Importing the Dependencies

In [33]:

```
from sklearn import svm
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

Data Collection and Processing

In [5]:

```
# loading the dataset to pandas DataFrame
loan_dataset = pd.read_csv('/content/train_u6lujuX_CVtuZ9i (1).csv')
```

In [6]:

```
type(loan_dataset)
```

Out[6]:

```
pandas.core.frame.DataFrame
```

In [7]:

```
# printing the first 5 rows of the dataframe
loan_dataset.head()
```

Out[7]:

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

In [8]:

```
# number of rows and columns
loan_dataset.shape
```

Out[8]:

```
(614, 13)
```

In [9]:

```
# statistical measures
loan_dataset.describe()
```

Out[9]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.00000
mean	5403.459283	1621.245798	146.412162	342.00000	0.84219
std	6109.041673	2926.248369	85.587325	65.12041	0.36487
min	150.000000	0.000000	9.000000	12.00000	0.00000
25%	2877.500000	0.000000	100.000000	360.00000	1.00000
50%	3812.500000	1188.500000	128.000000	360.00000	1.00000
75%	5795.000000	2297.250000	168.000000	360.00000	1.00000
max	81000.000000	41667.000000	700.000000	480.00000	1.00000

In [10]:

```
# number of missing values in each column
loan_dataset.isnull().sum()
```

Out[10]:

```
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
```

In [11]:

```
# dropping the missing values
loan_dataset = loan_dataset.dropna()
```

In [12]:

```
# number of missing values in each column
loan_dataset.isnull().sum()
```

Out[12]:

```
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 [13]:

```
# label encoding
loan_dataset.replace({"Loan_Status":{'N':0,'Y':1}},inplace=True)
```

In [14]:

```
# printing the first 5 rows of the dataframe
loan_dataset.head()
```

Out[14]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Co
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	
4	LP001008	Male	No	0	Graduate	No	6000	
5	LP001011	Male	Yes	2	Graduate	Yes	5417	

In [15]:

```
# Dependent column values
loan_dataset['Dependents'].value_counts()
```

Out[15]:

```
0     274
2      85
1      80
3+     41
Name: Dependents, dtype: int64
```

In [16]:

```
# replacing the value of 3+ to 4
loan_dataset = loan_dataset.replace(to_replace='3+', value=4)
```

In [17]:

```
# dependent values
loan_dataset['Dependents'].value_counts()
```

Out[17]:

```
0    274
2     85
1     80
4     41
Name: Dependents, dtype: int64
```

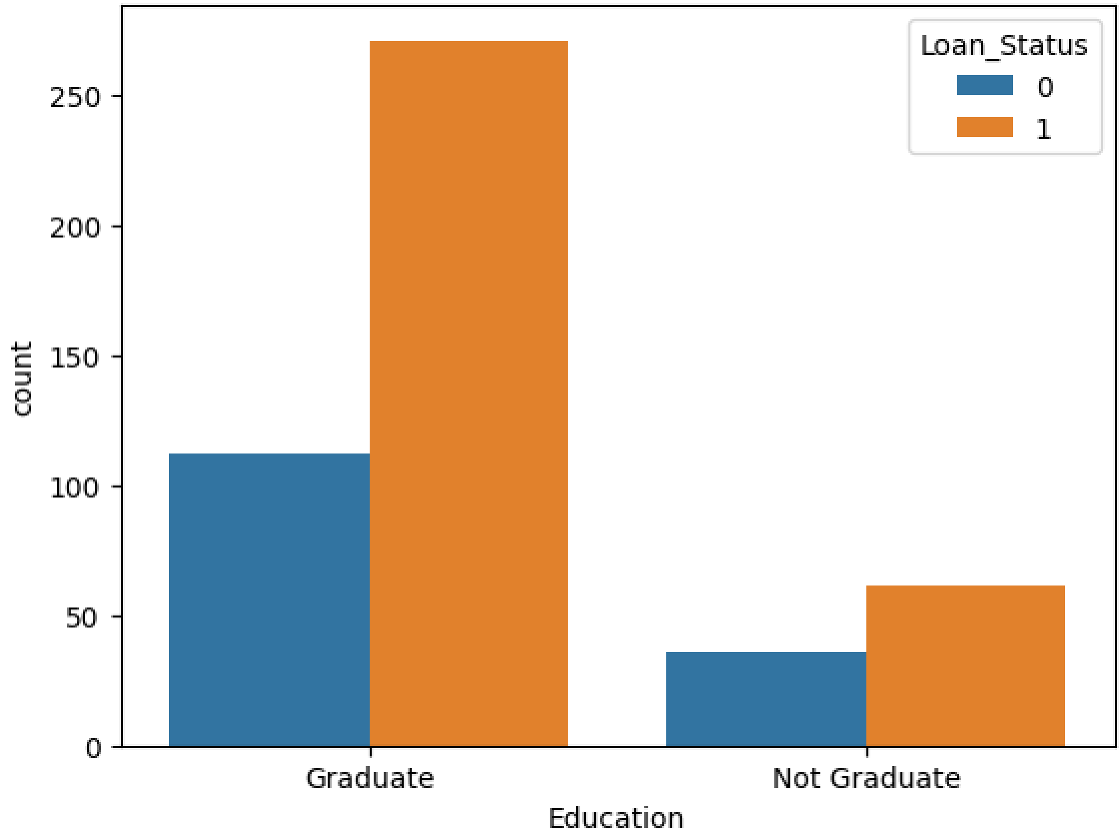
Data Visualization

In [18]:

```
# education & Loan Status
sns.countplot(x='Education',hue='Loan_Status',data=loan_dataset)
```

Out[18]:

```
<Axes: xlabel='Education', ylabel='count'>
```

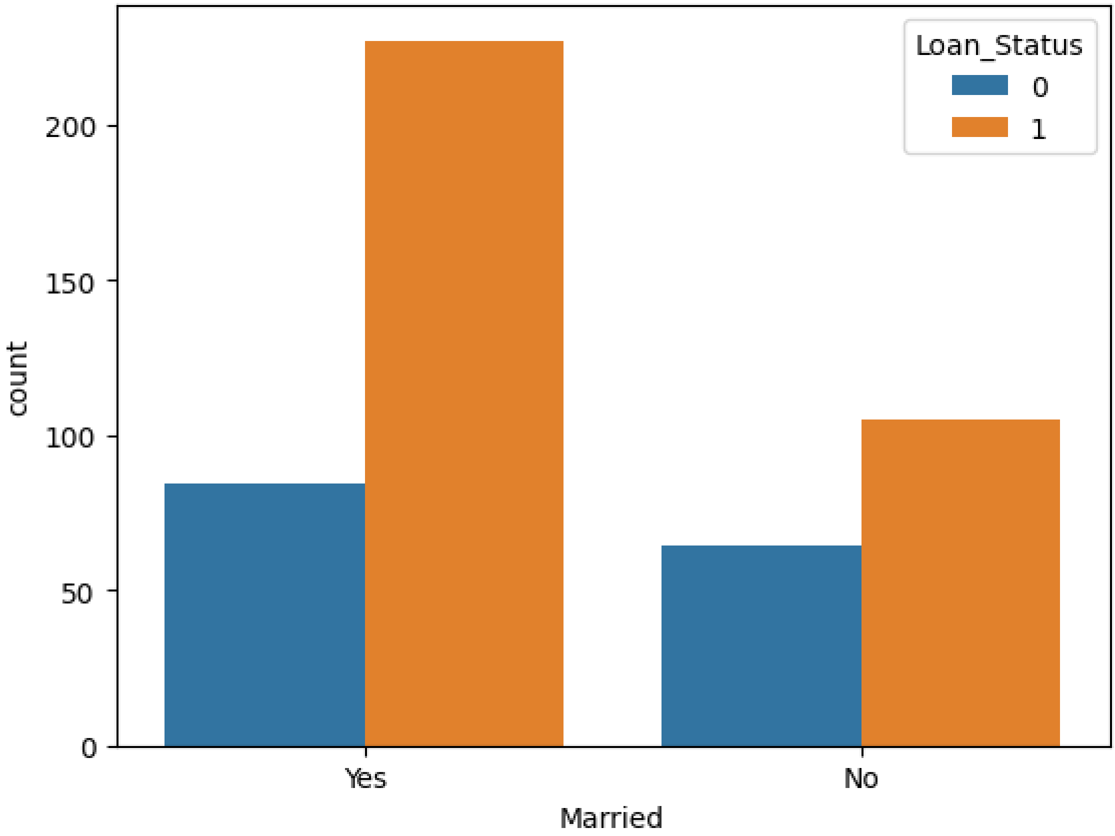


In [19]:

```
# marital status & Loan Status
sns.countplot(x='Married',hue='Loan_Status',data=loan_dataset)
```

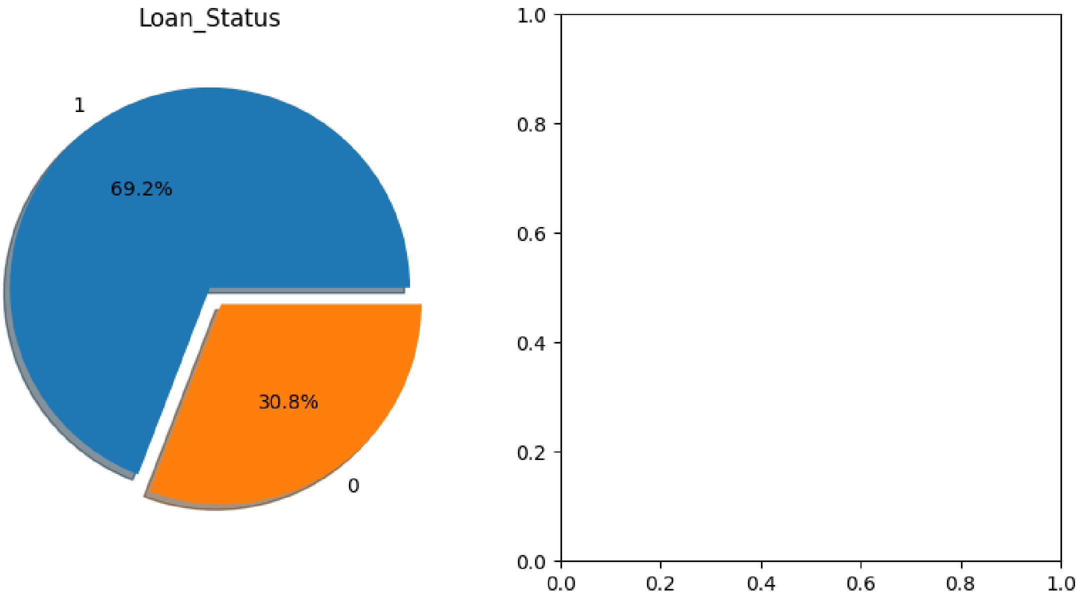
Out[19]:

```
<Axes: xlabel='Married', ylabel='count'>
```



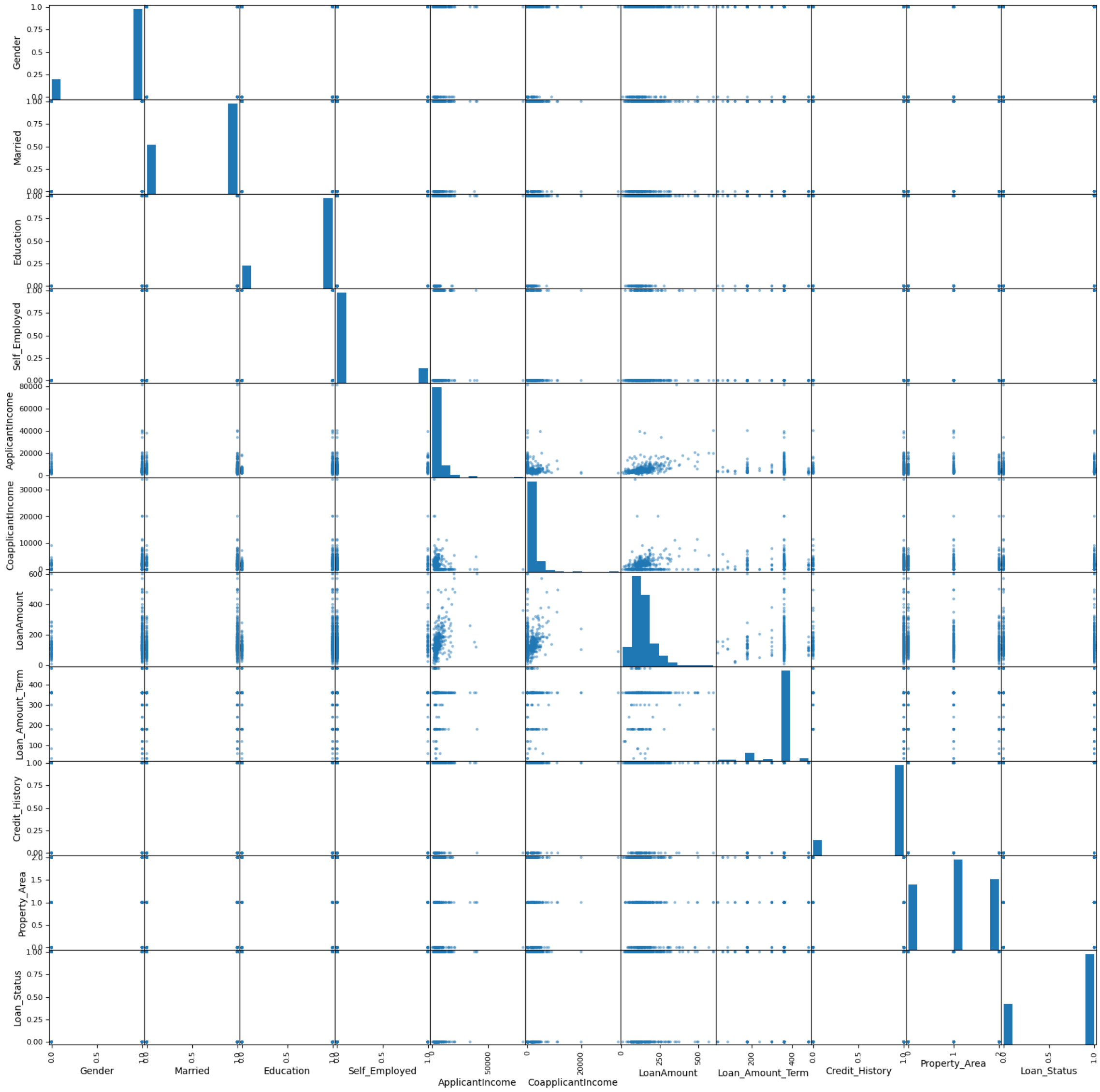
In [35]:

```
f,ax=plt.subplots(1,2,figsize=(10,5))
loan_dataset['Loan_Status'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax
ax[0].set_title('Loan_Status')
ax[0].set_ylabel('')
plt.show()
```



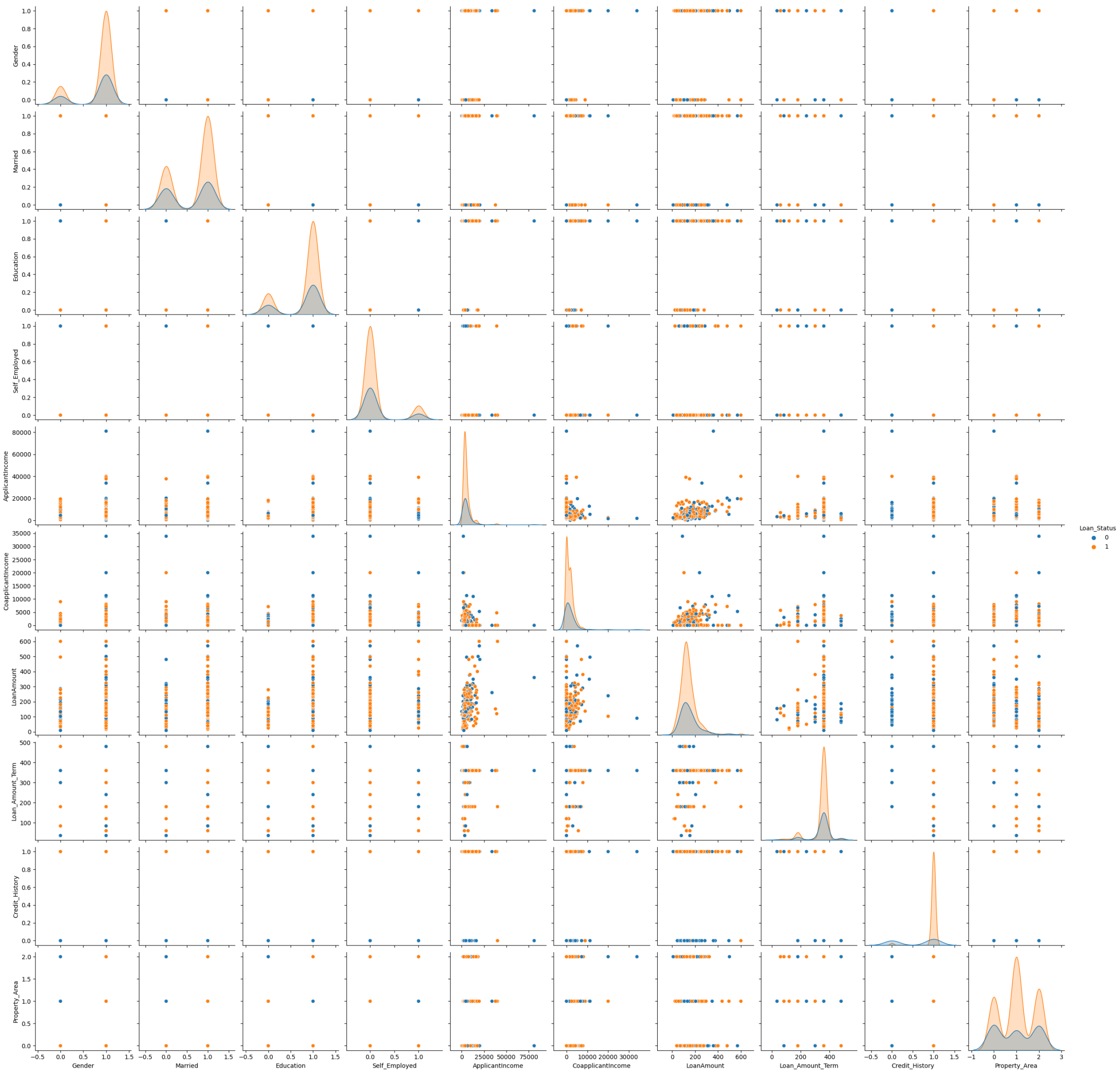
In [37]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(loan_dataset,figsize=(20,20));
```



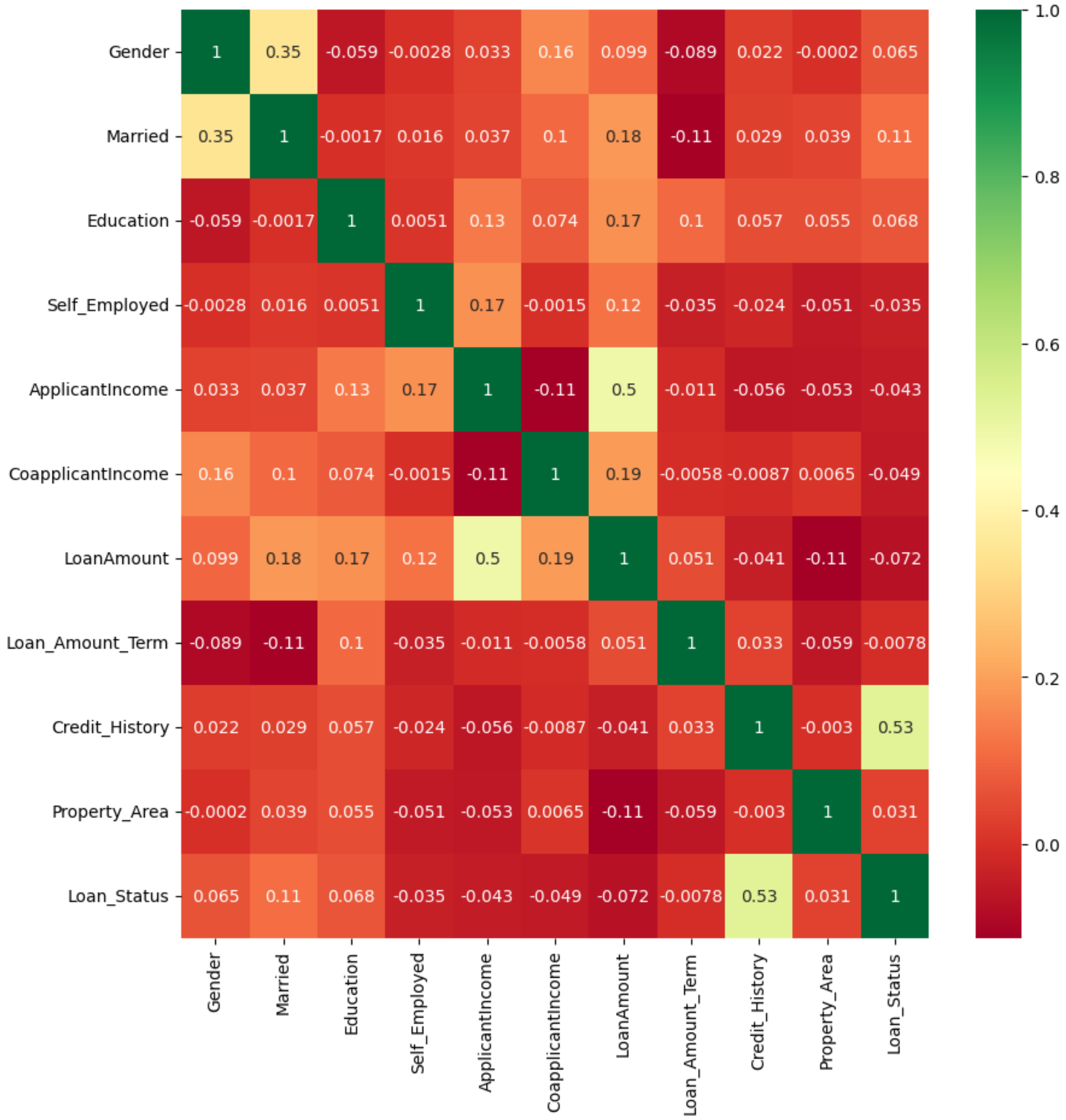
In [38]:

```
sns.pairplot(data=loan_dataset,hue='Loan_Status')
plt.show()
```



In [39]:

```
cormat=loan_dataset.corr()
top_corr_features=cormat.index
plt.figure(figsize=(10,10))
g=sns.heatmap(loan_dataset[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [20]:

```
# convert categorical columns to numerical values
loan_dataset.replace({'Married':{'No':0,'Yes':1},'Gender':{'Male':1,'Female':0},'Self_Em
                      'Property_Area':{'Rural':0,'Semiurban':1,'Urban':2},'Education':{'
```

In [21]:

```
loan_dataset.head()
```

Out[21]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coa
1	LP001003	1	1	1	1	0	4583	
2	LP001005	1	1	0	1	1	3000	
3	LP001006	1	1	0	0	0	2583	
4	LP001008	1	0	0	1	0	6000	
5	LP001011	1	1	2	1	1	5417	

In [22]:

```
# separating the data and label
X = loan_dataset.drop(columns=['Loan_ID','Loan_Status'],axis=1)
Y = loan_dataset['Loan_Status']
```

In [23]:

```
print(X)
print(Y)
```

```
     Gender  Married Dependents  Education  Self_Employed  ApplicantIncome  \
1         1        1          1          1              0             4583   
2         1        1          0          1              1             3000   
3         1        1          0          0              0             2583   
4         1        0          0          1              0             6000   
5         1        1          2          1              1             5417   
..      ...      ...        ...        ...            ...              ...   
609       0        0          0          1              0             2900   
610       1        1          4          1              0             4106   
611       1        1          1          1              0             8072   
612       1        1          2          1              0             7583   
613       0        0          0          1              1             4583   

     CoapplicantIncome  LoanAmount  Loan_Amount_Term  Credit_History  \
1               1508.0       128.0             360.0             1.0   
2                  0.0        66.0             360.0             1.0   
3               2358.0       120.0             360.0             1.0   
4                  0.0       141.0             360.0             1.0   
5               4196.0       267.0             360.0             1.0   
..                 ...         ...               ...             ...   
609                0.0        71.0             360.0             1.0   
610                0.0        40.0             180.0             1.0   
611              240.0       253.0             360.0             1.0   
612                0.0       187.0             360.0             1.0   
613                0.0       133.0             360.0             0.0   

     Property_Area  
1                0  
2                2  
3                2  
4                2  
5                2  
..             ...  
609              0  
610              0  
611              2  
612              2  
613              1  

[480 rows x 11 columns]
1      0
2      1
3      1
4      1
5      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 480, dtype: int64
```

In [44]:

```
target_name='Loan_Status'

y=loan_dataset[target_name]
X=loan_dataset.drop(target_name,axis=1)
```

In [41]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X)
SSX=scaler.transform(X)
```

Train Test Split

In [45]:

```
X_train,X_test,y_train,y_test=train_test_split(SSX,y,test_size=0.2,random_state=7)
```

In [46]:

```
X_train.shape
```

Out[46]:

```
(384, 11)
```

In [47]:

```
y_train.shape
```

Out[47]:

```
(384,)
```

In [48]:

```
X_test.shape
```

Out[48]:

```
(96, 11)
```

In [49]:

```
y_test.shape
```

Out[49]:

```
(96,)
```

In [50]:

```
from sklearn.neighbors import KNeighborsClassifier
knn= KNeighborsClassifier()
knn.fit(X_train,y_train)
```

Out[50]:

```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [51]:

```
from sklearn.svm import SVC
sv=SVC()
sv.fit(X_train,y_train)
```

Out[51]:

```
SVC()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [52]:

```
from sklearn.linear_model import LogisticRegression
lc = LogisticRegression()
lc.fit(X_train,y_train)
```

Out[52]:

```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [53]:

```
knn_pred=knn.predict(X_test)
```

In [54]:

```
sv_pred=sv.predict(X_test)
```

In [55]:

```
lc_pred=lc.predict(X_test)
```

In [56]:

```
##Accuracy
```

In [57]:

```
from sklearn.metrics import accuracy_score
```

In [58]:

```
print("Accuracy-Test of knn:",knn.score(X_train,y_train)*100)
print("Accuracy-Test of knn:",knn.score(X_test,y_test)*100)
acc_knn = accuracy_score(y_test, knn_pred)
print("Accuracy of knn:",accuracy_score(y_test,knn_pred)*100)
```

```
Accuracy-Test of knn: 80.98958333333334
Accuracy-Test of knn: 78.125
Accuracy of knn: 78.125
```

In [59]:

```
print("Accuracy-Test of Support vector:",sv.score(X_train,y_train)*100)
print("Accuracy-Test of Support vector:",sv.score(X_test,y_test)*100)
acc_sv = accuracy_score(y_test, sv_pred)
print("Accuracy of Support vector:",accuracy_score(y_test,sv_pred)*100)
```

```
Accuracy-Test of Support vector: 83.33333333333334
Accuracy-Test of Support vector: 79.16666666666666
Accuracy of Support vector: 79.16666666666666
```

In [60]:

```
print("Accuracy-Test of Linear Regression:",lc.score(X_train,y_train)*100)
print("Accuracy-Test of Linear Regression:",lc.score(X_test,y_test)*100)
acc_lc = accuracy_score(y_test, lc_pred)
print("Accuracy of Linear Regression:",accuracy_score(y_test,lc_pred)*100)
```

```
Accuracy-Test of Linear Regression: 81.51041666666666
Accuracy-Test of Linear Regression: 77.08333333333334
Accuracy of Linear Regression: 77.08333333333334
```

In [61]:

```
## CONFUSION MATRIX
```

In [62]:

```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,sv_pred)
cm
```

Out[62]:

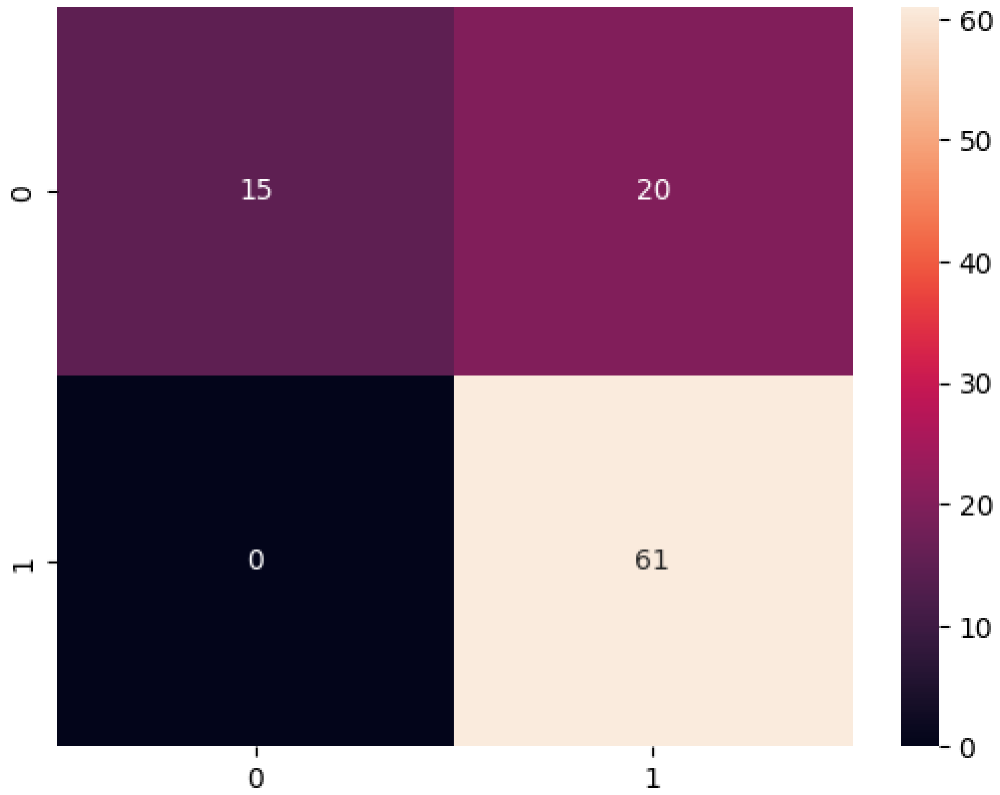
```
array([[15, 20],
       [ 0, 61]])
```

In [63]:

```
sns.heatmap(confusion_matrix(y_test,sv_pred),annot=True,fmt="d")
```

Out[63]:

```
<Axes: >
```



In [64]:

```
TN=cm[0,0]
FP=cm[0,1]
FN=cm[1,0]
TP=cm[1,1]
```

In [65]:

```
TN,FP,FN,TP
```

Out[65]:

```
(15, 20, 0, 61)
```

In [66]:

```
TP,FP
```

Out[66]:

```
(61, 20)
```

In [67]:

```
Precision=TP/(TP+FP)
Precision
```

Out[67]:

```
0.7530864197530864
```

In [68]:

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 100.0
```

In [69]:

```
from sklearn.metrics import f1_score
print('f1_score:',f1_score(y_test,sv_pred)*100)
```

```
f1_score: 85.91549295774648
```

In [70]:

```
#F1-SCORE,PRECISION,RECALL----KNN
```

In [71]:

```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,knn_pred)
cm
```

Out[71]:

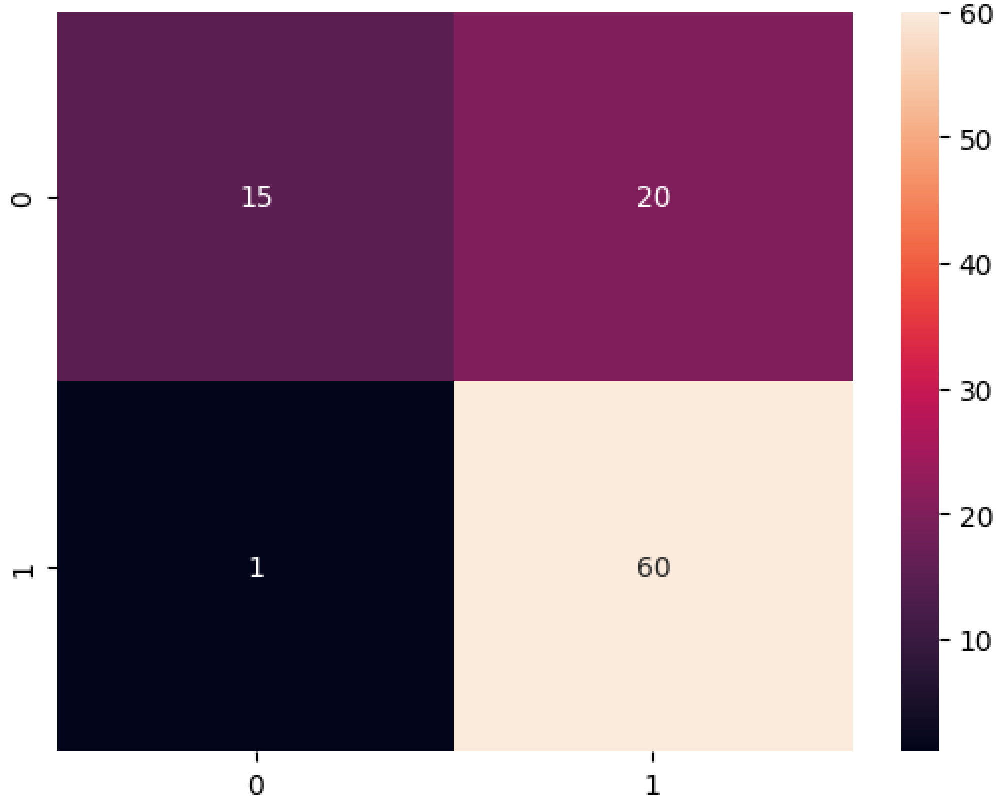
```
array([[15, 20],
       [ 1, 60]])
```

In [72]:

```
sns.heatmap(confusion_matrix(y_test,knn_pred),annot=True,fmt="d")
```

Out[72]:

```
<Axes: >
```



In [73]:

```
TN=cm[0,0]
FP=cm[0,1]
FN=cm[1,0]
TP=cm[1,1]
```

In [74]:

```
TN,FP,FN,TP
```

Out[74]:

```
(15, 20, 1, 60)
```

In [78]:

```
Precision=TP/(TP+FP)*100
Precision
```

Out[78]:

```
75.0
```

In [76]:

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 98.36065573770492
```

In [77]:

```
from sklearn.metrics import f1_score
print('f1_score:',f1_score(y_test,knn_pred)*100)
```

```
f1_score: 85.10638297872339
```

In [ ]:

```

```

#F1-SCORE,PRECISION,RECALL----LC

In [79]:

```
from sklearn.metrics import classification_report,confusion_matrix
cm=confusion_matrix(y_test,lc_pred)
cm
```

Out[79]:

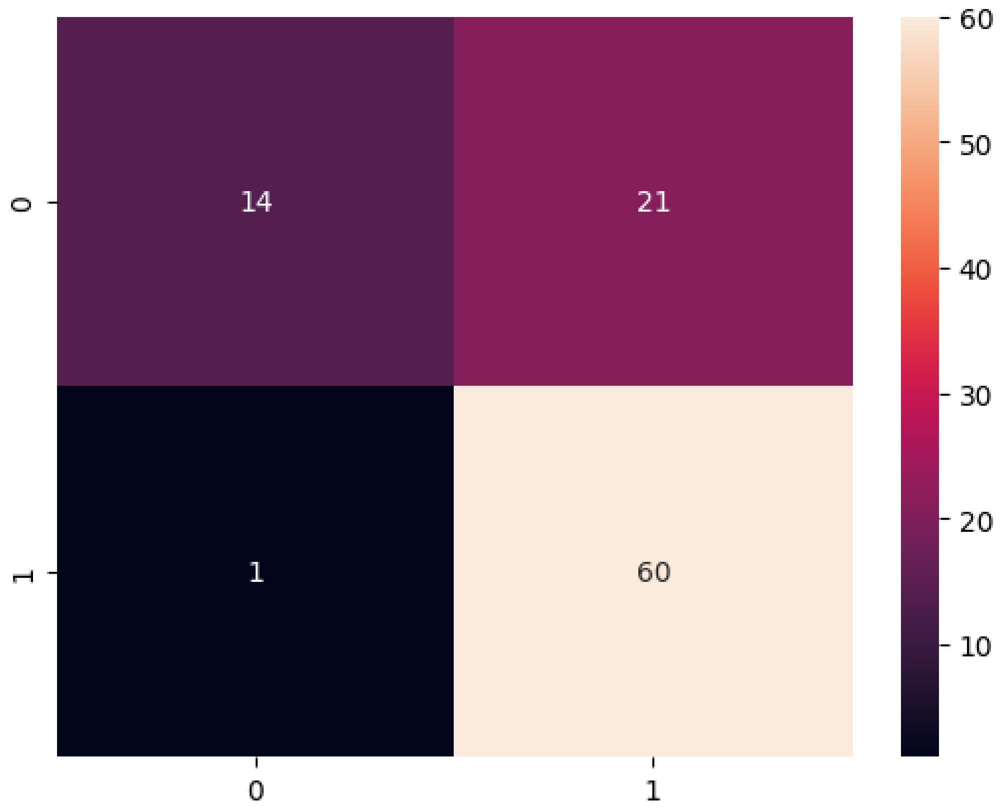
```
array([[14, 21],
       [ 1, 60]])
```

In [81]:

```
sns.heatmap(confusion_matrix(y_test,lc_pred),annot=True,fmt="d")
```

Out[81]:

<Axes: >



In [82]:

```
TN=cm[0,0]
FP=cm[0,1]
FN=cm[1,0]
TP=cm[1,1]
```

In [83]:

```
TN,FP,FN,TP
```

Out[83]:

(14, 21, 1, 60)

In [84]:

```
Precision=TP/(TP+FP)*100
Precision
```

Out[84]:

74.07407407407408

In [85]:

```
recall_score=TP/float(TP+FN)*100
print('recall_score',recall_score)
```

```
recall_score 98.36065573770492
```

In [86]:

```
from sklearn.metrics import f1_score
print('f1_score:',f1_score(y_test,lc_pred)*100)
```

```
f1_score: 84.50704225352112
```

In [93]:

```
data = {'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score'],
        'Value': [accuracy_score, Precision, recall_score, f1_score]}
df = pd.DataFrame(data)
```

In [94]:

```
print(df)
```

```
      Metric                                           Value
0   Accuracy  <function accuracy_score at 0x7f7512afddc0>
1  Precision                                       74.074074
2     Recall                                       98.360656
3   F1 Score        <function f1_score at 0x7f7512b0a160>
```