Loan-prediction

December 19, 2022

0.0.1 Importing the libraries

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set_style('darkgrid')
import warnings
warnings.filterwarnings('ignore')
```

0.0.2 Loading the train and test dataset

[ ]:
```
train = pd.read_csv('/content/train_u6lujuX_CVtuZ9i.csv')
test = pd.read_csv('/content/test_Y3wMUE5_7gLdaTN.csv')
```

[ ]:
```
train.head()
```

[ ]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No

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

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
```

```
3            1.0        Urban            Y
4            1.0        Urban            Y
```

[ ]:
```
print('Train shape:',train.shape)
```

```
Train shape: (614, 13)
```

[ ]:
```
test.head()
```

[ ]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001015   Male     Yes          0      Graduate            No
1  LP001022   Male     Yes          1      Graduate            No
2  LP001031   Male     Yes          2      Graduate            No
3  LP001035   Male     Yes          2      Graduate            No
4  LP001051   Male      No          0  Not Graduate            No

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5720                  0       110.0             360.0
1             3076               1500       126.0             360.0
2             5000               1800       208.0             360.0
3             2340               2546       100.0             360.0
4             3276                  0        78.0             360.0

   Credit_History Property_Area
0             1.0         Urban
1             1.0         Urban
2             1.0         Urban
3             NaN         Urban
4             1.0         Urban
```

[ ]:
```
print('Test shape:',test.shape)
```

```
Test shape: (367, 12)
```

0.0.3 Joining of the train dataset and test dataset for proper analysis

[ ]:
```
df = pd.concat([train,test],axis='rows')
df.head()
```

[ ]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
0  LP001002   Male      No          0      Graduate            No
1  LP001003   Male     Yes          1      Graduate            No
2  LP001005   Male     Yes          0      Graduate           Yes
3  LP001006   Male     Yes          0  Not Graduate            No
4  LP001008   Male      No          0      Graduate            No
```

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

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

[ ]:
```
#Shape
df.shape
```

[ ]: (981, 13)

[ ]:
```
#description
df.describe()
```

[ ]:
```
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       981.000000         981.000000  954.000000        961.000000
mean       5179.795107        1601.916330  142.511530        342.201873
std        5695.104533        2718.772806   77.421743         65.100602
min           0.000000           0.000000    9.000000          6.000000
25%        2875.000000           0.000000  100.000000        360.000000
50%        3800.000000        1110.000000  126.000000        360.000000
75%        5516.000000        2365.000000  162.000000        360.000000
max       81000.000000       41667.000000  700.000000        480.000000

       Credit_History
count      902.000000
mean         0.835920
std          0.370553
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

[ ]:
```
#columns
df.columns
```

[ ]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
```

```
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

[ ]:
```
#INFO
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 981 entries, 0 to 366
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            981 non-null    object
 1   Gender             957 non-null    object
 2   Married            978 non-null    object
 3   Dependents         956 non-null    object
 4   Education          981 non-null    object
 5   Self_Employed      926 non-null    object
 6   ApplicantIncome    981 non-null    int64
 7   CoapplicantIncome  981 non-null    float64
 8   LoanAmount         954 non-null    float64
 9   Loan_Amount_Term   961 non-null    float64
 10  Credit_History     902 non-null    float64
 11  Property_Area      981 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 107.3+ KB
```

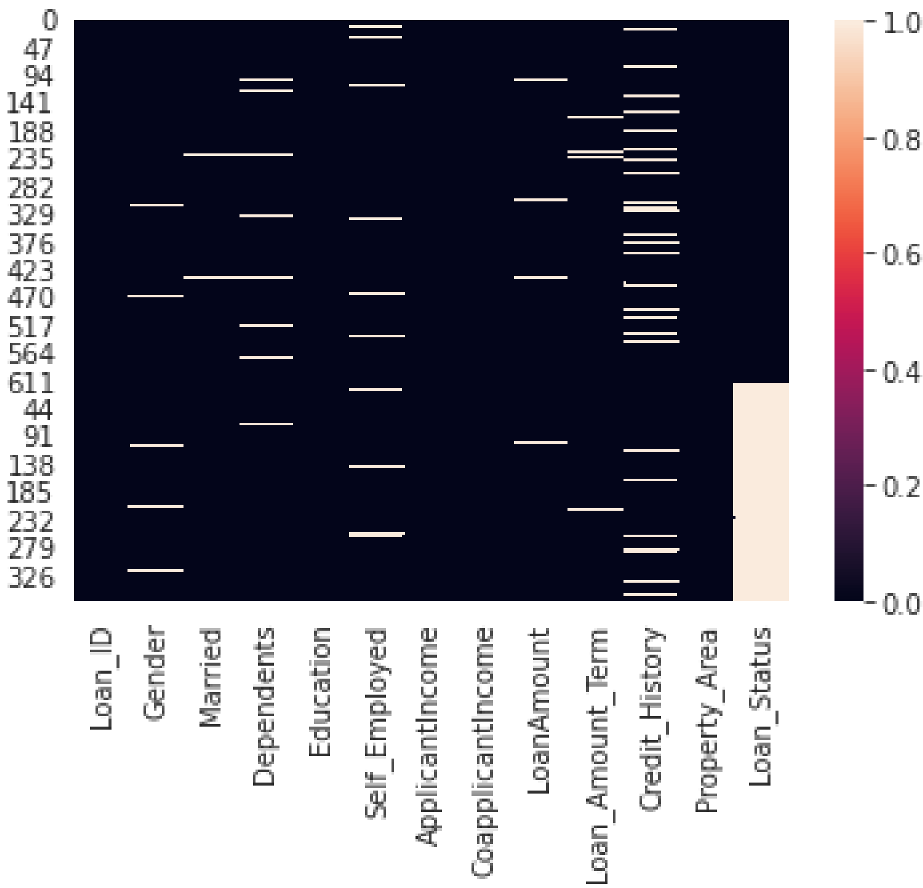
[ ]:
```
#Lets check for the null values
df.isnull().sum()
```

[ ]:
```
Loan_ID                0
Gender                24
Married                3
Dependents            25
Education              0
Self_Employed         55
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount            27
Loan_Amount_Term      20
Credit_History        79
Property_Area          0
Loan_Status          367
dtype: int64
```

0.1 Missing/Null data

[ ]:
```
#Isnull visualization
sns.heatmap(df.isnull())
```

[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9ab938a610>



[ ]:
```
df.duplicated().sum()
```

[ ]: 0

No duplicated values

0.2 Data Preprocessing

[ ]:
```
df['Total_Income'] = df['ApplicantIncome'] + df['CoapplicantIncome']
df['Total_Loan'] = df['LoanAmount'] + df['Loan_Amount_Term']
```

Putting some variables into one variable

[ ]:
```
data_df = df
```

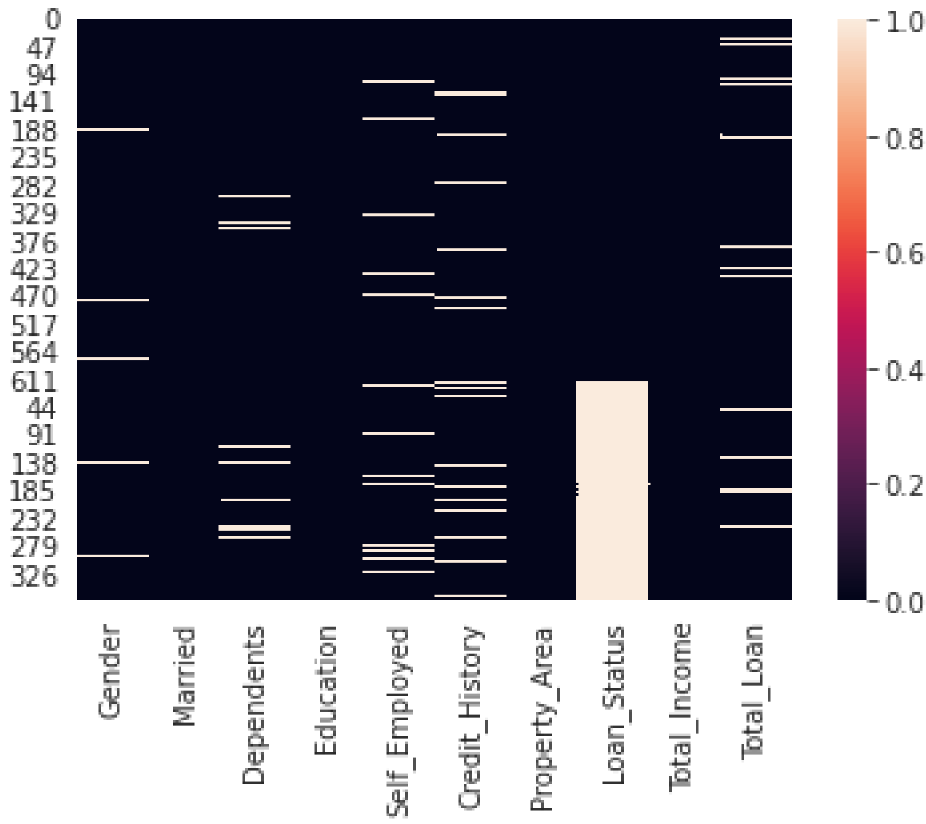
Dropping some variables in the dataset

[ ]:
```
df.drop(['Loan_ID'],axis='columns',inplace=True)
data_df.drop(['ApplicantIncome'],axis='columns',inplace=True)
data_df.drop(['CoapplicantIncome'],axis='columns',inplace=True)
data_df.drop(['LoanAmount'],axis='columns',inplace=True)
data_df.drop(['Loan_Amount_Term'],axis='columns',inplace=True)
```

The Loan_ID column is dropped because it is of no use to that analysis. The dependent is taking care of by fixing the wrong input data of 3+.

[ ]:
```
sns.heatmap(data_df.isnull())
```

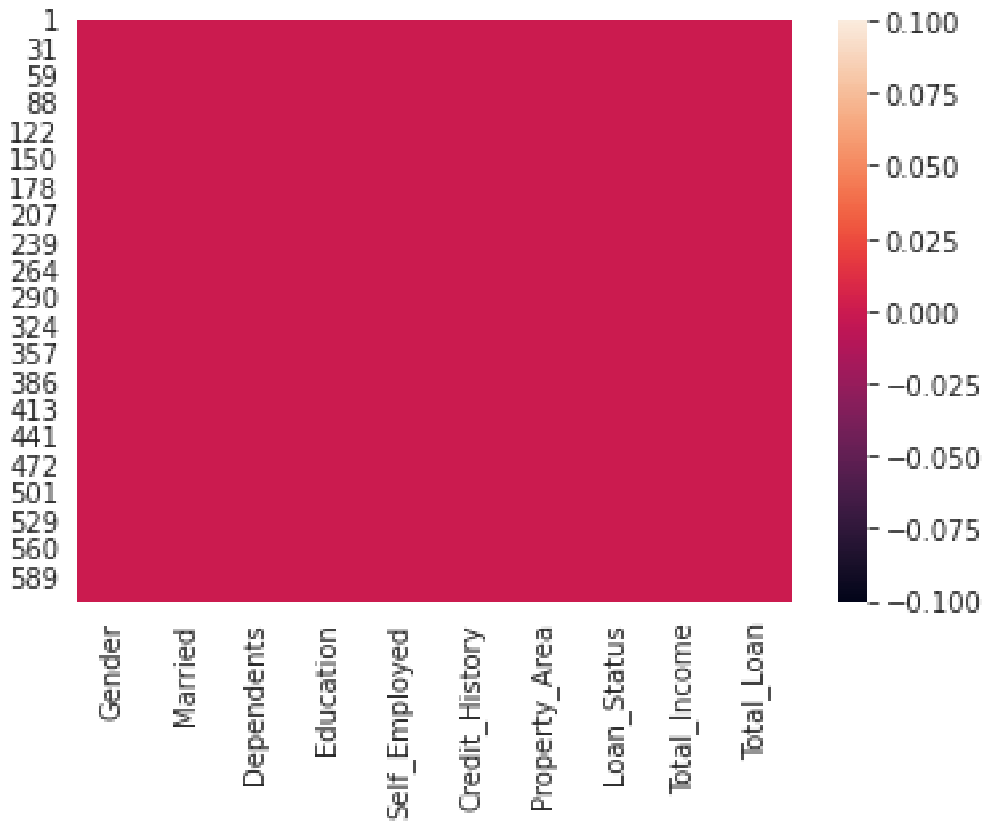
[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ab93cce20>
```



[ ]:
```python
data_df.dropna(inplace=True)
```

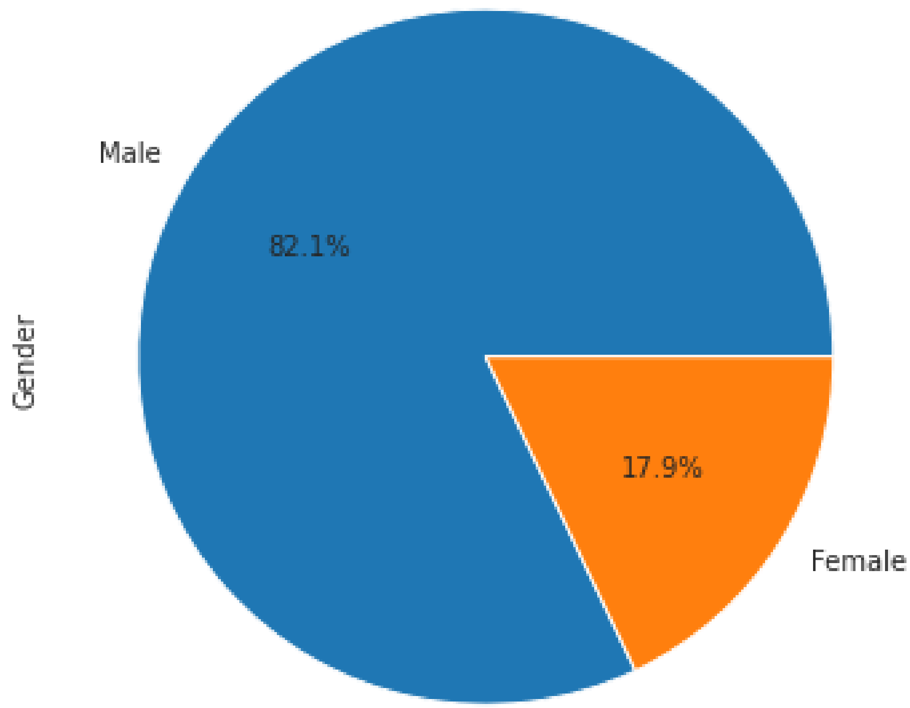
[ ]:
```python
sns.heatmap(data_df.isnull())
```

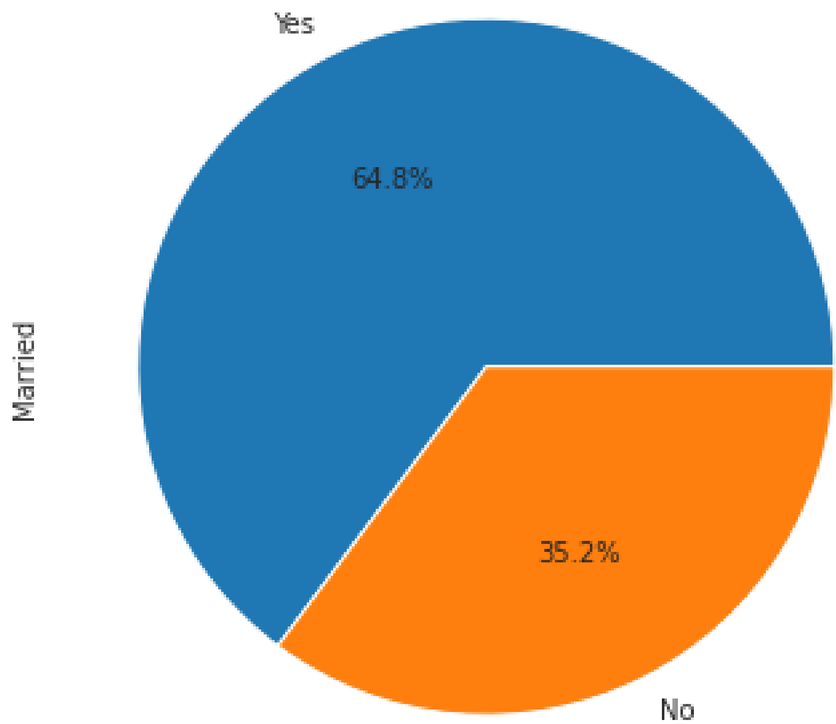
[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9ab651cdc0>

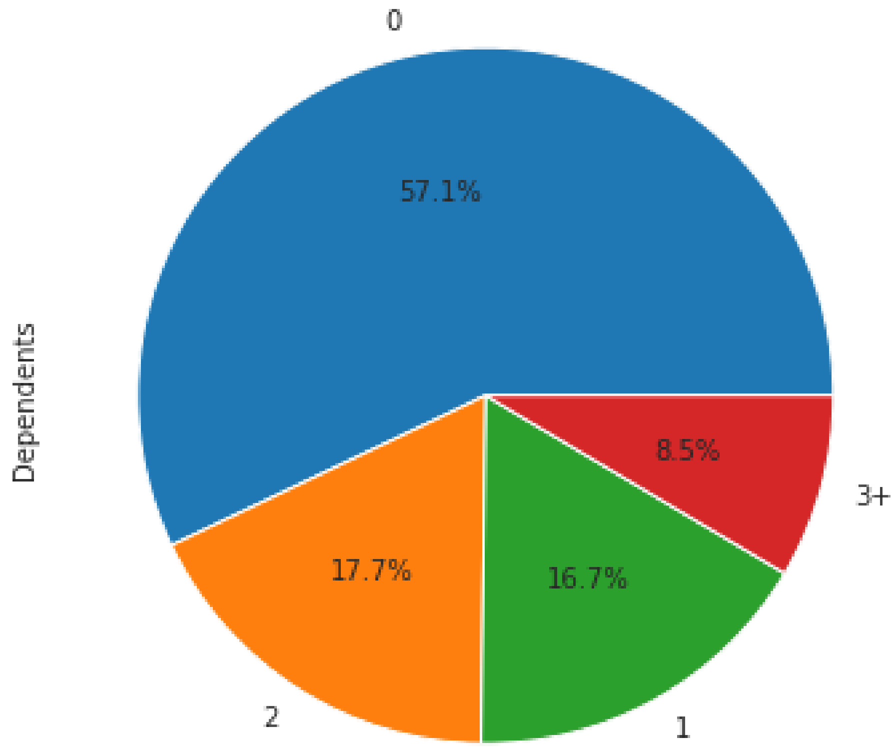


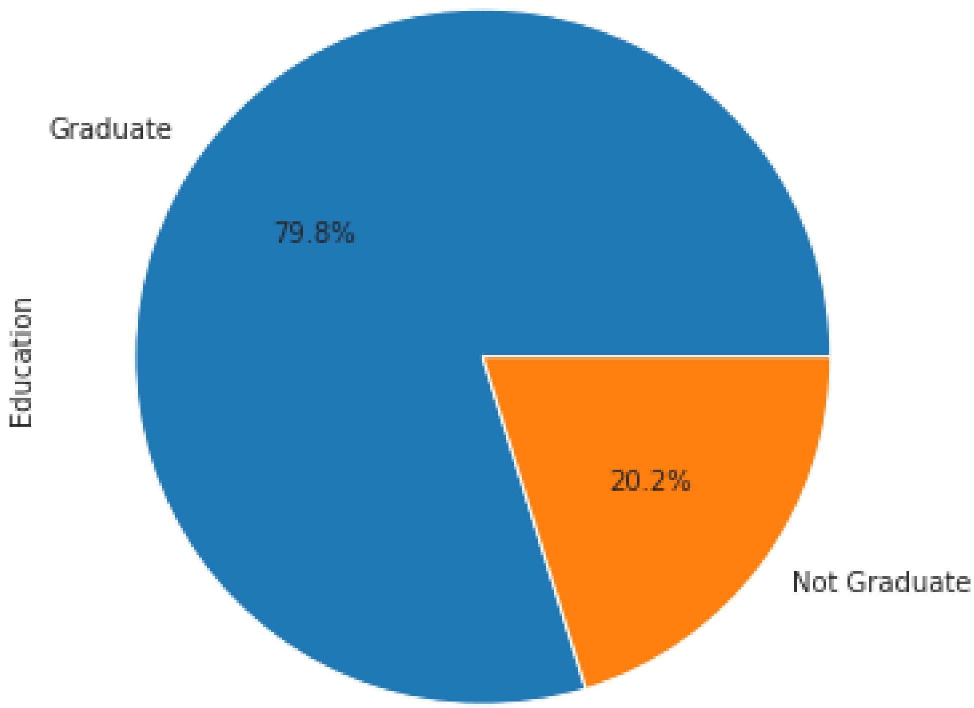
0.2.1 EXPLORATORY DATA ANALYSIS

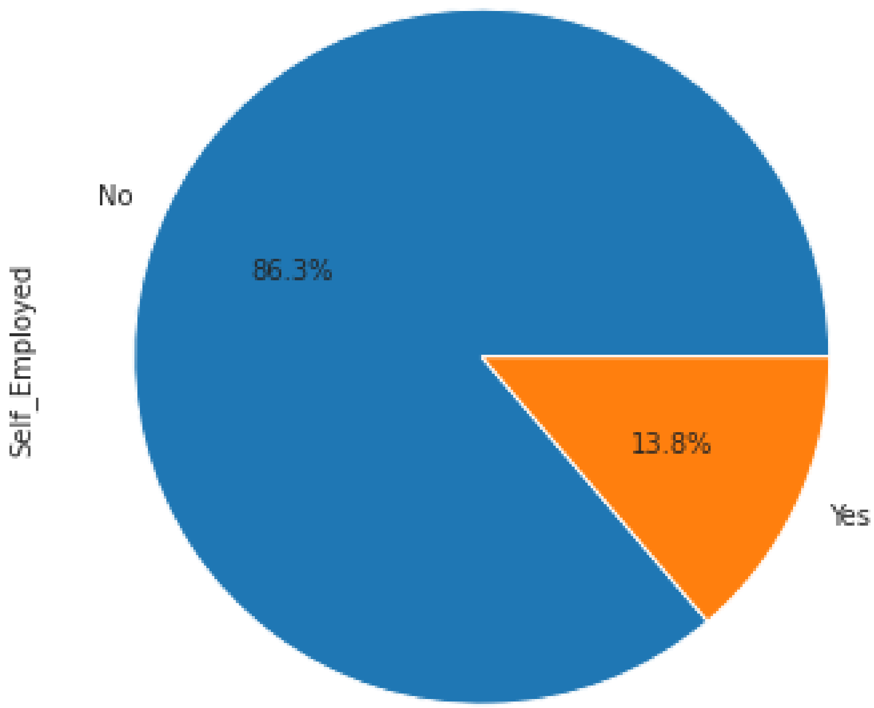
[ ]:
```python
for i in data_df.columns:
    plt.figure(figsize=(15,6))
    data_df[i].value_counts().plot(kind='pie',autopct='%1.1f%%')
```

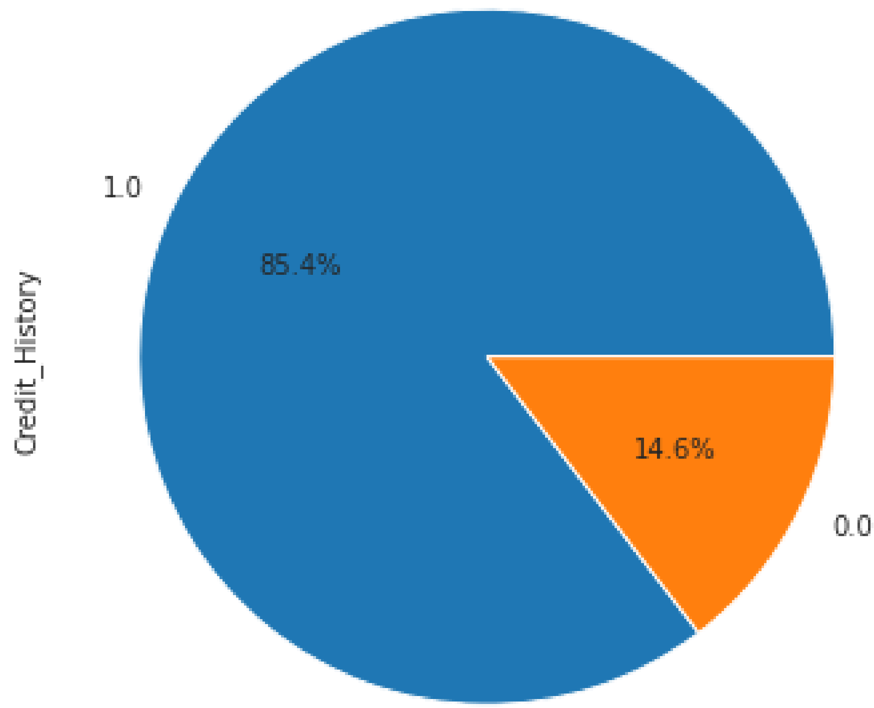

Gender
Male
82.1%
17.9%
Female

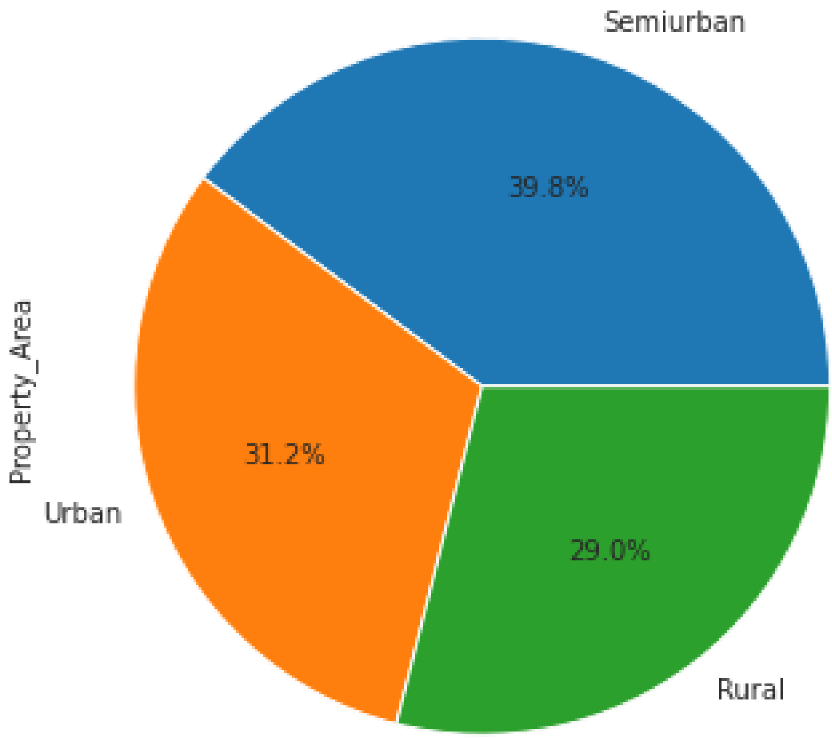

Yes
64.8%
Married
35.2%
No

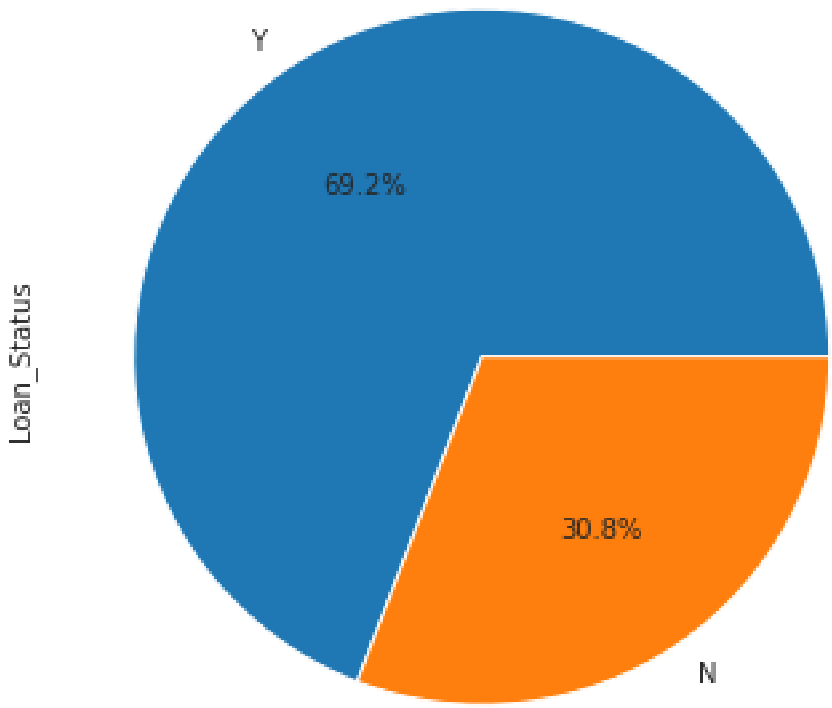

0
57.1%
Dependents
8.5%
3+
17.7%
16.7%
2
1

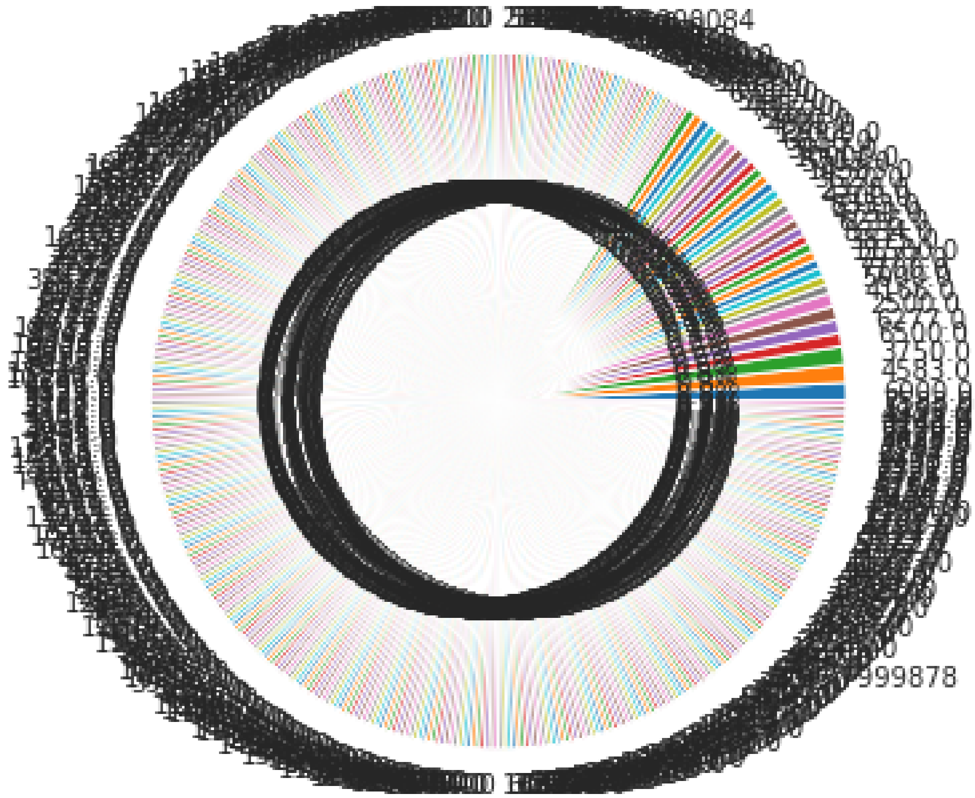

Graduate
79.8%
Education
20.2%
Not Graduate

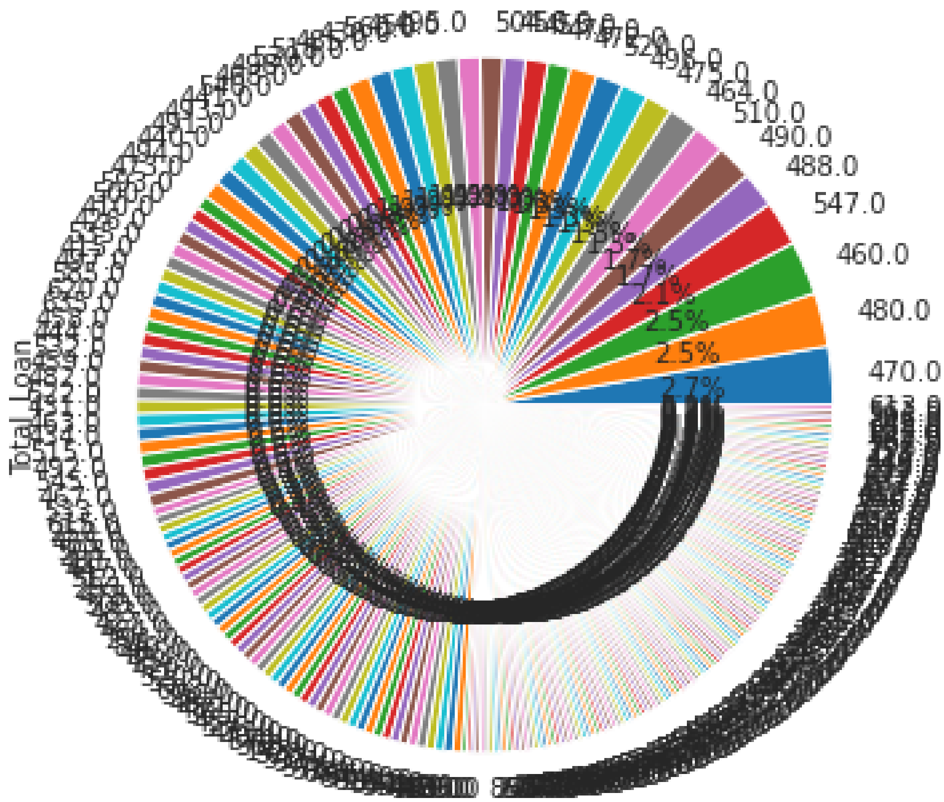

Self_Employed
No
86.3%
13.8%
Yes


Credit_History
1.0
85.4%
14.6%
0.0


Semiurban
39.8%
Property_Area
31.2%
Urban
29.0%
Rural


Y
69.2%
Loan_Status
30.8%
N





0.2.2 Report

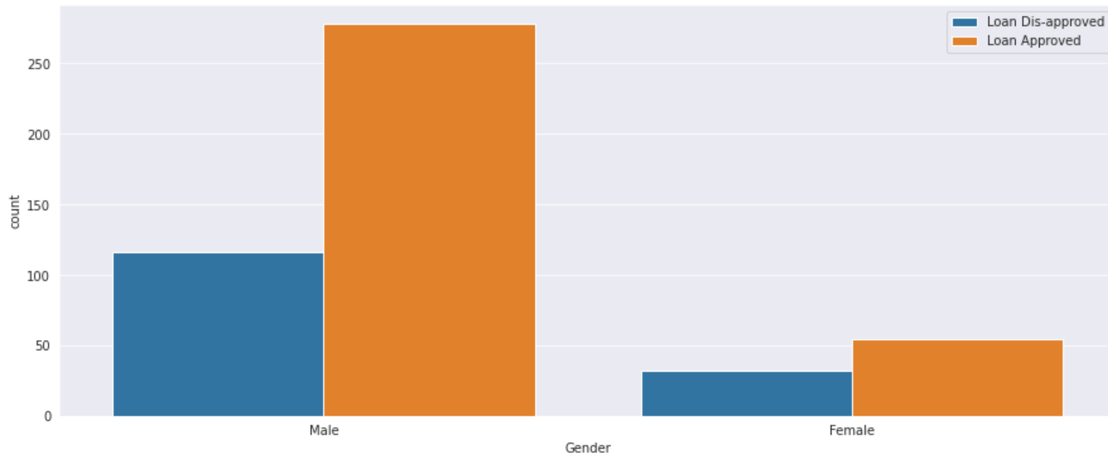
1. Male have the highest ratio of getting a loan approved (82.1%) than female(17.9)
2. Those who are married (64.8%) will have their loan approved compared to those not married (35.2%)
3. Those who are not dependent will have their loan approved (57.1%)
4. Graduate (79.8%) will have their loan approved while Not Graduate (20.2%)
5. People who have a Government work will be approved of loan (13.8%) and self-employed will not have their loan approved
6. Those with good credit history will have their loan approved
7. Areas also affect the loan approvement. Those living in Urban area have high ratio of loan approved (31.2%) compared to those living in the Rural area (29.0%).

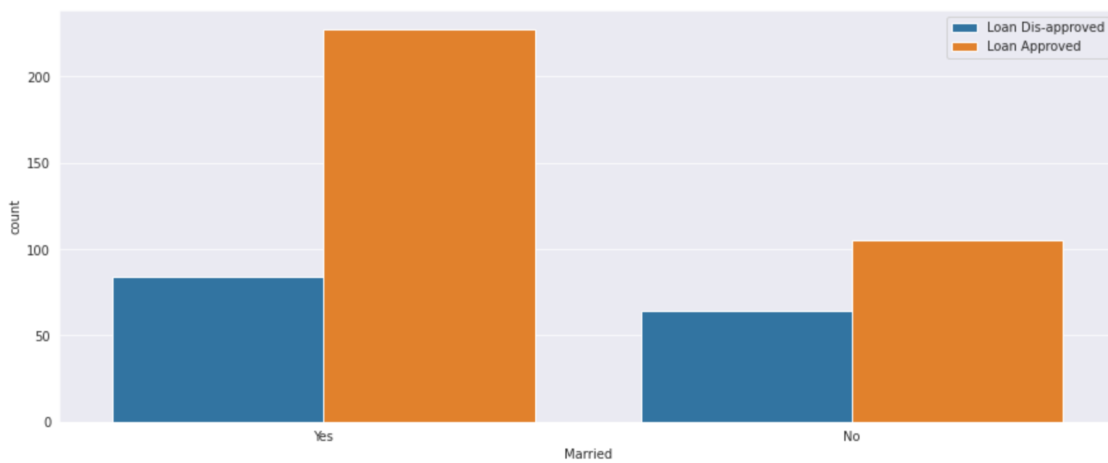
[ ]:
```
loan_summary = pd.pivot_table(data_df,index=data_df['Loan_Status'])
loan_summary
```

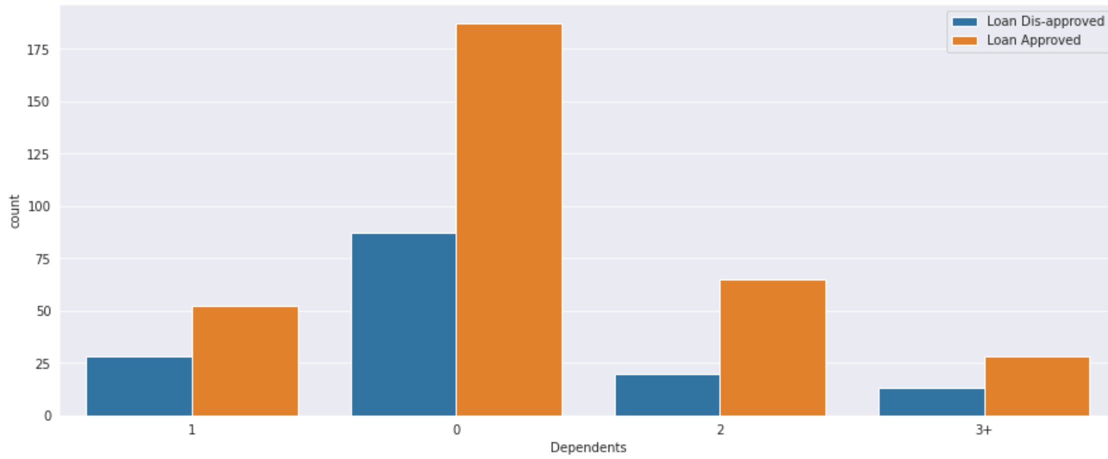
[ ]:
```
             Credit_History  Total_Income  Total_Loan
Loan_Status
```

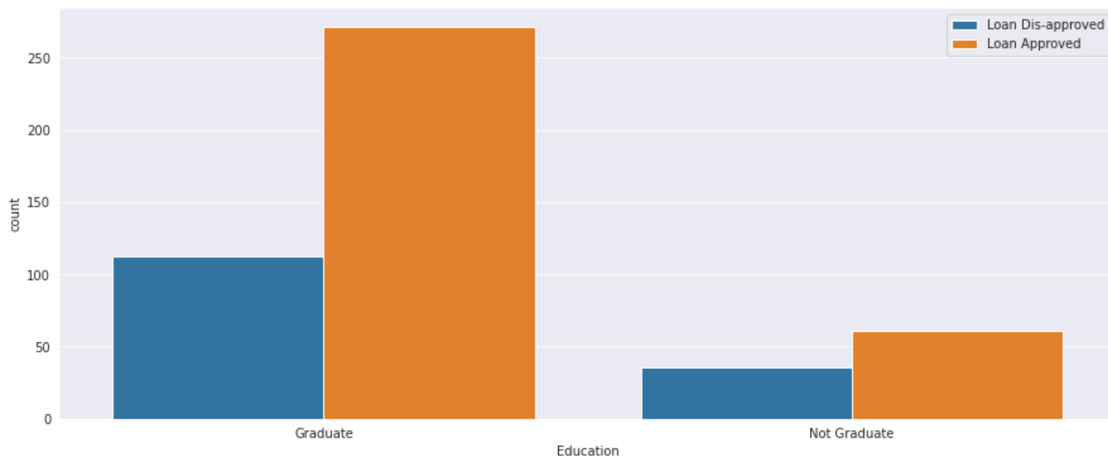
```
N                    0.574324   7503.270270  496.189189
Y                    0.978916   6696.602169  482.593373
```

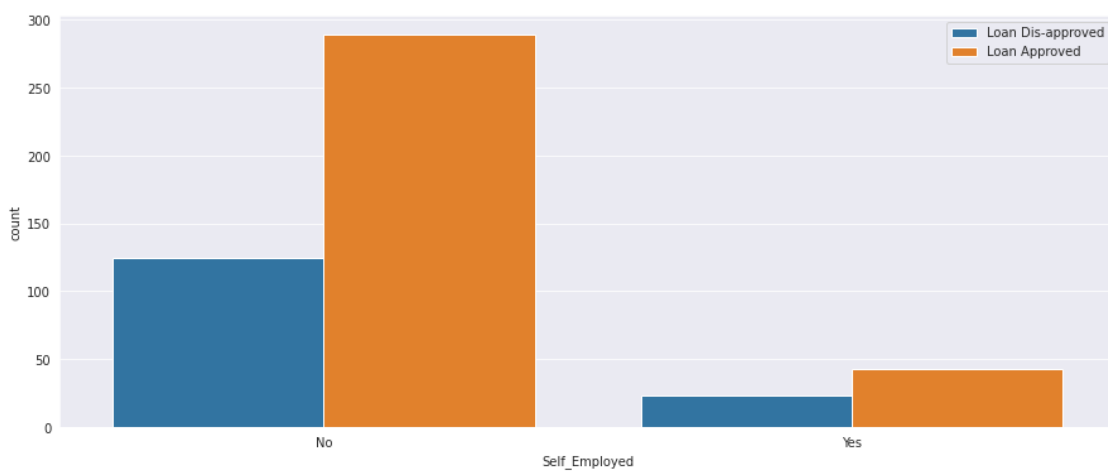
[ ]:
```
for i in data_df.columns:
    plt.figure(figsize=(15,6))
    sns.countplot(data_df[i], hue=data_df['Loan_Status'], data=data_df)
    plt.legend(['Loan Dis-approved','Loan Approved'])
```

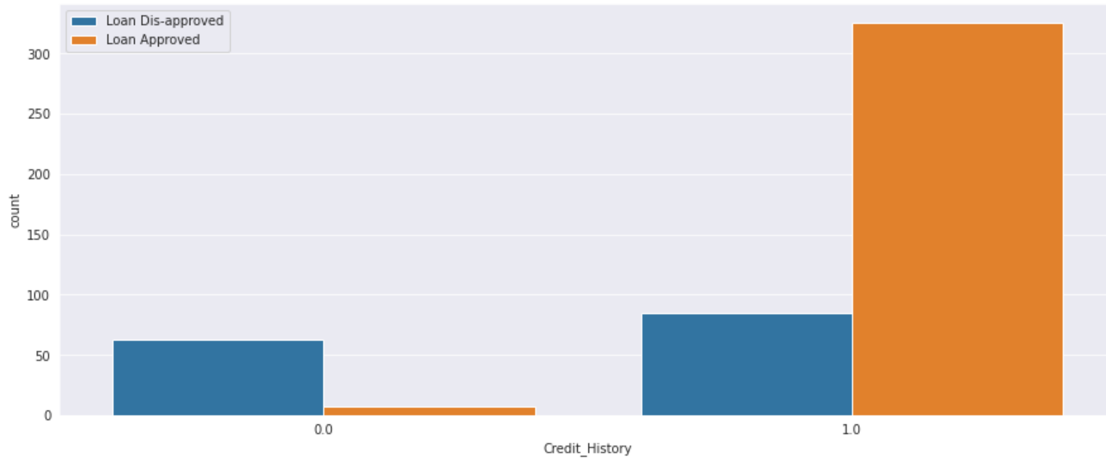




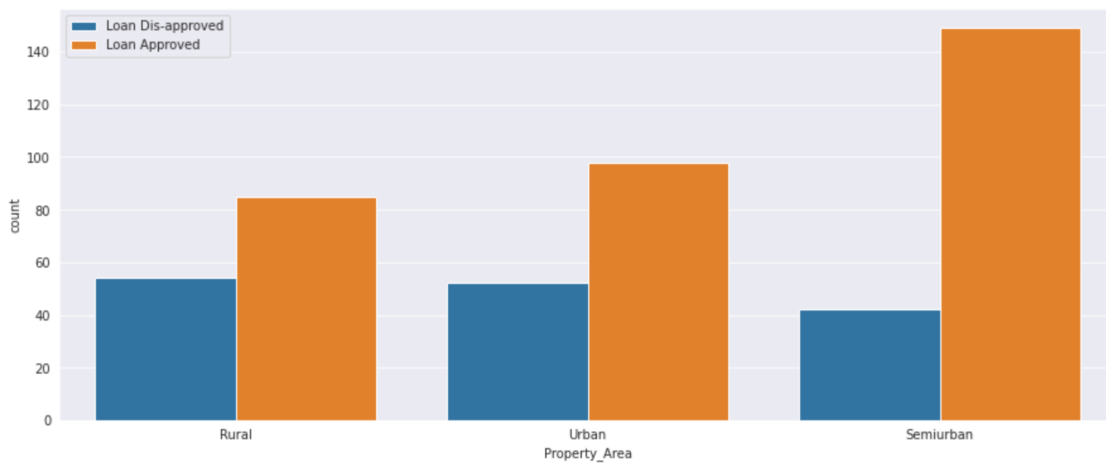

Loan Dis-approved
Loan Approved
count
175
150
125
100
75
50
25
0
1
0
2
3+
Dependents


Loan Dis-approved
Loan Approved
count
250
200
150
100
50
0
Graduate
Not Graduate
Education

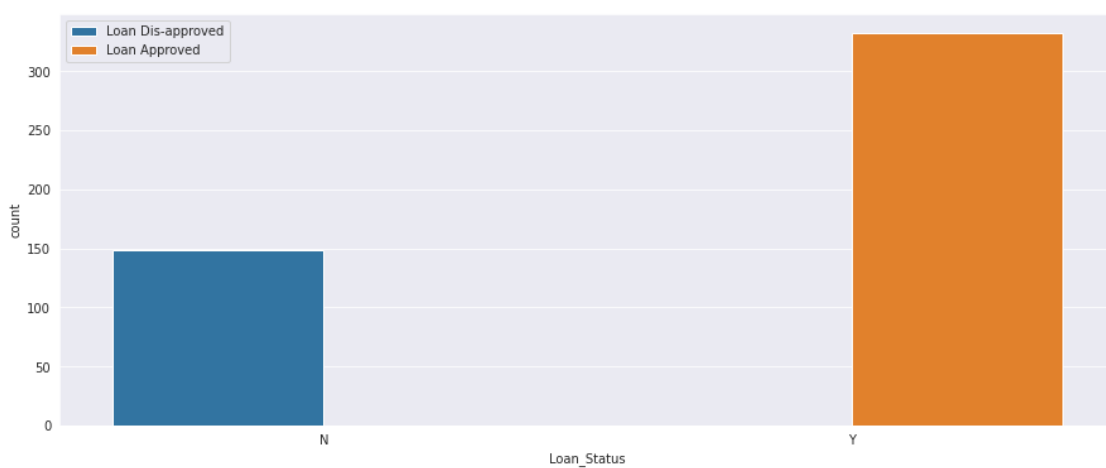

Loan Dis-approved
Loan Approved
count
300
250
200
150
100
50
0
No
Yes
Self_Employed



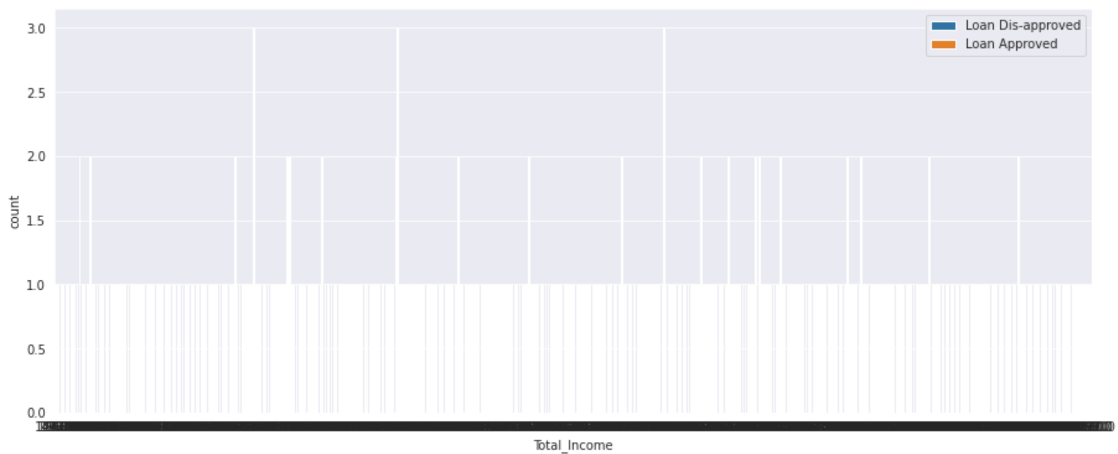
Loan Dis-approved
Loan Approved
300
250
200
150
100
50
0
count
0.0
1.0
Credit_History

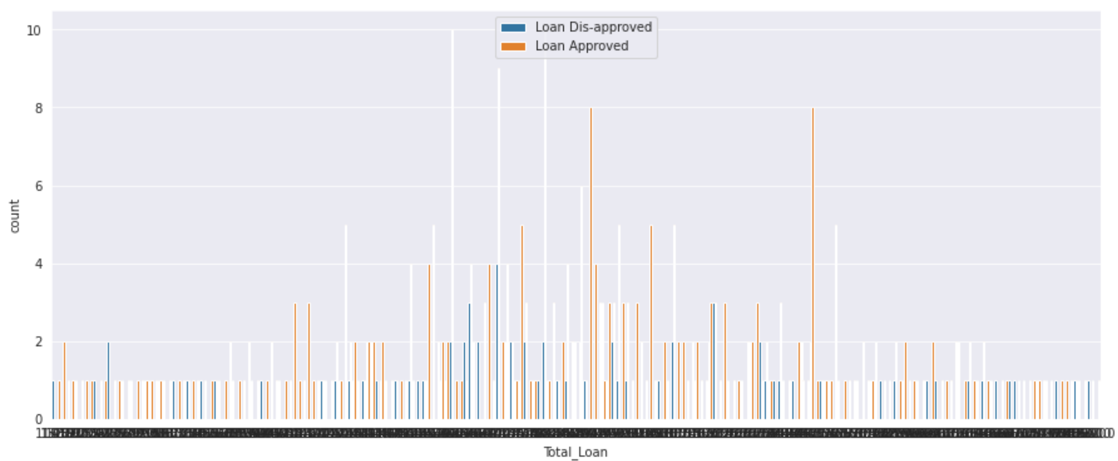


Loan Dis-approved
Loan Approved
140
120
100
80
60
40
20
0
count
Rural
Urban
Semiurban
Property_Area



Loan Dis-approved
Loan Approved
300
250
200
150
100
50
0
count
N
Y
Loan_Status





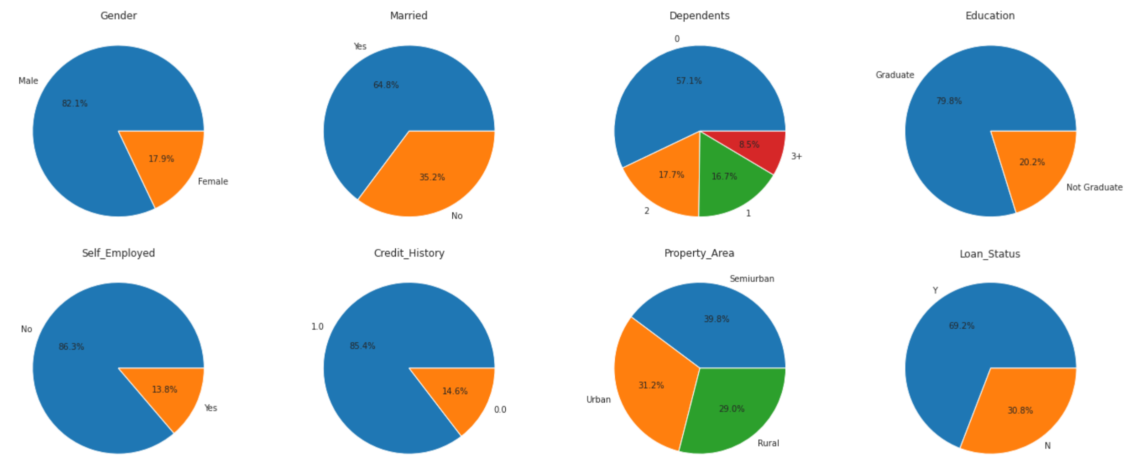
0.2.3 Univariant Analysis

[ ]:
```
continous_features = ['Total_Income', 'Total_Loan']
categorical_features = ['Gender', 'Married', 'Dependents', 'Education',␣
↪'Self_Employed', 'Credit_History'\
                        , 'Property_Area', 'Loan_Status']
print(pd.DataFrame(continous_features))
print(pd.DataFrame(categorical_features))
```

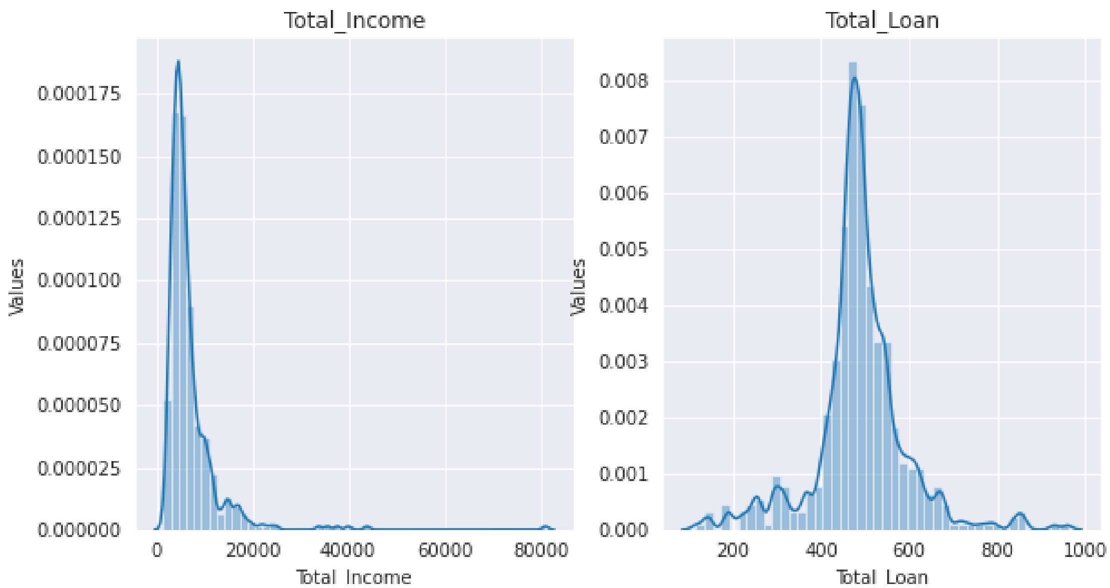
```
              0
0  Total_Income
1    Total_Loan
                0
0          Gender
```

```
1          Married
2       Dependents
3        Education
4    Self_Employed
5   Credit_History
6    Property_Area
7      Loan_Status
```

[ ]:
```
plt.figure(figsize=(20,8))
for index, col in enumerate(categorical_features,start=1):
    plt.subplot(2,4,index)
    plt.title(col)
    plt.pie(df[col].value_counts().values,autopct='%1.1f%%', labels=df[col].value_counts().index)
plt.tight_layout()
```

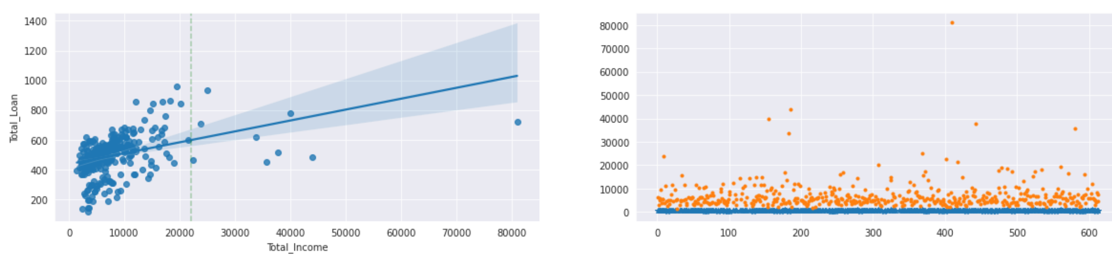


[ ]:
```
plt.figure(figsize=(20,5))
plt.tight_layout()
for index, col in enumerate(continous_features, start=1):
    plt.subplot(1,4,index)
    plt.title(col)
    plt.xlabel('Range')
    plt.ylabel('Values')
    sns.distplot(data_df[col], kde_kws={'bw':0.1})
```



[ ]:
```
plt.figure(figsize=(20,4))
plt.subplot(1,2,1)
sns.regplot(x='Total_Income', y='Total_Loan', data=data_df);
plt.axvline(x=22_000, c='green', alpha=0.3, linestyle='--');

plt.subplot(1,2,2)
plt.plot(data_df['Total_Loan'], marker="*", linestyle='')
plt.plot(data_df['Total_Income'], marker=".", linestyle='');
```



[ ]:
```
#Label Encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
# Dependent Variables
data_df['Gender'] = encoder.fit_transform(data_df['Gender'])
data_df['Married'] = encoder.fit_transform(data_df['Married'])
data_df['Self_Employed'] = encoder.fit_transform(data_df['Self_Employed'])
data_df['Education'] = encoder.fit_transform(data_df['Education'])
```

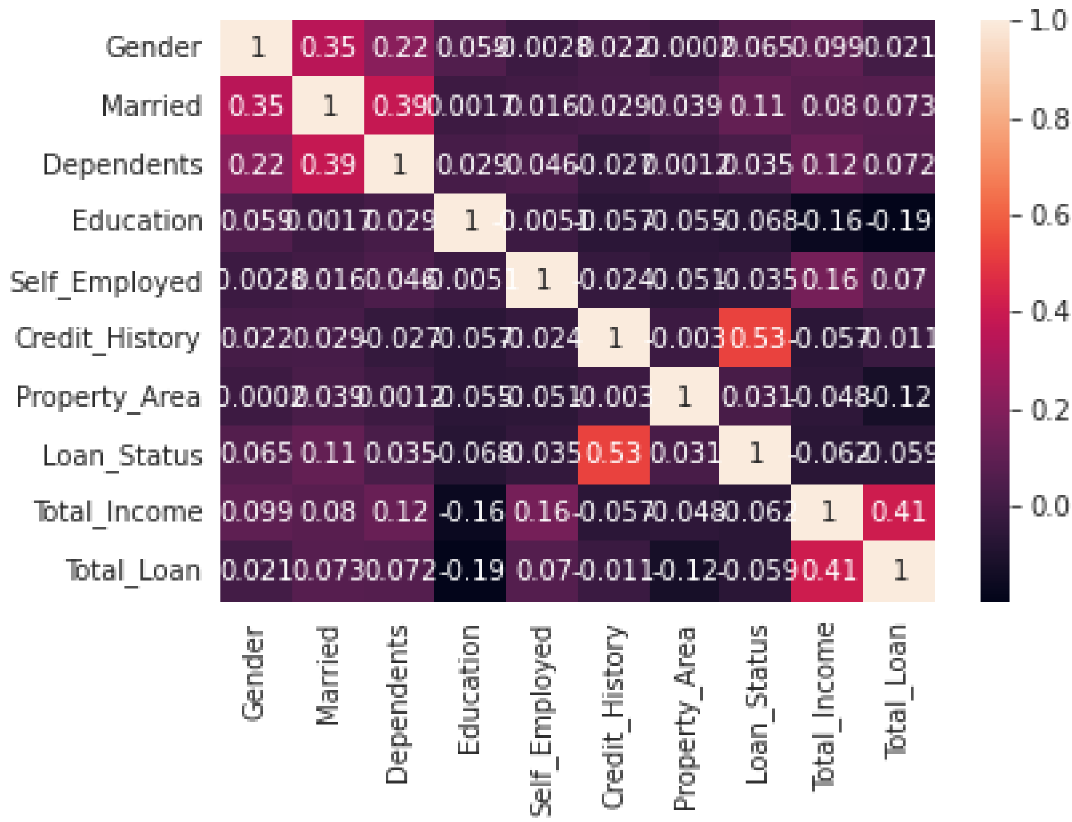
```python
data_df['Property_Area'] = encoder.fit_transform(data_df['Property_Area'])
dependent = {'0':0,'1':1,'2':2,'3+':3}
data_df['Dependents'] = data_df['Dependents'].map(dependent)
# Independent variable
data_df['Loan_Status'] = encoder.fit_transform(data_df['Loan_Status'])
```

[ ]:
```python
#Feature Scaling
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()
scale.fit_transform(data_df)
```

[ ]:
```
array([[1.        , 1.        , 0.33333333, …, 0.        , 0.05843536,
        0.44075829],
       [1.        , 1.        , 0.        , …, 1.        , 0.0195832 ,
        0.36729858],
       [1.        , 1.        , 0.        , …, 1.        , 0.04398049,
        0.43127962],
       …,
       [1.        , 1.        , 0.33333333, …, 1.        , 0.0863521 ,
        0.58886256],
       [1.        , 1.        , 0.66666667, …, 1.        , 0.07718897,
        0.51066351],
       [0.        , 0.        , 0.        , …, 0.        , 0.03948063,
        0.44668246]])
```

[ ]:
```python
#Correlation
corr = data_df.corr()
sns.heatmap(corr,annot=True)
```

[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9ab5372310>
```




```
[ ]: #correlation table
     corr = data_df.corr()
     corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

```
[ ]: <pandas.io.formats.style.Styler at 0x7f9ab554a250>
```

```
[ ]: #Spliting into X and y
     X = data_df.drop(['Loan_Status'],axis=1)
     y = data_df['Loan_Status']
```

0.2.4 Model Build

```
[ ]: from sklearn.model_selection import train_test_split
     X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
      ↪2,random_state=0)
     print(X_train.shape)
     print(y_train.shape)
     print(X_test.shape)
     print(y_test.shape)
```

```
(384, 9)
(384,)
(96, 9)
(96,)
```

0.2.5 Importing the libraries

[ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

[ ]:
```
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
```

[ ]:
```
LogisticRegression()
```

[ ]:
```
lr_pred = lr_model.predict(X_test)
pd.DataFrame(lr_pred)
```

[ ]:
```
    0
0   1
1   1
2   1
3   1
4   1
..  ..
91  0
92  1
93  1
94  1
95  1

[96 rows x 1 columns]
```

[ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,lr_pred)
cm
```

[ ]:
```
array([[13, 22],
       [ 0, 61]])
```

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,lr_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.37      0.54        35
           1       0.73      1.00      0.85        61

    accuracy                           0.77        96
   macro avg       0.87      0.69      0.69        96
weighted avg       0.83      0.77      0.74        96
```

[ ]:
```
dtc_model = DecisionTreeClassifier()
dtc_model.fit(X_train,y_train)
```

[ ]:
```
DecisionTreeClassifier()
```

[ ]:
```
dtc_pred = dtc_model.predict(X_test)
pd.DataFrame(dtc_pred)
```

[ ]:
```
    0
0   1
1   1
2   1
3   1
4   0
..  ..
91  0
92  0
93  1
94  1
95  0

[96 rows x 1 columns]
```

[ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,dtc_pred)
cm
```

[ ]:
```
array([[19, 16],
       [19, 42]])
```

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,dtc_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      0.54      0.52        35
           1       0.72      0.69      0.71        61
```

```
    accuracy                           0.64        96
   macro avg       0.61      0.62      0.61        96
weighted avg       0.64      0.64      0.64        96
```

[ ]:
```
svc_model = SVC()
svc_model.fit(X_train,y_train)
```

[ ]:
```
SVC()
```

[ ]:
```
svc_pred = svc_model.predict(X_test)
pd.DataFrame(svc_pred)
```

[ ]:
```
    0
0   1
1   1
2   1
3   1
4   1
.. ..
91  1
92  1
93  1
94  1
95  1

[96 rows x 1 columns]
```

[ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,svc_pred)
cm
```

[ ]:
```
array([[ 0, 35],
       [ 0, 61]])
```

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,svc_pred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        35
           1       0.64      1.00      0.78        61

    accuracy                           0.64        96
   macro avg       0.32      0.50      0.39        96
weighted avg       0.40      0.64      0.49        96
```

[ ]:
```
rfc_model = RandomForestClassifier()
rfc_model.fit(X_train,y_train)
```

[ ]:
```
RandomForestClassifier()
```

[ ]:
```
rfc_pred = rfc_model.predict(X_test)
pd.DataFrame(rfc_pred)
```

[ ]:
```
    0
0   1
1   1
2   1
3   1
4   0
.. ..
91  0
92  1
93  1
94  1
95  1

[96 rows x 1 columns]
```

[ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,rfc_pred)
cm
```

[ ]:
```
array([[15, 20],
       [ 1, 60]])
```

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,rfc_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.43      0.59        35
           1       0.75      0.98      0.85        61

    accuracy                           0.78        96
   macro avg       0.84      0.71      0.72        96
weighted avg       0.82      0.78      0.76        96
```

[ ]:
```
Knc_model = KNeighborsClassifier()
Knc_model.fit(X_train,y_train)
```

[ ]:
```
KNeighborsClassifier()
```

[ ]:
```
Knc_pred = Knc_model.predict(X_test)
pd.DataFrame(Knc_pred)
```

[ ]:
```
    0
0   1
1   1
2   0
3   0
4   0
..  ..
91  1
92  1
93  1
94  1
95  1

[96 rows x 1 columns]
```

[ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,Knc_pred)
cm
```

[ ]:
```
array([[ 6, 29],
       [ 6, 55]])
```

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,Knc_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      0.17      0.26        35
           1       0.65      0.90      0.76        61

    accuracy                           0.64        96
   macro avg       0.58      0.54      0.51        96
weighted avg       0.60      0.64      0.58        96
```

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

[ ]:
```
GaussianNB()
```

[ ]:
```
Gb_pred = Gb_model.predict(X_test)
pd.DataFrame(Gb_pred)
```

[ ]:
```
    0
0   1
1   1
2   1
3   1
4   1
..  ..
91  0
92  0
93  1
94  1
95  1

[96 rows x 1 columns]
```

[ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,Gb_pred)
cm
```

[ ]:
```
array([[15, 20],
       [ 3, 58]])
```

[ ]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,Gb_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.43      0.57        35
           1       0.74      0.95      0.83        61

    accuracy                           0.76        96
   macro avg       0.79      0.69      0.70        96
weighted avg       0.78      0.76      0.74        96
```

0.2.6 Accuarcy Check

Logistic Regression

[ ]:
```
print('Training Accuracy :',lr_model.score(X_train,y_train))
print('Testing Accuracy :',lr_model.score(X_test,y_test))
```

```
Training Accuracy : 0.8177083333333334
Testing Accuracy : 0.7708333333333334
```

Decision Tree

```
[ ]: print('Training Accuracy :',dtc_model.score(X_train,y_train))
     print('Testing Accuracy :',dtc_model.score(X_test,y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.6354166666666666
```

SVC

```
[ ]: print('Training Accuracy :',svc_model.score(X_train,y_train))
     print('Testing Accuracy :',svc_model.score(X_test,y_test))
```

```
Training Accuracy : 0.7057291666666666
Testing Accuracy : 0.6354166666666666
```

Random Forest

```
[ ]: print('Training Accuracy :',rfc_model.score(X_train,y_train))
     print('Testing Accuracy :',rfc_model.score(X_test,y_test))
```

```
Training Accuracy : 1.0
Testing Accuracy : 0.78125
```

KNeighbors Classifier

```
[ ]: print('Training Accuracy :',Knc_model.score(X_train,y_train))
     print('Testing Accuracy :',Knc_model.score(X_test,y_test))
```

```
Training Accuracy : 0.7447916666666666
Testing Accuracy : 0.6354166666666666
```

```
[ ]: print('Training Accuracy :',Gb_model.score(X_train,y_train))
     print('Testing Accuracy :',Gb_model.score(X_test,y_test))
```

```
Training Accuracy : 0.8151041666666666
Testing Accuracy : 0.7604166666666666
```

```
[ ]: !apt-get install texlive texlive-xetex texlive-latex-extra pandoc
     !pip install pypandoc
```