```
#les importations:
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.cluster import AgglomerativeClustering


#importation de la data:
df=pd.read_csv('train_u6lujuX_CVtuZ9i.csv')
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncom
0	LP001002	Male	No	0	Graduate	No	584
1	LP001003	Male	Yes	1	Graduate	No	458
2	LP001005	Male	Yes	0	Graduate	Yes	300
3	LP001006	Male	Yes	0	Not Graduate	No	258
4	LP001008	Male	No	0	Graduate	No	600
...	...	...	...	...	...	...	.
609	LP002978	Female	No	0	Graduate	No	290
610	LP002979	Male	Yes	3+	Graduate	No	410
611	LP002983	Male	Yes	1	Graduate	No	807
612	LP002984	Male	Yes	2	Graduate	No	758
613	LP002990	Female	No	0	Graduate	Yes	458

614 rows × 13 columns

Data Preprocessing:

```
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.columns
```

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

```
df.isnull().sum()
```

```
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
```

```
df['Gender'].value_counts()
```

```
Male      489
Female    112
Name: Gender, dtype: int64
```

```
df['Gender'].fillna('Male',inplace=True)
df['Gender'].value_counts()
```

```
Male      502
Female    112
Name: Gender, dtype: int64
```

```
df['Gender'].isnull().sum()
```

```
0
```

```
df['Married'].value_counts()
```

```
Yes    398
No     213
Name: Married, dtype: int64
```

```
df['Married'].fillna('Yes',inplace=True)
df['Married'].value_counts()
```

```
Yes    401
No     213
Name: Married, dtype: int64
```

```
df['Married'].isnull().sum()
```

```
0
```

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

```
0     345
1     102
2     101
3+     51
Name: Dependents, dtype: int64
```

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

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

```
df['Dependents'].isnull().sum()
```

```
0
```

```
df['Self_Employed'].value_counts()
```

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

```
df['Self_Employed'].fillna('No',inplace=True)
df['Self_Employed'].value_counts()
```

```
No     532
Yes     82
Name: Self_Employed, dtype: int64
```

```
df['Self_Employed'].isnull().sum()
```

```
0
```

```
df['Loan_Amount_Term'].value_counts()
```

```
360.0    512
180.0     44
480.0     15
300.0     13
84.0       4
240.0      4
120.0      3
36.0       2
60.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

```
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(),inplace=True)
df['Loan_Amount_Term'].value_counts()
```

```
360.0    512
180.0     44
480.0     15
342.0     14
300.0     13
84.0       4
240.0      4
120.0      3
36.0       2
60.0       2
12.0       1
Name: Loan_Amount_Term, dtype: int64
```

```
df['Loan_Amount_Term'].isnull().sum()
```

```
0
```

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

```
120.0    20
110.0    17
100.0    15
187.0    12
160.0    12
         ..
570.0     1
300.0     1
376.0     1
117.0     1
311.0     1
Name: LoanAmount, Length: 203, dtype: int64
```

```
df['LoanAmount'].fillna(df['LoanAmount'].mean(),inplace=True)
df['LoanAmount'].value_counts()
```

```
146.412162    22
120.000000    20
110.000000    17
100.000000    15
160.000000    12
              ..
570.000000     1
300.000000     1
376.000000     1
117.000000     1
311.000000     1
Name: LoanAmount, Length: 204, dtype: int64
```

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

```
0
```

```
df['Credit_History'].value_counts()
```

```
1.0    475
0.0     89
Name: Credit_History, dtype: int64
```

```
df['Credit_History'].fillna(df['Credit_History'].mean(),inplace=True)
df['Credit_History'].value_counts()
```

```
1.000000    475
0.000000     89
0.842199     50
Name: Credit_History, dtype: int64
```

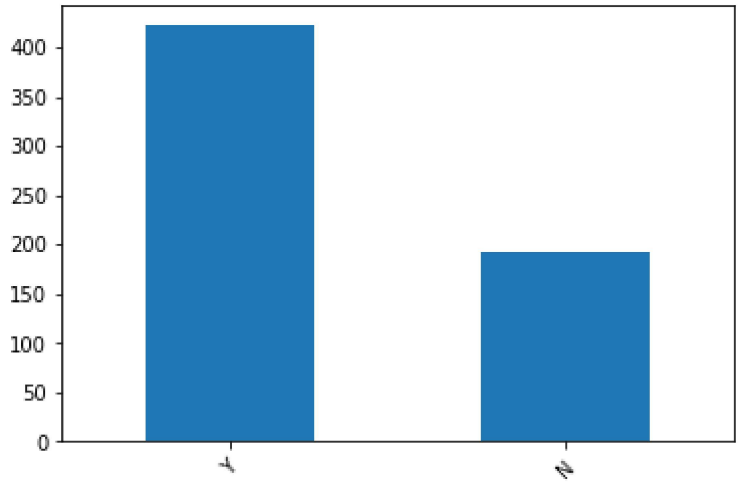
```
df.isnull().sum()
```

```
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
```

Data Visualization:

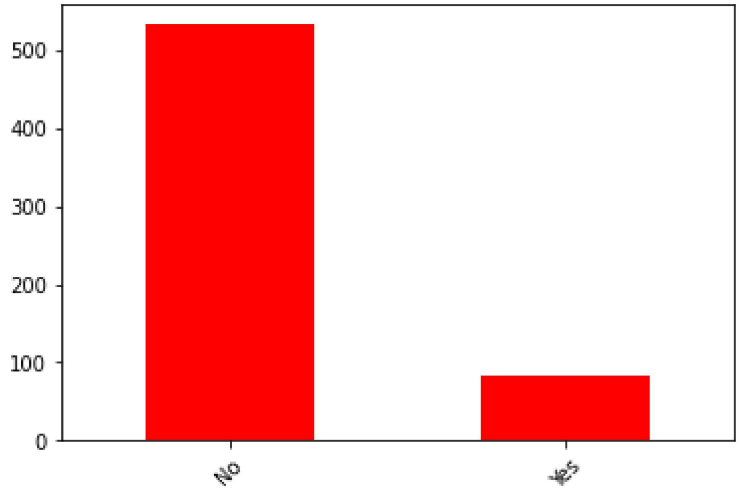
```
v=df['Loan_Status'].value_counts()
v.plot.bar(rot=45)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f620a6b05d0>
```



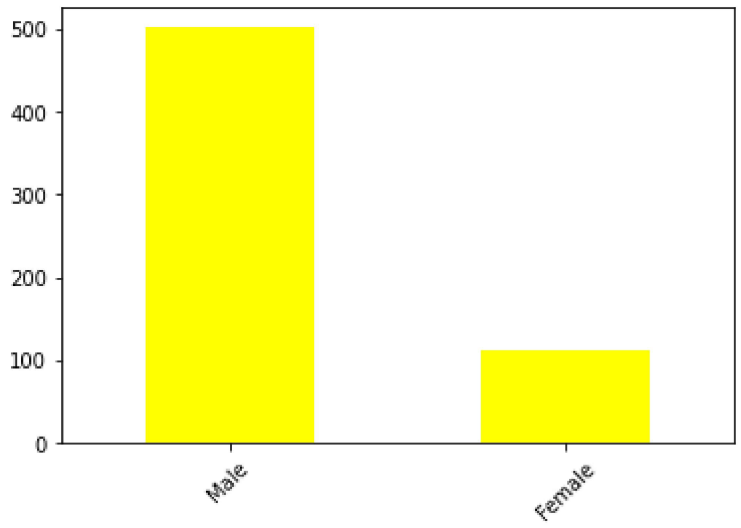

```
b=df['Self_Employed'].value_counts()
b.plot.bar(rot=45,color='red')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f620a6f17d0>
```




```
n=df['Gender'].value_counts()
n.plot.bar(rot=45,color='yellow')
```

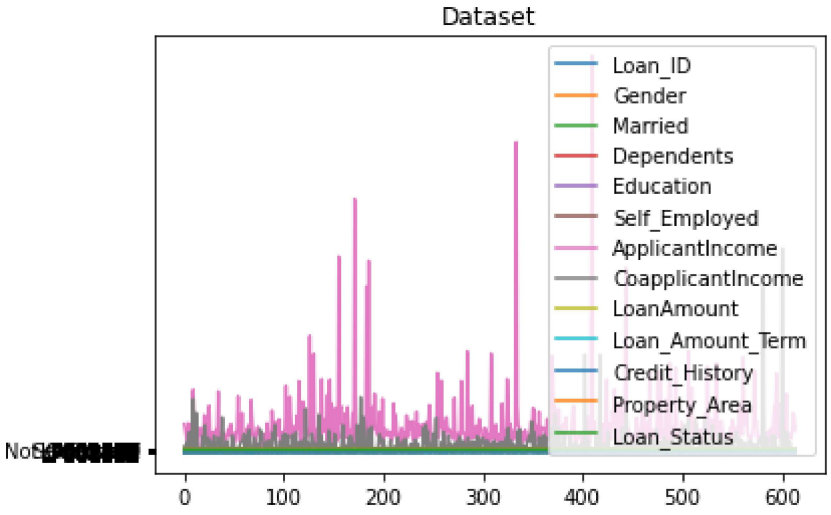
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f620a183e10>
```




```
columns = df.columns
```

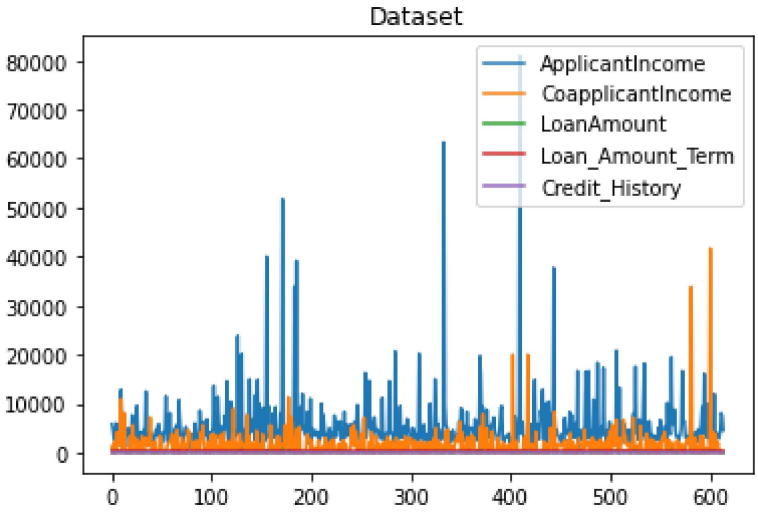
```
x_data = range(0, df.shape[0])
fig, ax = plt.subplots()
for column in columns:
    ax.plot(x_data, df[column], label=column)
ax.set_title('Dataset')
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7f620a0b1450>
```




```
df.plot.line(title='Dataset')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f62098c7390>
```

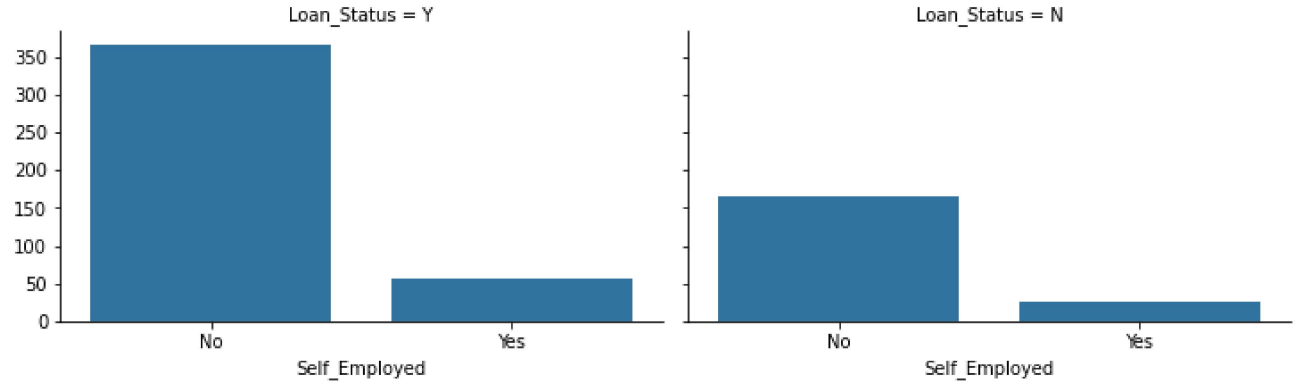



```
# Sexe
grid=sns.FacetGrid(df,col='Loan_Status',size=3.2,aspect=1.6)
grid.map(sns.countplot,'Gender')
```

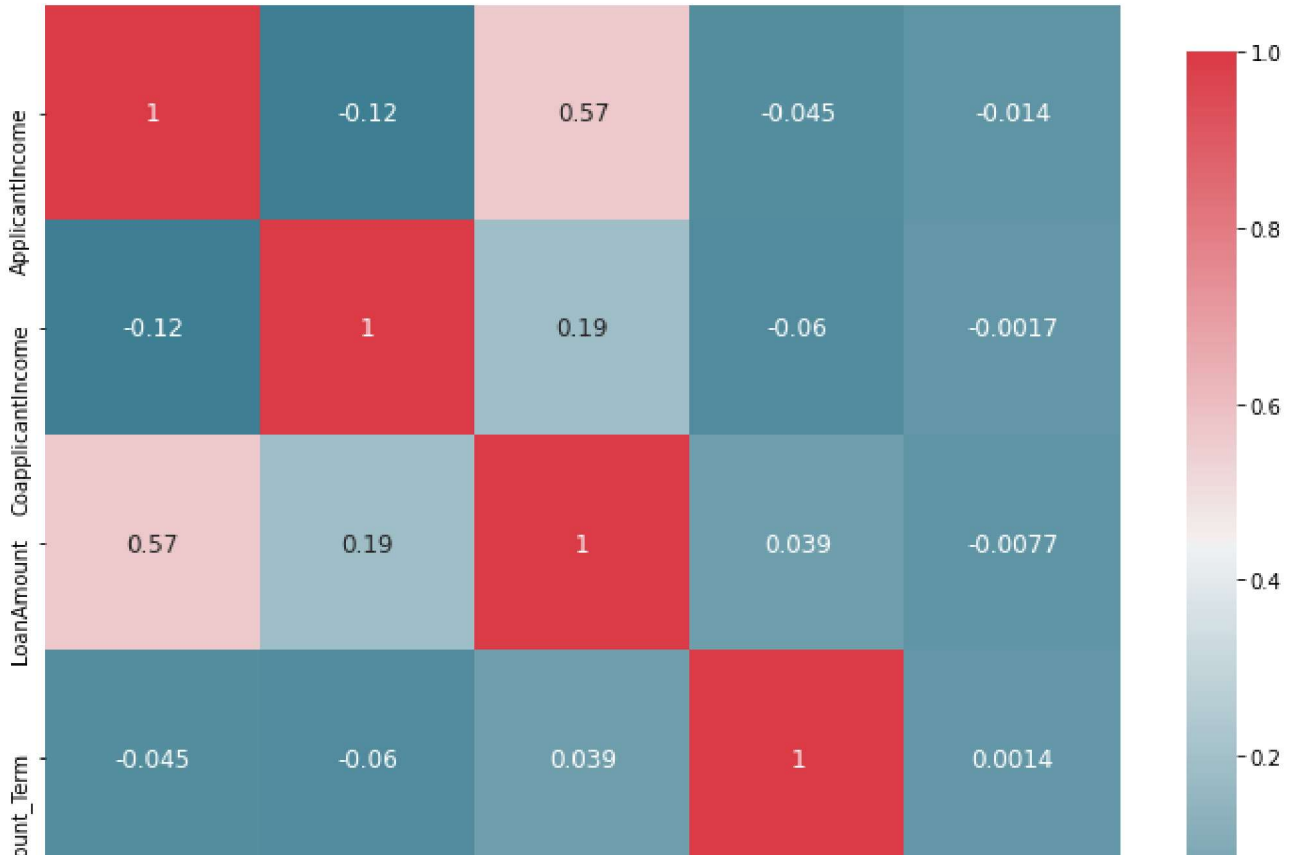
```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:316: UserWarning: The `siz
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:643: UserWarning: Using th
  warnings.warn(warning)
<seaborn.axisgrid.FacetGrid at 0x7f62099b7690>
```


```
# Self_Employed
grid=sns.FacetGrid(df,col='Loan_Status',size=3.2,aspect=1.6)
grid.map(sns.countplot,'Self_Employed')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:316: UserWarning: The `siz
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/axisgrid.py:643: UserWarning: Using th
  warnings.warn(warning)
<seaborn.axisgrid.FacetGrid at 0x7f620178af50>
```




```
def plot_correlation_map(df):
  corr=df.corr()
  s,ax=plt.subplots(figsize=(12,10))
  cmap=sns.diverging_palette(220,10,as_cmap=True)
  s=sns.heatmap(
      corr,
      cmap=cmap,
      square=True,
      cbar_kws={'shrink':.9},
      ax=ax,
      annot=True,
      annot_kws={'fontsize':12}
  )
plot_correlation_map(df)
```




```
# Supprimer loan_id
df.drop('Loan_ID',axis=1,inplace=True)
df.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplica
0	Male	No	0	Graduate	No	5849	
1	Male	Yes	1	Graduate	No	4583	
2	Male	Yes	0	Graduate	Yes	3000	
3	Male	Yes	0	Not Graduate	No	2583	
4	Male	No	0	Graduate	No	6000	

```
#convertir Categorical Data to numerical (Gender)
df['Gender']=df['Gender'].map({"Male":0,"Female":1})
df['Gender'].value_counts()
```

```
0    502
1    112
Name: Gender, dtype: int64
```

```
#convertir Categorical Data to numerical (Gender)
df['Married']=df['Married'].map({"No":0,"Yes":1})
df['Married'].value_counts()
```

```
1    401
0    213
```

```
Name: Married, dtype: int64
```

```
#convertir Categorical Data to numerical (Self_Employed)
df['Self_Employed']=df['Self_Employed'].map({"No":0,"Yes":1})
df['Self_Employed'].value_counts()
```

```
0    532
1     82
Name: Self_Employed, dtype: int64
```

```
#convertir Categorical Data to numerical (Dependents)
df['Dependents']=df['Dependents'].map({"0":0,"1":1,"2":2,"3+":3})
df['Dependents'].value_counts()
```

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

```
#convertir Categorical Data to numerical (Education)
df['Education']=df['Education'].map({"Not Graduate":0,"Graduate":1})
df['Education'].value_counts()
```

```
1    480
0    134
Name: Education, dtype: int64
```

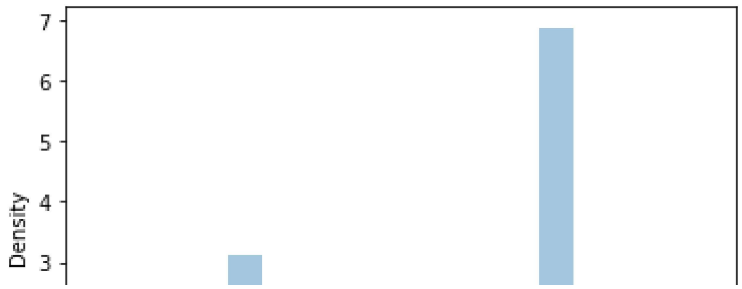
```
#convertir Categorical Data to numerical (Loan_Status)
df['Loan_Status']=df['Loan_Status'].map({"N":0,"Y":1})
df['Loan_Status'].value_counts()
```

```
1    422
0    192
Name: Loan_Status, dtype: int64
```

```
#Splitting Data:train/test
x=df[['Gender','Married','Dependents','Education','Self_Employed','ApplicantIncome','Coapp
       'Loan_Amount_Term', 'Credit_History']]
y=df['Loan_Status'].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=40)
```

```
sns.distplot(df['Loan_Status'], bins=10, kde=True)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning:
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f620154a050>
```



Model : Logistic Regression


```
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred  = logreg.predict(x_test)
print("Accuracy={:.2f}".format(logreg.score(x_test, y_test)))
```

```
Accuracy=0.82
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:940: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

```
x=df[['Gender','Married','Education','Self_Employed','Credit_History']]
y=df['Loan_Status'].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=40)
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred  = logreg.predict(x_test)
print("Accuracy={:.2f}".format(logreg.score(x_test, y_test)))
```

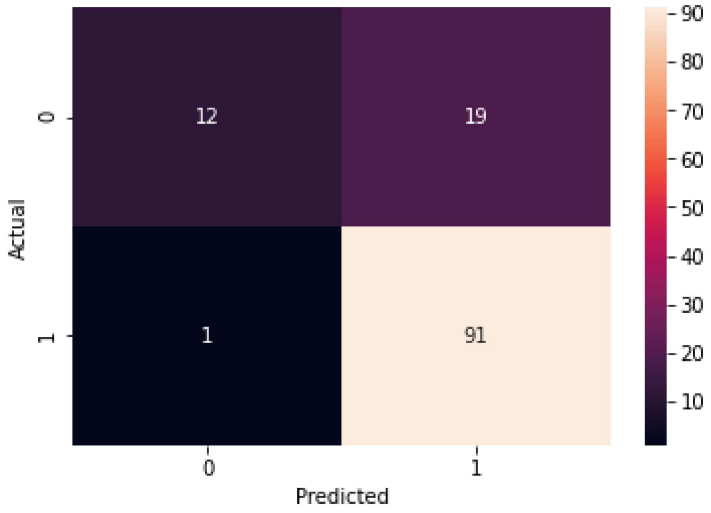
```
Accuracy=0.84
```

```
sns.regplot(x='Credit_History',y='Loan_Status',data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61fe208150>
```


```
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'])
sns.heatmap(confusion_matrix, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f61fe12bb90>
```




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

```
                precision    recall  f1-score   support

             0       0.92      0.39      0.55        31
             1       0.83      0.99      0.90        92

      accuracy                           0.84       123
     macro avg       0.88      0.69      0.72       123
  weighted avg       0.85      0.84      0.81       123
```

Model : KNN

```
knn=KNeighborsClassifier(n_neighbors=20) #build our knn classifier
knn.fit(x_train,y_train) #Training KNN classifier
y_pred=knn.predict(x_test)  #Testing
print('Acuuracy=',accuracy_score(y_pred,y_test))
```

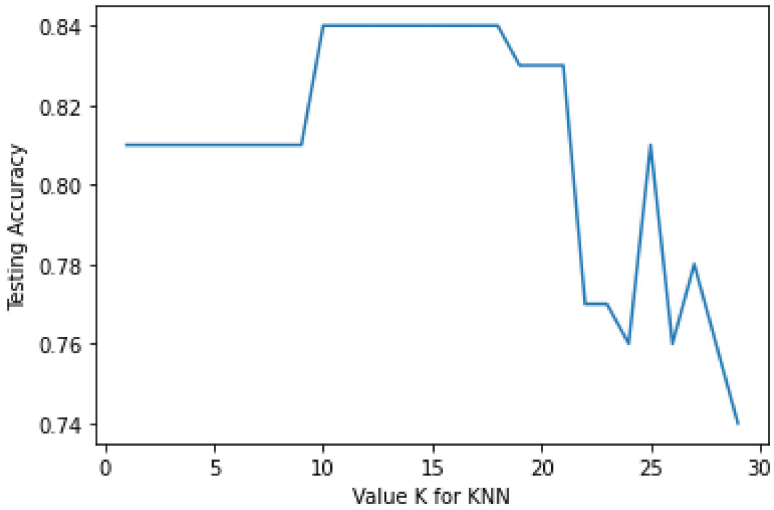
```
Acuuracy= 0.8373983739837398
```

```
n_neighbors=30
scores=[]
for k in range(1,30):
    knn=KNeighborsClassifier(n_neighbors-k)
    knn.fit(x_train,y_train)
    y_pred=knn.predict(x_test)
    print('Accuracy for k=',k,'is:',round(accuracy_score(y_pred,y_test),2))
    scores.append(round(accuracy_score(y_pred,y_test),2))
```

```
Accuracy for k= 1 is: 0.81
Accuracy for k= 2 is: 0.81
Accuracy for k= 3 is: 0.81
Accuracy for k= 4 is: 0.81
Accuracy for k= 5 is: 0.81
Accuracy for k= 6 is: 0.81
Accuracy for k= 7 is: 0.81
Accuracy for k= 8 is: 0.81
Accuracy for k= 9 is: 0.81
Accuracy for k= 10 is: 0.84
Accuracy for k= 11 is: 0.84
Accuracy for k= 12 is: 0.84
Accuracy for k= 13 is: 0.84
Accuracy for k= 14 is: 0.84
Accuracy for k= 15 is: 0.84
Accuracy for k= 16 is: 0.84
Accuracy for k= 17 is: 0.84
Accuracy for k= 18 is: 0.84
Accuracy for k= 19 is: 0.83
Accuracy for k= 20 is: 0.83
Accuracy for k= 21 is: 0.83
Accuracy for k= 22 is: 0.77
Accuracy for k= 23 is: 0.77
Accuracy for k= 24 is: 0.76
Accuracy for k= 25 is: 0.81
Accuracy for k= 26 is: 0.76
Accuracy for k= 27 is: 0.78
Accuracy for k= 28 is: 0.76
Accuracy for k= 29 is: 0.74
```

```
plt.plot(range(1,30),scores)
plt.xlabel('Value K for KNN')
plt.ylabel('Testing Accuracy')
```

```
Text(0, 0.5, 'Testing Accuracy')
```



Model : Decision Tree

```
t = tree.DecisionTreeClassifier(criterion="gini",splitter='random',max_leaf_nodes=10,min_s
t.fit(x_train, y_train)   #fitting our model
y_pred=t.predict(x_test)   # evaluating our model
print("score:{}".format(accuracy_score(y_test, y_pred)))
```

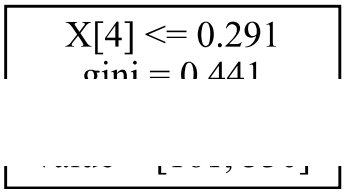
```
score:0.8373983739837398
```

Model : RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
clf=RandomForestClassifier(n_estimators=10)
clf.fit(x_train, y_train)
y_pred=clf.predict(x_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8292682926829268
```

```
import graphviz
from sklearn.tree import export_graphviz
dot_data=tree.export_graphviz(t,out_file=None)
graph=graphviz.Source(dot_data)
graph.render('data')
graph
```



Unsupervised Learning:

```
df1=df[['Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History']]
model=AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='complete')
clust_labels=model.fit_predict(df1)
agglomerative=pd.DataFrame(clust_labels)
agglomerative
```

	0
0	1
1	1
2	1
3	1
4	1
...	...
609	1
610	1
611	4
612	4
613	1

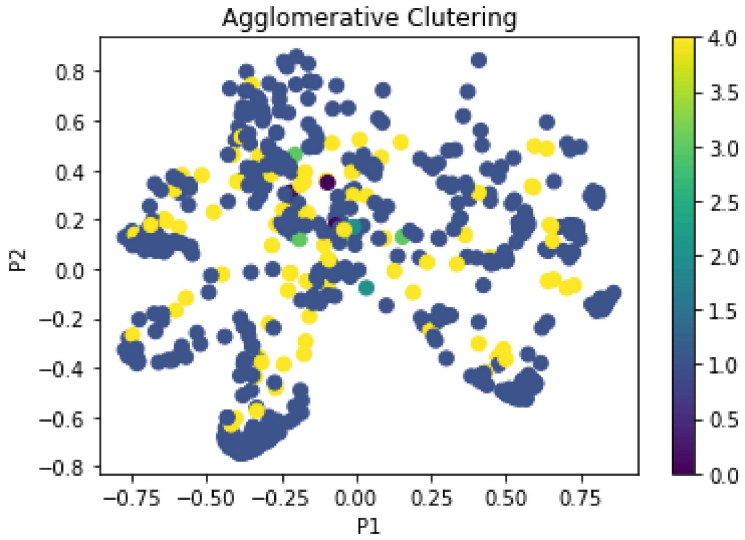
614 rows × 1 columns

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df1)
normalized_df = normalize(scaled_df)
normalized_df = pd.DataFrame(normalized_df)
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(normalized_df)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
X_principal.head(2)
```

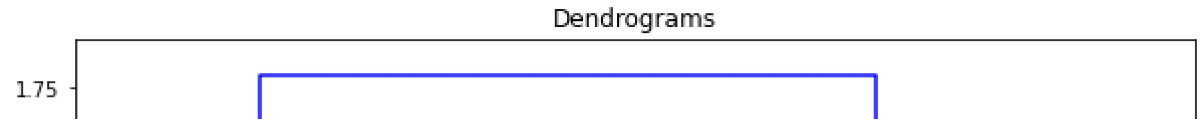
	P1	P2

```
import matplotlib.pyplot as plt
fig =plt.figure()
ax = fig.add_subplot(111)
scatter = ax.scatter(X_principal['P1'],X_principal['P2'], c= agglomerative[0], s=50)
ax.set_title("Agglomerative Clutering")
ax.set_xlabel("P1")
ax.set_ylabel("P2")
plt.colorbar(scatter)
```

```
<matplotlib.colorbar.Colorbar at 0x7f61fdcd89d0>
```




```
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10,7))
plt.title("Dendrograms")
dend=shc.dendrogram(shc.linkage(X_principal, method="complete"))
```




```
from sklearn.cluster import KMeans  #Importing our clustering algorithm: KMeans
kmeans=KMeans(n_clusters=5, random_state=0)  #Cluster our data by choosing 5 as number of
kmeans.fit(df1)
labels=pd.DataFrame(kmeans.labels_)
labels
```

	0
0	2
1	2
2	2
3	2
4	2
...	...
609	2
610	2
611	2
612	2
613	2

614 rows × 1 columns

```
kmeans.predict(df1)
print(kmeans.cluster_centers_)
```

```
[[1.94444444e-01 6.52777778e-01 8.47222222e-01 9.72222222e-01
  3.19444444e-01 1.26745278e+04 8.57763889e+02 2.45406062e+02
  3.43333333e+02 8.82313830e-01]
 [0.00000000e+00 8.00000000e-01 1.60000000e+00 1.00000000e+00
  2.00000000e-01 4.04948000e+04 9.50000000e+02 3.66400000e+02
  3.12000000e+02 8.00000000e-01]
 [1.82674200e-01 6.53483992e-01 7.15630885e-01 7.51412429e-01
  1.07344633e-01 3.86439736e+03 1.53188685e+03 1.29498740e+02
  3.42576271e+02 8.37553926e-01]
 [2.50000000e-01 2.50000000e-01 1.25000000e+00 1.00000000e+00
  2.50000000e-01 1.58800000e+03 2.88760000e+04 1.95500000e+02
  3.15000000e+02 9.60549645e-01]
 [0.00000000e+00 1.00000000e+00 1.50000000e+00 1.00000000e+00
  0.00000000e+00 7.21685000e+04 0.00000000e+00 4.25000000e+02
  2.70000000e+02 5.00000000e-01]]
```

```
from sklearn.cluster import KMeans
import sklearn.cluster as cluster
import time
import matplotlib.pyplot as plt
```

```
distortions = []
K = range(1,6)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(df1)
    distortions.append(kmeanModel.inertia_)
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.show()
```