In [1]:
```
#Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import accuracy_score

import warnings
warnings.filterwarnings('ignore', category=UserWarning)
```

In [2]:
```
#Importing the dataset
loan = pd.read_csv('loan.csv')
```

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

Out[3]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	

In [4]:
```
#shape of the data
loan.shape
```

Out[4]:
```
(614, 13)
```

In [5]:
```
#Checking for null values
loan.isnull().sum()
```

Out[5]:
```
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 [6]:
```
#columns
loan.columns
```

Out[6]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [7]:
```
columns = ['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status']
```

In [8]:
```
#filling the missing values with mode
loan['Gender'].fillna(loan['Gender'].mode()[0], inplace = True)
loan['Married'].fillna(loan['Married'].mode()[0], inplace = True)
loan['Dependents'].fillna(loan['Dependents'].mode()[0], inplace = True)
loan['Self_Employed'].fillna(loan['Self_Employed'].mode()[0], inplace = True)

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

In [9]:
```
#chicking for null values again
loan.isnull().sum()
```

Out[9]:
```
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 [10]:
```
loan.head()
```

Out[10]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	L
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	

In [11]:
```
loan.shape
```

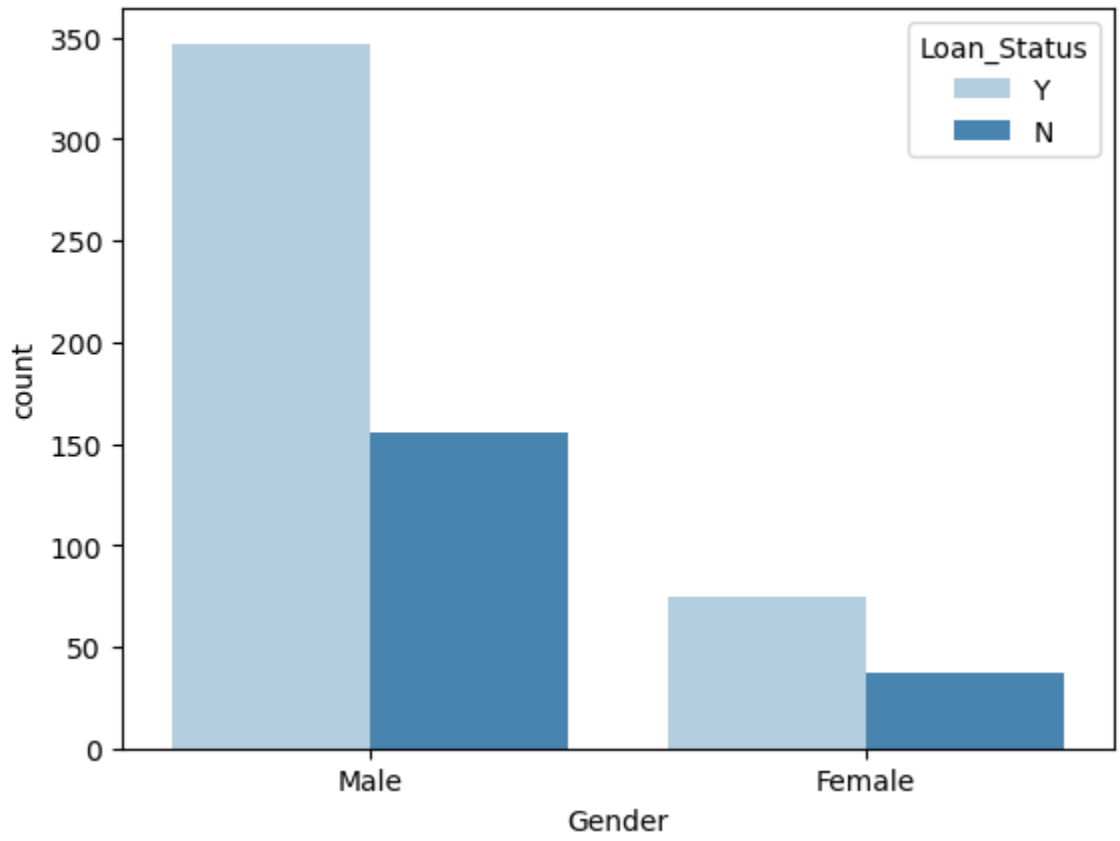
Out[11]:
```
(614, 13)
```

Visualizations and Counts

In [13]:
```
sns.countplot(x ='Gender', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking the Gender Status of loan Applicants
loan['Gender'].value_counts()
```

Out[13]:
```
Gender
Male      502
Female    112
Name: count, dtype: int64
```

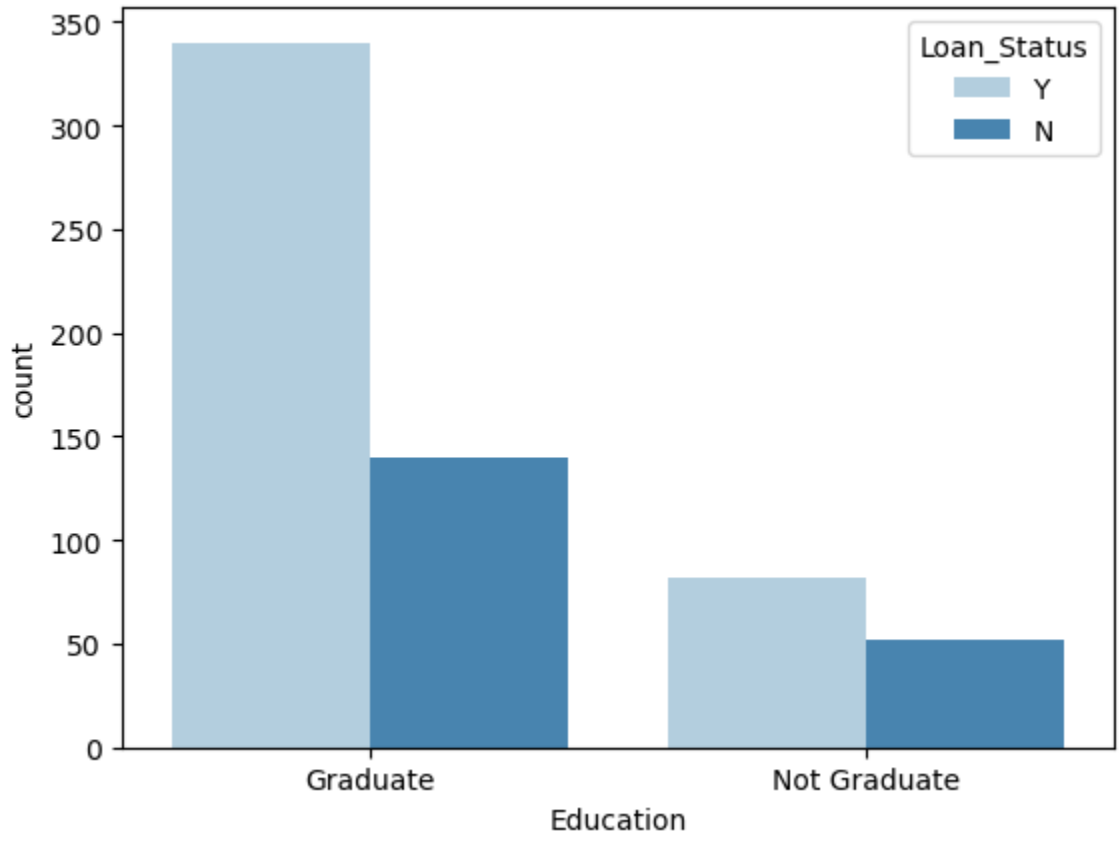


In [14]:
```
sns.countplot(x ='Education', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking Educational Status of loan Applicants

loan['Education'].value_counts()
```

Out[14]:
```
Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64
```

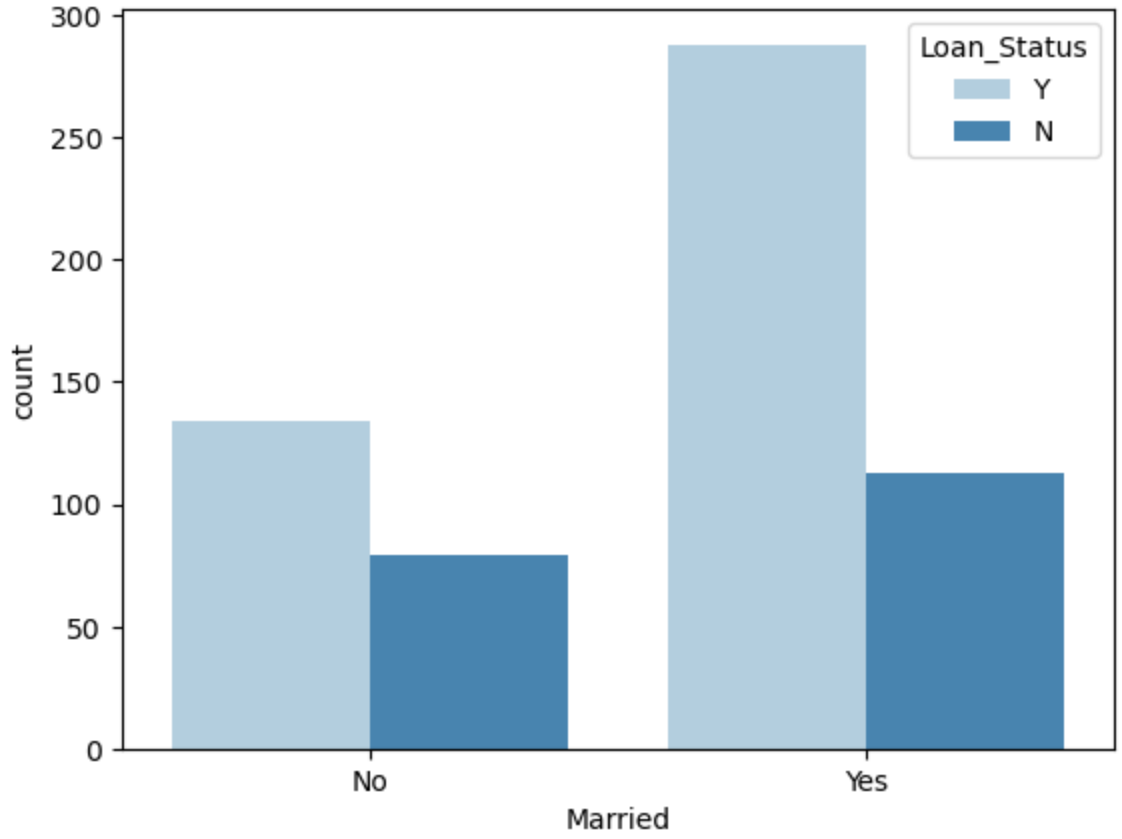


In [15]:
```
sns.countplot(x ='Married', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking the  Marrital Status of loan Applicants

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

Out[15]:
```
Married
Yes    401
No     213
Name: count, dtype: int64
```



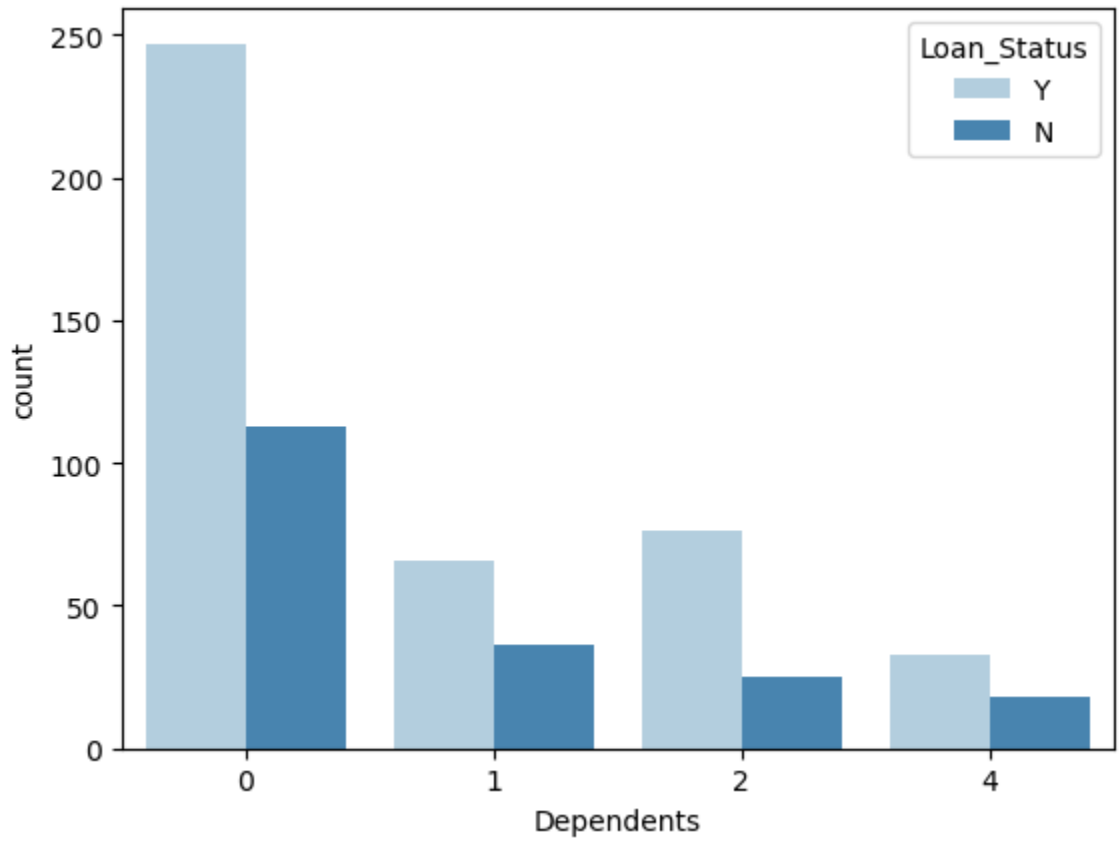
In [16]:
```
#Replacing 3+ in the 'Dependant' column with 4 so our model can read
loan.replace('3+', 4, inplace = True)
```

In [17]:
```
sns.countplot(x ='Dependents', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking the count of Dependents in the loan Applicants

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

Out[17]:
```
Dependents
0    360
1    102
2    101
4     51
Name: count, dtype: int64
```

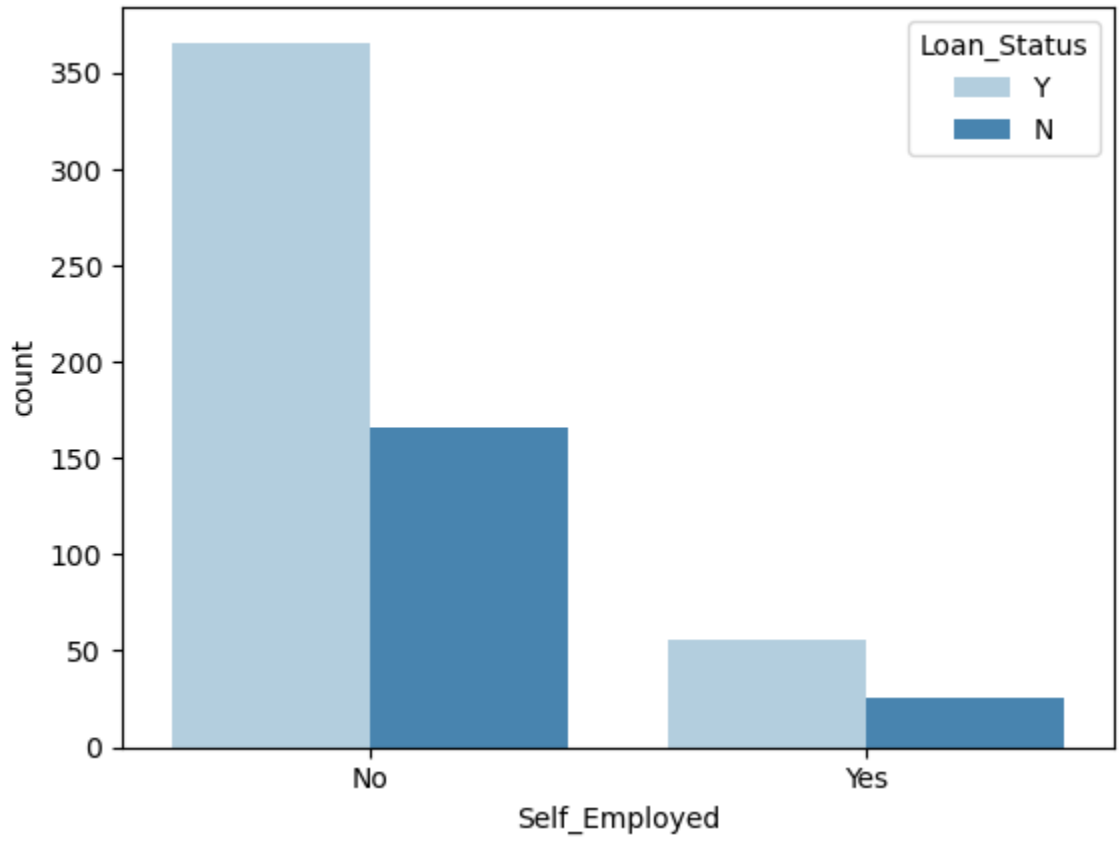


In [18]:
```
sns.countplot(x ='Self_Employed', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking the Employment Staus of loan Applicants

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

Out[18]:
```
Self_Employed
No     532
Yes     82
Name: count, dtype: int64
```

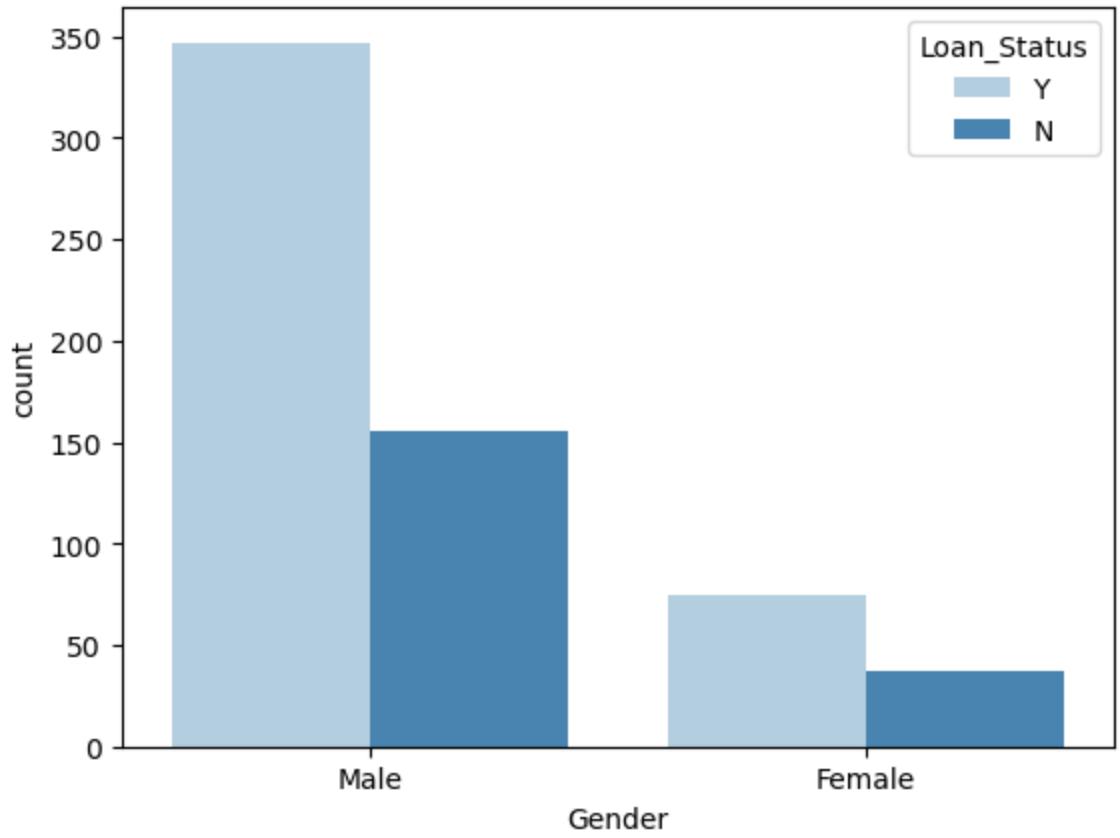


In [19]:
```
sns.countplot(x ='Gender', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking the Gender Status of loan Applicants

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

Out[19]:
```
Gender
Male      502
Female    112
Name: count, dtype: int64
```

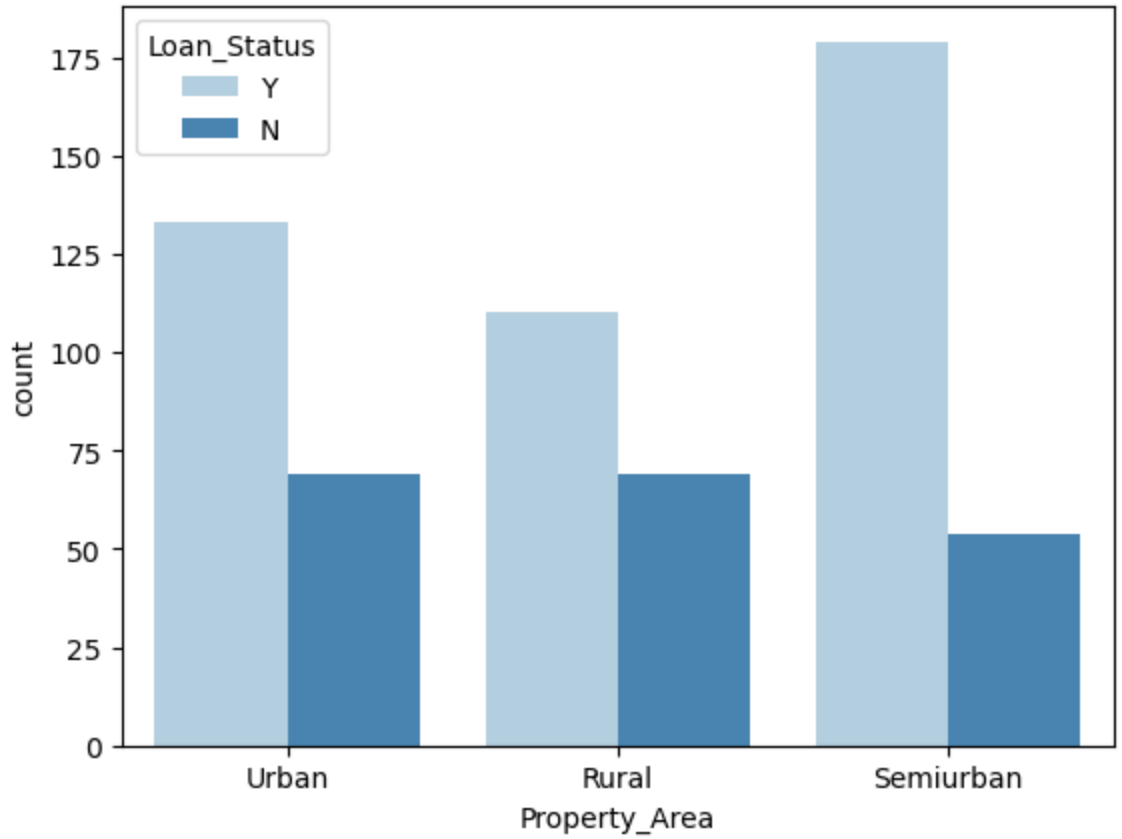


In [20]:
```
sns.countplot(x ='Property_Area', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking the classification of regions

loan['Property_Area'].value_counts()
```

Out[20]:
```
Property_Area
Semiurban    233
Urban        202
Rural        179
Name: count, dtype: int64
```



In [21]:
```
#Replacing 3+ in the 'Dependant' column with 4 so our model can read
#data.replace('3+', 4, inplace = True)
```

In [22]:
```
#Renaming the 'target' columns to Loan_Statusabs

#data.rename(columns= {'target': 'Loan_Status'}, inplace = True)
```

In [23]:
```
loan_status_counts = loan['Loan_Status'].value_counts()
```

In [24]:
```
#Percentage of each loan status
loan_status_percentage = (loan_status_counts / loan_status_counts.sum()) * 100
# Print the results
print(loan_status_percentage)
```

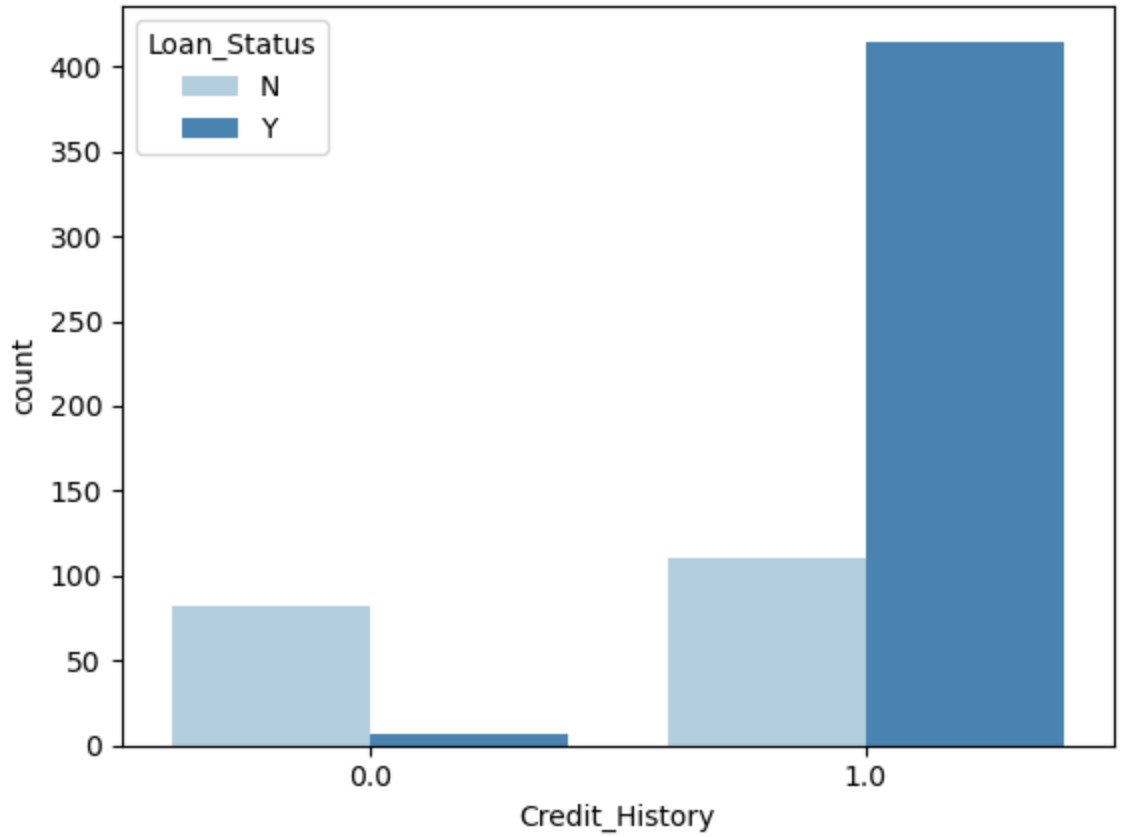
```
Loan_Status
Y    68.729642
N    31.270358
Name: count, dtype: float64
```

In [25]:
```
sns.countplot(x ='Credit_History', hue = 'Loan_Status', data = loan, palette ='Blues');

#Checking the count of Educated loan Applicants and non-educated

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

Out[25]:
```
Credit_History
1.0    525
0.0     89
Name: count, dtype: int64
```



In [26]:
```
#statistical measures
loan.describe()
```

Out[26]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	614.000000	614.000000	614.000000
mean	5403.459283	1621.245798	145.465798	342.410423	0.855049
std	6109.041673	2926.248369	84.180967	64.428629	0.352339
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.250000	360.000000	1.000000
50%	3812.500000	1188.500000	125.000000	360.000000	1.000000
75%	5795.000000	2297.250000	164.750000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

In [27]:
```
loan.columns
```

Out[27]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status'],
      dtype='object')
```

In [28]:
```
columns = ['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Status']
```

In [29]:
```
# Rename multiple columns and rename the new data
loan2 = loan.rename(columns={'ApplicantIncome': 'Applicant _Income', 'LoanAmount': 'Loan
loan2.head()
```

Out[29]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applicant _Income	Coapplicant_Income	Loan_A

0	LP001002	Male	No	0	Graduate	No	5849	0.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0

In [30]:
```
#data.replace({'Loan_Status':{'N': 0, 'Y': 1}}, inplace = True)
```

In [31]:
```
#x = data.drop(columns = 'Loan_Status')
#y = data['Loan_Status']
```

In [32]:
```
#loading the label encoder function, and it labels values between 0 and 1
label_encoder = LabelEncoder()
```

In [33]:
```
#transforming values of the data  to 0 nad 1
labels = label_encoder.fit_transform(loan2.Loan_Status)
```

0 = NO 1 = YES

In [35]:
```
#Appending the labels to the data frame. It will create a new column call target for the
loan2['target'] = labels
```

In [36]:
```
loan2.head()
```

Out[36]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_A
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	

In [37]:
```
#Droping the 'Loan_Status' since we now have a new column 'target' to replace the Loan_S
loan2.drop(columns = 'Loan_Status', inplace = True)
```

In [38]:
```
loan2.head()
```

Out[38]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_A
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	

In [39]:
```
#Renaming the 'target' columns to Loan_Statusabs

loan2.rename(columns= {'target': 'Loan_Status'}, inplace = True)
```

In [40]:
```
loan2
```

Out[40]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan
0	LP001002	Male	No	0	Graduate	No	5849	0.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	
...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	
610	LP002979	Male	Yes	4	Graduate	No	4106	0.0	
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	

614 rows × 13 columns

In [41]:
```
#Converting columns with text format to numerical values for the model to be able to int
loan2.replace({'Married':{'No':0, 'Yes': 1},
               'Education':{'Not Graduate':0, 'Graduate': 1},
               'Self_Employed':{'No':0, 'Yes': 1},
                'Property_Area':{'Rural':0, 'Semiurban': 1, 'Urban': '2'},
                'Gender':{'Female':0, 'Male': 1}}, inplace = True)
```

In [42]:
```
loan2.head()
```

Out[42]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Applicant_Income	Coapplicant_Income	Loan_A
0	LP001002	1	0	0	1	0	5849	0.0	
1	LP001003	1	1	1	1	0	4583	1508.0	
2	LP001005	1	1	0	1	1	3000	0.0	
3	LP001006	1	1	0	0	0	2583	2358.0	
4	LP001008	1	0	0	1	0	6000	0.0	

In [43]:
```
#Seperating the Features from Target

x = loan2.drop(columns = 'Loan_Status', axis = 1)
y = loan2['Loan_Status']
```

In [44]:
```
print(x)
```

```
      Loan_ID  Gender  Married Dependents  Education  Self_Employed  \
0    LP001002       1        0          0          1              0
1    LP001003       1        1          1          1              0
2    LP001005       1        1          0          1              1
3    LP001006       1        1          0          0              0
4    LP001008       1        0          0          1              0
..        ...     ...      ...        ...        ...            ...
609  LP002978       0        0          0          1              0
610  LP002979       1        1          4          1              0
```

```
611  LP002983       1        1          1          1               0
612  LP002984       1        1          2          1               0
613  LP002990       0        0          0          1               1

     Applicant _Income  Coapplicant_Income  Loan_Amount  Loan_Amount_Term  \
0                 5849                 0.0        120.0             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
..                 ...                 ...          ...               ...
609               2900                 0.0         71.0             360.0
610               4106                 0.0         40.0             180.0
611               8072               240.0        253.0             360.0
612               7583                 0.0        187.0             360.0
613               4583                 0.0        133.0             360.0

     Credit_History Property_Area
0               1.0             2
1               1.0             0
2               1.0             2
3               1.0             2
4               1.0             2
..              ...           ...
609             1.0             0
610             1.0             0
611             1.0             2
612             1.0             2
613             0.0             1

[614 rows x 12 columns]
```

In [45]:
```
print(y)
```

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int32
```

In [46]:
```
#Dropping the 'Loan_ID' column
loan2.drop(columns = 'Loan_ID', inplace = True)
```

In [47]:
```
loan2
```

Out[47]:

	Gender	Married	Dependents	Education	Self_Employed	Applicant _Income	Coapplicant_Income	Loan_Amount	I
0	1	0	0	1	0	5849	0.0	120.0	
1	1	1	1	1	0	4583	1508.0	128.0	
2	1	1	0	1	1	3000	0.0	66.0	
3	1	1	0	0	0	2583	2358.0	120.0	
4	1	0	0	1	0	6000	0.0	141.0	
...	...	...	...	...	...	...	...	...	...
609	0	0	0	1	0	2900	0.0	71.0	

610	1	1	4	1	0	4106	0.0	40.0
611	1	1	1	1	0	8072	240.0	253.0
612	1	1	2	1	0	7583	0.0	187.0
613	0	0	0	1	1	4583	0.0	133.0

614 rows × 12 columns

In [48]:
```
loan2.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    object 
 3   Education           614 non-null    int64  
 4   Self_Employed       614 non-null    int64  
 5   Applicant _Income   614 non-null    int64  
 6   Coapplicant_Income  614 non-null    float64
 7   Loan_Amount         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    object 
 11  Loan_Status         614 non-null    int32  
dtypes: float64(4), int32(1), int64(5), object(2)
memory usage: 55.3+ KB
```

In [49]:
```
loan2.shape
```

Out[49]:
```
(614, 12)
```

In [50]:
```
#Converting 'Dependents' column to integer
#loan2['Dependents'] = loan2['Dependents'].astype(int)

# Convert 'Property_Area' column to integer
#loan2['Property_Area'] = loan2['Property_Area'].astype(int)
```

In [51]:
```
#loan2.info()
```

In [52]:
```
#Seperating the Features from the  Target

x = loan2.drop(columns = 'Loan_Status', axis = 1)
y = loan2['Loan_Status']
```

In [53]:
```
print(x)
```

```
     Gender  Married Dependents  Education  Self_Employed  Applicant _Income  \
0         1        0          0          1              0               5849   
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   
..      ...      ...        ...        ...            ...                ...   
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   
```

```
     Coapplicant_Income  Loan_Amount  Loan_Amount_Term  Credit_History  \
0                   0.0        120.0             360.0             1.0
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
..                  ...          ...               ...             ...
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
0               2
1               0
2               2
3               2
4               2
..            ...
609             0
610             0
611             2
612             2
613             1

[614 rows x 11 columns]
```

In [54]:
```
print(y)
```

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int32
```

In [55]:
```
#Data standardization; to convert the values to a common range between 0 & 1 for our ML
scaler = StandardScaler()
```

In [56]:
```
#Fitting and transforming the 'x' data into the scaler to be in a common range
standardized_data = scaler.fit_transform(x)
```

In [57]:
```
#loan2['Dependents'] = loan['Dependents'].astype(int)
```

In [58]:
```
#Printing the 'standardized_data' data
print(standardized_data)
```

```
[[ 0.47234264 -1.37208932 -0.6827291  ...  0.2732313   0.41173269
   1.22329839]
 [ 0.47234264  0.72881553  0.14245922 ...  0.2732313   0.41173269
  -1.31851281]
 [ 0.47234264  0.72881553 -0.6827291  ...  0.2732313   0.41173269
   1.22329839]
 ...
 [ 0.47234264  0.72881553  0.14245922 ...  0.2732313   0.41173269
   1.22329839]
 [ 0.47234264  0.72881553  0.96764754 ...  0.2732313   0.41173269
   1.22329839]
```

```
 [-2.11710719 -1.37208932 -0.6827291  ...  0.2732313  -2.42876026
  -0.04760721]]
```

In [59]:
```
x = standardized_data
```

In [60]:
```
print(x)
```

```
[[ 0.47234264 -1.37208932 -0.6827291  ...  0.2732313   0.41173269
   1.22329839]
 [ 0.47234264  0.72881553  0.14245922 ...  0.2732313   0.41173269
  -1.31851281]
 [ 0.47234264  0.72881553 -0.6827291  ...  0.2732313   0.41173269
   1.22329839]
 ...
 [ 0.47234264  0.72881553  0.14245922 ...  0.2732313   0.41173269
   1.22329839]
 [ 0.47234264  0.72881553  0.96764754 ...  0.2732313   0.41173269
   1.22329839]
 [-2.11710719 -1.37208932 -0.6827291  ...  0.2732313  -2.42876026
  -0.04760721]]
```

In [61]:
```
print(y)
```

```
0      1
1      0
2      1
3      1
4      1
      ..
609    1
610    1
611    1
612    1
613    0
Name: Loan_Status, Length: 614, dtype: int32
```

In [62]:
```
loan2.to_csv('C:/Users/DONATUS/Desktop/DataScience/MLEngr/focus/pythonML/MLprojects_part
```

Spliting the Data into Training 80% & Test 20%

In [64]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, stratify=y, r
```

In [65]:
```
loan2.shape, x_train.shape, x_test.shape
```

Out[65]:
```
((614, 12), (491, 11), (123, 11))
```

Model Training using Suport Vector Machine

In [67]:
```
classifier = svm.SVC(kernel = 'linear')
```

In [68]:
```
#Fitting our training data into the classifier
# x_training data and y_train is the label
classifier.fit(x_train, y_train)
```

Out[68]:

SVC

```
SVC(kernel='linear')
```

Model Evaluation

In [70]:
```
x_train_prediction = classifier.predict(x_train)
```

```
training_data_accuracy = accuracy_score(x_train_prediction, y_train)
```

In [71]:
```
#Accuracy score of the train data

print('Accuracy score of the training data is:', training_data_accuracy)
```

```
Accuracy score of the training data is: 0.8105906313645621
```

In [72]:
```
#Evaluating the model to know the accuracy the test data

x_test_prediction = classifier.predict(x_test)

test_data_accuracy = accuracy_score(x_test_prediction, y_test)
```

In [73]:
```
print('Accuracy score of the test data is:', test_data_accuracy)
```

```
Accuracy score of the test data is: 0.8048780487804879
```

In [74]:
```
#Making a predictive system
input_data = (1,0,0,1,0,5849,0.0,120.0,360.0,1.0,2)

#Changing the input data to numpy arrray since the processing is more efficient
convert_data_to_numpy = np.asarray(input_data)

#reshaping the array as we are predicting for one data point
#parameter for reshaping ==> '.reshape(1, -1)'
reshape_input_data = convert_data_to_numpy.reshape(1, -1)

#standardizing the input_data
std_data = scaler.transform(reshape_input_data)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if prediction==0:
    print("We're sorry, Your Loan has not Approved.")
else:
    print('Congratulations, Your Loan has been Approved.')
```

```
[[ 0.47234264 -1.37208932 -0.6827291   0.52836225 -0.39260074  0.07299082
  -0.55448733 -0.30275919  0.2732313   0.41173269  1.22329839]]
[1]
Congratulations, Your Loan has been Approved.
```

In [75]:
```
#Making a predictive system
input_data = (1,1,4,1,0,3036,2504.0,158.0,360.0,0.0,1)

#Changing the input data to numpy arrray since the processing is more efficient
convert_data_to_numpy = np.asarray(input_data)

#reshaping the array as we are predicting for one data point
#parameter for reshaping ==> '.reshape(1, -1)'
reshape_input_data = convert_data_to_numpy.reshape(1, -1)

#standardizing the input_data
std_data = scaler.transform(reshape_input_data)
print(std_data)

prediction = classifier.predict(std_data)
print(prediction)

if prediction==0:
    print("We're sorry, Your Loan has not Approved.")
```

```
else:
    print('Congratulations, Your Loan has been Approved.')
```

```
[[ 0.47234264  0.72881553  2.61802418  0.52836225 -0.39260074 -0.38784963
   0.30191352  0.14901731  0.2732313  -2.42876026 -0.04760721]]
[0]
We're sorry, Your Loan has not Approved.
```

In [ ]: