MACHINE LEARNING THEORY DA

Loan Approval Prediction

NAME:BOGGAVARAPU CH N V SHIVANI

REG NO:20BCE0563

1)Details of Dataset:

Title: Loan Approval Prediction

Problem Statement:

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

Problem
Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling 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 customers segments, those are eligible for loan amount so that they can specifically target these customers. Here they have provided a partial data set.

Dataset Description:

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

Description of dataset:

In [96]:

```
# Importing libraries
import pandas as pd
train_df = pd.read_csv('train_u6lujuX_CVtuZ9i.csv')
train_df.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
```

Shape of the dataset:

```
In [99]:  #Preview data information
          train_df.shape
```

```
Out[99]: (614, 13)
```

```
In [100]:  #Check missing values
           train_df.isnull().sum()
```

```
Out[100]: 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
```

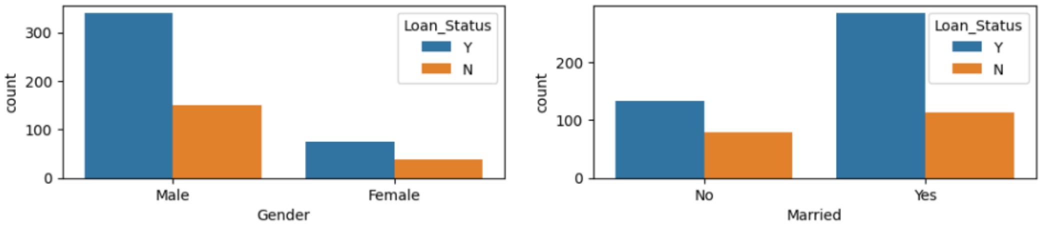
Counting categorical and numerical columns:

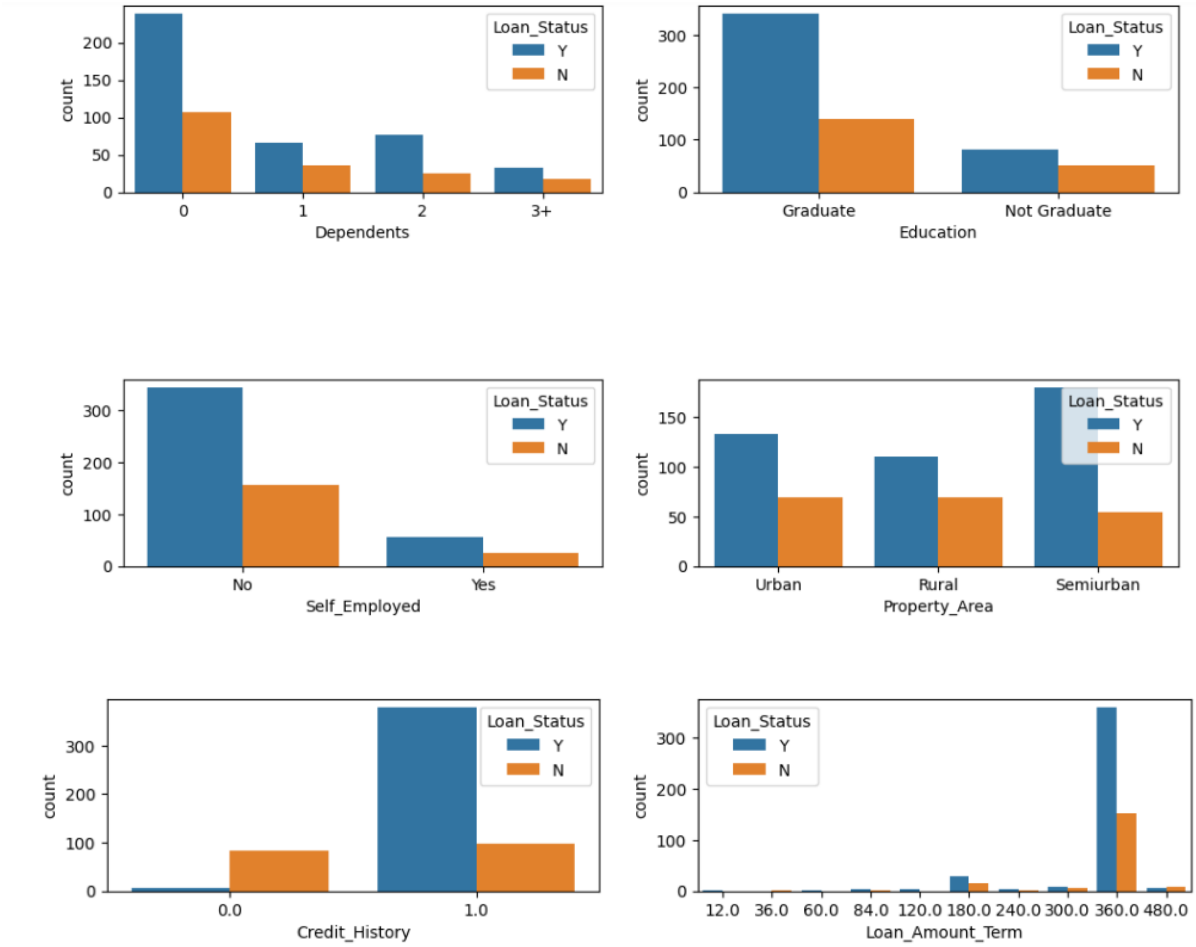
```
In [122]:  # Count number of Categorical and Numerical Columns
           train_df = train_df.drop(columns=['Loan_ID'])
           categorical_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area','Credit_History','Loa
           print(categorical_columns)
           numerical_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
           print(numerical_columns)
```

```
['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area', 'Credit_History', 'Loan_Amount_Term']
['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
```

Analysing values assigned to columns:

```
In [123]:  # Data Visualization Libraries
           import seaborn as sns
           import matplotlib.pyplot as plt


           fig,axes = plt.subplots(4,2,figsize=(12,15))
           for idx,cat_col in enumerate(categorical_columns):
               row,col = idx//2,idx%2
               sns.countplot(x=cat_col,data=train_df,hue='Loan_Status',ax=axes[row,col])


           plt.subplots_adjust(hspace=1)
```





Plots above convey following things about the dataset:

1. Loan Approval Status: About 2/3rd of applicants have been granted loan.
2. Sex: There are more Men than Women (approx. 3x)
3. Martial Status: 2/3rd of the population in the dataset is Marred; Married applicants are more likely to be granted loans.
4. Dependents: Majority of the population have zero dependents and are also likely to accepted for loan.
5. Education: About 5/6th of the population is Graduate and graduates have higher proportion of loan approval
6. Employment: 5/6th of population is not self employed.
7. Property Area: More applicants from Semi-urban and also likely to be granted loans.
8. Applicant with credit history are far more likely to be accepted.
9. Loan Amount Term: Majority of the loans taken are for 360 Months (30 years).

Pre-processing Data:

Input data needs to be pre-processed before we feed it to model. Following things need to be taken care:

1. Encoding Categorical Features.
2. Imputing missing values

Encoding the categorical features:

```
In [125]: # Encoding categrical Features
          train_df_encoded = pd.get_dummies(train_df,drop_first=True)
          train_df_encoded.head()
```

Out[125]:

rried_Yes	Dependents_1	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Property_Area_Urban	Loan_Status_Y
0	0	0	0	0	0	0	1	1
1	1	0	0	0	0	0	0	0
1	0	0	0	0	1	0	1	1
1	0	0	0	1	0	0	1	1
0	0	0	0	0	0	0	1	1

Splitting the dataset into training and testing and handling missing values in the dataset:

```
In [126]: # Split Features and Target Varible
          X = train_df_encoded.drop(columns='Loan_Status_Y')
          y = train_df_encoded['Loan_Status_Y']

          # Splitting into Train -Test Data
          from sklearn.model_selection import train_test_split
          X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,stratify =y,random_state =42)

          # Handling/Imputing Missing values
          from sklearn.impute import SimpleImputer
          imp = SimpleImputer(strategy='mean')
          imp_train = imp.fit(X_train)
          X_train = imp_train.transform(X_train)
          X_test_imp = imp_train.transform(X_test)
```

2)Algorithms used:

Loan_Status is the target variable which is categorical in nature.So,I choose to use Decision tree algorithm and Logistic Regression.

Decision Tree Classifier:

A decision tree is a supervised learning algorithm that is used for classification and regression modelling. Regression is a method used for predictive modelling, so these trees are used to either classify data or predict what will come next. Decision trees are mainly used to perform **classification tasks**. Samples are submitted to a test in each node of the tree and guided through the tree based on the result.

Model 1: Decision Tree Classifier

In [127]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,f1_score


tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train,y_train)
y_pred = tree_clf.predict(X_train)
print("Training Data Set Accuracy: ", accuracy_score(y_train,y_pred))
print("Training Data F1 Score ", f1_score(y_train,y_pred))

print("Validation Mean F1 Score: ",cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='f1_macro').mean())
print("Validation Mean Accuracy: ",cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='accuracy').mean())
```

```
Training Data Set Accuracy:  1.0
Training Data F1 Score  1.0
Validation Mean F1 Score:  0.6495619698928218
Validation Mean Accuracy:  0.6944547515976087
```

Overfitting Problem:

We can see from above metrics that Training Accuracy > Test Accuracy with default settings of Decision Tree classifier. Hence, model is overfit. We will try some Hyper-parameter tuning and see if it helps.

First let's try tuning 'Max_Depth' of tree:

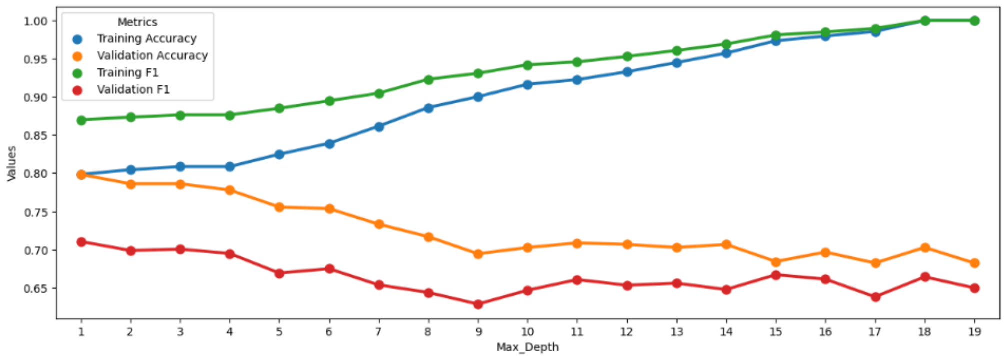
Tuning 'Max_Depth' of tree

In [128]:

```python
training_accuracy = []
val_accuracy = []
training_f1 = []
val_f1 = []
tree_depths = []

for depth in range(1,20):
    tree_clf = DecisionTreeClassifier(max_depth=depth)
    tree_clf.fit(X_train,y_train)
    y_training_pred = tree_clf.predict(X_train)

    training_acc = accuracy_score(y_train,y_training_pred)
    train_f1 = f1_score(y_train,y_training_pred)
    val_mean_f1 = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='f1_macro').mean()
    val_mean_accuracy = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='accuracy').mean()

    training_accuracy.append(training_acc)
    val_accuracy.append(val_mean_accuracy)
    training_f1.append(train_f1)
    val_f1.append(val_mean_f1)
    tree_depths.append(depth)


Tuning_Max_depth = {"Training Accuracy": training_accuracy, "Validation Accuracy": val_accuracy, "Training F1": training_f1,
Tuning_Max_depth_df = pd.DataFrame.from_dict(Tuning_Max_depth)

plot_df = Tuning_Max_depth_df.melt('Max_Depth',var_name='Metrics',value_name="Values")
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Max_Depth", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

```
Tuning_Max_depth = {"Training Accuracy": training_accuracy, "Validation Accuracy": val_accuracy, "Training F1": training_f1,
Tuning_Max_depth_df = pd.DataFrame.from_dict(Tuning_Max_depth)

plot_df = Tuning_Max_depth_df.melt('Max_Depth',var_name='Metrics',value_name="Values")
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Max_Depth", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

Out[128]: <Axes: xlabel='Max_Depth', ylabel='Values'>



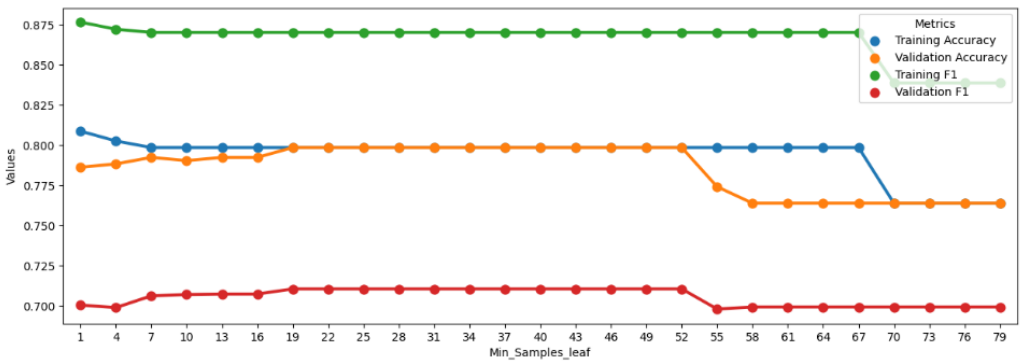
Visualising Decision Tree with Max Depth = 3

In [133]:

```
training_accuracy = []
val_accuracy = []
training_f1 = []
val_f1 = []
min_samples_leaf = []
import numpy as np
for samples_leaf in range(1,80,3): ### Sweeping from 1% samples to 10% samples per leaf
    tree_clf = DecisionTreeClassifier(max_depth=3,min_samples_leaf = samples_leaf)
    tree_clf.fit(X_train,y_train)
    y_training_pred = tree_clf.predict(X_train)

    training_acc = accuracy_score(y_train,y_training_pred)
    train_f1 = f1_score(y_train,y_training_pred)
    val_mean_f1 = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='f1_macro').mean()
    val_mean_accuracy = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='accuracy').mean()

    training_accuracy.append(training_acc)
    val_accuracy.append(val_mean_accuracy)
    training_f1.append(train_f1)
    val_f1.append(val_mean_f1)
    min_samples_leaf.append(samples_leaf)


Tuning_min_samples_leaf = {"Training Accuracy": training_accuracy, "Validation Accuracy": val_accuracy, "Training F1": traini
Tuning_min_samples_leaf_df = pd.DataFrame.from_dict(Tuning_min_samples_leaf)

plot_df = Tuning_min_samples_leaf_df.melt('Min_Samples_leaf',var_name='Metrics',value_name="Values")
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Min_Samples_leaf", y="Values",hue="Metrics", data=plot_df,ax=ax)
```


```
Tuning_min_samples_leaf = {"Training Accuracy": training_accuracy, "Validation Accuracy": val_accuracy, "Training F1": traini
Tuning_min_samples_leaf_df = pd.DataFrame.from_dict(Tuning_min_samples_leaf)

plot_df = Tuning_min_samples_leaf_df.melt('Min_Samples_leaf',var_name='Metrics',value_name="Values")
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Min_Samples_leaf", y="Values",hue="Metrics", data=plot_df,ax=ax)
```


Out[133]: <Axes: xlabel='Min_Samples_leaf', ylabel='Values'>



From above plot, we will choose Min_Samples_leaf to 35 to improve test accuracy.

Logistic Regression:

Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring. Logistic regression is a supervised learning **classification algorithm** used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

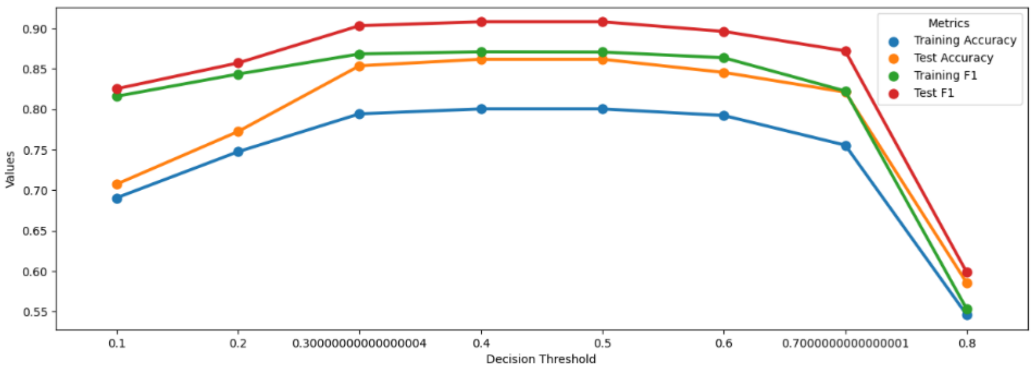
Finally, we will try Logistic Regression Model by sweeping threshold values.

Model 2: Logistic Regression

In [135]:

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

train_accuracies = []
train_f1_scores = []
test_accuracies = []
test_f1_scores = []
thresholds = []

#for thresh in np.linspace(0.1,0.9,8): ## Sweeping from threshold of 0.1 to 0.9
for thresh in np.arange(0.1,0.9,0.1): ## Sweeping from threshold of 0.1 to 0.9
    logreg_clf = LogisticRegression(solver='liblinear')
    logreg_clf.fit(X_train,y_train)

    y_pred_train_thresh = logreg_clf.predict_proba(X_train)[:,1]
    y_pred_train = (y_pred_train_thresh > thresh).astype(int)

    train_acc = accuracy_score(y_train,y_pred_train)
    train_f1 = f1_score(y_train,y_pred_train)

    y_pred_test_thresh = logreg_clf.predict_proba(X_test_imp)[:,1]
    y_pred_test = (y_pred_test_thresh > thresh).astype(int)

    test_acc = accuracy_score(y_test,y_pred_test)
    test_f1 = f1_score(y_test,y_pred_test)

    train_accuracies.append(train_acc)
    train_f1_scores.append(train_f1)
    test_accuracies.append(test_acc)
    test_f1_scores.append(test_f1)
    thresholds.append(thresh)

Threshold_logreg = {"Training Accuracy": train_accuracies, "Test Accuracy": test_accuracies, "Training F1": train_f1_scores,
Threshold_logreg_df = pd.DataFrame.from_dict(Threshold_logreg)

plot_df = Threshold_logreg_df.melt('Decision Threshold',var_name='Metrics',value_name="Values")
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Decision Threshold", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

Out[135]: <Axes: xlabel='Decision Threshold', ylabel='Values'>



3)Performance evolutions:

As, this is a classification dataset the metrics we use are Precision,Recall,Accuracy,Error rate and F1 square.

- **Recall**: the ability of a classification model to identify all data points in a relevant class.
- **Precision:** the ability of a classification model to return only the data points in a class
- **F1 score:** a single metric that combines recall and precision using the harmonic mean.
- **Accuracy** : The percentage of our predictions are right.
- **Error Rate :**The percentage of our prediction are wrong**.**

Decision Tree:

In [151]:
```
from sklearn.metrics import confusion_matrix
tree_clf = DecisionTreeClassifier(max_depth=3,min_samples_leaf = 35)
tree_clf.fit(X_train,y_train)
y_pred = tree_clf.predict(X_test_imp)
print("Test Accuracy: ",accuracy_score(y_test,y_pred))
print("Test Error Rate: ",1-accuracy_score(y_test,y_pred))
print("Test F1 Score: ",f1_score(y_test,y_pred))
print("Test Precision: ",precision_score(y_test,y_pred))
print("Test Recall: ",recall_score(y_test,y_pred))
print("Confusion Matrix on Test Data")
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

```
Test Accuracy:  0.8536585365853658
Test Error Rate:  0.14634146341463417
Test F1 Score:  0.903225806451613
Test Precision:  0.8316831683168316
Test Recall:  0.9882352941176471
Confusion Matrix on Test Data
```

Out[151]:

Predicted	0	1	All
True			
0	21	17	38
1	1	84	85
All	22	101	123

In [153]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.55      0.70        38
           1       0.83      0.99      0.90        85

    accuracy                           0.85       123
   macro avg       0.89      0.77      0.80       123
weighted avg       0.87      0.85      0.84       123
```

So ,by observing the metrics we can say that accuracy achieved using decision tree is 86%.Recall is 99% ,precision is 83% and F1 score is 90%.

Logistic Regression:

Logistic Regression does slightly better than Decision Tree. Based on the above Test/Train curves, we can keep threshold to 0.4.

In [143]:
```
thresh = 0.4
y_pred_test_thresh = logreg_clf.predict_proba(X_test_imp)[:,1]
y_pred = (y_pred_test_thresh > thresh).astype(int)
print("Test Accuracy: ",accuracy_score(y_test,y_pred))
print("Test Error Rate: ",1-accuracy_score(y_test,y_pred))
print("Test F1 Score: ",f1_score(y_test,y_pred))
print("Test Precision: ",precision_score(y_test,y_pred))
print("Test Recall: ",recall_score(y_test,y_pred))
print("Confusion Matrix on Test Data")
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

```
Test Accuracy:  0.8617886178861789
Test Error Rate:  0.1382113821138211
Test F1 Score:  0.9081081081081082
Test Precision:  0.84
Test Recall:  0.9882352941176471
Confusion Matrix on Test Data
```

Out[143]:

Predicted	0	1	All
True			
0	22	16	38
1	1	84	85
All	23	100	123

In [152]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.55      0.70        38
           1       0.83      0.99      0.90        85

    accuracy                           0.85       123
   macro avg       0.89      0.77      0.80       123
weighted avg       0.87      0.85      0.84       123
```

So ,by observing the metrics we can say that accuracy achieved using decision tree is 86%.Recall is 99% ,precision is 84% and F1 score is 90%.

4)Result and Discussions:

Logistic Regression Confusion matrix is very similar to Decision Tree Classifier. In this analysis, I did extensive analysis of input data and was able to achieve Test Accuracy of **86 %.**So, there is no much difference in using Decision tree classifier and logistic regression as both of the algorithms achieve accuracy of 86%.

Loan Approval Prediction

NAME:BOGGAVARAPU CH N V SHIVANI

REG NO:20BCE0563

In [19]:
```
# Importing libraries
import pandas as pd
import seaborn as sns
train_df = pd.read_csv('train_u6lujuX_CVtuZ9i.csv')
train_df.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 [20]:
```
#Preview data information
train_df.shape
```

Out[20]:
```
(614, 13)
```

In [21]:
```
#Check missing values
train_df.isnull().sum()
```

Out[21]:
```
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
```

In [22]:

```
# percent of missing "Gender"
print('Percent of missing "Gender" records is %.2f%%' %((train_df['Gender'].isnul
```

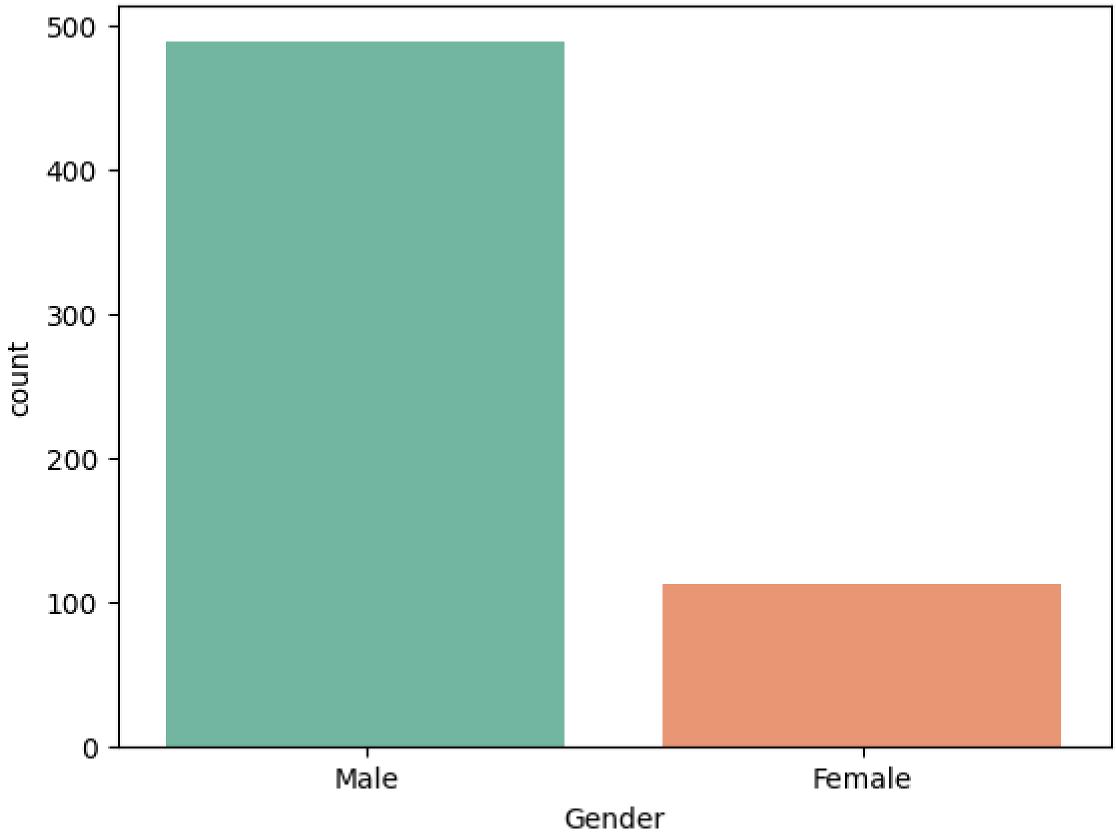
```
Percent of missing "Gender" records is 2.12%
```

In [23]:

```
print("Number of people who take a loan group by gender :")
print(train_df['Gender'].value_counts())
sns.countplot(x='Gender', data=train_df, palette = 'Set2')
```

```
Number of people who take a loan group by gender :
Male      489
Female    112
Name: Gender, dtype: int64
```

Out[23]: `<Axes: xlabel='Gender', ylabel='count'>`



In [24]:
```
train_df.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
```

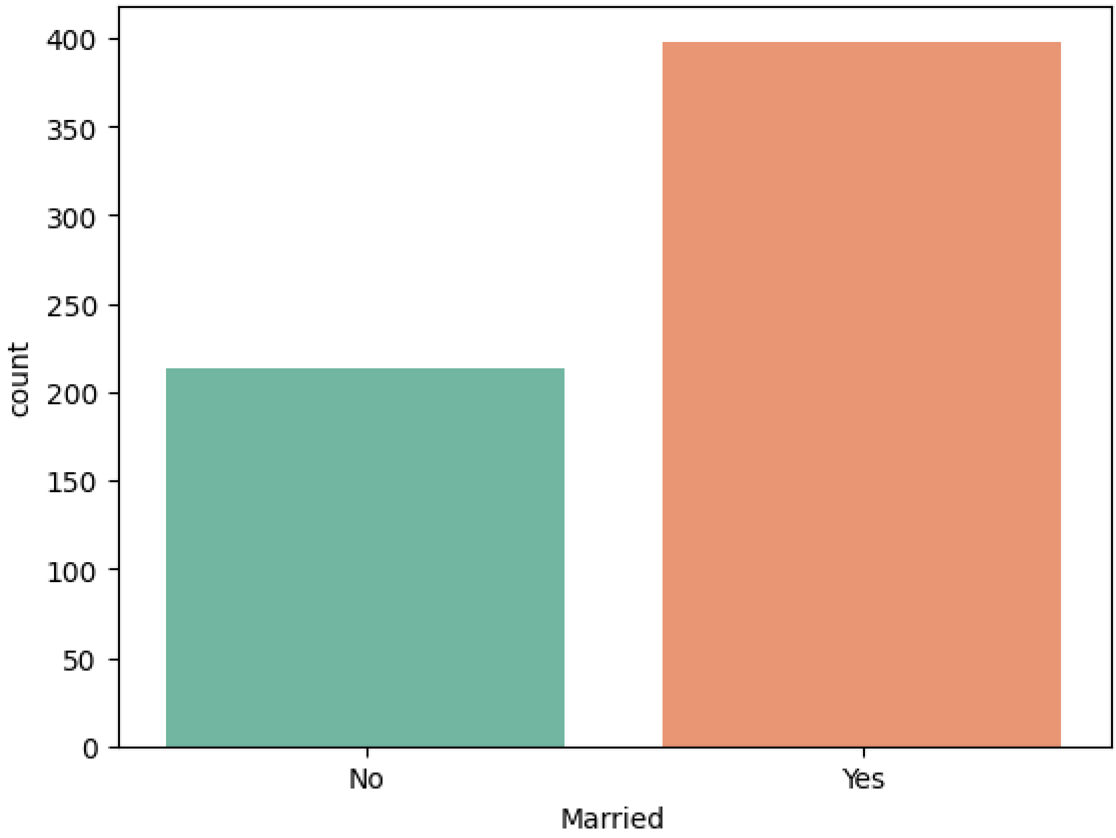
In [25]:
```
# percent of missing "Married"
print('Percent of missing "Married" records is %.2f%%' %((train_df['Married'].isnu
```

```
Percent of missing "Married" records is 0.49%
```

In [26]:
```python
print("Number of people who take a loan group by marital status :")
print(train_df['Married'].value_counts())
sns.countplot(x='Married', data=train_df, palette = 'Set2')
```

```
Number of people who take a loan group by marital status :
Yes    398
No     213
Name: Married, dtype: int64
```

Out[26]: <Axes: xlabel='Married', ylabel='count'>



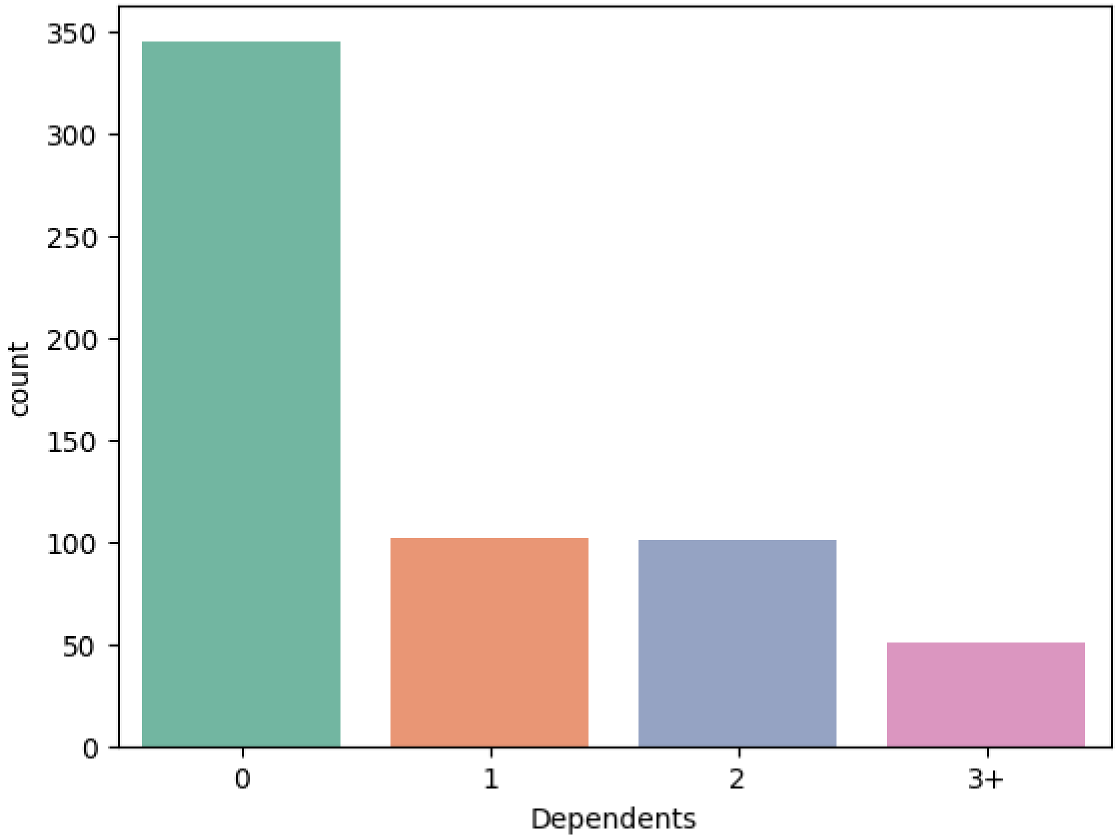
In [27]:
```python
# percent of missing "Dependents"
print('Percent of missing "Dependents" records is %.2f%%' %((train_df['Dependents
```

```
Percent of missing "Dependents" records is 2.44%
```

In [28]:
```
print("Number of people who take a loan group by dependents :")
print(train_df['Dependents'].value_counts())
sns.countplot(x='Dependents', data=train_df, palette = 'Set2')
```

```
Number of people who take a loan group by dependents :
0      345
1      102
2      101
3+      51
Name: Dependents, dtype: int64
```

Out[28]: <Axes: xlabel='Dependents', ylabel='count'>



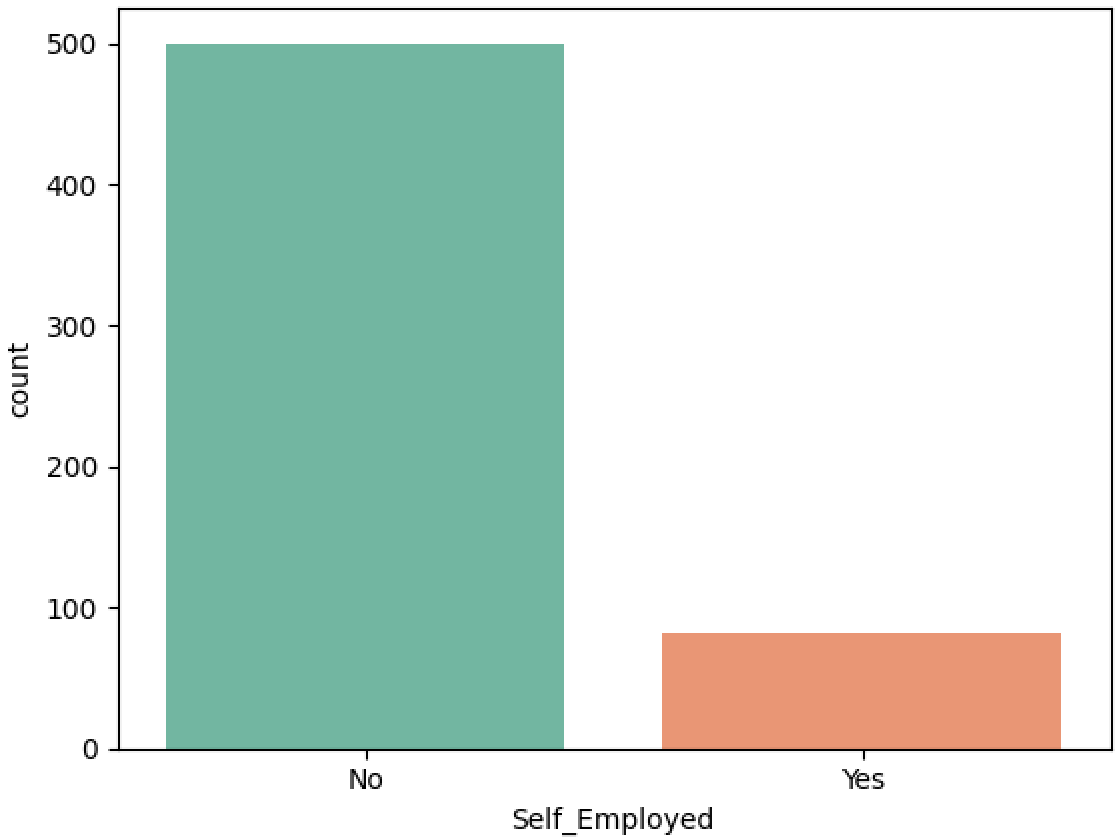
In [29]:
```
# percent of missing "Self_Employed"
print('Percent of missing "Self_Employed" records is %.2f%%' %((train_df['Self_Em
```

```
Percent of missing "Self_Employed" records is 5.21%
```

In [30]:
```python
print("Number of people who take a loan group by self employed :")
print(train_df['Self_Employed'].value_counts())
sns.countplot(x='Self_Employed', data=train_df, palette = 'Set2')
```

```
Number of people who take a loan group by self employed :
No     500
Yes     82
Name: Self_Employed, dtype: int64
```

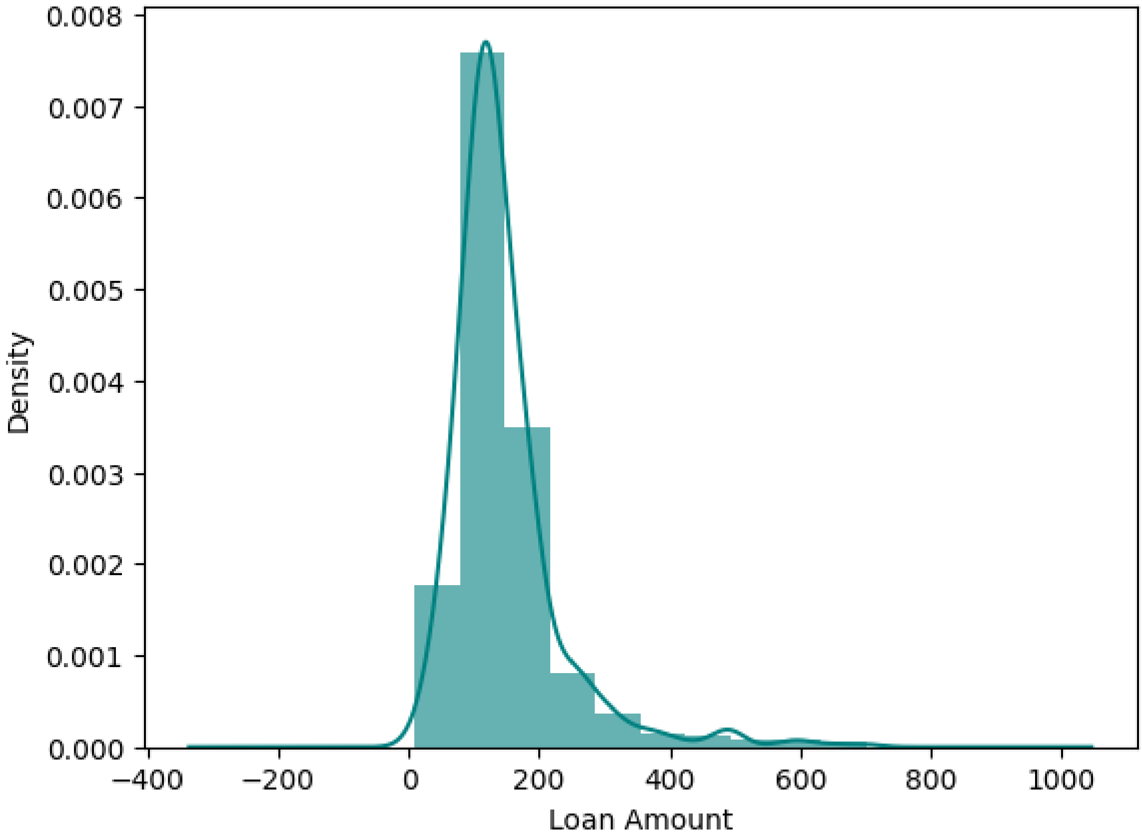
Out[30]:
```
<Axes: xlabel='Self_Employed', ylabel='count'>
```



In [31]:
```python
# percent of missing "LoanAmount"
print('Percent of missing "LoanAmount" records is %.2f%%' %((train_df['LoanAmount
```

```
Percent of missing "LoanAmount" records is 3.58%
```

In [33]:
```
import matplotlib.pyplot as plt
ax = train_df["LoanAmount"].hist(density=True, stacked=True, color='teal', alpha=(
train_df["LoanAmount"].plot(kind='density', color='teal')
ax.set(xlabel='Loan Amount')
plt.show()
```



In [34]:
```
# percent of missing "Loan_Amount_Term"
print('Percent of missing "Loan_Amount_Term" records is %.2f%%' %((train_df['Loan_
```

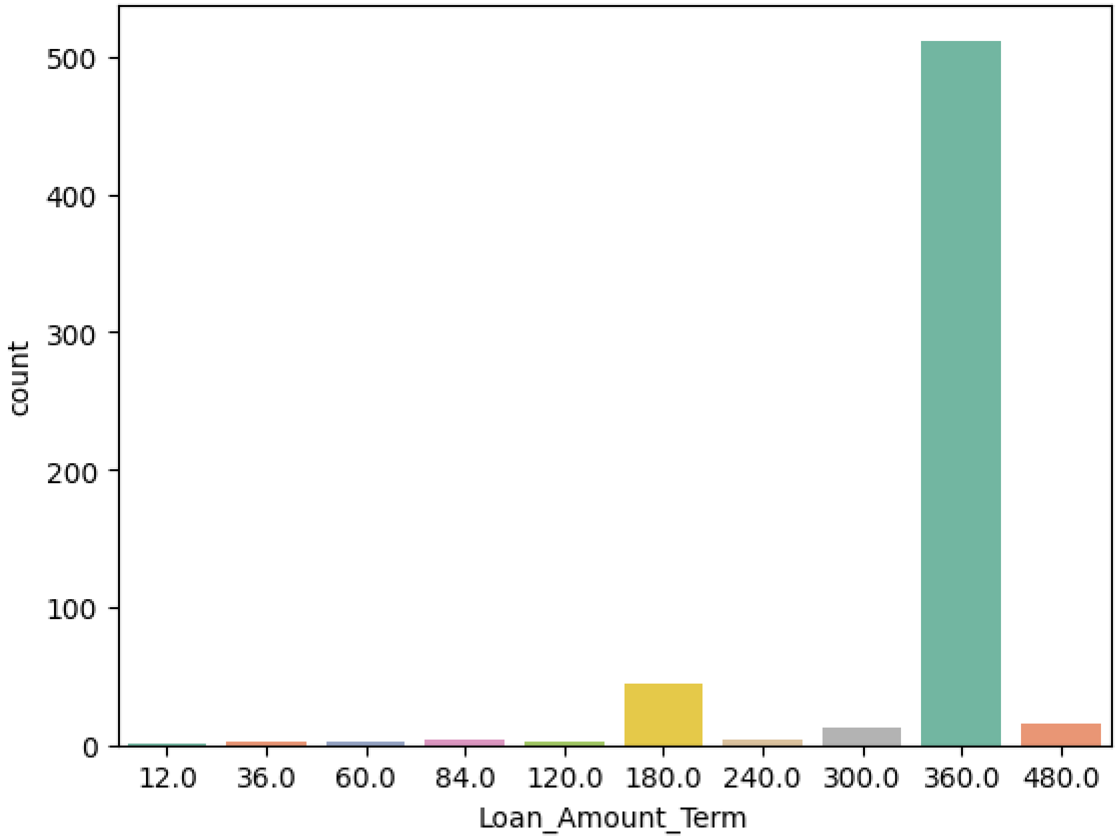
```
Percent of missing "Loan_Amount_Term" records is 2.28%
```

In [35]:

```python
print("Number of people who take a loan group by loan amount term :")
print(train_df['Loan_Amount_Term'].value_counts())
sns.countplot(x='Loan_Amount_Term', data=train_df, palette = 'Set2')
```

```
Number of people who take a loan group by loan amount term :
360.0    512
180.0     44
480.0     15
300.0     13
240.0      4
84.0       4
120.0      3
60.0       2
36.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

Out[35]: <Axes: xlabel='Loan_Amount_Term', ylabel='count'>



In [36]:

```
train_df.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 [37]:

```
# percent of missing "Credit_History"
print('Percent of missing "Credit_History" records is %.2f%%' %((train_df['Credit_
```

```
Percent of missing "Credit_History" records is 8.14%
```

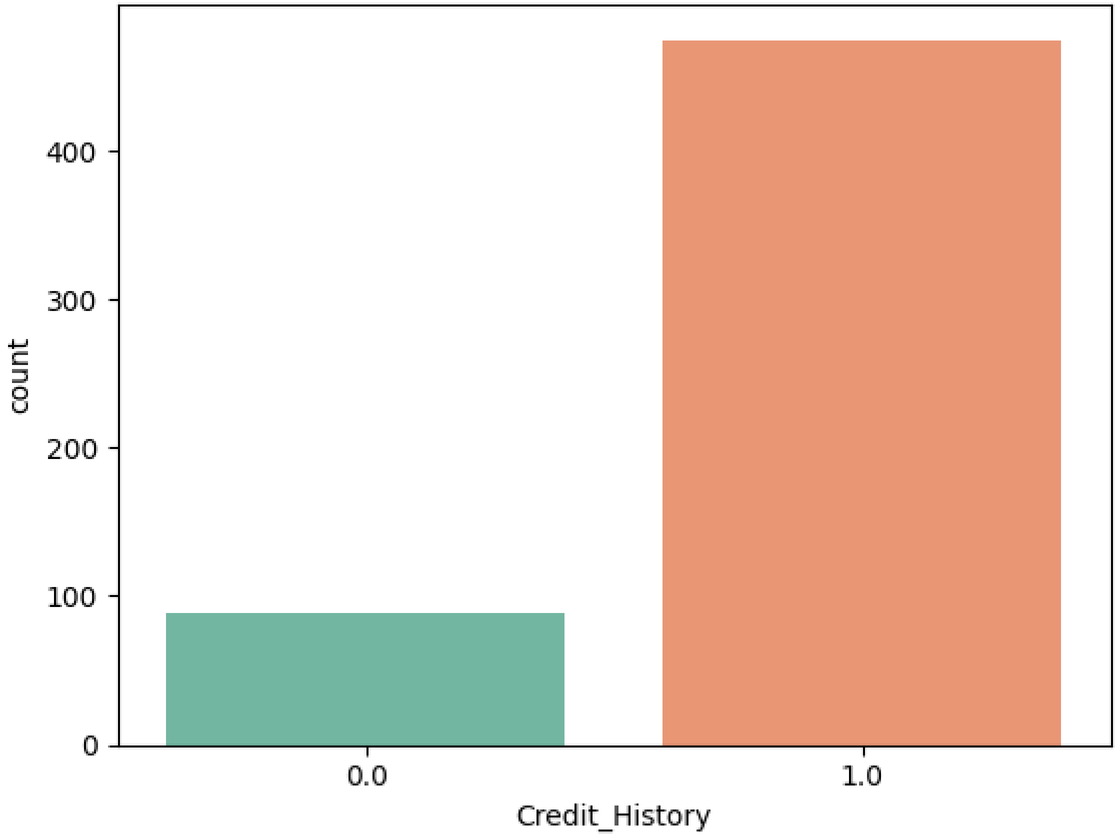
In [38]:

```
print("Number of people who take a loan group by credit history :")
print(train_df['Credit_History'].value_counts())
sns.countplot(x='Credit_History', data=train_df, palette = 'Set2')
```

```
Number of people who take a loan group by credit history :
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

Out[38]:

```
<Axes: xlabel='Credit_History', ylabel='count'>
```



In [39]:

```
train_df.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
```

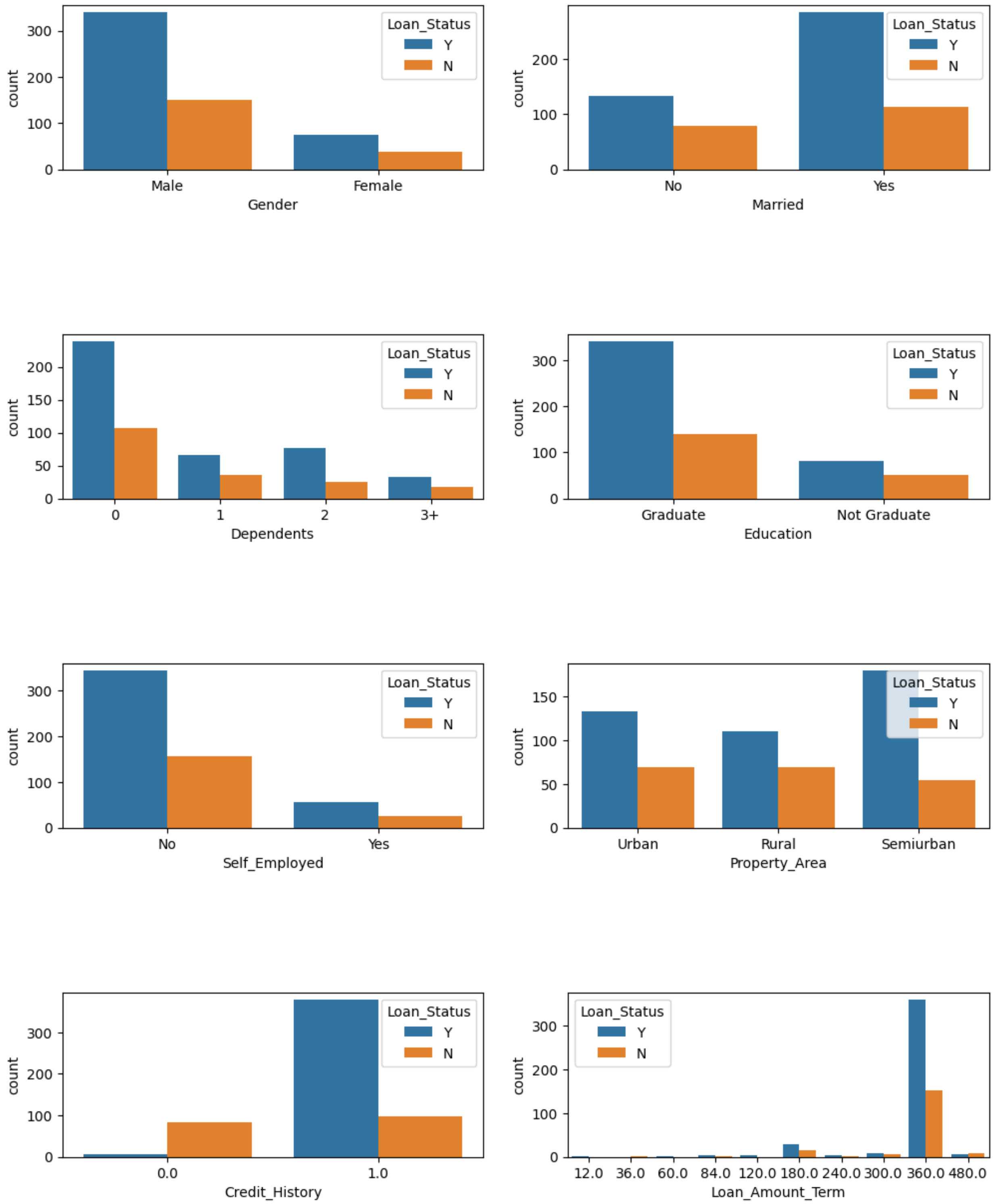
In [40]:

```
# Count number of Categorical and Numerical Columns
train_df = train_df.drop(columns=['Loan_ID'])
categorical_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Emplo
print(categorical_columns)
numerical_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
print(numerical_columns)
```

```
['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Are
a', 'Credit_History', 'Loan_Amount_Term']
['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
```

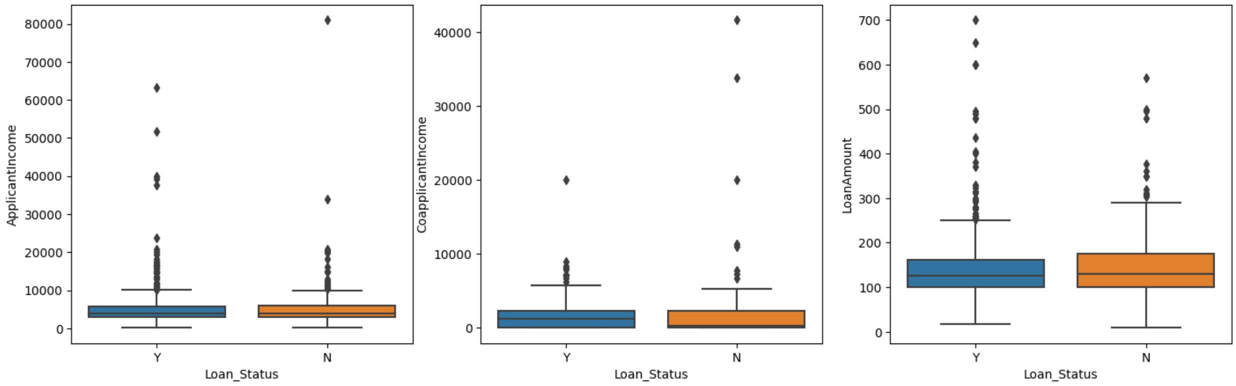
In [41]:

```python
# Data Visualization libraries
import seaborn as sns
import matplotlib.pyplot as plt


fig,axes = plt.subplots(4,2,figsize=(12,15))
for idx,cat_col in enumerate(categorical_columns):
    row,col = idx//2,idx%2
    sns.countplot(x=cat_col,data=train_df,hue='Loan_Status',ax=axes[row,col])


plt.subplots_adjust(hspace=1)
```



In [42]:
```python
fig,axes = plt.subplots(1,3,figsize=(17,5))
for idx,cat_col in enumerate(numerical_columns):
    sns.boxplot(y=cat_col,data=train_df,x='Loan_Status',ax=axes[idx])

print(train_df[numerical_columns].describe())
plt.subplots_adjust(hspace=1)
```

```
       ApplicantIncome  CoapplicantIncome  LoanAmount
count       614.000000         614.000000  592.000000
mean       5403.459283        1621.245798  146.412162
std        6109.041673        2926.248369   85.587325
min         150.000000           0.000000    9.000000
25%        2877.500000           0.000000  100.000000
50%        3812.500000        1188.500000  128.000000
75%        5795.000000        2297.250000  168.000000
max       81000.000000       41667.000000  700.000000
```



In [43]:
```python
# Encoding categrical Features
train_df_encoded = pd.get_dummies(train_df,drop_first=True)
train_df_encoded.head()
```

Out[43]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Ma
0	5849	0.0	NaN	360.0	1.0	
1	4583	1508.0	128.0	360.0	1.0	
2	3000	0.0	66.0	360.0	1.0	
3	2583	2358.0	120.0	360.0	1.0	
4	6000	0.0	141.0	360.0	1.0	

In [44]:

```
# Split Features and Target Varible
X = train_df_encoded.drop(columns='Loan_Status_Y')
y = train_df_encoded['Loan_Status_Y']

# Splitting into Train -Test Data
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,stratify =y,ra

# Handling/Imputing Missing values
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='mean')
imp_train = imp.fit(X_train)
X_train = imp_train.transform(X_train)
X_test_imp = imp_train.transform(X_test)
```

Model 1: Decision Tree Classifier

In [45]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score,f1_score


tree_clf = DecisionTreeClassifier()
tree_clf.fit(X_train,y_train)
y_pred = tree_clf.predict(X_train)
print("Training Data Set Accuracy: ", accuracy_score(y_train,y_pred))
print("Training Data F1 Score ", f1_score(y_train,y_pred))

print("Validation Mean F1 Score: ",cross_val_score(tree_clf,X_train,y_train,cv=5,
print("Validation Mean Accuracy: ",cross_val_score(tree_clf,X_train,y_train,cv=5,
```

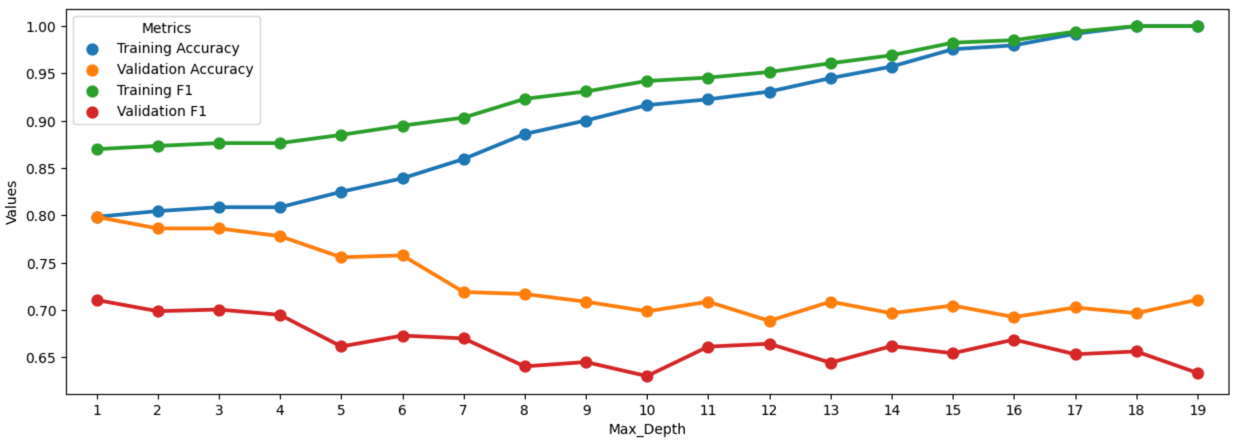
```
Training Data Set Accuracy:  1.0
Training Data F1 Score  1.0
Validation Mean F1 Score:  0.6562875539531455
Validation Mean Accuracy:  0.6964131106988249
```

Tuning 'Max_Depth' of tree

In [46]:

```
training_accuracy = []
val_accuracy = []
training_f1 = []
val_f1 = []
tree_depths = []

for depth in range(1,20):
    tree_clf = DecisionTreeClassifier(max_depth=depth)
    tree_clf.fit(X_train,y_train)
    y_training_pred = tree_clf.predict(X_train)

    training_acc = accuracy_score(y_train,y_training_pred)
    train_f1 = f1_score(y_train,y_training_pred)
    val_mean_f1 = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='f1_macro
    val_mean_accuracy = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='acc

    training_accuracy.append(training_acc)
    val_accuracy.append(val_mean_accuracy)
    training_f1.append(train_f1)
    val_f1.append(val_mean_f1)
    tree_depths.append(depth)


Tuning_Max_depth = {"Training Accuracy": training_accuracy, "Validation Accuracy"
Tuning_Max_depth_df = pd.DataFrame.from_dict(Tuning_Max_depth)

plot_df = Tuning_Max_depth_df.melt('Max_Depth',var_name='Metrics',value_name="Valu
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Max_Depth", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

Out[46]: <Axes: xlabel='Max_Depth', ylabel='Values'>

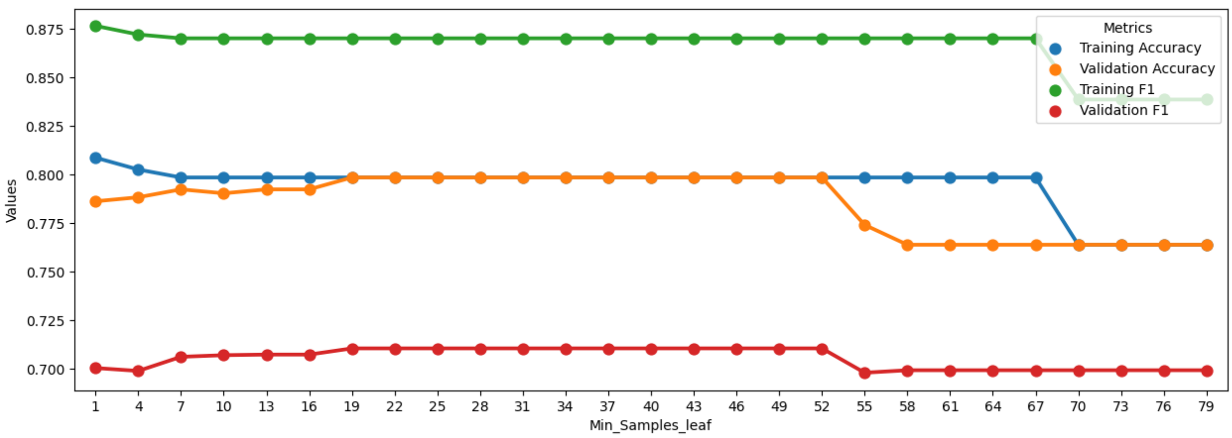


Decision Tree

In [47]:

```python
training_accuracy = []
val_accuracy = []
training_f1 = []
val_f1 = []
min_samples_leaf = []
import numpy as np
for samples_leaf in range(1,80,3): ### Sweeping from 1% samples to 10% samples per
    tree_clf = DecisionTreeClassifier(max_depth=3,min_samples_leaf = samples_leaf)
    tree_clf.fit(X_train,y_train)
    y_training_pred = tree_clf.predict(X_train)

    training_acc = accuracy_score(y_train,y_training_pred)
    train_f1 = f1_score(y_train,y_training_pred)
    val_mean_f1 = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='f1_macro
    val_mean_accuracy = cross_val_score(tree_clf,X_train,y_train,cv=5,scoring='acc

    training_accuracy.append(training_acc)
    val_accuracy.append(val_mean_accuracy)
    training_f1.append(train_f1)
    val_f1.append(val_mean_f1)
    min_samples_leaf.append(samples_leaf)


Tuning_min_samples_leaf = {"Training Accuracy": training_accuracy, "Validation Acc
Tuning_min_samples_leaf_df = pd.DataFrame.from_dict(Tuning_min_samples_leaf)

plot_df = Tuning_min_samples_leaf_df.melt('Min_Samples_leaf',var_name='Metrics',va
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Min_Samples_leaf", y="Values",hue="Metrics", data=plot_df,ax=ax)
```

Out[47]: <Axes: xlabel='Min_Samples_leaf', ylabel='Values'>



In [51]:
```
from sklearn.metrics import confusion_matrix,precision_score,recall_score
tree_clf = DecisionTreeClassifier(max_depth=3,min_samples_leaf = 35)
tree_clf.fit(X_train,y_train)
y_pred = tree_clf.predict(X_test_imp)
print("Test Accuracy: ",accuracy_score(y_test,y_pred))
print("Test Error Rate: ",1-accuracy_score(y_test,y_pred))
print("Test F1 Score: ",f1_score(y_test,y_pred))
print("Test Precision: ",precision_score(y_test,y_pred))
print("Test Recall: ",recall_score(y_test,y_pred))
print("Confusion Matrix on Test Data")
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=Tru
```

```
Test Accuracy:  0.8536585365853658
Test Error Rate:  0.14634146341463417
Test F1 Score:  0.903225806451613
Test Precision:  0.8316831683168316
Test Recall:  0.9882352941176471
Confusion Matrix on Test Data
```

Out[51]:

Predicted	0	1	All
True			
0	21	17	38
1	1	84	85
All	22	101	123

In [52]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

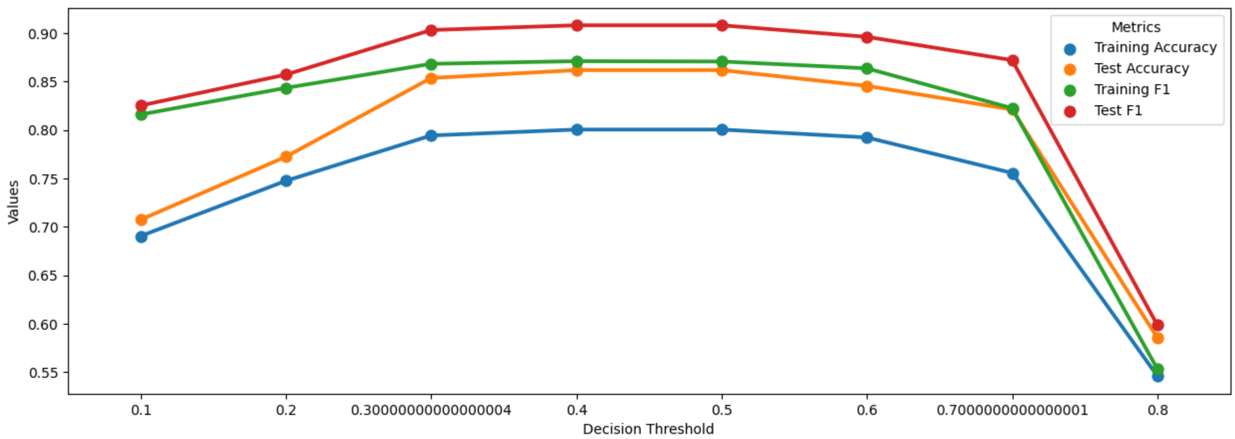
```
              precision    recall  f1-score   support

           0       0.95      0.55      0.70        38
           1       0.83      0.99      0.90        85

    accuracy                           0.85       123
   macro avg       0.89      0.77      0.80       123
weighted avg       0.87      0.85      0.84       123
```

Model 2: Logistic Regression

In [53]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,recall_score,precision_score
from sklearn.model_selection import cross_val_predict

train_accuracies = []
train_f1_scores = []
test_accuracies = []
test_f1_scores = []
thresholds = []

#for thresh in np.linspace(0.1,0.9,8): ## Sweeping from threshold of 0.1 to 0.9
for thresh in np.arange(0.1,0.9,0.1): ## Sweeping from threshold of 0.1 to 0.9
    logreg_clf = LogisticRegression(solver='liblinear')
    logreg_clf.fit(X_train,y_train)

    y_pred_train_thresh = logreg_clf.predict_proba(X_train)[:,1]
    y_pred_train = (y_pred_train_thresh > thresh).astype(int)

    train_acc = accuracy_score(y_train,y_pred_train)
    train_f1 = f1_score(y_train,y_pred_train)

    y_pred_test_thresh = logreg_clf.predict_proba(X_test_imp)[:,1]
    y_pred_test = (y_pred_test_thresh > thresh).astype(int)

    test_acc = accuracy_score(y_test,y_pred_test)
    test_f1 = f1_score(y_test,y_pred_test)

    train_accuracies.append(train_acc)
    train_f1_scores.append(train_f1)
    test_accuracies.append(test_acc)
    test_f1_scores.append(test_f1)
    thresholds.append(thresh)


Threshold_logreg = {"Training Accuracy": train_accuracies, "Test Accuracy": test_a
Threshold_logreg_df = pd.DataFrame.from_dict(Threshold_logreg)

plot_df = Threshold_logreg_df.melt('Decision Threshold',var_name='Metrics',value_n
fig,ax = plt.subplots(figsize=(15,5))
sns.pointplot(x="Decision Threshold", y="Values",hue="Metrics", data=plot_df,ax=ax
```

Out[53]: <Axes: xlabel='Decision Threshold', ylabel='Values'>



In [55]:

```python
thresh = 0.4
y_pred_test_thresh = logreg_clf.predict_proba(X_test_imp)[:,1]
y_pred = (y_pred_test_thresh > thresh).astype(int)
print("Test Accuracy: ",accuracy_score(y_test,y_pred))
print("Test Error Rate: ",1-accuracy_score(y_test,y_pred))
print("Test F1 Score: ",f1_score(y_test,y_pred))
print("Test Precision: ",precision_score(y_test,y_pred))
print("Test Recall: ",recall_score(y_test,y_pred))
print("Confusion Matrix on Test Data")
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=Tru
```

```
Test Accuracy:  0.8617886178861789
Test Error Rate:  0.1382113821138211
Test F1 Score:  0.9081081081081082
Test Precision:  0.84
Test Recall:  0.9882352941176471
Confusion Matrix on Test Data
```

Out[55]:

Predicted	0	1	All
True			
0	22	16	38
1	1	84	85
All	23	100	123

In [56]:

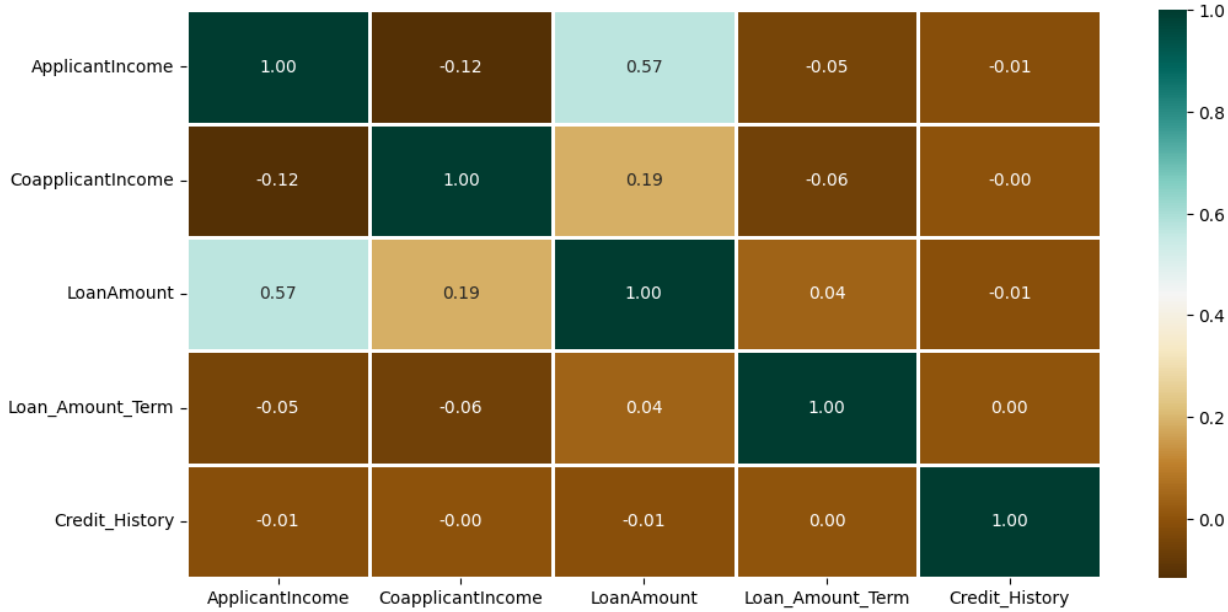
```python
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.58      0.72        38
           1       0.84      0.99      0.91        85

    accuracy                           0.86       123
   macro avg       0.90      0.78      0.81       123
weighted avg       0.88      0.86      0.85       123
```

In [57]:
```python
plt.figure(figsize=(12,6))

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

```
C:\Users\kiran\AppData\Local\Temp\ipykernel_5872\2424999678.py:3: FutureWarning:
The default value of numeric_only in DataFrame.corr is deprecated. In a future v
ersion, it will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  sns.heatmap(train_df.corr(),cmap='BrBG',fmt='.2f',
```

Out[57]: <Axes: >



In [ ]:
```python
# Logistic Regression Confusion matrix is very similar to Decision Tree. In this c
```