BDSN Assignment using Pyspark

Credit Card Approval Prediction

YOGITA KUMARI PATEL

INTRODUCTION

Credit cards have become a part and parcel of our financial routine. They not only bring in convenience but also help tide over short-term crunches. Credit card enables the holder to make purchases of goods & services or withdraw advance cash on credit.

Being **approved** for a credit card can take time if you don't have a lengthy credit history or your credit score is recovering from a past mistake.

It is as important for the banks offering credit card facility as it allows banks to view an active credit history, based on your card repayments and card usage.

Banks launch a credit card with a lifestyle or travel brand and charge fee from the brand as they encourage the consumer to use the services of the brand by offering **extra rewards or direct discount**.

Why Credit Card Approval ?

- **Credit card issuers** consider credit scores while approving credit card applications. As credit bureaus calculate credit score on the basis of your past repayment behaviour, card issuer(s) use your credit score to judge your creditworthiness.
- Credit score is one of them most important factors behind the acceptance or rejection of credit cards as well as other loans. Most institutions have a standard credit score rate for considering the creditworthiness of the customer, i.e., whether he/she is reliable in terms of repayment of the credit availed from them.
- If the Creditworthiness is predicted/determined wrongly then this can lead to potential fraud and eventually **loss** of the Bank or Credit card issuer.

Dataset

There are two datasets on which this assignment has been made-

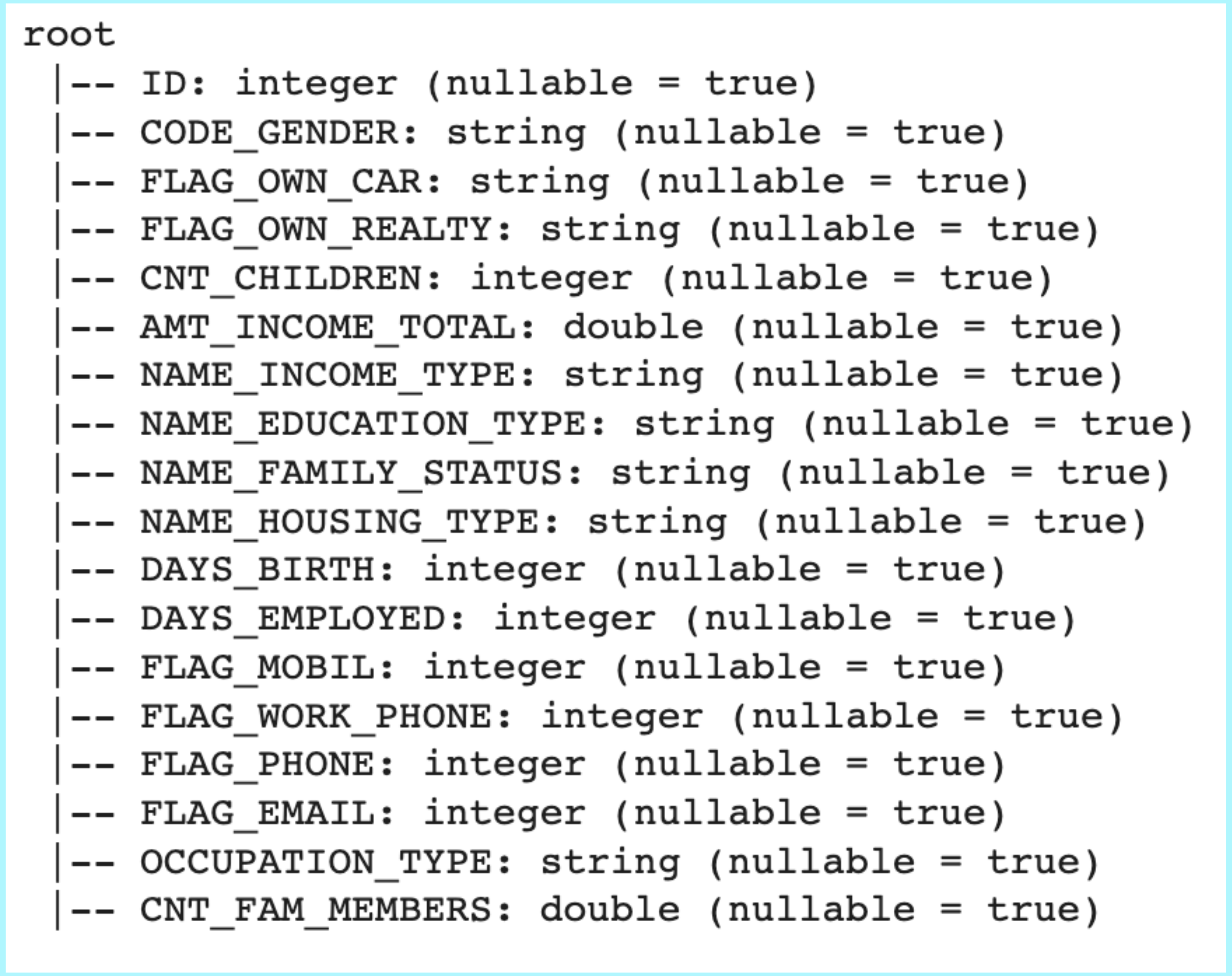
- Application Record Dataset
- Credit Record Dataset

Application Record Dataset

The Application_record dataset contains all the personal data of customers applying for credit card. This data is provided by the customer in the application form filled why them.

Dataset contains the following Features(After Renaming) -

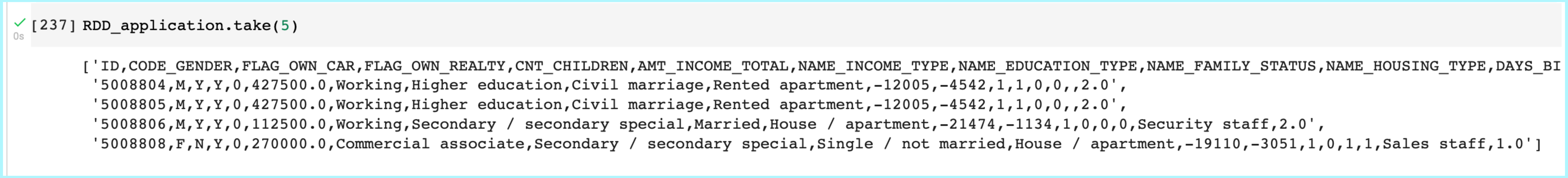
ID', 'Gender', 'Car', 'Realty', 'Children', 'Income', ‘Income_Type' 'Education_Type', 'Family_Status', 'Housing_Type', 'Age', 'Years_Experience' , 'Mobile_Phone', 'Work_Phone', 'Phone', 'Email', 'Job_Title', 'Total_Family'



```
root
 |-- ID: integer (nullable = true)
 |-- CODE_GENDER: string (nullable = true)
 |-- FLAG_OWN_CAR: string (nullable = true)
 |-- FLAG_OWN_REALTY: string (nullable = true)
 |-- CNT_CHILDREN: integer (nullable = true)
 |-- AMT_INCOME_TOTAL: double (nullable = true)
 |-- NAME_INCOME_TYPE: string (nullable = true)
 |-- NAME_EDUCATION_TYPE: string (nullable = true)
 |-- NAME_FAMILY_STATUS: string (nullable = true)
 |-- NAME_HOUSING_TYPE: string (nullable = true)
 |-- DAYS_BIRTH: integer (nullable = true)
 |-- DAYS_EMPLOYED: integer (nullable = true)
 |-- FLAG_MOBIL: integer (nullable = true)
 |-- FLAG_WORK_PHONE: integer (nullable = true)
 |-- FLAG_PHONE: integer (nullable = true)
 |-- FLAG_EMAIL: integer (nullable = true)
 |-- OCCUPATION_TYPE: string (nullable = true)
 |-- CNT_FAM_MEMBERS: double (nullable = true)
```

Application Record Dataset

There are total 438558 instances.



```
[237] RDD_application.take(5)

      ['ID,CODE_GENDER,FLAG_OWN_CAR,FLAG_OWN_REALTY,CNT_CHILDREN,AMT_INCOME_TOTAL,NAME_INCOME_TYPE,NAME_EDUCATION_TYPE,NAME_FAMILY_STATUS,NAME_HOUSING_TYPE,DAYS_BI
       '5008804,M,Y,Y,0,427500.0,Working,Higher education,Civil marriage,Rented apartment,-12005,-4542,1,1,0,0,,2.0',
       '5008805,M,Y,Y,0,427500.0,Working,Higher education,Civil marriage,Rented apartment,-12005,-4542,1,1,0,0,,2.0',
       '5008806,M,Y,Y,0,112500.0,Working,Secondary / secondary special,Married,House / apartment,-21474,-1134,1,0,0,0,Security staff,2.0',
       '5008808,F,N,Y,0,270000.0,Commercial associate,Secondary / secondary special,Single / not married,House / apartment,-19110,-3051,1,0,1,1,Sales staff,1.0']
```

Firstly created a Resilient Distributed Dataset(RDD) using code

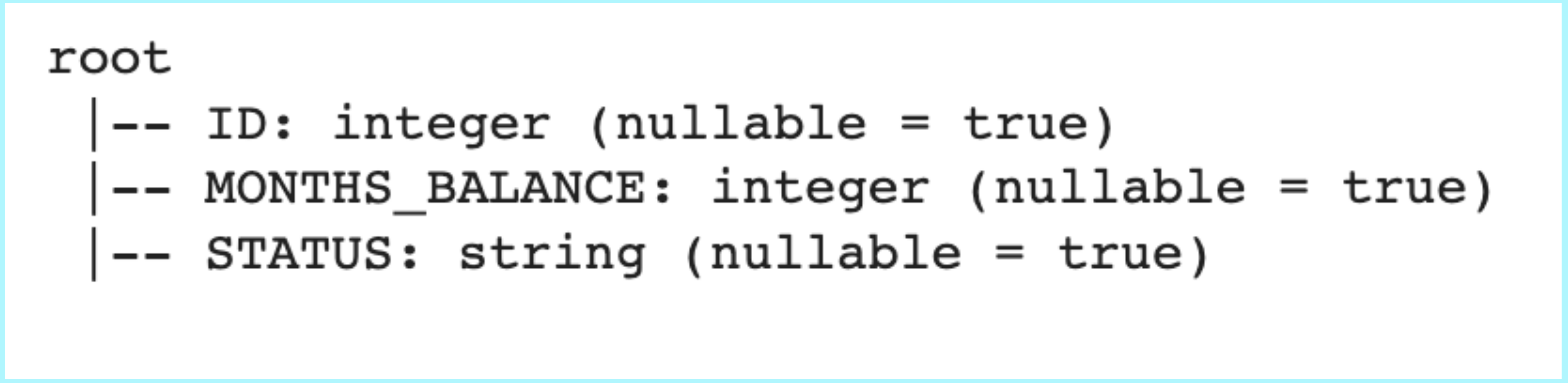
RDD_application = sc.textFile(‘application_record.csv’).

Then this RDD was mapped to a Dataframe

Credit Record Dataset

The Credit_record dataset contains credit card appliers status details along with their month's balance and ID.

Status is the categorical which will later be used to create the dependent variable.



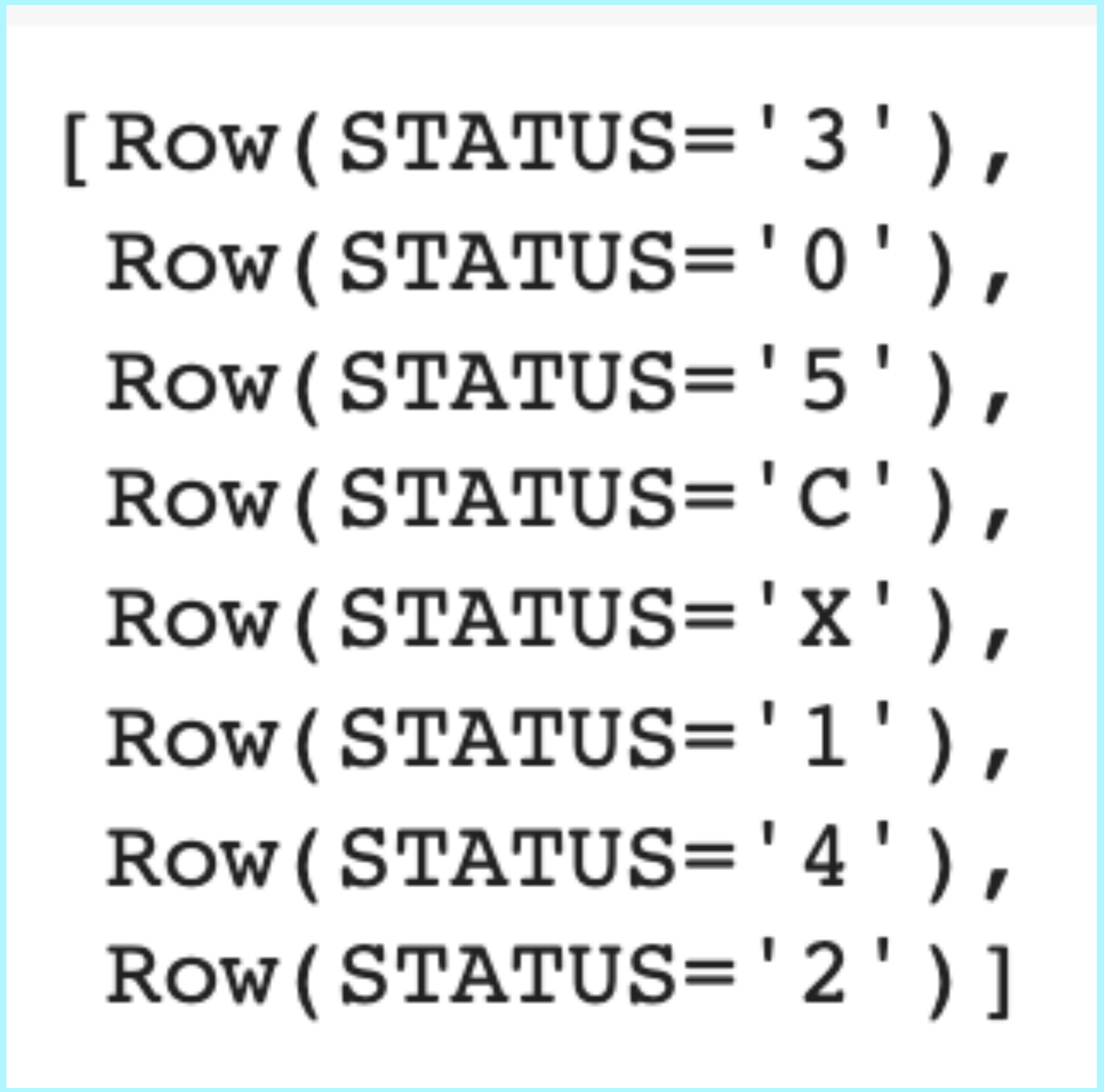
```
root
 |-- ID: integer (nullable = true)
 |-- MONTHS_BALANCE: integer (nullable = true)
 |-- STATUS: string (nullable = true)
```

Credit Record Dataset

Defining Status of Customer as Good and Bad-

Good : C, X

Bad : 0, 1, 2, 3, 4, 5



```
[Row(STATUS='3'),
 Row(STATUS='0'),
 Row(STATUS='5'),
 Row(STATUS='C'),
 Row(STATUS='X'),
 Row(STATUS='1'),
 Row(STATUS='4'),
 Row(STATUS='2')]
```

Modification of Dataset

- Converting Status into binary form

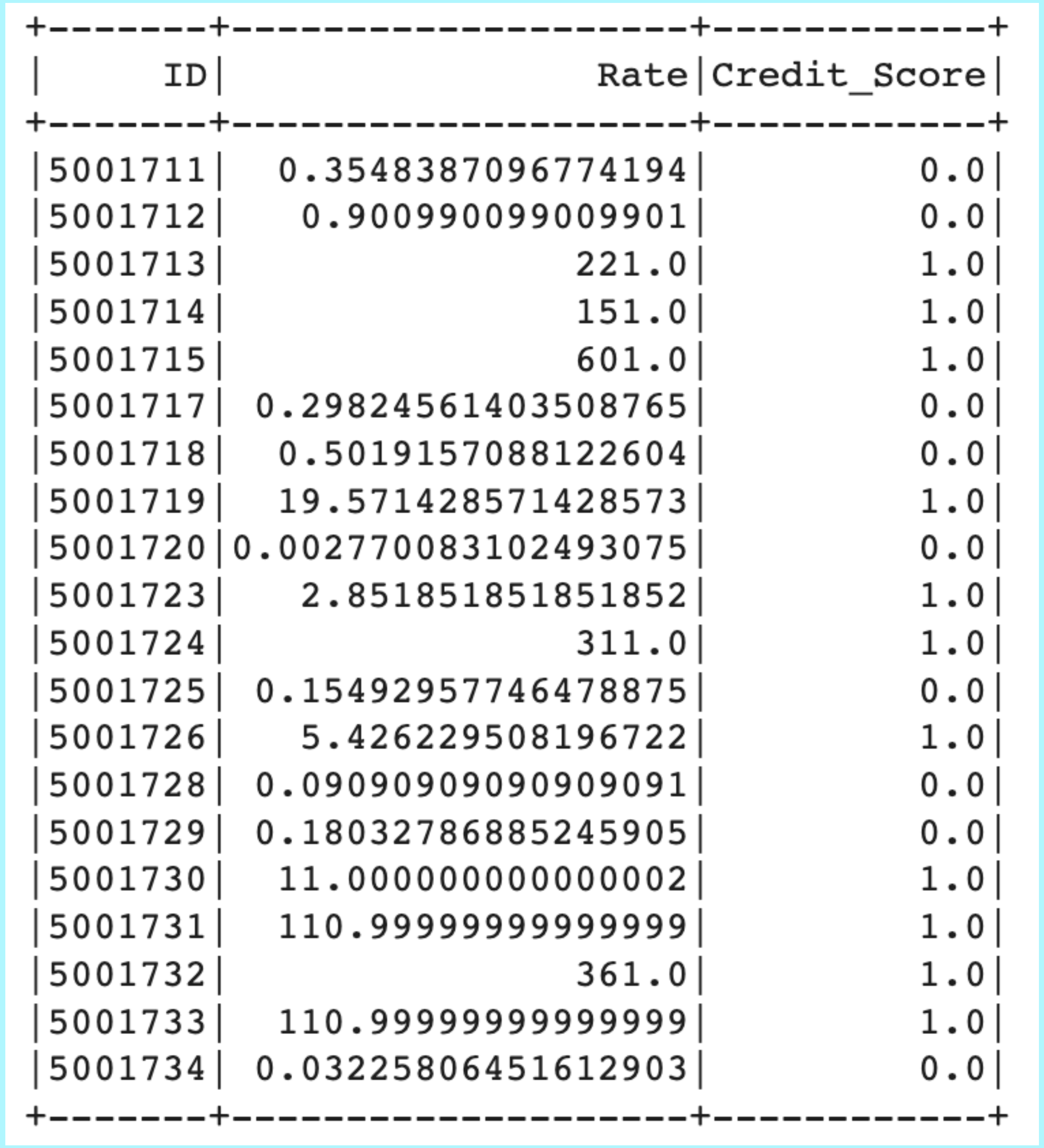
  0 - Good

  1 - Bad

- Aggregating the dataset by distinct count of Satus_Binary and grouping it with respect to ID and creating two new columns with total Good and Bad Status per customer ID.
- Creating new column **Credit Score** using withColumn.
- If no. of Good Status more than no. of Bad Status then Credit Score is set to 1, else set to 0.

New Credit Score Dataset

- **Rate** is the rate of good debts for each user.
- **Credit_Score** is the dependent variable.
- This dataset will now be merged with Application Record Dataset into **Final Dataframe(df_final)** by ID.

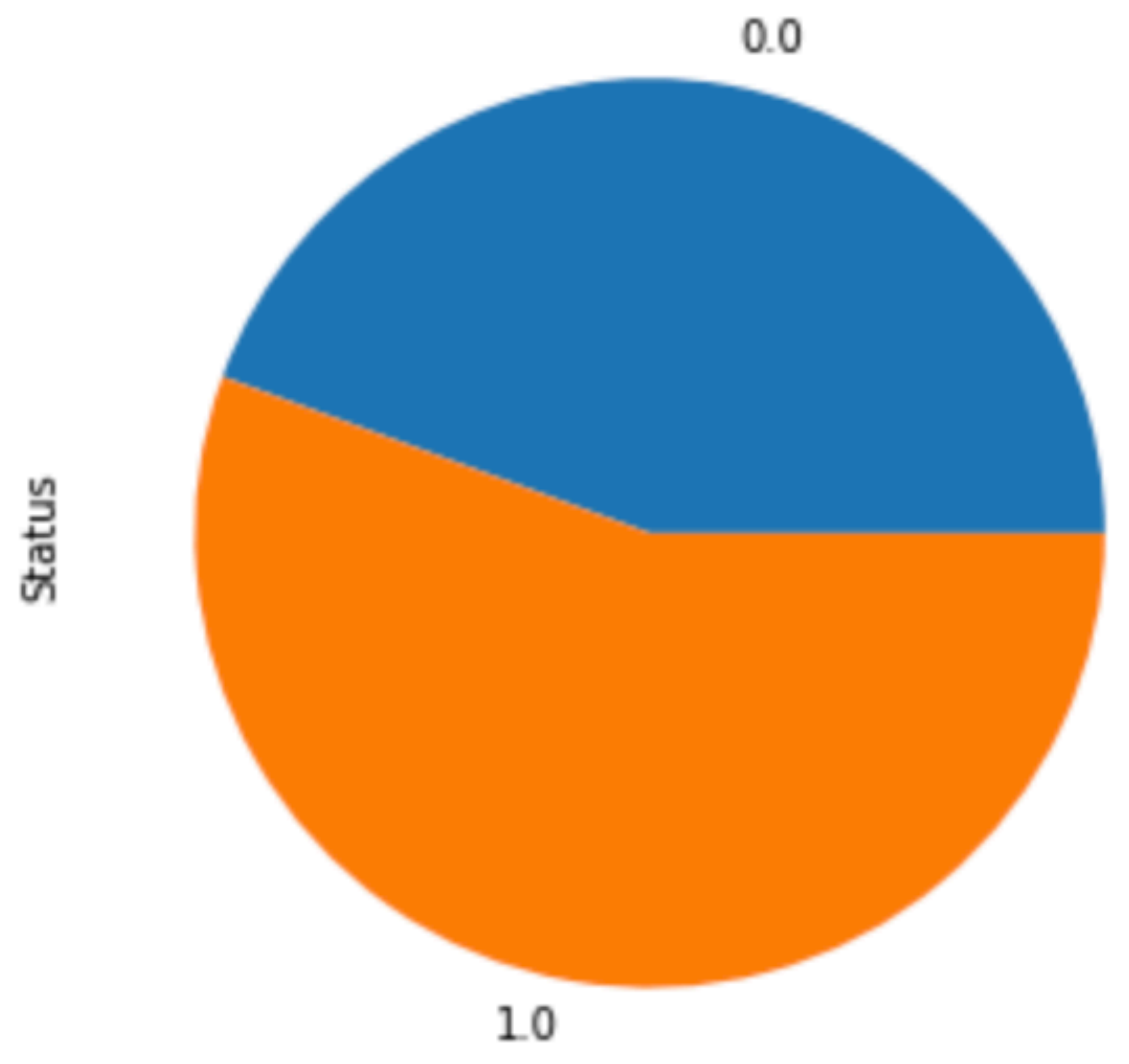


```
+-------+--------------------+------------+
|     ID|                Rate|Credit_Score|
+-------+--------------------+------------+
|5001711|  0.3548387096774194|         0.0|
|5001712|   0.900990099009901|         0.0|
|5001713|               221.0|         1.0|
|5001714|               151.0|         1.0|
|5001715|               601.0|         1.0|
|5001717| 0.29824561403508765|         0.0|
|5001718|  0.5019157088122604|         0.0|
|5001719|  19.571428571428573|         1.0|
|5001720|0.002770083102493075|         0.0|
|5001723|   2.851851851851852|         1.0|
|5001724|               311.0|         1.0|
|5001725| 0.15492957746478875|         0.0|
|5001726|   5.426229508196722|         1.0|
|5001728| 0.09090909090909091|         0.0|
|5001729| 0.18032786885245905|         0.0|
|5001730|  11.000000000000002|         1.0|
|5001731|  110.99999999999999|         1.0|
|5001732|               361.0|         1.0|
|5001733|  110.99999999999999|         1.0|
|5001734| 0.03225806451612903|         0.0|
+-------+--------------------+------------+
```

Exploratory Data Analysis

Total number of Good and Bad Credit Score

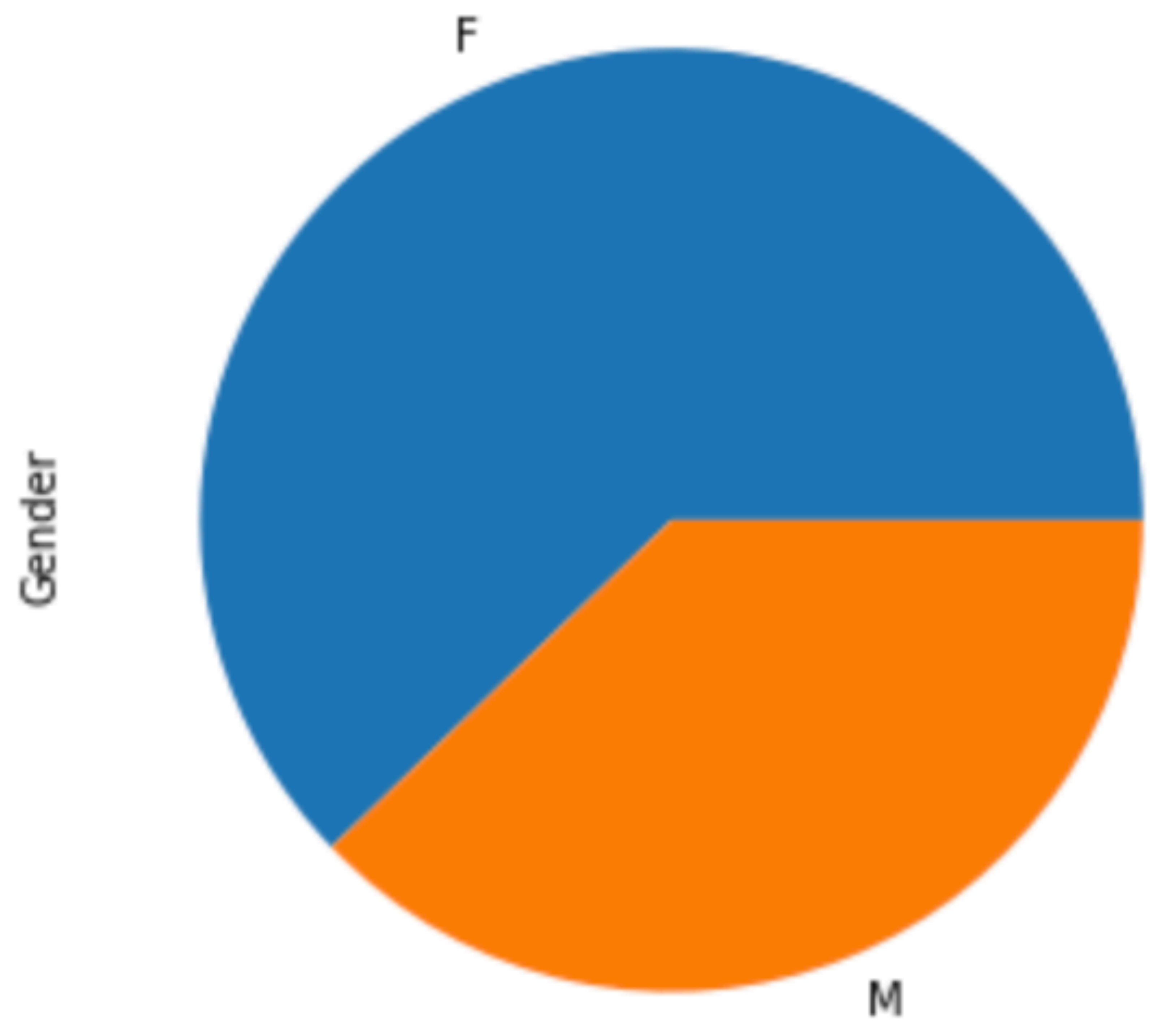
- There are more customers with Good Credit Score Status than customers with Bad Credit Score Status



Exploratory Data Analysis

Female Customers vs. Male Customers

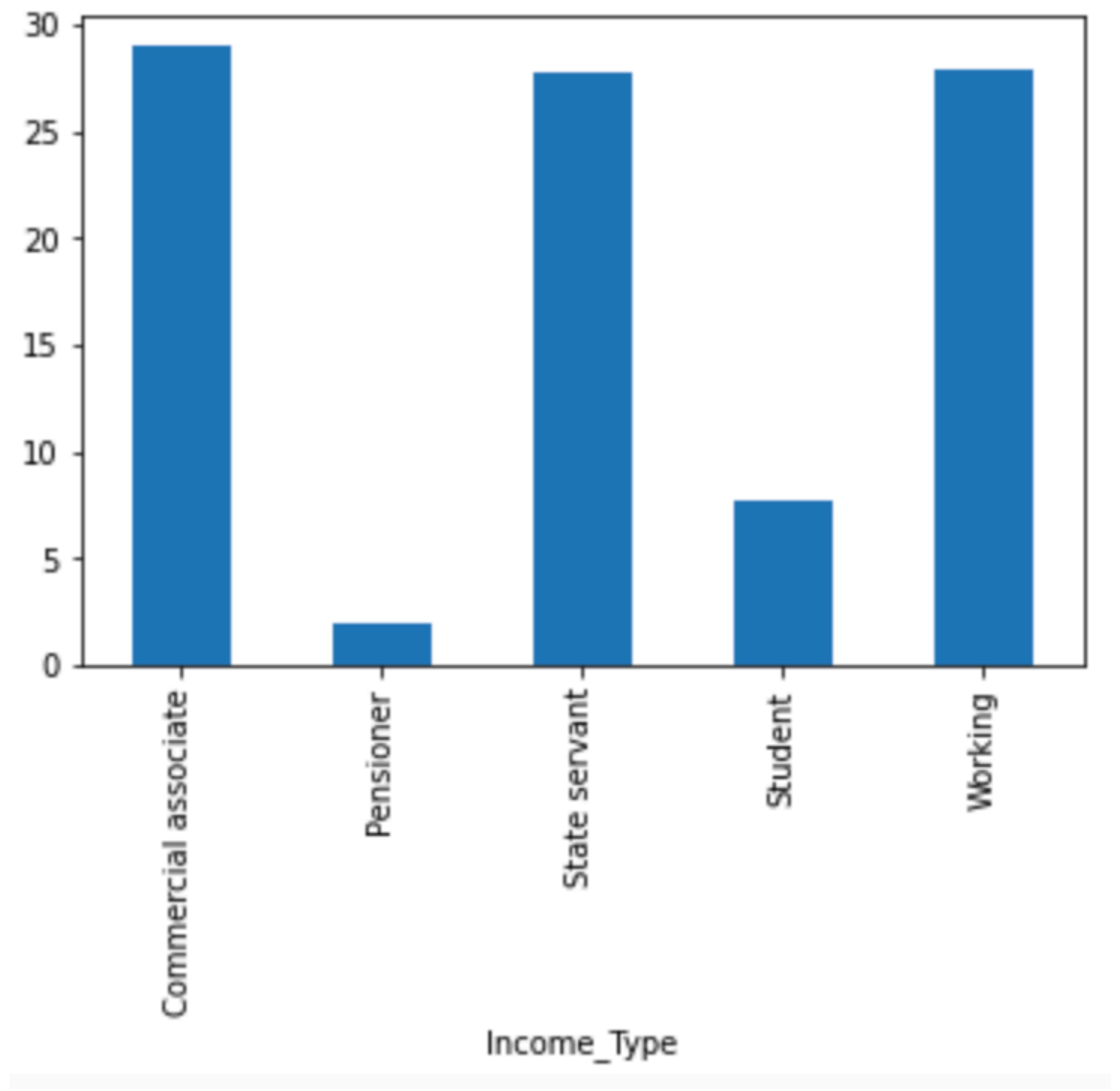
- Most of the applicants are Female.



Exploratory Data Analysis

Income Type vs. Rate

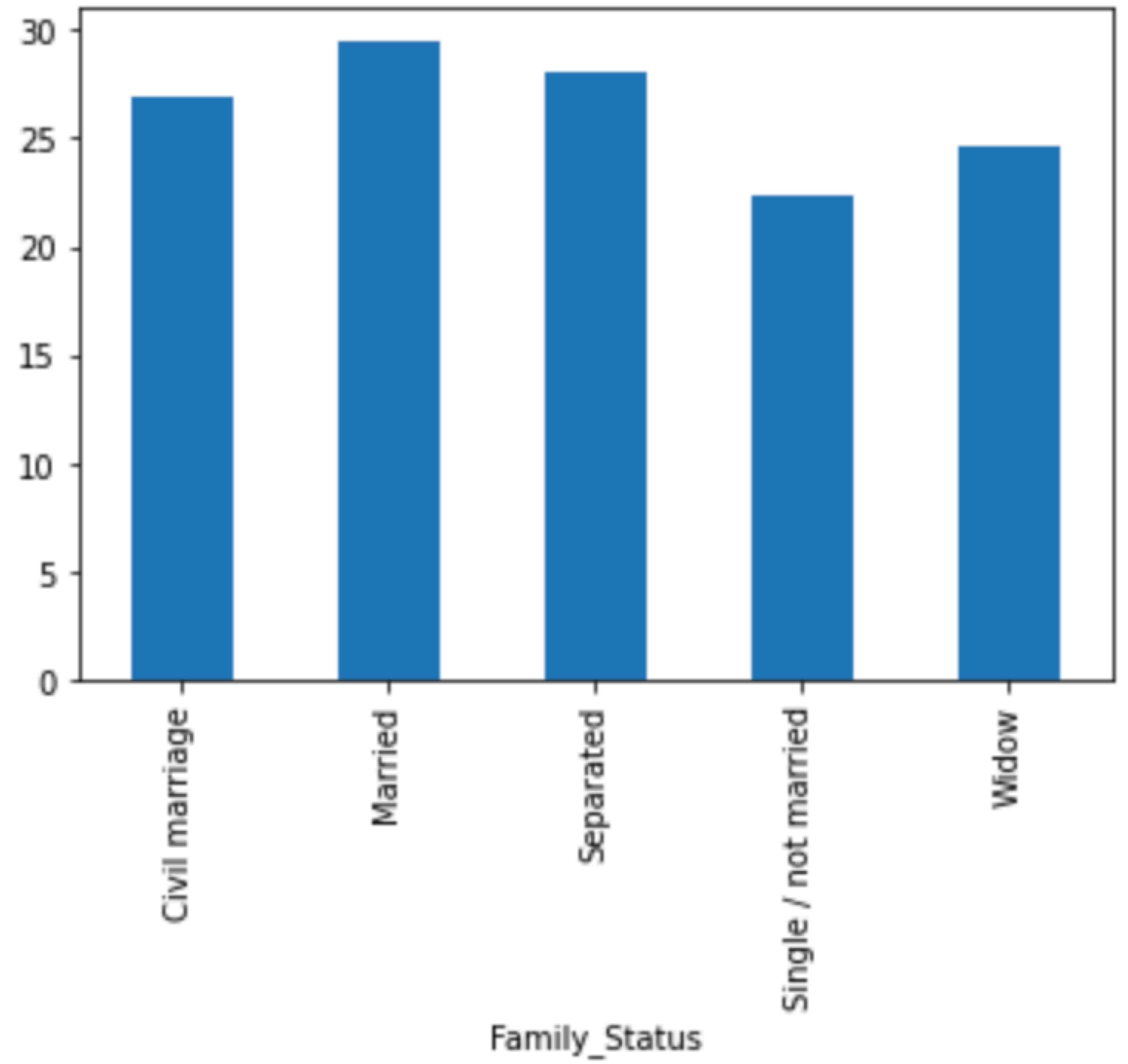
- Pensioners have the worst rate of Good debt followed by Students.
- Customers with Commercial Associates, State Servants and Working as profession have best rate.



Exploratory Data Analysis

Family Status vs. Rate

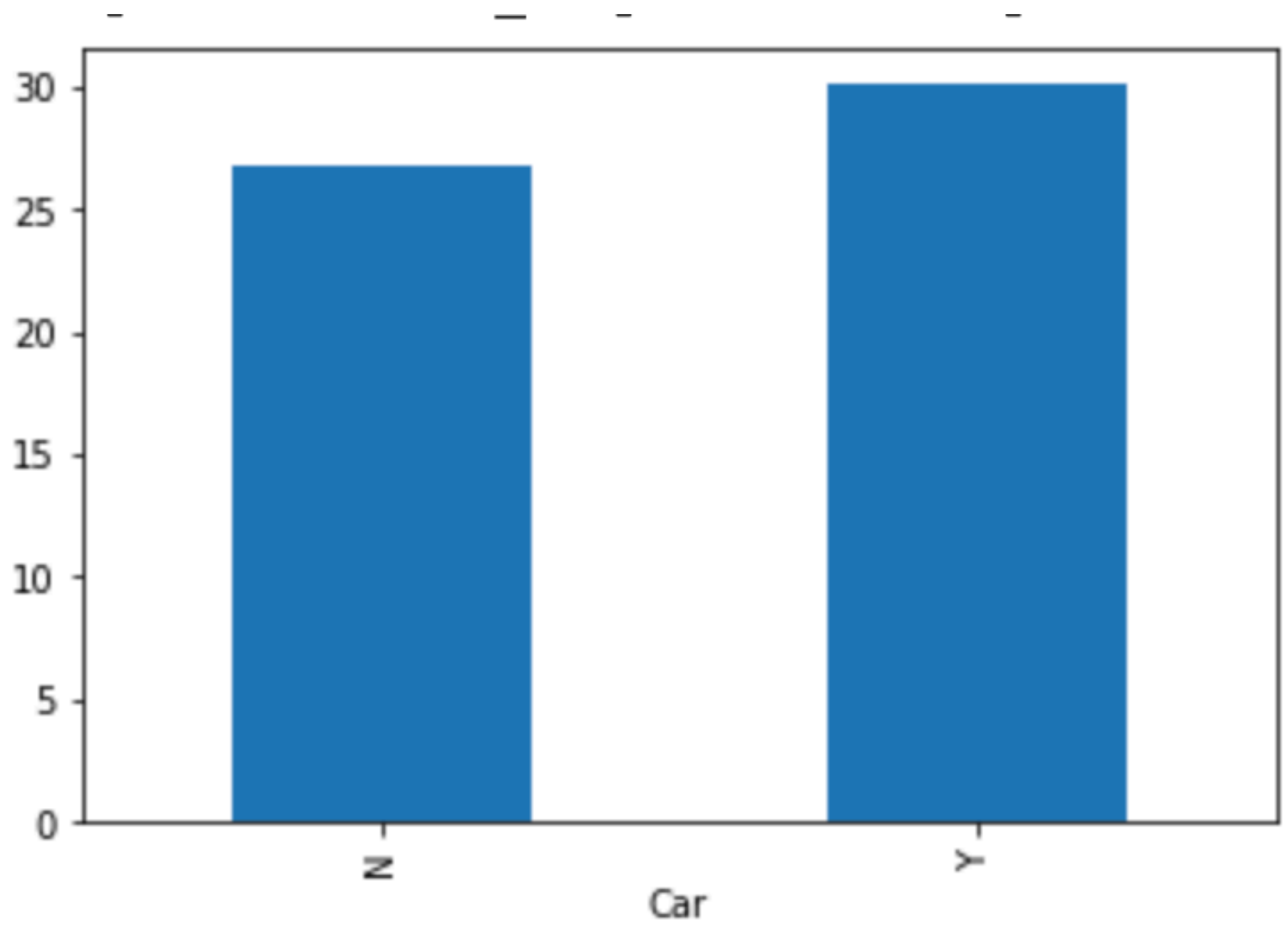
- Singles/ Not married customers have low rate of Good debt.
- Married customers have highest rate followed by Separated customers.



Exploratory Data Analysis

Car vs. Rate

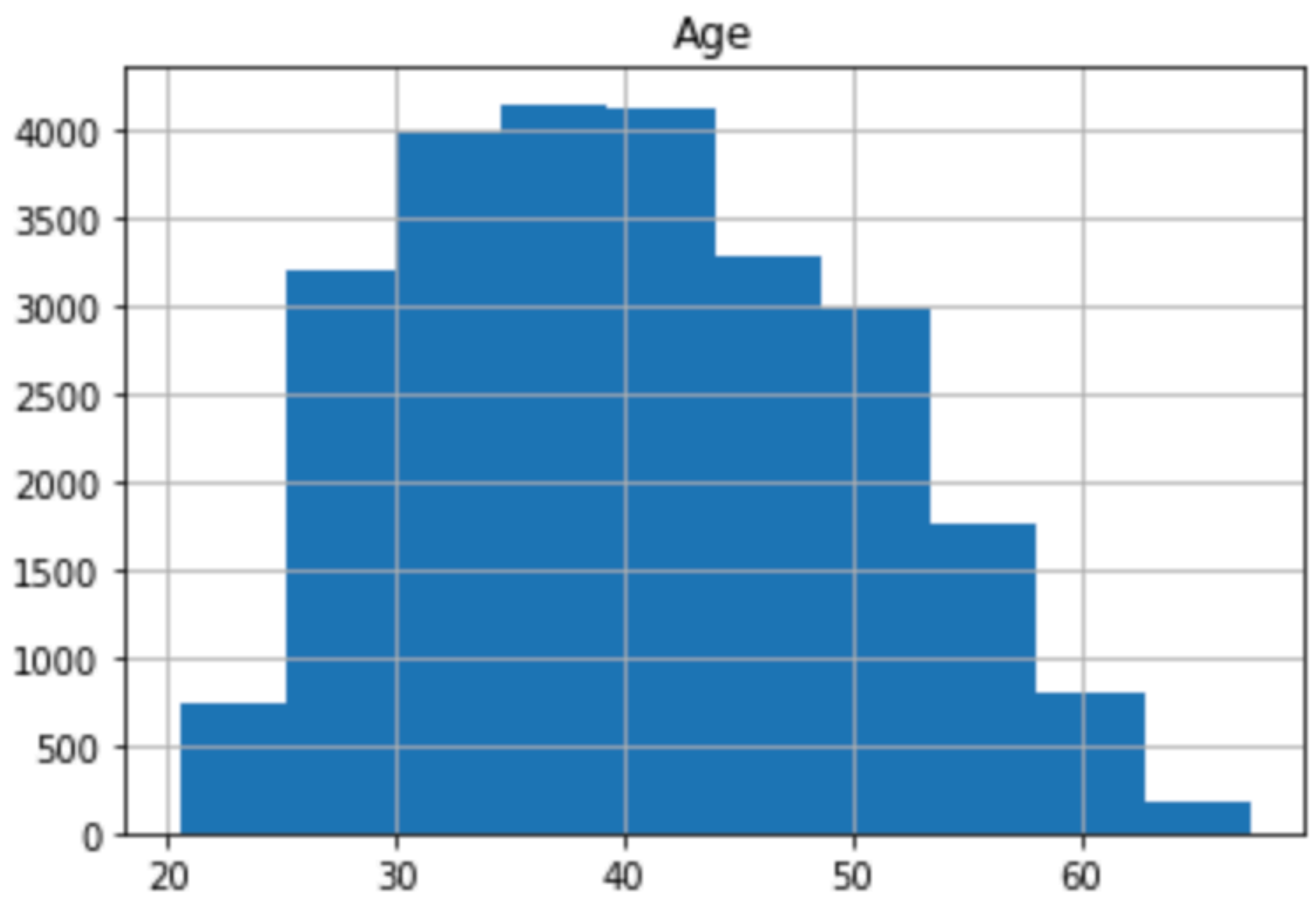
- Customers with cars have higher rate of good debt in comparison with customers owning no car.



Exploratory Data Analysis

Histogram of Age

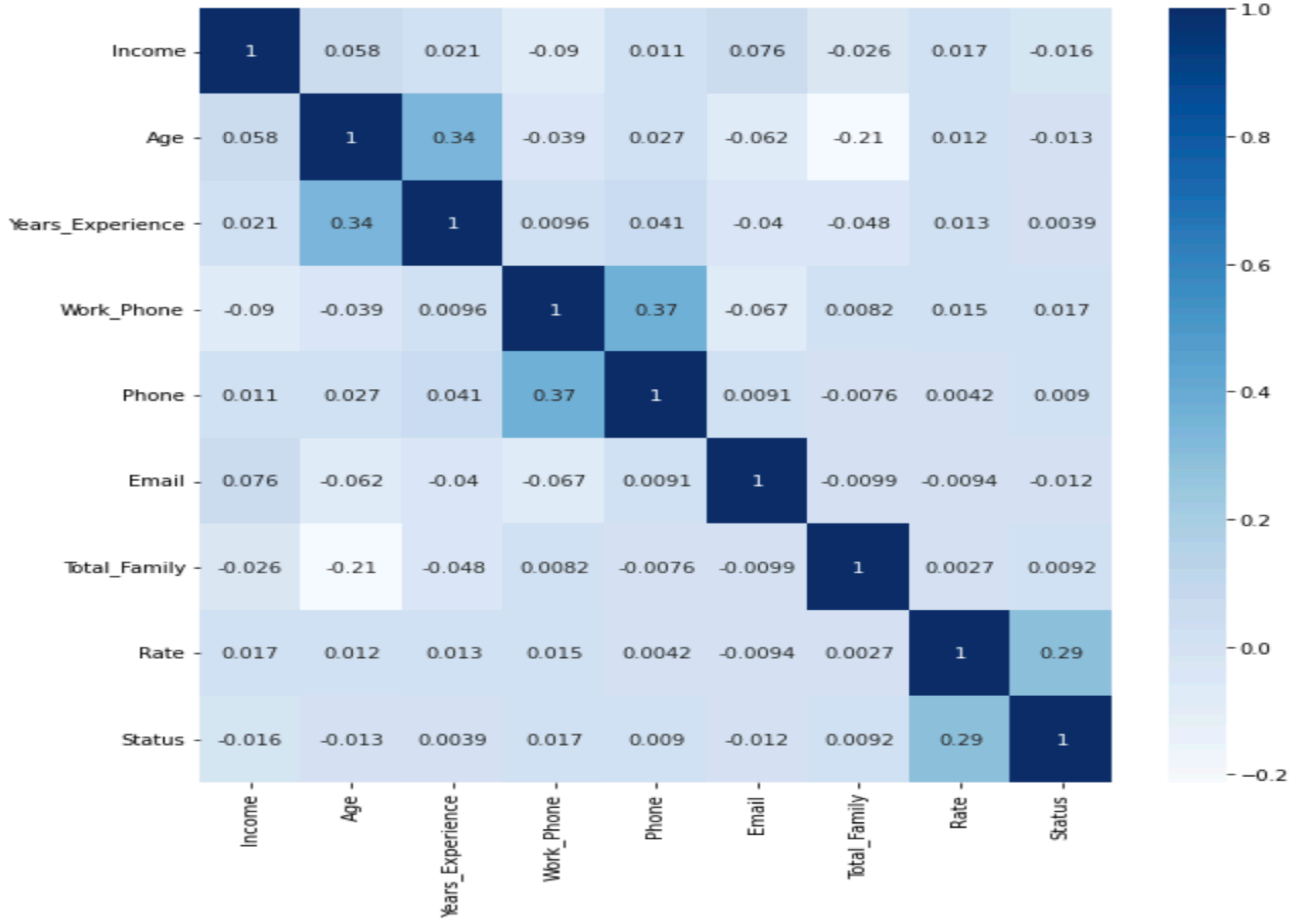
- Age is first converted into years from days since born.
- Most of the customers applying for credit card are around the age of 35-45.



Exploratory Data Analysis

Checking the Correlation using heat map

- Rate has highest correlation with Income, Work phone, Years of Experience and Age, other than Status.
- Children and Total Family members had a high correlation, so Children column was dropped.

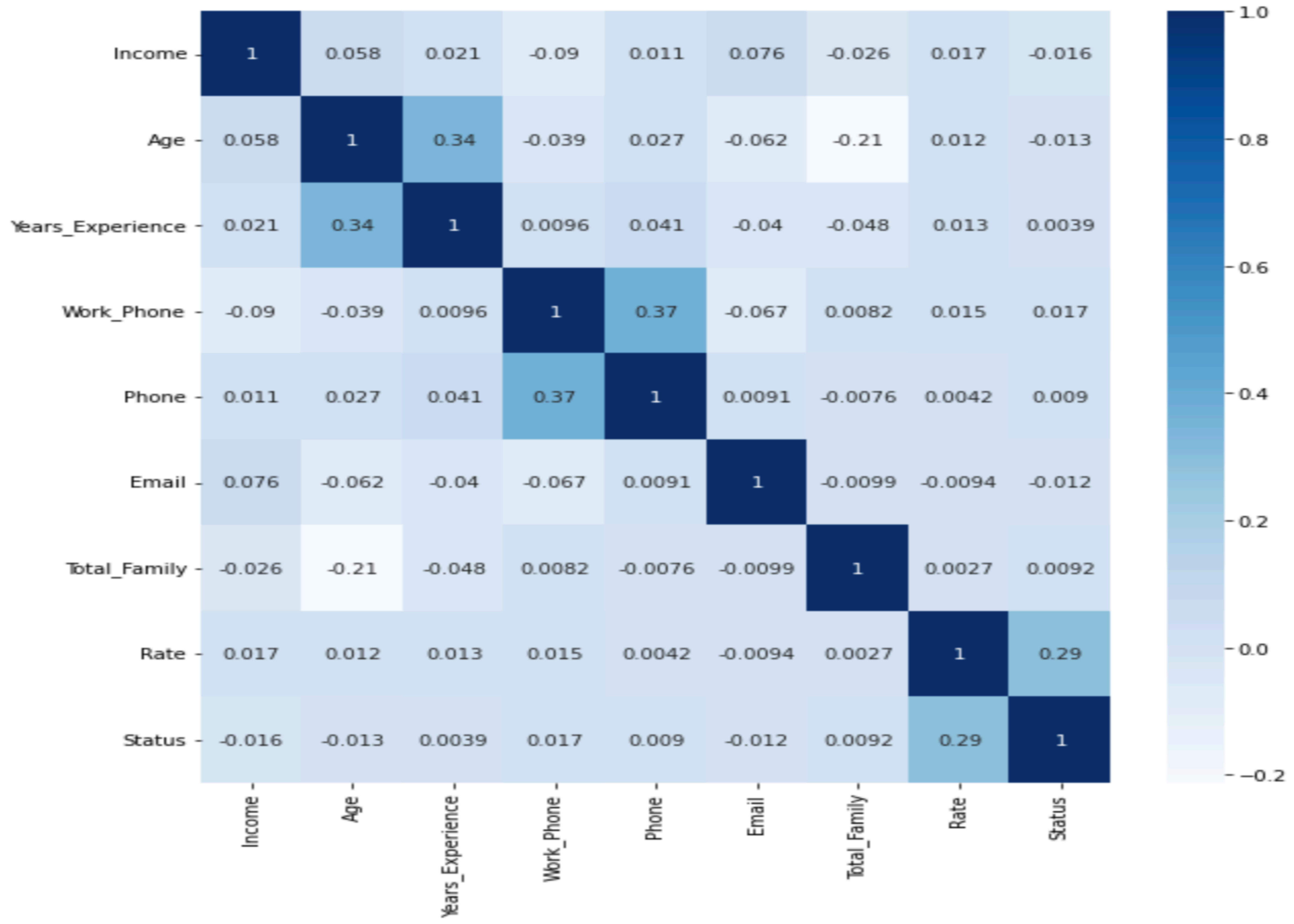


	Income	Age	Years_Experience	Work_Phone	Phone	Email	Total_Family	Rate	Status
Income	1	0.058	0.021	-0.09	0.011	0.076	-0.026	0.017	-0.016
Age	0.058	1	0.34	-0.039	0.027	-0.062	-0.21	0.012	-0.013
Years_Experience	0.021	0.34	1	0.0096	0.041	-0.04	-0.048	0.013	0.0039
Work_Phone	-0.09	-0.039	0.0096	1	0.37	-0.067	0.0082	0.015	0.017
Phone	0.011	0.027	0.041	0.37	1	0.0091	-0.0076	0.0042	0.009
Email	0.076	-0.062	-0.04	-0.067	0.0091	1	-0.0099	-0.0094	-0.012
Total_Family	-0.026	-0.21	-0.048	0.0082	-0.0076	-0.0099	1	0.0027	0.0092
Rate	0.017	0.012	0.013	0.015	0.0042	-0.0094	0.0027	1	0.29
Status	-0.016	-0.013	0.0039	0.017	0.009	-0.012	0.0092	0.29	1

Exploratory Data Analysis

Checking the Correlation using heat map

- Rate has highest correlation with Income, Work phone, Years of Experience and Age, other than Status.
- Children and Total Family members had a high correlation, so Children column was dropped.



	Income	Age	Years_Experience	Work_Phone	Phone	Email	Total_Family	Rate	Status
Income	1	0.058	0.021	-0.09	0.011	0.076	-0.026	0.017	-0.016
Age	0.058	1	0.34	-0.039	0.027	-0.062	-0.21	0.012	-0.013
Years_Experience	0.021	0.34	1	0.0096	0.041	-0.04	-0.048	0.013	0.0039
Work_Phone	-0.09	-0.039	0.0096	1	0.37	-0.067	0.0082	0.015	0.017
Phone	0.011	0.027	0.041	0.37	1	0.0091	-0.0076	0.0042	0.009
Email	0.076	-0.062	-0.04	-0.067	0.0091	1	-0.0099	-0.0094	-0.012
Total_Family	-0.026	-0.21	-0.048	0.0082	-0.0076	-0.0099	1	0.0027	0.0092
Rate	0.017	0.012	0.013	0.015	0.0042	-0.0094	0.0027	1	0.29
Status	-0.016	-0.013	0.0039	0.017	0.009	-0.012	0.0092	0.29	1

Preparation for Modelling

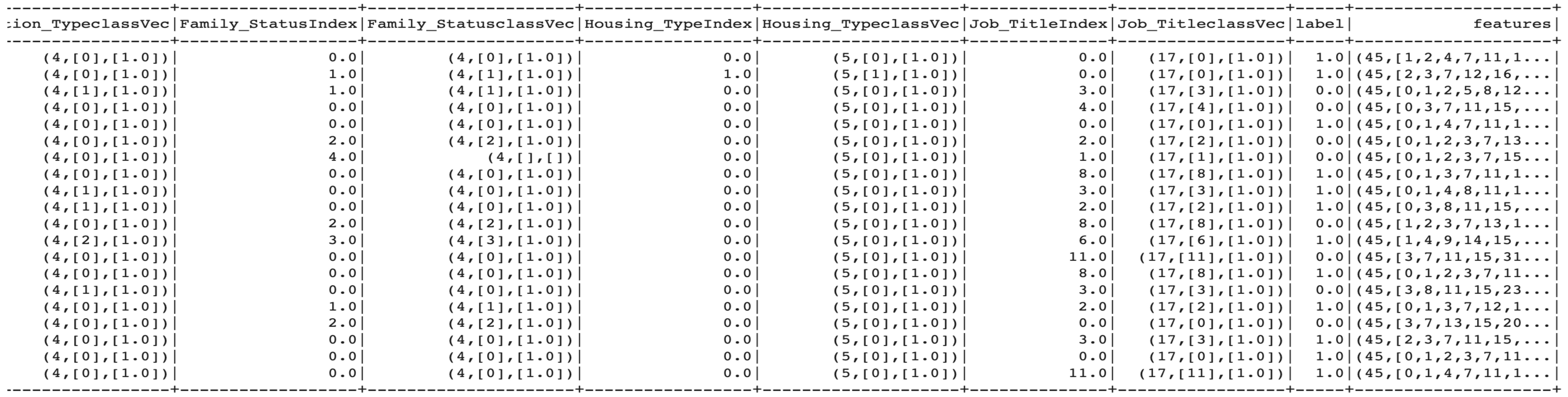
Category Indexing, One-Hot Encoding and VectorAssembler

- The process includes **Category Indexing, One-Hot Encoding and VectorAssembler**, a feature transformer that merges multiple columns into a vector column.
- Each categorical column is indexed using the **StringIndexer**, then the indexed categories are converted into one-hot encoded variables.
- The resulting output with binary vectors is then appended to the end of each row.
- Again we encoded our labels to label indices using the StringIndexer.
- Then, the VectorAssembler to combine all the feature columns into a single vector column.

Preparation for Modelling

Pipelines

- We use Pipeline to chain multiple Transformers and Estimators together to specify our machine learning workflow.
- A Pipeline's stages are specified as an ordered array.



```
----------------+------------------+---------------------+-----------------+--------------------+--------------+-----------------+-----+--------------------+
ion_TypeclassVec|Family_StatusIndex|Family_StatusclassVec|Housing_TypeIndex|Housing_TypeclassVec|Job_TitleIndex|Job_TitleclassVec|label|            features|
----------------+------------------+---------------------+-----------------+--------------------+--------------+-----------------+-----+--------------------+
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           0.0|   (17,[0],[1.0])|  1.0|(45,[1,2,4,7,11,1...|
   (4,[0],[1.0])|               1.0|        (4,[1],[1.0])|              1.0|       (5,[1],[1.0])|           0.0|   (17,[0],[1.0])|  1.0|(45,[2,3,7,12,16,...|
   (4,[1],[1.0])|               1.0|        (4,[1],[1.0])|              0.0|       (5,[0],[1.0])|           3.0|   (17,[3],[1.0])|  0.0|(45,[0,1,2,5,8,12...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           4.0|   (17,[4],[1.0])|  0.0|(45,[0,3,7,11,15,...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           0.0|   (17,[0],[1.0])|  1.0|(45,[0,1,4,7,11,1...|
   (4,[0],[1.0])|               2.0|        (4,[2],[1.0])|              0.0|       (5,[0],[1.0])|           2.0|   (17,[2],[1.0])|  0.0|(45,[0,1,2,3,7,13...|
   (4,[0],[1.0])|               4.0|            (4,[],[])|              0.0|       (5,[0],[1.0])|           1.0|   (17,[1],[1.0])|  0.0|(45,[0,1,2,3,7,15...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           8.0|   (17,[8],[1.0])|  1.0|(45,[0,1,3,7,11,1...|
   (4,[1],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           3.0|   (17,[3],[1.0])|  1.0|(45,[0,1,4,8,11,1...|
   (4,[1],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           2.0|   (17,[2],[1.0])|  1.0|(45,[0,3,8,11,15,...|
   (4,[0],[1.0])|               2.0|        (4,[2],[1.0])|              0.0|       (5,[0],[1.0])|           8.0|   (17,[8],[1.0])|  0.0|(45,[1,2,3,7,13,1...|
   (4,[2],[1.0])|               3.0|        (4,[3],[1.0])|              0.0|       (5,[0],[1.0])|           6.0|   (17,[6],[1.0])|  1.0|(45,[1,4,9,14,15,...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|          11.0|  (17,[11],[1.0])|  0.0|(45,[3,7,11,15,31...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           8.0|   (17,[8],[1.0])|  1.0|(45,[0,1,2,3,7,11...|
   (4,[1],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           3.0|   (17,[3],[1.0])|  0.0|(45,[3,8,11,15,23...|
   (4,[0],[1.0])|               1.0|        (4,[1],[1.0])|              0.0|       (5,[0],[1.0])|           2.0|   (17,[2],[1.0])|  1.0|(45,[0,1,3,7,12,1...|
   (4,[0],[1.0])|               2.0|        (4,[2],[1.0])|              0.0|       (5,[0],[1.0])|           0.0|   (17,[0],[1.0])|  0.0|(45,[3,7,13,15,20...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           3.0|   (17,[3],[1.0])|  1.0|(45,[2,3,7,11,15,...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|           0.0|   (17,[0],[1.0])|  1.0|(45,[0,1,2,3,7,11...|
   (4,[0],[1.0])|               0.0|        (4,[0],[1.0])|              0.0|       (5,[0],[1.0])|          11.0|  (17,[11],[1.0])|  1.0|(45,[0,1,4,7,11,1...|
----------------+------------------+---------------------+-----------------+--------------------+--------------+-----------------+-----+--------------------+
```

Modelling

Splitting the data into Training ans Testing data

- We split the dataset in train and test set using **randomSplit() i**n 0.8 and 0.2 respectively.
- Training dataset count : 20024
- Test Dataset Count : 5110

Modelling

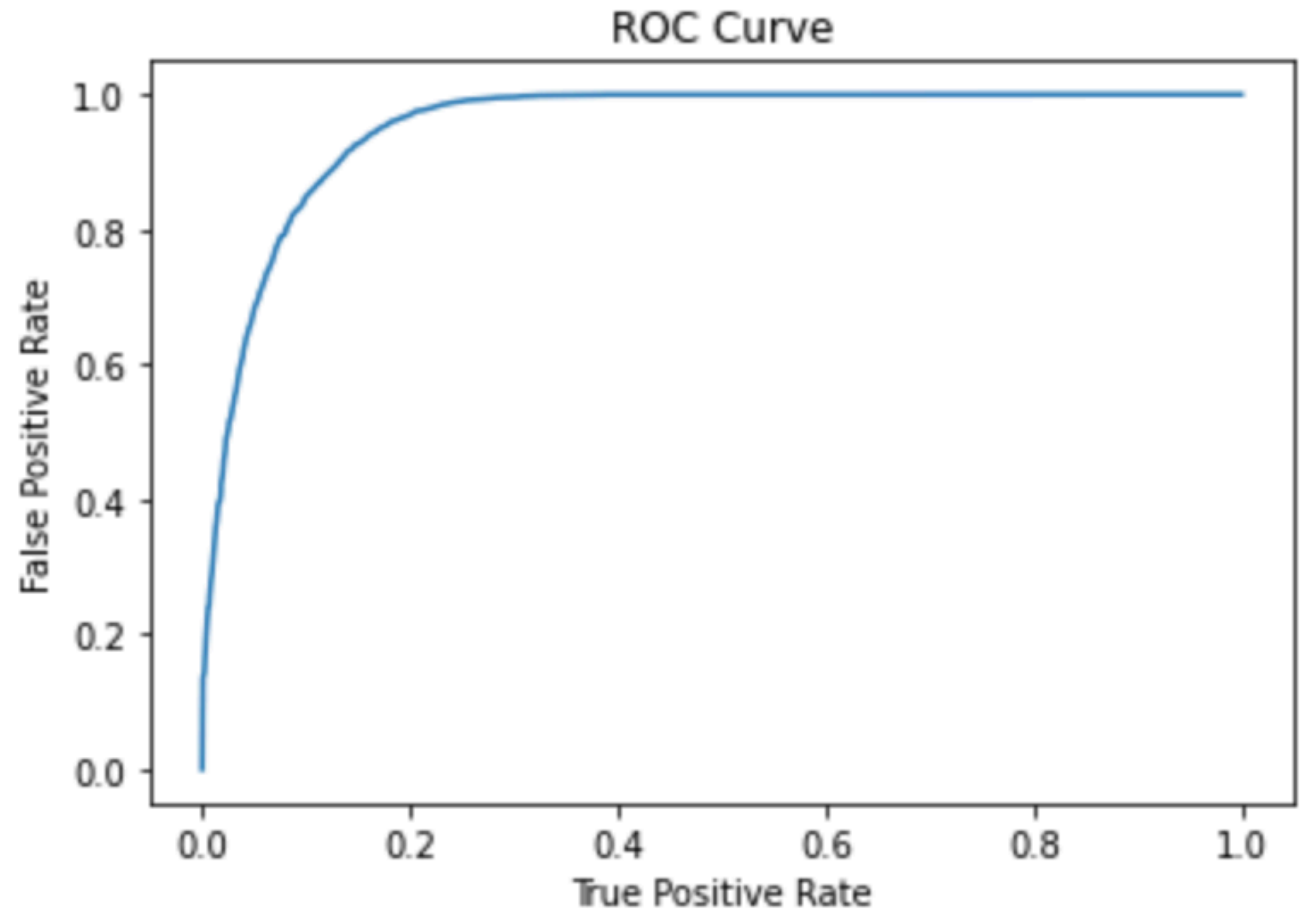
Logistic Regression

- **Logistic regression** is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist.
- In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).
- It is a process of modeling the probability of a discrete outcome given an input variable.

Logistic Regression

ROC Curve

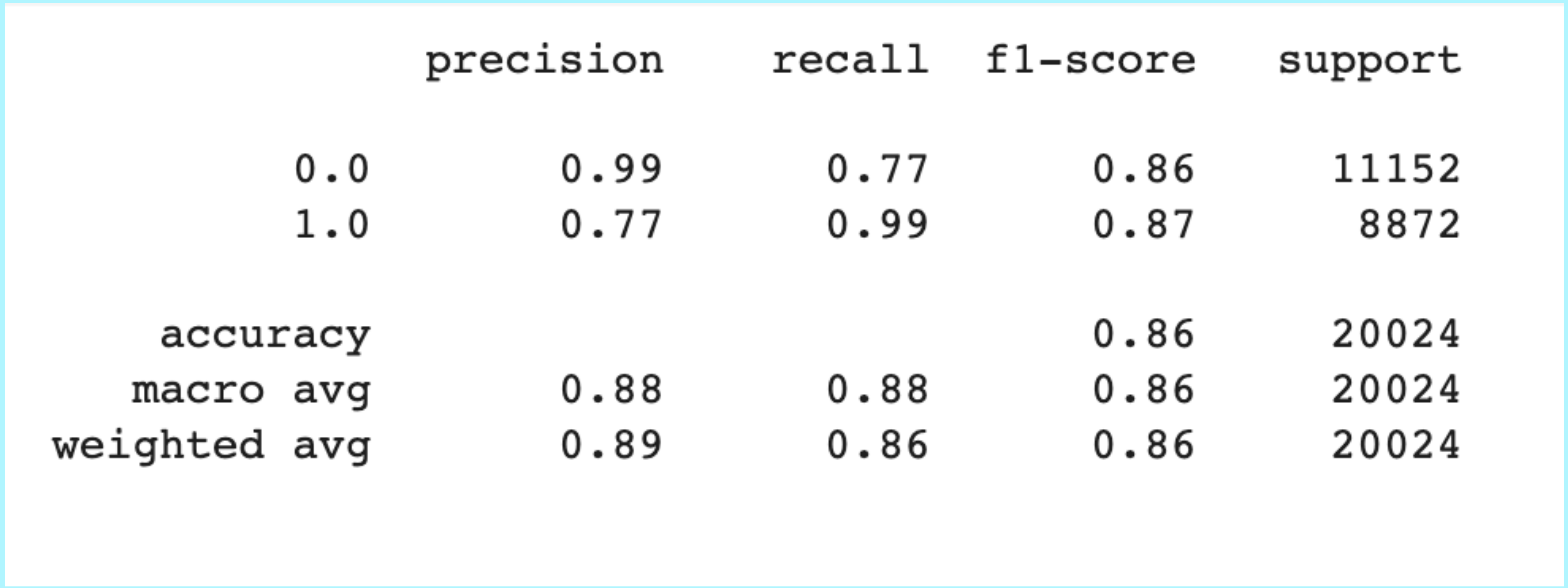
- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds.
- This curve plots two parameters: True Positive Rate. False Positive Rate.
- Training Area under ROC is **0.95275** where as test areas under ROC is **0.9542678**
- BinaryClassificationEvaluator is used for computing test area under ROC



```
Training set areaUnderROC: 0.952753640610668
```

Classification Report

- Accuracy is 0.86
- Precision for 0 class is 0.99 whereas only 0.77 for class 1
- F1- Score for both class 0 and 1 are almost same - 0.86 and 0.87 respectively.



```
              precision    recall  f1-score   support

         0.0       0.99      0.77      0.86     11152
         1.0       0.77      0.99      0.87      8872

    accuracy                           0.86     20024
   macro avg       0.88      0.88      0.86     20024
weighted avg       0.89      0.86      0.86     20024
```

THANK
YOU!