```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

```
train = pd.read_csv("/content/sample_data/train.csv")
test = pd.read_csv("/content/sample_data/test.csv")
```

```
train_original = train.copy()
test_original = test.copy()
```

```
train.columns
# We have 12 independent variables and 1 target variable, i.e. Loan_Status in the train dataset
```

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

```
test.columns
```

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

```
train.dtypes
```

```
Loan_ID               object
Gender                object
Married               object
Dependents            object
Education             object
Self_Employed         object
ApplicantIncome        int64
```

```
CoapplicantIncome    float64
LoanAmount           float64
Loan_Amount_Term     float64
Credit_History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

```
print('Training data shape: ', train.shape)
train.head()
```

```
print('Test data shape: ', test.shape)
test.head()
```

```
#train["Loan_Status"].size
train["Loan_Status"].count()
```

```
614
```

```
train["Loan_Status"].value_counts()
```

```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

```
# Normalize can be set to True to print proportions instead of number
train["Loan_Status"].value_counts(normalize=True)*100
```

```
Y    68.729642
N    31.270358
Name: Loan_Status, dtype: float64
```

```
train["Loan_Status"].value_counts(normalize=True).plot.bar(title = 'Loan_Status',color = 'orange')
```

```
train["Married"].count()
```

```
611
```

```
train["Married"].value_counts()
```

```
Yes    398
No     213
Name: Married, dtype: int64
```

```
train['Married'].value_counts(normalize=True)*100
```

```
Yes    65.139116
No     34.860884
Name: Married, dtype: float64
```

```
train['Married'].value_counts(normalize=True).plot.bar(title= 'Married')
```

```
train["Self_Employed"].count()
```

```
582
```

```
train["Self_Employed"].value_counts()
```

```
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

```
train['Self_Employed'].value_counts(normalize=True)*100
```

```
No     85.910653
Yes    14.089347
Name: Self_Employed, dtype: float64
```

```
train['Self_Employed'].value_counts(normalize=True).plot.bar(title='Self_Employed')
```

```
train["Credit_History"].count()
```

```
564
```

```
train["Credit_History"].value_counts()
```

```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

```
train['Credit_History'].value_counts(normalize=True)*100
```

```
1.0    84.219858
0.0    15.780142
Name: Credit_History, dtype: float64
```

```
train['Credit_History'].value_counts(normalize=True).plot.bar(title='Credit_History')
```

```
train['Dependents'].count()
```

```
599
```

```
train["Dependents"].value_counts()
```

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

```
train['Dependents'].value_counts(normalize=True)*100
```

```
0     57.595993
1     17.028381
2     16.861436
```

```
3+     8.514190
Name: Dependents, dtype: float64
```

```
train['Dependents'].value_counts(normalize=True).plot.bar(title="Dependents")
```

```
train["Education"].count()
```

```
614
```

```
train["Education"].value_counts()
```

```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

```
train["Education"].value_counts(normalize=True)*100
```

```
Graduate        78.175896
Not Graduate    21.824104
Name: Education, dtype: float64
```

```
train["Education"].value_counts(normalize=True).plot.bar(title = "Education")
```

```
train["Property_Area"].count()
```

```
614
```

```
train["Property_Area"].value_counts()
```

```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

```
train["Property_Area"].value_counts(normalize=True)*100
```

```
Semiurban    37.947883
Urban        32.899023
Rural        29.153094
Name: Property_Area, dtype: float64
```

```
train["Property_Area"].value_counts(normalize=True).plot.bar(title="Property_Area")
```

```
# Independent Variable (Numerical)


plt.figure(1)
plt.subplot(121)
sns.distplot(train["ApplicantIncome"]);

plt.subplot(122)
train["ApplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```

```
train.boxplot(column='ApplicantIncome',by="Education" )
plt.suptitle(" ")
plt.show()
```

```
plt.figure(1)
plt.subplot(121)
sns.distplot(train["CoapplicantIncome"]);

plt.subplot(122)
train["CoapplicantIncome"].plot.box(figsize=(16,5))
plt.show()
```

```
plt.figure(1)
plt.subplot(121)
df=train.dropna()
sns.distplot(df['LoanAmount']);

plt.subplot(122)
train['LoanAmount'].plot.box(figsize=(16,5))

plt.show()
```

```
plt.figure(1)
plt.subplot(121)
df = train.dropna()
```

```
sns.distplot(df["Loan_Amount_Term"]);

plt.subplot(122)
df["Loan_Amount_Term"].plot.box(figsize=(16,5))
plt.show()
```

```
# i)Applicants with high income should have more chances of loan approval.

# ii)Applicants who have repaid their previous debts should have higher chances of loan approval.

# iii)Loan approval should also depend on the loan amount. If the loan amount is less, chances of loan approval should be high.

# iv)Lesser the amount to be paid monthly to repay the loan, higher the chances of loan approval.
```

```
matrix = train.corr()
f, ax = plt.subplots(figsize=(10, 12))
sns.heatmap(matrix, vmax=.8, square=True, cmap="BuPu",annot=True);
```

```
# Missing Value and Outlier Treatment

# After exploring all the variables in our data, we can now impute the missing values and treat the outliers
# because missing data and outliers can have adverse effect on the model performance.

train.isnull()
```

```
train.isnull().sum()
```

```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

```
# There are missing values in Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_History features.

# We will treat the missing values in all the features one by one.

# We can consider these methods to fill the missing values:

# For numerical variables: imputation using mean or median

# For categorical variables: imputation using mode
```

```
train["Gender"].fillna(train["Gender"].mode()[0],inplace=True)
train["Married"].fillna(train["Married"].mode()[0],inplace=True)
train['Dependents'].fillna(train["Dependents"].mode()[0],inplace=True)
train["Self_Employed"].fillna(train["Self_Employed"].mode()[0],inplace=True)
train["Credit_History"].fillna(train["Credit_History"].mode()[0],inplace=True)
```

```
train["Loan_Amount_Term"].value_counts()
```

```
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
```

```
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

```
train["Loan_Amount_Term"].fillna(train["Loan_Amount_Term"].mode()[0],inplace=True)
```

```
train["Loan_Amount_Term"].value_counts()
```

```
360.0    526
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

```
train["LoanAmount"].fillna(train["LoanAmount"].median(),inplace = True)   # we see original chaes to the dataset
```

```
train.isnull()
```

```
train.isnull().sum()
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

```
test.isnull().sum()
```

```
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Self_Employed        23
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
```

```
Property_Area         0
dtype: int64
```

```
test["Gender"].fillna(test["Gender"].mode()[0],inplace=True)
test['Dependents'].fillna(test["Dependents"].mode()[0],inplace=True)
test["Self_Employed"].fillna(test["Self_Employed"].mode()[0],inplace=True)
test["Loan_Amount_Term"].fillna(test["Loan_Amount_Term"].mode()[0],inplace=True)
test["Credit_History"].fillna(test["Credit_History"].mode()[0],inplace=True)
test["LoanAmount"].fillna(test["LoanAmount"].median(),inplace=True)
```

```
test.isnull().sum()
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
dtype: int64
```

```
train["TotalIncome"]=train["ApplicantIncome"]+train["CoapplicantIncome"]
```

```
train[["TotalIncome"]].head()
```

```
test["TotalIncome"]=test["ApplicantIncome"]+test["CoapplicantIncome"]
```

```
# Feature Engineering
# Based on the domain knowledge, we can come up with new features that might affect the target variable. We will create the following three new feat

# Total Income - As discussed during bivariate analysis we will combine the Applicant Income and Coapplicant Income. If the total income is high, ch

# EMI - EMI is the monthly amount to be paid by the applicant to repay the loan. Idea behind making this variable is that people who have high EMI’s

# Balance Income - This is the income left after the EMI has been paid. Idea behind creating this variable is that if this value is high, the chance
```

```
test[["TotalIncome"]].head()
```

```
train["EMI"]=train["LoanAmount"]/train["Loan_Amount_Term"]
test["EMI"]=test["LoanAmount"]/test["Loan_Amount_Term"]
```

```
train[["EMI"]].head()
```

```
test[["EMI"]].head()
```

```
train["Balance_Income"] = train["TotalIncome"]-train["EMI"]*1000 # To make the units equal we multiply with 1000
test["Balance_Income"] = test["TotalIncome"]-test["EMI"]
```

```
train[["Balance_Income"]].head()
```

```
test[["Balance_Income"]].head()
```

```
# Let us now drop the variables which we used to create these new features. Reason for doing this is, the correlation between
# those old features and these new features will be very high and logistic regression assumes that the variables are not
# highly correlated.
# We also wants to remove the noise from the dataset, so removing correlated features will help in reducing the noise too.
```

```
train=train.drop(["ApplicantIncome","CoapplicantIncome","LoanAmount","Loan_Amount_Term"],axis=1)
```

```
train.head()
```

```
test = test.drop(["ApplicantIncome","CoapplicantIncome","LoanAmount","Loan_Amount_Term"],axis=1)
```

```
test.head()
```

```
# After creating new features, we can continue the model building process.
# So we will start with logistic regression model and then move over to more complex models like RandomForest and XGBoost

# We will build the following models in this section.

# i)Logistic Regression

# ii)Decision Tree

# iii)Random Forest

# iv)Random Forest with Grid Search

# v)XGBClassifier
```

```
# Let's drop the "Loan_ID" variable as it do not have any effect on the loan status.
# We will do the same changes to the test dataset which we did for the training dataset.
```

```
train=train.drop("Loan_ID",axis=1)
test=test.drop("Loan_ID",axis=1)
```

```
train.head(3)
```

```
test.head(3)
```

```
# We will use scikit-learn (sklearn) for making different models which is an open source library for Python.
# It is one of the most efficient tool which contains many inbuilt functions that can be used for modeling in Python.

# Sklearn requires the target variable in a separate dataset. So, we will drop our target variable from the train dataset and
# save it in another dataset.
```

```
X=train.drop("Loan_Status",1)
```

```
X.head(2)
```

```
y=train[["Loan_Status"]]
```

```
y.head()
```

```
# Now we will make dummy variables for the categorical variables. Dummy variable turns categorical variables into a series of 0 and 1, making them l

# Let us understand the process of dummies first:

# Consider the “Gender” variable. It has two classes, Male and Female.

# As logistic regression takes only the numerical values as input, we have to change male and female into numerical value.

# Once we apply dummies to this variable, it will convert the “Gender” variable into two variables(Gender_Male and Gender_Female), one for each clas

# Gender_Male will have a value of 0 if the gender is Female and a value of 1 if the gender is Male.
```

```
X = pd.get_dummies(X)
```

```
X.head(3)
```

```
train=pd.get_dummies(train)
test=pd.get_dummies(test)
```

```
train.head(3)
```

```
test.head(3)
```

```
# Now we will train the model on training dataset and
# make predictions for the test dataset. But can we validate these predictions?
# One way of doing this is we can divide our train dataset into two parts:train and validation.
```

```
#  We can train the model on this train part and using that make predictions for the validation part.
#  In this way we can validate our predictions as we have the true predictions for the validation part
#   (which we do not have for the test dataset).

# We will use the train_test_split function from sklearn to divide our train dataset. So, first let us import train_test_split.
```

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_cv,y_train,y_cv=train_test_split(X,y,test_size=0.3,random_state=1)
```

```
# Logistic Regression
# Let's import LogisticRegression and accuracy_score from sklearn and fit the logistic regression model.
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
logistic_model = LogisticRegression(random_state=1)
```

```
logistic_model.fit(x_train,y_train)
```

```
pred_cv_logistic=logistic_model.predict(x_train)
```

```
score_logistic =accuracy_score(pred_cv_logistic,y_train)*100
```

```
score_logistic
```

    81.81818181818183

```
pred_test_logistic = logistic_model.predict(test)
```

```
pred_test_logistic
```

```
array(['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y',
       'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y',
       'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
       'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
       'Y', 'Y', 'Y'], dtype=object)
```

```
predtest=logistic_model.predict(x_cv)
```

```
scoretest =accuracy_score(predtest,y_cv)*100
```

```
scoretest
```

```
78.91891891891892
```

```
len(pred_test_logistic)
```

```
from sklearn.tree import DecisionTreeClassifier
tree_model = DecisionTreeClassifier(random_state=1)
```

```
tree_model.fit(x_train,y_train)
```

```
pred_cv_tree=tree_model.predict(x_train)
```

```
score_tree =accuracy_score(pred_cv_tree,y_train)*100
```

```
score_tree
```

```
100.0
```