Introduction

The main goal of this project is to use Machine Learning techniques to determine whether a loan should be approved or not based on the past information of a person. This project includes:

1. Data Cleaning
2. Data Visualizations
3. Transforming data
4. Identifying outliers
5. Model Evaluations.

The libraries used in this project are:

1. sklearn
2. matplotlib
3. numpy
4. pandas
5. seaborn

There are different models to train your data, here we will be using:

1. logistic regression
2. decision trees
3. random forest
4. Hyperparameter Tuning method

Dataset

This dataset is named Loan Prediction Dataset data set. The dataset contains 613 records and attributes: Loan_ID, Gender, Married, Dependents, Education, Self_Employed, Applicant Income, Co-applicant Income, Loan Amount, Loan Amount Term, Credit History, Property Area , and Loan_Status.

Libraries

In [1]:
```
import os #paths to file
import numpy as np # linear algebra
import pandas as pd # data processing
import warnings# warning filter


#ploting libraries
import matplotlib.pyplot as plt
import seaborn as sns
import warnings# warning filter

#Machine learning libraries
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
warnings.filterwarnings("ignore")
```

Load the datasets

In [2]:
```
#load the dataset
#training set
#download the data to run the report
tr_df = pd.read_csv(r"/Users/mirandacheng7/Downloads/Loan Prediction/train_u6luj
#testing set
te_df= pd.read_csv(r"/Users/mirandacheng7/Downloads/Loan Prediction/test_Y3wMUE5
```

Processing the dataset

Take a look at the datesets

Training set:

In [3]:
```
#display the first 5 rows of the training set
tr_df.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
4	LP001008	Male	No	0	Graduate	No	6000	

Testing set:

In [4]:
```
#display the first 5 rows of the testing set
te_df.head()
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappli
0	LP001015	Male	Yes	0	Graduate	No	5720	
1	LP001022	Male	Yes	1	Graduate	No	3076	
2	LP001031	Male	Yes	2	Graduate	No	5000	
3	LP001035	Male	Yes	2	Graduate	No	2340	
4	LP001051	Male	No	0	Not Graduate	No	3276	

Size of each data set:

In [5]:
```
#print the size of each dataset
print(f"training set (row, col): {tr_df.shape}\n\ntesting set (row, col): {te_df
```

```
training set (row, col): (614, 13)

testing set (row, col): (367, 12)
```

Data Cleaning

Find the missing values

In [6]:
```
tr_df.isnull().sum()
```

Out[6]:
```
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
```

Fill the missing values

As Gender, Married, credit_history, and self_employed are categorical data, we will replace the missing value with the most requent value.

In [7]:
```
#filling the missing data with mode
```

```
null_cols = ['Credit_History', 'Self_Employed', 'LoanAmount','Dependents', 'Loan

for col in null_cols:
    tr_df[col] = tr_df[col].fillna(tr_df[col].dropna().mode().values[0])

tr_df.isnull().sum().sort_values
```

Out[7]:
```
<bound method Series.sort_values of 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>
```

In [8]:
```
#check if there are any duplicates
tr_df.duplicated().any()
```

Out[8]: False

In [9]:
```
#remove the id column for both datasets as it's not needed
tr_df.drop('Loan_ID',axis=1,inplace=True)
te_df.drop('Loan_ID',axis=1,inplace=True)

#print the size of each dataset
print(f"training set (row, col): {tr_df.shape}\n\ntesting set (row, col): {te_df
```

```
training set (row, col): (614, 12)

testing set (row, col): (367, 11)
```

Data visalization

First, let's split data into categorical and numberical data. For categorical data, we want to show counts in each categorical bin using bars, for numberic data, we want to see the distribution.

In [10]:
```
#categorical columns
cat = tr_df.select_dtypes('object').columns.to_list()

#numerical columns
num = tr_df.select_dtypes('number').columns.to_list()

#numberical data
loan_num =  tr_df[num]
#categorical df
loan_cat = tr_df[cat]
```

In [11]:
```
loan_cat
```

Out[11]:

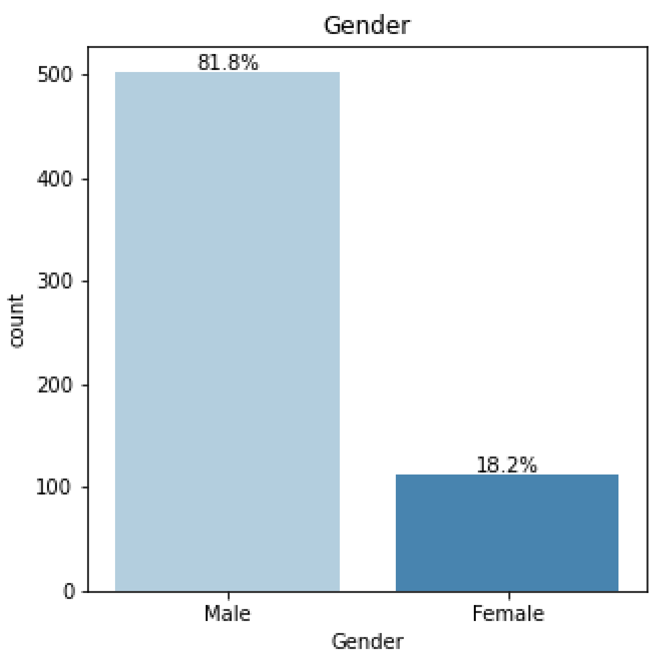
	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
0	Male	No	0	Graduate	No	Urban	Y

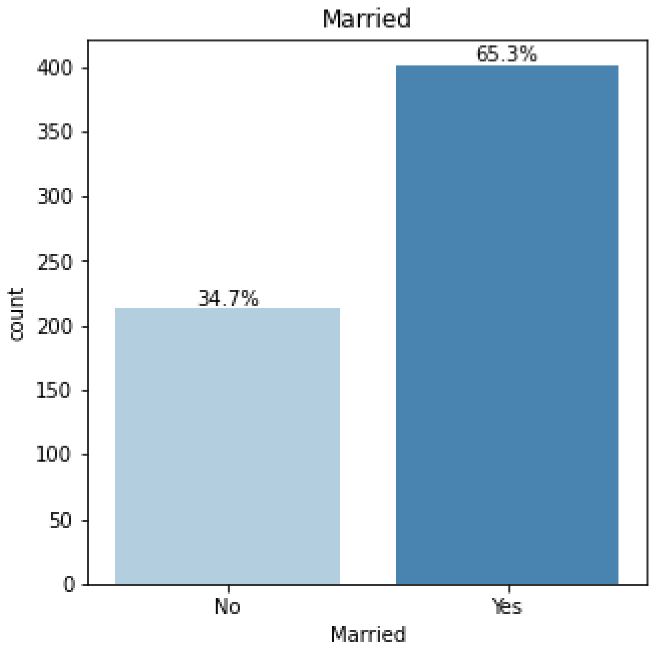
	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loan_Status
1	Male	Yes	1	Graduate	No	Rural	N
2	Male	Yes	0	Graduate	Yes	Urban	Y
3	Male	Yes	0	Not Graduate	No	Urban	Y
4	Male	No	0	Graduate	No	Urban	Y
...	...	...	...	...	...	...	...
609	Female	No	0	Graduate	No	Rural	Y
610	Male	Yes	3+	Graduate	No	Rural	Y
611	Male	Yes	1	Graduate	No	Urban	Y
612	Male	Yes	2	Graduate	No	Urban	Y
613	Female	No	0	Graduate	Yes	Semiurban	N

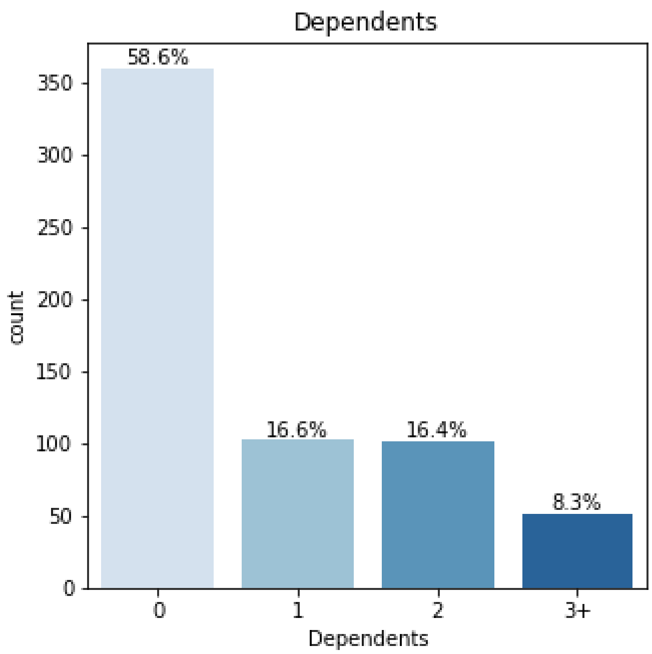
614 rows × 7 columns

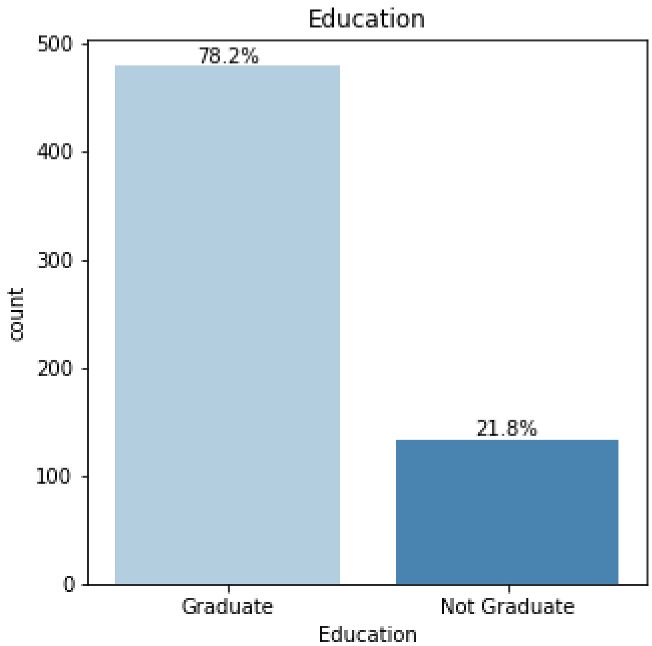
In [12]:

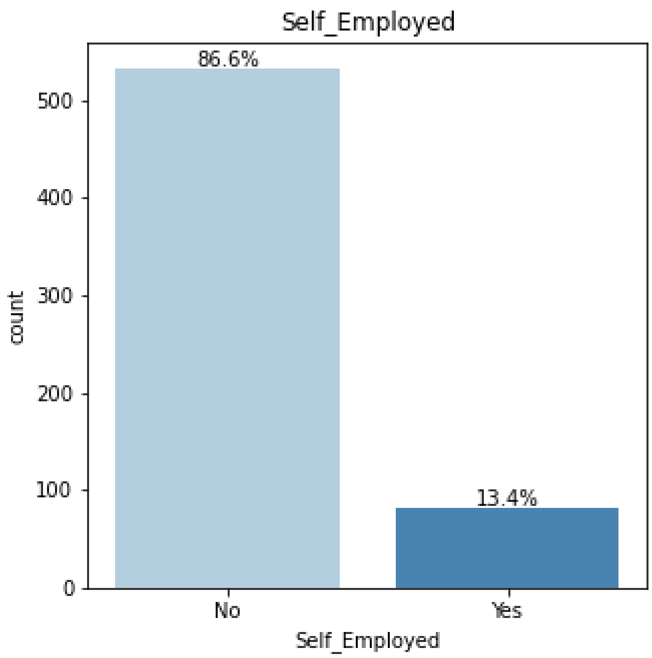
```
#display the counts of observations using bars for each categorical column
for i in loan_cat:
    plt.figure(figsize=(5,5))
    total = float(len(loan_cat[i]))
    ax = sns.countplot(loan_cat[i],palette='Blues')
    for p in ax.patches:
        height = p.get_height()
        ax.text(p.get_x()+p.get_width()/2,height + 3,'{:.1f}%'.format(height/tot
    ax.set_title(i)
    plt.show()
```

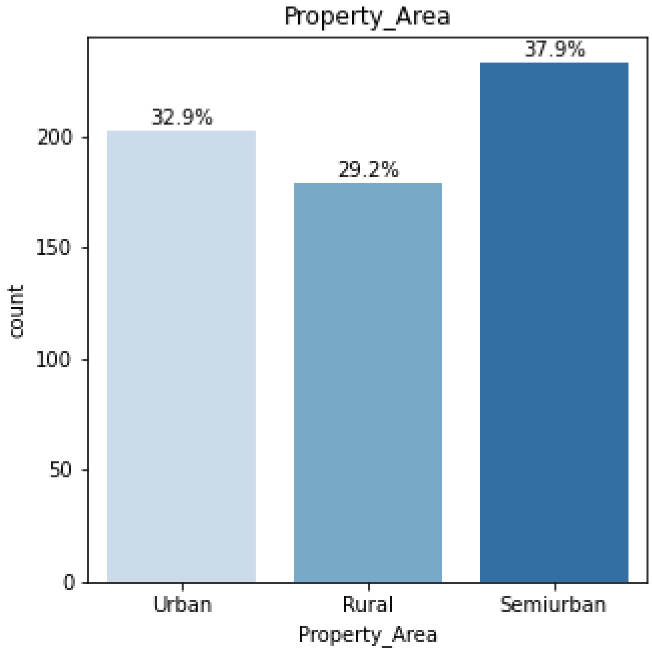


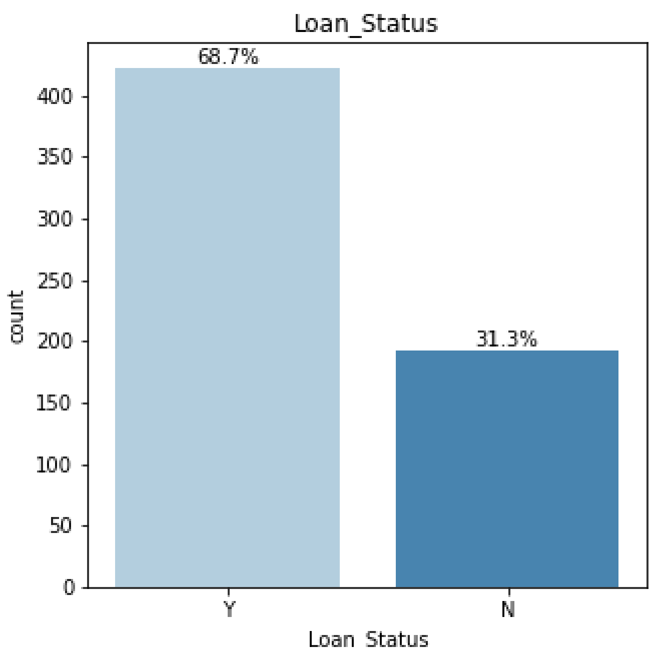

Married
65.3%
34.7%
count
400
350
300
250
200
150
100
50
0
No
Yes
Married


Dependents
58.6%
16.6%
16.4%
8.3%
count
350
300
250
200
150
100
50
0
0
1
2
3+
Dependents


Education
78.2%
21.8%
count
0
100
200
300
400
500
Graduate
Not Graduate
Education

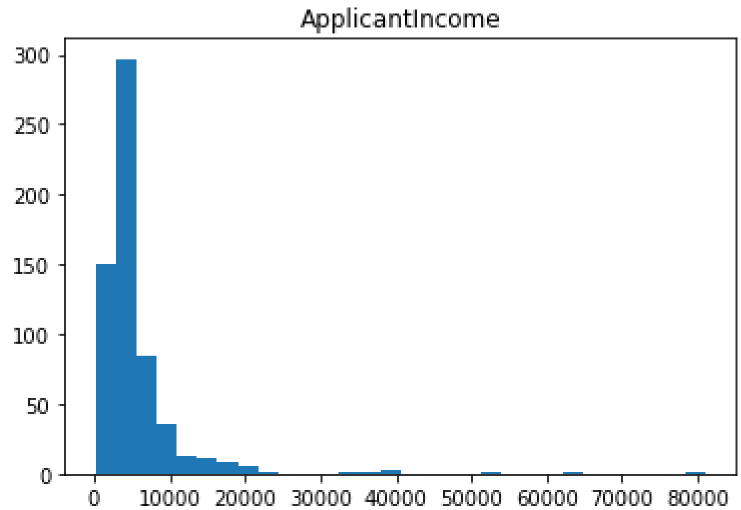

Self_Employed
86.6%
13.4%
count
0
100
200
300
400
500
No
Yes
Self_Employed

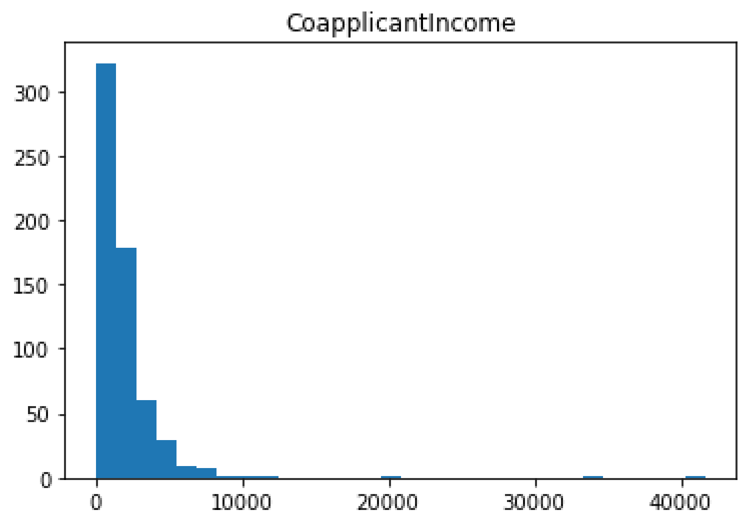


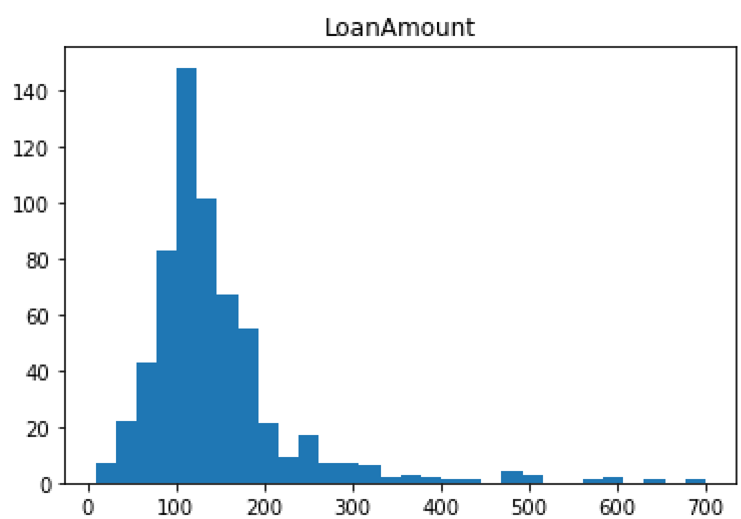


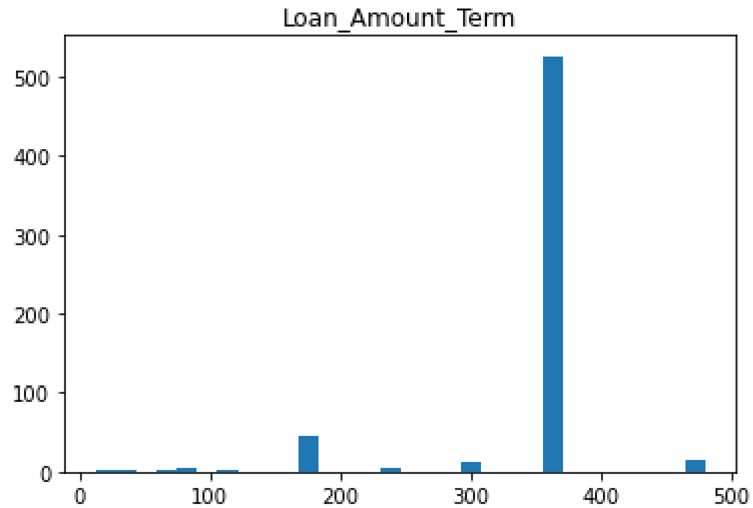
In [28]:

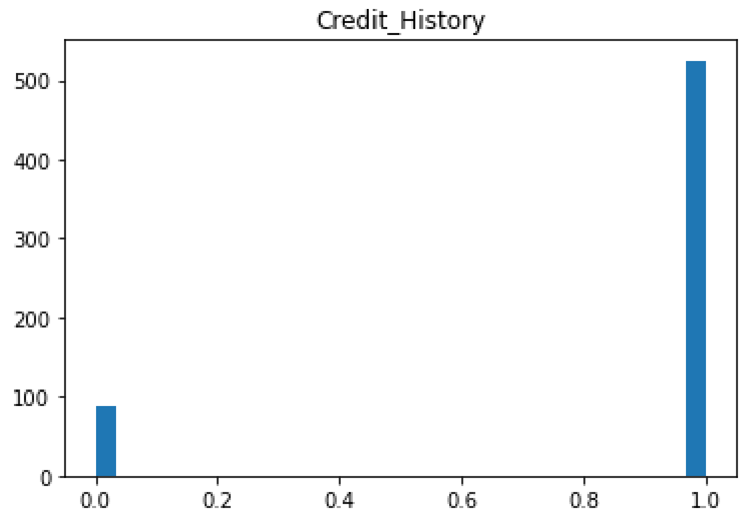
```
#display the distribution of each numerical column
for i in loan_num:
    plt.hist(loan_num[i], bins=30)
    plt.title(i)
    plt.show()
```


ApplicantIncome
300
250
200
150
100
50
0
0
10000
20000
30000
40000
50000
60000
70000
80000


CoapplicantIncome
300
250
200
150
100
50
0
0
10000
20000
30000
40000

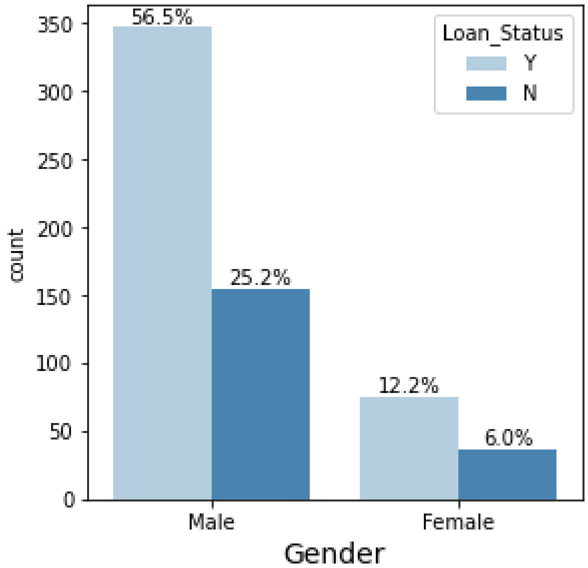

LoanAmount
140
120
100
80
60
40
20
0
0
100
200
300
400
500
600
700



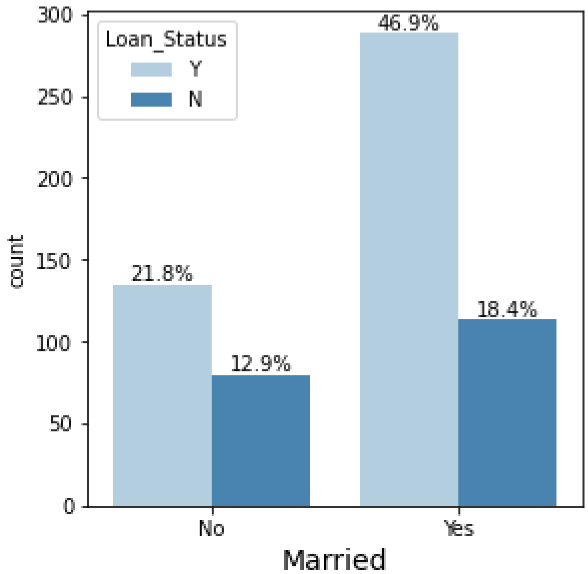


Display categorical data by Loan status.

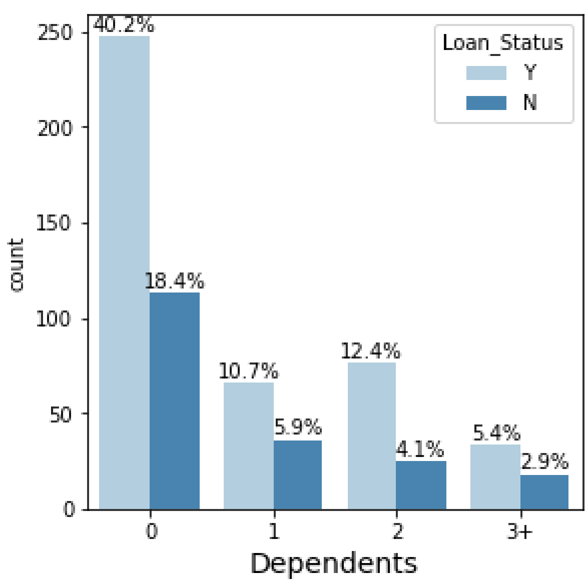
In [14]:
```
for i in cat[:-1]:
    plt.figure(figsize=(15,10))
    total = float(len(loan_cat[i]))
    plt.subplot(2,3,1)
    ax=sns.countplot(x=i ,hue='Loan_Status', data=tr_df ,palette='Blues')
    plt.xlabel(i, fontsize=14)
    for p in ax.patches:
        height = p.get_height()
        ax.text(p.get_x()+p.get_width()/2,height + 3,'{:.1f}%'.format(height/tot
```



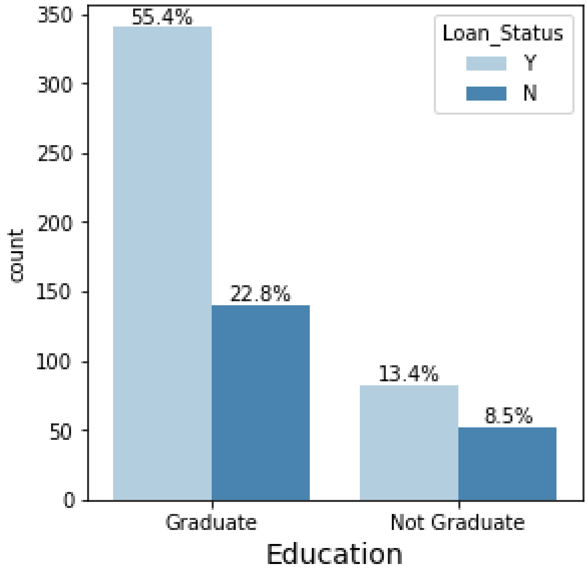
Loan_Status
Y
N
56.5%
25.2%
12.2%
6.0%
count
0
50
100
150
200
250
300
350
Male
Female
Gender

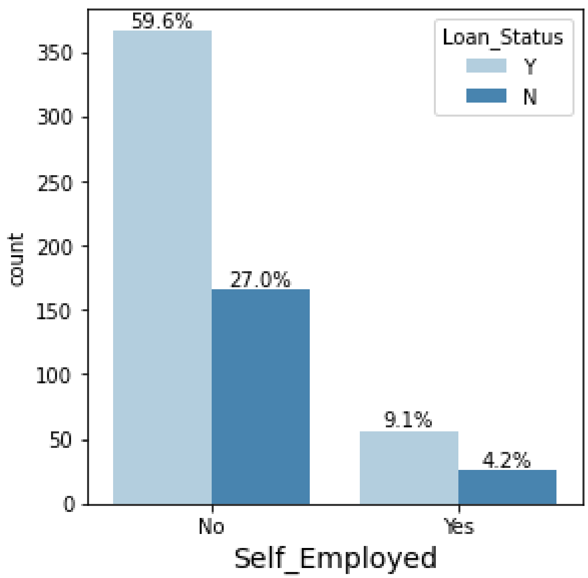


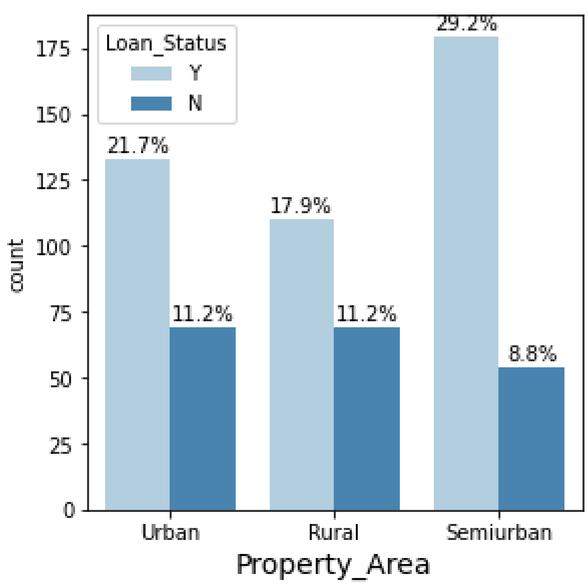
Loan_Status
Y
N
21.8%
12.9%
46.9%
18.4%
count
0
50
100
150
200
250
300
No
Yes
Married



Loan_Status
Y
N
40.2%
18.4%
10.7%
5.9%
12.4%
4.1%
5.4%
2.9%
count
0
50
100
150
200
250
0
1
2
3+
Dependents







Encoding data to numeric

change categorical data into numeric format

In [16]:
```
from sklearn.preprocessing import LabelEncoder
cols = ["Gender", "Married", "Education", "Self_Employed","Property_Area","Loan_
le = LabelEncoder()
for col in cols:
    tr_df[col] = le.fit_transform(tr_df[col].astype(str))
```

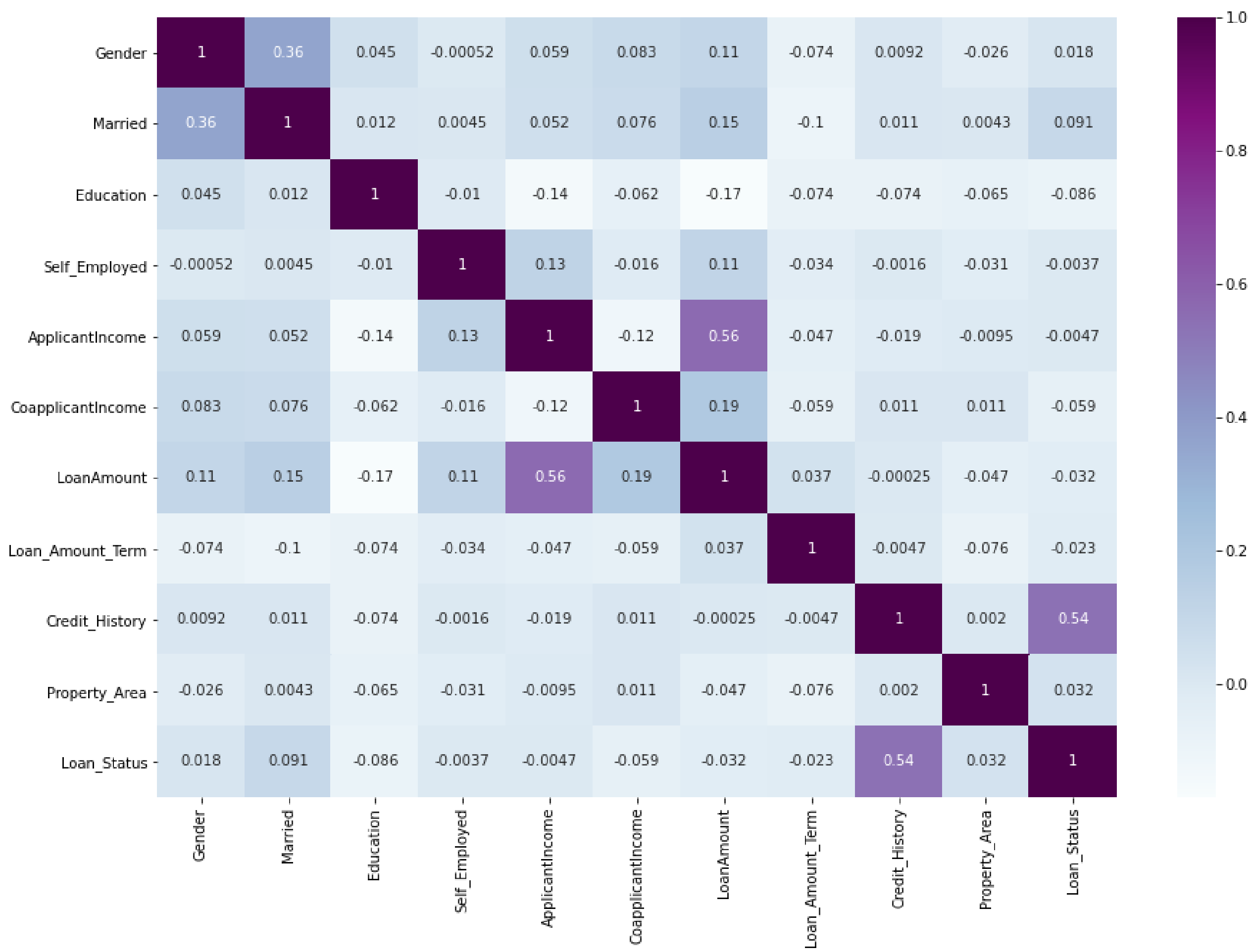
In [17]:
```
tr_df['Dependents'].value_counts()
```

Out[17]:
```
0      360
1      102
2      101
3+      51
Name: Dependents, dtype: int64
```

In [18]:
```
# As 3+ in Dependents column has not been changed to numberic, so we should repl
tr_df['Dependents'] = np.where((tr_df.Dependents == '3+'), 3, tr_df.Dependents)
```

In [19]:
```
#plotting the correlation matrix
plt.figure(figsize=(15,10))
sns.heatmap(tr_df.corr(), annot = True, cmap='BuPu')
```

Out[19]:
```
<AxesSubplot:>
```



Train-Test Split

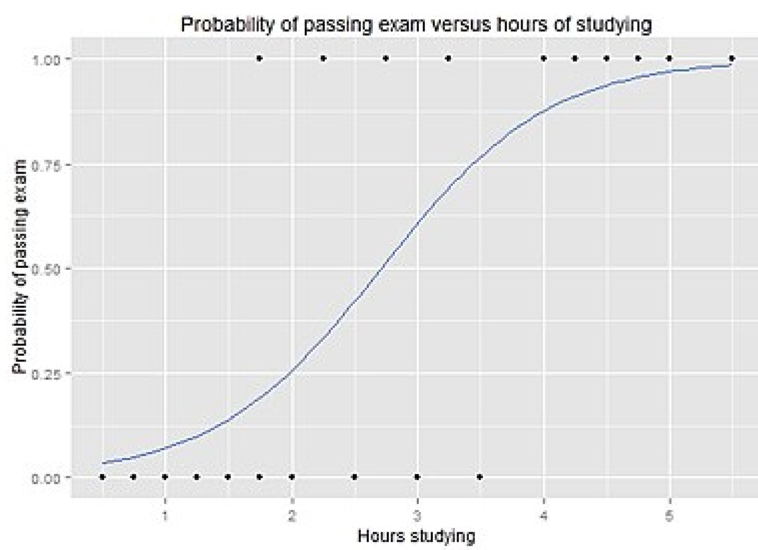
In [20]:
```
X= tr_df.drop(columns = ['Loan_Status'], axis = 1)
y = tr_df['Loan_Status']
```

In [21]:
```
#Split the data into train-test split:

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, rand

print(f"X_training set (row, col): {X_train.shape}\n\ny_train (row, col): {y_tra
```

```
X_training set (row, col): (460, 11)

y_train (row, col): (460,)

X_test set (row, col): (154, 11)

y_test set (row, col): (154,)
```

Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. (Wikipedia)



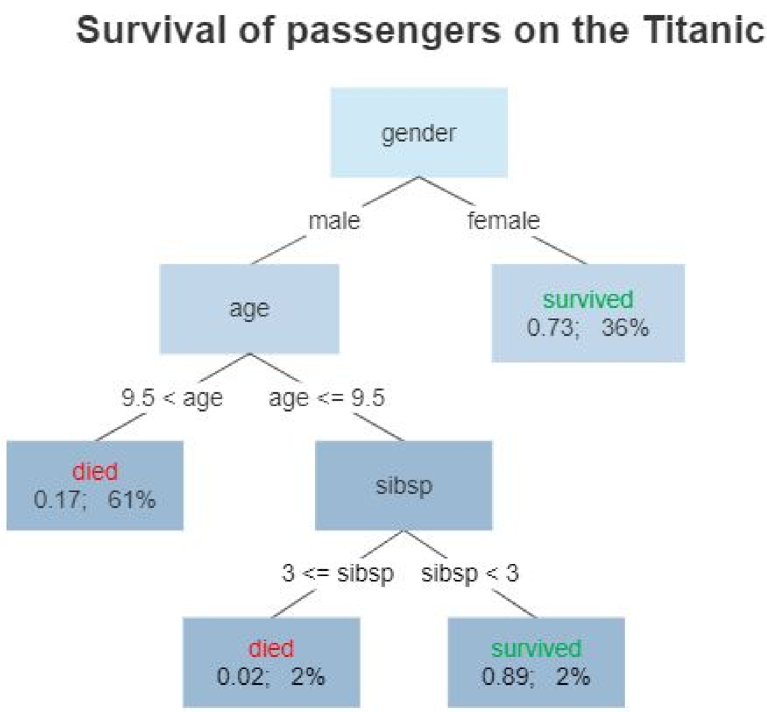
In [22]:
```
LR = LogisticRegression()
LR.fit(X_train, y_train)


y_predict = LR.predict(X_test)

print(classification_report(y_test, y_predict))

# print out the accuracy score
LR_SC = accuracy_score(y_predict,y_test)
print(f"{round(LR_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.91      0.39      0.55        54
           1       0.75      0.98      0.85       100

    accuracy                           0.77       154
   macro avg       0.83      0.68      0.70       154
weighted avg       0.81      0.77      0.74       154
```

```
77.27% Accurate
```

Decision Tree

Decision Trees are constructed by splitting a data set based on different conditions and the goal is to create a model that predicts the value of a target variable by learning simple decisions inferred from the data features.



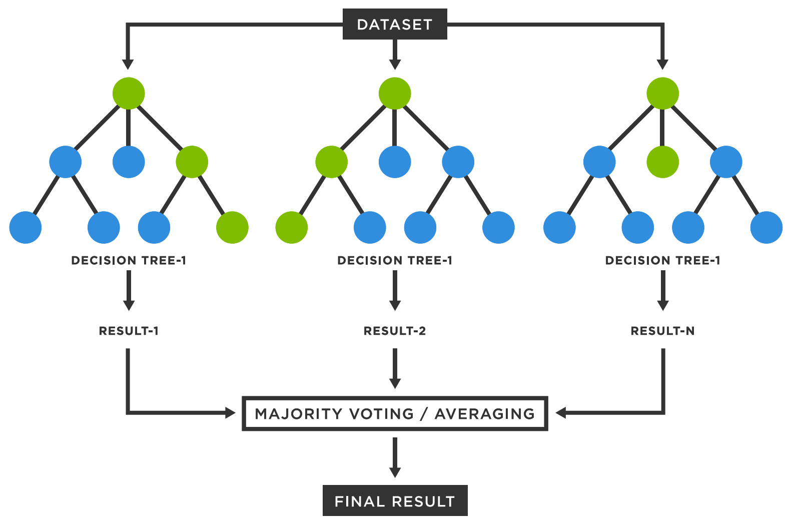
In [23]:

```
DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)

y_predict = DT.predict(X_test)

#prediction summary
print(classification_report(y_test, y_predict))

# print out the accuracy score
DT_SC = accuracy_score(y_predict,y_test)
print(f"{round(DT_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.53      0.44      0.48        54
           1       0.72      0.79      0.76       100

    accuracy                           0.67       154
   macro avg       0.63      0.62      0.62       154
weighted avg       0.66      0.67      0.66       154

66.88% Accurate
```

Random Forest

Random forest is an ensemble of decision trees that are trained with a combination of learning models to increase the overall results. Random forest builds multiple decisions trees and combines them together to get a better prediction.



In [24]:

```
RF = RandomForestClassifier()
RF.fit(X_train, y_train)

y_predict = RF.predict(X_test)

#  prediction Summary
print(classification_report(y_test, y_predict))

# print out accuracy score
RF_SC = accuracy_score(y_predict,y_test)
print(f"{round(RF_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.88      0.43      0.57        54
           1       0.76      0.97      0.85       100

    accuracy                           0.78       154
   macro avg       0.82      0.70      0.71       154
weighted avg       0.80      0.78      0.75       154

77.92% Accurate
```

Hyperparameter tuning

In [25]:

```
RF_tuning = RandomForestClassifier(n_estimators= 70,min_samples_split=25,max_dep
RF_tuning.fit(X_train, y_train)

y_predict = RF.predict(X_test)

#  prediction Summary
```

```
print(classification_report(y_test, y_predict))

# print out the accuracy score
RF_tuning_SC = accuracy_score(y_predict,y_test)
print(f"{round(RF_tuning_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.88      0.43      0.57        54
           1       0.76      0.97      0.85       100

    accuracy                           0.78       154
   macro avg       0.82      0.70      0.71       154
weighted avg       0.80      0.78      0.75       154

77.92% Accurate
```

In [26]:
```
score = [DT_SC,RF_SC,RF_tuning_SC,LR_SC]
Models = pd.DataFrame({
    'Model': ["Decision Tree","Random Forest","Hyperparameter Tunning", "Logisti
    'Accuracy Score': score})
Models.sort_values(by='Accuracy Score', ascending=False)
```

Out[26]:

	Model	Accuracy Score
1	Random Forest	0.779221
2	Hyperparameter Tunning	0.779221
3	Logistic Regression	0.772727
0	Decision Tree	0.668831

Conclusions

1. Loan Status is the most dependent on credit history as they have a high correlation ratio
2. The random forest after using hyperparameter tuning is the most accurate as modifying parameters could achieve an optimal model architecture