Importing Libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn import metrics
from xgboost import XGBClassifier
import plotly.express as px
import xgboost as xgb
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score

import warnings

warnings.filterwarnings("ignore", category=ImportWarning, module='specific_module')
# Code that might trigger the warning for specific_module
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Importing dataset

```
# Load your dataset
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/Credit Card Score/Credit-Score-Data.csv')
df
```

	ID	Customer_ID	Month	Name	Age	SSN	Occupation	Annual_Income
0	5634	3392	1	Aaron Maashoh	23	821000265	Scientist	19114.12
1	5635	3392	2	Aaron Maashoh	23	821000265	Scientist	19114.12
2	5636	3392	3	Aaron Maashoh	23	821000265	Scientist	19114.12
3	5637	3392	4	Aaron Maashoh	23	821000265	Scientist	19114.12
4	5638	3392	5	Aaron Maashoh	23	821000265	Scientist	19114.12
...	...	...	...	...	...	...	...	...
99995	155625	37932	4	Nicks	25	78735990	Mechanic	39628.99
99996	155626	37932	5	Nicks	25	78735990	Mechanic	39628.99
99997	155627	37932	6	Nicks	25	78735990	Mechanic	39628.99
99998	155628	37932	7	Nicks	25	78735990	Mechanic	39628.99
99999	155629	37932	8	Nicks	25	78735990	Mechanic	39628.99

100000 rows × 28 columns

```
df.shape
```

```
(100000, 28)
```

Data pre-processing

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 28 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   ID                        100000 non-null  int64  
 1   Customer_ID               100000 non-null  int64  
 2   Month                     100000 non-null  int64  
 3   Name                      100000 non-null  object 
 4   Age                       100000 non-null  int64  
 5   SSN                       100000 non-null  int64  
 6   Occupation                100000 non-null  object 
 7   Annual_Income             100000 non-null  float64
 8   Monthly_Inhand_Salary     100000 non-null  float64
 9   Num_Bank_Accounts         100000 non-null  int64  
 10  Num_Credit_Card           100000 non-null  int64  
 11  Interest_Rate             100000 non-null  int64  
 12  Num_of_Loan               100000 non-null  int64  
 13  Type_of_Loan              100000 non-null  object 
 14  Delay_from_due_date       100000 non-null  int64  
 15  Num_of_Delayed_Payment    100000 non-null  int64  
 16  Changed_Credit_Limit      100000 non-null  float64
 17  Num_Credit_Inquiries      100000 non-null  int64  
 18  Credit_Mix                100000 non-null  object 
 19  Outstanding_Debt          100000 non-null  float64
 20  Credit_Utilization_Ratio  100000 non-null  float64
 21  Credit_History_Age        100000 non-null  int64  
 22  Payment_of_Min_Amount     100000 non-null  object 
 23  Total_EMI_per_month       100000 non-null  float64
 24  Amount_invested_monthly   100000 non-null  float64
 25  Payment_Behaviour         100000 non-null  object 
 26  Monthly_Balance           100000 non-null  float64
 27  Credit_Score              100000 non-null  object 
dtypes: float64(8), int64(13), object(7)
memory usage: 21.4+ MB
```

```
df.columns
```

```
Index(['ID', 'Customer_ID', 'Month', 'Name', 'Age', 'SSN', 'Occupation',
       'Annual_Income', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
       'Num_Credit_Card', 'Interest_Rate', 'Num_of_Loan', 'Type_of_Loan',
       'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
       'Num_Credit_Inquiries', 'Credit_Mix', 'Outstanding_Debt',
       'Credit_Utilization_Ratio', 'Credit_History_Age',
       'Payment_of_Min_Amount', 'Total_EMI_per_month',
       'Amount_invested_monthly', 'Payment_Behaviour', 'Monthly_Balance',
       'Credit_Score'],
      dtype='object')
```

```
#Checking for null values
df.isna().sum()
```

```
ID                          0
Customer_ID                 0
Month                       0
Name                        0
Age                         0
SSN                         0
Occupation                  0
Annual_Income               0
Monthly_Inhand_Salary       0
Num_Bank_Accounts           0
Num_Credit_Card             0
Interest_Rate               0
Num_of_Loan                 0
Type_of_Loan                0
Delay_from_due_date         0
Num_of_Delayed_Payment      0
Changed_Credit_Limit        0
Num_Credit_Inquiries        0
Credit_Mix                  0
Outstanding_Debt            0
Credit_Utilization_Ratio    0
Credit_History_Age          0
Payment_of_Min_Amount       0
```

```
Total_EMI_per_month         0
Amount_invested_monthly     0
Payment_Behaviour           0
Monthly_Balance             0
Credit_Score                0
dtype: int64
```

```
#Checking for duplicated
df.duplicated().sum()
```

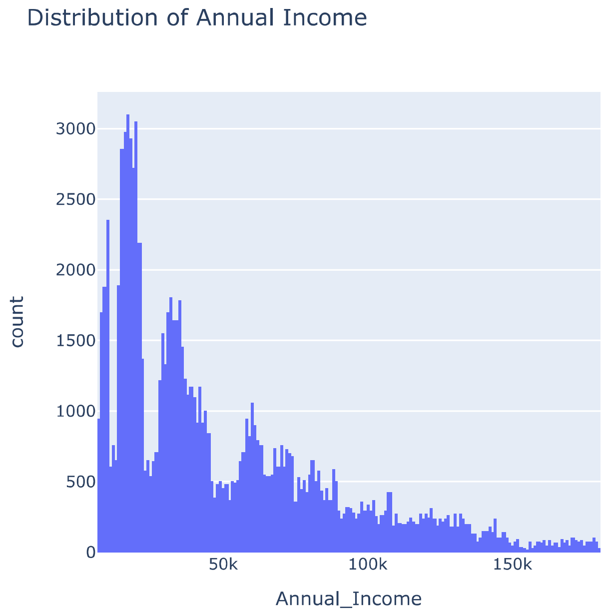
```
0
```

```
# Value count for each value
for i in df.columns:
    print(i,'\n',df[i].value_counts())
    print('-'*90)
```

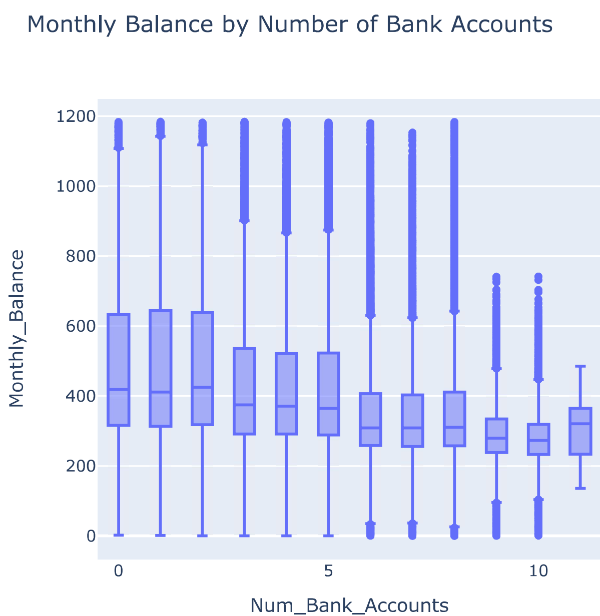
```
Total_EMI_per_month
 0.000000      10985
49.574949        8
46.354927        8
68.686857        8
137.778067       8
              ...
841.840387       1
82.544594        1
57.268474        1
144.179455       1
859.516628       1
Name: Total_EMI_per_month, Length: 11890, dtype: int64
------------------------------------------------------------------------------------------
Amount_invested_monthly
 0.000000      1920
110.634894       8
14.954207        8
82.215338        8
37.771077        8
              ...
58.064394        8
132.912602       8
83.539607        8
33.925532        8
24.028477        8
Name: Amount_invested_monthly, Length: 12261, dtype: int64
------------------------------------------------------------------------------------------
Payment_Behaviour
 Low_spent_Small_value_payments      28616
High_spent_Medium_value_payments    19738
High_spent_Large_value_payments     14726
Low_spent_Medium_value_payments     14399
High_spent_Small_value_payments     11764
Low_spent_Large_value_payments      10757
Name: Payment_Behaviour, dtype: int64
------------------------------------------------------------------------------------------
Monthly_Balance
 236.241829     8
1183.930696    8
235.451585     7
281.207616     7
1040.212843    7
              ..
469.067198     1
645.980641     1
463.759938     1
623.262089     1
393.673696     1
Name: Monthly_Balance, Length: 98492, dtype: int64
------------------------------------------------------------------------------------------
Credit_Score
 Standard    53174
Poor        28998
Good        17828
Name: Credit_Score, dtype: int64
------------------------------------------------------------------------------------------
```

EDA

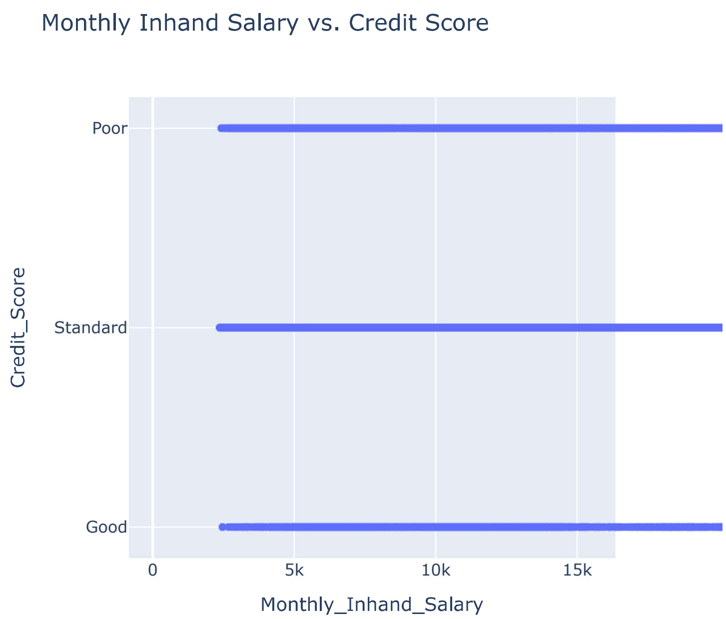
```
Income = px.histogram(df, x='Annual_Income', title='Distribution of Annual Income')
Income.show()
```




```
Mon_balance = px.box(df, x='Num_Bank_Accounts', y='Monthly_Balance', title='Monthly Balance by Number of Bank Accounts')
Mon_balance.show()
```




```
score = px.scatter(df, x='Monthly_Inhand_Salary', y='Credit_Score', title='Monthly Inhand Salary vs. Credit Score')
score.show()
```



Label encoder

```
# Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include=['object']).columns:
    label_encoders[column] = LabelEncoder()
    df[column] = label_encoders[column].fit_transform(df[column])

# Define features and target variable
# Sample data (replace with your actual data)
X = df[["Annual_Income", "Monthly_Inhand_Salary",
        "Num_Bank_Accounts", "Num_Credit_Card",
        "Interest_Rate", "Num_of_Loan",
        "Delay_from_due_date", "Num_of_Delayed_Payment",
        "Credit_Mix", "Outstanding_Debt",
        "Credit_History_Age", "Monthly_Balance"]]
y = df["Credit_Score"]
```

```
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
y
```

```
0        0
1        0
2        0
3        0
4        0
        ..
99995    1
99996    1
99997    1
99998    2
99999    1
Name: Credit_Score, Length: 100000, dtype: int64
```

Logistic Regression

```
# Initialize and train the Logistic Regression model
lr = LogisticRegression()
lr.fit(X_train, y_train)

# Make predictions and calculate accuracy
y_pred_lr = lr.predict(X_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

print(f"Logistic Regression Accuracy: {accuracy_lr}")
```

```
Logistic Regression Accuracy: 0.5479
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

Descision Tree

```
# Initialize and train the Decision Tree model
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# Make predictions and calculate accuracy
y_pred_dt = dt.predict(X_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)

print(f"Decision Tree Accuracy: {accuracy_dt}")
```

```
Decision Tree Accuracy: 0.76325
```

Random forest

```
# Initialize and train the Random Forest model
rf = RandomForestClassifier()
rf.fit(X_train, y_train)

# Make predictions and calculate accuracy
y_pred_rf = rf.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

print(f"Random Forest Accuracy: {accuracy_rf}")
```

```
Random Forest Accuracy: 0.8137
```

XG Boost

```
# Initialize and train the XGBoost model
xgb = XGBClassifier()
xgb.fit(X_train, y_train)

# Make predictions and calculate accuracy
y_pred_xgb = xgb.predict(X_test)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)

print(f"XGBoost Accuracy: {accuracy_xgb}")
```

```
XGBoost Accuracy: 0.7645
```

Comparing model

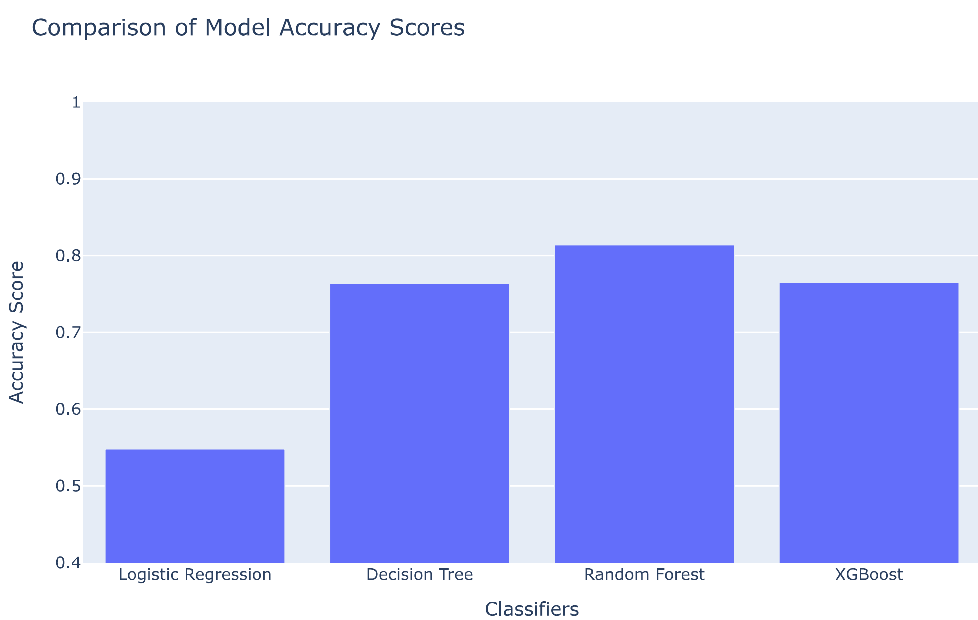
```
import plotly.graph_objects as go

# Accuracy scores for each classifier
accuracies = {
    'Logistic Regression': accuracy_lr,
    'Decision Tree': accuracy_dt,
    'Random Forest': accuracy_rf,
    'XGBoost': accuracy_xgb
}

# Create a bar graph using Plotly
fig = go.Figure(data=[
    go.Bar(name='Accuracy', x=list(accuracies.keys()), y=list(accuracies.values()))
])

# Update the layout
fig.update_layout(title='Comparison of Model Accuracy Scores',
                  xaxis_title='Classifiers',
                  yaxis_title='Accuracy Score',
                  yaxis_range=[0.4, 1.0])

# Show the plot
fig.show()
```



New Data Predict

```
# Sample data for prediction
sample_data = pd.DataFrame({
    'Annual_Income': [20867.670],
    'Monthly_Inhand_Salary': [6769.130000],
    'Num_Bank_Accounts': [6],
    'Num_Credit_Card': [5],
    'Interest_Rate': [8],
    'Num_of_Loan': [3],
    'Delay_from_due_date': [15],
    'Num_of_Delayed_Payment': [19],
    'Credit_Mix': [2],
    'Outstanding_Debt': [1109.03],
    'Credit_History_Age': [190],
    'Monthly_Balance': [236.241829]
})

# Predict using the trained RandomForestClassifier model
predictions = rf.predict(sample_data)
```

```
# Map the predicted values to labels
predicted_labels = [map_credit_mix(pred) for pred in predictions]

# Print the predicted labels
print(predicted_labels)
```

```
['Good']
```