Project Name:Loan Approval Prediction using Machine Learning

Introduction:

In [ ]:
```
LOANS are the major requirement of the modern world. By this only, Banks get a major part of the total profit.
It is beneficial for students to manage their education and living expenses, and for people to buy any kind of
luxury like houses, cars, etc.

But when it comes to deciding whether the applicant’s profile is relevant to be granted with loan or not.
Banks have to look after many aspects.

So, here we will be using Machine Learning with Python to ease their work and predict whether the candidate’s
profile is relevant or not using key features like Marital Status, Education, Applicant Income, Credit History, etc.
```

The dataset contains 13 features :

In [ ]:
```
1   Loan    A unique id
2   Gender  Gender of the applicant Male/female
3   Married Marital Status of the applicant, values will be Yes/ No
4   Dependents  It tells whether the applicant has any dependents or not.
5   Education   It will tell us whether the applicant is Graduated or not.
6   Self_Employed   This defines that the applicant is self-employed i.e. Yes/ No
7   ApplicantIncome Applicant income
8   CoapplicantIncome   Co-applicant income
9   LoanAmount  Loan amount (in thousands)
10  Loan_Amount_Term    Terms of loan (in months)
11  Credit_History  Credit history of individual’s repayment of their debts
12  Property_Area   Area of property i.e. Rural/Urban/Semi-urban
13  Loan_Status Status of Loan Approved or not i.e. Y- Yes, N-No
```

Importing Libraries and Dataset

Firstly we have to import libraries :

```
Pandas – To load the Dataframe
Matplotlib – To visualize the data features i.e. barplot
Seaborn – To see the correlation between features using heatmap
```

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("LoanApprovalPrediction.csv")
data
```

Out[1]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Are
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urba
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rura
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urba
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urba
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urba
...	...	...	...	...	...	...	...	...	...	...	...	.
593	LP002978	Female	No	0.0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rura
594	LP002979	Male	Yes	3.0	Graduate	No	4106	0.0	40.0	180.0	1.0	Rura
595	LP002983	Male	Yes	1.0	Graduate	No	8072	240.0	253.0	360.0	1.0	Urba
596	LP002984	Male	Yes	2.0	Graduate	No	7583	0.0	187.0	360.0	1.0	Urba
597	LP002990	Female	No	0.0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurba

598 rows × 13 columns

Once we imported the dataset, let’s view it using the below command.

In [3]:
```
data.head(5)
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	LP001002	Male	No	0.0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban
1	LP001003	Male	Yes	1.0	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban
4	LP001008	Male	No	0.0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban

Data Preprocessing and Visualization

Get the number of columns of object datatype.

In [4]:
```
obj = (data.dtypes == 'object')
print("Categorical variables:",len(list(obj[obj].index)))
```

```
Categorical variables: 7
```

As Loan_ID is completely unique and not correlated with any of the other column, So we will drop it using .drop() function.

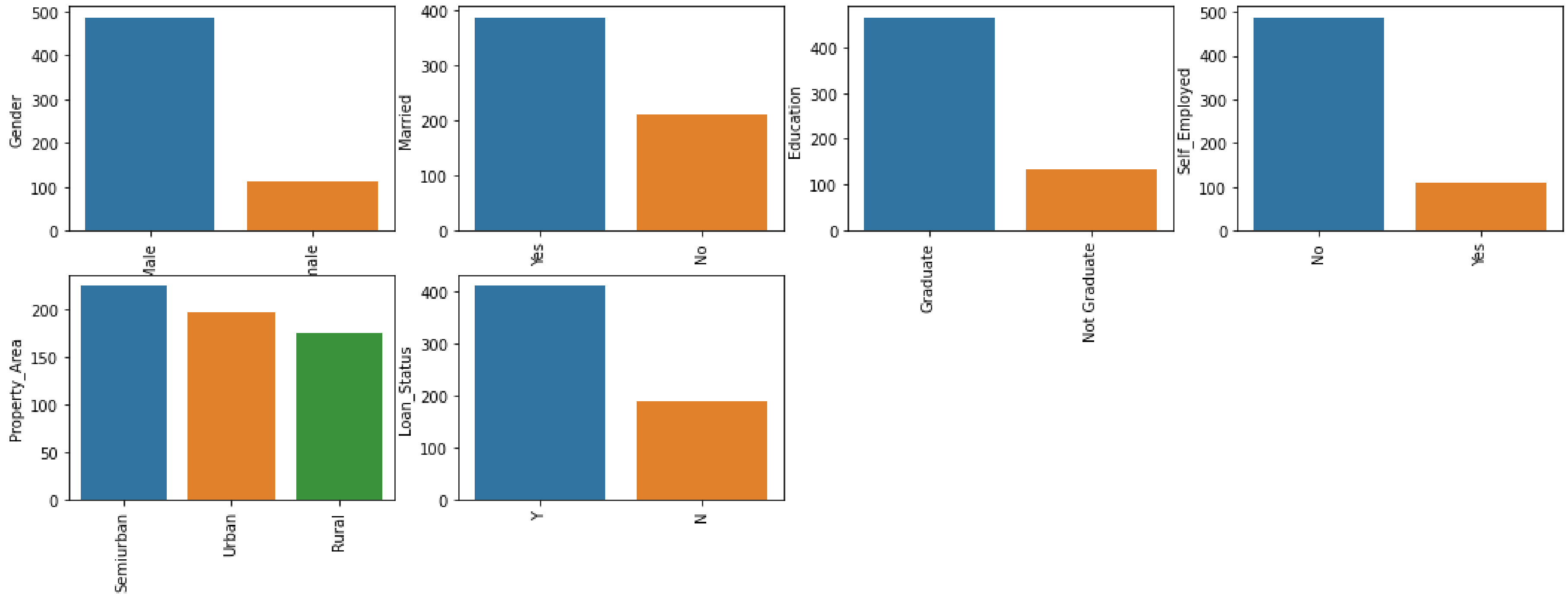
Dropping Loan_ID column

In [5]:
```
data.drop(['Loan_ID'],axis=1,inplace=True)
```

Visualize all the unique values in columns using barplot. This will simply show which value is dominating as per our dataset.

In [6]:
```
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)
plt.figure(figsize=(18,36))
index = 1

for col in object_cols:
  y = data[col].value_counts()
  plt.subplot(11,4,index)
  plt.xticks(rotation=90)
  sns.barplot(x=list(y.index), y=y)
  index +=1
```



As all the categorical values are binary so we can use Label Encoder for all such columns and the values will change into int datatype.

In [7]:
```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how
# to understand word labels.
label_encoder = preprocessing.LabelEncoder()
obj = (data.dtypes == 'object')
for col in list(obj[obj].index):
  data[col] = label_encoder.fit_transform(data[col])
```

Again check the object datatype columns. Let’s find out if there is still any left.

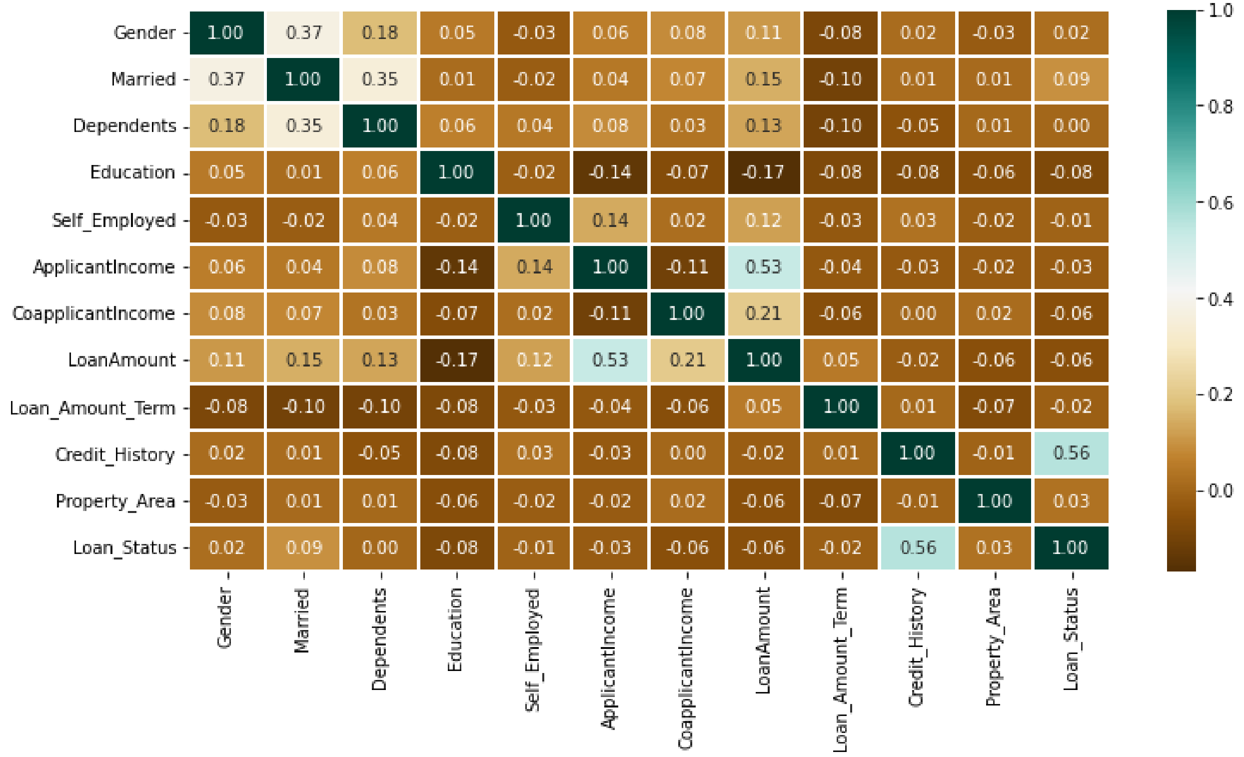
In [8]:
```
# To find the number of columns with
# datatype==object
obj = (data.dtypes == 'object')
print("Categorical variables:",len(list(obj[obj].index)))
```

```
Categorical variables: 0
```

In [9]:
```
plt.figure(figsize=(12,6))

sns.heatmap(data.corr(),cmap='BrBG',fmt='.2f',
            linewidths=2,annot=True)
```

Out[9]: <AxesSubplot:>

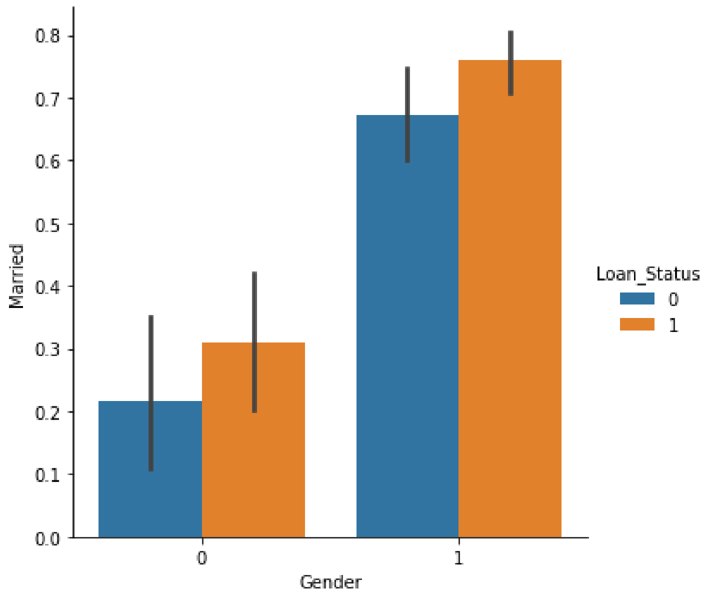


The above heatmap is showing the correlation between Loan Amount and ApplicantIncome. It also shows that Credit_History has a high impact on Loan_Status.

Now we will use Catplot to visualize the plot for the Gender, and Marital Status of the applicant.

In [10]:
```
sns.catplot(x="Gender", y="Married",
            hue="Loan_Status",
            kind="bar",
            data=data)
```

Out[10]: <seaborn.axisgrid.FacetGrid at 0x1c6343162e0>



Now we will find out if there is any missing values in the dataset using below code.

In [11]:
```
for col in data.columns:
  data[col] = data[col].fillna(data[col].mean())

data.isna().sum()
```

Out[11]:
```
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
```

As there is no missing value then we must proceed to model training.

In [15]:
```
from sklearn.model_selection import train_test_split
X_train,X_test, Y_train, Y_test = train_test_split(X, Y,test_size=0.2,random_state=1)

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

Model Training and Evaluation

As this is a classification problem so we will be using these models :

```
KNeighborsClassifiers
RandomForestClassifiers
Support Vector Classifiers (SVC)
Logistics Regression
```

To predict the accuracy we will use the accuracy score function from scikit-learn library.

In [16]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression

from sklearn import metrics

knn = KNeighborsClassifier(n_neighbors=3)
rfc = RandomForestClassifier(n_estimators = 7,
                             criterion = 'entropy',
                             random_state =7)
svc = SVC()
lc = LogisticRegression()

# making predictions on the training set
for clf in (rfc, knn, svc,lc):
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_train)
    print("Accuracy score of ",
          clf.__class__.__name__,
          "=",100*metrics.accuracy_score(Y_train,
                                         Y_pred))
```

```
Accuracy score of  RandomForestClassifier = 96.02510460251045
Accuracy score of  KNeighborsClassifier = 79.9163179916318
Accuracy score of  SVC = 68.41004184100419
Accuracy score of  LogisticRegression = 79.70711297071131
```

Prediction on the test set

In [17]:
```
# making predictions on the testing set
for clf in (rfc, knn, svc,lc):
    clf.fit(X_train, Y_train)
    Y_pred = clf.predict(X_test)
    print("Accuracy score of ",
          clf.__class__.__name__,"=",
          100*metrics.accuracy_score(Y_test,
                                     Y_pred))
```

```
Accuracy score of  RandomForestClassifier = 80.0
Accuracy score of  KNeighborsClassifier = 60.0
Accuracy score of  SVC = 72.5
Accuracy score of  LogisticRegression = 85.0
```

Conclusion :

Random Forest Classifier is giving the best accuracy with an accuracy score of 82% for the testing dataset. And to get much better results ensemble learning techniques like Bagging and Boosting can also be used.

In [ ]:

```

```