LOAN PREDICTION

To Predict The Loan will Approve or Not

```
import numpy as np
import pandas as pd
import seaborn as sns # data visualization
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.naive_bayes import GaussianNB,BernoulliNB,MultinomialNB
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardScaler
from sklearn.model_selection import train_test_split,GridSearchCV
from sklearn.metrics import confusion_matrix,f1_score,classification_report,accuracy_score
```

```
df=pd.read_csv("/content/train_u6lujuX_CVtuZ9i (1).csv")
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncor
0	LP001002	Male	No	0	Graduate	No	5849	0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358
4	LP001008	Male	No	0	Graduate	No	6000	0
...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0
611	LP002983	Male	Yes	1	Graduate	No	8072	240
612	LP002984	Male	Yes	2	Graduate	No	7583	0
613	LP002990	Female	No	0	Graduate	Yes	4583	0

614 rows × 13 columns

```
df.shape
```

```
(614, 13)
```

```
df.info()
```

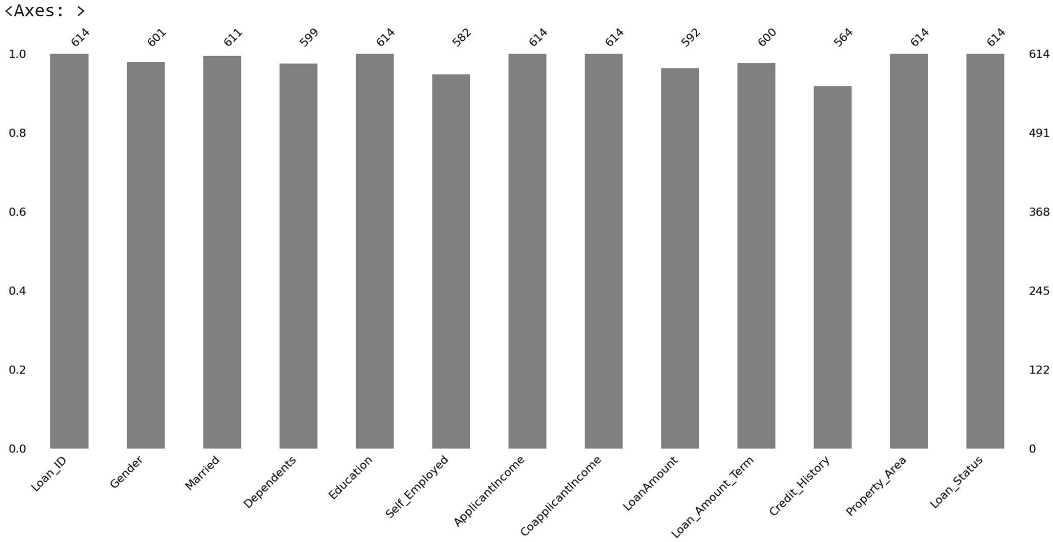
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            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
```

```
df.isna().sum().reset_index()
```

	index	0
0	Loan_ID	0
1	Gender	13
2	Married	3
3	Dependents	15
4	Education	0
5	Self_Employed	32
6	ApplicantIncome	0
7	CoapplicantIncome	0
8	LoanAmount	22
9	Loan_Amount_Term	14
10	Credit_History	50
11	Property_Area	0
12	Loan_Status	0

```
import missingno as mns
mns.bar(df,color='grey')
```

```
<Axes: >
```




```
df.tail()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncor
609	LP002978	Female	No	0	Graduate	No	2900	0

```
df.drop("Loan_ID",axis=1,inplace=True)
```

```
df['Gender'].fillna(df['Gender'].mode()[0],inplace=True)
```

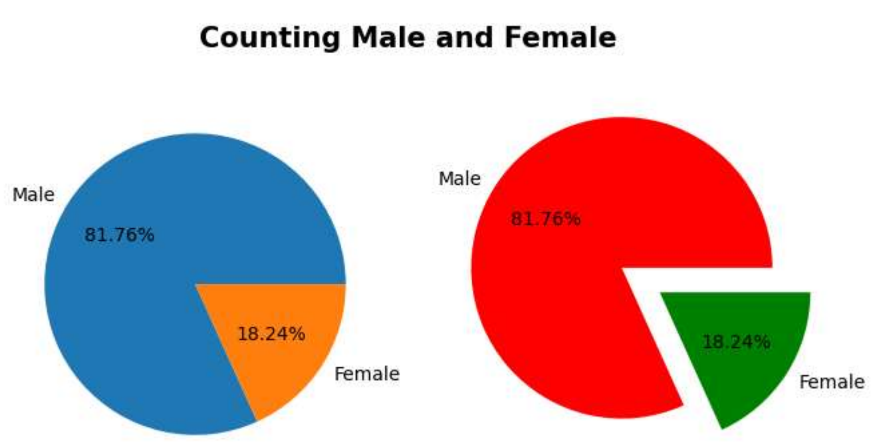
```
df.head(3)
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmoun
0	Male	No	0	Graduate	No	5849	0.0	Na
1	Male	Yes	1	Graduate	No	4583	1508.0	128.
2	Male	Yes	0	Graduate	Yes	3000	0.0	66.

```
v1=df.value_counts('Gender').reset_index()
v1
```

	Gender	0
0	Male	502
1	Female	112

```
plt.figure(figsize=(8,4),facecolor="w")
plt.subplot(1,2,1)
plt.pie(v1[0],labels=v1["Gender"],autopct="%0.2f%%")
plt.subplot(1,2,2)
plt.pie(v1[0],labels=v1["Gender"],autopct="%0.2f%%",explode=[0.2,.1],colors=["r","g"])
plt.suptitle("Counting Male and Female",fontweight="bold",fontsize=15)
plt.show()
```



There are 489 male and 112 female.

```
df['Gender']=LabelEncoder().fit_transform(df['Gender'])
```

```
df.head()
```

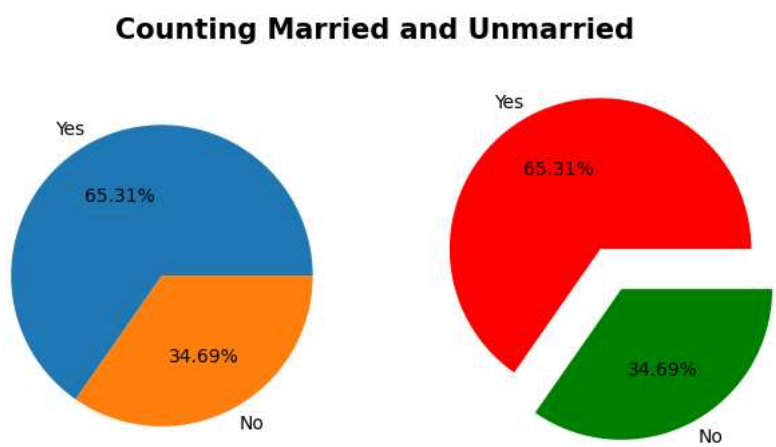
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmoun
0	1	No	0	Graduate	No	5849	0.0	Na
1	1	Yes	1	Graduate	No	4583	1508.0	128.
2	1	Yes	0	Graduate	Yes	3000	0.0	66.
[illegible]	[illegible]	[illegible]	[illegible]	Graduate	[illegible]	[illegible]	[illegible]	[illegible]

```
df['Married'].fillna(df['Married'].mode()[0],inplace=True)
```

```
counts=df['Married'].value_counts('index').reset_index()
counts
```

	index	Married
0	Yes	0.653094
1	No	0.346906

```
plt.figure(figsize=(8,4),facecolor="w")
plt.subplot(1,2,1)
plt.pie(counts["Married"],labels=counts['index'],autopct="%0.2f%%")
plt.subplot(1,2,2)
plt.pie(counts["Married"],labels=counts['index'],autopct="%0.2f%%",explode=[0.2,.1],colors=["r","g"])
plt.suptitle("Counting Married and Unmarried",fontweight="bold",fontsize=15)
plt.show()
```



There are 401 Married and 213 Unmarried

```
#label encoding
df['Married']=LabelEncoder().fit_transform(df['Married'])
```

```
df.tail()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmo
609	0	0	0	Graduate	No	2900	0.0	[illegible]
610	1	1	3+	Graduate	No	4106	0.0	[illegible]
611	1	1	1	Graduate	No	8072	240.0	2
612	1	1	2	Graduate	No	7583	0.0	1
613	0	0	0	Graduate	Yes	4583	0.0	1

```
df['Dependents'].unique()
```

```
array(['0', '1', '2', '3+', nan], dtype=object)
```

```
df['Dependents'].fillna(df['Dependents'].mode()[0],inplace=True)
```

```
df['Dependents']=df['Dependents'].str.replace('+','').astype(int)
df['Dependents']
```

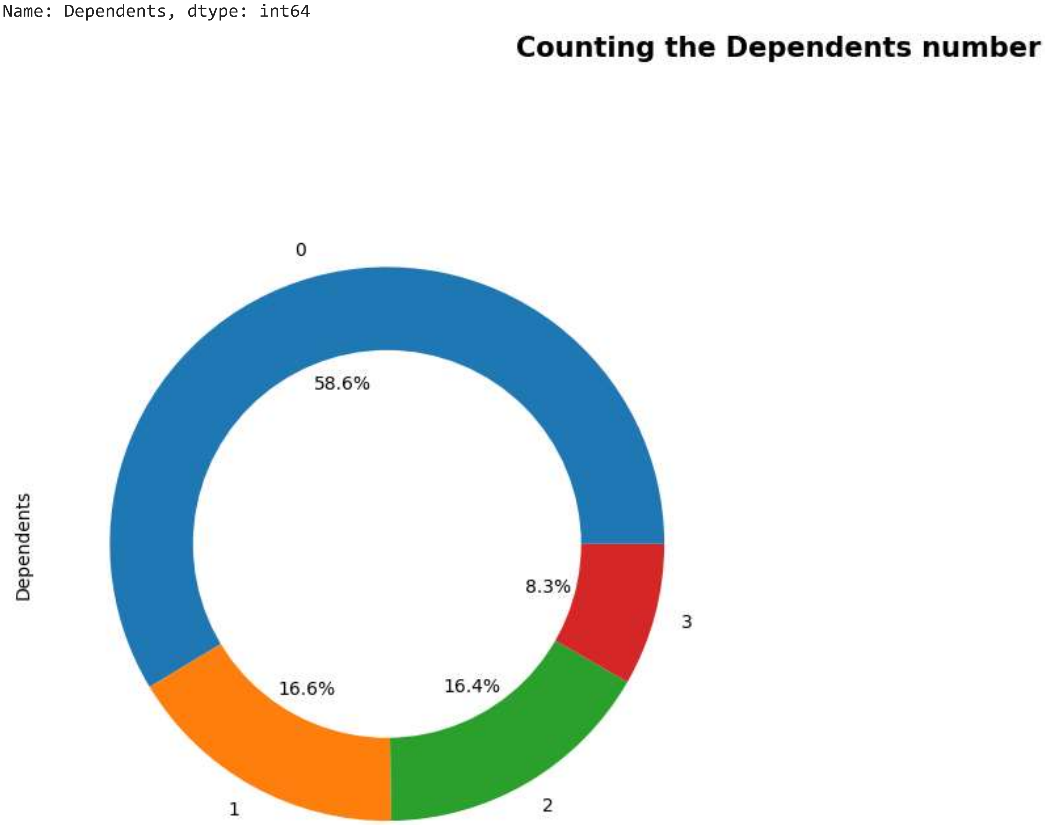
```
0      0
1      1
2      0
3      0
4      0
      ..
609    0
610    3
611    1
612    2
613    0
Name: Dependents, Length: 614, dtype: int64
```

```
df['Dependents'].unique()
```

```
array([0, 1, 2, 3])
```

```
plt.figure(figsize=(15,8))
plt.subplot(1,2,1)
df['Dependents'].value_counts().plot.pie(autopct='%1.1f%%')
centre=plt.Circle((0,0),0.7,fc='white')
fig=plt.gcf()
fig.gca().add_artist(centre)
plt.suptitle("Counting the Dependents number",fontweight="bold",fontsize=15)
df['Dependents'].value_counts()
```

```
0    360
1    102
2    101
3     51
Name: Dependents, dtype: int64
```




```
df.head()
```

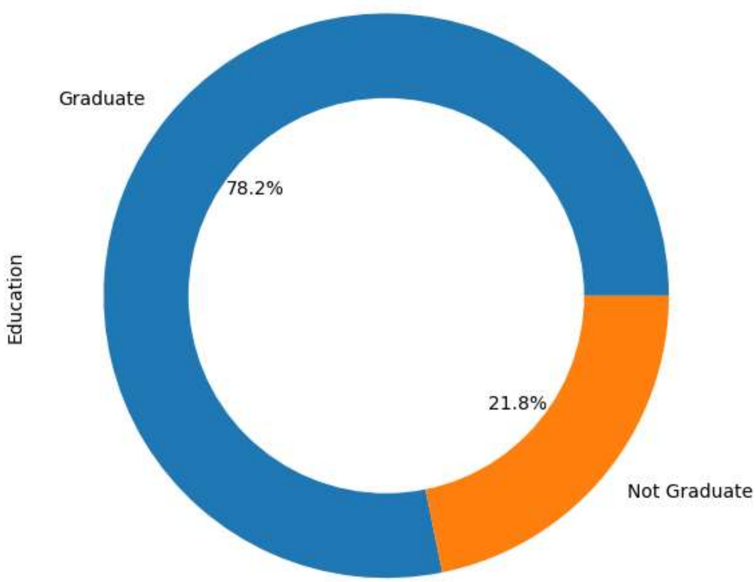
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmoun
0	1	0	0	Graduate	No	5849	0.0	Na
1	1	1	1	Graduate	No	4583	1508.0	128.
2	1	1	0	Graduate	Yes	3000	0.0	66.
3	1	1	0	Not Graduate	No	2583	2358.0	120.
4	1	0	0	Graduate	No	6000	0.0	141.

```
v2=df['Education'].value_counts()
v2
```

```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```

```
plt.figure(figsize=(15,8))
plt.subplot(1,2,1)
df['Education'].value_counts().plot.pie(autopct='%1.1f%%')
centre=plt.Circle((0,0),0.7,fc='white')
fig=plt.gcf()
fig.gca().add_artist(centre)
df['Education'].value_counts()
```

```
Graduate        480
Not Graduate    134
Name: Education, dtype: int64
```



There are 78.2% Person Graduate and 21.8% Not Graduate

```
df['Education'].unique()
```

```
df['Education']=df['Education'].map({"Graduate":0,"Not Graduate":1})
```

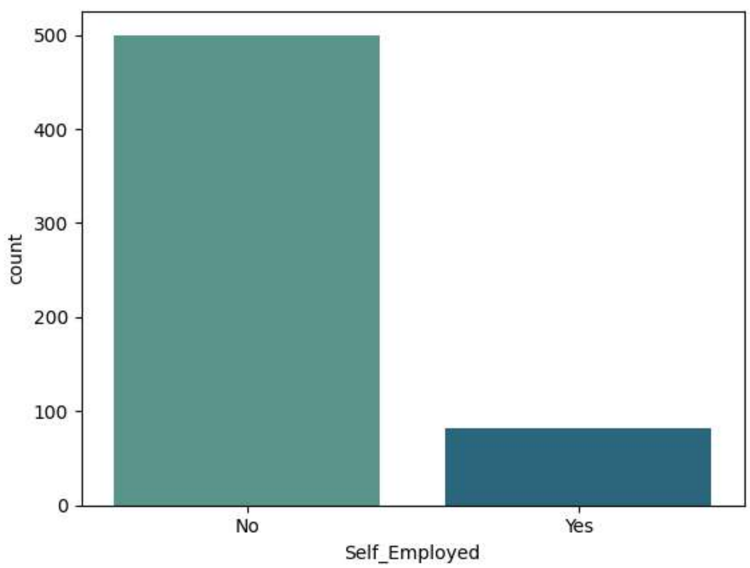
```
df.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmoun
0	1	0	0	0	No	5849	0.0	Na
1	1	1	1	0	No	4583	1508.0	128.
2	1	1	0	0	Yes	3000	0.0	66.
3	1	1	0	1	No	2583	2358.0	120.
4	1	0	0	0	No	6000	0.0	141.

```
df.Self_Employed.value_counts(dropna=False)
```

```
No     500
Yes     82
NaN     32
Name: Self_Employed, dtype: int64
```

```
sns.countplot(x="Self_Employed", data=df, palette="crest")
plt.show()
```




```
countNo = len(df[df.Self_Employed == 'No'])
countYes = len(df[df.Self_Employed == 'Yes'])
countNull = len(df[df.Self_Employed.isnull()])

print("Percentage of Not self employed: {:.2f}%".format((countNo / (len(df.Self_Employed))*100)))
print("Percentage of self employed: {:.2f}%".format((countYes / (len(df.Self_Employed))*100)))
print("Missing values percentage: {:.2f}%".format((countNull / (len(df.Self_Employed))*100)))
```

```
Percentage of Not self employed: 81.43%
Percentage of self employed: 13.36%
Missing values percentage: 5.21%
```

```
df['Self_Employed']=df['Self_Employed'].map({"No":0,"Yes":1})
```

```
df.tail()
```

```
df.info()
```

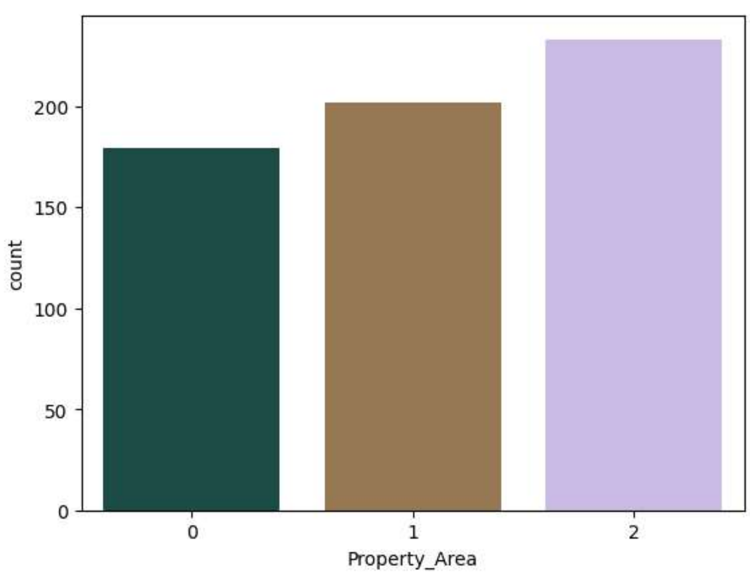
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             614 non-null    int64  
 1   Married            614 non-null    int64  
 2   Dependents         614 non-null    int64  
 3   Education          614 non-null    int64  
 4   Self_Employed      582 non-null    float64
 5   ApplicantIncome    614 non-null    int64  
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         592 non-null    float64
 8   Loan_Amount_Term   600 non-null    float64
 9   Credit_History     564 non-null    float64
 10  Property_Area      614 non-null    object 
 11  Loan_Status        614 non-null    object 
dtypes: float64(5), int64(5), object(2)
memory usage: 57.7+ KB
```

```
df['Property_Area']=df['Property_Area'].map({'Rural':0,'Urban':1,'Semiurban':2})
df['Property_Area']
```

```
0      1
1      0
2      1
3      1
4      1
      ..
609    0
610    0
611    1
612    1
613    2
Name: Property_Area, Length: 614, dtype: int64
```

```
df.tail()
```

```
sns.countplot(x="Property_Area", data=df, palette="cubehelix")
plt.show()
```




```
df['Loan_Status']=LabelEncoder().fit_transform(df['Loan_Status'])
```

```
df.tail()
```

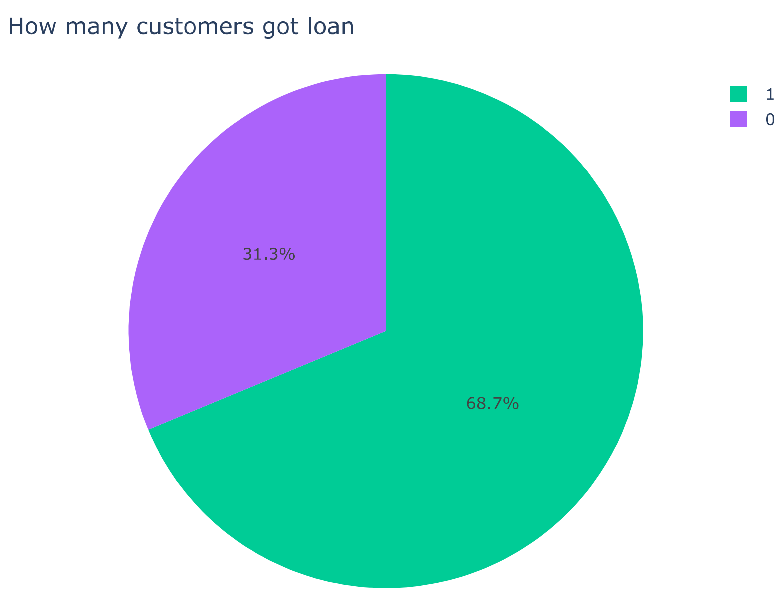
	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmo
609	0	0	0	0	0.0	2900	0.0	7
610	1	1	3	0	0.0	4106	0.0	4
611	1	1	1	0	0.0	8072	240.0	25
612	1	1	2	0	0.0	7583	0.0	18
613	0	0	0	0	1.0	4583	0.0	13

```
loan_counts = df['Loan_Status'].value_counts()
df_loan = pd.DataFrame(loan_counts).reset_index()
df_loan = df_loan.rename(columns={"index": "Got Loan", "Loan_Status": "count"})
df_loan
```

	Got Loan	count
0	1	422
1	0	192

```
import plotly.express as px
fig = px.pie(df_loan, values='count', names='Got Loan', color='Got Loan',
             color_discrete_map={'Yes':'lightgreen','No':'lightred'})

fig.update_layout(title_text='How many customers got loan')
fig.show()
```




```
df.isna().sum()
```

```
Gender               0
Married              0
Dependents           0
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
```

```
df['LoanAmount'].isna().sum()
```

```
22
```

```
df['LoanAmount'].mean()
```

```
146.41216216216216
```

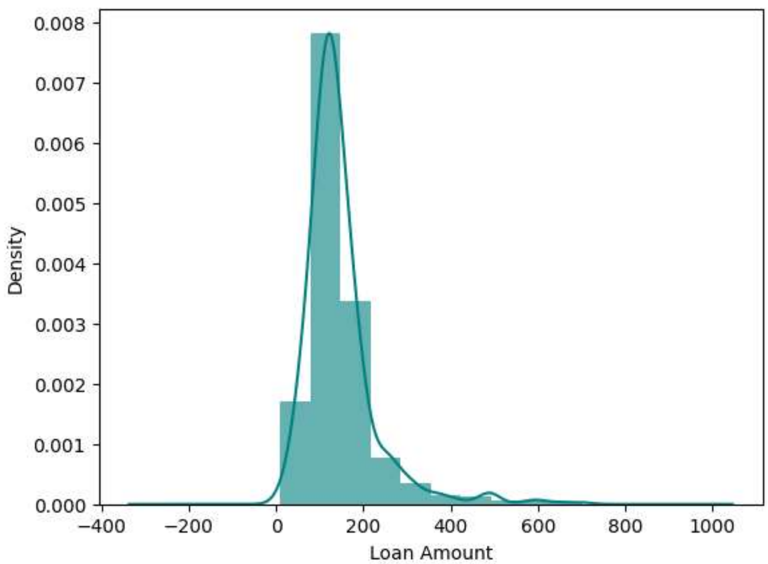
```
df['LoanAmount'].median()
```

```
128.0
```

If mean and median of difference is more than 10 then fill nun value with mean else median.

```
df['LoanAmount'].fillna(df['LoanAmount'].mean(),inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0],inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(),inplace=True)
```

```
ax = df["LoanAmount"].hist(density=True, stacked=True, color='teal', alpha=0.6)
df["LoanAmount"].plot(kind='density', color='teal')
ax.set(xlabel='Loan Amount')
plt.show()
```




```
df.head()
```

```
df.tail()
```

```
df.isna().sum()
```

```
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      0
Credit_History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

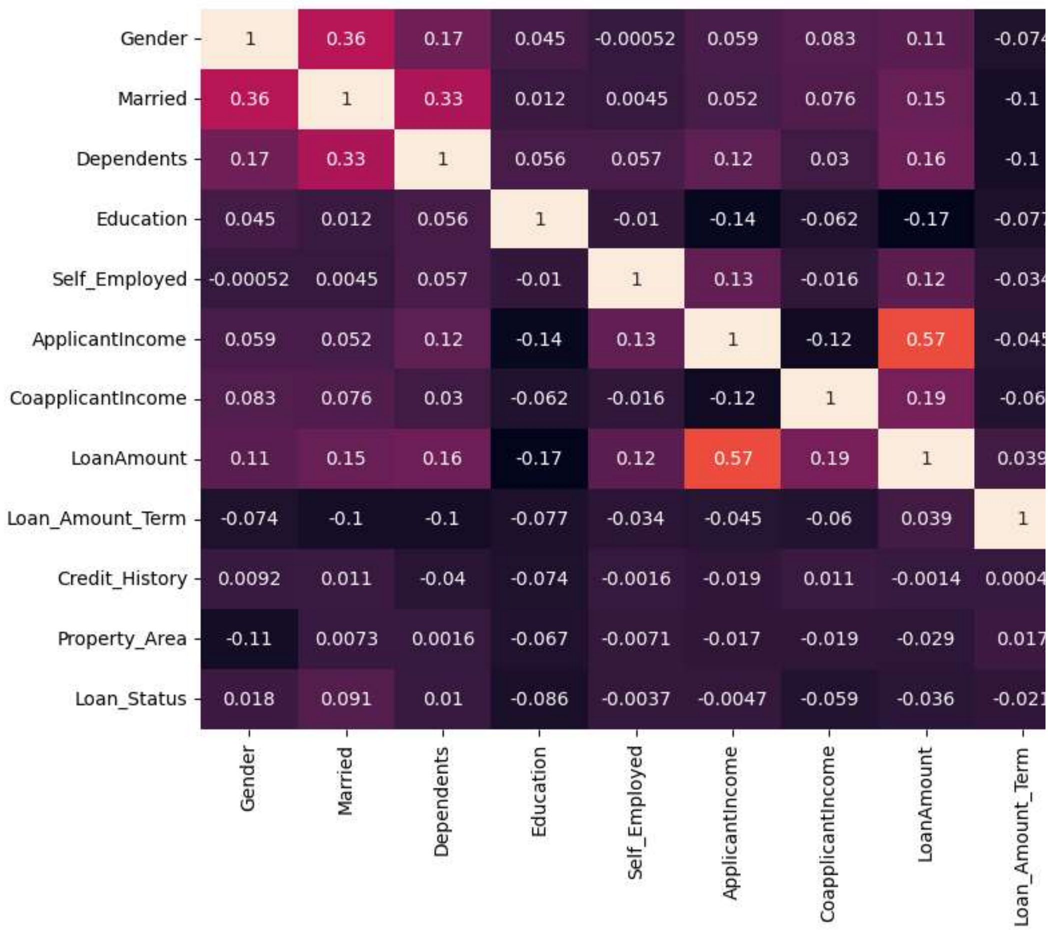
```
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0],inplace=True)
```

```
df.info()
```

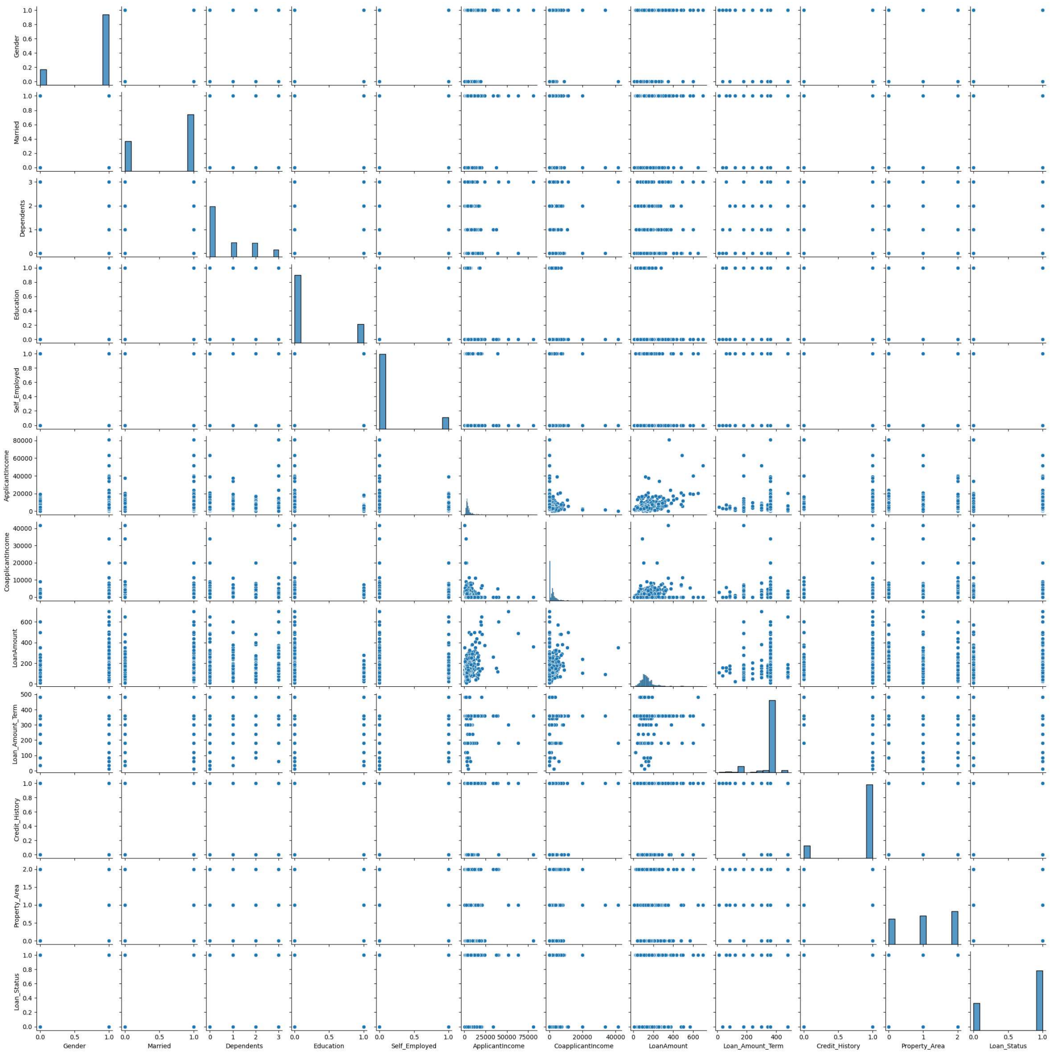
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
```

```
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             614 non-null    int64  
 1   Married            614 non-null    int64  
 2   Dependents         614 non-null    int64  
 3   Education          614 non-null    int64  
 4   Self_Employed      614 non-null    float64
 5   ApplicantIncome    614 non-null    int64  
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         614 non-null    float64
 8   Loan_Amount_Term   614 non-null    float64
 9   Credit_History     614 non-null    float64
 10  Property_Area      614 non-null    int64  
 11  Loan_Status        614 non-null    int64  
dtypes: float64(5), int64(7)
memory usage: 57.7 KB
```

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize = (14,7))
sns.heatmap(df.corr(), annot = True);
```




```
sns.pairplot(df, size=2);
```



Now extract the features and target from the dataset

```
x=df.drop(columns=['Loan_Status'])
x.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmoun
0	1	0	0	0	0.0	5849	0.0	146.41216
1	1	1	1	0	0.0	4583	1508.0	128.00000
2	1	1	0	0	1.0	3000	0.0	66.00000
3	1	1	0	1	0.0	2583	2358.0	120.00000
4	1	0	0	0	0.0	6000	0.0	141.00000

standard scalling the features

```
sc=StandardScaler()
x1=sc.fit_transform(x)
```

```
x1
```

```
array([[ 0.47234264, -1.37208932, -0.73780632, ...,  0.27985054,
         0.41173269, -0.10798877],
       [ 0.47234264,  0.72881553,  0.25346957, ...,  0.27985054,
         0.41173269, -1.33586108],
       [ 0.47234264,  0.72881553, -0.73780632, ...,  0.27985054,
         0.41173269, -0.10798877],
       ...,
       [ 0.47234264,  0.72881553,  0.25346957, ...,  0.27985054,
         0.41173269, -0.10798877],
       [ 0.47234264,  0.72881553,  1.24474546, ...,  0.27985054,
         0.41173269, -0.10798877],
       [-2.11710719, -1.37208932, -0.73780632, ...,  0.27985054,
        -2.42876026,  1.11988354]])
```

```
y=df[['Loan_Status']].values
y
```

Distribution with train test split

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=32, test_size = 0.2)
print("X_train",x_train.shape)
print('Y_train',y_train.shape)
print("X_test",x_test.shape)
print("Y_test",y_test.shape)
```

```
X_train (491, 11)
Y_train (491, 1)
X_test (123, 11)
Y_test (123, 1)
```

```
from sklearn.svm import SVC
model = SVC(kernel='linear')
#training the support Vector Macine model
model.fit(x_train,y_train)
```

```
▾ SVC
SVC(kernel='linear')
```

```
x_train_pred=model.predict(x_train)
train_data_accuracy=accuracy_score(x_train_pred,y_train)
print(f'Accuracy on training data: {train_data_accuracy}')
```

```
Accuracy on training data: 0.7942973523421588
```

```
x_test_pred=model.predict(x_test)
test_data_accuracy=accuracy_score(x_test_pred,y_test)
print(f'Accuracy on test data : {test_data_accuracy}')
```

```
Accuracy on test data : 0.7479674796747967
```