**Loan Application Status Prediction**

**Problem Definition:**

In banks a **loan** is the lending of money by one or more individuals, organizations, or other entities to other individuals, organizations, etc. The recipient (i.e. the borrower) incurs a debt, and is usually liable to pay interest on that debt until it is repaid, and also to repay the principal amount borrowed. To read more check out Wikipedia. The whole process of ascertaining if a burrower would pay back loans might be tedious hence the need to automate the procedure.

In this blog post, I’d be walking us through Loan prediction using some selected Machine Learning Algorithms.

**Source of Dataset:** Datatrained Institution.

The problem: The major aim of this project is to predict which of the customers will have their loan paid or not.

Therefore, this is a classification problem to be trained with algorithms like:

1. Logistic Regression
2. Decision Tree
3. Random Forest

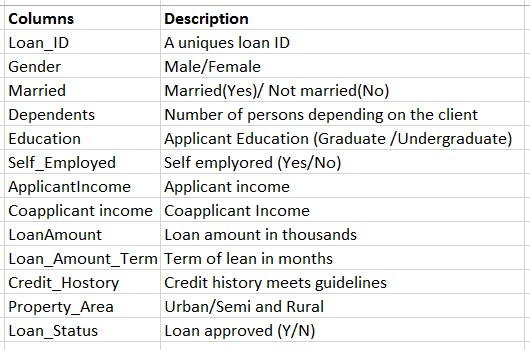
The machine learning classifier that can be used is not limited to the aforementioned. Other models like XGBoost, CatBoost and the likes can be applied in the training of the model. The choice of these three algorithms is sequel upon the desire to keep the model explanatory of itself and also, the data set is small.

**Disclaimer:**

1. In this project, default hyper-parameter values are employed.

2. More visualization can be done beyond what’s executed in this post.

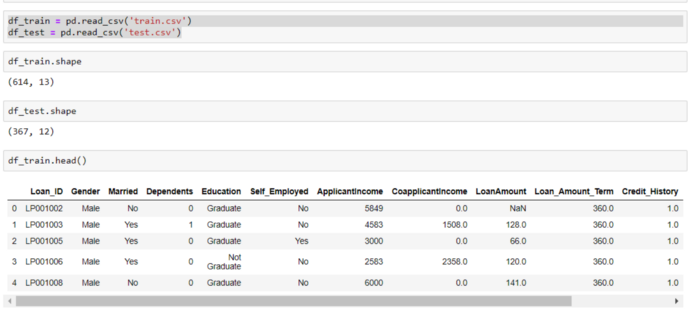
3. The training data set provided is the focus because we are not making a submission to kaggle for scoring. Hence, we split the train into a validation set to get our evaluations estimated.



This table shows the variable names and their corresponding descriptions in our Dataset.

**Importing the required libraries to build a model:**

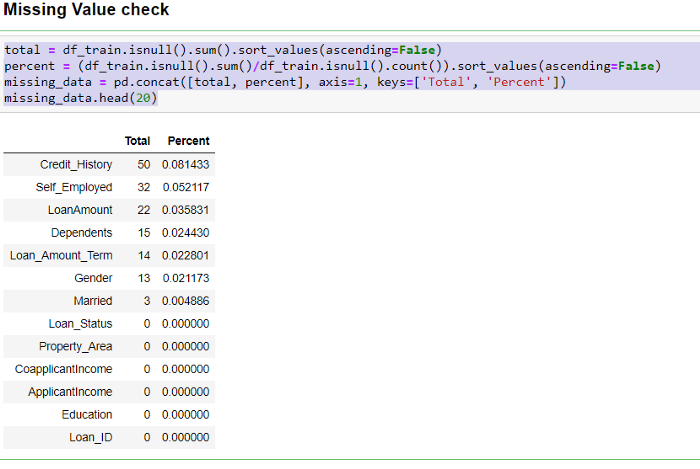
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
%matplotlib inline



Reading the data and checked the shape. It has has 614 rows and 13 columns.

**Missing Values:**Checked where we have missing values and fixed them appropriately

total = df\_train.isnull().sum().sort\_values(ascending=False)  
percent = (df\_train.isnull().sum()/df\_train.isnull().count()).sort\_values(ascending=False)  
missing\_data = pd.concat([total, percent], axis=1, keys=[‘Total’, ‘Percent’])  
missing\_data.head(20)

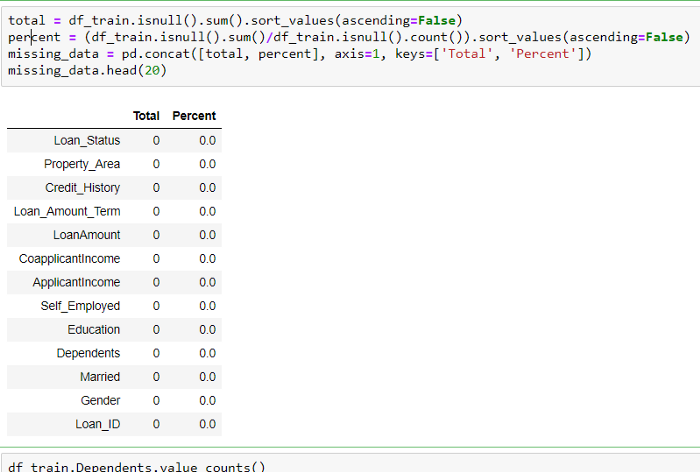


Variable: Credit\_History, Self\_Employed, Loan Amount, Dependents, Loan\_Amount\_Term, Gender and Married have missing values.

Filling missing values:

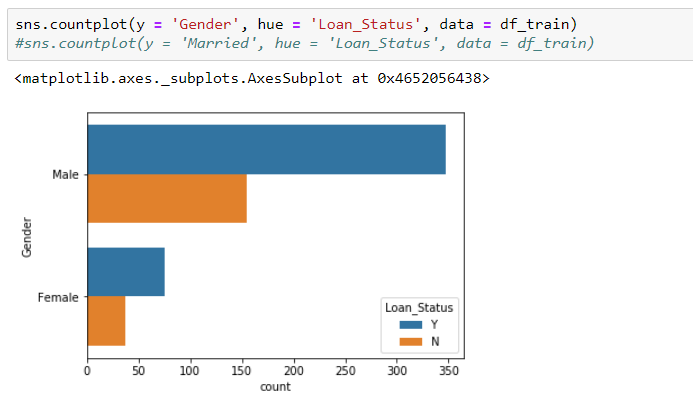
df\_train[‘Gender’] = df\_train[‘Gender’].fillna(  
df\_train[‘Gender’].dropna().mode().values[0] )  
df\_train[‘Married’] = df\_train[‘Married’].fillna(  
df\_train[‘Married’].dropna().mode().values[0] )  
df\_train[‘Dependents’] = df\_train[‘Dependents’].fillna(  
df\_train[‘Dependents’].dropna().mode().values[0] )  
df\_train[‘Self\_Employed’] = df\_train[‘Self\_Employed’].fillna(

df\_train[‘Self\_Employed’].dropna().mode().values[0] )  
df\_train[‘LoanAmount’] = df\_train[‘LoanAmount’].fillna(  
df\_train[‘LoanAmount’].dropna().median() )  
df\_train[‘Loan\_Amount\_Term’] = df\_train[‘Loan\_Amount\_Term’].fillna(  
df\_train[‘Loan\_Amount\_Term’].dropna().mode().values[0] )  
df\_train[‘Credit\_History’] = df\_train[‘Credit\_History’].fillna(  
df\_train[‘Credit\_History’].dropna().mode().values[0] )

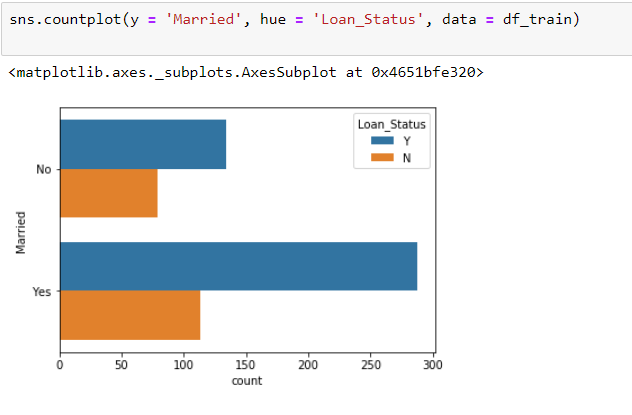


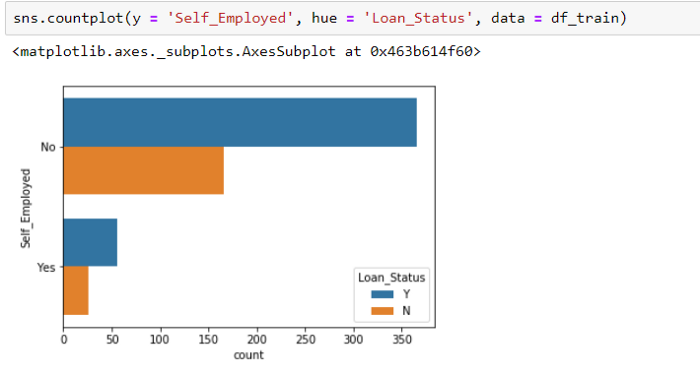
Missing values are treated now.

**Exploratory Data Analysis: We want to show the power of visualizations**

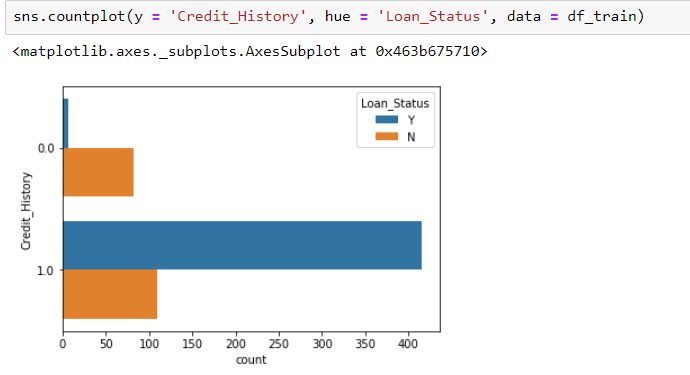


We have more males are on loan than females.

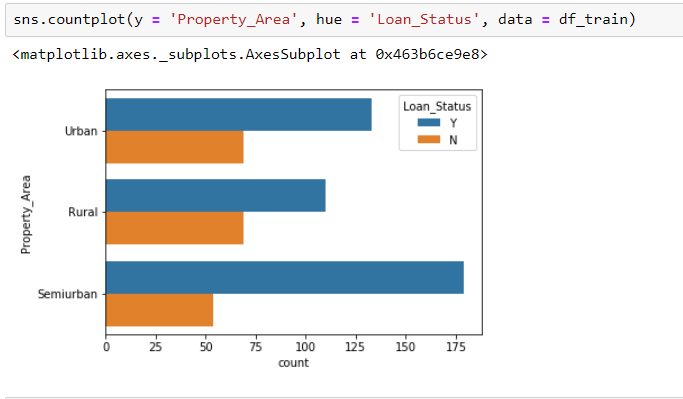
Married people collect more loan than unmarried.



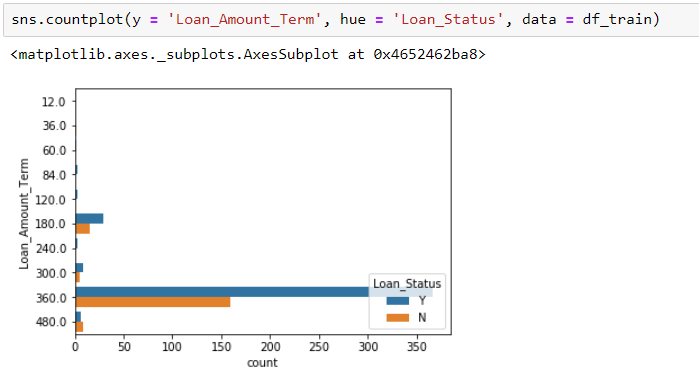
The category of those that take loans is less of self-employed people. That’s those are not self-employed probably salary earners obtain more loan.

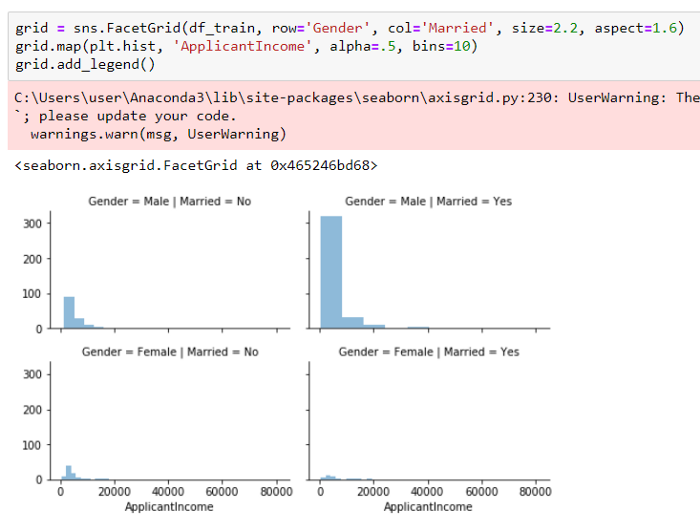


According to the credit history, greater number of people pay back their loans.

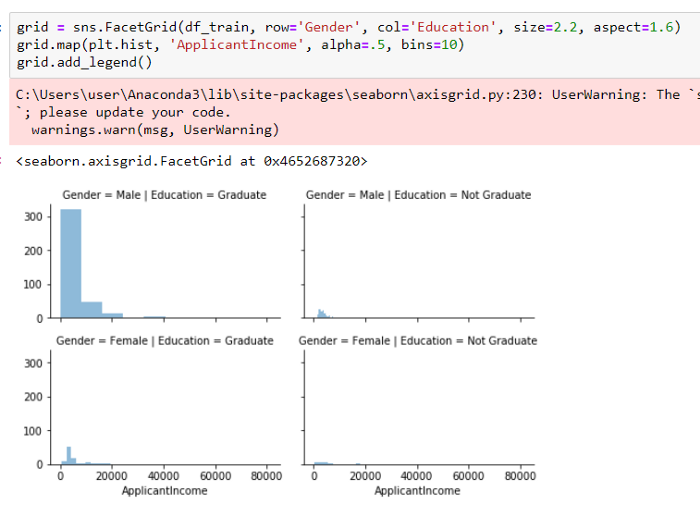


Semi-urban obtain more loan, followed by Urban and then rural. This is logical!

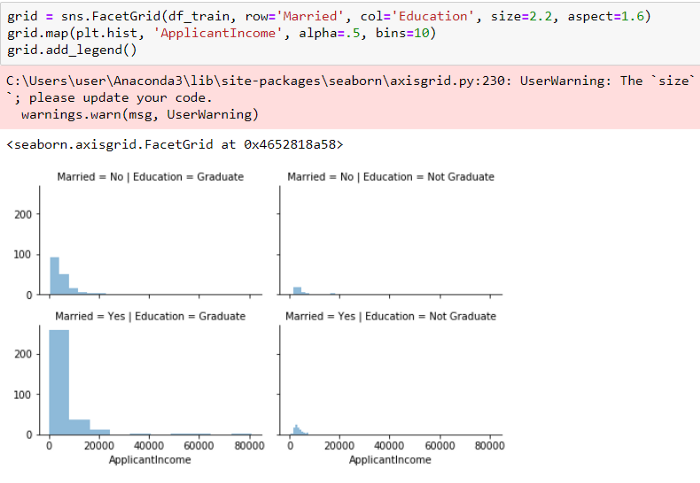
 An extremely high number of them go for a 360 cyclic loan term. That’s pay back within a year



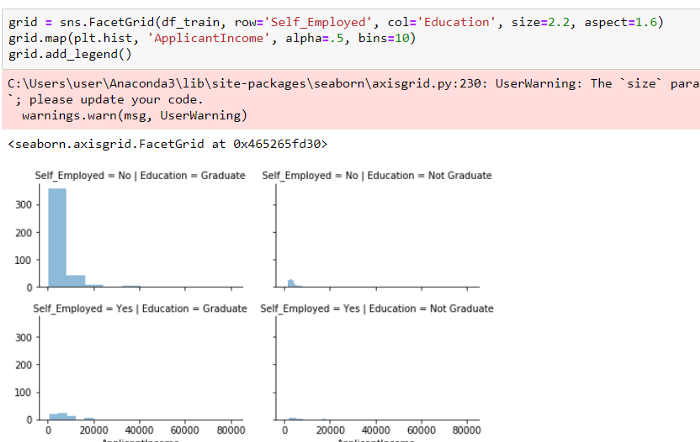
Males generally have the highest income. Explicitly, Males that are married have greater income that unmarried male.



A graduate who is a male has more income.



A graduate and married individual has more income.



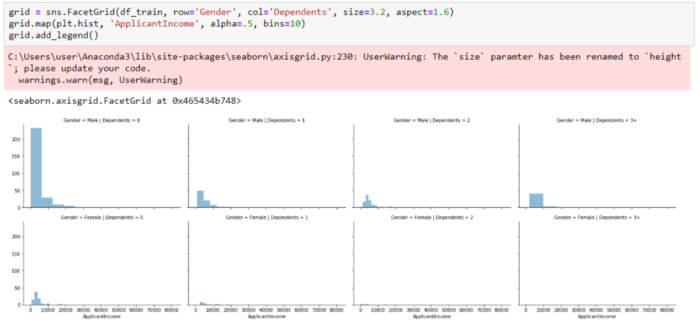
A graduate but not self-employed has more income.



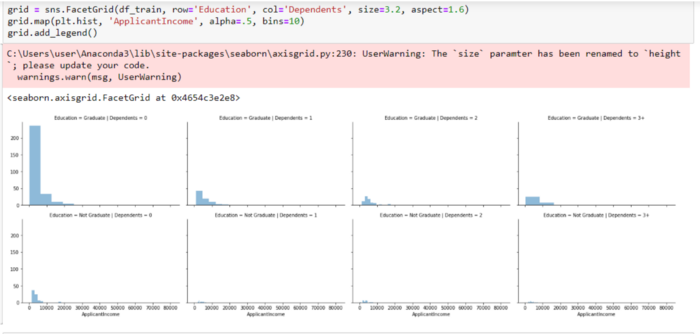
Not married and no one is dependent on such has more income. Also, Married and no one dependent has greater income with a decreasing effect as the dependents increases



No one is dependent and self-employed has more income.



No one is dependent and a male tremendously has more income.



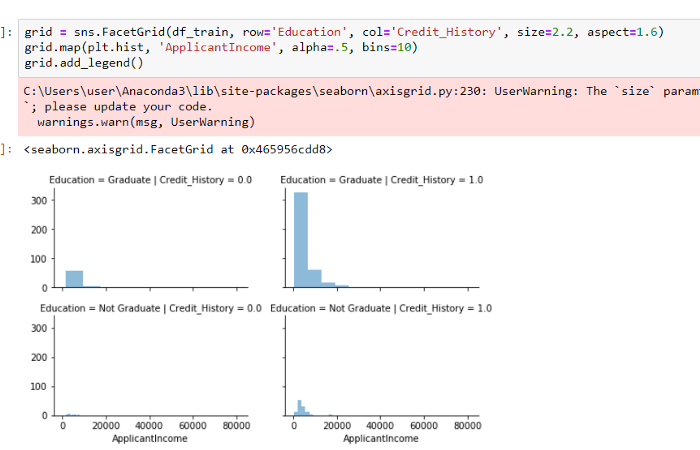
A graduate with no one dependent has more income.



No one is dependent and have property in urban, rural and semi urban has more income.



Married and has a good credit history depicts more income. Also, Not married but has a good credit history follows in the hierarchy.



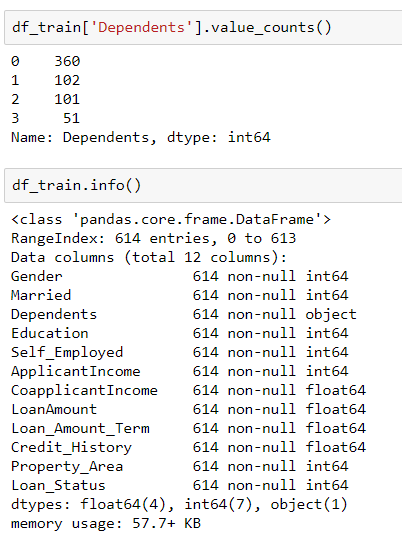
Educated with good credit history depicts a good income. Also, not a graduate and have a good credit history can be traced to having a better income than a fellow with no degree

**Encoding to numeric data and getting ready for training:**

code\_numeric = {‘Male’: 1, ‘Female’: 2,  
‘Yes’: 1, ‘No’: 2,  
‘Graduate’: 1, ‘Not Graduate’: 2,  
‘Urban’: 3, ‘Semiurban’: 2,’Rural’: 1,  
‘Y’: 1, ’N’: 0,  
‘3+’: 3}

df\_train = df\_train.applymap(lambda s: code\_numeric.get(s) if s in code\_numeric else s)  
df\_test = df\_test.applymap(lambda s: code\_numeric.get(s) if s in code\_numeric else s)

#droping the uniques loan id   
df\_train.drop(‘Loan\_ID’, axis = 1, inplace = True)

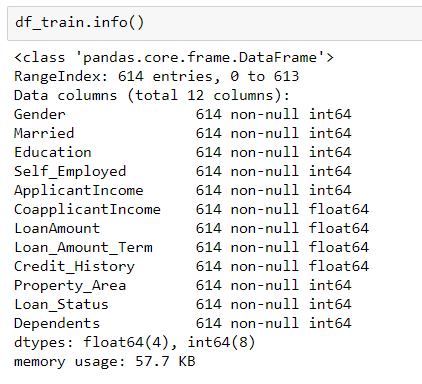


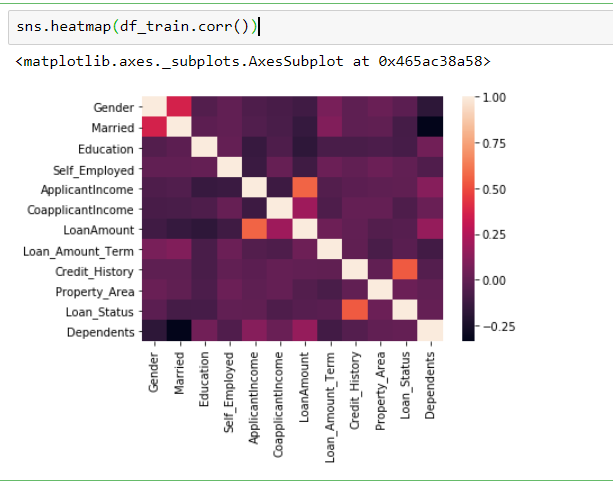
Need to convert ‘Dependents’ feature to numeric using pd to numeric.

Dependents\_ = pd.to\_numeric(df\_train.Dependents)  
Dependents\_\_ = pd.to\_numeric(df\_test.Dependents)

df\_train.drop([‘Dependents’], axis = 1, inplace = True)  
df\_test.drop([‘Dependents’], axis = 1, inplace = True)

df\_train = pd.concat([df\_train, Dependents\_], axis = 1)  
df\_test = pd.concat([df\_test, Dependents\_\_], axis = 1)





**Heat map**: Showing the correlations of features with the target. No correlations are extremely high. The correlations between Loan Amount and Applicant Income can be explained.

**Separating Target from the feature for training**

y = df\_train[‘Loan\_Status’]  
X = df\_train.drop(‘Loan\_Status’, axis = 1)

from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import f1\_score  
from sklearn.model\_selection import GridSearchCV  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.ensemble import RandomForestClassifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=0)

**Using Machine Learning Models:**

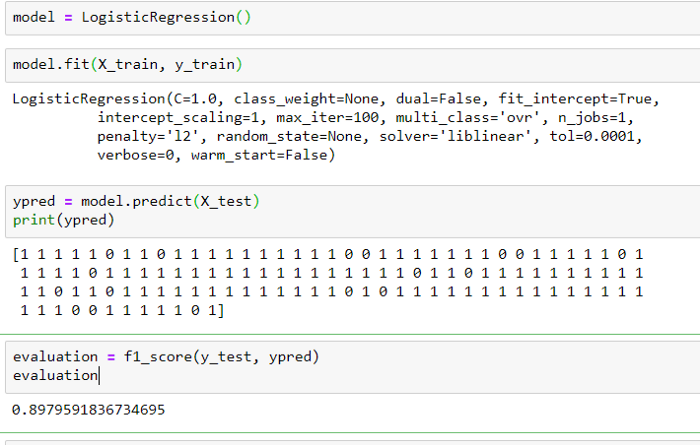
**Using Logistic Regression**

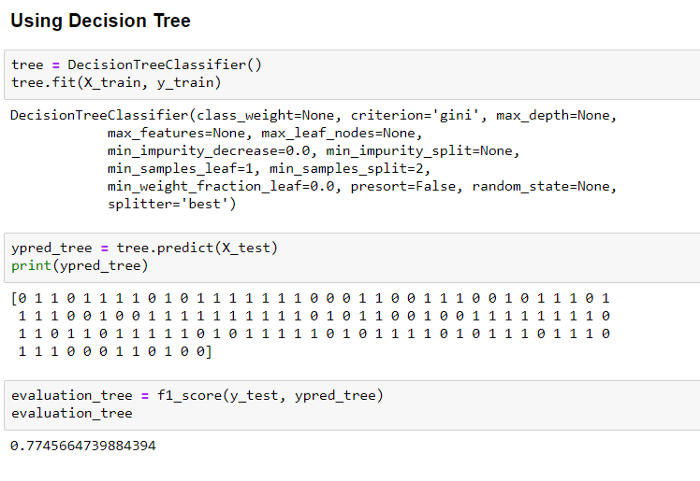
model = LogisticRegression()

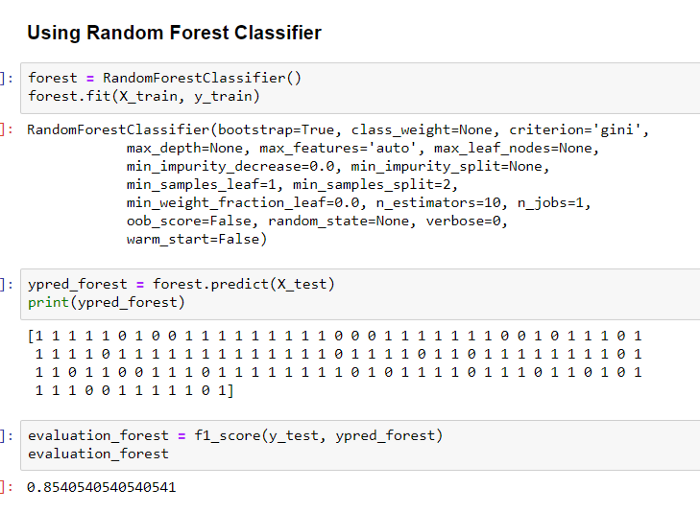
model.fit(X\_train, y\_train)

ypred = model.predict(X\_test)

evaluation = f1\_score(y\_test, ypred)  
evaluation







**Conclusion:**

From the Exploratory Data Analysis, we could generate insight from the data. How each of the features relates to the target. Also, it can be seen from the evaluation of three models that Logistic Regression performed better than others, Random Forest did better than Decision Tree.

**Summary:**

In this blog, I have presented you a modern Data Science Problem with the basics concepts of Machine learning and I hope this blog was helpful and would have motivated you enough to get interested in the topic.