# HOME LOAN PREDICTIONS

**Executive summary**

1. Housing Finance company which provides home loans for the houses that are present across all urban, semi urban and rural areas for their valued customers.
2. The company validates the eligibility of the loan after customer applies for the loan. However, it consumes a lot of time for the manual validation of the eligibility process.
3. Hence, the company wants to automate the loan eligibility process based on the customer information and identify the factors/customer segments who are eligible for taking the loan.
4. As banks would give loans to only those customers who are eligible so that they can be assured of getting the money back.
5. Hence, the more accurate we are in predicting the eligible customers, the more beneficial it would be for the company.

**Detailed Overview of the Mortgage Approval & Funding Process:**

1. Pre-Assessment Discussion (15 minute conversation)
2. Pre-Approval Kick-Off (takes us no more than 1 day)
3. Opening a File (takes us no more than 1 day)
4. Lender Underwriting (takes 1 - 7 days from our formal submission)

- Credit history - Your lender will want to make sure when you've borrowed money, you've paid it back
- Capital - Ensuring you've accumulated assets
- Collateral - When it comes to a mortgage, you're putting your house up as collateral
- Capacity - In short, capacity is debt servicing. For instance, your housing cost shouldn't exceed 30 per cent to 32 percent of your gross income and all of your debts shouldn't exceed 40 per cent to 42 percent of your gross income
- Character - It's an evaluation of all four previous C's as well as subjective and objective things such as how long have you been in your job, what type of job you have and how long you have lived in your current residence.

**Overall Time Consumed for single Loan Application:**

1. Conditional Commitment Processing (takes 2 - 4 days from lender approval)
2. Pre-Closing (takes 7 - 10 days from 'file complete')
3. Closing (typically by noon on the funding/possession date)

**Problem Statement**

1. Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling an online application form.
2. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others.
3. To automate this process, the company has given us a problem to identify the customer segments that are eligible for loan amount so that they can specifically target these customers. Here, we have been provided a partial data set for further analysis.

**Structured Analysis Planning:** The SMART (Specific Measurable Assignable Relevant Time-based) objective was employed to analyze the data and understand the problem statement. The next step is to identify our independent variables and our dependent variable, the below map illustrates the process which was conducted to structure plan the project.

To this approach, we employed Exploratory data analysis which include univariate analysis and bivariate analysis.

**Assumptions for EDA:**

1. The customers whose salary is more can have a greater chance of loan approval.
2. The applicants who are graduates, have a better chance of loan approval than non-graduate applicants.
3. Married applicants would have upper hand than single or no-relationship applicants for loan approval.
4. The applicant who has less dependents has a high probability for loan approval.
5. The lesser the loan amount, the higher chances of loan getting approved.

**Type of Problem:**

The above problem is a clear classification problem as we need to classify whether the Loan_Status is yes or no. So this can be solved by any of the classification techniques like

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

**Description about the Data Columns:**

There are 2 data sets that are given. One is training data and one is testing data. It's very useful to know about the data columns before getting into the actual problem for avoiding confusion at a later state. Now let us understand the data columns (that has been already given by the company itself ) first so that we will get a glance.

| Variable | Description |
|---|---|
| **Loan_ID** | Unique Loan ID |
| **Gender** | Male/ Female |
| **Married** | Applicant married (Y/N) |
| **Dependents** | Number of dependents |
| **Education** | Applicant Education (Graduate/ Under Graduate) |
| **Self_Employed** | Self employed (Y/N) |
| **ApplicantIncome** | Applicant income |
| **CoapplicantIncome** | Coapplicant income |
| **LoanAmount** | Loan amount in thousands |
| **Loan_Amount_Term** | Term of loan in months |
| **Credit_History** | credit history meets guidelines |
| **Property_Area** | Urban/ Semi Urban/ Rural |
| **Loan_Status** | Loan approved (Y/N) |

There are altogether 13 columns in our data set. Of them Loan_Status is the response variable and rest all are the variables /factors that decide the approval of the loan or not.

Now let us look in to the each variable and can make some assumptions.(It's just assumptions right, there is no harm in just assuming few statements)

- Loan ID -> As the name suggests each person should have a unique loan ID.
- Gender -> In general it is male or female. No offence for not including the third gender.
- Married -> Applicant who is married is represented by Y and not married is represented as N. The information regarding whether the applicant who is married is divorced or not has not been provided. So we don't need to worry regarding all these.
- Dependents -> the number of people dependent on the applicant who has taken loan has been provided.
- Education -> It is either non -graduate or graduate. The assumption I can make is " The probability of clearing the loan amount would be higher if the applicant is a graduate".
- Self_Employed -> As the name suggests Self Employed means , he/she is employed for himself/herself only. So freelancers or having their own business might come in this category. An applicant who is self employed is represented by Y and the one who is not is represented by N.
- Applicant Income -> Applicant Income suggests the income by Applicant.So the general assumption that i can make would be "The one who earns more have a high probability of clearing loan amount and would be highly eligible for loan **"**
- Co Applicant income -> this represents the income of co-applicant. I can also assume that " If co applicant income is higher , the probability of being eligible would be higher "
- Loan Amount -> This amount represents the loan amount in thousands. One assumption I can make is that " If Loan amount is higher , the probability of repaying would be lesser and vice versa"
- Loan_Amount_Term -> This represents the number of months required to repay the loan.
- Credit_History -> When I googled it , I got this information. A **credit history** is a record of a borrower's responsible repayment of debts. It suggests → 1 denotes that the credit history is good and 0 otherwise.

- Property_Area -> The area where they belong to is my general assumption as nothing more is told. Here there can be three types. Urban or Semi Urban or Rural
- Loan_Status -> If the applicant is eligible for loan it’s yes represented by Y else it’s no represented by N.

# Tidying the data

Now that we've identified several errors in the data set, we need to fix them before we continue with our analysis. Let's review the issues:

There are missing values in some variables. Based on the importance of the variables, we will decide on the method to use.

Looking at the distributions of the data, we noticed that ApplicantIncome and LoanAmount have outliers.

Fixing outliers can be tricky. It's hard to tell if they were caused by measurement error, errors while recording, or if the outliers are real anomalies. If we decide to remove records, we have to document the reason behind this decision.

In this data set, we will assume that missing values are systematic because the missing data are coming in certain variables in a random manner. Also, we note that missing values are on both numerical and categorical data, therefore, we will be creating different functions to handle these scenarios. These functions help in imputing missing values with plausible data values. These values are inferred from a distribution that is designed for each missing data point.

**Type of Variables:**

1. Input variable (Predictor): Gender, Married, Education, Self_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term, Credit_History
2. Output variable (Target): Loan_Status

**Variable category:**

1. Categorical variables: Loan_ID, Gender, Married, Dependents, Education, Self_Employed, Property_Area, Loan_Status
2. Continuous variables: ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term

**Data Pre-processing Steps:**

1. Handling missing values
2. Creation of required variables
3. Replacing the data-values

**Handling Missing Values:**

1. Defining function to fetch “most_common” value of the feature
2. Defining function to “replace_missing” values with most_common/mean values
3. Calling “replace_missing” function for all features with null values by passing the feature name to the function.

```
In [6]: def most_common(col):
            most_cmn=pd.get_dummies(col).sum().sort_values(ascending = False).index[0]
            return most_cmn
```

```
In [7]: def replace_missing(col):
            most_cmn = most_common(col)
            for i in range(len(col)):
                if pd.isnull(col[i]):
                    col[i] = most_cmn
                    inplace=True
```

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               601 non-null object
Married              611 non-null object
Dependents           599 non-null object
Education            614 non-null object
Self_Employed        582 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           592 non-null float64
Loan_Amount_Term     600 non-null float64
Credit_History       564 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

**Before Handling Null Values**

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
Loan_ID              614 non-null object
Gender               614 non-null object
Married              614 non-null object
Dependents           614 non-null object
Education            614 non-null object
Self_Employed        614 non-null object
ApplicantIncome      614 non-null int64
CoapplicantIncome    614 non-null float64
LoanAmount           614 non-null float64
Loan_Amount_Term     614 non-null float64
Credit_History       614 non-null float64
Property_Area        614 non-null object
Loan_Status          614 non-null object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.4+ KB
```

**After Handling Null Values**

**Creation of required variables:**

1. Creating “Total_Income” variable by adding “ApplicantIncome” to “CoapplicantIncome”
2. Calculating the “EMI” variable

   By considering the above link, I have found that on an average it would be around 8.5% to 9.5%. Hence for the safe-side I am assuming that 9% is the interest rate.

   - A = P * R * (1+R)^N
   - B = (1+R)^(N-1)
   - EMI = A/B

```
data["Total_Income"] = data["ApplicantIncome"]+data["CoapplicantIncome"]
```

```
By considering the above link, I have found that on an average it would be around 8.5% to 9.5%. Hence for safe-side I am assuming that 9% is the interest rate.

A = P * R * (1+R)^N

B = (1+R)^(N-1)

EMI = A/B.
```

```
data["EMI"] = (data["LoanAmount"]*0.09*(1.09**data["Loan_Amount_Term"]))/(1.09**(data["Loan_Amount_Term"]-1))
```

**Replacing the data-values:**

1. “Dependents” variable has some data-values marked as “**3+**”, so changing them as value “**3**”.

```
data['Dependents'].replace('3+','3' ,inplace=True)
```

# Data Storytelling

This step is to explore the dataset to discover certain trends.

The goal of the exploratory data analysis was to find out whether a certain demographic is more likely to get approved compared to others.

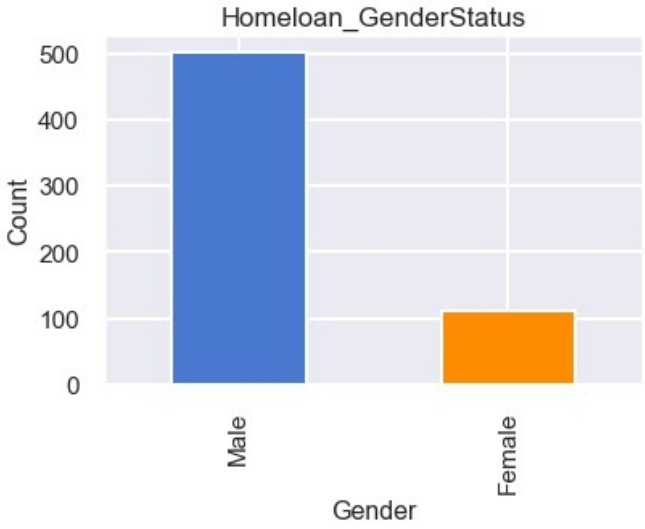
The dataset was truncated to create a dataset centered around gender and education. In order to compare the demographics perfectly, different histograms were used to explore these newly-formed dataset.

Plotting graphs for different features includes repetitive line of codes, so to overcome this scenario different plotting functions were defined. These functions were called as per the need of visualization.

```
def feature__countplot(col):
    plt.figure(figsize=(6, 4))
    data[col].value_counts().plot(kind='bar',color=('b','darkorange'))
    plt.xlabel(col, fontsize=16)
    plt.ylabel('Count', fontsize=16)
    plt.title("Homeloan_"+col+"Status")
    plt.savefig("Homeloan_"+col+"Status.jpeg",bbox_inches = 'tight')
    plt.show()
```

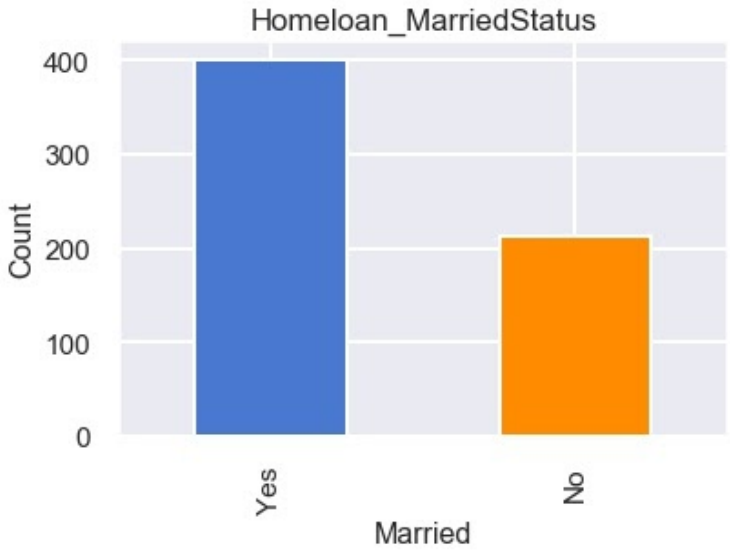
```
def feature_comp_plot(col1,col2):

    fig, ax = plt.subplots(figsize=(6,4))

    x2 = data.groupby([col1,col2])['Loan_ID'].count()
    x2 = x2.reset_index()

    stats = x2[col2].drop_duplicates()
    margin_bottom = np.zeros(len(x2[col1].drop_duplicates()))
    colors = ["#006D2C", "#31A354","#74C476"]

    for num, status in enumerate(stats):
        values = list(x2[x2[col2] == status].loc[:, 'Loan_ID'])
        x2[x2[col2] == status].plot.bar(x=col1,y='Loan_ID', ax=ax, stacked=True,
                                        bottom = margin_bottom, color=colors[num], label=status)
        margin_bottom += values
    ax.set_title(col1+" vs. "+col2)
    plt.savefig(col1+" vs. "+col2+".jpeg",bbox_inches = 'tight')
    plt.show()
```

# Single Variable Analysis



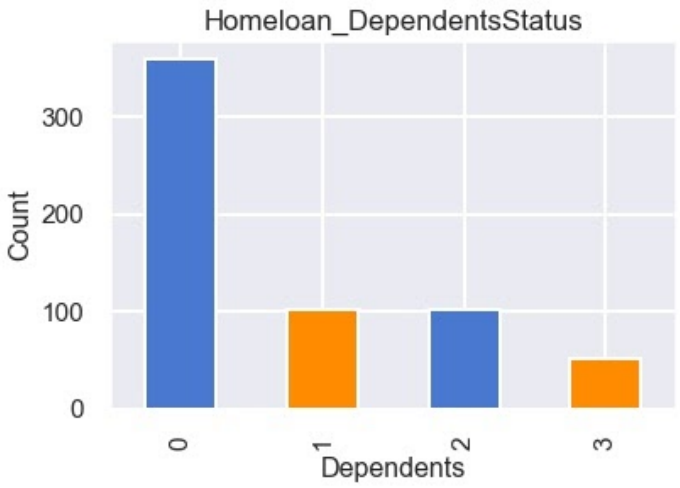

**Gender Column:**

According to our analysis, Gender may influence home loan approval. As we can conclude that, mortgage lenders were more inclined towards men than women expecting men to be the lead borrowers on single applications.



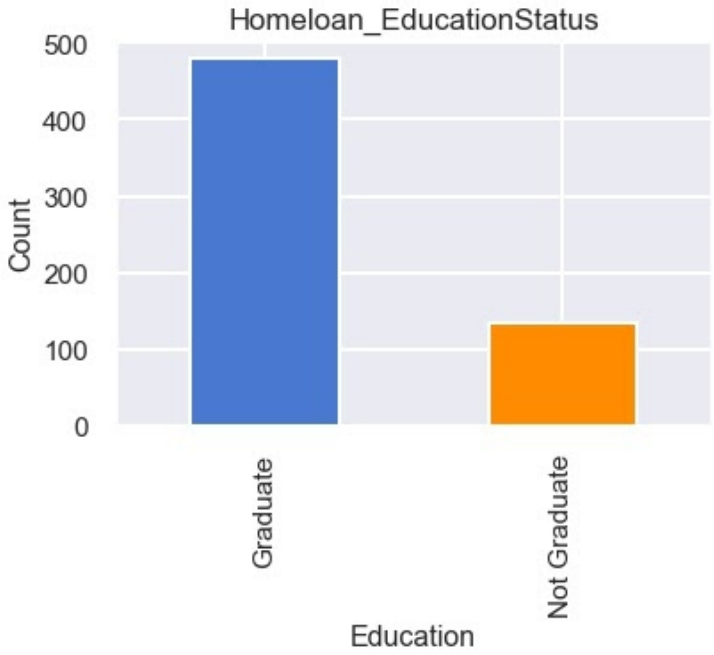

**Marital Status:**

From the above results, we can conclude that most of the home loans were approved to married couples compared to persons who are single or with no relationship.




**Dependents:**

From the analysis, we can conclude that the number of dependents may automatically affected the approvals of home loans. There is a higher chance of getting home loan approval for applicants who have less number of dependents or no dependents.



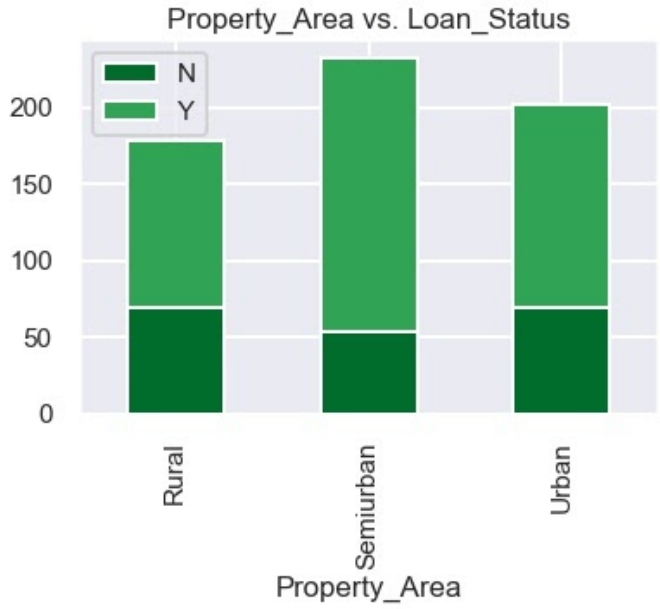

**Education:**

From the analysis, we can conclude that the educational status may automatically affect the approvals of home loans. There is a higher chance of getting home loan approval for applicants who are graduates.


**Employment:**

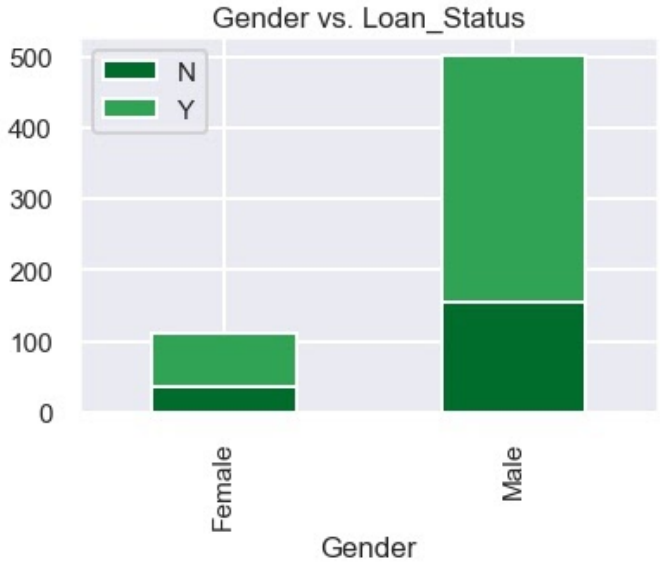
From the analysis, we can conclude that the employment status may automatically affected the approvals of home loans. There is a higher chance of getting home loan approval for applicants who are self_employed.

# Multiple Variable Analysis



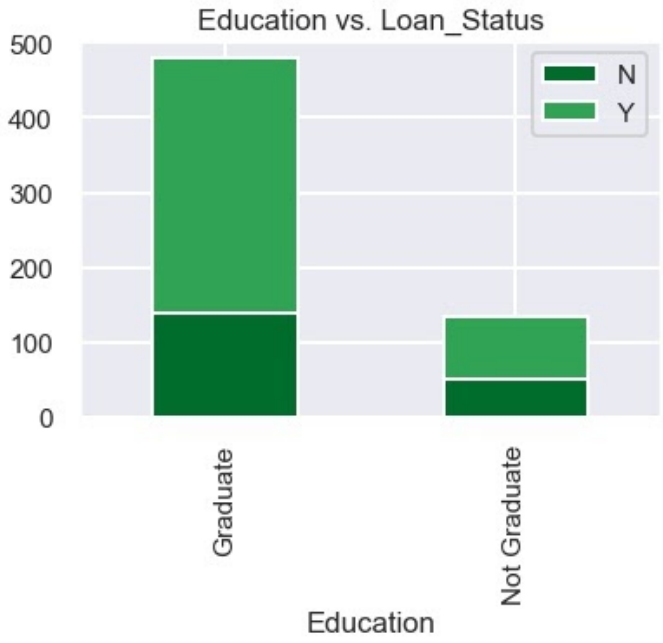

**Relationship between Property area and loan status:**

From the above results we can infer that the higher percentage of loan approval is for semi-urban houses followed by urban and rural houses.




**Relationship between Gender and Loan status:**

From the data analysis, we can conclude that male gender as primary applicants have a higher percentage of loan approval than female as primary applicants.




**Relationship between education vs Loan status:**

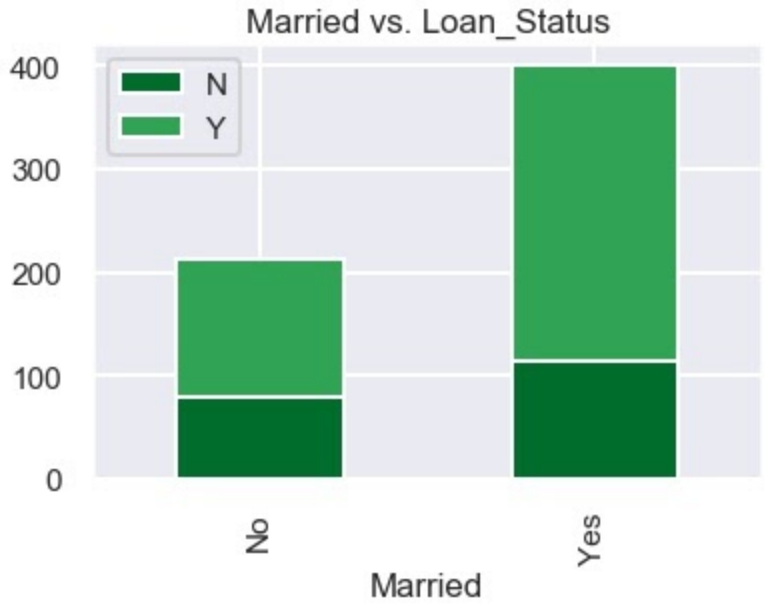
From the analysis, we can conclude that the applicants who are graduate were in a higher percentage of loan approval than non-graduate applicants.


**Relationship between Self-Employed vs Loan_Status:**

From the data analysis, we can conclude that home-ownership rates for self-employed households were more declined than for salaried households.


**Relationship between Dependents vs Loan status:**

From the analysis, we can conclude that the number of dependents may automatically affected the approvals of home loans. There is a higher chance of getting home loan approval for applicants who have less number of dependents or no dependents.




**Relationship between Marital status vs Loan status:**

From the analysis, we can conclude that the highest number of customers are married who were eligible for the home loan approval than single customers.

# Applying Machine Learning Algorithms

**Cleaning Categorical Values:**

Converting the categorical variables into numerical values using map function

```
data.Education=data.Education.map({'Not Graduate':0,'Graduate':1})
```

```
data.Property_Area=data.Property_Area.map({'Rural':0,'Semiurban':1,'Urban':2})
```

```
data.Loan_Status=data.Loan_Status.map({'N':0,'Y':1})
```

```
data.Self_Employed=data.Self_Employed.map({'No':0,'Yes':1})
```

```
data.Married=data.Married.map({'No':0,'Yes':1})
```

```
data.Gender=data.Gender.map({'Female':0,'Male':1})
```

```
data.drop(['Loan_ID'], axis=1, inplace=True)
```

**Separate Target Value From Data:**

Drop target column from data set and store rest values as "**X**" and target column in "**y**" variable.

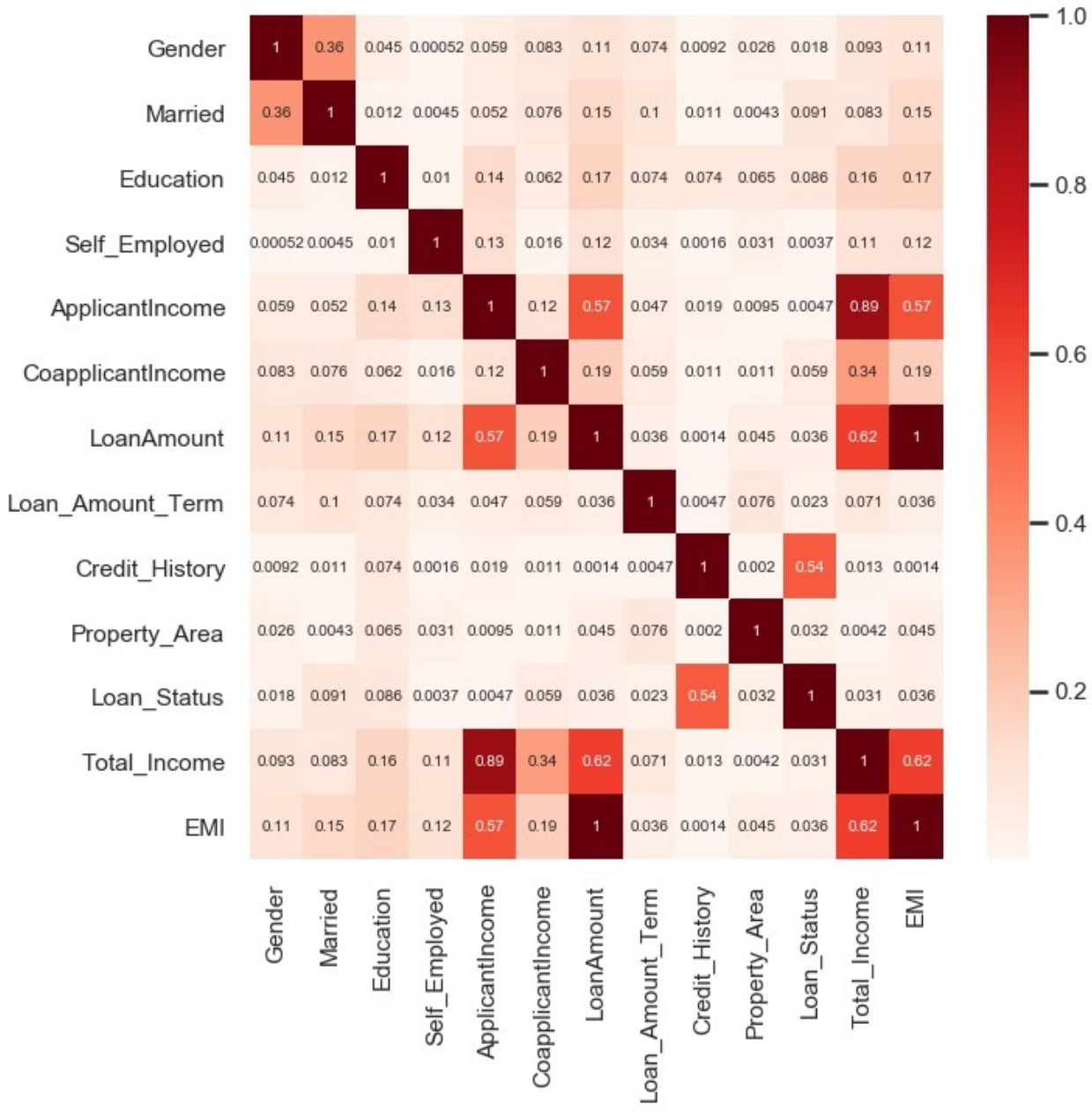
```
X = data.drop(['Loan_Status'], axis=1)
y = data["Loan_Status"]
```

**Test & Train Split:**

Split the data into 2 parts as "Test Data" & "Train Data" using the "**train_test_split**" model into X_train, X_test, y_train, y_test.

```
#split train and test sets

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, random_state=0)
```

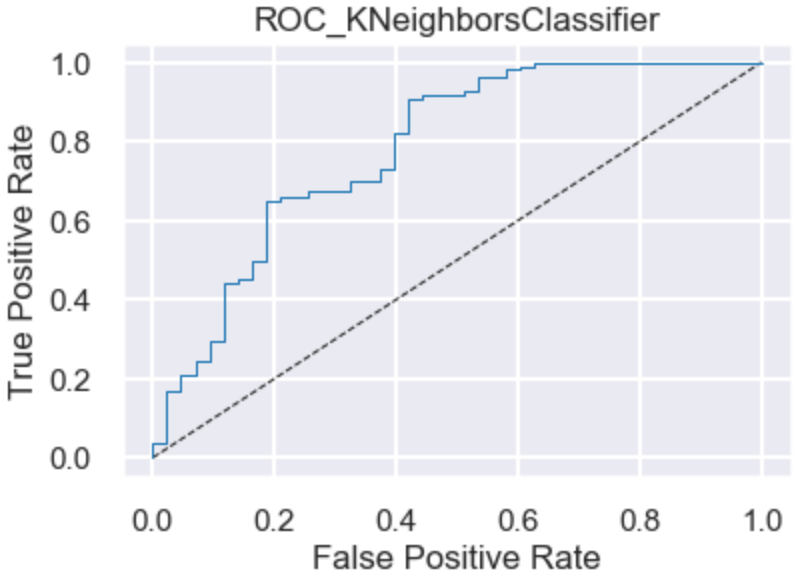
## Correlation Heatmap



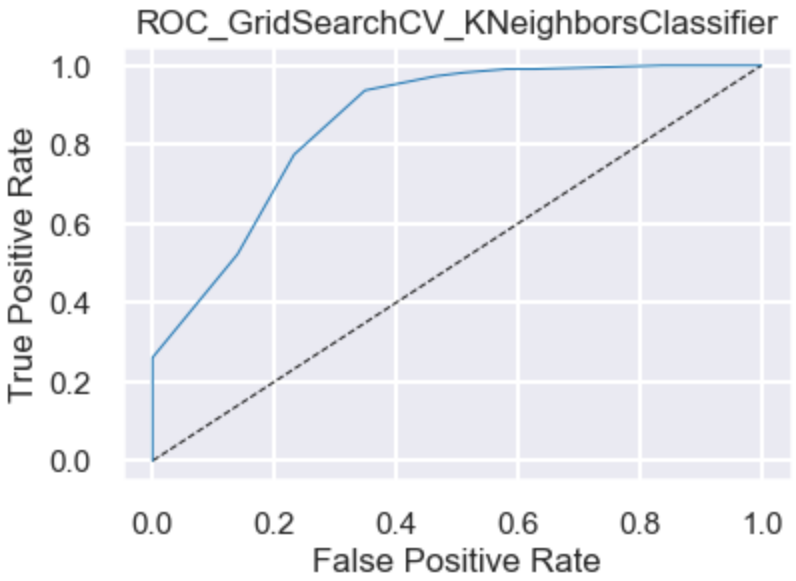
# Applying Different Machine Learning Models

**Logistic Regression: 83.7%**

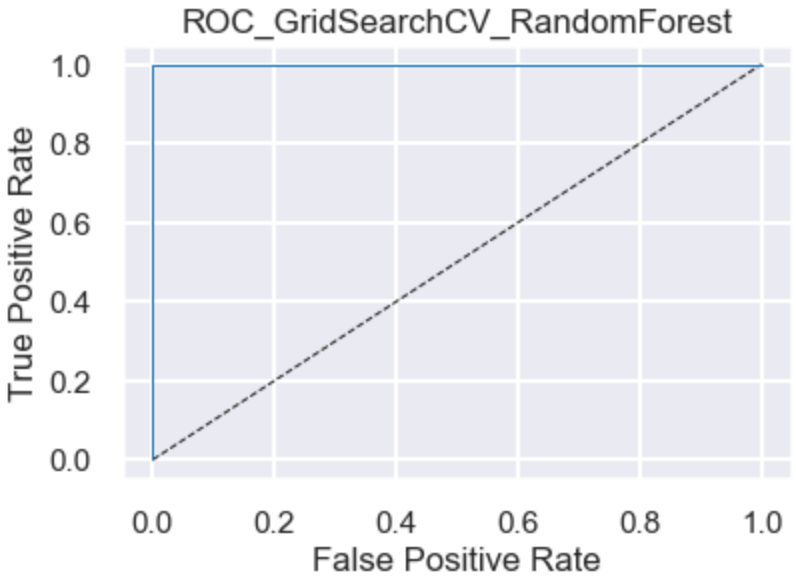

**KNeighborsClassifier: 81.1%**



**GridSearchCV_KNeighborsClassifier: 79.6%**




**GridSearchCV_RandomForest: 79.3%**



**GridSearchCV_AdaBoostClassifier: 80.4%**


# Conclusion:

From the results, we can conclude that the results were 79.6% accurate for loan status predictions from Random Forest model using GridSearchCV & ROC curve generated for this model is best among all used models.

**Results of Machine Learning models:**

Logistic regression model: 83.7%

KNeighborsClassifier model: 81.1%

RandomForestClassifier model: 79.3%

AdaBoostClassifier model: 80.4%

Support vector Machine model: 83.1%

KFold cross_val_score: 80.4%

# Project Code

```
#importing packages

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

from IPython import display
from ipywidgets import interact, widgets

%config InlineBackend.figure_format = 'retina'

# Setup Seaborn
sns.set_style("darkgrid")
sns.set_context("poster",font_scale=0.7,rc={"lines.linewidth": 1})
sns.set_color_codes(palette='muted')

#load data
data = pd.read_csv('Data/Train_Loan_Home.csv')
data.head()
data.info()  #data info before handling missing data

#method to generate most common value
def most_common(col):
    most_cmn=pd.get_dummies(col).sum().sort_values(ascending = False).index[0]
    return most_cmn

#method to replace missing values
def replace_missing(col):
    most_cmn = most_common(col)
    for i in range(len(col)):
        if pd.isnull(col[i]):
            col[i] = most_cmn
            inplace=True

#calling function on each column with missing values
replace_missing(data.Gender)
replace_missing(data.Married)
replace_missing(data.Self_Employed)
replace_missing(data.Dependents)
```

```
replace_missing(data.Credit_History)
replace_missing(data.Loan_Amount_Term)

#handling data values in correct format
data['Dependents'].replace('3+','3' ,inplace=True)
data.LoanAmount[data.LoanAmount.isnull()] = data.LoanAmount.mean()

data.info()   #data info after all operations

#adding a new column as Total_Income with sum of ApplicantIncome & CoapplicantIncome
data["Total_Income"] = data["ApplicantIncome"]+data["CoapplicantIncome"]

#generating EMI column
data["EMI"] =
(data["LoanAmount"]*0.09*(1.09**data["Loan_Amount_Term"]))/(1.09**(data["Loan_Amount_Term"]-
1))

data.head(5) #updated data

#method to plot countplot graphs
def feature__countplot(col):
    plt.figure(figsize=(6, 4))
    data[col].value_counts().plot(kind='bar',color=('b','darkorange'))
    plt.xlabel(col, fontsize=16)
    plt.ylabel('Count', fontsize=16)
    plt.title("Homeloan_"+col+"Status")
    plt.savefig("Graphs/Homeloan_"+col+"Status.jpeg",bbox_inches = 'tight')
    plt.show()

#calling countplot method on each column
feature__countplot("Gender")
feature__countplot("Married")
feature__countplot("Dependents")
feature__countplot("Education")
feature__countplot("Self_Employed")
```

```
#method to plot comparison plot
def feature_comp_plot(col1,col2):

    fig, ax = plt.subplots(figsize=(6,4))

    x2 = data.groupby([col1,col2])['Loan_ID'].count()
    x2 = x2.reset_index()

    stats = x2[col2].drop_duplicates()
    margin_bottom = np.zeros(len(x2[col1].drop_duplicates()))
    colors = ["#006D2C", "#31A354","#74C476"]

    for num, status in enumerate(stats):
        values = list(x2[x2[col2] == status].loc[:, 'Loan_ID'])
        x2[x2[col2] == status].plot.bar(x=col1,y='Loan_ID', ax=ax, stacked=True,
                                bottom = margin_bottom, color=colors[num], label=status)
        margin_bottom += values
    ax.set_title(col1+" vs. "+col2)
    plt.savefig("Graphs/"+col1+" vs. "+col2+".jpeg",bbox_inches = 'tight')
    plt.show()

#calling comparison plot on columns
feature_comp_plot("Property_Area","Loan_Status")
feature_comp_plot("Gender","Loan_Status")
feature_comp_plot("Education","Loan_Status")
feature_comp_plot("Self_Employed","Loan_Status")
feature_comp_plot("Dependents","Loan_Status")
feature_comp_plot("Married","Loan_Status")

#handling categorical values
data.Education=data.Education.map({'Not Graduate':0,'Graduate':1})
data.Property_Area=data.Property_Area.map({'Rural':0,'Semiurban':1,'Urban':2})
data.Loan_Status=data.Loan_Status.map({'N':0,'Y':1})
data.Self_Employed=data.Self_Employed.map({'No':0,'Yes':1})
data.Married=data.Married.map({'No':0,'Yes':1})
data.Gender=data.Gender.map({'Female':0,'Male':1})
data.drop(['Loan_ID'], axis=1, inplace=True)

#splitting data into X and y
X = data.drop(['Loan_Status'], axis=1)
y = data["Loan_Status"]
```

```
#draw heatmap
plt.figure(figsize=(10,10))
import math
cor = abs(data.corr())
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.savefig("Graphs/correlation_graph.jpeg",bbox_inches = 'tight')
plt.show()

#Data Preprocessing
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_scaled = pd.DataFrame(sc_X.fit_transform(X), columns=X.columns)

#split train and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, random_state=0)

#import classifier
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train,y_train)
y_pred = classifier.predict(X_test)
classifier.score(X_test,y_test) #classifier performance on test set

# importing performance measuring tools
from sklearn.metrics import accuracy_score,
confusion_matrix,recall_score,precision_score,classification_report

recall_score(y_test,y_pred,average='macro')          #recall score

cr=classification_report(y_test,y_pred)              #classification report
print(cr)

confusion_matrix(y_test,y_pred)                      #confusion matrix
accuracy_score(y_test,y_pred)                        #accuracy score
precision_score(y_test,y_pred,average='macro')       #precision score
```

```
#method to generate ROC curve
from sklearn.metrics import roc_curve

def roc_generator(model,title):
    y_pred_prob = model.predict_proba(X_test)[:,1]

    # Generate ROC curve values: fpr, tpr, thresholds
    fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

    # Plot ROC curve
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr)
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title("ROC_"+title)
    plt.savefig("Graphs/ROC_"+title,bbox_inches = 'tight')
    plt.show()


roc_generator(classifier,"LogisticRegression")

#import KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=21,weights='distance',p=1)
model.fit(X_train,y_train)
model.score(X_test,y_test)

roc_generator(model,"KNeighborsClassifier")

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train,y_train)
model.score(X_test,y_test)
param_dict={{'n_neighbors':range(3,11,2),'weights':['uniform','distance'],'p':[1,2,3,4,5]})
from sklearn.model_selection import GridSearchCV
best_model=GridSearchCV(model,param_dict,cv=5)
best_model.fit(X_scaled,y)
best_model.best_params_
best_model.best_score_

roc_generator(best_model,"GridSearchCV_KNeighborsClassifier")
```

```
#import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(max_depth=25)
model2.fit(X_train,y_train)
model2.score(X_test,y_test)
param_dict_2=({'n_estimators':range(2,50)})
from sklearn.model_selection import GridSearchCV
best_model=GridSearchCV(model2,param_dict_2,cv=5)
best_model.fit(X_scaled,y)
best_model.best_params_
best_model.best_score_

roc_generator(best_model,"GridSearchCV_RandomForest")

#import AdaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier
model3 = AdaBoostClassifier(n_estimators=20)
model 3.fit(X_train,y_train)
model 3.score(X_test,y_test)
param_dict_3=({'n_estimators':range(2,50)})
from sklearn.model_selection import GridSearchCV
best_model=GridSearchCV(model3,param_dict_3,cv=5)
best_model.fit(X_scaled,y)
best_model.best_params_
best_model.best_score_

roc_generator(best_model,"GridSearchCV_AdaBoostClassifier")

#Support vector Machine model
from sklearn.svm import SVC
model_svc = SVC(kernel='linear',gamma=0.001,C=1.0)
model_svc.fit(X_train,y_train)
model_svc.score(X_test,y_test)

#Estimating the best model using Cross-validation
new_model=best_model.best_estimator_ #gives the best model estimation
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
cross_val_score(new_model, X_scaled,y,cv=5).mean()
```