In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import precision_score,recall_score,accuracy_score,conf
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier,AdaBoostClassifier,RandomFore
```

In [2]:
```
df=pd.read_csv("C:/Users/hp/Downloads/loan_data.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
0	LP001003	Male	Yes	1	Graduate	No	4583	
1	LP001005	Male	Yes	0	Graduate	Yes	3000	
2	LP001006	Male	Yes	0	Not Graduate	No	2583	
3	LP001008	Male	No	0	Graduate	No	6000	
4	LP001013	Male	Yes	0	Not Graduate	No	2333	

In [4]:
```
df.columns
```

Out[4]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmoun
t',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Statu
s'],
      dtype='object')
```

In [5]:
```
df.info()          # information of dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381 entries, 0 to 380
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            381 non-null    object 
 1   Gender             376 non-null    object 
 2   Married            381 non-null    object 
 3   Dependents         373 non-null    object 
 4   Education          381 non-null    object 
 5   Self_Employed      360 non-null    object 
 6   ApplicantIncome    381 non-null    int64  
 7   CoapplicantIncome  381 non-null    float64
 8   LoanAmount         381 non-null    float64
 9   Loan_Amount_Term   370 non-null    float64
 10  Credit_History     351 non-null    float64
 11  Property_Area      381 non-null    object 
 12  Loan_Status        381 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 38.8+ KB
```

In [6]:
```
df.dtypes          # Types of DataTypes present in DataSet
```

Out[6]:
```
Loan_ID              object
Gender               object
Married              object
Dependents           object
Education            object
Self_Employed        object
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area        object
Loan_Status          object
dtype: object
```

In [7]:
```
df.shape           # Shape of Dataset i.e Number of Rows and Columns Preser
```

Out[7]:
```
(381, 13)
```

In [8]:
```
df.size            # Size of DataSet i.e Total number of Elements Present 
```

Out[8]:
```
4953
```

In [9]:
```
df.isnull().sum()       # Checking Null Values present in dataset
```

Out[9]:
```
Loan_ID               0
Gender                5
Married               0
Dependents            8
Education             0
Self_Employed        21
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term     11
Credit_History       30
Property_Area         0
Loan_Status           0
dtype: int64
```

In [10]:
```
df1=df.dropna()       # Dropping Null Values present in Dataset
```

In [11]:
```
df1.isnull().sum()
```

Out[11]:
```
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 [12]:
```
df1.dtypes
```

Out[12]:
```
Loan_ID              object
Gender               object
Married              object
Dependents           object
Education            object
Self_Employed        object
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Loan_Amount_Term    float64
Credit_History      float64
Property_Area        object
Loan_Status          object
dtype: object
```

Performing LabelEncoding On DataSet

In [13]:
```
le=LabelEncoder()          # To convert categorical data into numerical data
```

In [14]:
```
df1["Loan_ID"]=le.fit_transform(df1["Loan_ID"])
df1["Gender"]=le.fit_transform(df1["Gender"])
df1["Married"]=le.fit_transform(df1["Married"])
df1["Dependents"]=le.fit_transform(df1["Dependents"])
df1["Education"]=le.fit_transform(df1["Education"])
df1["Self_Employed"]=le.fit_transform(df1["Self_Employed"])
df1["Property_Area"]=le.fit_transform(df1["Property_Area"])
df1["Loan_Status"]=le.fit_transform(df1["Loan_Status"])
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:1: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Loan_ID"]=le.fit_transform(df1["Loan_ID"])
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:2: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Gender"]=le.fit_transform(df1["Gender"])
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:3: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Married"]=le.fit_transform(df1["Married"])
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:4: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Dependents"]=le.fit_transform(df1["Dependents"])
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:5: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Education"]=le.fit_transform(df1["Education"])
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:6: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Self_Employed"]=le.fit_transform(df1["Self_Employed"])
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:7: SettingWit
```

```
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df1["Property_Area"]=le.fit_transform(df1["Property_Area"])
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\4008765204.py:8: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df1["Loan_Status"]=le.fit_transform(df1["Loan_Status"])
```

In [15]:
```
df1.head()
```

Out[15]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coap
0	0	1	1	1	0	0	4583	
1	1	1	1	0	0	1	3000	
2	2	1	1	0	1	0	2583	
3	3	1	0	0	0	0	6000	
4	4	1	1	0	1	0	2333	

In [16]:
```
df1.describe().T       # Statistical Summary Of Dataset
```

Out[16]:

	count	mean	std	min	25%	50%	75%	ma
Loan_ID	308.0	153.500000	89.056162	0.0	76.75	153.5	230.25	307.
Gender	308.0	0.795455	0.404025	0.0	1.00	1.0	1.00	1.
Married	308.0	0.600649	0.490562	0.0	0.00	1.0	1.00	1.
Dependents	308.0	0.678571	0.997029	0.0	0.00	0.0	1.00	3.
Education	308.0	0.256494	0.437408	0.0	0.00	0.0	1.00	1.
Self_Employed	308.0	0.090909	0.287948	0.0	0.00	0.0	0.00	1.
ApplicantIncome	308.0	3599.126623	1462.359612	150.0	2568.75	3329.5	4291.00	9703.
CoapplicantIncome	308.0	1278.434805	2520.961308	0.0	0.00	871.5	1953.50	33837.
LoanAmount	308.0	104.623377	29.382256	9.0	89.75	110.0	128.00	150.
Loan_Amount_Term	308.0	341.181818	68.246006	36.0	360.00	360.0	360.00	480.
Credit_History	308.0	0.853896	0.353785	0.0	1.00	1.0	1.00	1.
Property_Area	308.0	1.042208	0.775125	0.0	0.00	1.0	2.00	2.
Loan_Status	308.0	0.711039	0.454017	0.0	0.00	1.0	1.00	1.

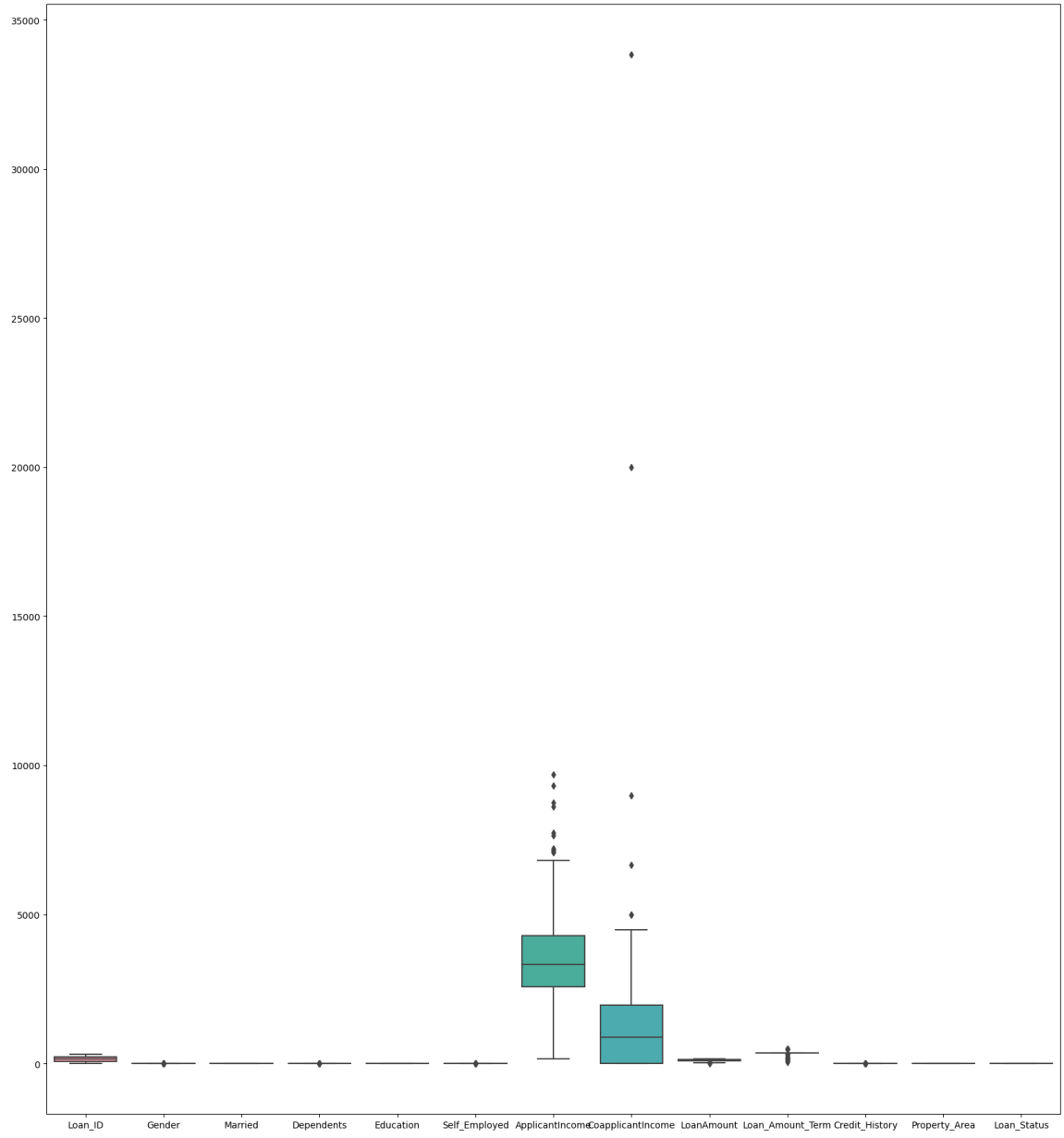
Changing DataTypes of Data in Dataset

In [17]:
```
df1["CoapplicantIncome"]=df1["CoapplicantIncome"].astype("int")
df1["LoanAmount"]=df1["LoanAmount"].astype("int")
df1["Loan_Amount_Term"]=df1["Loan_Amount_Term"].astype("int")
df1["Credit_History"]=df1["Credit_History"].astype("int")
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\1689938164.py:1: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["CoapplicantIncome"]=df1["CoapplicantIncome"].astype("int")
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\1689938164.py:2: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["LoanAmount"]=df1["LoanAmount"].astype("int")
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\1689938164.py:3: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Loan_Amount_Term"]=df1["Loan_Amount_Term"].astype("int")
C:\Users\hp\AppData\Local\Temp\ipykernel_24160\1689938164.py:4: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
view-versus-a-copy)
  df1["Credit_History"]=df1["Credit_History"].astype("int")
```

Removing Outliers present in Dataset

In [18]:
```
plt.figure(figsize=(20,22))        # Ploting Boxplot to detect outliers
sns.boxplot(data=df1)
plt.show()
```



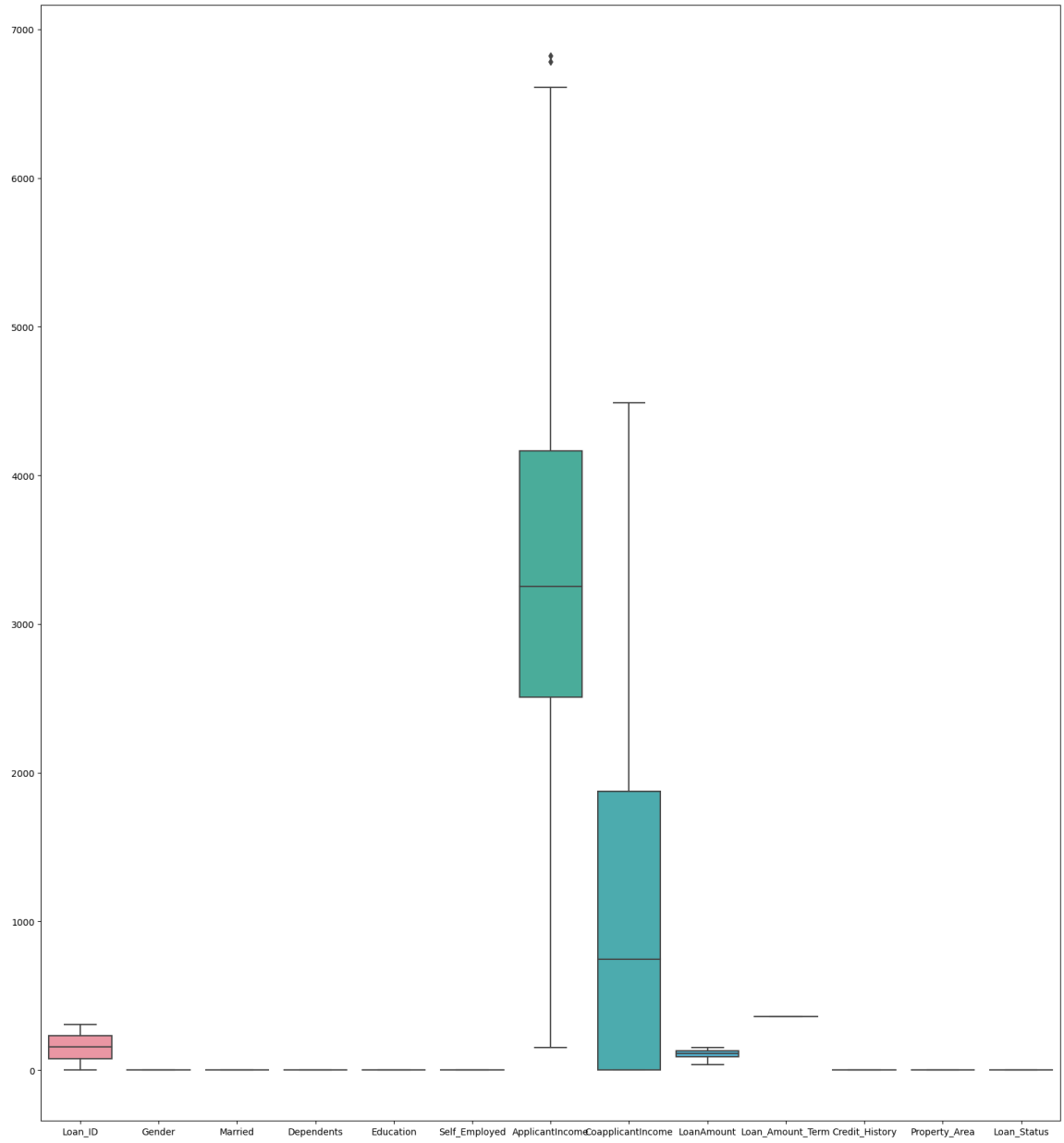
Steps to Remove Outliers

In [19]:
```
Q1=df1.quantile(q=0.25)          # finding Q1 value
Q3=df1.quantile(q=0.75)          # Finding Q3 value
IQR=Q3-Q1                        # Finding IQR Value i.e(InterQuantileRange)
upper=Q3+(1.5*IQR)               # to detect upper outliers
lower=Q1-(1.5*IQR)               # to detect lower outliers
```

In [20]:
```
df2=df1[~((df1>upper)|(df1<lower))] #(~) this is use for as we want both < >
                                    # if we not use ~ it will take outlier
```

In [21]:
```
plt.figure(figsize=(20,22))          # Boxplot after removing Outliers
sns.boxplot(df2)
```

Out[21]: <Axes: >



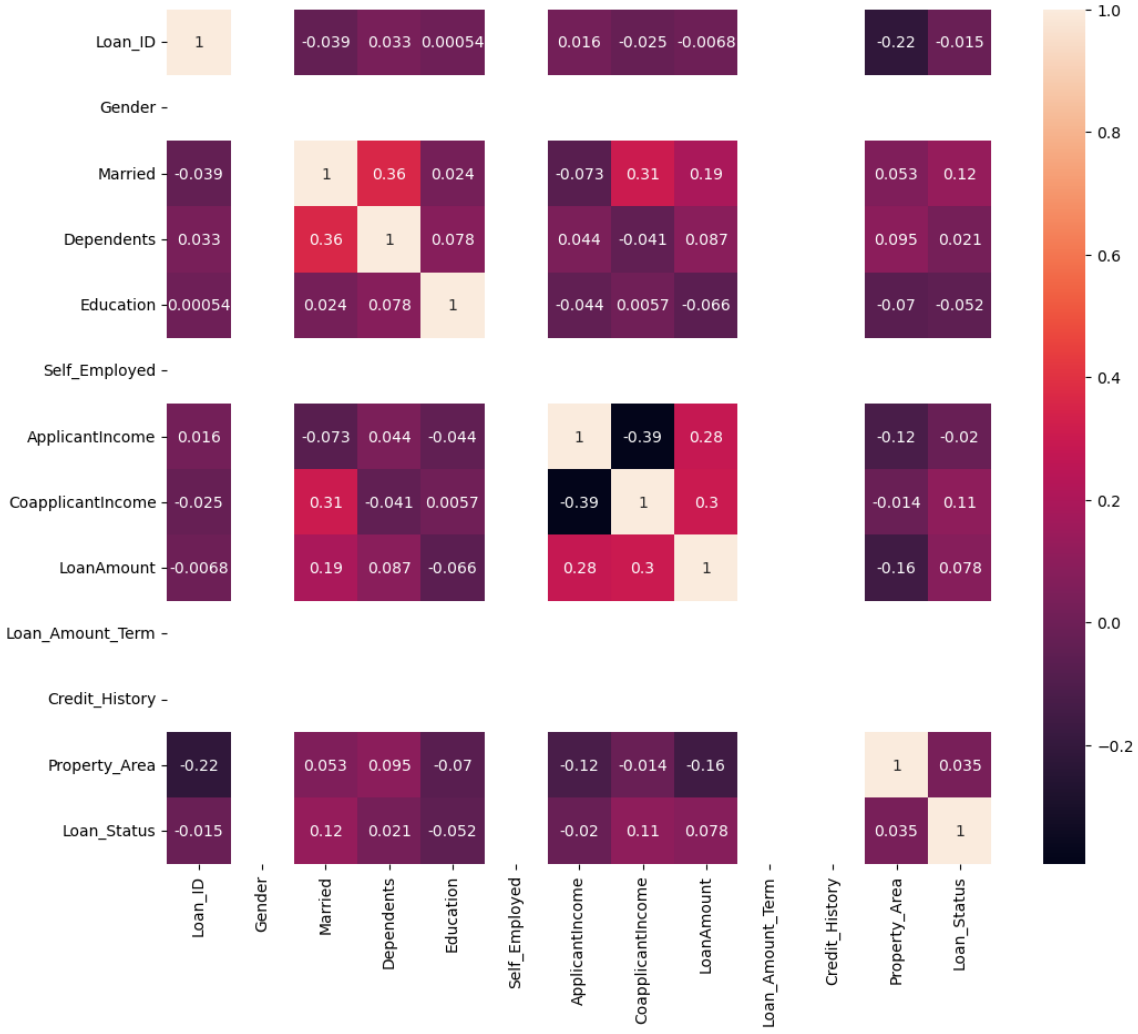
In [22]:
```
df2.isnull().sum()       # After removing outlier we get some nan values
```

Out[22]:
```
Loan_ID               0
Gender               63
Married               0
Dependents           24
Education             0
Self_Employed        28
ApplicantIncome      11
CoapplicantIncome     6
LoanAmount            7
Loan_Amount_Term     49
Credit_History       45
Property_Area         0
Loan_Status           0
dtype: int64
```

In [23]:
```
df3=df2.dropna()     # removing that nan values
```

HeatMap To Show Corelation between Data

In [24]:
```
plt.figure(figsize=(12,10))
sns.heatmap(df2.corr(),annot=True)
plt.show()
```

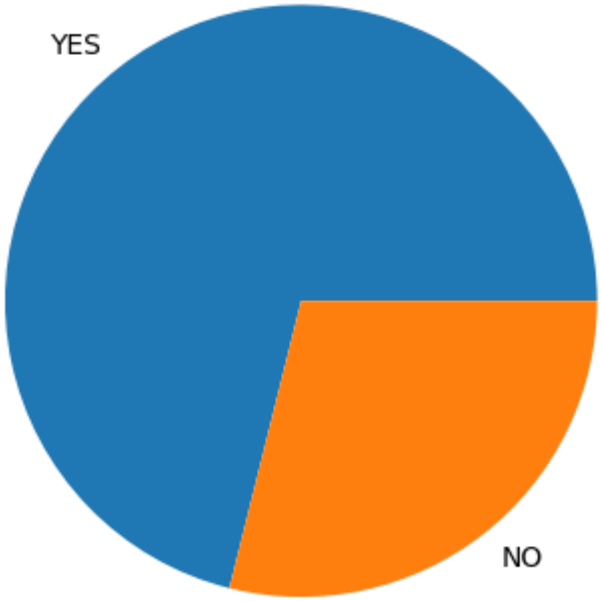


In [25]:
```
print(df1["Loan_Status"].unique())
print(df1["Loan_Status"].value_counts(normalize=True)*100)
```

```
[0 1]
Loan_Status
1    71.103896
0    28.896104
Name: proportion, dtype: float64
```

In [26]:
```
plt.pie(df2["Loan_Status"].value_counts(normalize=True)*100,labels=["YES","N
```

Out[26]:
```
([<matplotlib.patches.Wedge at 0x1e5d999fc10>,
  <matplotlib.patches.Wedge at 0x1e5da680bd0>],
 [Text(-0.6770310906241628, 0.866965340903693, 'YES'),
  Text(0.6770310906241627, -0.8669653409036931, 'NO')])
```



Model Building for DataSet

In [65]:
```
x=df3.drop(["Loan_Status"],axis=1)
y=df3["Loan_Status"]
```

In [66]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_stat
```

In [67]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(99, 12)
(43, 12)
(99,)
(43,)
```

LOGISTIC REGRESSION ALGORITHM

In [68]:
```
le=LogisticRegression()
le.fit(x_train,y_train)
```

Out[68]:
```
LogisticRegression()
```

In [69]:
```
y_true,y_pred=y_test,le.predict(x_test)
print(le.score(x_train,y_train)*100)
print(le.score(x_test,y_test)*100)
```

```
86.86868686868688
88.37209302325581
```

In [70]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
88.37209302325581
100.0
88.37209302325581
```

RANDOM FOREST CLASSIFIER ALGORITHM

In [71]:
```
rf=RandomForestClassifier(n_estimators=6,random_state=1)
rf.fit(x_train,y_train)
```

Out[71]:
```
RandomForestClassifier(n_estimators=6, random_state=1)
```

In [72]:
```
y_true,y_pred=y_test,rf.predict(x_test)
print(rf.score(x_train,y_train)*100)
print(rf.score(x_test,y_test)*100)
```

```
95.95959595959596
83.72093023255815
```

In [73]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
87.8048780487805
94.73684210526315
83.72093023255815
```

DECISION TREE CLASSIFIER ALGORITHM

In [74]:
```
dt=DecisionTreeClassifier(criterion="gini",max_depth=4,random_state=1)
dt.fit(x_train,y_train)
```

Out[74]:
```
DecisionTreeClassifier(max_depth=4, random_state=1)
```

In [75]:
```
y_true,y_pred=y_test,dt.predict(x_test)
print(dt.score(x_train,y_train)*100)
print(dt.score(x_test,y_test)*100)
```

```
94.94949494949495
88.37209302325581
```

In [76]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
90.2439024390244
97.36842105263158
88.37209302325581
```

GRADIENT BOOSTING CLASSIFIER ALGORITHM

In [77]:
```
gb=GradientBoostingClassifier(n_estimators=20)
gb.fit(x_train,y_train)
```

Out[77]:
```
GradientBoostingClassifier(n_estimators=20)
```

In [78]:
```
y_true,y_pred=y_test,gb.predict(x_test)
print(gb.score(x_train,y_train)*100)
print(gb.score(x_test,y_test)*100)
```

```
94.94949494949495
90.69767441860465
```

In [79]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
90.47619047619048
100.0
90.69767441860465
```

BAGGING CLASSIFIER ALGORITHM

In [80]:
```
bg=BaggingClassifier(n_estimators=20)
bg.fit(x_train,y_train)
```

Out[80]:
```
BaggingClassifier
BaggingClassifier(n_estimators=20)
```

In [81]:
```
y_true,y_pred=y_test,bg.predict(x_test)
print(bg.score(x_train,y_train)*100)
print(gb.score(x_test,y_test)*100)
```

```
98.98989898989899
90.69767441860465
```

In [82]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
88.09523809523809
97.36842105263158
86.04651162790698
```

ADABOOST CLASSIFIER ALGORITHM

In [83]:
```
ad=AdaBoostClassifier(n_estimators=20,estimator=dt,random_state=1)
ad.fit(x_train,y_train)
```

Out[83]:
```
AdaBoostClassifier
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [84]:
```
y_true,y_pred=y_test,ad.predict(x_test)
print(ad.score(x_train,y_train)*100)
print(ad.score(x_test,y_test)*100)
```

```
100.0
90.69767441860465
```

In [85]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
90.47619047619048
100.0
90.69767441860465
```

KNeighbors CLASSIFIER ALGORITHM

In [92]:
```
kn=KNeighborsClassifier(weights="distance")
kn.fit(x_train,y_train)
```

Out[92]:
```
KNeighborsClassifier
KNeighborsClassifier(weights='distance')
```

In [93]:
```
y_true,y_pred=y_test,kn.predict(x_test)
print(kn.score(x_train,y_train)*100)
print(kn.score(x_test,y_test)*100)
```

```
100.0
81.3953488372093
```

In [94]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
87.5
92.10526315789474
81.3953488372093
```

SVC (SUPPORT VECTOR CLASSIFIER) ALGORITHM

In [89]:
```
svc=SVC(C=1.0,kernel="linear")
svc.fit(x_train,y_train)
```

Out[89]:
```
SVC
SVC(kernel='linear')
```

In [90]:
```
y_true,y_pred=y_test,svc.predict(x_test)
print(svc.score(x_train,y_train)*100)
print(svc.score(x_test,y_test)*100)
```

```
87.87878787878788
88.37209302325581
```

In [91]:
```
print(precision_score(y_true,y_pred)*100)
print(recall_score(y_true,y_pred)*100)
print(accuracy_score(y_true,y_pred)*100)
```

```
88.37209302325581
100.0
88.37209302325581
```