1. Introduction

1.1 Problem Statement:

Dream Housing Finance company deals in all home loans. They have a presence across all urban, semi-urban, and rural areas. Customer-first applies for a home loan after that company validates the customer eligibility for a loan.

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

1.2 Objective

1. Analyze the dataset and find the hidden patterns through EDA(Exploratory Data Analysis).
2. Preprocess the data to get a clean data.
3. Train the best possible Model to predict the outcome and tune it for best outcome.
4. Deploy the Model.

Machine Learning algo used in this project:

1. Logistic Regression
2. Decision Tree
3. Random Forest
4. Naive Bayes
5. k-Nearest Neighbours(kNN)

Dataset key information:

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 a loan in months

Credit_History ----> credit history meets guidelines

Property_Area ----> Urban/ Semi-Urban/ Rural

Loan_Status ----> Loan approved (Y/N)

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

2. Importing Libraries

Librearies are collection of related modules which contains code that can be used repeatedly in different programs. Importing libraries used in this project

In [4]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\aksha\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\aksha\anaconda3\lib\site-packages (from imblearn) (0.9.0)
Requirement already satisfied: numpy>=1.14.6 in c:\users\aksha\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.20.3)
Requirement already satisfied: joblib>=0.11 in c:\users\aksha\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\aksha\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (2.2.0)
Requirement already satisfied: scipy>=1.1.0 in c:\users\aksha\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.7.1)
Requirement already satisfied: scikit-learn>=1.0.1 in c:\users\aksha\anaconda3\lib\site-packages (from imbalanced-learn->imblearn) (1.1.0)
```

In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import CategoricalNB
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

3. Exploratory Data Analysis

- Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.
- Your goal during EDA is to develop an understanding of your data. The easiest way to do this is to use questions as tools to guide your investigation. When you ask a question, the question focuses your attention on a specific part of your dataset and helps you decide which graphs, models, or transformations to make.

- Search for answers by visualising, transforming, and modelling your data.
- Let's start exploring our data

3.1 Importing dataset

In [6]:
```
data = pd.read_csv("loan-train.csv")
databackup = data.copy()
#creating backup for future
```

3.2 Understanding the "raw" data

In [7]:
```
data.shape
```

Out[7]:
```
(614, 13)
```

In [8]:
```
data.head()
```

Out[8]:

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

In [9]:
```
data.columns
```

Out[9]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [10]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

In [11]:
```
data.describe()
#This function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.
```

Out[11]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [12]:
```
# Inference
# 1. Raw data has missing values in Gender, Married, Dependents, Self_Employed, LoanAmount, Loan_Amount_Term and Credit_history.
# 2. Mean in ApplicantIncome and CoapplicantIncome is much lower then their max value, probablity of an outlier.
# 3. The Raw data is right skewed as the values are much more spread out at higher range.
```

In [13]:
```
data["Loan_Status"].value_counts()
# The raw data is imbalanced
```

Out[13]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [14]:
```
categorical_columns = [column for column in data.columns if data[column].dtypes == 'O']
categorical_columns
```

Out[14]:
```
['Loan_ID',
 'Gender',
 'Married',
 'Dependents',
 'Education',
 'Self_Employed',
 'Property_Area',
 'Loan_Status']
```

In [15]:
```
numerical_columns = [column for column in data.columns if data[column].dtypes != 'O']
numerical_columns
```

Out[15]:
```
['ApplicantIncome',
 'CoapplicantIncome',
 'LoanAmount',
 'Loan_Amount_Term',
 'Credit_History']
```

In [16]:
```
data.isnull().sum()
#to check the null values in the data set
#since the dataset is small , dropping the rows with null values is not right.
```

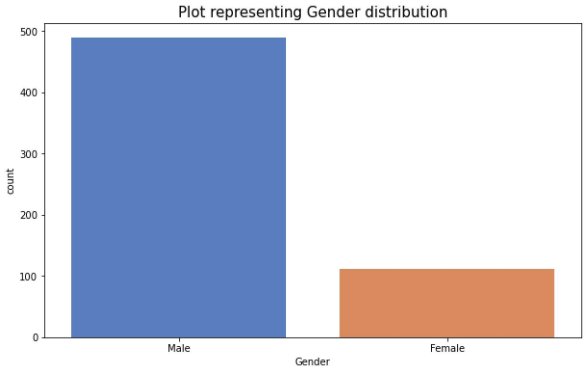
Out[16]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
```

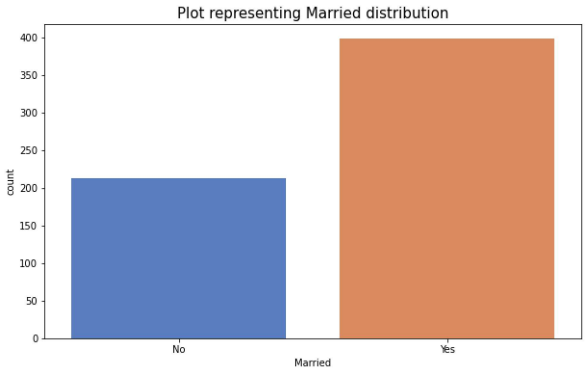
```
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

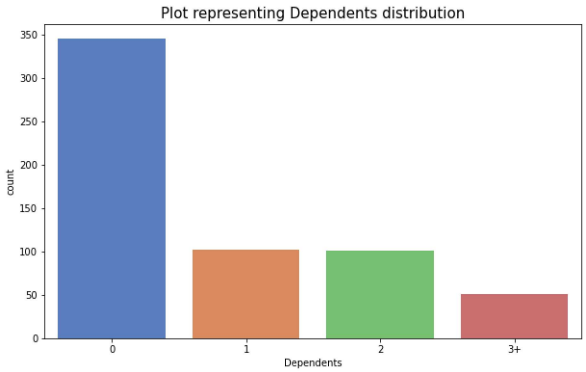
3.3 Categorical Columns

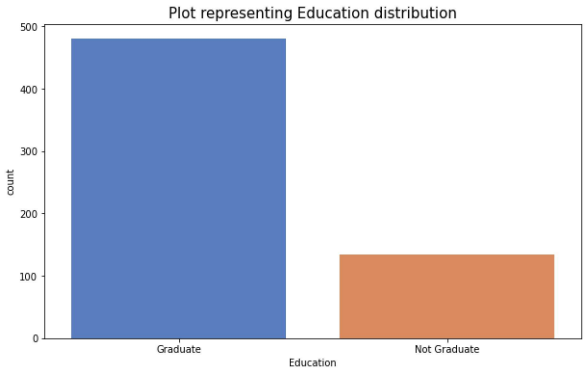
Columns that does not have continuous values and have discreet entries are call categorical columns.

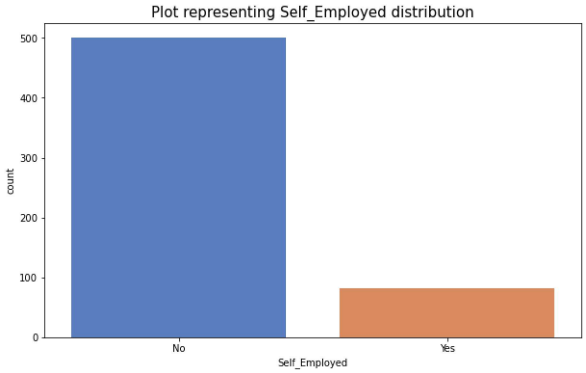
In [17]:
```
for col in categorical_columns:
    if(col != "Loan_Status" and col!="Loan_ID"):
        plt.figure(figsize=(10, 6))
        sns.countplot(x = col,data = data,palette = "muted")
        plt.title("Plot representing {} distribution".format(col),fontsize = 15)
        plt.show()
```



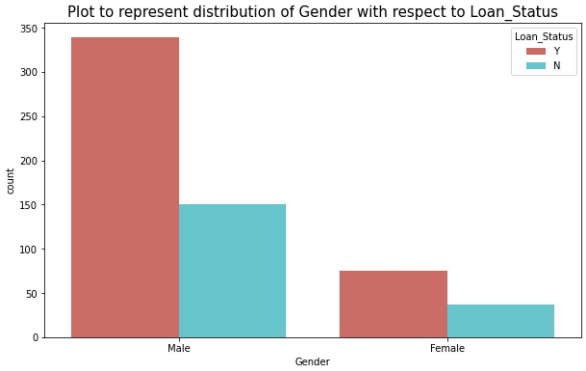


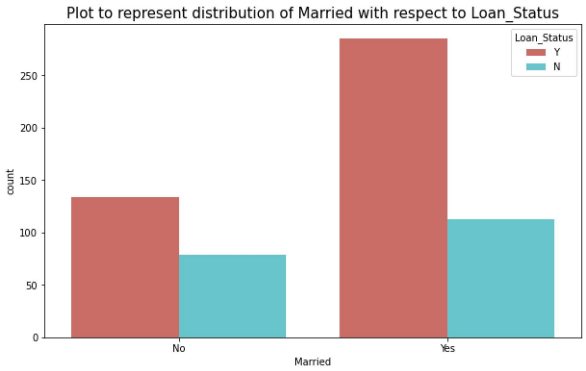


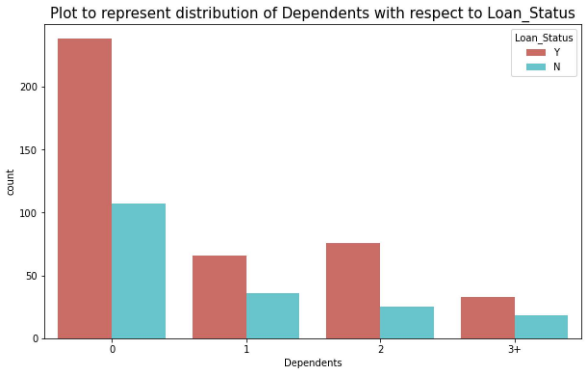


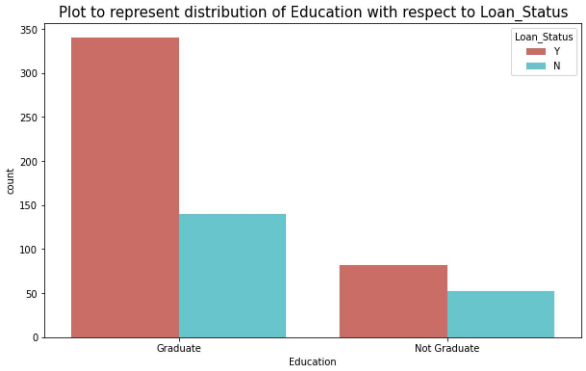


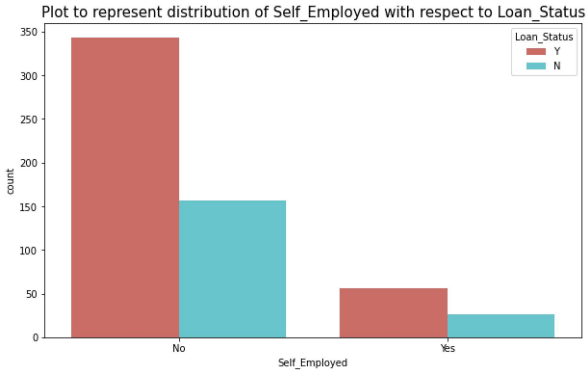
In [18]:
```
for index,col_name in enumerate(categorical_columns):
    if(col_name!="Loan_ID" and col_name!="Loan_Status"):
        plt.figure(figsize = (10,6))
        sns.countplot(x = col_name,data = data,hue = "Loan_Status", palette = "hls")
        plt.title("Plot to represent distribution of {} with respect to Loan_Status".format(col_name),fontsize = 15)
        plt.show()
```

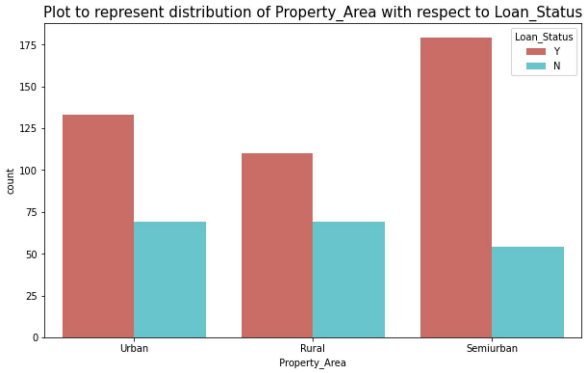












In [19]:
```
for col in categorical_columns:
    if(col!="Loan_Status" and col!="Loan_ID"):
        print("{} column value distribution".format(col))
        print(data[col].value_counts(dropna = False))
        print("**********")
```

```
Gender column value distribution
Male      489
Female    112
NaN        13
Name: Gender, dtype: int64
**********
Married column value distribution
Yes    398
No     213
NaN      3
Name: Married, dtype: int64
**********
Dependents column value distribution
0      345
1      102
2      101
3+      51
NaN     15
Name: Dependents, dtype: int64
**********
Education column value distribution
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
**********
Self_Employed column value distribution
No     500
Yes     82
NaN     32
Name: Self_Employed, dtype: int64
**********
Property_Area column value distribution
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
**********
```

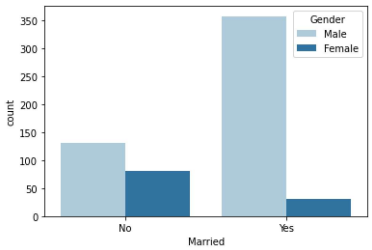
In [20]:
```
# The data shows that the data has more male applicants and married applicants.
# It also shows that among the applicants, they are more likely to be graduated and less self employed.
# Applicants are more likely to get the loan if they are graduate and living in suburban areas.
```

3.3.1 Gender

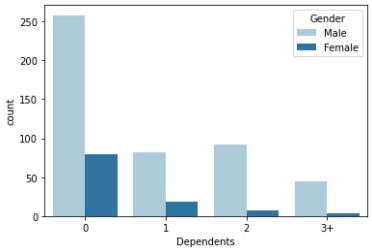
In [21]:
```
for col in categorical_columns:
    if(col!="Loan_ID" and col!="Gender"):
        print(data.groupby(col)["Gender"].value_counts(dropna = False))

        sns.countplot(x = col,hue="Gender",data = data,palette = "Paired")
        plt.show()
```

```
        print('********************')
```

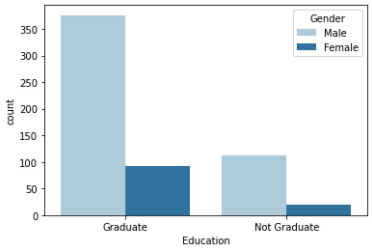
```
Married  Gender
No       Male      130
         Female     80
         NaN         3
Yes      Male      357
         Female     31
         NaN        10
Name: Gender, dtype: int64
```



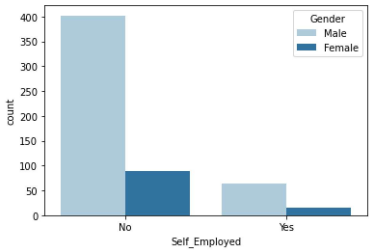

```
********************
Dependents  Gender
0           Male      258
            Female     80
            NaN         7
1           Male       82
            Female     19
            NaN         1
2           Male       92
            Female      7
            NaN         2
3+          Male       45
            NaN         3
            Female      3
Name: Gender, dtype: int64
```



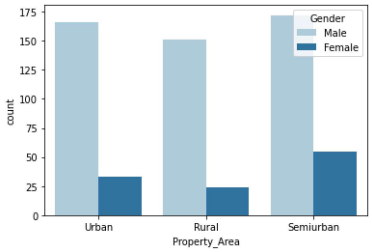

```
********************
Education     Gender
Graduate      Male      376
              Female     92
              NaN        12
Not Graduate  Male      113
              Female     20
              NaN         1
Name: Gender, dtype: int64
```



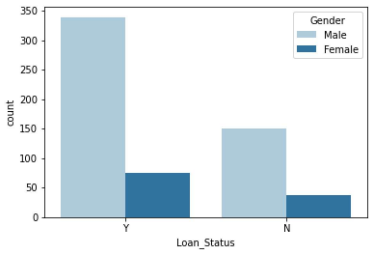

```
********************
Self_Employed  Gender
No             Male      402
               Female     89
               NaN         9
Yes            Male       63
               Female     15
               NaN         4
Name: Gender, dtype: int64
```




```
********************
Property_Area  Gender
Rural          Male      151
               Female     24
               NaN         4
Semiurban      Male      172
               Female     55
               NaN         6
Urban          Male      166
               Female     33
               NaN         3
Name: Gender, dtype: int64
```




```
********************
Loan_Status  Gender
N            Male      150
             Female     37
             NaN         5
Y            Male      339
             Female     75
             NaN         8
Name: Gender, dtype: int64
```



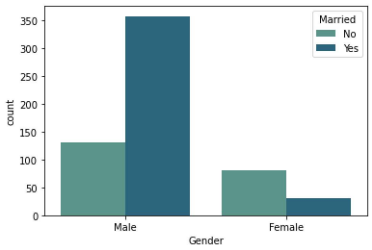

```
********************
```

In [22]:
```
# Most of the married applicants are male.
# Most of the Applicants that are not self employed are males.
# Applicants whose Loan was approved are male.
```

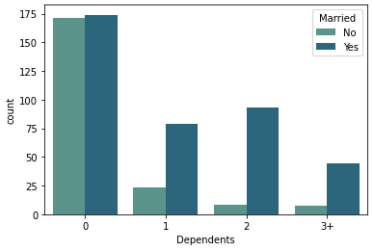
3.3.2 Married

In [23]:
```
for col in categorical_columns:
    if(col!="Loan_ID" and col!="Married"):
        print(data.groupby(col)["Married"].value_counts(dropna = False))

        sns.countplot(x = col,hue="Married",data = data,palette = "crest")
        plt.show()

        print('********************')
```

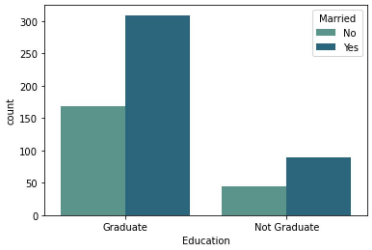
```
Gender  Married
Female  No          80
        Yes         31
        NaN          1
Male    Yes        357
        No         130
        NaN          2
Name: Married, dtype: int64
```



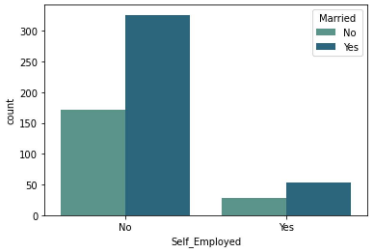

```
********************
Dependents  Married
0           Yes        174
            No         171
1           Yes         79
            No          23
2           Yes         93
            No           8
3+          Yes         44
            No           7
Name: Married, dtype: int64
```



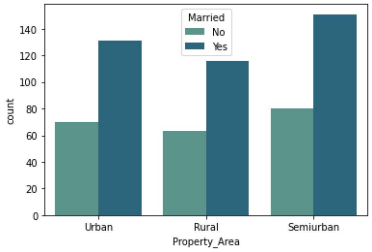

```
********************
Education     Married
Graduate      Yes        309
              No         168
              NaN          3
Not Graduate  Yes         89
              No          45
Name: Married, dtype: int64
```



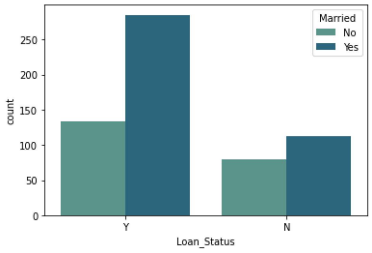

```
**********
Self_Employed  Married
No             Yes        326
               No         171
               NaN          3
Yes            Yes         54
               No          28
Name: Married, dtype: int64
```




```
**********
Property_Area  Married
Rural          Yes        116
               No          63
Semiurban      Yes        151
               No          80
               NaN          2
Urban          Yes        131
               No          70
               NaN          1
Name: Married, dtype: int64
```




```
**********
Loan_Status  Married
N            Yes        113
             No          79
Y            Yes        285
             No         134
             NaN          3
Name: Married, dtype: int64
```



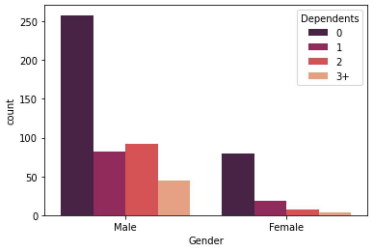

```
**********
```

In [24]:
```
# Most Graduates are married.
# Most non self employed applicants are married.
```

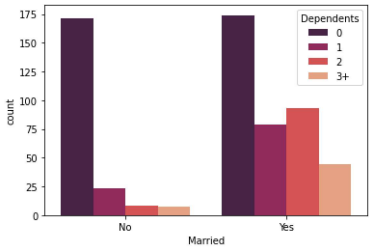
3.3.3 Dependents

In [25]:
```
for col in categorical_columns:
    if(col!="Loan_ID" and col!="Dependents"):
        print(data.groupby(col)["Dependents"].value_counts(dropna = False))

        sns.countplot(x = col,data= data,hue = "Dependents",palette = "rocket")
        plt.show()
        print('**********')
```

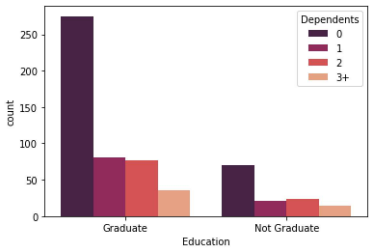
```
Gender  Dependents
Female  0             80
        1             19
        2              7
        NaN            3
        3+             3
Male    0            258
        2             92
        1             82
        3+            45
        NaN           12
Name: Dependents, dtype: int64
```



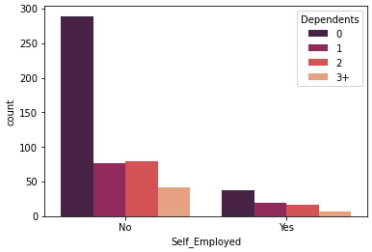

```
********************
Married  Dependents
No       0             171
         1              23
         2               8
         3+              7
         NaN             4
Yes      0             174
         2              93
         1              79
         3+             44
         NaN             8
Name: Dependents, dtype: int64
```



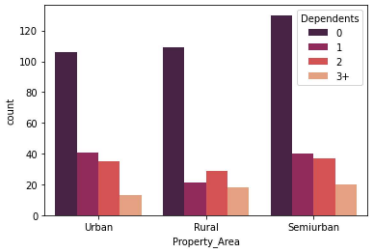

```
********************
Education     Dependents
Graduate      0             275
              1              81
              2              77
              3+             36
              NaN            11
Not Graduate  0              70
              2              24
              1              21
              3+             15
              NaN             4
Name: Dependents, dtype: int64
```



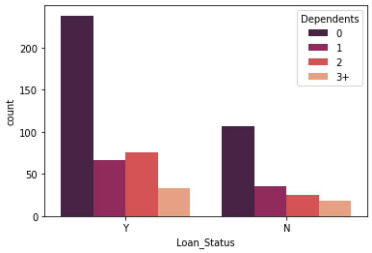

```
********************
Self_Employed  Dependents
No             0             289
               2              80
               1              76
               3+             42
               NaN            13
Yes            0              37
               1              20
               2              16
               3+              7
               NaN             2
Name: Dependents, dtype: int64
```




```
********************
Property_Area  Dependents
Rural          0             109
               2              29
               1              21
               3+             18
               NaN             2
Semiurban      0             130
               1              40
               2              37
               3+             20
               NaN             6
Urban          0             106
               1              41
               2              35
               3+             13
               NaN             7
Name: Dependents, dtype: int64
```




```
********************
Loan_Status  Dependents
N            0             107
             1              36
             2              25
             3+             18
             NaN             6
Y            0             238
             2              76
             1              66
             3+             33
             NaN             9
Name: Dependents, dtype: int64
```



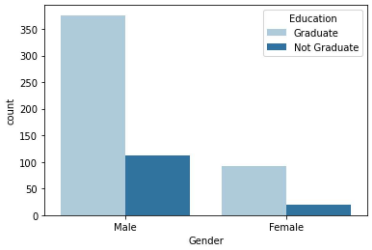

```
********************
```

In [26]:
```
# Semi urban and urban have higher number of dependents.
```

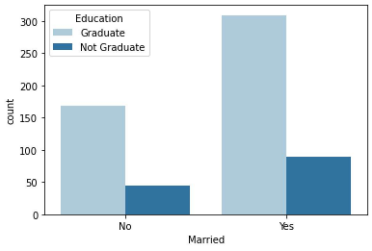
3.3.4 Education

In [27]:
```
for col in categorical_columns:
    if(col!="Loan_ID" and col!="Education"):
        print(data.groupby(col)["Education"].value_counts(dropna = False))

        sns.countplot(x = col,data = data,hue = "Education",palette = "Paired")
        plt.show()

        print('********************')
```

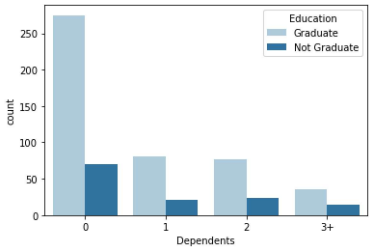
```
Gender  Education
Female  Graduate         92
        Not Graduate     20
Male    Graduate        376
        Not Graduate    113
Name: Education, dtype: int64
```



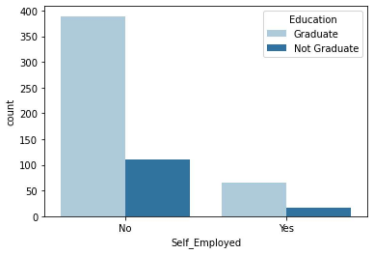

```
********************
Married  Education
No       Graduate        168
         Not Graduate     45
Yes      Graduate        309
         Not Graduate     89
Name: Education, dtype: int64
```



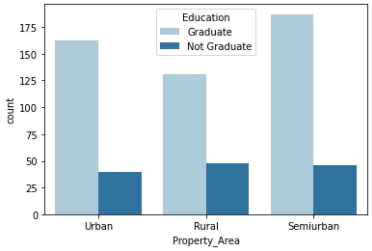

```
********************
Dependents  Education
0           Graduate        275
            Not Graduate     70
1           Graduate         81
            Not Graduate     21
2           Graduate         77
            Not Graduate     24
3+          Graduate         36
            Not Graduate     15
Name: Education, dtype: int64
```



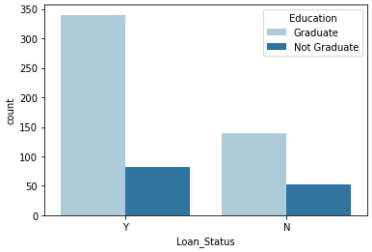

```
********************
Self_Employed  Education
No             Graduate        389
               Not Graduate    111
Yes            Graduate         65
               Not Graduate     17
Name: Education, dtype: int64
```




```
********************
Property_Area  Education
Rural          Graduate        131
               Not Graduate     48
Semiurban      Graduate        187
               Not Graduate     46
Urban          Graduate        162
               Not Graduate     40
Name: Education, dtype: int64
```




```
********************
Loan_Status  Education
N            Graduate        140
             Not Graduate     52
Y            Graduate        340
             Not Graduate     82
Name: Education, dtype: int64
```

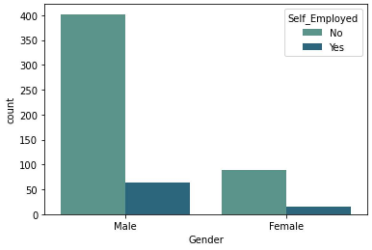



```
********************
```

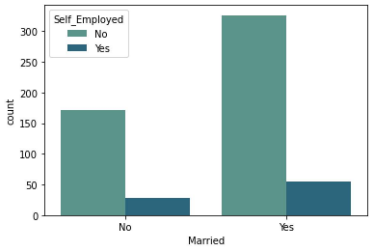
In [28]:

```
for col in categorical_columns:
    if(col!="Loan_ID" and col!="Self_Employed"):
        print(data.groupby(col)["Self_Employed"].value_counts(dropna = False))

        sns.countplot(x =col,data = data,hue = "Self_Employed",palette = "crest")
        plt.show()

        print('**********')
```

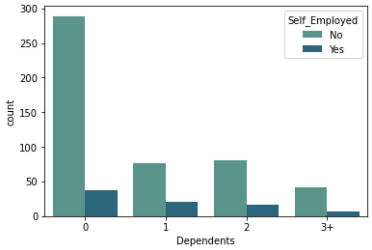
```
Gender  Self_Employed
Female  No               89
        Yes              15
        NaN               8
Male    No              402
        Yes              63
        NaN              24
Name: Self_Employed, dtype: int64
```



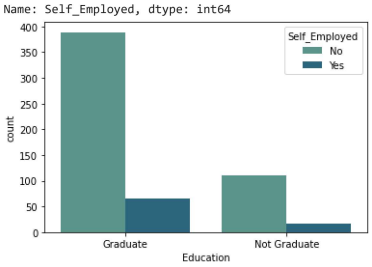

```
********************
Married  Self_Employed
No       No              171
         Yes              28
         NaN              14
Yes      No              326
         Yes              54
         NaN              18
Name: Self_Employed, dtype: int64
```



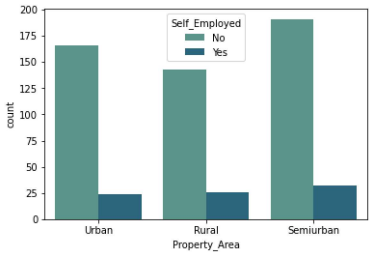

```
********************
Dependents  Self_Employed
0           No               289
            Yes               37
            NaN               19
1           No                76
            Yes               20
            NaN                6
2           No                80
            Yes               16
            NaN                5
3+          No                42
            Yes                7
            NaN                2
Name: Self_Employed, dtype: int64
```



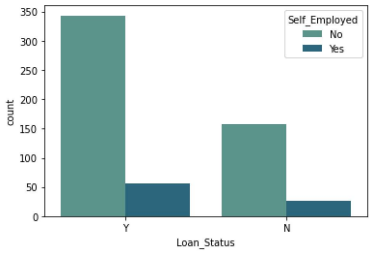

```
********************
Education     Self_Employed
Graduate      No               389
              Yes               65
              NaN               26
Not Graduate  No               111
              Yes               17
              NaN                6
Name: Self_Employed, dtype: int64
```




```
********************
Property_Area  Self_Employed
Rural          No               143
               Yes               26
               NaN               10
Semiurban      No               191
               Yes               32
               NaN               10
Urban          No               166
               Yes               24
               NaN               12
Name: Self_Employed, dtype: int64
```




```
********************
Loan_Status  Self_Employed
N            No               157
             Yes               26
             NaN                9
Y            No               343
             Yes               56
             NaN               23
Name: Self_Employed, dtype: int64
```



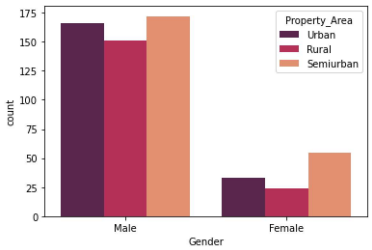

```
********************
```

In [29]:
```
# Males are much less self employed
```

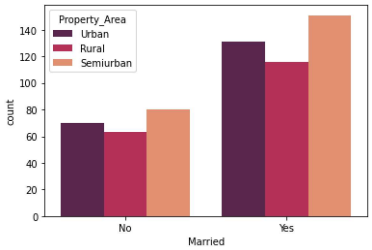
In [30]:
```
```

```
for col in categorical_columns:
    if(col!="Loan_ID" and col!="Property_Area"):
        print(data.groupby(col)["Property_Area"].value_counts(dropna = False))

        sns.countplot(x = col, data = data,hue = "Property_Area",palette = "rocket")
        plt.show()

        print('********************')
```

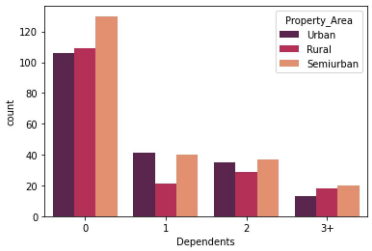
```
Gender  Property_Area
Female  Semiurban          55
        Urban              33
        Rural              24
Male    Semiurban         172
        Urban             166
        Rural             151
Name: Property_Area, dtype: int64
```



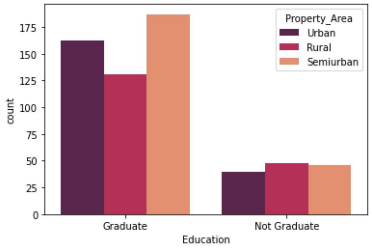

```
********************
Married  Property_Area
No       Semiurban          80
         Urban              70
         Rural              63
Yes      Semiurban         151
         Urban             131
         Rural             116
Name: Property_Area, dtype: int64
```



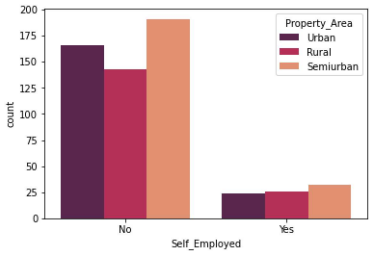

```
********************
Dependents  Property_Area
0           Semiurban        130
            Rural            109
            Urban            106
1           Urban             41
            Semiurban         40
            Rural             21
2           Semiurban         37
            Urban             35
            Rural             29
3+          Semiurban         20
            Rural             18
            Urban             13
Name: Property_Area, dtype: int64
```



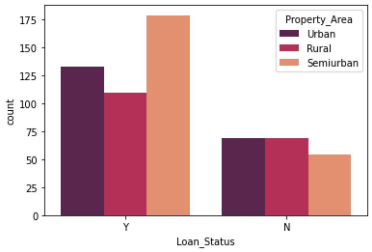

```
********************
Education     Property_Area
Graduate      Semiurban        187
              Urban            162
              Rural            131
Not Graduate  Rural             48
              Semiurban         46
              Urban             40
Name: Property_Area, dtype: int64
```




```
********************
Self_Employed  Property_Area
No             Semiurban        191
               Urban            166
               Rural            143
Yes            Semiurban         32
               Rural             26
               Urban             24
Name: Property_Area, dtype: int64
```




```
********************
Loan_Status  Property_Area
N            Rural            69
             Urban            69
             Semiurban        54
Y            Semiurban       179
             Urban           133
             Rural           110
Name: Property_Area, dtype: int64
```

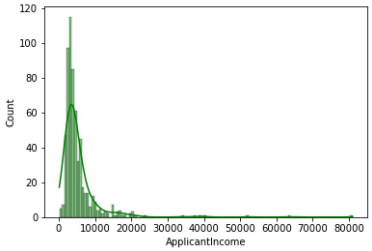


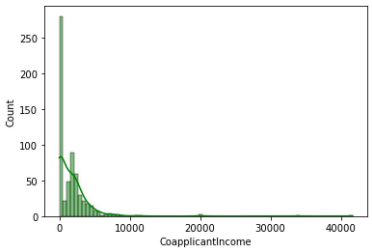

```
********************
```

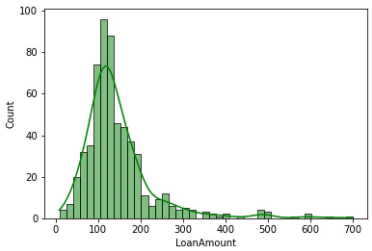
3.4 Numerical Columns

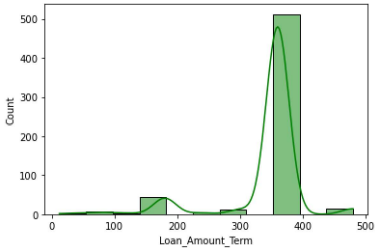
Numerical columns have data types as numericals(int, float) , they have continuous values.

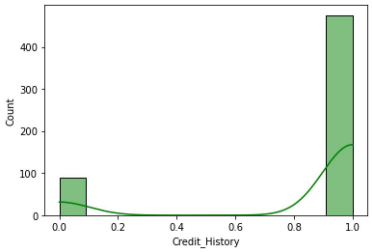
In [31]:
```
for col in numerical_columns:
    sns.histplot(data[col],kde = True,color = "green")
    plt.show()
```







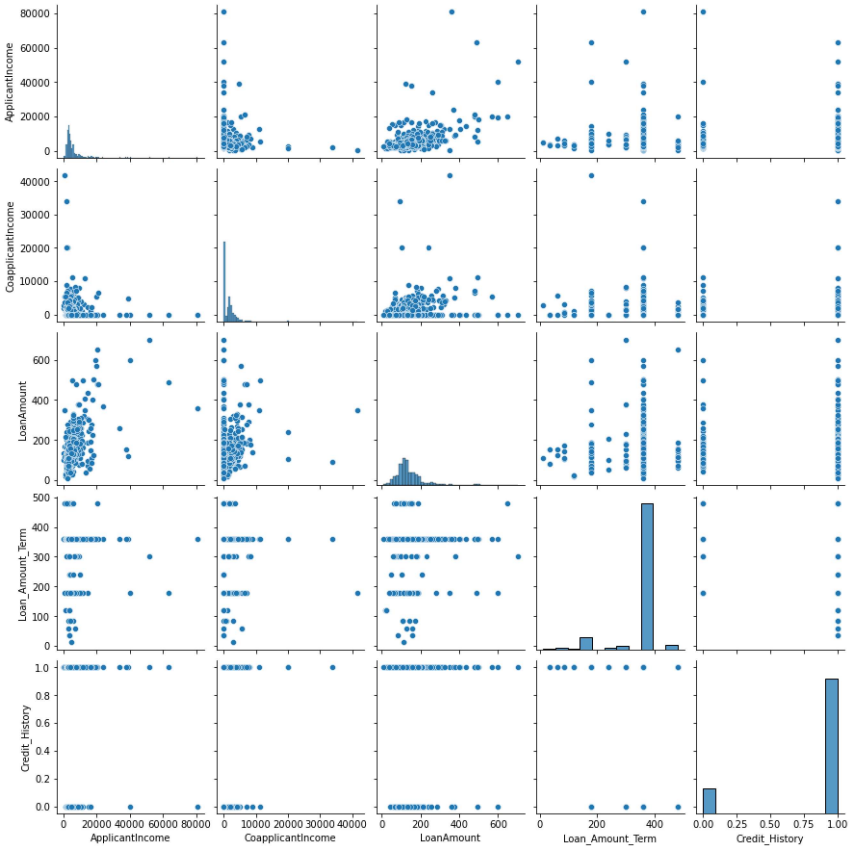




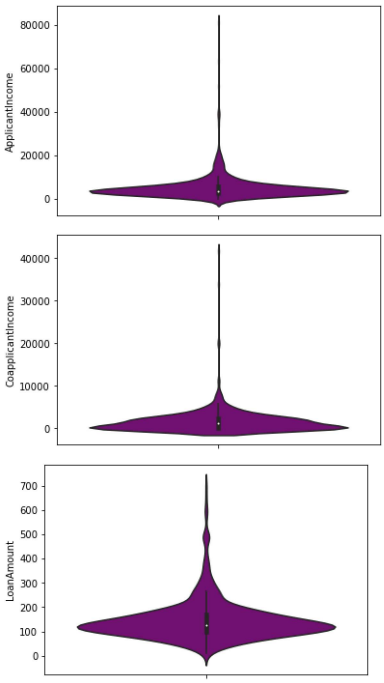
In [32]:
```
# We can clearly see that credit history and Loan Amount Term has only few values which are its categories.
# Applicant Income ,Coapplicant Income and Loan Amount have a wide distribution.
# They are right skewed , we need to normalise them.
```

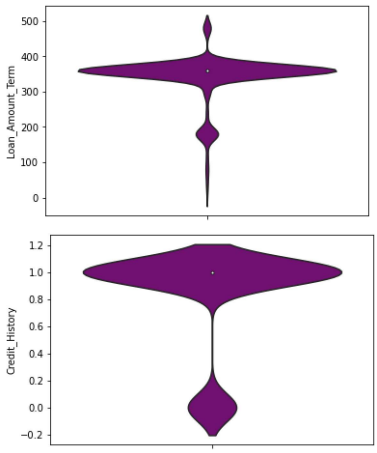
In [33]:
```
sns.pairplot(data)
```

Out[33]:
```
<seaborn.axisgrid.PairGrid at 0x1d478856250>
```



In [34]:
```
for col in numerical_columns:
    sns.violinplot(y = col,data = data,kde = True,color = "purple")
    plt.show()
```

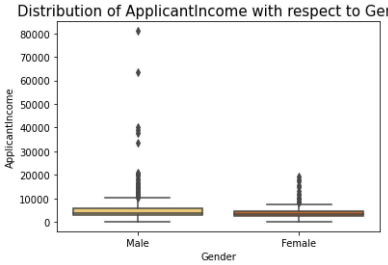




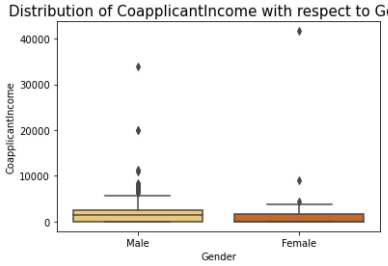
3.4 Numerical and Categorical

In [35]:
```
for cat in categorical_columns:
    if(cat!="Loan_ID"):
        for num in numerical_columns:
            sns.boxplot(data = data, x = cat  ,y = num, palette = "YlOrBr")
            plt.title("Distribution of {} with respect to {}".format(num,cat),fontsize = 15)
            plt.show()
        print("**********************************************************************")
        print("                                                                      ")
```

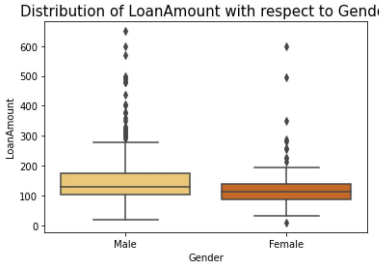
Distribution of ApplicantIncome with respect to Gender



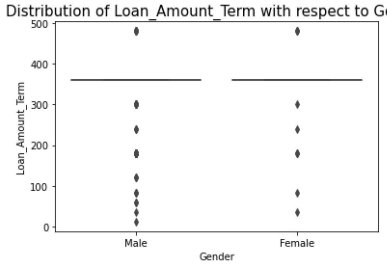
Distribution of CoapplicantIncome with respect to Gender



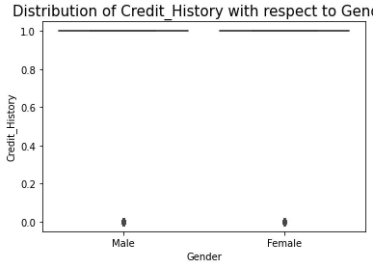
Distribution of LoanAmount with respect to Gender



Distribution of Loan_Amount_Term with respect to Gender

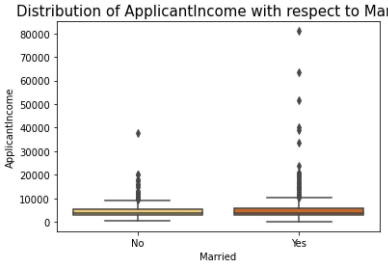


Distribution of Credit_History with respect to Gender

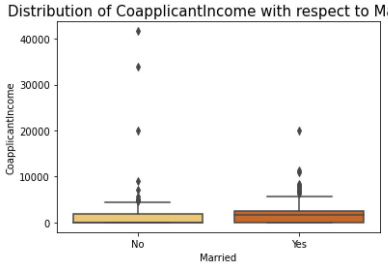


****

Distribution of ApplicantIncome with respect to Married

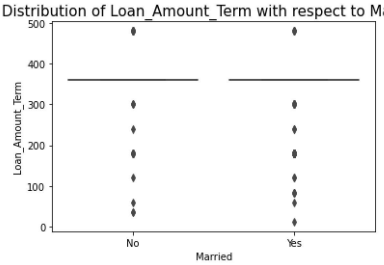


Distribution of CoapplicantIncome with respect to Married

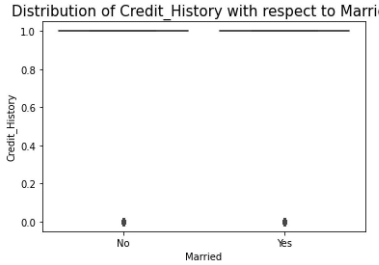


Distribution of LoanAmount with respect to Married

Distribution of Loan_Amount_Term with respect to Married

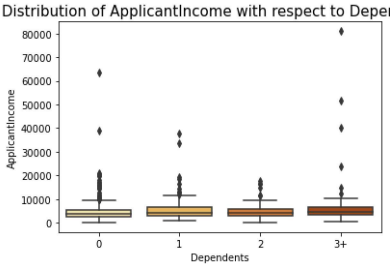


Distribution of Credit_History with respect to Married

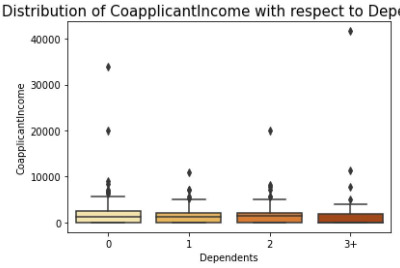


****

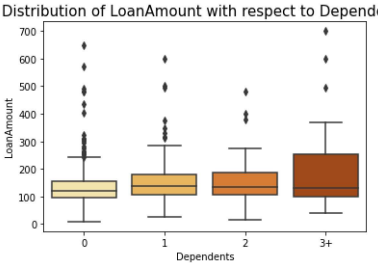
Distribution of ApplicantIncome with respect to Dependents



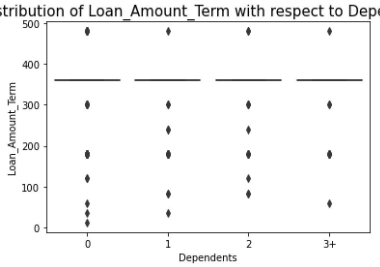
Distribution of CoapplicantIncome with respect to Dependents



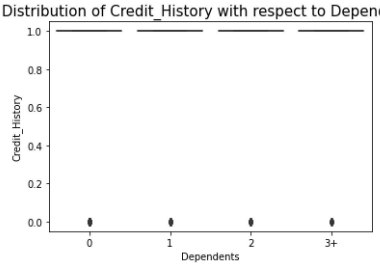
Distribution of LoanAmount with respect to Dependents



Distribution of Loan_Amount_Term with respect to Dependents

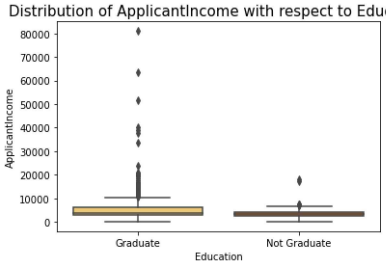


Distribution of Credit_History with respect to Dependents

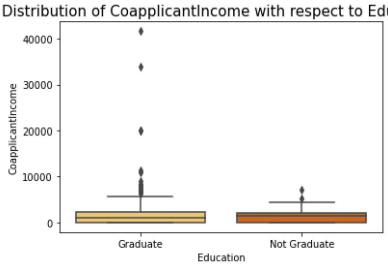


****

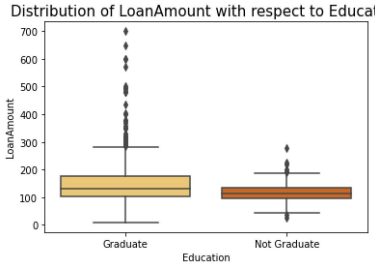
Distribution of ApplicantIncome with respect to Education



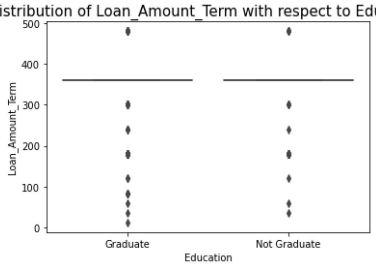
Distribution of CoapplicantIncome with respect to Education



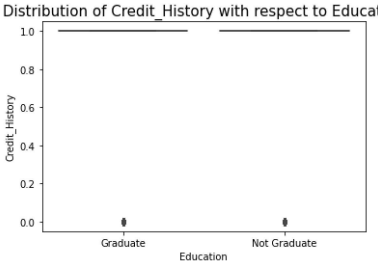
Distribution of LoanAmount with respect to Education



Distribution of Loan_Amount_Term with respect to Education

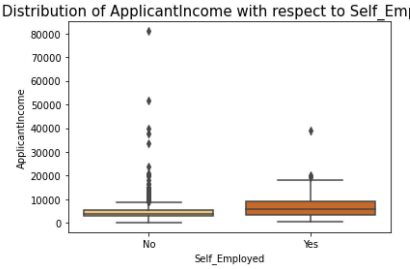


Distribution of Credit_History with respect to Education

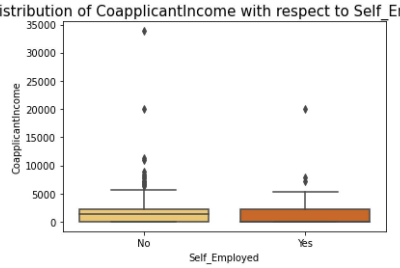


****

Distribution of ApplicantIncome with respect to Self_Employed



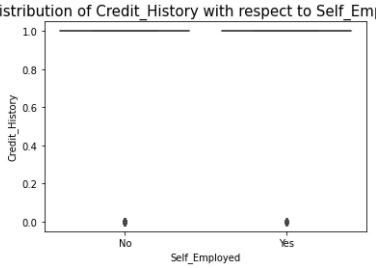
Distribution of CoapplicantIncome with respect to Self_Employed



Distribution of LoanAmount with respect to Self_Employed

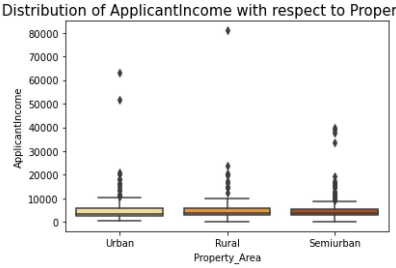
Distribution of Loan_Amount_Term with respect to Self_Employed

Distribution of Credit_History with respect to Self_Employed

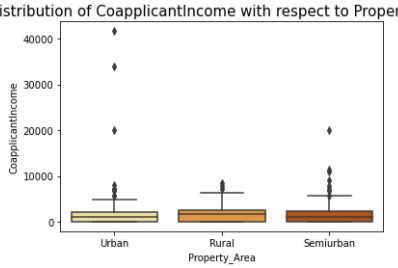


****

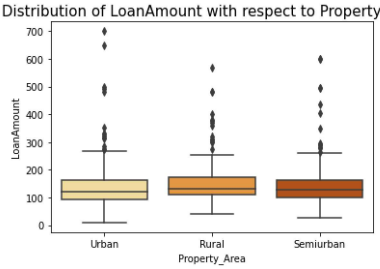
Distribution of ApplicantIncome with respect to Property_Area



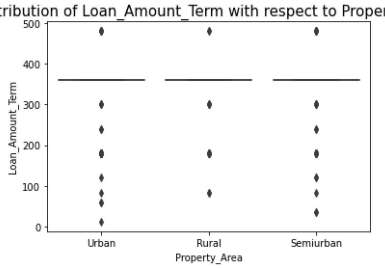
Distribution of CoapplicantIncome with respect to Property_Area



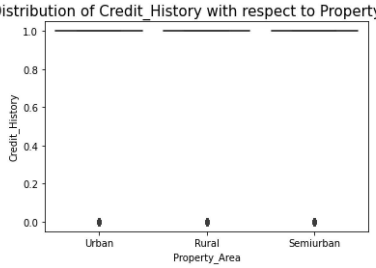
Distribution of LoanAmount with respect to Property_Area



Distribution of Loan_Amount_Term with respect to Property_Area

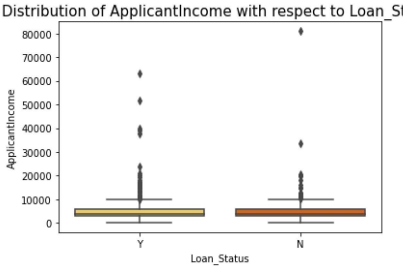


Distribution of Credit_History with respect to Property_Area

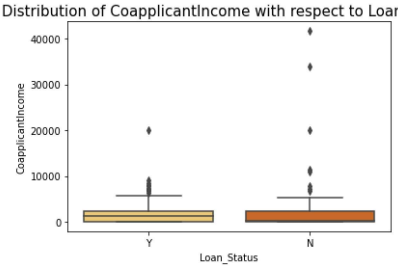


****

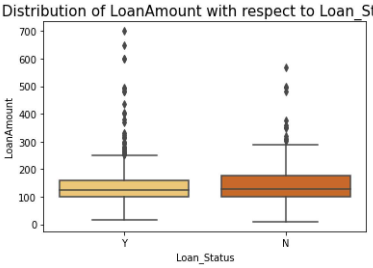
Distribution of ApplicantIncome with respect to Loan_Status

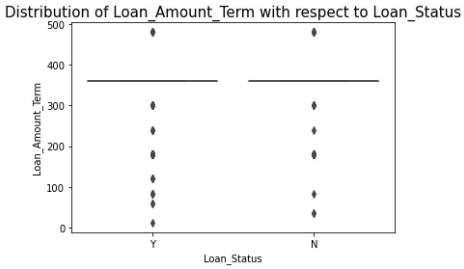


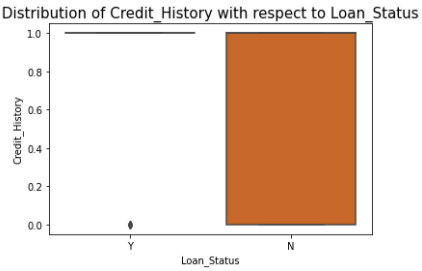
Distribution of CoapplicantIncome with respect to Loan_Status



Distribution of LoanAmount with respect to Loan_Status







****

In [36]:
```
# Male , Graduate and Married have higer Income, Coapplicant Income and Loan Amount.
# 3+ dependents have much more income.
```

3.5 Correlation

Correlation determines how much is one feature correlated to others.

In [37]:
```
plt.figure(figsize = (15,10))
sns.heatmap(data.corr(),annot = True,linewidths = 0.5,cmap = "Purples")
```

Out[37]:
```
<AxesSubplot:>
```

In [38]:
```
# Loan Amount and Applicant Income has 60% positive correlation.
```

3.6 Summary of data

- We will work in an area where men are dominant.
- More than half of the people are married
- Min Income =150
- Max Income =81k
- Mean Income =5350
- Data is right skewed for Applicant Income , Coapplicant Income and Loan Amount.
- There are outliers in the dataset.
- 85 percent of accepted applications have a positive credit history
- Rate of accepted applications 70 meaning that the data set is unbalanced

4. Data Preprocessing

4.1 Handling Missing values

Missing values are those values which which does not have data filled for that particular variable.

In [39]:
```
data.isnull().sum()
```

Out[39]:
```
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
```

4.1.1 Categorical Variables

In [40]:
```
data['Gender'].fillna(data['Gender'].mode()[0],inplace=True)
data['Married'].fillna(data['Married'].mode()[0],inplace=True)
data['Dependents'].fillna(data['Dependents'].mode()[0],inplace=True)
data['Self_Employed'].fillna(data['Self_Employed'].mode()[0],inplace=True)
data['Credit_History'].fillna(data['Credit_History'].mode()[0],inplace=True)
data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0],inplace=True)
```

4.1.2 Numrical Variables

In [41]:
```
data["LoanAmount"].fillna(data["LoanAmount"].mean(),inplace = True)
```

In [42]:
```
data.isnull().sum()
```

Out[42]:
```
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
```

4.2 Droping Unecessary Features

Features from which no information can be gained for the machine learning model are dropped to reduce training time.

In [43]:
```
data = data.drop(["Loan_ID"],axis = "columns")
```

4.3 Detecting and Removing Outliers

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.

In [44]:
```
data.size
```

Out[44]:
```
7368
```

In [45]:
```
def detect_outliers_iqr(data,outliers):
    data = sorted(data)
    q1 = np.percentile(data, 25)
    q3 = np.percentile(data, 75)
    # print(q1, q3)
    IQR = q3-q1
    lwr_bound = q1-(1.5*IQR)
    upr_bound = q3+(1.5*IQR)
    # print(lwr_bound, upr_bound)
    for i in data:
        if (i<lwr_bound or i>upr_bound):
            outliers.append(i)

ap_outliers = []
co_outliers = []
la_outliers = []

detect_outliers_iqr(data["ApplicantIncome"],ap_outliers)
detect_outliers_iqr(data["CoapplicantIncome"],co_outliers)
detect_outliers_iqr(data["LoanAmount"],la_outliers)
```

In [46]:
```
ap_outliers
```

Out[46]:
```
[10408,
 10416,
 10513,
 10750,
 10833,
 11000,
 11146,
 11250,
 11417,
 11500,
 11757,
 12000,
 12000,
 12500,
 12841,
 12876,
 13262,
 13650,
 14583,
 14583,
 14683,
 14866,
 14880,
 14999,
 15000,
 15759,
 16120,
 16250,
 16525,
 16666,
 16667,
 16692,
 17263,
 17500,
 18165,
 18333,
 19484,
 19730,
 20166,
 20233,
 20667,
 20833,
 23803,
 33846,
 37719,
 39147,
 39999,
 51763,
 63337,
 81000]
```

In [47]:
```
co_outliers
```

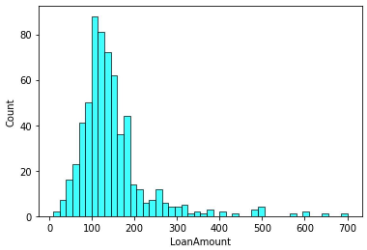
Out[47]:
```
[6250.0,
 6666.0,
 6667.0,
 7101.0,
 7166.0,
 7210.0,
 7250.0,
 7750.0,
 7873.0,
 8106.0,
 8333.0,
 8980.0,
 10968.0,
 11300.0,
 20000.0,
 20000.0,
 33837.0,
 41667.0]
```

In [48]:
```
la_outliers
```

Out[48]:
```
[265.0,
 267.0,
 275.0,
 275.0,
 275.0,
 279.0,
 280.0,
 286.0,
 290.0,
 292.0,
 296.0,
 300.0,
 304.0,
 308.0,
 311.0,
 312.0,
 315.0,
 320.0,
 324.0,
 330.0,
 349.0,
 350.0,
 360.0,
 370.0,
 376.0,
 380.0,
 400.0,
 405.0,
 436.0,
 480.0,
 480.0,
 480.0,
 490.0,
 495.0,
 496.0,
 500.0,
 570.0,
 600.0,
 600.0,
 650.0,
 700.0]
```

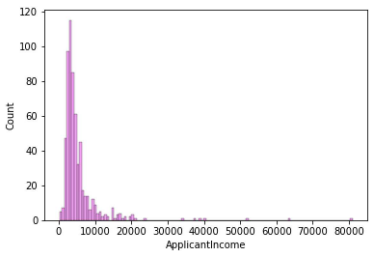
In [49]:
```
sns.histplot(x = "LoanAmount",data = data,color = "cyan")
```

Out[49]:
```
<AxesSubplot:xlabel='LoanAmount', ylabel='Count'>
```



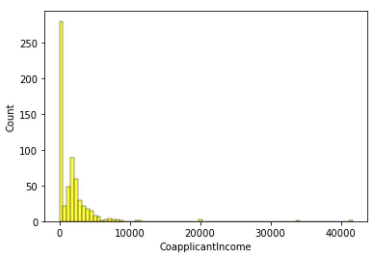
In [50]:
```
sns.histplot(x = "ApplicantIncome",data = data, color = "violet")
```

Out[50]:
```
<AxesSubplot:xlabel='ApplicantIncome', ylabel='Count'>
```



In [51]:
```
sns.histplot(x = "CoapplicantIncome",data =data,color = "yellow")
```

Out[51]:
```
<AxesSubplot:xlabel='CoapplicantIncome', ylabel='Count'>
```



In [52]:
```
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
```

```
IQR = Q3 - Q1

data = data[~((data < (Q1 - 1.5 * IQR)) |(data > (Q3 + 1.5 * IQR))).any(axis=1)]
```

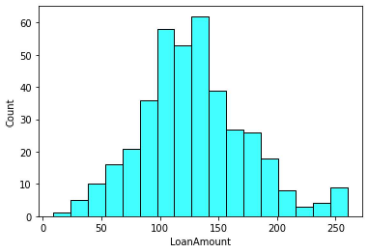
```
C:\Users\aksha\AppData\Local\Temp/ipykernel_15716/1705376073.py:5: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons is deprecated and will raise ValueError in a future ver
sion.  Do `left, right = left.align(right, axis=1, copy=False)` before e.g. `left == right`
  data = data[~((data < (Q1 - 1.5 * IQR)) |(data > (Q3 + 1.5 * IQR))).any(axis=1)]
```

In [53]:
```
data.size
```

Out[53]:
```
4752
```

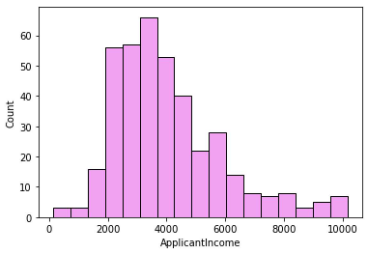
In [54]:
```
sns.histplot(data = data,x = "LoanAmount",color = "cyan")
```

Out[54]:
```
<AxesSubplot:xlabel='LoanAmount', ylabel='Count'>
```



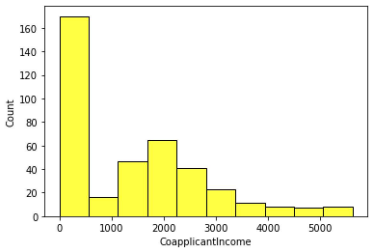
In [55]:
```
sns.histplot(data = data,x = "ApplicantIncome",color = "violet")
```

Out[55]:
```
<AxesSubplot:xlabel='ApplicantIncome', ylabel='Count'>
```



In [56]:
```
sns.histplot(data = data,x = "CoapplicantIncome",color = "yellow")
```

Out[56]:
```
<AxesSubplot:xlabel='CoapplicantIncome', ylabel='Count'>
```



In [57]:
```
data.ApplicantIncome = np.sqrt(data.ApplicantIncome)
data.CoapplicantIncome = np.sqrt(data.CoapplicantIncome)
data.LoanAmount = np.sqrt(data.LoanAmount)
```

4.4 One-Hot Encoding

One Hot Encoding is used to treat categorical variables by making new column for each category in column so to improve predictions as well as classification accuracy of a model.

In [58]:
```
data = pd.get_dummies(data)

data = data.drop(['Gender_Female', 'Married_No', 'Education_Not Graduate',
                  'Self_Employed_No', 'Loan_Status_N'], axis = 1)

latcolumns = {"Gender_Male":"Gender","Married_Yes":"Married","Education_Graduare":"Education","Self_Employed_Yes":"Self_Employed","Loan_Status_Y":"Loan_Status"}

data.rename(columns = latcolumns,inplace=True)
```

In [59]:
```
data
```

Out[59]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender	Married	Dependents_0	Dependents_1	Dependents_2	Dependents_3+	Education_Graduate	Self_Employed	Property_Area_Rura
0	76.478755	0.000000	12.100089	360.0	1.0	1	0	1	0	0	0	1	0	
1	67.697858	38.832976	11.313708	360.0	1.0	1	1	0	1	0	0	1	0	
2	54.772256	0.000000	8.124038	360.0	1.0	1	1	1	0	0	0	1	1	
3	50.823223	48.559242	10.954451	360.0	1.0	1	1	1	0	0	0	0	0	
4	77.459667	0.000000	11.874342	360.0	1.0	1	0	1	0	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
607	63.142696	37.563280	12.529964	360.0	1.0	1	1	0	0	1	0	0	0	
608	56.850682	44.158804	10.392305	360.0	1.0	1	1	1	0	0	0	1	0	
609	53.851648	0.000000	8.426150	360.0	1.0	0	0	1	0	0	0	1	0	
611	89.844310	15.491933	15.905974	360.0	1.0	1	1	0	1	0	0	1	0	
612	87.080423	0.000000	13.674794	360.0	1.0	1	1	0	0	1	0	1	0	

396 rows × 17 columns

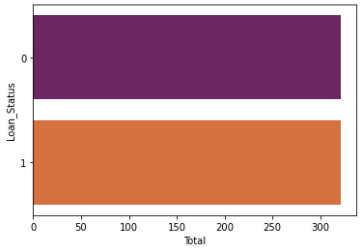
In [60]:
```
X = data.drop(["Loan_Status"],axis = 1)
y = data["Loan_Status"]
```

4.5 SMOTE

Smote (synthetic minority oversampling technique) is used when dataset is imbalanced and it balance class distribution by randomly increasing minority class examples by replicating them. It introduces new entries in existing data of minority class.

In [61]:
```
X,y = SMOTE().fit_resample(X,y)
```

In [62]:
```
# new distribution after resampling
sns.countplot(y =y,data = data, palette = "inferno" )
plt.ylabel("Loan_Status")
plt.xlabel("Total")
plt.show()
```



4.6 Scaling the data

Scaling is done to make sure that data is not spread out a lot, so model can learn from it easily and efficiently.

In [63]:
```
scaling = MinMaxScaler()
X = scaling.fit_transform(X)
```

4.7 Data Splitting

Splitting data to test the models.

In [64]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.2,random_state = 42)
```

In [65]:
```
print('X_train :',X_train.shape)
print('X_test :',X_test.shape)
print('y_train :',y_train.shape)
print('y_test :',y_test.shape)
```

```
X_train : (515, 16)
X_test : (129, 16)
y_train : (515,)
y_test : (129,)
```

In [66]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 396 entries, 0 to 612
Data columns (total 17 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ApplicantIncome          396 non-null    float64
 1   CoapplicantIncome        396 non-null    float64
 2   LoanAmount               396 non-null    float64
 3   Loan_Amount_Term         396 non-null    float64
 4   Credit_History           396 non-null    float64
 5   Gender                   396 non-null    uint8  
 6   Married                  396 non-null    uint8  
 7   Dependents_0             396 non-null    uint8  
 8   Dependents_1             396 non-null    uint8  
 9   Dependents_2             396 non-null    uint8  
 10  Dependents_3+            396 non-null    uint8  
 11  Education_Graduate       396 non-null    uint8  
 12  Self_Employed            396 non-null    uint8  
 13  Property_Area_Rural      396 non-null    uint8  
 14  Property_Area_Semiurban  396 non-null    uint8  
 15  Property_Area_Urban      396 non-null    uint8  
 16  Loan_Status              396 non-null    uint8  
dtypes: float64(5), uint8(12)
memory usage: 39.4 KB
```

5. Models

5.1 Logistic Regression

Logistic regression is a simple and more efficient method for binary and linear classification problems. It is a classification model, which is very easy to realize and achieves very good performance with linearly separable classes.

In [67]:
```
LogisticClassifier = LogisticRegression(solver="liblinear",max_iter = 500,random_state = 42)
LogisticClassifier.fit(X_train,y_train)
LogisticClassifier
```

Out[67]:
```
LogisticRegression(max_iter=500, random_state=42, solver='liblinear')
```

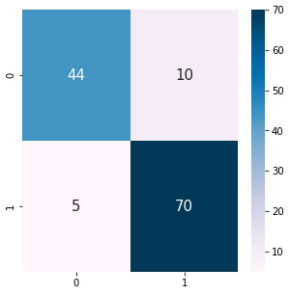
In [68]:
```
y_pred_lr = LogisticClassifier.predict(X_test)
```

In [69]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_pred_lr,y_test)
```

Out[69]:
```
0.8837209302325582
```

In [70]:
```
cm_lr = confusion_matrix(y_test,y_pred_lr)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm_lr, annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[70]:
```
<AxesSubplot:>
```



In [71]:
```
print("Training Accuracy :", LogisticClassifier.score(X_train, y_train))

print("Testing Accuracy :", LogisticClassifier.score(X_test, y_test))
```

```
Training Accuracy : 0.8077669902912621
Testing Accuracy : 0.8837209302325582
```

In [72]:
```
cross_val_score(LogisticClassifier,X_test,y_test,cv = 20).mean()
```

Out[72]:
```
0.8678571428571429
```

5.2 K-Nearest Neighbour (KNN)

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

In [73]:
```
knn = KNeighborsClassifier()
knn_model = knn.fit(X_train, y_train)
knn_model
```

Out[73]:
```
KNeighborsClassifier
KNeighborsClassifier()
```

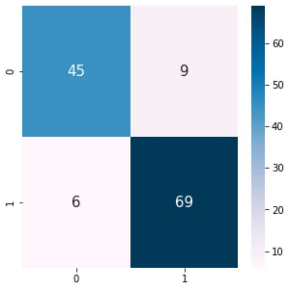
In [74]:
```
y_pred_knn = knn_model.predict(X_test)
```

In [75]:
```
accuracy_score(y_pred_knn,y_test)
```

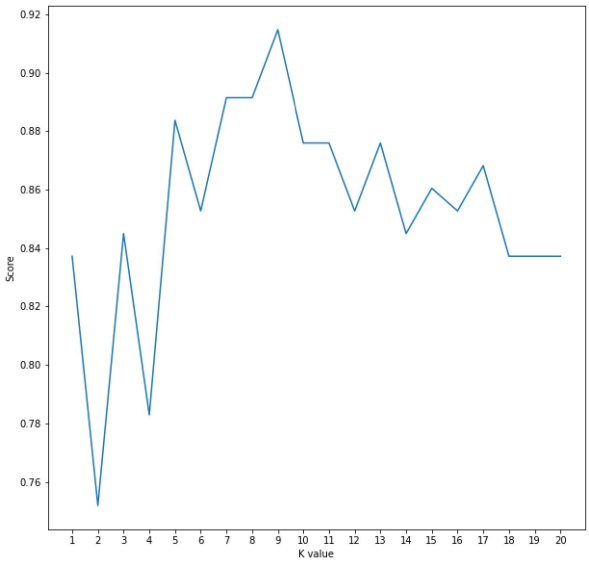
Out[75]:
```
0.8837209302325582
```

In [76]:
```
cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm_knn, annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[76]:
```
<AxesSubplot:>
```



In [77]:
```
# checking the best model for knn
knn_score = []
for i in range(1,21):
    knnclassifier = KNeighborsClassifier(n_neighbors = i)
    knnclassifier.fit(X_train,y_train)
    knn_score.append(knnclassifier.score(X_test,y_test))
plt.figure(figsize = (10,10))
plt.plot(range(1,21), knn_score)
plt.xticks(np.arange(1,21,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
KNAcc = max(knn_score)
print("KNN best accuracy: {:.2f}%".format(KNAcc*100))
```




```
KNN best accuracy: 91.47%
```

5.3 Naive Bayes

Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering, recommendation systems etc. They are fast and easy to implement but their biggest disadvantage is that the requirement of predictors to be independent.

In [78]:
```
GaussianNB = GaussianNB()
GaussianNB.fit(X_train,y_train)
```

Out[78]:
```
GaussianNB()
```

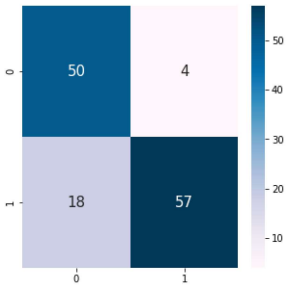
In [79]:
```
y_pred_NB = GaussianNB.predict(X_test)
```

In [80]:
```
accuracy_score(y_pred_NB,y_test)
```

Out[80]:
```
0.8294573643410853
```

In [81]:
```
cm_NB = confusion_matrix(y_test, y_pred_NB)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm_NB, annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[81]:
```
<AxesSubplot:>
```



5.4 Decision Tree

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

In [82]:
```
dt = DecisionTreeClassifier()
dt.fit(X_train,y_train)
dt
```

Out[82]:
```
DecisionTreeClassifier()
```

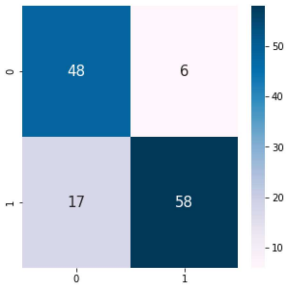
In [83]:
```
y_pred_dt = dt.predict(X_test)
```

In [84]:
```
accuracy_score(y_test,y_pred_dt)
```

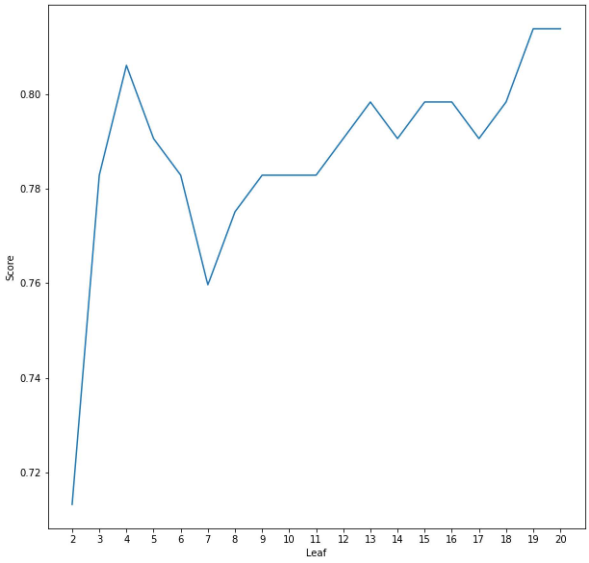
Out[84]:
```
0.8217054263565892
```

In [85]:
```
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm_dt, annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[85]:
```
<AxesSubplot:>
```



In [86]:
```
dt_score = []
for i in range(2,21):
    DtClassifier = DecisionTreeClassifier(max_leaf_nodes = i)
    DtClassifier.fit(X_train,y_train)
    dt_score.append(DtClassifier.score(X_test,y_test))

plt.figure(figsize = (10,10))
plt.plot(range(2,21), dt_score)
plt.xticks(np.arange(2,21,1))
plt.xlabel("Leaf")
plt.ylabel("Score")
plt.show()
DTAcc = max(dt_score)
print("Decision Tree Accuracy: {:.2f}%".format(DTAcc*100))
```




```
Decision Tree Accuracy: 81.40%
```

5.5 Random Forest

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

In [87]:
```
rf = RandomForestClassifier()
rf.fit(X_train,y_train)
```

Out[87]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

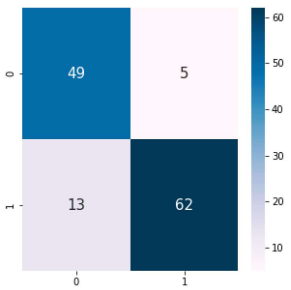
In [88]:
```
y_pred_rf = rf.predict(X_test)
```

In [89]:
```
accuracy_score(y_pred_rf,y_test)
```

Out[89]:
```
0.8604651162790697
```

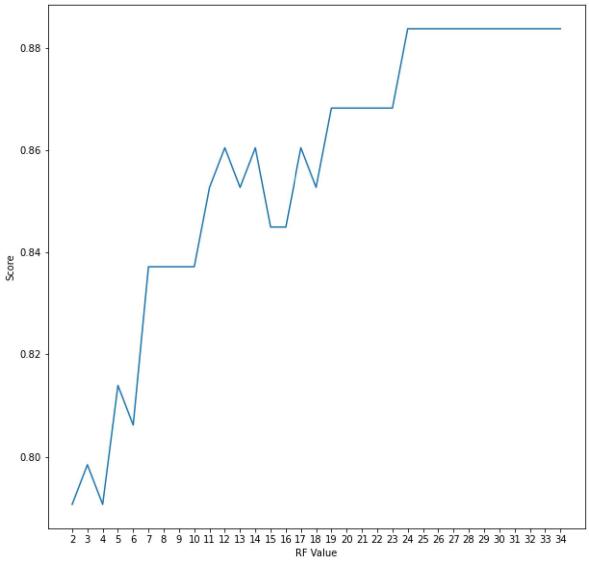
In [90]:
```
cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm_rf, annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[90]:
```
<AxesSubplot:>
```



In [91]:
```
rf_score = []
for i in range(2,35):
    RFclassifier = RandomForestClassifier(n_estimators = 1000, random_state = 1, max_leaf_nodes=i)
    RFclassifier.fit(X_train, y_train)
    rf_score.append(RFclassifier.score(X_test, y_test))

plt.figure(figsize = (10,10))
plt.plot(range(2,35), rf_score)
plt.xticks(np.arange(2,35,1))
```

```
plt.xlabel("RF Value")
plt.ylabel("Score")
plt.show()
RFAcc = max(rf_score)
print("Random Forest Accuracy:  {:.2f}%".format(RFAcc*100))
```




```
Random Forest Accuracy:  88.37%
```

In [92]:

```
finmodel = GridSearchCV(RandomForestClassifier(criterion = "gini"),{
            "max_depth": [8,10],
            "max_features": [5,8],
            "n_estimators": [500,1000,1500],
            "min_samples_split": [2,5],
            "max_leaf_nodes" : [15,20,25,30]
},cv = 5,n_jobs = -1)

finmodel.fit(X_train,y_train)
finmodel.cv_results_
```

Out[92]:

```
{'mean_fit_time': array([ 3.1686038 ,  6.76659784, 10.00398746,  3.32368073,  6.49900689,
         9.57891254,  3.19766421,  6.77770548,  9.81963573,  3.24781413,
         6.4594882 , 10.00338559,  3.23802681,  6.65669708, 10.07216387,
         3.42852573,  6.49201365, 10.21721921,  3.30375929,  6.93701224,
        10.26177087,  3.25569782,  6.9697732 ,  9.81053071,  3.5679832 ,
         7.26297593, 10.75483527,  3.56033421,  7.46102057, 10.60391426,
         3.6036283 ,  7.1830564 , 11.00590053,  3.64481425,  7.1862967 ,
        11.26714158,  3.60152292,  7.22413836, 11.04968128,  3.8302135 ,
         7.70964537, 11.30870738,  3.91531935,  7.24321837, 11.52590795,
         3.83733912,  7.3542767 , 11.19128079,  3.34876471,  6.56358981,
        10.1204155 ,  3.34901175,  6.65293689,  9.62933702,  3.19403186,
         6.84059205,  9.73772993,  3.34549184,  6.68808522, 10.03521457,
         3.42371087,  6.46546125, 10.01104808,  3.40156293,  6.82672114,
        10.55020323,  3.27541571,  6.65498834,  9.79572339,  3.32257724,
         6.83679852,  9.65460777,  3.50680447,  7.3725626 , 10.63557448,
         3.54389749,  7.59528756, 11.11928773,  3.84560318,  7.17686911,
        11.02756038,  3.60097709,  7.1011529 , 11.02522373,  3.62291899,
         7.2252708 , 11.80162597,  4.09642334,  9.12663202, 13.42664986,
         4.79966416,  8.87081089, 11.96372375,  3.74609661,  7.1916667 ,
        10.15581431]),
 'std_fit_time': array([0.02791287, 0.0983946 , 0.2470992 , 0.14896676, 0.16084425,
        0.16450034, 0.05582902, 0.2151227 , 0.33427764, 0.07757426,
        0.24594746, 0.50127783, 0.11994983, 0.16159243, 0.45707007,
        0.17948854, 0.10762051, 0.61165289, 0.1591313 , 0.30572206,
        0.46193804, 0.0573136 , 0.46597126, 0.29966009, 0.15562432,
        0.30997875, 0.31730765, 0.04459973, 0.15990601, 0.12432098,
        0.08343829, 0.14393807, 0.13450324, 0.08767713, 0.1503358 ,
        0.37753031, 0.04890475, 0.13574659, 0.22151916, 0.15185253,
        0.20528414, 0.39093367, 0.1904395 , 0.12786534, 0.42573374,
        0.31556959, 0.19469309, 0.39603502, 0.19614335, 0.24916826,
        0.40760156, 0.14584211, 0.32493564, 0.16698504, 0.07202024,
        0.18287613, 0.26512585, 0.1459865 , 0.1877679 , 0.3936717 ,
        0.19700259, 0.11420476, 0.29484994, 0.15821692, 0.30074757,
        0.48323647, 0.03975582, 0.11344075, 0.25929864, 0.16333942,
        0.30105187, 0.12172606, 0.02486082, 0.25177868, 0.29278245,
        0.13855357, 0.34583414, 0.186525  , 0.13538493, 0.16640051,
        0.262296  , 0.07669905, 0.12385965, 0.33754334, 0.0850558 ,
        0.12077114, 0.90429744, 0.44739891, 0.26713831, 0.92883233,
        0.24967458, 0.39898574, 0.89331211, 0.20905851, 0.14293008,
        0.53672581]),
 'mean_score_time': array([0.20117807, 0.37998724, 0.60818353, 0.20533485, 0.37214394,
        0.59230514, 0.2008584 , 0.3949461 , 0.59456491, 0.19936819,
        0.38735113, 0.5802711 , 0.18709884, 0.41022301, 0.58908701,
        0.20188384, 0.38680577, 0.61951222, 0.20093398, 0.39540386,
        0.66007309, 0.21886716, 0.42470412, 0.60317569, 0.20095119,
        0.37738471, 0.57356067, 0.18804188, 0.40805669, 0.56112146,
        0.18554668, 0.40121541, 0.57426925, 0.1923903 , 0.41492281,
        0.56809916, 0.19267974, 0.40727925, 0.6488986 , 0.20026197,
        0.435674  , 0.57441511, 0.21159453, 0.38532968, 0.5856874 ,
        0.19620247, 0.38871131, 0.59746566, 0.20291624, 0.37811093,
        0.61766772, 0.2100122 , 0.37967038, 0.60581183, 0.18927498,
        0.40225139, 0.57220187, 0.18845005, 0.42206144, 0.58784995,
        0.19352293, 0.40231285, 0.61940141, 0.20949039, 0.39905443,
        0.61531072, 0.19793591, 0.37990041, 0.59652181, 0.20408216,
        0.39731441, 0.58319478, 0.19674501, 0.38725209, 0.59332676,
        0.18754716, 0.42613969, 0.60310278, 0.19250264, 0.4032794 ,
        0.57665286, 0.18472223, 0.39119272, 0.57293167, 0.18783464,
        0.39233923, 0.73486056, 0.19676785, 0.50142446, 0.66184778,
        0.21267481, 0.4650825 , 0.58284202, 0.18152447, 0.38509016,
        0.46358261]),
 'std_score_time': array([0.00594995, 0.01470617, 0.03125186, 0.00573664, 0.00253233,
        0.02400176, 0.00710557, 0.01317585, 0.05582327, 0.01324   ,
        0.01202911, 0.01769877, 0.00378114, 0.04495315, 0.05103165,
        0.01513041, 0.00615784, 0.03894148, 0.01777783, 0.02859879,
        0.04799284, 0.03388286, 0.03913379, 0.05826286, 0.0217174 ,
        0.02215067, 0.02445451, 0.01012328, 0.02144114, 0.01841908,
        0.00415244, 0.02289237, 0.01070287, 0.01109224, 0.04863726,
        0.01253891, 0.00587142, 0.0361521 , 0.08526501, 0.00563579,
        0.05015494, 0.01004287, 0.0284573 , 0.0297327 , 0.03179192,
        0.01201444, 0.02058599, 0.04304654, 0.02017022, 0.00799935,
        0.04522185, 0.03462715, 0.00790753, 0.02543624, 0.00470217,
        0.01000935, 0.01514178, 0.00040565, 0.03963269, 0.03242915,
        0.0066714 , 0.02050217, 0.08488661, 0.01386964, 0.01221183,
        0.02773564, 0.01350916, 0.0139384 , 0.04294027, 0.01444406,
        0.04495005, 0.03450622, 0.00701946, 0.02344716, 0.04561064,
        0.00381569, 0.04079725, 0.02752491, 0.008819  , 0.03185529,
        0.04963204, 0.00395582, 0.0220512 , 0.02056863, 0.00618851,
```

```
       0.01949236, 0.09734969, 0.01067369, 0.08617019, 0.01890927,
       0.021879  , 0.06774722, 0.06164823, 0.00511064, 0.00859087,
       0.02123862]),
 'param_max_depth': masked_array(data=[8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                   8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                   8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 10, 10, 10, 10, 10,
                   10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
                   10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
                   10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
                   10],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_max_features': masked_array(data=[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
                   5, 5, 5, 5, 5, 5, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                   8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 5, 5, 5, 5, 5, 5,
                   5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
                   8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,
                   8, 8, 8, 8, 8, 8],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_max_leaf_nodes': masked_array(data=[15, 15, 15, 15, 15, 15, 20, 20, 20, 20, 20, 20, 25, 25,
                   25, 25, 25, 25, 30, 30, 30, 30, 30, 30, 15, 15, 15, 15,
                   15, 15, 20, 20, 20, 20, 20, 20, 25, 25, 25, 25, 25, 25,
                   30, 30, 30, 30, 30, 30, 15, 15, 15, 15, 15, 15, 20, 20,
                   20, 20, 20, 20, 25, 25, 25, 25, 25, 25, 30, 30, 30, 30,
                   30, 30, 15, 15, 15, 15, 15, 15, 20, 20, 20, 20, 20, 20,
                   25, 25, 25, 25, 25, 25, 30, 30, 30, 30, 30, 30],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_min_samples_split': masked_array(data=[2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5,
                   2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5,
                   2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5,
                   2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5,
                   2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5, 2, 2, 2, 5, 5, 5,
                   2, 2, 2, 5, 5, 5],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'param_n_estimators': masked_array(data=[500, 1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500, 500, 1000, 1500, 500,
                   1000, 1500, 500, 1000, 1500],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False,
                   False, False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object),
 'params': [{'max_depth': 8,
   'max_features': 5,
   'max_leaf_nodes': 15,
   'min_samples_split': 2,
   'n_estimators': 500},
  {'max_depth': 8,
   'max_features': 5,
   'max_leaf_nodes': 15,
   'min_samples_split': 2,
   'n_estimators': 1000},
  {'max_depth': 8,
   'max_features': 5,
   'max_leaf_nodes': 15,
   'min_samples_split': 2,
   'n_estimators': 1500},
  {'max_depth': 8,
   'max_features': 5,
   'max_leaf_nodes': 15,
   'min_samples_split': 5,
   'n_estimators': 500},
  {'max_depth': 8,
   'max_features': 5,
   'max_leaf_nodes': 15,
   'min_samples_split': 5,
```

```
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 20,
```

```
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 8,
 'max_features': 8,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
```

```
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 25,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 5,
 'max_leaf_nodes': 30,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 15,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 2,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1000},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 20,
 'min_samples_split': 5,
 'n_estimators': 1500},
{'max_depth': 10,
 'max_features': 8,
 'max_leaf_nodes': 25,
 'min_samples_split': 2,
 'n_estimators': 500},
{'max_depth': 10,
```

```
  'max_features': 8,
  'max_leaf_nodes': 25,
  'min_samples_split': 2,
  'n_estimators': 1000},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 25,
  'min_samples_split': 2,
  'n_estimators': 1500},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 25,
  'min_samples_split': 5,
  'n_estimators': 500},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 25,
  'min_samples_split': 5,
  'n_estimators': 1000},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 25,
  'min_samples_split': 5,
  'n_estimators': 1500},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 30,
  'min_samples_split': 2,
  'n_estimators': 500},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 30,
  'min_samples_split': 2,
  'n_estimators': 1000},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 30,
  'min_samples_split': 2,
  'n_estimators': 1500},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 30,
  'min_samples_split': 5,
  'n_estimators': 500},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 30,
  'min_samples_split': 5,
  'n_estimators': 1000},
 {'max_depth': 10,
  'max_features': 8,
  'max_leaf_nodes': 30,
  'min_samples_split': 5,
  'n_estimators': 1500}],
'split0_test_score': array([0.7961165 , 0.80582524, 0.80582524, 0.81553398, 0.80582524,
       0.80582524, 0.81553398, 0.81553398, 0.7961165 , 0.81553398,
       0.81553398, 0.82524272, 0.80582524, 0.82524272, 0.83495146,
       0.82524272, 0.82524272, 0.83495146, 0.83495146, 0.83495146,
       0.82524272, 0.82524272, 0.82524272, 0.83495146, 0.80582524,
       0.80582524, 0.80582524, 0.7961165 , 0.80582524, 0.80582524,
       0.81553398, 0.82524272, 0.81553398, 0.82524272, 0.81553398,
       0.82524272, 0.84466019, 0.83495146, 0.83495146, 0.82524272,
       0.82524272, 0.83495146, 0.84466019, 0.83495146, 0.83495146,
       0.84466019, 0.83495146, 0.83495146, 0.7961165 , 0.80582524,
       0.81553398, 0.82524272, 0.80582524, 0.80582524, 0.82524272,
       0.82524272, 0.82524272, 0.82524272, 0.82524272, 0.82524272,
       0.83495146, 0.81553398, 0.82524272, 0.82524272, 0.83495146,
       0.83495146, 0.83495146, 0.82524272, 0.82524272, 0.83495146,
       0.83495146, 0.83495146, 0.80582524, 0.80582524, 0.7961165 ,
       0.80582524, 0.80582524, 0.81553398, 0.82524272, 0.81553398,
       0.82524272, 0.81553398, 0.81553398, 0.81553398, 0.82524272,
       0.82524272, 0.83495146, 0.82524272, 0.82524272, 0.82524272,
       0.84466019, 0.84466019, 0.84466019, 0.83495146, 0.82524272,
       0.84466019]),
'split1_test_score': array([0.76699029, 0.76699029, 0.77669903, 0.77669903, 0.76699029,
       0.77669903, 0.76699029, 0.77669903, 0.77669903, 0.76699029,
       0.76699029, 0.76699029, 0.77669903, 0.77669903, 0.78640777,
       0.7961165 , 0.78640777, 0.77669903, 0.7961165 , 0.7961165 ,
       0.78640777, 0.78640777, 0.80582524, 0.7961165 , 0.77669903,
       0.78640777, 0.78640777, 0.7961165 , 0.7961165 , 0.78640777,
       0.78640777, 0.78640777, 0.78640777, 0.77669903, 0.77669903,
       0.78640777, 0.77669903, 0.80582524, 0.80582524, 0.7961165 ,
       0.80582524, 0.7961165 , 0.80582524, 0.80582524, 0.80582524,
       0.81553398, 0.80582524, 0.80582524, 0.77669903, 0.76699029,
       0.77669903, 0.77669903, 0.76699029, 0.77669903, 0.77669903,
       0.76699029, 0.77669903, 0.77669903, 0.77669903, 0.77669903,
       0.78640777, 0.7961165 , 0.7961165 , 0.80582524, 0.80582524,
       0.7961165 , 0.81553398, 0.80582524, 0.78640777, 0.7961165 ,
       0.7961165 , 0.80582524, 0.7961165 , 0.77669903, 0.77669903,
       0.78640777, 0.77669903, 0.77669903, 0.78640777, 0.7961165 ,
       0.78640777, 0.78640777, 0.78640777, 0.78640777, 0.7961165 ,
       0.80582524, 0.80582524, 0.80582524, 0.80582524, 0.7961165 ,
       0.80582524, 0.80582524, 0.80582524, 0.80582524, 0.81553398,
       0.80582524]),
'split2_test_score': array([0.7961165 , 0.7961165 , 0.80582524, 0.78640777, 0.7961165 ,
       0.7961165 , 0.78640777, 0.7961165 , 0.81553398, 0.80582524,
       0.7961165 , 0.7961165 , 0.78640777, 0.80582524, 0.80582524,
       0.7961165 , 0.80582524, 0.80582524, 0.82524272, 0.80582524,
       0.80582524, 0.78640777, 0.80582524, 0.7961165 , 0.80582524,
       0.77669903, 0.78640777, 0.80582524, 0.7961165 , 0.78640777,
       0.7961165 , 0.80582524, 0.7961165 , 0.7961165 , 0.78640777,
       0.80582524, 0.80582524, 0.7961165 , 0.81553398, 0.7961165 ,
       0.80582524, 0.7961165 , 0.81553398, 0.82524272, 0.7961165 ,
       0.81553398, 0.81553398, 0.81553398, 0.77669903, 0.80582524,
       0.80582524, 0.77669903, 0.80582524, 0.78640777, 0.7961165 ,
       0.80582524, 0.7961165 , 0.80582524, 0.80582524, 0.81553398,
       0.80582524, 0.80582524, 0.80582524, 0.82524272, 0.80582524,
       0.80582524, 0.78640777, 0.80582524, 0.7961165 , 0.80582524,
       0.80582524, 0.81553398, 0.81553398, 0.81553398, 0.80582524,
       0.80582524, 0.80582524, 0.78640777, 0.78640777, 0.7961165 ,
       0.7961165 , 0.7961165 , 0.7961165 , 0.80582524, 0.81553398,
       0.7961165 , 0.78640777, 0.78640777, 0.7961165 , 0.7961165 ,
       0.80582524, 0.80582524, 0.80582524, 0.80582524, 0.80582524,
       0.7961165 , 0.81553398, 0.81553398, 0.83495146, 0.82524272,
       0.82524272]),
'split3_test_score': array([0.85436893, 0.86407767, 0.86407767, 0.88349515, 0.86407767,
       0.86407767, 0.89320388, 0.88349515, 0.88349515, 0.87378641,
       0.87378641, 0.86407767, 0.88349515, 0.88349515, 0.89320388,
       0.88349515, 0.88349515, 0.88349515, 0.89320388, 0.89320388,
       0.90291262, 0.90291262, 0.90291262, 0.89320388, 0.86407767,
       0.86407767, 0.88349515, 0.87378641, 0.87378641, 0.87378641,
       0.88349515, 0.88349515, 0.91262136, 0.90291262, 0.90291262,
       0.90291262, 0.91262136, 0.90291262, 0.90291262, 0.91262136,
       0.89320388, 0.89320388, 0.90291262, 0.91262136, 0.89320388,
       0.9223301 , 0.90291262, 0.89320388, 0.87378641, 0.85436893,
       0.86407767, 0.85436893, 0.86407767, 0.86407767, 0.88349515,
       0.87378641, 0.87378641, 0.88349515, 0.87378641, 0.87378641,
       0.89320388, 0.90291262, 0.87378641, 0.90291262, 0.88349515,
       0.89320388, 0.91262136, 0.89320388, 0.89320388, 0.90291262,
       0.90291262, 0.89320388, 0.85436893, 0.87378641, 0.88349515,
       0.86407767, 0.86407767, 0.88349515, 0.88349515, 0.90291262,
       0.89320388, 0.90291262, 0.91262136, 0.90291262, 0.88349515,
       0.90291262, 0.90291262, 0.9223301 , 0.91262136, 0.90291262,
       0.88349515, 0.9223301 , 0.9223301 , 0.87378641, 0.91262136,
       0.91262136]),
```

```
 'split4_test_score': array([0.82524272, 0.81553398, 0.81553398, 0.81553398, 0.81553398,
        0.82524272, 0.82524272, 0.80582524, 0.81553398, 0.82524272,
        0.82524272, 0.80582524, 0.7961165 , 0.82524272, 0.81553398,
        0.82524272, 0.81553398, 0.80582524, 0.82524272, 0.83495146,
        0.81553398, 0.82524272, 0.85436893, 0.84466019, 0.83495146,
        0.83495146, 0.83495146, 0.81553398, 0.83495146, 0.83495146,
        0.82524272, 0.82524272, 0.83495146, 0.82524272, 0.82524272,
        0.82524272, 0.83495146, 0.82524272, 0.82524272, 0.81553398,
        0.82524272, 0.82524272, 0.82524272, 0.82524272, 0.82524272,
        0.81553398, 0.82524272, 0.82524272, 0.81553398, 0.81553398,
        0.82524272, 0.80582524, 0.81553398, 0.81553398, 0.80582524,
        0.82524272, 0.82524272, 0.80582524, 0.82524272, 0.82524272,
        0.83495146, 0.81553398, 0.82524272, 0.81553398, 0.81553398,
        0.81553398, 0.83495146, 0.82524272, 0.82524272, 0.83495146,
        0.81553398, 0.82524272, 0.83495146, 0.83495146, 0.83495146,
        0.81553398, 0.83495146, 0.83495146, 0.83495146, 0.83495146,
        0.83495146, 0.83495146, 0.83495146, 0.83495146, 0.82524272,
        0.82524272, 0.82524272, 0.82524272, 0.83495146, 0.82524272,
        0.81553398, 0.82524272, 0.82524272, 0.82524272, 0.82524272,
        0.82524272]),
 'mean_test_score': array([0.80776699, 0.80970874, 0.81359223, 0.81553398, 0.80970874,
        0.81359223, 0.81747573, 0.81553398, 0.81747573, 0.81747573,
        0.81553398, 0.81165049, 0.80970874, 0.82330097, 0.82718447,
        0.82524272, 0.82330097, 0.82135922, 0.83495146, 0.83300971,
        0.82718447, 0.82524272, 0.83883495, 0.83300971, 0.81747573,
        0.81359223, 0.81941748, 0.81747573, 0.82135922, 0.81747573,
        0.82135922, 0.82524272, 0.82912621, 0.82524272, 0.82135922,
        0.82912621, 0.83495146, 0.83300971, 0.8368932 , 0.82912621,
        0.83106796, 0.82912621, 0.83883495, 0.8407767 , 0.83106796,
        0.84271845, 0.8368932 , 0.83495146, 0.80776699, 0.80970874,
        0.81747573, 0.80776699, 0.81165049, 0.80970874, 0.81747573,
        0.81941748, 0.81941748, 0.81941748, 0.82135922, 0.82330097,
        0.83106796, 0.82718447, 0.82330097, 0.83495146, 0.82912621,
        0.82912621, 0.84271845, 0.83300971, 0.82912621, 0.83495146,
        0.83106796, 0.83300971, 0.81553398, 0.81553398, 0.81747573,
        0.81359223, 0.81553398, 0.82135922, 0.82718447, 0.83300971,
        0.82718447, 0.82524272, 0.82718447, 0.82718447, 0.82524272,
        0.83300971, 0.83495146, 0.8368932 , 0.8368932 , 0.83106796,
        0.82912621, 0.84271845, 0.84271845, 0.83495146, 0.8407767 ,
        0.84271845]),
 'std_test_score': array([0.02970303, 0.03166894, 0.02840532, 0.03735026, 0.03166894,
        0.02970303, 0.0431575 , 0.03632677, 0.03601405, 0.03440785,
        0.0352736 , 0.03225873, 0.03814929, 0.03495146, 0.03653376,
        0.03190617, 0.03272291, 0.03611859, 0.03190617, 0.03385553,
        0.03998303, 0.04254156, 0.03663682, 0.03601405, 0.02970303,
        0.03214164, 0.03663682, 0.02906142, 0.02982969, 0.03329404,
        0.03396671, 0.03249165, 0.04495471, 0.04298241, 0.04453338,
        0.03960404, 0.04553802, 0.03755161, 0.03440785, 0.04324477,
        0.03225873, 0.03559282, 0.03451726, 0.03714782, 0.03396671,
        0.04137335, 0.03440785, 0.03070172, 0.03601405, 0.02786932,
        0.02840532, 0.02970303, 0.03106796, 0.03045512, 0.03653376,
        0.03451726, 0.03283793, 0.03559282, 0.03166894, 0.03094636,
        0.03611859, 0.03854259, 0.02840532, 0.03473504, 0.02919087,
        0.03451726, 0.03601405, 0.03094636, 0.03505917, 0.03735026,
        0.03814929, 0.03214164, 0.02531729, 0.0352736 , 0.03805033,
        0.02704541, 0.03070172, 0.03663682, 0.03272291, 0.03704618,
        0.03755161, 0.04298241, 0.04652097, 0.04137335, 0.03190617,
        0.03601405, 0.03580406, 0.04359212, 0.03950872, 0.03765188,
        0.03166894, 0.04182652, 0.04182652, 0.02213933, 0.03611859,
        0.03704618]),
 'rank_test_score': array([94, 89, 83, 77, 89, 83, 68, 77, 68, 68, 77, 87, 89, 54, 41, 48, 54,
        58, 14, 21, 41, 48,  8, 21, 68, 83, 64, 68, 58, 68, 58, 48, 33, 48,
        58, 33, 14, 21, 10, 33, 28, 33,  8,  6, 28,  1, 10, 14, 94, 89, 68,
        94, 87, 89, 68, 64, 64, 64, 58, 54, 28, 41, 54, 14, 33, 33,  1, 21,
        33, 14, 28, 21, 77, 77, 68, 83, 77, 58, 41, 21, 41, 48, 41, 41, 48,
        21, 14, 10, 10, 28, 33,  1,  1, 14,  6,  1])}
```

In [93]:

```
res = pd.DataFrame(finmodel.cv_results_)
print(res)
```

```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0        3.168604      0.027913         0.201178        0.005950
1        6.766598      0.098395         0.379987        0.014706
2       10.003987      0.247099         0.608184        0.031252
3        3.323681      0.148967         0.205335        0.005737
4        6.499007      0.160844         0.372144        0.002532
..            ...           ...              ...             ...
91       8.870811      0.398986         0.465083        0.067747
92      11.963724      0.893312         0.582842        0.061648
93       3.746097      0.209059         0.181524        0.005111
94       7.191667      0.142930         0.385090        0.008591
95      10.155814      0.536726         0.463583        0.021239

   param_max_depth param_max_features param_max_leaf_nodes  \
0                8                 5                   15
1                8                 5                   15
2                8                 5                   15
3                8                 5                   15
4                8                 5                   15
..             ...               ...                  ...
91              10                 8                   30
92              10                 8                   30
93              10                 8                   30
94              10                 8                   30
95              10                 8                   30

   param_min_samples_split param_n_estimators  \
0                        2                500
1                        2               1000
2                        2               1500
3                        5                500
4                        5               1000
..                     ...                ...
91                       2               1000
92                       2               1500
93                       5                500
94                       5               1000
95                       5               1500

                                               params  split0_test_score  \
0   {'max_depth': 8, 'max_features': 5, 'max_leaf_...           0.796117
1   {'max_depth': 8, 'max_features': 5, 'max_leaf_...           0.805825
2   {'max_depth': 8, 'max_features': 5, 'max_leaf_...           0.805825
3   {'max_depth': 8, 'max_features': 5, 'max_leaf_...           0.815534
4   {'max_depth': 8, 'max_features': 5, 'max_leaf_...           0.805825
..                                                ...                ...
91  {'max_depth': 10, 'max_features': 8, 'max_leaf...           0.844660
92  {'max_depth': 10, 'max_features': 8, 'max_leaf...           0.844660
93  {'max_depth': 10, 'max_features': 8, 'max_leaf...           0.834951
94  {'max_depth': 10, 'max_features': 8, 'max_leaf...           0.825243
95  {'max_depth': 10, 'max_features': 8, 'max_leaf...           0.844660

    split1_test_score  split2_test_score  split3_test_score  \
0            0.766990           0.796117           0.854369
1            0.766990           0.796117           0.864078
2            0.776699           0.805825           0.864078
3            0.776699           0.786408           0.883495
4            0.766990           0.796117           0.864078
..                ...                ...                ...
91           0.805825           0.815534           0.922330
92           0.805825           0.815534           0.922330
93           0.805825           0.834951           0.873786
94           0.815534           0.825243           0.912621
```

```
95          0.805825          0.825243          0.912621

    split4_test_score  mean_test_score  std_test_score  rank_test_score
0            0.825243         0.807767        0.029703               94
1            0.815534         0.809709        0.031669               89
2            0.815534         0.813592        0.028405               83
3            0.815534         0.815534        0.037350               77
4            0.815534         0.809709        0.031669               89
..                ...              ...             ...              ...
91           0.825243         0.842718        0.041827                1
92           0.825243         0.842718        0.041827                1
93           0.825243         0.834951        0.022139               14
94           0.825243         0.840777        0.036119                6
95           0.825243         0.842718        0.037046                1

[96 rows x 18 columns]
```

In [94]:
```
print("Best parameters - {}".format(finmodel.best_params_))
```

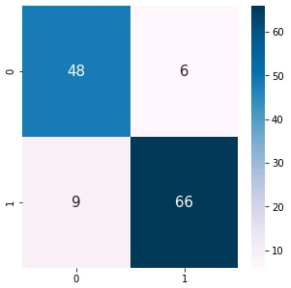
```
Best parameters - {'max_depth': 8, 'max_features': 8, 'max_leaf_nodes': 30, 'min_samples_split': 5, 'n_estimators': 500}
```

In [98]:
```
tuned_rf = RandomForestClassifier(max_depth =  8, max_features = 8, min_samples_split = 5,max_leaf_nodes = 30, n_estimators = 500)
tuned_rf.fit(X_train,y_train)
y_pred_trf = tuned_rf.predict(X_test)

print(accuracy_score(y_pred_trf,y_test))
print(cross_val_score(RandomForestClassifier(max_depth =  8, max_features = 8, min_samples_split = 5,max_leaf_nodes = 30, n_estimators = 500),X_train,y_train,cv = 5).mean())
```

```
0.8837209302325582
0.833009708737864
```

In [99]:
```
cm_trf = confusion_matrix(y_test, y_pred_trf)
plt.rcParams['figure.figsize'] = (5, 5)
sns.heatmap(cm_trf, annot = True, annot_kws = {'size':15}, cmap = 'PuBu')
```

Out[99]:
```
<AxesSubplot:>
```



In [100…
```
data
```

Out[100…

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender	Married	Dependents_0	Dependents_1	Dependents_2	Dependents_3+	Education_Graduate	Self_Employed	Property_Area_Rura
0	76.478755	0.000000	12.100089	360.0	1.0	1	0	1	0	0	0	1	0	
1	67.697858	38.832976	11.313708	360.0	1.0	1	1	0	1	0	0	1	0	
2	54.772256	0.000000	8.124038	360.0	1.0	1	1	1	0	0	0	1	1	
3	50.823223	48.559242	10.954451	360.0	1.0	1	1	1	0	0	0	0	0	
4	77.459667	0.000000	11.874342	360.0	1.0	1	0	1	0	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
607	63.142696	37.563280	12.529964	360.0	1.0	1	1	0	0	1	0	0	0	
608	56.850682	44.158804	10.392305	360.0	1.0	1	1	1	0	0	0	1	0	
609	53.851648	0.000000	8.426150	360.0	1.0	0	0	1	0	0	0	1	0	
611	89.844310	15.491933	15.905974	360.0	1.0	1	1	0	1	0	0	1	0	
612	87.080423	0.000000	13.674794	360.0	1.0	1	1	0	0	1	0	1	0	

396 rows × 17 columns

6. Test Dataset

In [101…
```
test_data = pd.read_csv("loan-test.csv")
```

In [102…
```
test_data
```

Out[102…

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	1.0	Urban
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	1.0	Urban
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	1.0	Urban
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	NaN	Urban
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	1.0	Urban
...	...	...	...	...	...	...	...	...	...	...	...	...
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009	1777	113.0	360.0	1.0	Urban
363	LP002975	Male	Yes	0	Graduate	No	4158	709	115.0	360.0	1.0	Urban
364	LP002980	Male	No	0	Graduate	No	3250	1993	126.0	360.0	NaN	Semiurban
365	LP002986	Male	Yes	0	Graduate	No	5000	2393	158.0	360.0	1.0	Rural
366	LP002989	Male	No	0	Graduate	Yes	9200	0	98.0	180.0	1.0	Rural

367 rows × 12 columns

7. Preparing test data

In [103…
```
test_data.isnull().sum()
```

Out[103…
```
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
```

In [104…
```
test_data.shape
```

Out[104…
```
(367, 12)
```

In [105…
```
test_data = test_data.dropna()
```

In [106…
```
test_data.shape
```

Out[106…
```
(289, 12)
```

In [107…
```
test_data = test_data.drop(["Loan_ID"],axis = "columns")
```

In [108…
```
test_data.ApplicantIncome = np.sqrt(test_data.ApplicantIncome)
test_data.CoapplicantIncome = np.sqrt(test_data.CoapplicantIncome)
test_data.LoanAmount = np.sqrt(test_data.LoanAmount)
```

In [109…
```
test_data = pd.get_dummies(test_data)

test_data = test_data.drop(['Gender_Female', 'Married_No', 'Education_Not Graduate',
              'Self_Employed_No'], axis = 1)

latcolumnstest = {"Gender_Male":"Gender","Married_Yes":"Married","Education_Graduare":"Education","Self_Employed_Yes":"Self_Employed"}

test_data.rename(columns = latcolumnstest,inplace=True)
```

In [110…
```
test_data
```

Out[110…

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender	Married	Dependents_0	Dependents_1	Dependents_2	Dependents_3+	Education_Graduate	Self_Employed	Property_Area_Rura
0	75.630682	0.000000	10.488088	360.0	1.0	1	1	1	0	0	0	1	0	
1	55.461698	38.729833	11.224972	360.0	1.0	1	1	0	1	0	0	1	0	
2	70.710678	42.426407	14.422205	360.0	1.0	1	1	0	0	1	0	1	0	
4	57.236352	0.000000	8.831761	360.0	1.0	1	0	1	0	0	0	0	0	
5	46.529560	58.497863	12.328828	360.0	1.0	1	1	1	0	0	0	0	1	
…	…	…	…	…	…	…	…	…	…	…	…	…	…	.
361	47.634021	46.551047	9.949874	360.0	1.0	1	1	0	1	0	0	1	0	
362	63.316664	42.154478	10.630146	360.0	1.0	1	1	0	0	0	1	0	1	
363	64.482556	26.627054	10.723805	360.0	1.0	1	1	1	0	0	0	1	0	
365	70.710678	48.918299	12.569805	360.0	1.0	1	1	1	0	0	0	1	0	
366	95.916630	0.000000	9.899495	180.0	1.0	1	0	1	0	0	0	1	1	

289 rows × 16 columns

In [111…
```
test_data = scaling.transform(test_data)
```

In [112…
```
test_pred = tuned_rf.predict(test_data)
```

In [113…
```
test_pred
```

Out[113…
```
array([1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 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, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 0, 1], dtype=uint8)
```

8. Conclusion

- The data is small, so models can't train properly.
- We cant train properly as the y is imbalanced (values of N are less hence model cant train properly on it).
- The models are overfeeding due to above reasons.
- Feature Engineering can be done for better results.
- We can completely reject or accept newer calls.