In [2]:
```python
import pandas as pd
train = pd.read_csv("train.csv")
train.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coappl
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 [3]:
```python
train.shape
```

Out[3]: (614, 13)

In [4]:
```python
train.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
```

In [5]:
```
train.isna().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]:
```
train.describe()
```

Out[6]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

In [7]:
```
train.describe(include=[object])
```

Out[7]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Property_Area	Loa
count	614	601	611	599	614	582	614	
unique	614	2	2	4	2	2	3	
top	LP001002	Male	Yes	0	Graduate	No	Semiurban	
freq	1	489	398	345	480	500	233	

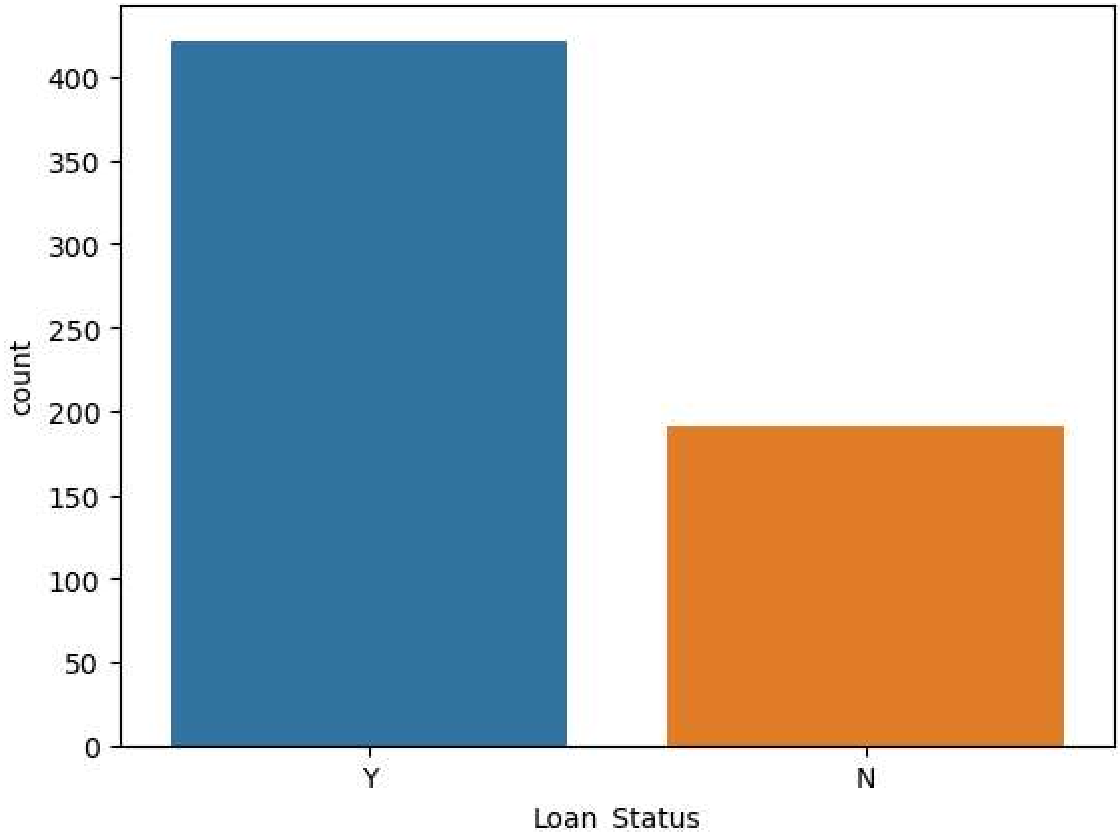
In [8]:
```
train = train.drop(['Loan_ID'], axis=1)
```

In [9]:
```
train['Loan_Status'].value_counts()
```

Out[9]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

In [10]:
```
import seaborn as sns
sns.countplot(x=train['Loan_Status'])
```

Out[10]: <Axes: xlabel='Loan_Status', ylabel='count'>

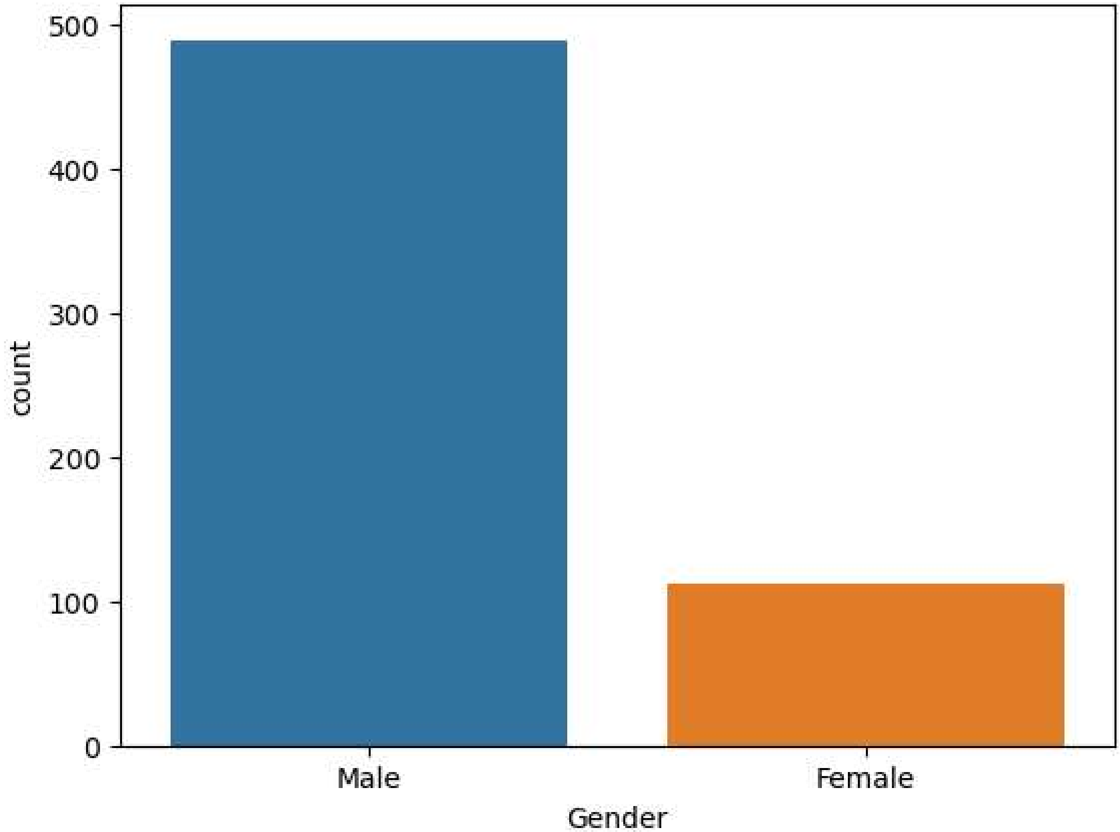


In [11]:
```
train['Gender'].value_counts()
```

Out[11]:
```
Male      489
Female    112
Name: Gender, dtype: int64
```

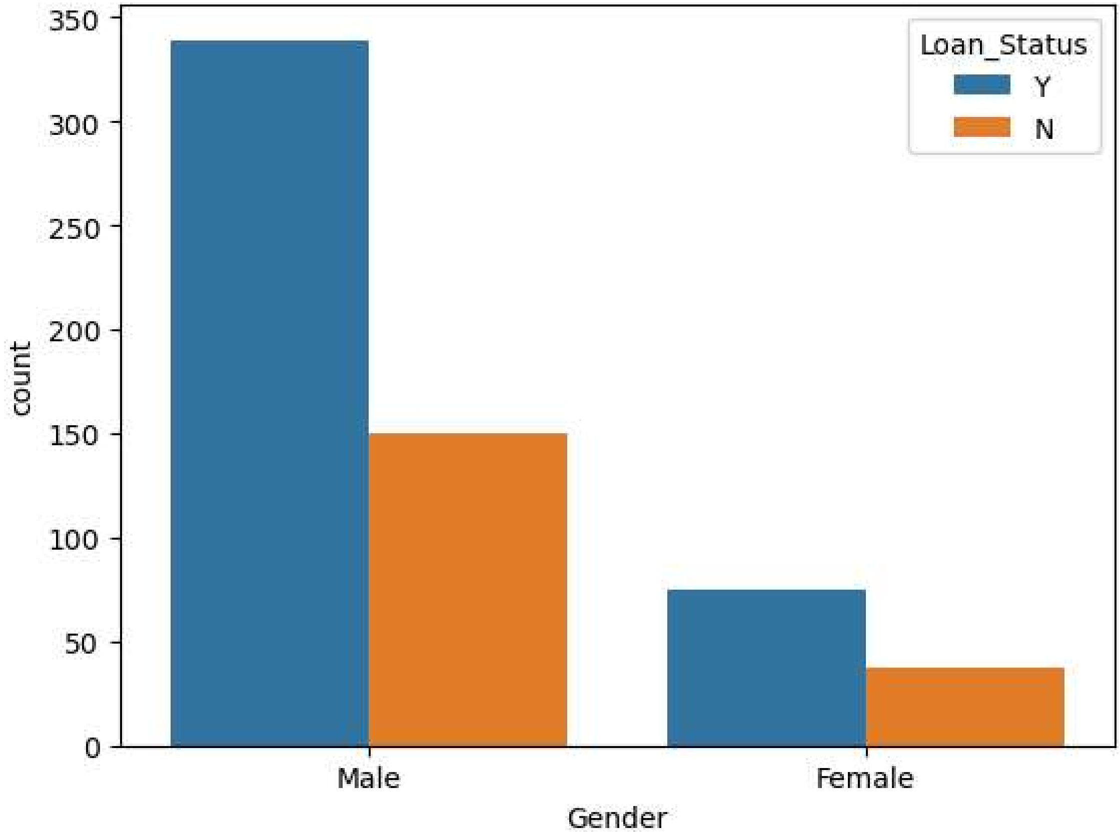
In [12]:
```
sns.countplot(x=train['Gender'])
```

Out[12]: <Axes: xlabel='Gender', ylabel='count'>



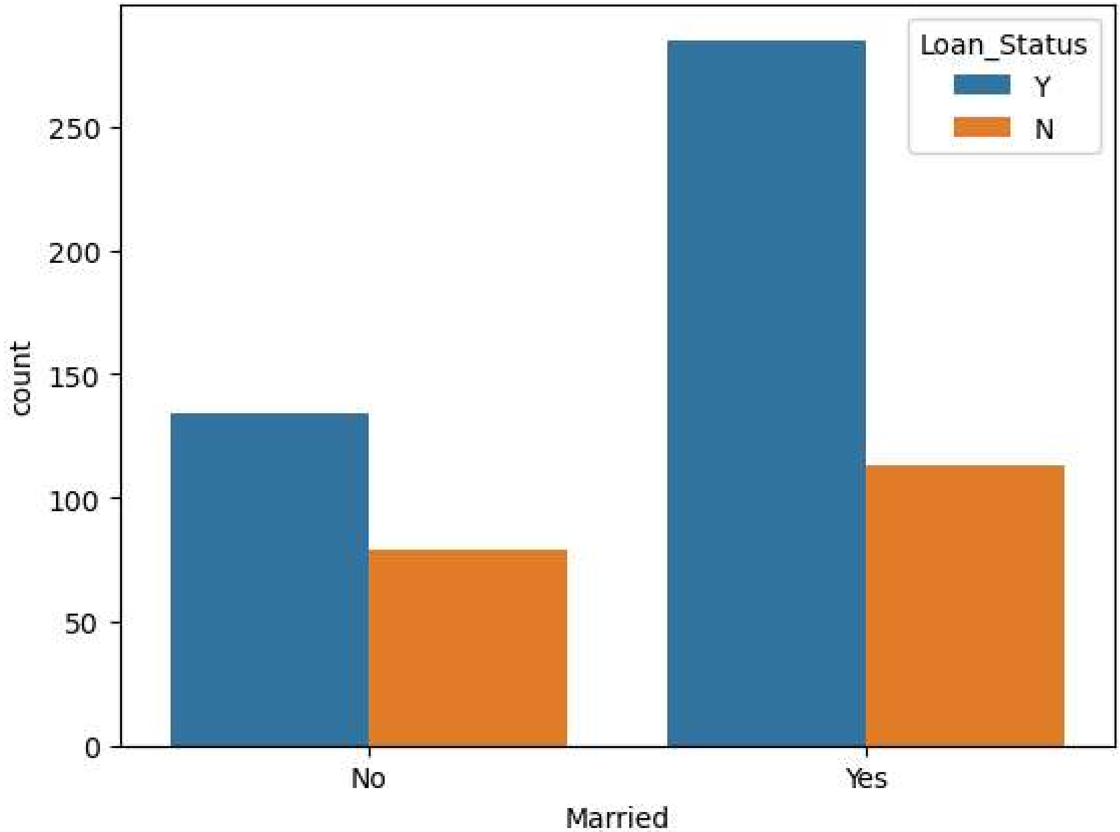
In [13]:
```
sns.countplot(x=train['Gender'], hue=train['Loan_Status'])
```

Out[13]:
```
<Axes: xlabel='Gender', ylabel='count'>
```



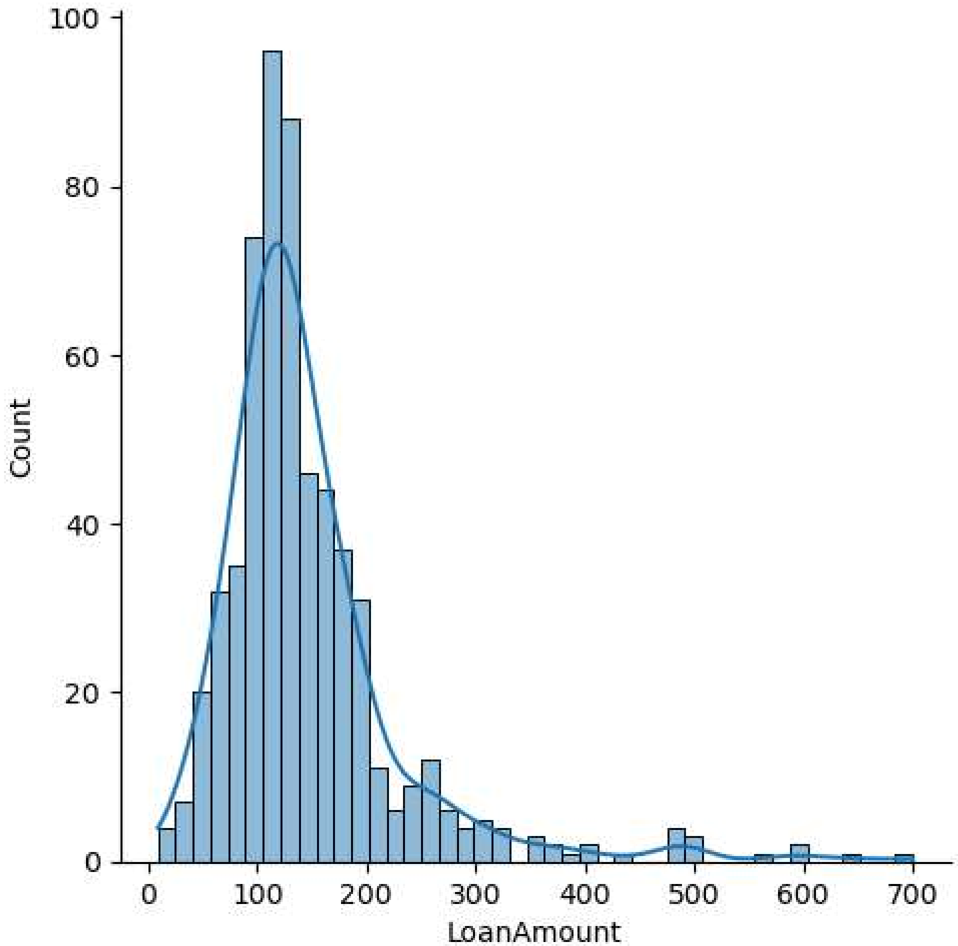
In [14]:
```
sns.countplot(x='Married', data=train, hue='Loan_Status')
```

Out[14]: <Axes: xlabel='Married', ylabel='count'>



In [15]:
```
sns.displot(train['LoanAmount'], kde=True)
```

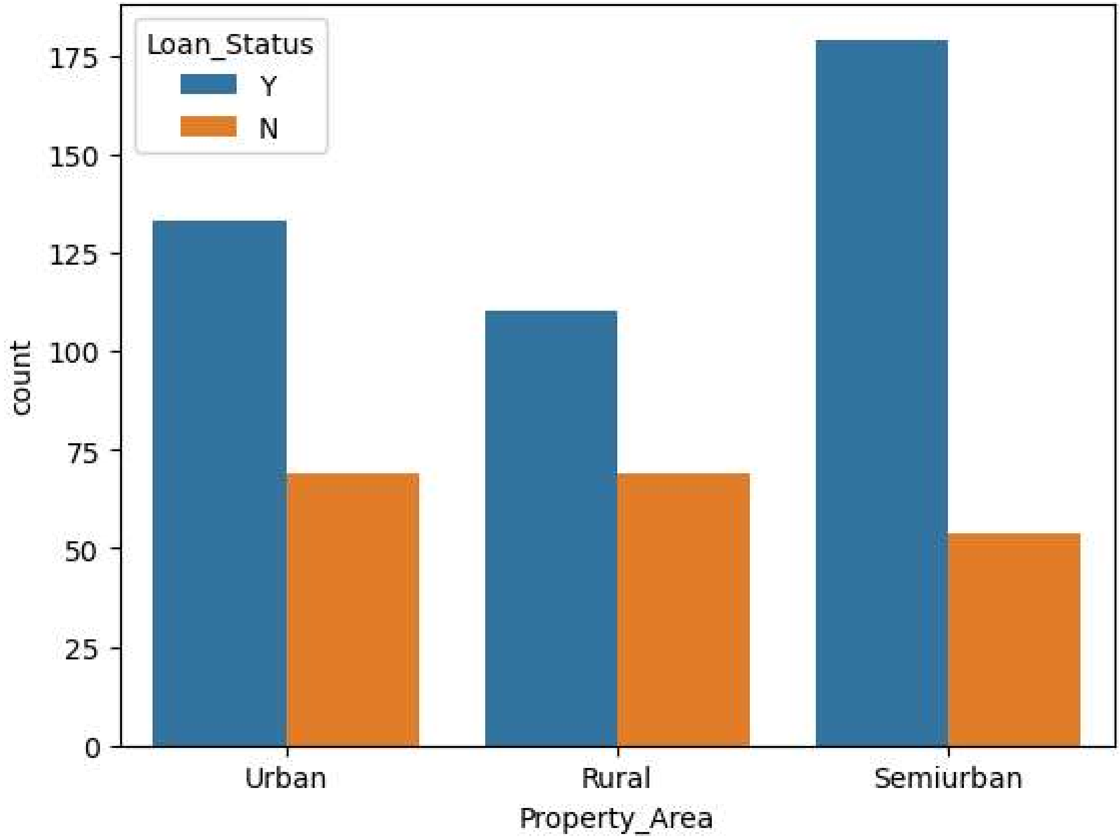
Out[15]: <seaborn.axisgrid.FacetGrid at 0x1a3b64dda50>



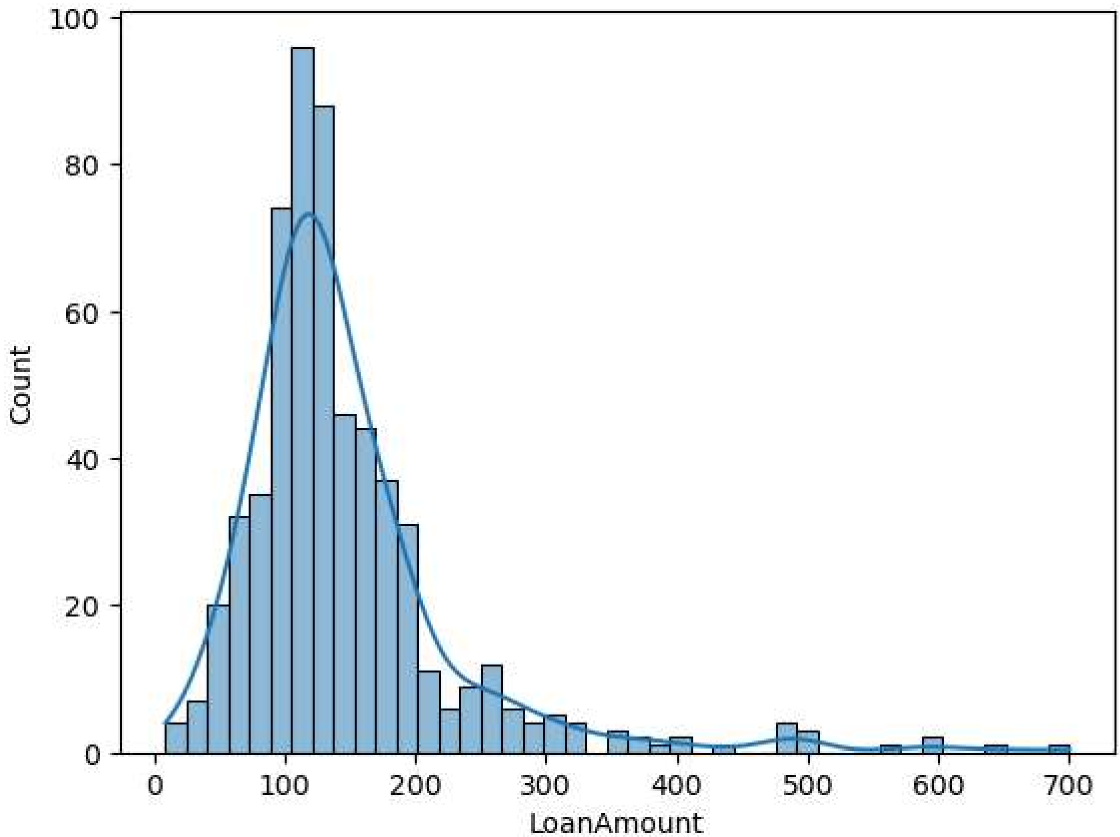
In [16]:
```
train['LoanAmount'].skew()
```

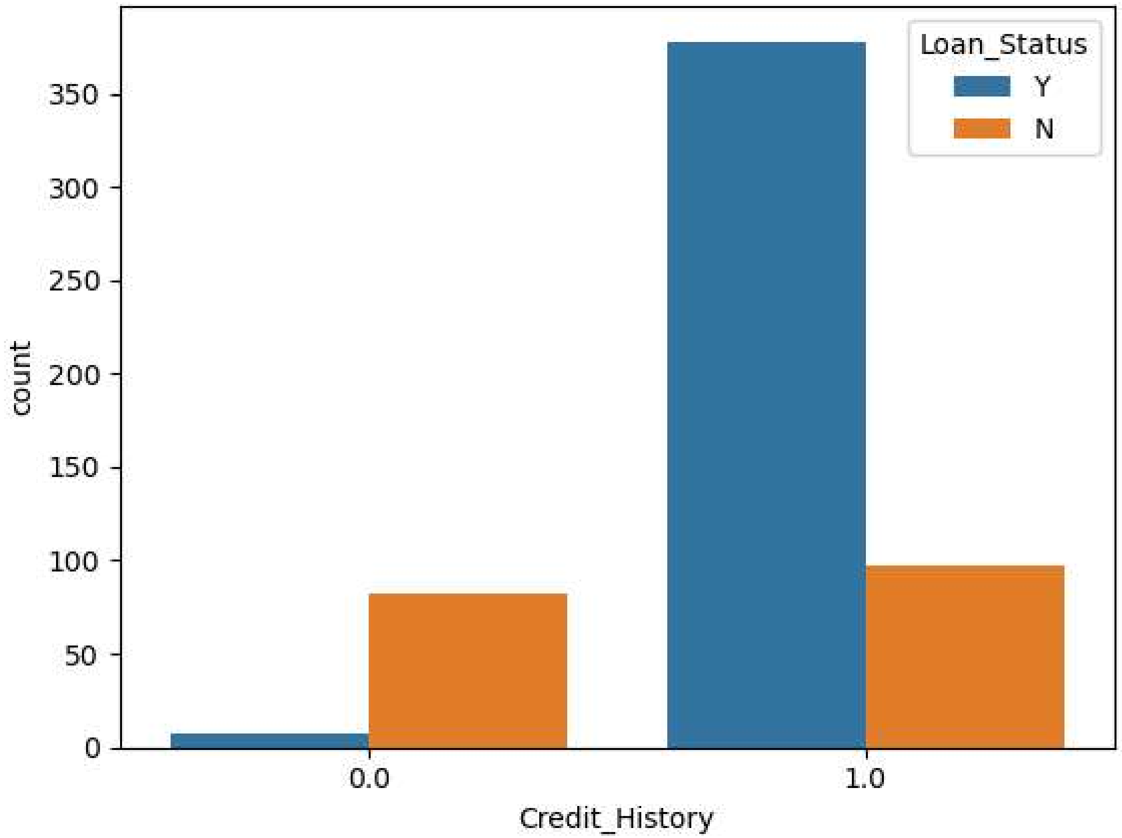
Out[16]: 2.677551679256059

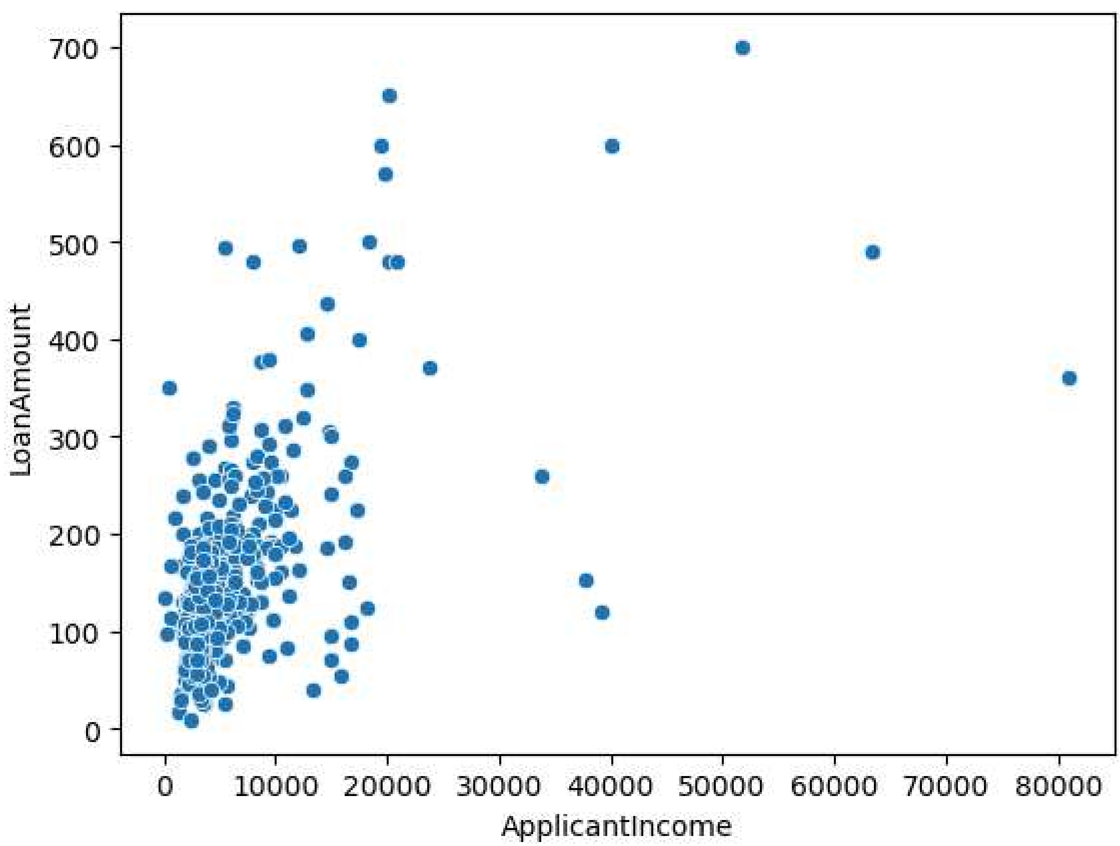
In [17]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.countplot(data=train, x='Property_Area', hue='Loan_Status')
plt.show()
```



In [18]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.histplot(train['LoanAmount'], kde=True)
plt.show()
sns.countplot(data=train, x='Credit_History', hue='Loan_Status')
plt.show()
sns.scatterplot(data=train, x='ApplicantIncome', y='LoanAmount')
plt.show()
```

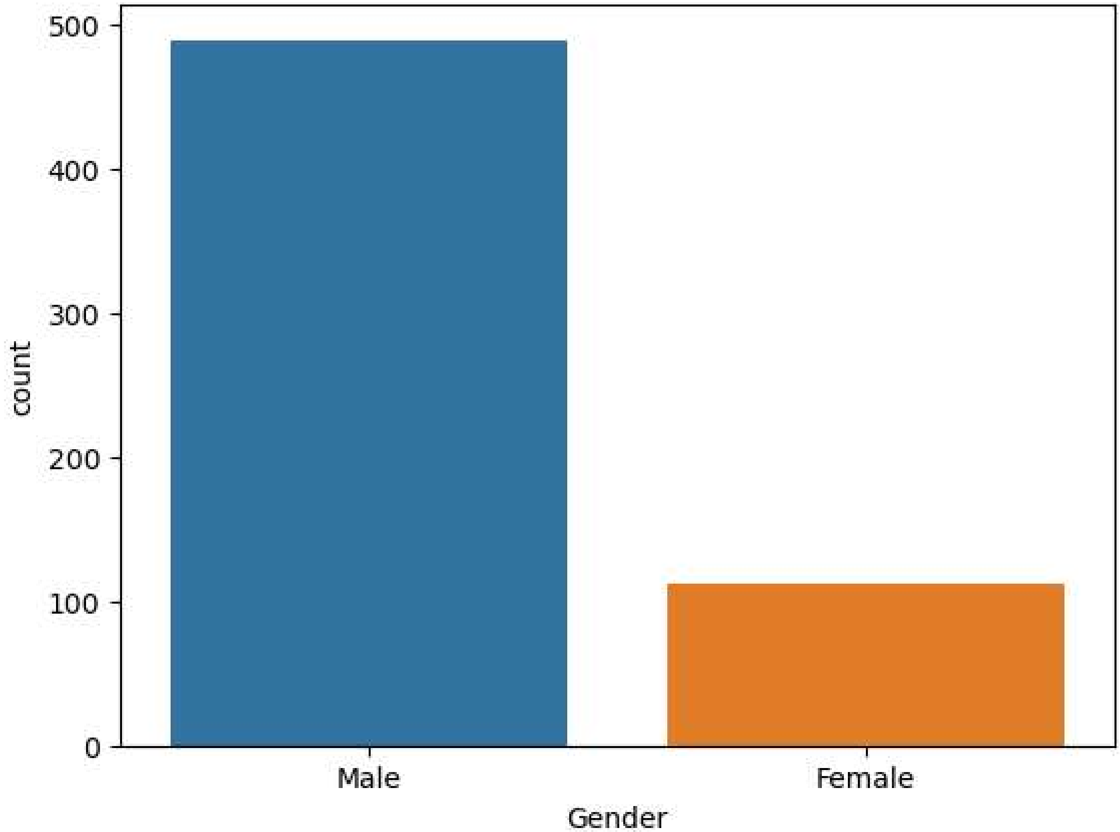



Loan_Status
Y
N
count
Credit_History
0.0
1.0
0
50
100
150
200
250
300
350


LoanAmount
ApplicantIncome
0
100
200
300
400
500
600
700
0
10000
20000
30000
40000
50000
60000
70000
80000

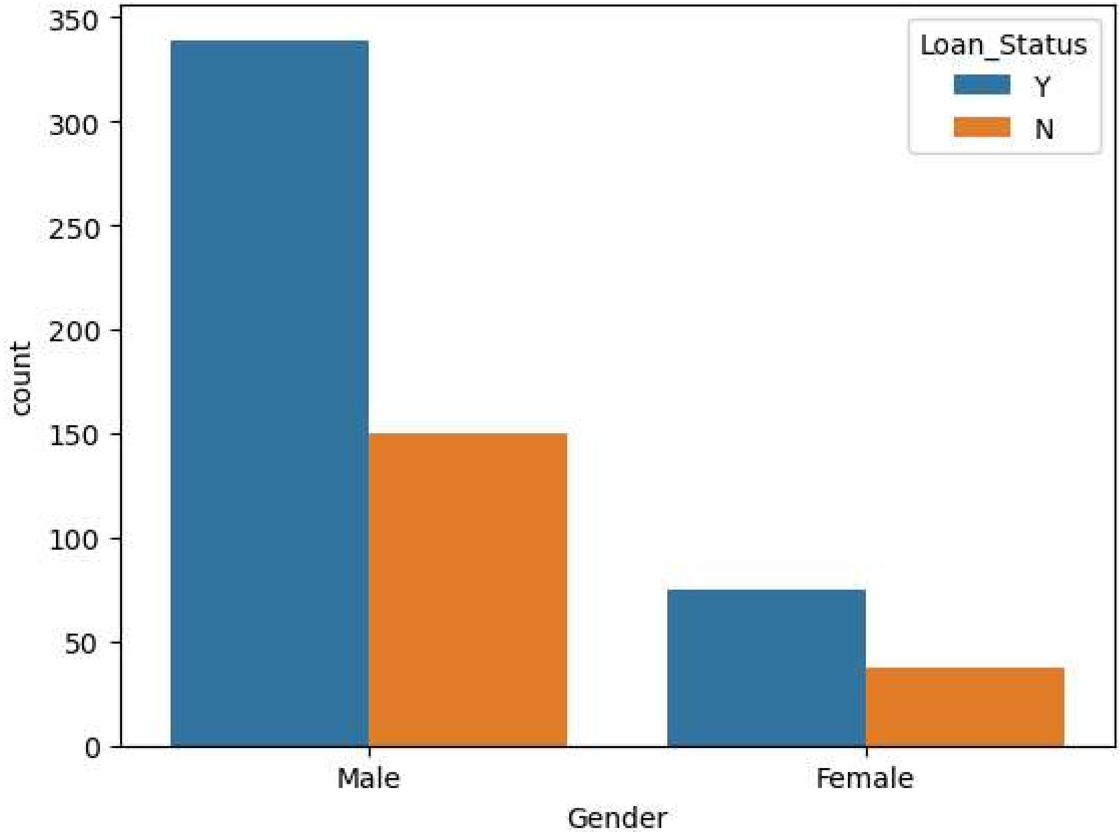
In [19]:
```
sns.countplot(x=train['Gender'])
```

Out[19]: <Axes: xlabel='Gender', ylabel='count'>



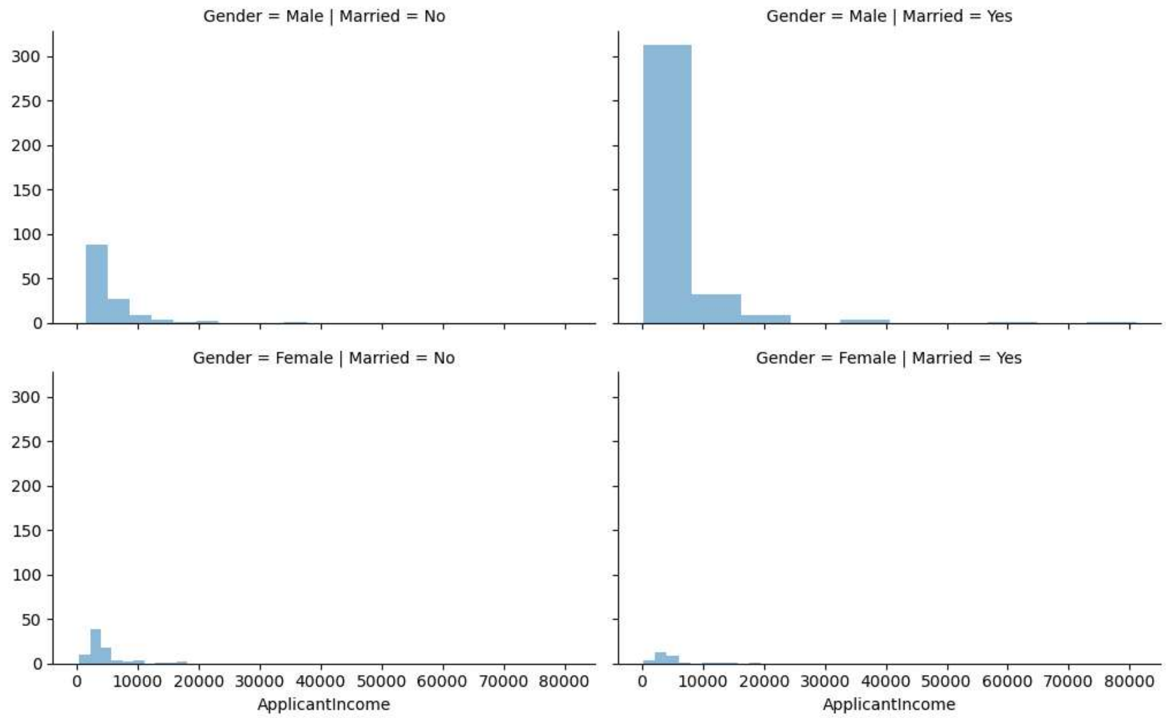
In [20]:
```
sns.countplot(x=train['Gender'],hue=train['Loan_Status'])
```

Out[20]: <Axes: xlabel='Gender', ylabel='count'>



In [21]:
```
import matplotlib.pyplot as plt
grid = sns.FacetGrid(train,row="Gender",col="Married",height=3.2,aspect=1.6)
grid.map(plt.hist,"ApplicantIncome",alpha=.5,bins=10)
grid.add_legend()
```

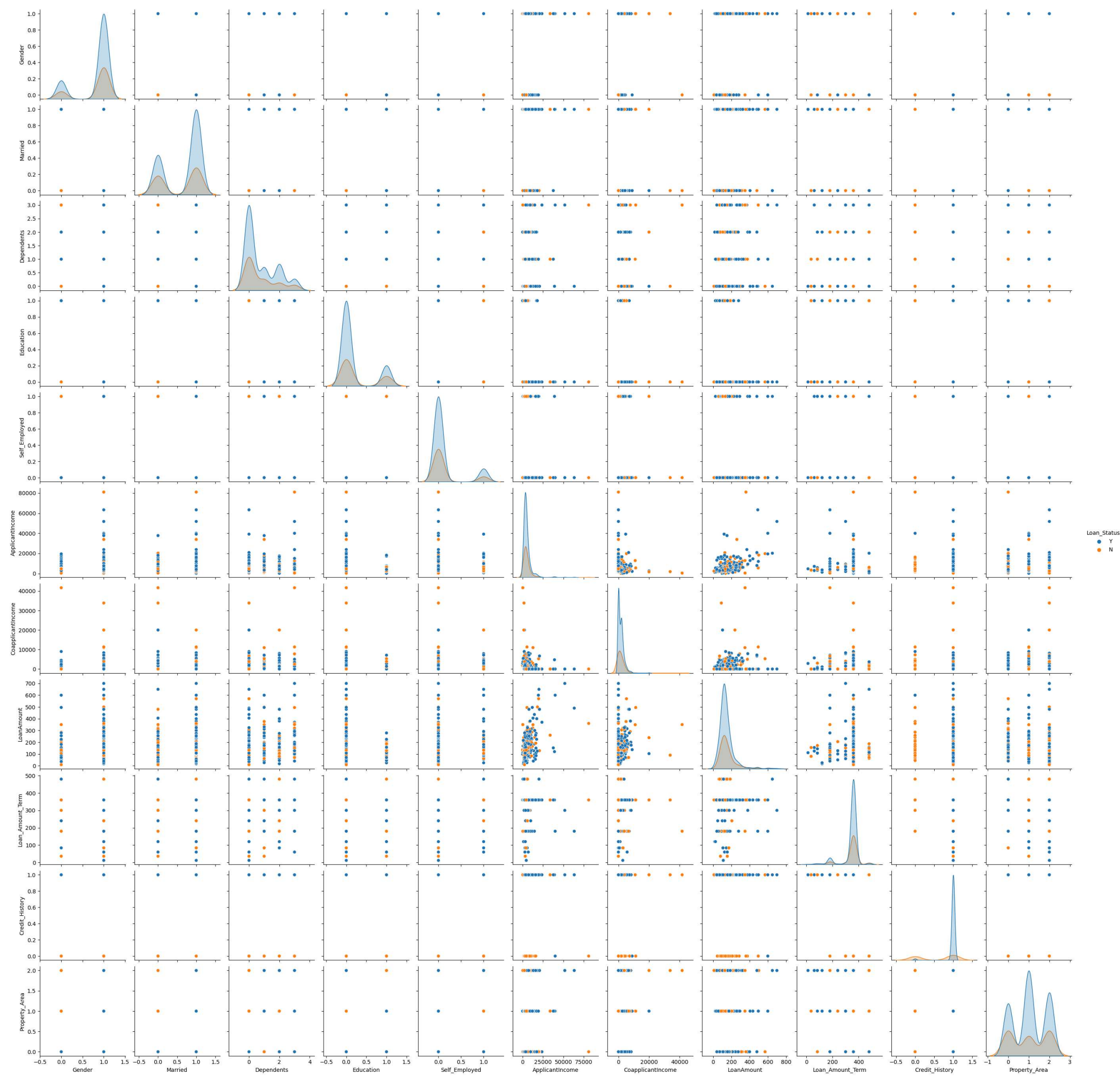
Out[21]: <seaborn.axisgrid.FacetGrid at 0x1a3b910add0>



In [22]:
```
# Filling missing values
train['Gender'] = train['Gender'].fillna(train['Gender'].mode()[0])
train['Married'] = train['Married'].fillna(train['Married'].mode()[0])
train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
train['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=
train['Credit_History'].fillna(train['Credit_History'].median(), inplace=True)

# Encoding categorical features
from sklearn.preprocessing import LabelEncoder
feature_col = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed'
label_encoder = LabelEncoder()
for col in feature_col:
    train[col] = label_encoder.fit_transform(train[col])
```

In [23]:
```
sns.pairplot(train,hue="Loan_Status",height=2.5)
```

Out[23]: <seaborn.axisgrid.PairGrid at 0x1a3b9223310>



In [24]:
```
train.isna().sum()
```

Out[24]:
```
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 [25]:
```
train['Gender'] = train['Gender'].fillna(train['Gender'].mode()[0])
train['Married'] = train['Married'].fillna(train['Married'].mode()[0])
train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
train['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=
train['Credit_History'].fillna(train['Credit_History'].median(), inplace=True)

from sklearn.preprocessing import LabelEncoder
feature_col = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed'
label_encoder = LabelEncoder()
for col in feature_col:
    train[col] = label_encoder.fit_transform(train[col])
train.isna().sum()
```

Out[25]:
```
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 [30]:
```
from sklearn.preprocessing import LabelEncoder
feature_col = ['Gender','Married','Dependents','Education', 'Self_Employed','P
le = LabelEncoder()
for col in feature_col:
    train[col] = le.fit_transform(train[col])
```

In [33]:
```
train.Loan_Status = train.Loan_Status.replace({"Y": 1, "N" : 0})
```

In [34]:
```
train.head(3)
```

Out[34]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome
0	1	0	0	0	0	5849	0.0
1	1	1	1	0	0	4583	1508.0
2	1	1	0	0	1	3000	0.0

In [36]:
```
train['total_income'] = train['ApplicantIncome'] + train['CoapplicantIncome']
```

In [37]:
```
train.drop(columns = ['ApplicantIncome', 'CoapplicantIncome'], inplace=True)
```

In [38]:
```
train.head(3)
```

Out[38]:

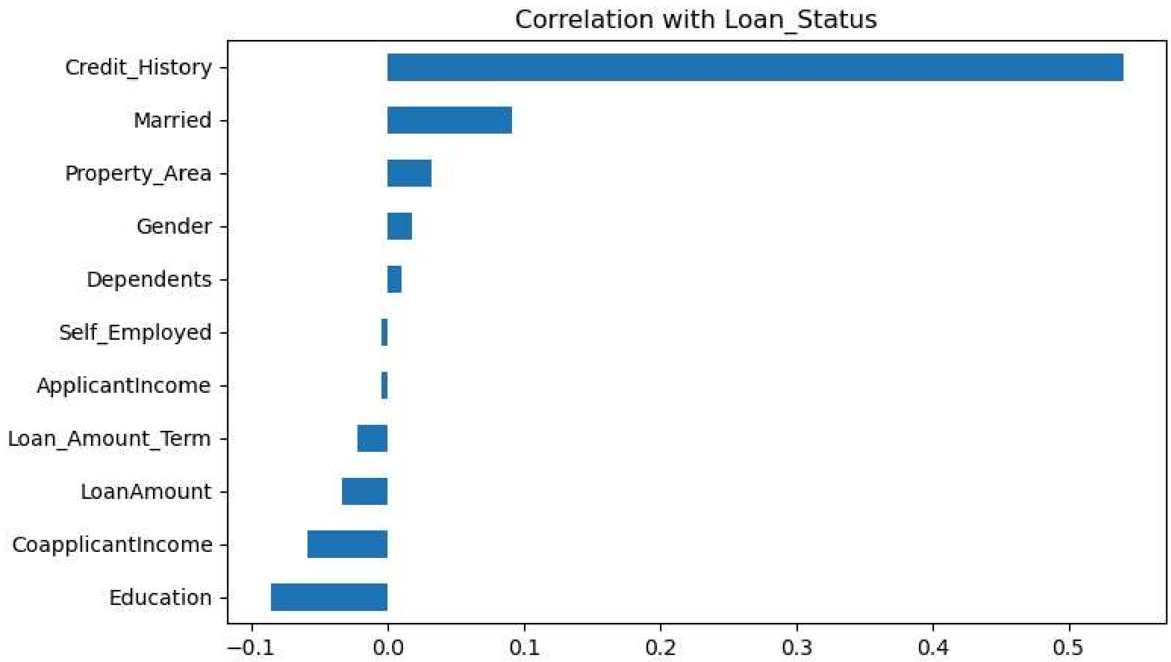
	Gender	Married	Dependents	Education	Self_Employed	LoanAmount	Loan_Amount_Term
0	1	0	0	0	0	128.0	360.0
1	1	1	1	0	0	128.0	360.0
2	1	1	0	0	1	66.0	360.0

In [39]:
```
train.columns
```

Out[39]:
```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area',
       'Loan_Status', 'total_income'],
      dtype='object')
```

In [40]:
```
# Importing necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

# Load the dataset
train = pd.read_csv("train.csv")
train = train.drop(['Loan_ID'], axis=1)

# Fill missing values
train['Gender'] = train['Gender'].fillna(train['Gender'].mode()[0])
train['Married'] = train['Married'].fillna(train['Married'].mode()[0])
train['Dependents'] = train['Dependents'].fillna(train['Dependents'].mode()[0]
train['Self_Employed'] = train['Self_Employed'].fillna(train['Self_Employed'].
train['LoanAmount'] = train['LoanAmount'].fillna(train['LoanAmount'].median())
train['Loan_Amount_Term'] = train['Loan_Amount_Term'].fillna(train['Loan_Amoun
train['Credit_History'] = train['Credit_History'].fillna(train['Credit_History

# Encode categorical features
feature_col = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed'
label_encoder = LabelEncoder()
for col in feature_col:
    train[col] = label_encoder.fit_transform(train[col])

train['Loan_Status'] = train['Loan_Status'].map({'Y': 1, 'N': 0})
rel_feat = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
            'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area
            'ApplicantIncome', 'CoapplicantIncome', 'Loan_Status']

rel_feat_corr = train.corr(numeric_only=True)['Loan_Status'][rel_feat[:-1]]
print(rel_feat_corr)
plt.figure(figsize=(8, 5))
rel_feat_corr.sort_values().plot(kind='barh')
plt.title('Correlation with Loan_Status')
plt.show()
```

```
Gender               0.017987
Married              0.091478
Dependents           0.010118
Education           -0.085884
Self_Employed       -0.003700
LoanAmount          -0.033214
Loan_Amount_Term    -0.022549
Credit_History       0.540556
Property_Area        0.032112
ApplicantIncome     -0.004710
CoapplicantIncome   -0.059187
Name: Loan_Status, dtype: float64
```


Correlation with Loan_Status
Credit_History
Married
Property_Area
Gender
Dependents
Self_Employed
ApplicantIncome
Loan_Amount_Term
LoanAmount
CoapplicantIncome
Education
−0.1
0.0
0.1
0.2
0.3
0.4
0.5

In [41]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r

# Load the dataset
train = pd.read_csv("train.csv")

# Data preprocessing
# Drop the Loan_ID column (not required for modeling)
train = train.drop(['Loan_ID'], axis=1)

# Fill missing values
train['Gender'].fillna(train['Gender'].mode()[0], inplace=True)
train['Married'].fillna(train['Married'].mode()[0], inplace=True)
train['Dependents'].fillna(train['Dependents'].mode()[0], inplace=True)
train['Self_Employed'].fillna(train['Self_Employed'].mode()[0], inplace=True)
train['LoanAmount'].fillna(train['LoanAmount'].median(), inplace=True)
train['Loan_Amount_Term'].fillna(train['Loan_Amount_Term'].mode()[0], inplace=
train['Credit_History'].fillna(train['Credit_History'].median(), inplace=True)

# Encode categorical variables
from sklearn.preprocessing import LabelEncoder
categorical_cols = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Empl
le = LabelEncoder()
for col in categorical_cols:
    train[col] = le.fit_transform(train[col])

# Define features and target variable
X = train.drop(['Loan_Status'], axis=1)  # Features
y = train['Loan_Status']  # Target

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Train a Random Forest Classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)
print("Classification Report:")
print(class_report)
```

```
Accuracy: 0.7560975609756098
Confusion Matrix:
[[18 25]
 [ 5 75]]
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.42      0.55        43
           1       0.75      0.94      0.83        80

    accuracy                           0.76       123
   macro avg       0.77      0.68      0.69       123
weighted avg       0.76      0.76      0.73       123
```

In [ ]: