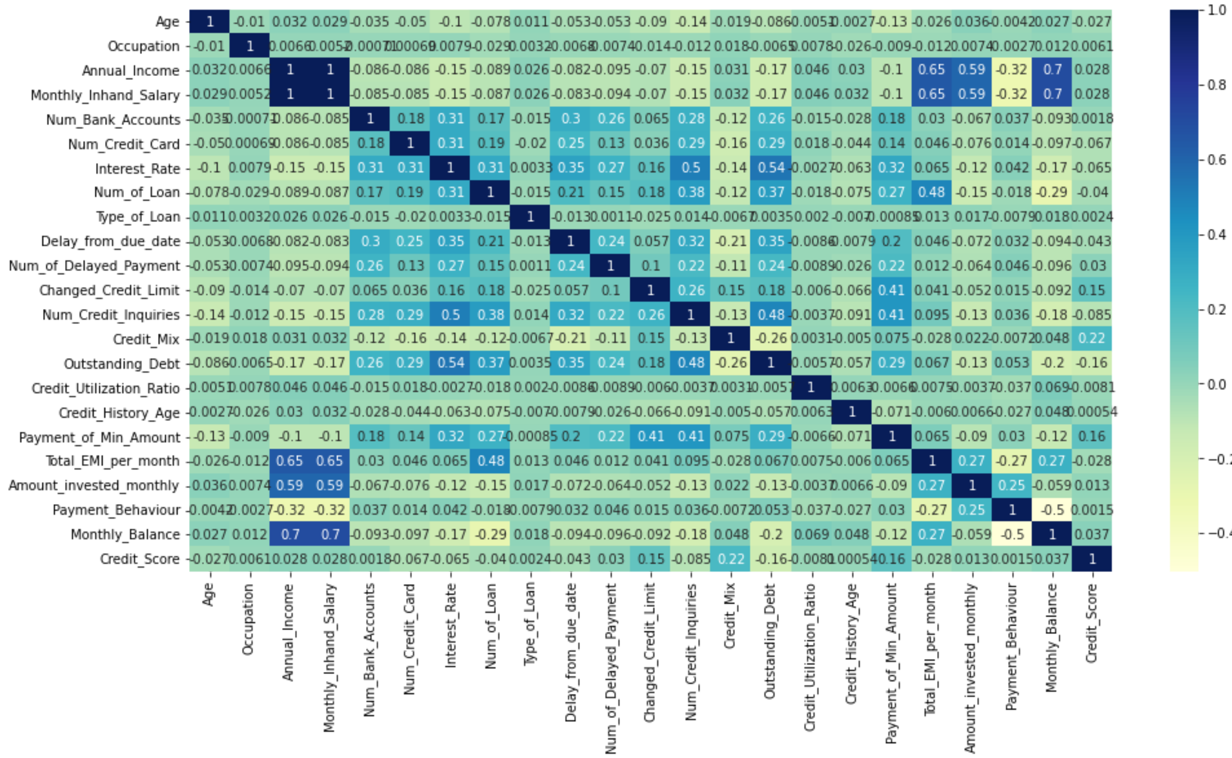
```
plt.rcParams['figure.figsize'] = [3,3]
sns.countplot(x='Payment_of_Min_Amount', data=df)
plt.xticks(rotation=60, ha='right')
plt.tight_layout()
plt.show()
```


```
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)
```

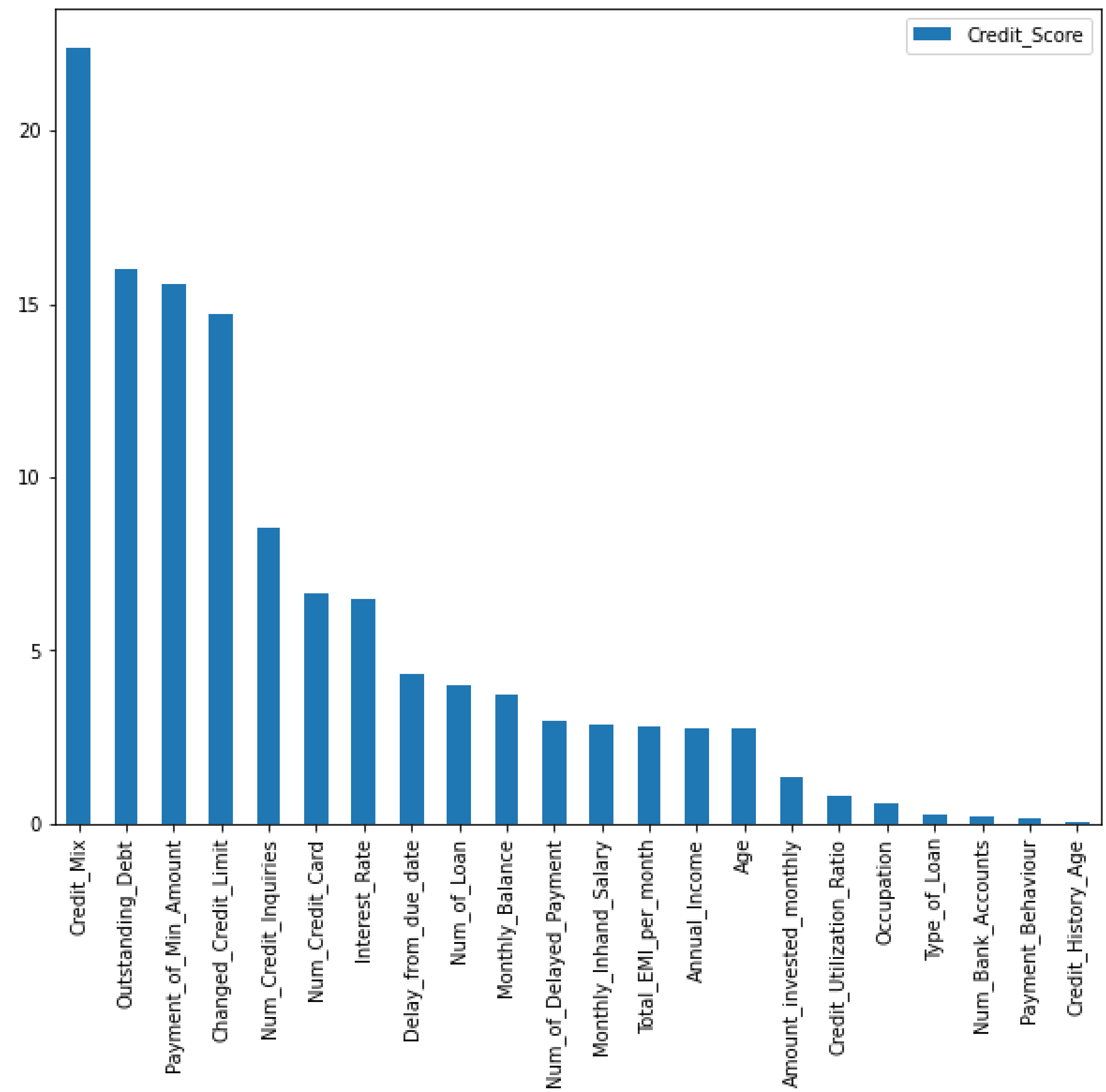
```
      Age  Annual_Income  Monthly_Inhand_Salary  Num_Bank_Accounts  \
0.05  16.0        9743.51             836.125833                1.0
0.95  53.0      134533.32           10828.226500               10.0

      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
```

```
df.interpolate(method='linear', inplace=True)
```

```
Occupation_le = le()
Type_of_Loan_le = le()
Credit_Mix_le = le()
Credit_History_Age_le = le()
Payment_of_Min_Amount_le = le()
Payment_Behaviour_le = le()
Credit_Score_le = le()

df['Occupation'] = Occupation_le.fit_transform(df['Occupation'])
df['Type_of_Loan'] = Type_of_Loan_le.fit_transform(df['Type_of_Loan'])
df['Credit_Mix'] = Credit_Mix_le.fit_transform(df['Credit_Mix'])
df['Credit_History_Age'] =
Credit_History_Age_le.fit_transform(df['Credit_History_Age'])
df['Payment_of_Min_Amount'] =
Payment_of_Min_Amount_le.fit_transform(df['Payment_of_Min_Amount'])
df['Payment_Behaviour'] =
Payment_Behaviour_le.fit_transform(df['Payment_Behaviour'])
df['Credit_Score'] = Credit_Score_le.fit_transform(df['Credit_Score'])
plt.figure(figsize = (16,8))
sns.heatmap(df.corr() , annot = True, cmap = "YlGnBu")
```




```
pd.DataFrame(abs(df.corr()['Credit_Score'].drop('Credit_Score')*100).sort_
values(
    ascending=False)).plot.bar(figsize = (10,8))
```



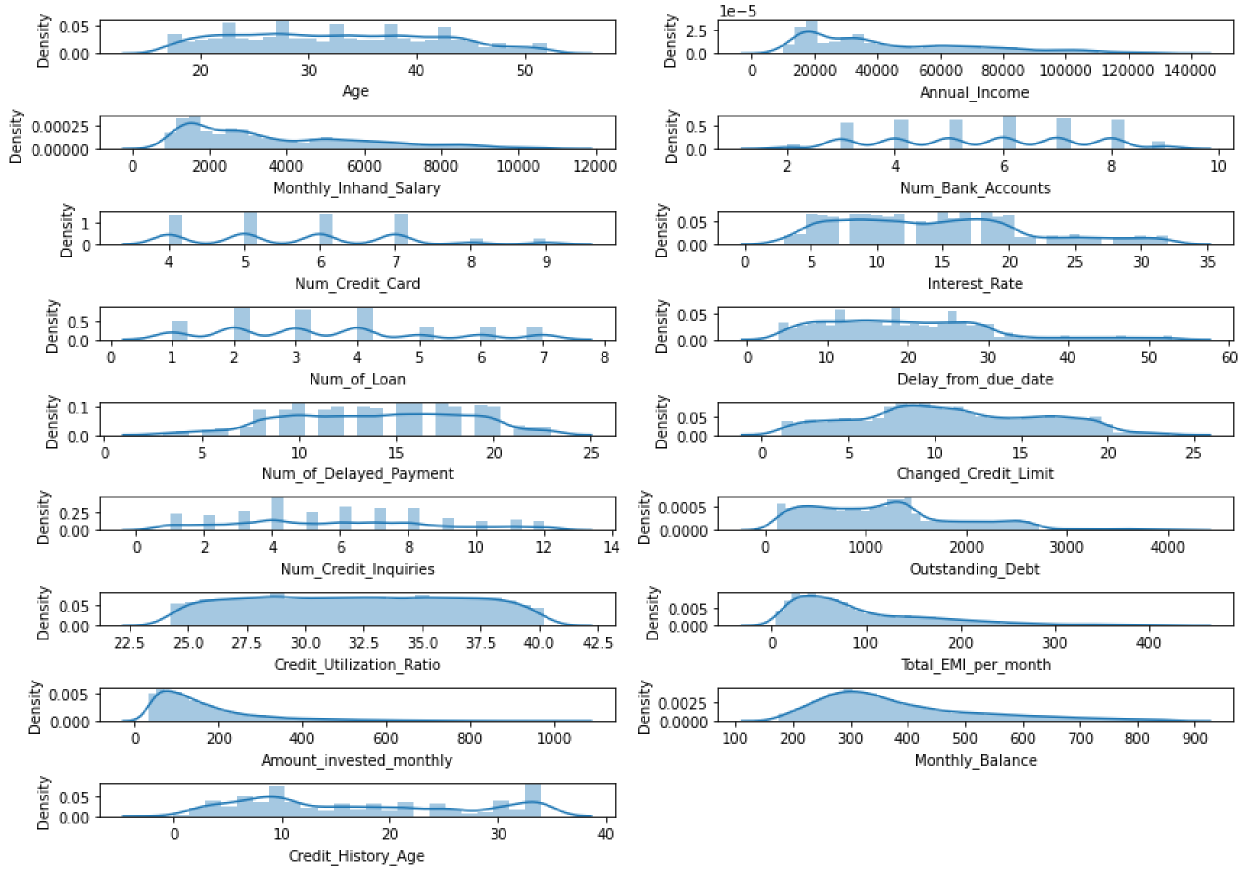
Credit_Score
0
5
10
15
20
Credit_Mix
Outstanding_Debt
Payment_of_Min_Amount
Changed_Credit_Limit
Num_Credit_Inquiries
Num_Credit_Card
Interest_Rate
Delay_from_due_date
Num_of_Loan
Monthly_Balance
Num_of_Delayed_Payment
Monthly_Inhand_Salary
Total_EMI_per_month
Annual_Income
Age
Amount_invested_monthly
Credit_Utilization_Ratio
Occupation
Type_of_Loan
Num_Bank_Accounts
Payment_Behaviour
Credit_History_Age

```
round(abs(df.corr()['Credit_Score']*100).sort_values(ascending=False), 2)
```

```
Credit_Score                  100.00
Credit_Mix                     22.41
Outstanding_Debt               16.01
Payment_of_Min_Amount          15.55
Changed_Credit_Limit           14.70
Num_Credit_Inquiries            8.52
Num_Credit_Card                 6.66
Interest_Rate                   6.48
Delay_from_due_date             4.30
Num_of_Loan                     3.98
Monthly_Balance                 3.74
Num_of_Delayed_Payment          2.97
Monthly_Inhand_Salary           2.84
Total_EMI_per_month             2.80
Annual_Income                   2.77
Age                             2.75
Amount_invested_monthly         1.34
Credit_Utilization_Ratio        0.81
Occupation                      0.61
Type_of_Loan                    0.24
Num_Bank_Accounts               0.18
Payment_Behaviour               0.15
Credit_History_Age              0.05
Name: Credit_Score, dtype: float64
```

```
numeric_cols = df.select_dtypes(exclude = "object").columns

vif_df = df[numeric_cols]
vif_data = pd.DataFrame()
vif_data["feature"] = vif_df.columns
vif_data["VIF"] = [variance_inflation_factor(vif_df.values ,i) for i in
range(len(vif_df.columns))]
vif_data.head(17)
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_missing})

missing_value_df.sort_values('percent_missing', ascending=False,
inplace=True)
missing_value_df
numeric_cols = df.select_dtypes(exclude = "object").columns

vif_df = df[numeric_cols]
```

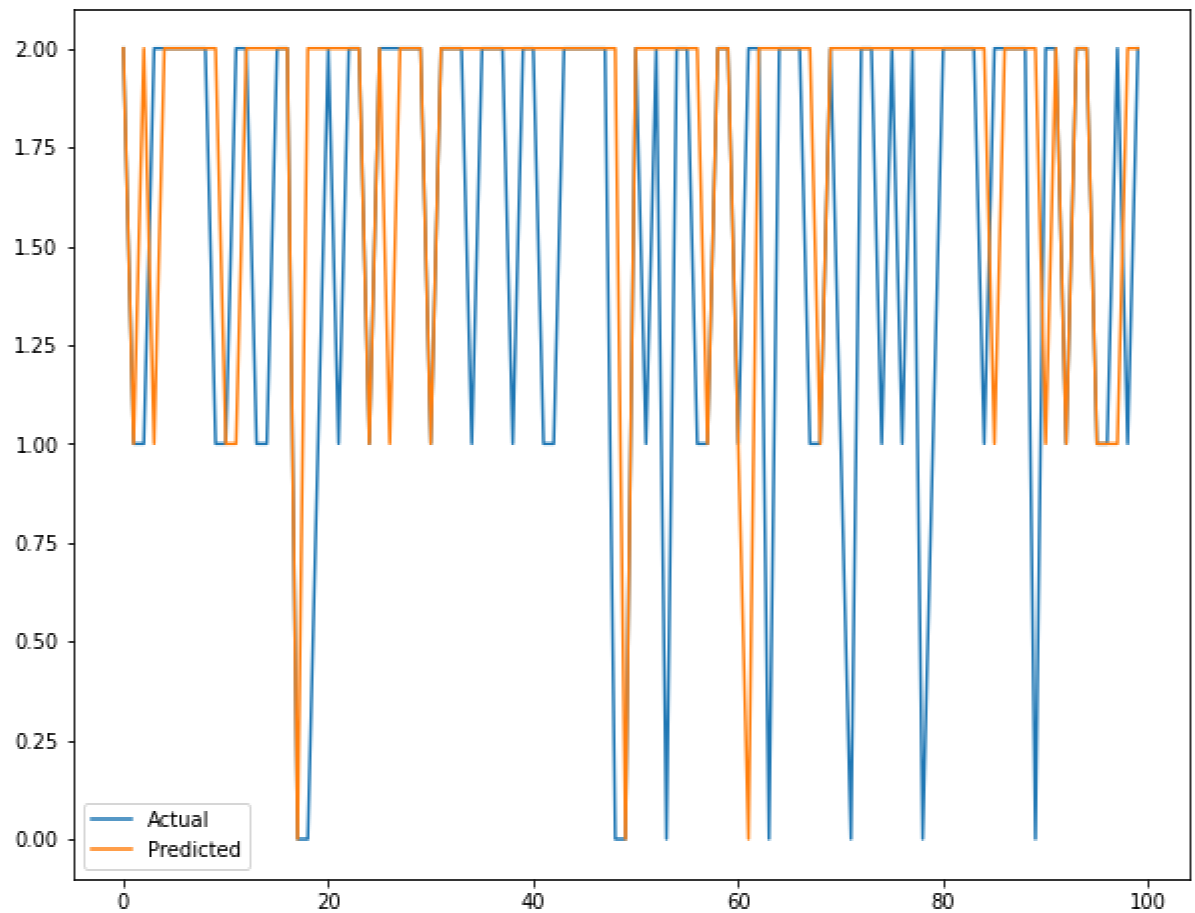
```
vif_data = pd.DataFrame()
vif_data["feature"] = vif_df.columns
vif_data["VIF"] = [variance_inflation_factor(vif_df.values ,i) for i in
range(len(vif_df.columns))]
vif_data.head(17)
rows=10
cols=2
counter=1
plt.rcParams['figure.figsize']=[12, 9]
for i in num_cols:
    plt.subplot(rows, cols, counter)
    sns.distplot(df[i])
    counter+=1
plt.tight_layout()
plt.show()
```




```python
mdf = df[
    ['Credit_Score','Changed_Credit_Limit',
      'Payment_of_Min_Amount', 'Credit_Mix',
      'Delay_from_due_date', 'Annual_Income',
      'Age', 'Monthly_Balance', 'Outstanding_Debt',
      'Payment_Behaviour', 'Credit_History_Age',
      'Num_Bank_Accounts'
    ]
]
```

```python
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}")
```




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

```
Train Score: 0.6983261597321856
Test Score: 0.6866870696250956
```

Conclusion:

- Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring. An example of logistic regression could be applying machine learning to determine if a person is likely to be infected with COVID-19 or not. Since we have two possible outcomes to this question - yes they are infected, or no they are not infected - this is called binary classification.
- In linear regression, the outcome is continuous and can be any possible value. However in the case of logistic regression, the predicted outcome is discrete and restricted to a limited number of values.
- Logistic regression is a classification algorithm used to find the probability of event success and event failure. It is used when the dependent variable is binary(0/1, True/False, Yes/No) in nature. It supports categorizing data into discrete classes by studying the relationship from a given set of labeled data. It learns a linear relationship from the given dataset and then introduces a non-linearity in the form of the Sigmoid function.
- We successfully carried out the experiment.