Loan Approval Prediction

Using Knime and python.

This document includes the prediction of loan approval by the company. The major approaches are described via CRISP-DM model.

Group Members:

Jawad Ur Rehman-18016
Tayyaba Zubair-18049
Shazia Kanwal-18026

Table of contents

1. Problem Understanding

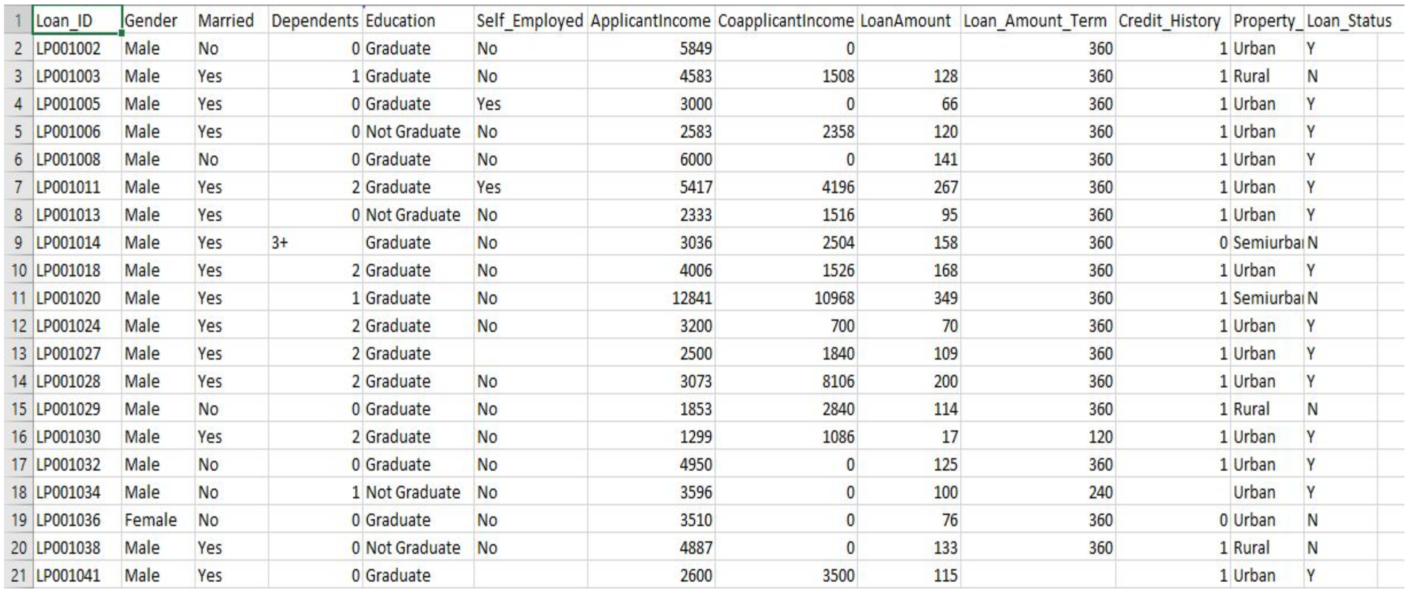
Our problem is Loan Approval Prediction. Dream Housing Finance company deals with all Home Loans. Customers first apply for Home Loan, after that company validates the customer eligibility for loan and gives him approval.
Dream Housing Finance company was facing problems while checking loan applications and approving them manually. So, the company wants to automate the eligibility process for the loans. The customers apply for the Home Loan through online applications.They are using customer details to validate the eligibility criteria for the approval. The online application form requires the customer detail consists of Gender, Marital Status, Number of Dependence, Applicant Education, Applicant Income, Loan Amount, Credit History, Type of Employment. The company wants to get ease on approving all applications.

2. Data Understanding

In this given dataset, there are 615 records and each record has 13 attributes through which we are predicting eligibility of loans. The attributes are: Loan_ID, Married, Dependents, Education, Self_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term, Credit_History, Property_Area, Loan_Status. Some of the fields are continuous. All of the attributes are listed below.

- **Loan_ID**: A unique id for every loan.
- **Married**: This means Marital Status: Single, Married, Divorced, Separated, Widowed.
- **Dependents**: This means Number of Dependents: children or wife or any other family member who are dependent for their livings/basic needs.
- **Education**: This means education or qualification of the applicant like Matriculation, Intermediate, Bachelors, Masters, Doctorate(Phd), Preschool, 7th or 8th Grade.
- **Self_Employed**: This means that the applicant is self employed or not.
- **ApplicantIncome**: Income of the applicant.
- **CoapplicantIncome:** Income from other sources.
- **LoanAmount**: This means how much amount the applicant wants as a loan.
- **Loan_Amount_Term**: For how much time, Loan is taken.
- **Credit_History**: This means if the applicant has or hasn't the ability to repay the loan.
- **Property_Area**: This means applicant's area: Urban, Rural, Semi-Urban.
- **Loan_Status**: This means that applicant's application for loan is approved or not.

Sample:



1	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_	Loan_Status
2	LP001002	Male	No	0	Graduate	No	5849	0		360	1	Urban	Y
3	LP001003	Male	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N
4	LP001005	Male	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y
5	LP001006	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y
6	LP001008	Male	No	0	Graduate	No	6000	0	141	360	1	Urban	Y
7	LP001011	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1	Urban	Y
8	LP001013	Male	Yes	0	Not Graduate	No	2333	1516	95	360	1	Urban	Y
9	LP001014	Male	Yes	3+	Graduate	No	3036	2504	158	360	0	Semiurba	N
10	LP001018	Male	Yes	2	Graduate	No	4006	1526	168	360	1	Urban	Y
11	LP001020	Male	Yes	1	Graduate	No	12841	10968	349	360	1	Semiurba	N
12	LP001024	Male	Yes	2	Graduate	No	3200	700	70	360	1	Urban	Y
13	LP001027	Male	Yes	2	Graduate		2500	1840	109	360	1	Urban	Y
14	LP001028	Male	Yes	2	Graduate	No	3073	8106	200	360	1	Urban	Y
15	LP001029	Male	No	0	Graduate	No	1853	2840	114	360	1	Rural	N
16	LP001030	Male	Yes	2	Graduate	No	1299	1086	17	120	1	Urban	Y
17	LP001032	Male	No	0	Graduate	No	4950	0	125	360	1	Urban	Y
18	LP001034	Male	No	1	Not Graduate	No	3596	0	100	240		Urban	Y
19	LP001036	Female	No	0	Graduate	No	3510	0	76	360	0	Urban	N
20	LP001038	Male	Yes	0	Not Graduate	No	4887	0	133	360	1	Rural	N
21	LP001041	Male	Yes	0	Graduate		2600	3500	115		1	Urban	Y

3. Data Preparation:

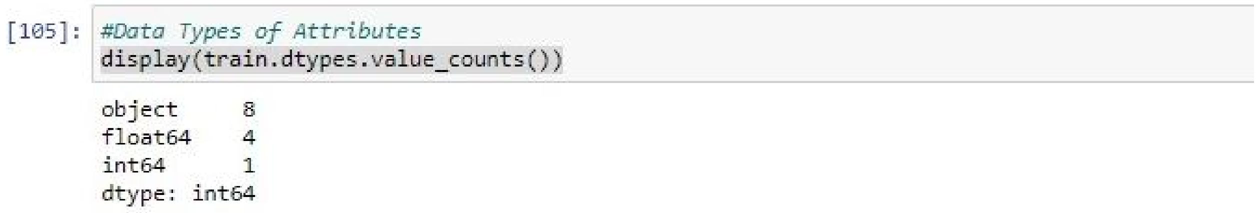
We were in need of maintaining this dataset in a way that we can observe useful patterns. We took four main steps to prepare this data to predict if the applicant is eligible for the loan or not.

1. Table, record and attribute selection, an overview of the data
2. Data Analysis
3. Cleaning Missing Values
4. Transformations on the Data
5. Pre-processing the Test Dataset

We prepared data on both Knime and Python.

3.1 Table, record and attribute selection, an overview of the data

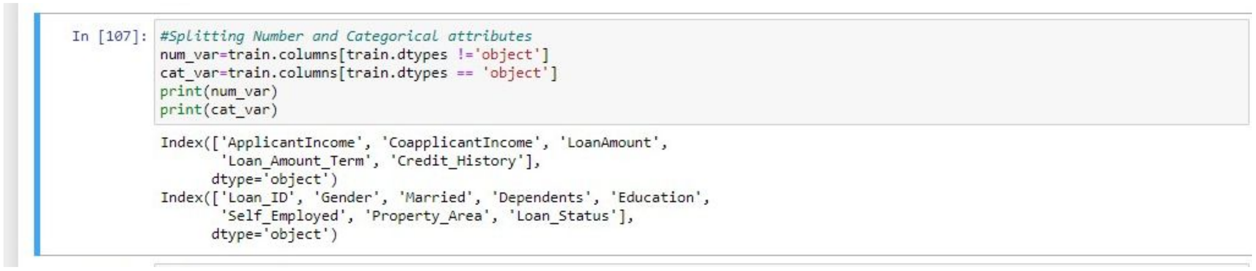
After loading the file, we need to check the structure of File or Data.
To find out the data types of every attribute or how many of the columns are categorical and numerical the python function: “train.dtypes.value_counts()” has been used. The following result obtained.



```
[105]: #Data Types of Attributes
       display(train.dtypes.value_counts())

       object     8
       float64    4
       int64      1
       dtype: int64
```

3.2 Separating Categorical and numerical Value:

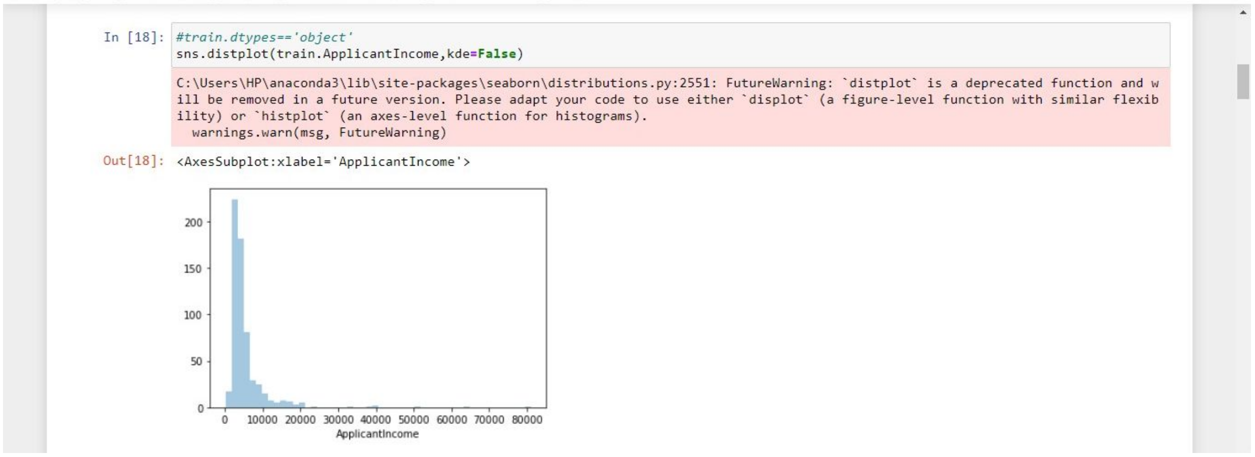
In the next step, we will separate the categorical and numerical column. The reason to separate the columns is handling missing values in python is different for different data types.



```
In [107]: #Splitting Number and Categorical attributes
          num_var=train.columns[train.dtypes !='object']
          cat_var=train.columns[train.dtypes == 'object']
          print(num_var)
          print(cat_var)

          Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
                 'Loan_Amount_Term', 'Credit_History'],
                dtype='object')
          Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
                 'Self_Employed', 'Property_Area', 'Loan_Status'],
                dtype='object')
```

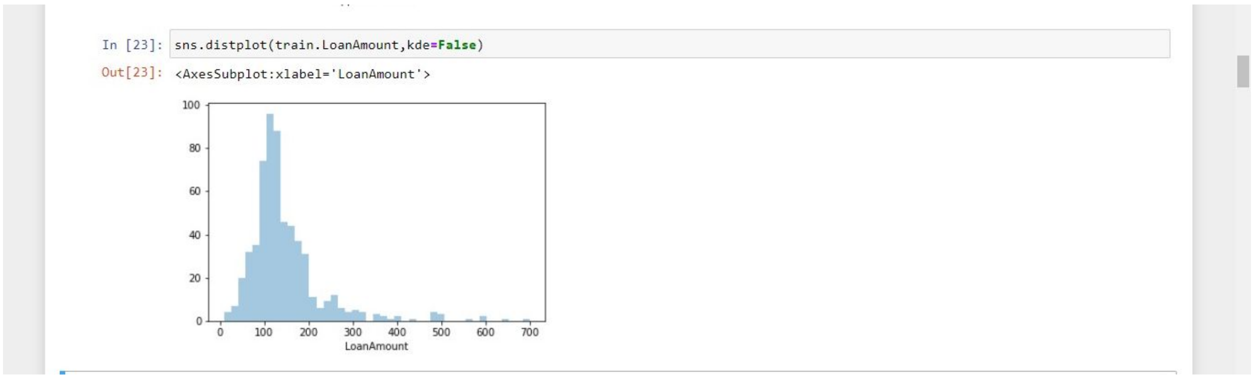
3.3 Data Analysis:

Next step is to check if there are any outliers in the data. It would be better to check the outliers in the data if there are numerical values or especially dependent attributes. We will check for the outliers in Applicant Income , Coapplicant income and Loan Amount.



```
In [18]: #train.dtypes=='object'
         sns.distplot(train.ApplicantIncome,kde=False)

         C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and w
         ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexib
         ility) or `histplot` (an axes-level function for histograms).
           warnings.warn(msg, FutureWarning)

Out[18]: <AxesSubplot:xlabel='ApplicantIncome'>
```

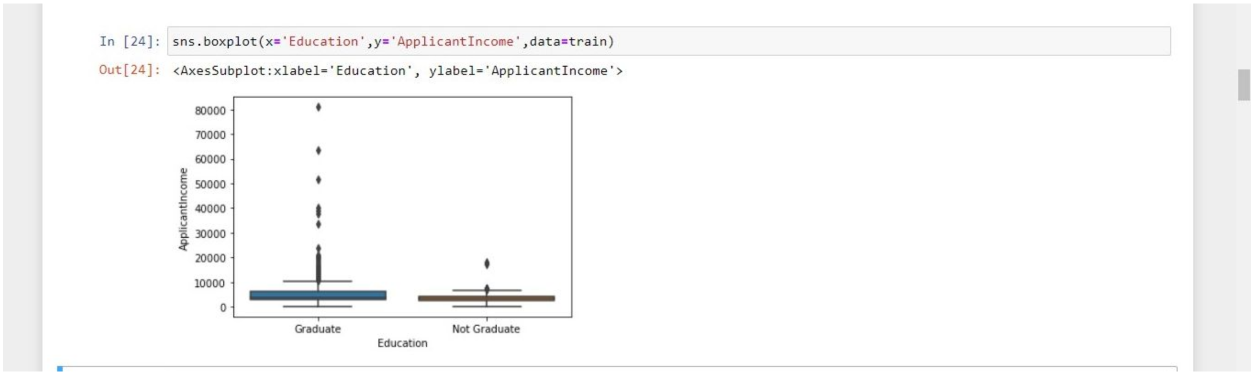
The given distribution is Skewed.



```
In [23]: sns.distplot(train.LoanAmount,kde=False)

Out[23]: <AxesSubplot:xlabel='LoanAmount'>
```

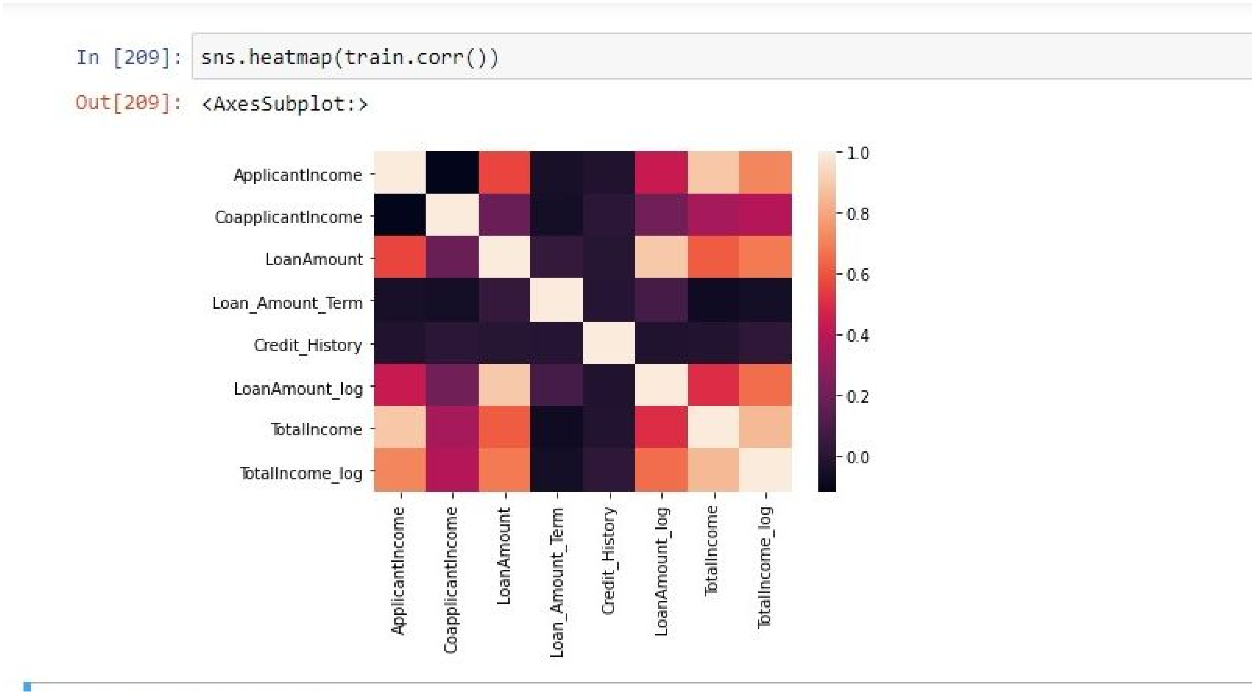
Outliers for Loan amount.

Next, we will find out the higher dependency attribute. We have tried out many dependencies in which Education---->higher income has higher connection: people with huge income are most likely well educated.



```
In [24]: sns.boxplot(x='Education',y='ApplicantIncome',data=train)
Out[24]: <AxesSubplot:xlabel='Education', ylabel='ApplicantIncome'>
```

The correlation can be view as:



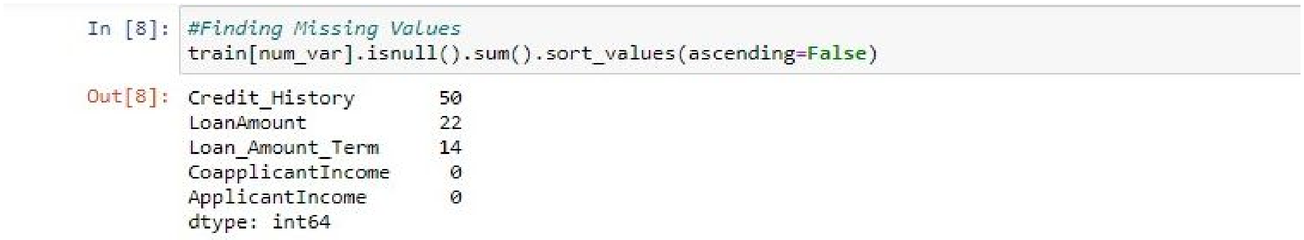
```
In [209]: sns.heatmap(train.corr())
Out[209]: <AxesSubplot:>
```

correlations of features with the target. No correlations are extremely high..

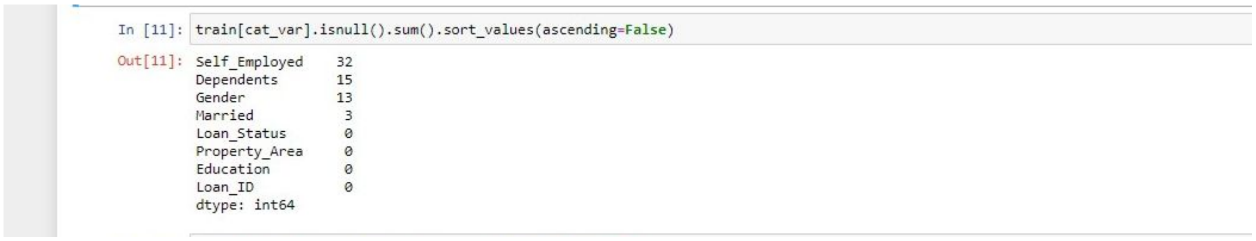
3.4 Find the missing values

The next step is to find the missing values and count of missing values in each attribute.

- **train[num_var]** will consist of all the columns in the data frame which are not object data type.
- **train[cat_var]** will consist of all the columns in the data frame which are object data type.
- **sum()** along with **isnull()** is used to get missing values in each column.
- To sort the data with highest missing value is done by using function: **sort_values()**



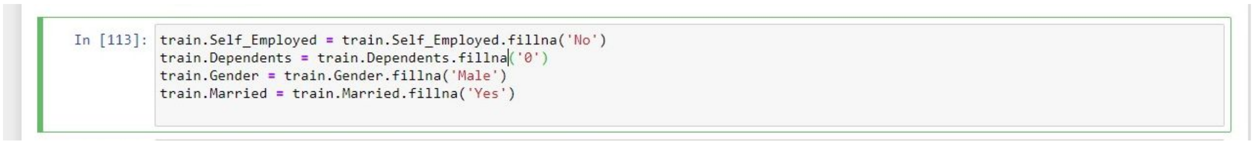
```
In [8]:  #Finding Missing Values
         train[num_var].isnull().sum().sort_values(ascending=False)

Out[8]:  Credit_History        50
         LoanAmount            22
         Loan_Amount_Term      14
         CoapplicantIncome      0
         ApplicantIncome        0
         dtype: int64
```



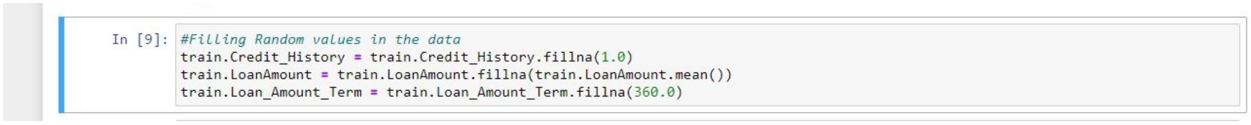
```
In [11]: train[cat_var].isnull().sum().sort_values(ascending=False)

Out[11]: Self_Employed    32
         Dependents       15
         Gender           13
         Married           3
         Loan_Status       0
         Property_Area     0
         Education         0
         Loan_ID           0
         dtype: int64
```

3.4.1 Dealing with Missing Values:

To fill null values we used Fillna() and filled all the attributes logically.

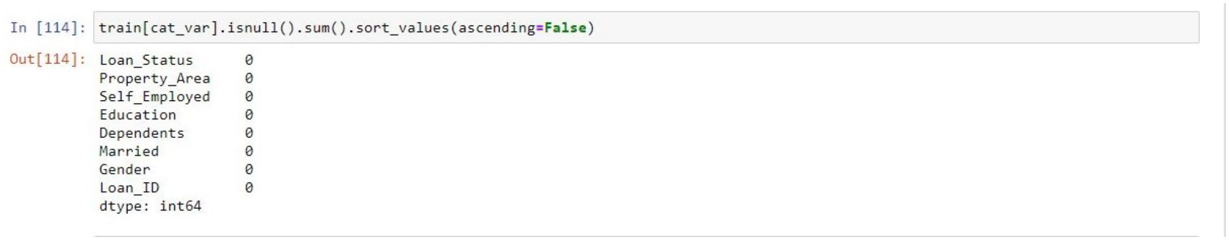


```
In [113]: train.Self_Employed = train.Self_Employed.fillna('No')
          train.Dependents = train.Dependents.fillna('0')
          train.Gender = train.Gender.fillna('Male')
          train.Married = train.Married.fillna('Yes')
```

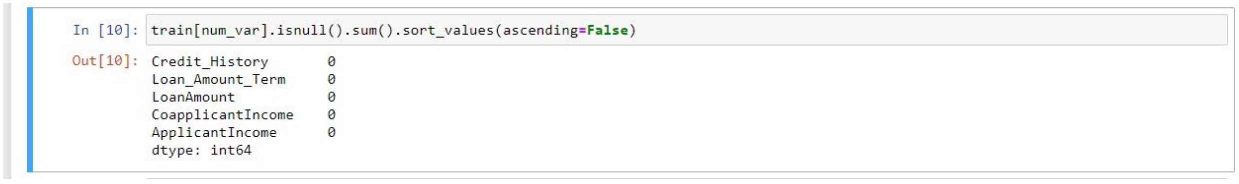


```
In [9]: #Filling Random values in the data
        train.Credit_History = train.Credit_History.fillna(1.0)
        train.LoanAmount = train.LoanAmount.fillna(train.LoanAmount.mean())
        train.Loan_Amount_Term = train.Loan_Amount_Term.fillna(360.0)
```

After filling the missing values check if there are any other attributes left.



```
In [114]: train[cat_var].isnull().sum().sort_values(ascending=False)

Out[114]: Loan_Status      0
          Property_Area    0
          Self_Employed    0
          Education        0
          Dependents       0
          Married          0
          Gender           0
          Loan_ID          0
          dtype: int64
```



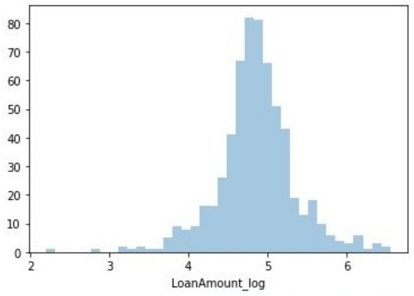
```
In [10]: train[num_var].isnull().sum().sort_values(ascending=False)

Out[10]: Credit_History       0
         Loan_Amount_Term     0
         LoanAmount           0
         CoapplicantIncome    0
         ApplicantIncome      0
         dtype: int64
```

3.5 Removing Outliers:

For removing outliers we have two approaches :Either remove them or we can also log transform them. We have log transformed them to minimize their effect.

```
In [136]: train['LoanAmount_log']=np.log(train['LoanAmount'])
          train['TotalIncome']= train['ApplicantIncome'] +train['CoapplicantIncome']
          train['TotalIncome_log']=np.log(train['TotalIncome'])
```

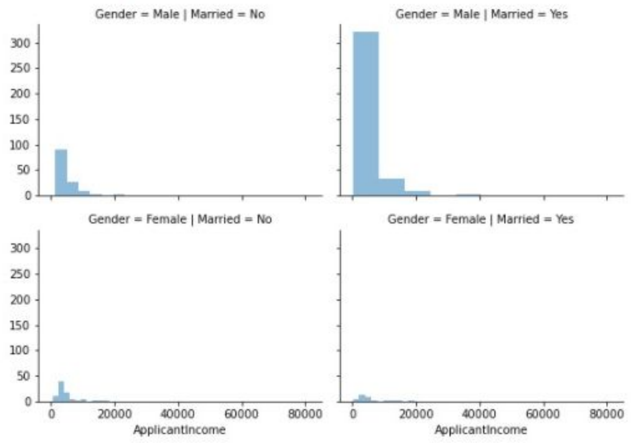
```
In [137]: sns.distplot(train.LoanAmount_log,kde=False)

C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and w
ill be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexib
ility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)

Out[137]: <AxesSubplot:xlabel='LoanAmount_log'>
```



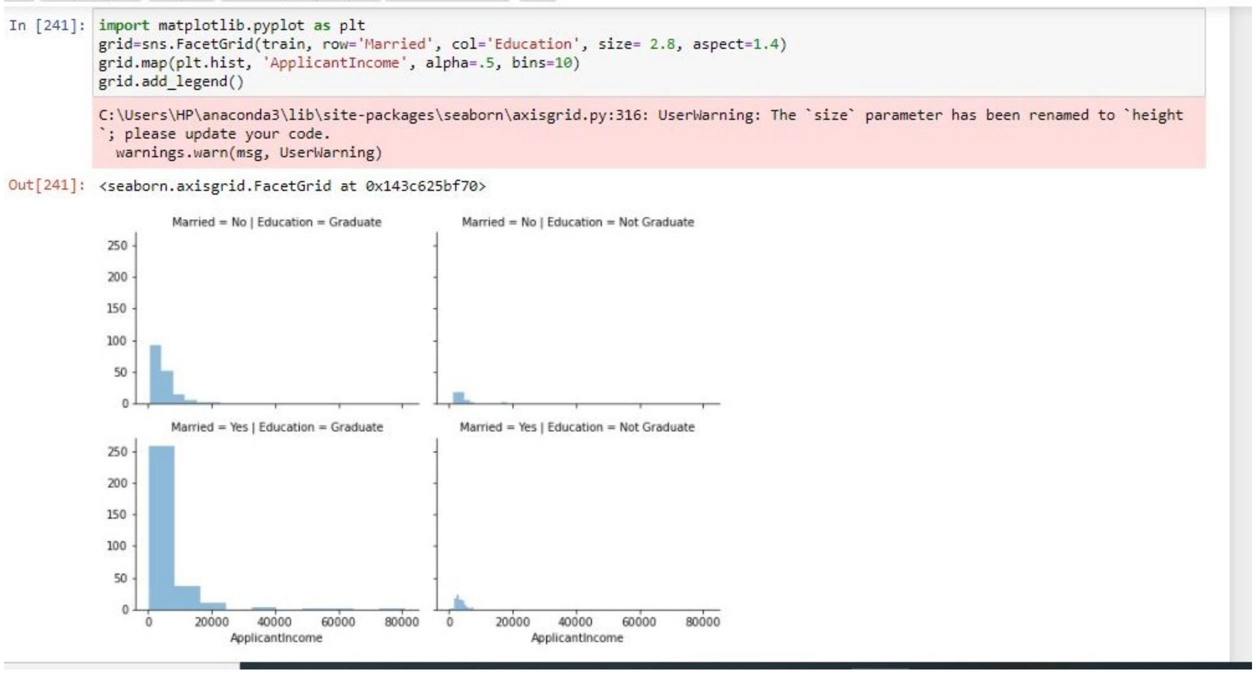
3.6 Data Visualization:

To get the idea of dependency and relation between the attributes, we tried to visualize the data. We will brief it here with some snapshots.

```
In [242]: import matplotlib.pyplot as plt
          grid=sns.FacetGrid(train, row='Gender', col='Married', size= 2.8, aspect=1.4)
          grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)
          grid.add_legend()

C:\Users\HP\anaconda3\lib\site-packages\seaborn\axisgrid.py:316: UserWarning: The `size` parameter has been renamed to `height
`; please update your code.
  warnings.warn(msg, UserWarning)

Out[242]: <seaborn.axisgrid.FacetGrid at 0x143c7f01370>
```



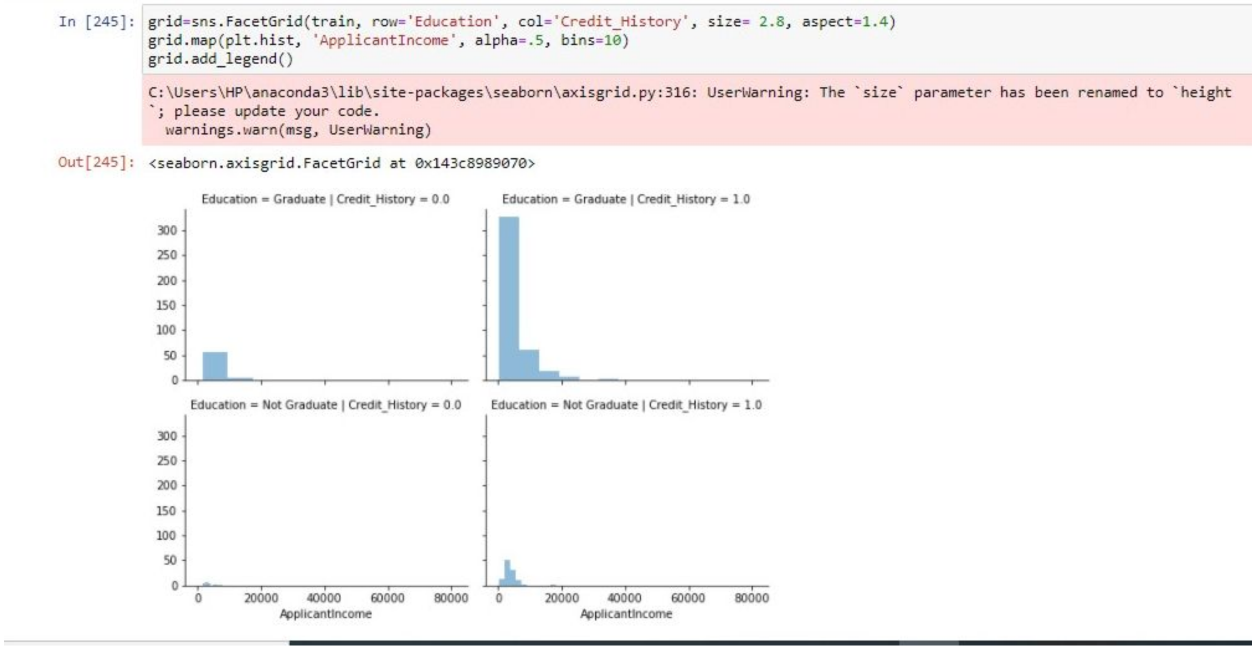
Males that are married have greater income that unmarried male



```
In [241]: import matplotlib.pyplot as plt
          grid=sns.FacetGrid(train, row='Married', col='Education', size= 2.8, aspect=1.4)
          grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)
          grid.add_legend()

          C:\Users\HP\anaconda3\lib\site-packages\seaborn\axisgrid.py:316: UserWarning: The `size` parameter has been renamed to `height
          `; please update your code.
            warnings.warn(msg, UserWarning)

Out[241]: <seaborn.axisgrid.FacetGrid at 0x143c625bf70>
```

Married = No | Education = Graduate
Married = No | Education = Not Graduate
Married = Yes | Education = Graduate
Married = Yes | Education = Not Graduate
ApplicantIncome
ApplicantIncome

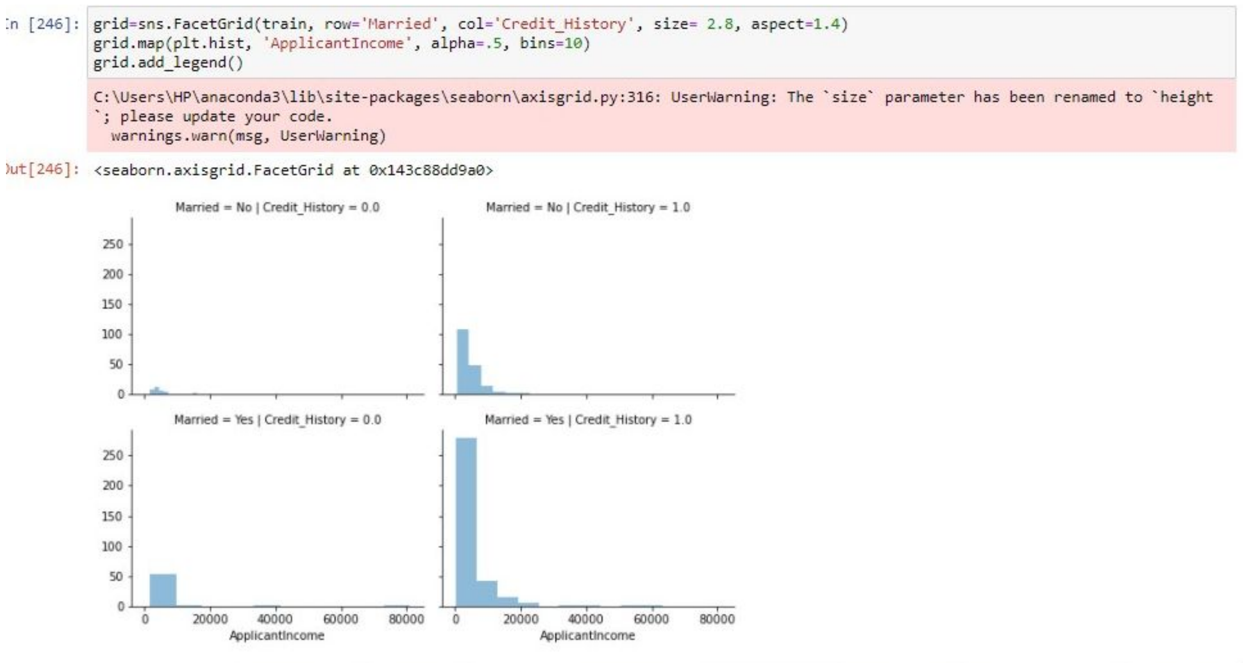
Graduate and married individual has more income



```
In [245]: grid=sns.FacetGrid(train, row='Education', col='Credit_History', size= 2.8, aspect=1.4)
          grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)
          grid.add_legend()

          C:\Users\HP\anaconda3\lib\site-packages\seaborn\axisgrid.py:316: UserWarning: The `size` parameter has been renamed to `height
          `; please update your code.
            warnings.warn(msg, UserWarning)

Out[245]: <seaborn.axisgrid.FacetGrid at 0x143c8989070>
```

Education = Graduate | Credit_History = 0.0
Education = Graduate | Credit_History = 1.0
Education = Not Graduate | Credit_History = 0.0
Education = Not Graduate | Credit_History = 1.0
ApplicantIncome
ApplicantIncome

Non-graduate and good credit history can have better income than a non-degree fellow

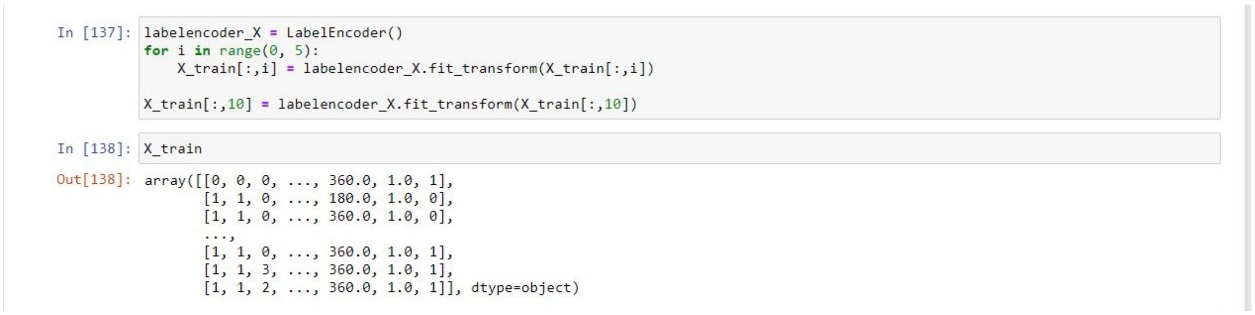


```
In [246]: grid=sns.FacetGrid(train, row='Married', col='Credit_History', size= 2.8, aspect=1.4)
          grid.map(plt.hist, 'ApplicantIncome', alpha=.5, bins=10)
          grid.add_legend()

          C:\Users\HP\anaconda3\lib\site-packages\seaborn\axisgrid.py:316: UserWarning: The `size` parameter has been renamed to `height
          `; please update your code.
            warnings.warn(msg, UserWarning)

Out[246]: <seaborn.axisgrid.FacetGrid at 0x143c88dd9a0>
```

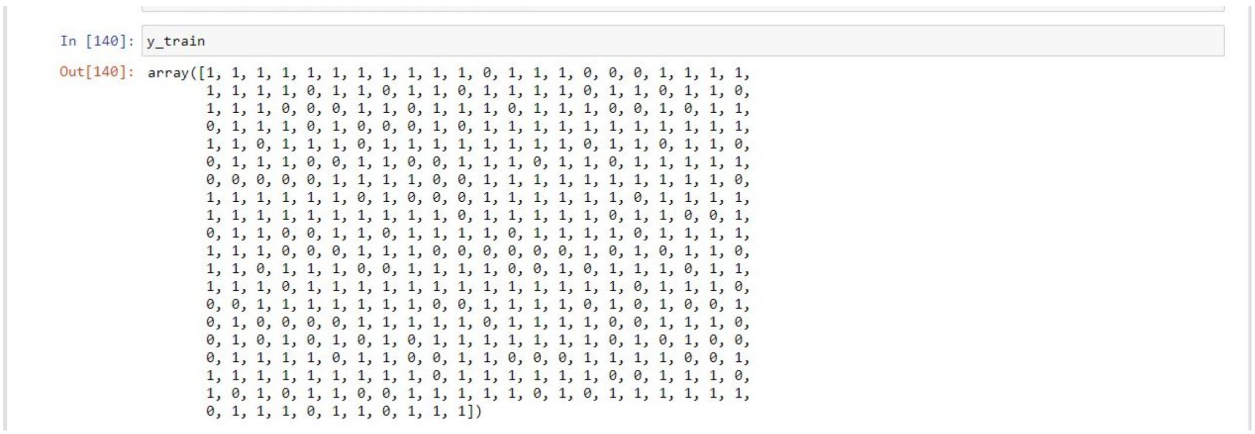
Married and high credit history--->income.

3.7 Transformation on the Data

The next step is to split data into train and test data, the ratio was kept 70:30. **Labelencoder(), OneHotEncoder()** these encoders are part of Scikit Learner and are used to normalize the data and convert the text, categorical data into numerical data.



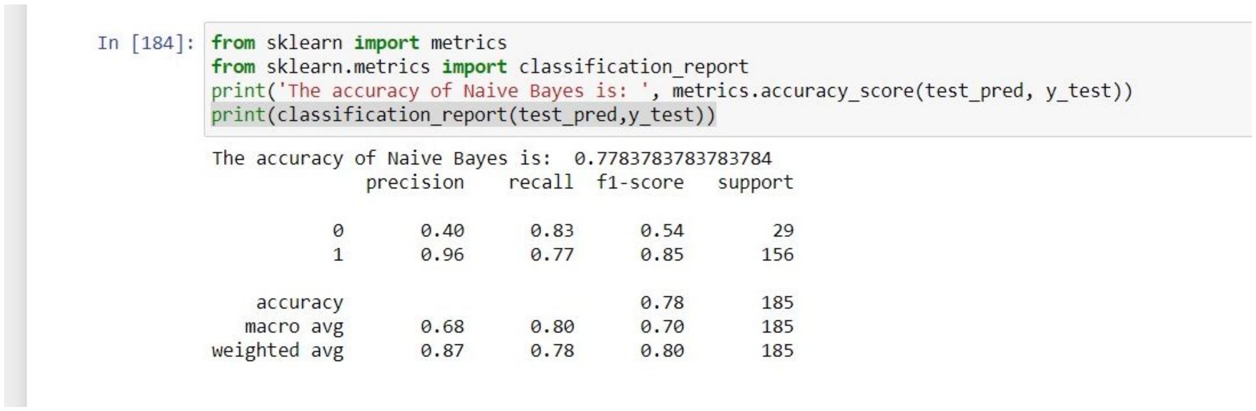
```
In [137]: labelencoder_X = LabelEncoder()
          for i in range(0, 5):
              X_train[:,i] = labelencoder_X.fit_transform(X_train[:,i])

          X_train[:,10] = labelencoder_X.fit_transform(X_train[:,10])

In [138]: X_train

Out[138]: array([[0, 0, 0, ..., 360.0, 1.0, 1],
                 [1, 1, 0, ..., 180.0, 1.0, 0],
                 [1, 1, 0, ..., 360.0, 1.0, 0],
                 ...,
                 [1, 1, 0, ..., 360.0, 1.0, 1],
                 [1, 1, 3, ..., 360.0, 1.0, 1],
                 [1, 1, 2, ..., 360.0, 1.0, 1]], dtype=object)
```



```
In [140]: y_train

Out[140]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1,
                 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
                 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1,
                 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
                 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
                 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
                 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
                 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
                 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1,
                 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
                 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1,
                 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0,
                 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1,
                 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1])
```

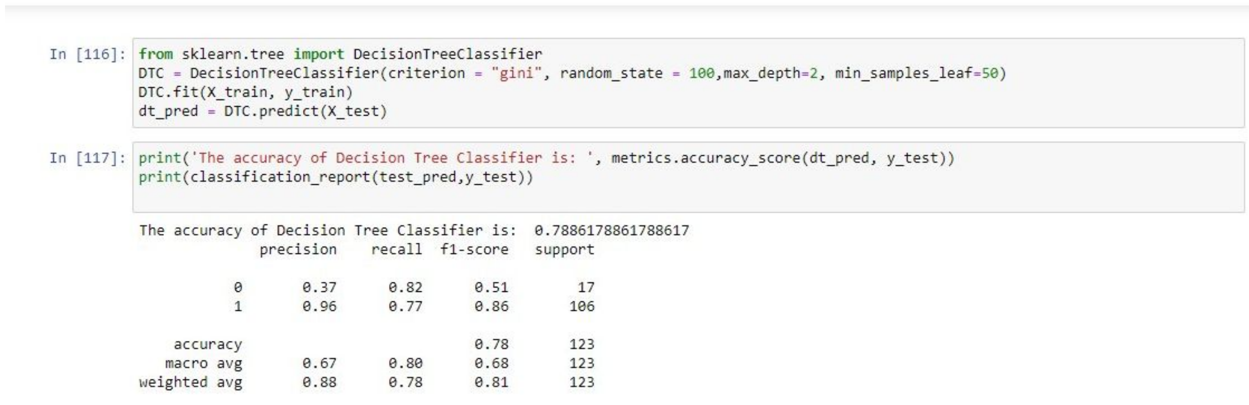
4. Modeling Building + Evaluation:

1. **NaiveBayes**
   Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on independence assumptions between the features.It’s a supervised learning and known for its simplicity and robustness.By using this classification model, we obtained a Accuracy measure by using scorer in KINME and metrics.accuracy.score() in python.



```
In [184]: from sklearn import metrics
          from sklearn.metrics import classification_report
          print('The accuracy of Naive Bayes is: ', metrics.accuracy_score(test_pred, y_test))
          print(classification_report(test_pred,y_test))

          The accuracy of Naive Bayes is:  0.7783783783783784
                        precision    recall  f1-score   support

                     0       0.40      0.83      0.54        29
                     1       0.96      0.77      0.85       156

              accuracy                           0.78       185
             macro avg       0.68      0.80      0.70       185
          weighted avg       0.87      0.78      0.80       185
```

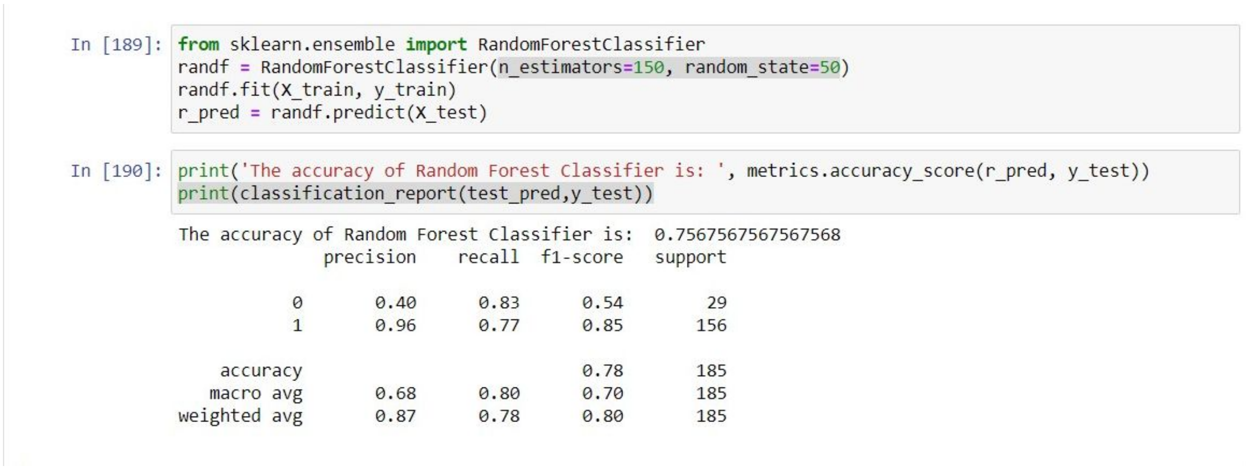
2. **Decision Tree:**
   Decision tree is a highly used classification model in the industries due to it’s high accuracy and ability to formulate a statistical model. Decision tree form a treelike structure in which each node represents a feature and each shows a decision rule. The leaf node represents an outcome/result. The root node is selected on the basis of Gini Index/Gain ratio. The decision tree has shown an efficient accuracy of the model by evaluating it through the Gini Index.



```
In [116]: from sklearn.tree import DecisionTreeClassifier
          DTC = DecisionTreeClassifier(criterion = "gini", random_state = 100,max_depth=2, min_samples_leaf=50)
          DTC.fit(X_train, y_train)
          dt_pred = DTC.predict(X_test)

In [117]: print('The accuracy of Decision Tree Classifier is: ', metrics.accuracy_score(dt_pred, y_test))
          print(classification_report(test_pred,y_test))

          The accuracy of Decision Tree Classifier is:  0.7886178861788617
                        precision    recall  f1-score   support

                     0       0.37      0.82      0.51        17
                     1       0.96      0.77      0.86       106

              accuracy                           0.78       123
             macro avg       0.67      0.80      0.68       123
          weighted avg       0.88      0.78      0.81       123
```

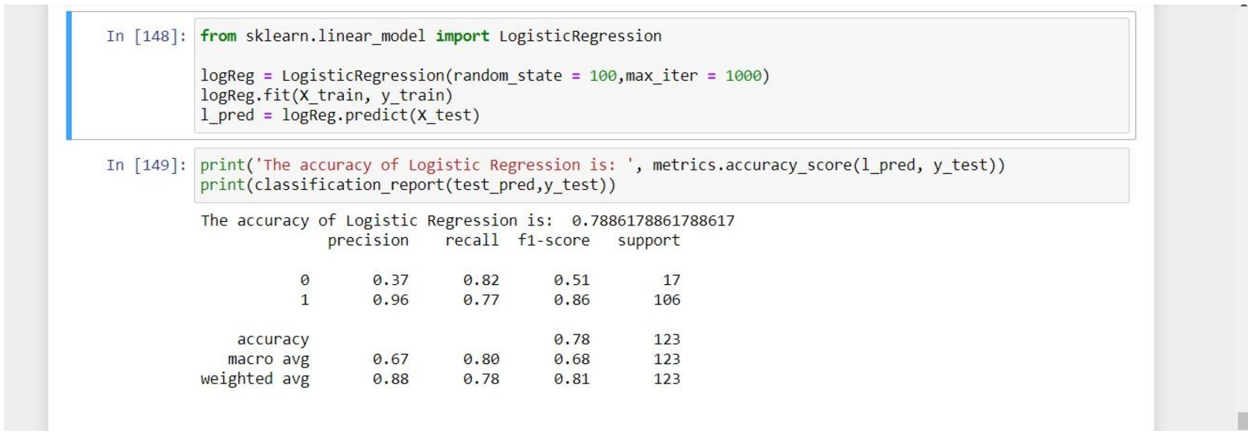
]

3. **Random Forest:**
   Random forest is an ensemble learning method and belongs to supervised learning algorithms.
   It constructs a multitude of decision trees at training time and outputs the class that is the mode of the classes or means prediction of the individual trees.We tried many executions in python and the following result obtained.



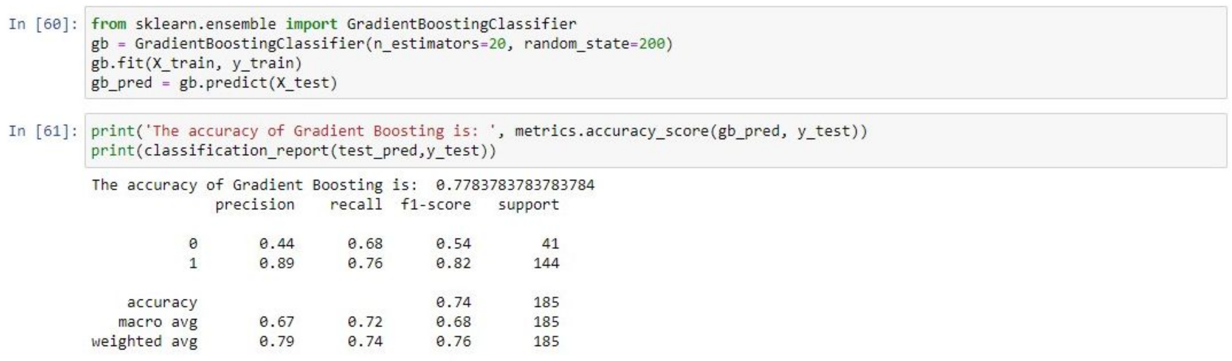
```
In [189]: from sklearn.ensemble import RandomForestClassifier
          randf = RandomForestClassifier(n_estimators=150, random_state=50)
          randf.fit(X_train, y_train)
          r_pred = randf.predict(X_test)

In [190]: print('The accuracy of Random Forest Classifier is: ', metrics.accuracy_score(r_pred, y_test))
          print(classification_report(test_pred,y_test))

          The accuracy of Random Forest Classifier is:  0.7567567567567568
                        precision    recall  f1-score   support

                     0       0.40      0.83      0.54        29
                     1       0.96      0.77      0.85       156

              accuracy                           0.78       185
             macro avg       0.68      0.80      0.70       185
          weighted avg       0.87      0.78      0.80       185
```

4. **Logistic Regression:**
    Logistic regression is used for solving the predictive classification problems. It is used when the dependent variable(target) is binary in nature and It gives some probabilistic values which lie between 0 and 1.
   While using Logistic regression for the loan prediction it predicted a good accuracy score.The following result obtained.



```
In [148]: from sklearn.linear_model import LogisticRegression

          logReg = LogisticRegression(random_state = 100,max_iter = 1000)
          logReg.fit(X_train, y_train)
          l_pred = logReg.predict(X_test)

In [149]: print('The accuracy of Logistic Regression is: ', metrics.accuracy_score(l_pred, y_test))
          print(classification_report(test_pred,y_test))

          The accuracy of Logistic Regression is:  0.7886178861788617
                        precision    recall  f1-score   support

                     0       0.37      0.82      0.51        17
                     1       0.96      0.77      0.86       106

              accuracy                           0.78       123
             macro avg       0.67      0.80      0.68       123
          weighted avg       0.88      0.78      0.81       123
```

5. **Gradient Boosting Classifier:**
6. When we take tree depth = 20 and number of models = 400, we get 74.90% accuracy.



```
In [60]: from sklearn.ensemble import GradientBoostingClassifier
         gb = GradientBoostingClassifier(n_estimators=20, random_state=200)
         gb.fit(X_train, y_train)
         gb_pred = gb.predict(X_test)
```

```
In [61]: print('The accuracy of Gradient Boosting is: ', metrics.accuracy_score(gb_pred, y_test))
         print(classification_report(test_pred,y_test))
```

```
The accuracy of Gradient Boosting is:  0.7783783783783784
              precision    recall  f1-score   support

           0       0.44      0.68      0.54        41
           1       0.89      0.76      0.82       144

    accuracy                           0.74       185
   macro avg       0.67      0.72      0.68       185
weighted avg       0.79      0.74      0.76       185
```

5. Findings:

This project gave us a massive increment in our knowledge as we learned:

- How to manage data for a business problem.
- How to understand data to do the attribute selection.
- How to prepare the data for processing.
- How to manage missing data,
- How to prepare a good-working model for data.
- How to extract results from the models.

We got to know that the most important part is Data Pre-Processing, that is how we manage data to process it. Problems that occurred in Data Pre-processing were:

- There were many missing values in each column of data.
- Filling those missing values with a large set data is a bit of a challenging task.

We managed missing values by filling them logically since it was not a big data set. We ignored the values that were not predictable.

This problem was prediction based which comes under the umbrella of Classification problems. We implemented all methods on this data that were possible in the classification. We choose these models to implement them in a manner that they give us a good outcome.

- Naive Bayes
- Decision Tree

- Random Forest
- Logistic Regression
- Gradient Boosted Trees

By using these models,, we are able to conclude as:

Models	Description	Accuracy	Description	Accuracy
Naive Bayes	Splitting ratio:70:30, random states =100	74%	Splitting ratio:80:20, random states =100	78%
Decision Tree	Min depth>5, no pruning with gini index, leaf depth>70	69%	Min depth<5, no pruning with gini index, leaf depth< 80	78.8%
Random Forest	n_estimators<50 , random_state=50	<69%	n_estimators=150, random_state=50	75.2%
Logistic Regression	random_state = 0,max_iter = 500	78%	random_state = 100,max_iter = 1000	78.2%
Gradient Boosted Trees	N-estimator >20, random states>100	72%	n-estimator=20, random states=200	77%

It shows that the models trained on the concepts of logistic regression performed well in our Loan prediction scenario. We can claim that the logistic regression has is proved as a good learner for loan prediction scenarios with the highest accuracy of 78%in both worst and max scenario..

6. Advices for Organization:

The advices for the organization in terms of Data Collection are:

- The attributes that are important in terms of making a decision, make them mandatory in Data Collection.
- Make a note to not to facilitate the ones who do not enter their whole information.
- Once data is collected, tManage it efficiently. Always have a backup.

Once the data has been collected efficiently the next step is to train the model. In predictive analysis it’s better to use supervised machine learning algorithms such as decision tree, random forest and logistic regression. Some of the models need to update once in the years because of the flow of data and requirements. Else A good approach is based on the collection of data and maintenance of the data.