Loan Approval Predication

Problem Statement:

Loan providers face challenges in manually assessing loan applications, which can lead to delays and inconsistencies. The objective of this project is to develop a machine learning model that automates the loan approval process by analyzing historical data, thereby improving efficiency and accuracy.

Import libraries

In [4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

Import dataset

In [6]:
```
df = pd.read_csv("loan approval dataset.csv")
```

In [7]:
```
df.head()
```

Out[7]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_A
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	

Exploratery Data Analysis

In [9]:
```
df.shape
```

Out[9]: (614, 13)

In [10]:
```
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   601 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 [11]:
```
df.describe()
```

Out[11]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	601.000000	564.000000
mean	5403.459283	1621.245798	146.412162	284.732113	0.842199
std	6109.041673	2926.248369	85.587325	100.412199	0.364878
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	180.000000	1.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	168.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	600.000000	1.000000

Handling Null Values

In [13]:
```
df.isna().sum()
```

Out[13]:
```
Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          22
Loan_Amount_Term    13
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

In [14]:
```
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
```

In [15]:
```
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
```

In [16]:
```
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
```

In [17]:
```
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)
```

In [18]:
```
df['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)
```

In [19]:
```
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].median(), inplace=True)
```

In [20]:
```
df['Credit_History'].value_counts()
```

Out[20]:
```
Credit_History
1.0    475
0.0     89
Name: count, dtype: int64
```

In [21]:
```
df['Credit_History'].fillna(1.0, inplace=True)
```

In [22]:
```
df['Credit_History'] = df['Credit_History'].astype('int')
```

In [23]:
```
df['Credit_History'].value_counts()
```

Out[23]:
```
Credit_History
1    525
0     89
Name: count, dtype: int64
```

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

Out[24]:
```
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 [25]:
```
df['Dependents'].value_counts()
```

Out[25]:
```
Dependents
0     360
1     102
2     101
3+     51
Name: count, dtype: int64
```

In [26]:
```
df['Dependents'].replace('3+',3, inplace=True)
print(df['Dependents'].dtypes)
```

```
object
```

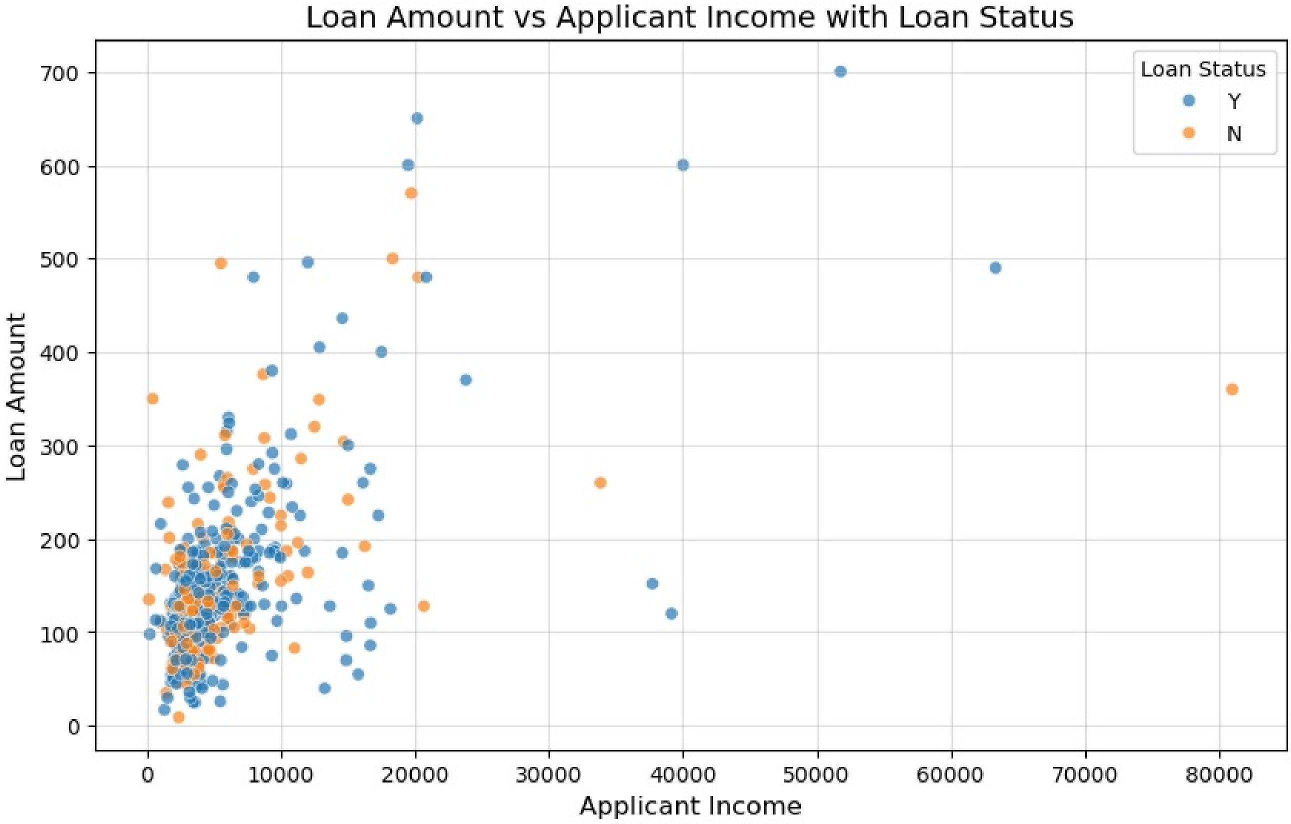
In [27]:
```
df['Dependents'] = df['Dependents'].astype(int)
print(df['Dependents'].dtypes)
```

```
int32
```

In [28]:
```
df.duplicated().sum()
```

Out[28]:
```
0
```

In [29]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df,x='ApplicantIncome', y='LoanAmount', hue='Loan_Status',alpha=0.7 )
plt.title('Loan Amount vs Applicant Income with Loan Status', fontsize=14)
plt.xlabel('Applicant Income', fontsize=12)
plt.ylabel('Loan Amount', fontsize=12)
plt.legend(title='Loan Status', fontsize=10)
plt.grid(alpha=0.4)
plt.show()
```



Numerical columns

In [31]:
```
numerical_columns = df.select_dtypes(exclude=object)
```

```
numerical_columns.head()
```

Out[31]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	0	5849	0.0	128.0	240.0	1
1	1	4583	1508.0	128.0	240.0	1
2	0	3000	0.0	66.0	240.0	1
3	0	2583	2358.0	120.0	240.0	1
4	0	6000	0.0	141.0	240.0	1

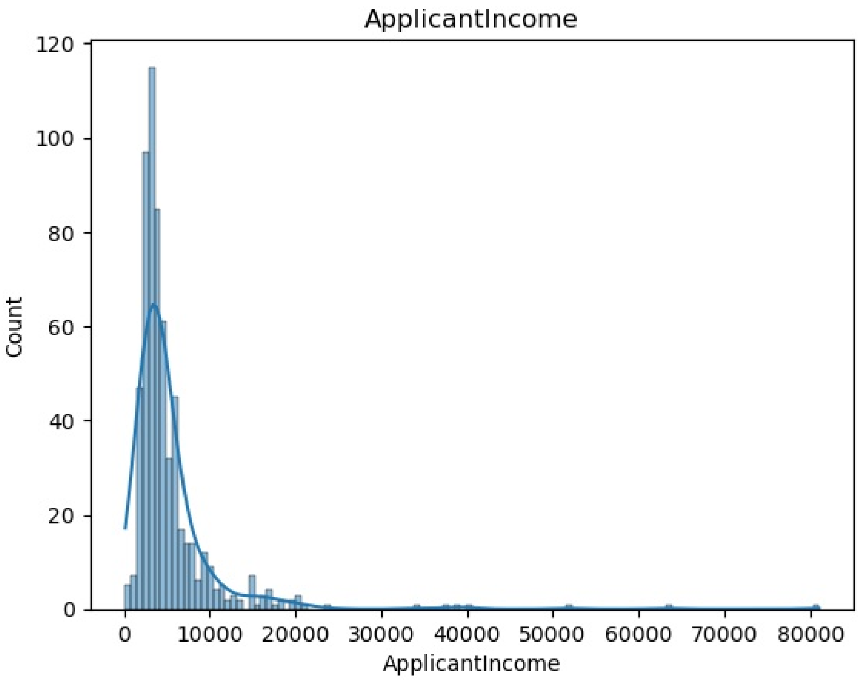
In [32]:

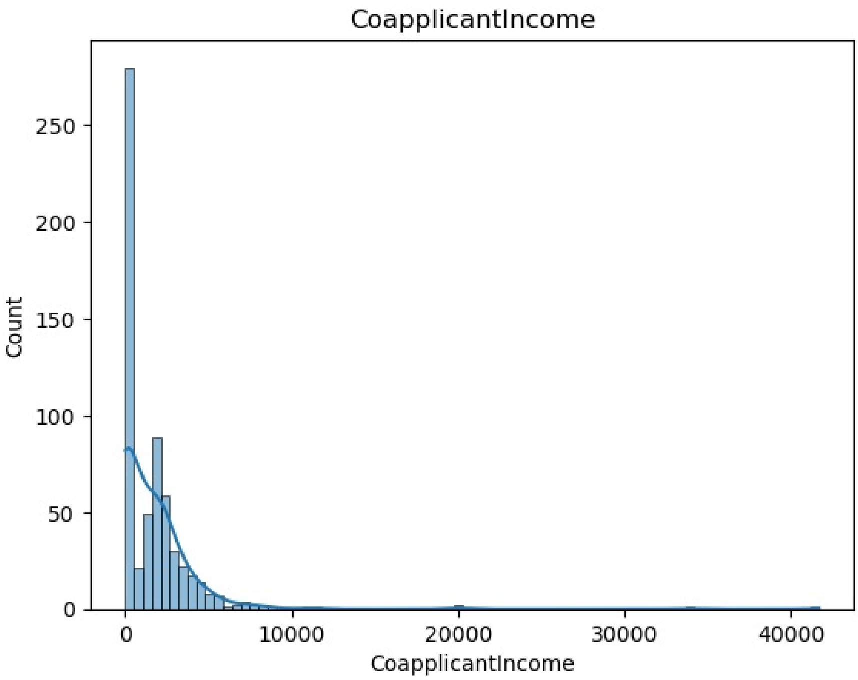
```
columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term']
```

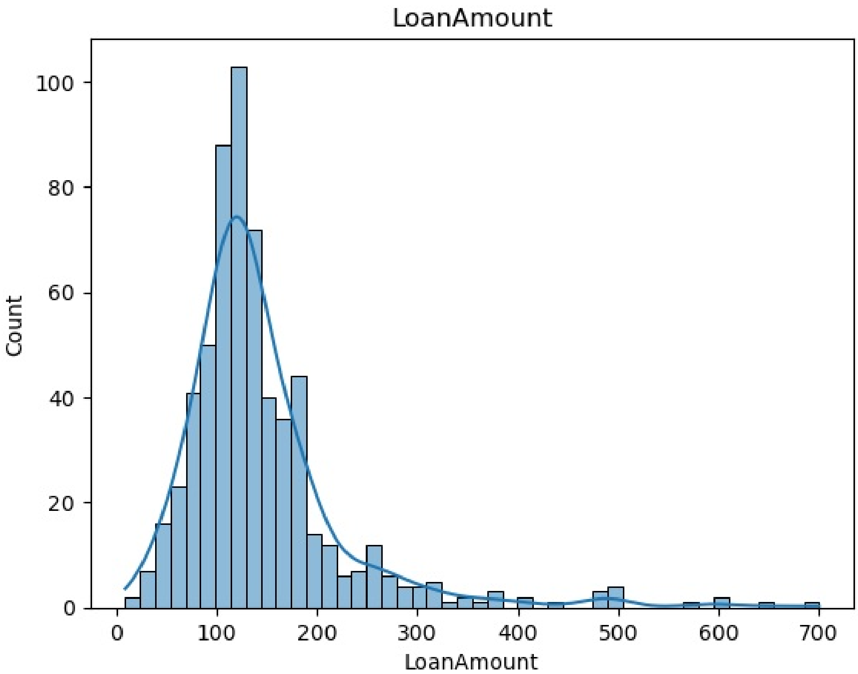
Check Skewness

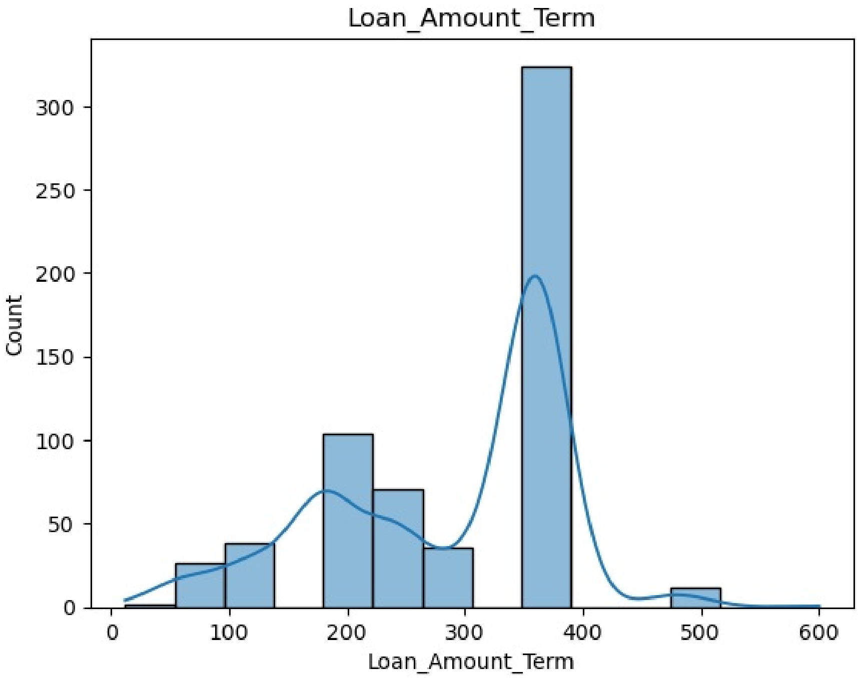
In [34]:

```
for i in columns:
    sns.histplot(numerical_columns[i], kde=True)
    plt.title(i)
    plt.show()
```









In [35]:
```
from scipy.stats import skew
```

In [36]:
```
for i in columns:
    print(f"{i}---->{skew(numerical_columns[i])}")
```

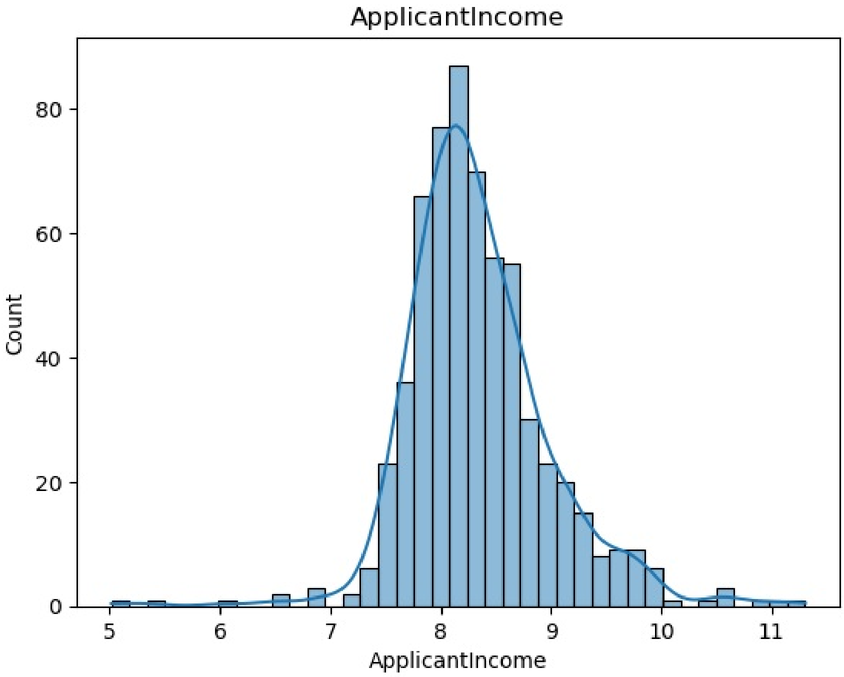
```
ApplicantIncome---->6.523526250899361
CoapplicantIncome---->7.473216996340462
LoanAmount---->2.736346927149759
Loan_Amount_Term---->-0.6277838196730566
```

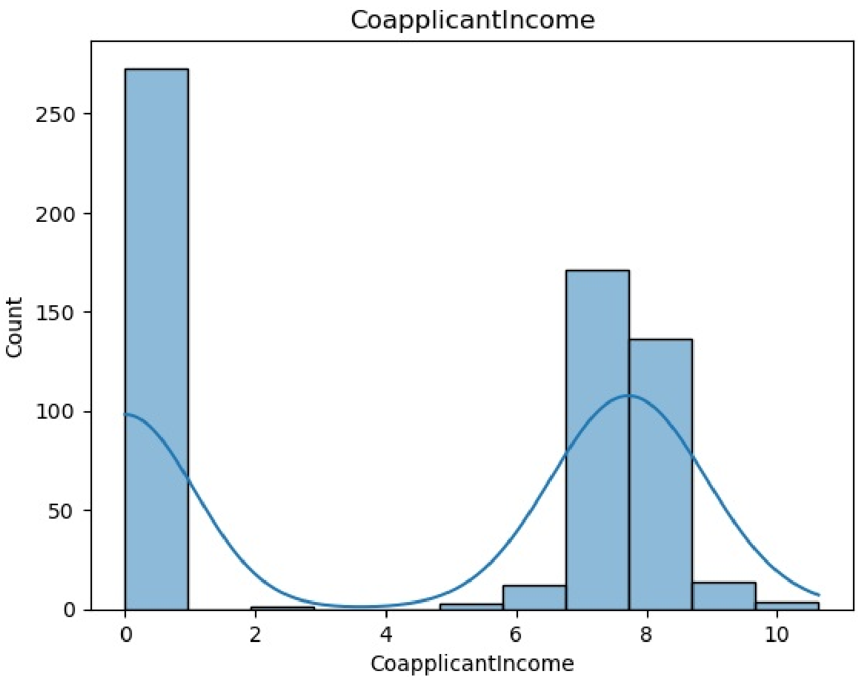
In [37]:
```
numerical_columns['ApplicantIncome']=np.log(numerical_columns['ApplicantIncome']+1)
numerical_columns['CoapplicantIncome']=np.log(numerical_columns['CoapplicantIncome']+1)
numerical_columns['LoanAmount']=np.log(numerical_columns['LoanAmount']+1)
```

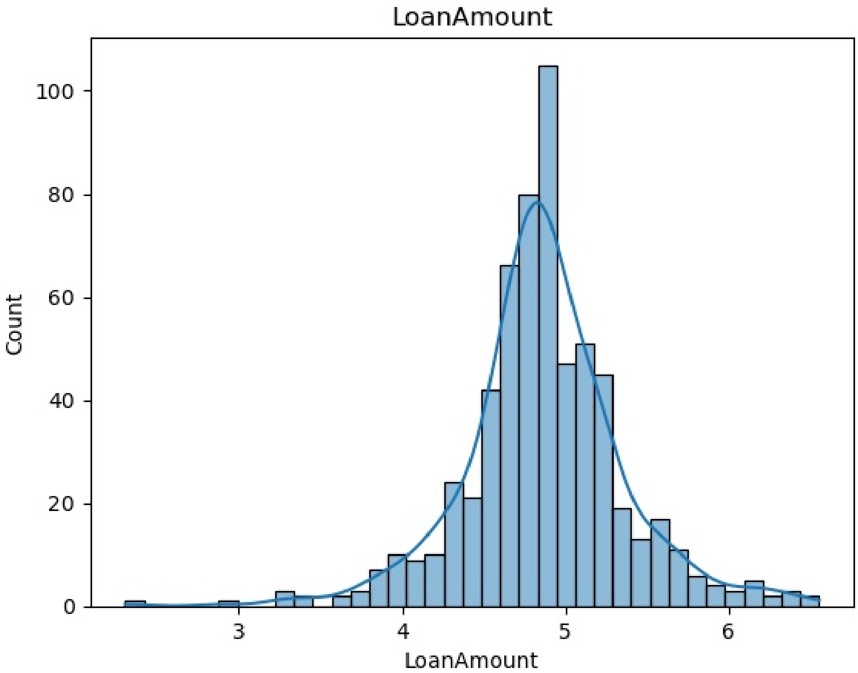
In [38]:
```
for i in columns:
    print(f"{i}---->{skew(numerical_columns[i])}")
```

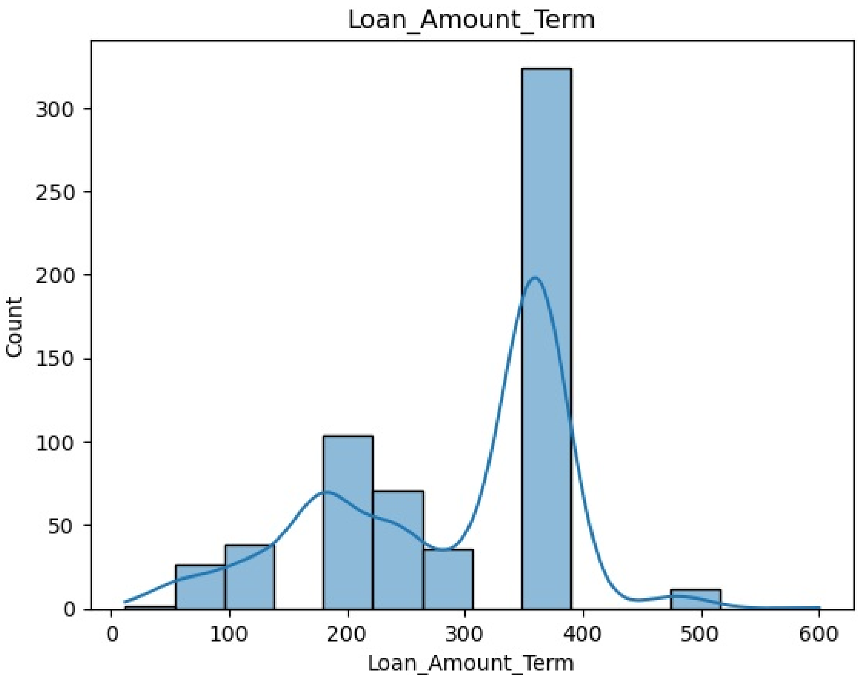
```
ApplicantIncome---->0.4809493580463021
CoapplicantIncome---->-0.17265017128703458
LoanAmount---->-0.1512069504656971
Loan_Amount_Term---->-0.6277838196730566
```

In [39]:
```
for i in columns:
    sns.histplot(numerical_columns[i], kde=True)
    plt.title(i)
    plt.show()
```





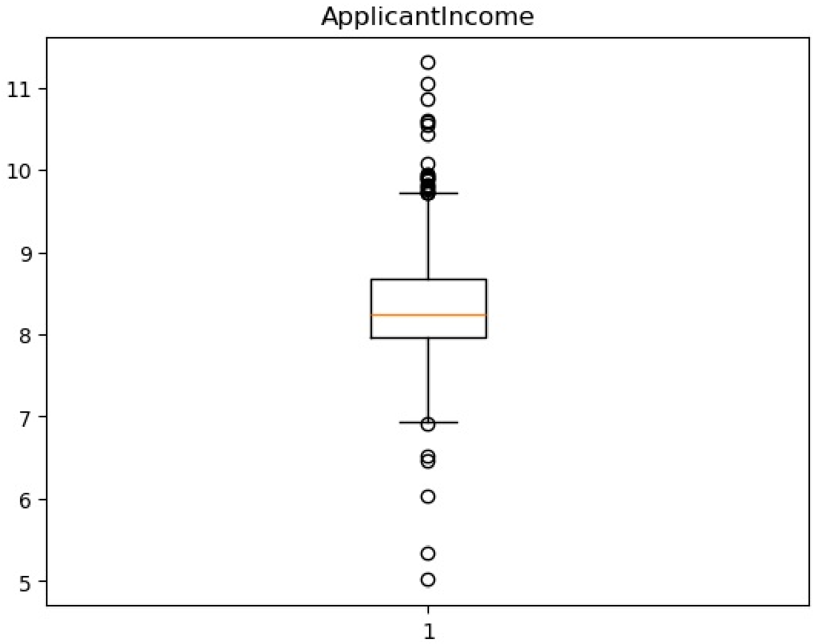


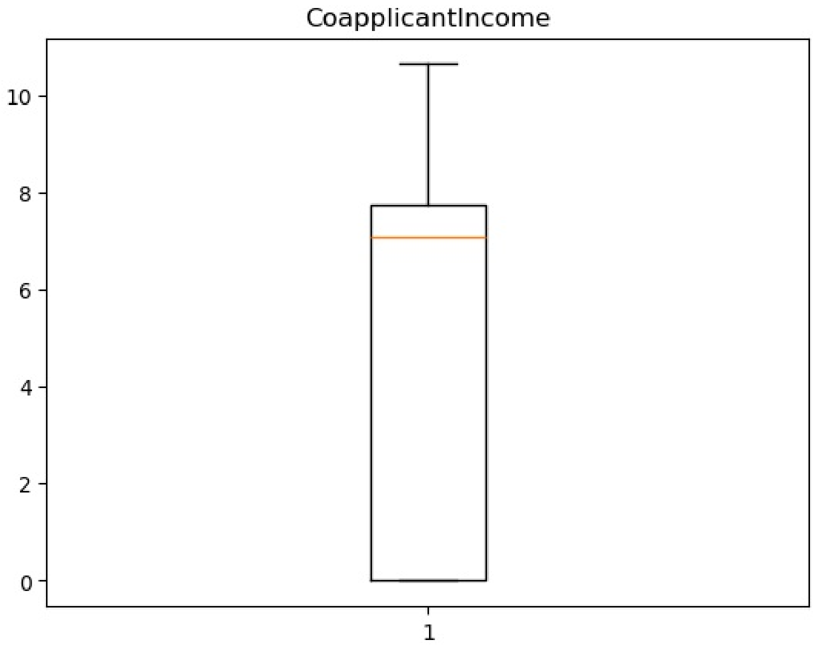


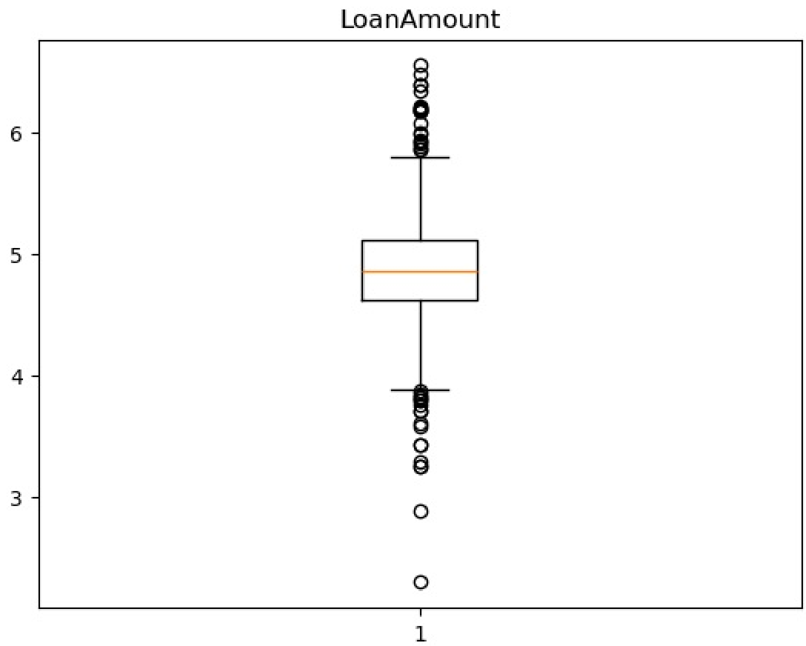
Outlier Detection

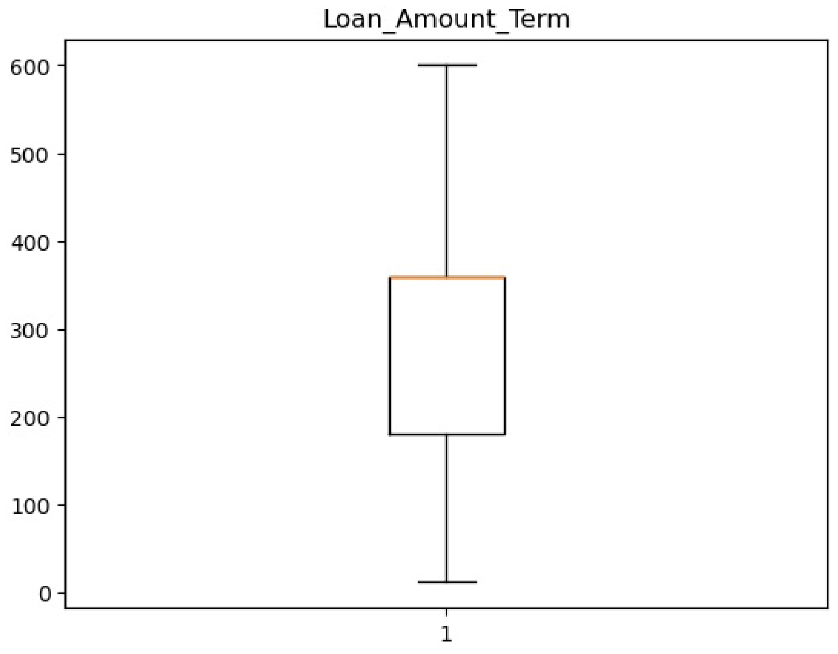
In [41]:
```
# Outliers
```

```python
for i in columns:
    plt.boxplot(numerical_columns[i])
    plt.title(i)
    plt.show()
```

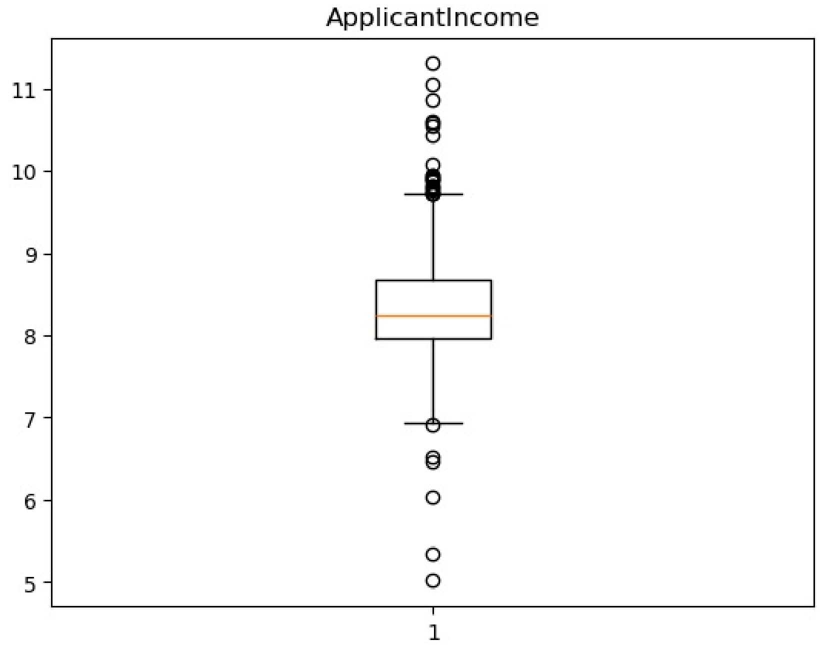








In [42]:
```
## ApplicantIncome
plt.figure()
plt.boxplot(numerical_columns['ApplicantIncome'])
plt.title('ApplicantIncome')
plt.show()
q1=numerical_columns['ApplicantIncome'].quantile(0.25)
q3=numerical_columns['ApplicantIncome'].quantile(0.75)
iqr=q3-q1
upper_bound=q3+1.5*iqr
lower_bound=q1-1.5*iqr
print("Q1=",q1)
print("Q3=",q3)
print("IQR=",iqr)
print("Upper_Bound=",upper_bound)
print("Lower_Bound=",lower_bound)
```

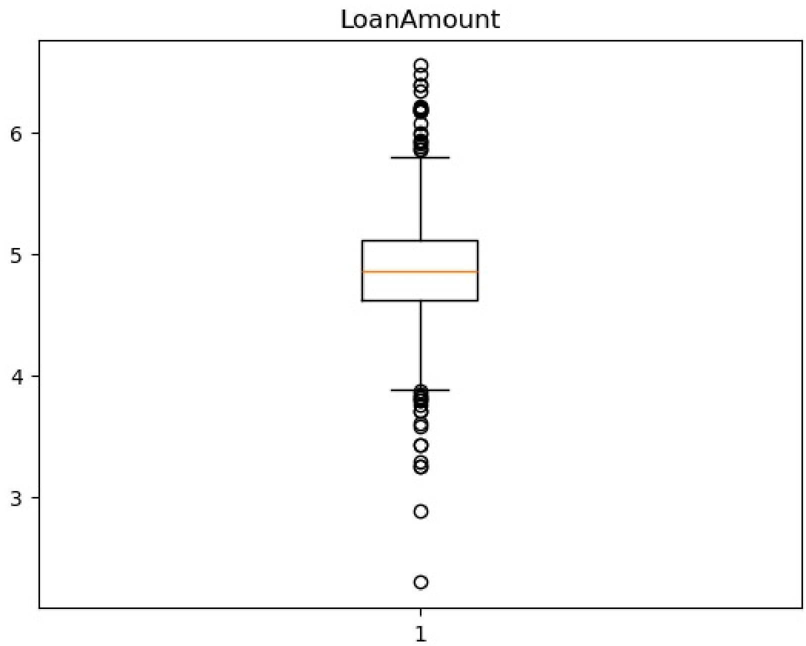



```
Q1= 7.965024197073261
Q3= 8.664922185870166
IQR= 0.6998979887969048
Upper_Bound= 9.714769169065523
Lower_Bound= 6.915177213877904
```

In [43]:
```
numerical_columns.loc[numerical_columns['ApplicantIncome']>upper_bound,'ApplicantIncome']=upper_bound
numerical_columns.loc[numerical_columns['ApplicantIncome']<lower_bound,'ApplicantIncome']=lower_bound
```

In [44]:
```
## LoanAmount
plt.figure()
plt.boxplot(numerical_columns['LoanAmount'])
plt.title('LoanAmount')
```

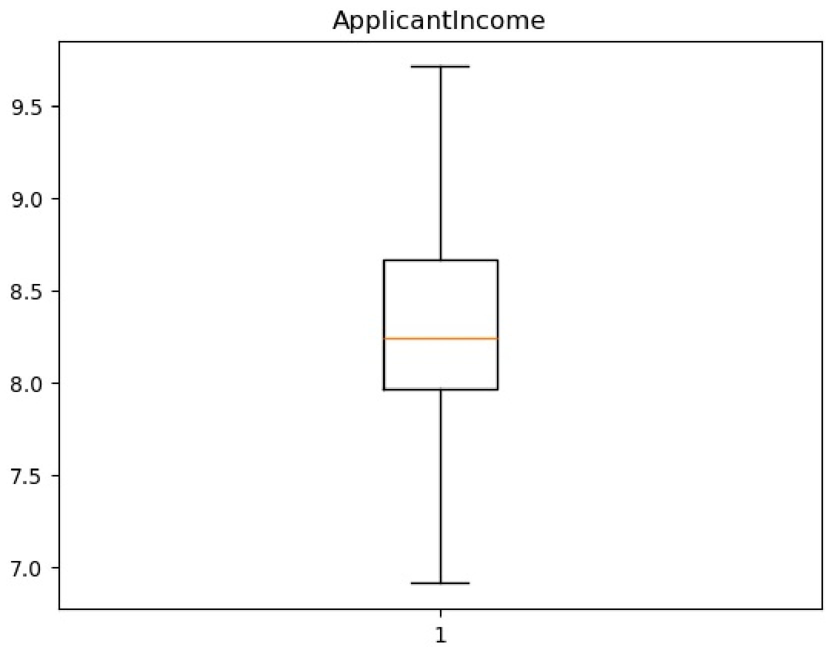
```
plt.show()
q1=numerical_columns['LoanAmount'].quantile(0.25)
q3=numerical_columns['LoanAmount'].quantile(0.75)
iqr=q3-q1
upper_bound=q3+1.5*iqr
lower_bound=q1-1.5*iqr
print("Q1=",q1)
print("Q3=",q3)
print("IQR=",iqr)
print("Upper_Bound=",upper_bound)
print("Lower_Bound=",lower_bound)
```

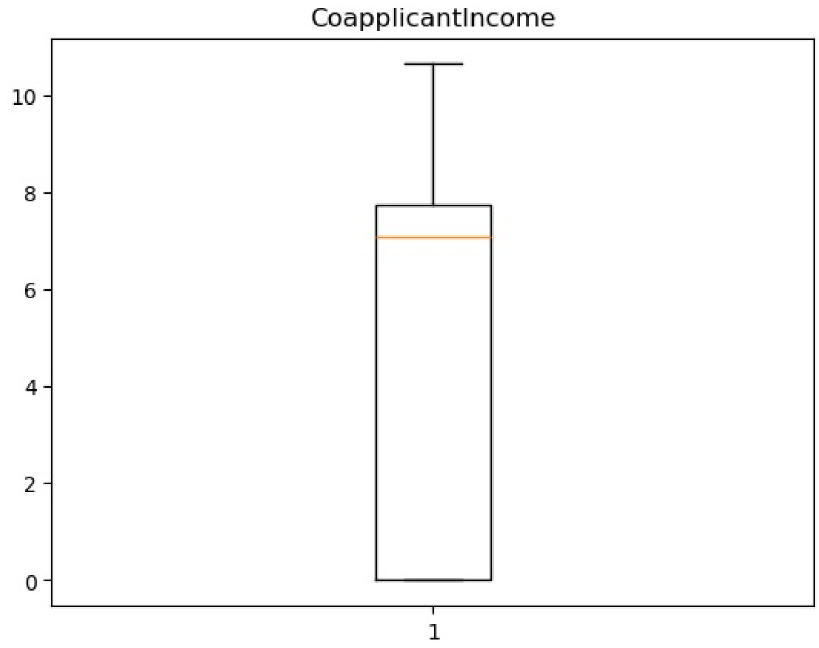


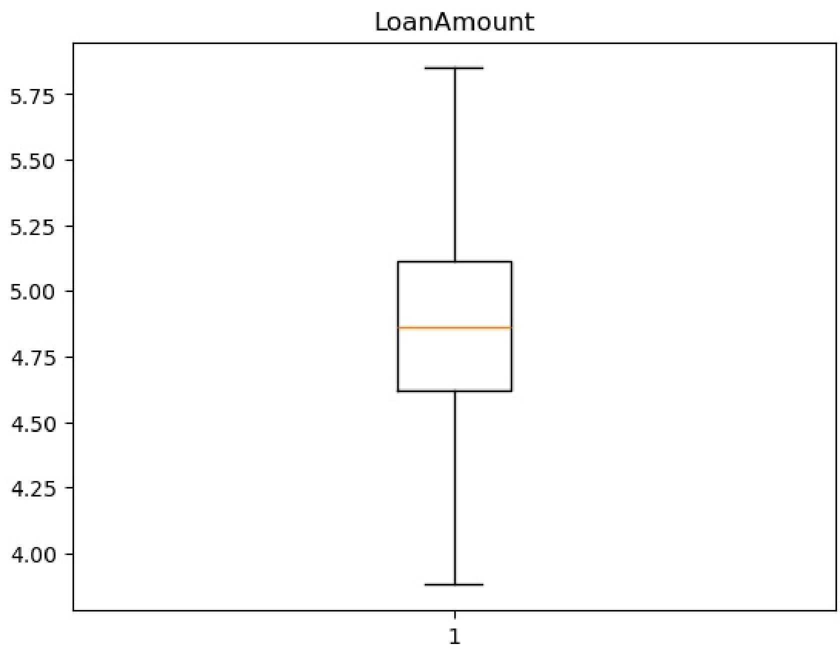

```
Q1= 4.617583590952012
Q3= 5.110477209742553
IQR= 0.49289361879054017
Upper_Bound= 5.849817637928362
Lower_Bound= 3.878243162766202
```

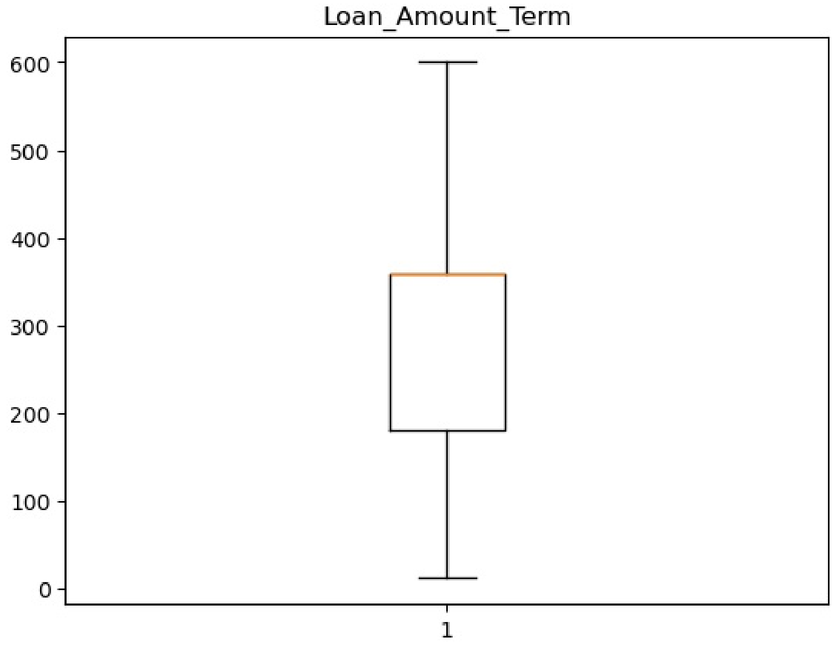
In [45]:
```
numerical_columns.loc[numerical_columns['LoanAmount']>upper_bound,'LoanAmount']=upper_bound
numerical_columns.loc[numerical_columns['LoanAmount']<lower_bound,'LoanAmount']=lower_bound
```

In [46]:
```
for i in columns:
    plt.boxplot(numerical_columns[i])
    plt.title(i)
    plt.show()
```




CoapplicantIncome
10
8
6
4
2
0
1


LoanAmount
5.75
5.50
5.25
5.00
4.75
4.50
4.25
4.00
1



In [47]:
```
from sklearn.preprocessing import StandardScaler
```

In [48]:
```
scaler = StandardScaler()
```

In [49]:
```
columns = ['Dependents', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term']
```

In [50]:
```
for i in columns:
    numerical_columns[i] = scaler.fit_transform(numerical_columns[[i]])
```

In [51]:
```
numerical_columns.head()
```

Out[51]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	-0.737806	0.595849	-1.107783	-0.016022	-0.463951	1
1	0.253470	0.167494	0.782158	-0.016022	-0.463951	1
2	-0.737806	-0.576603	-1.107783	-1.531848	-0.463951	1
3	-0.737806	-0.839386	0.897526	-0.164156	-0.463951	1
4	-0.737806	0.640612	-1.107783	0.206139	-0.463951	1

Categorical Columns

In [53]:
```
categorical_columns = df.select_dtypes(include=object)
categorical_columns
```

Out[53]:

	Loan_ID	Gender	Married	Education	Self_Employed	Property_Area	Loan_Status
0	LP001002	Male	No	Graduate	No	Urban	Y
1	LP001003	Male	Yes	Graduate	No	Rural	N
2	LP001005	Male	Yes	Graduate	Yes	Urban	Y
3	LP001006	Male	Yes	Not Graduate	No	Urban	Y
4	LP001008	Male	No	Graduate	No	Urban	Y
...	...	...	...	...	...	...	...
609	LP002978	Female	No	Graduate	No	Rural	Y
610	LP002979	Male	Yes	Graduate	No	Rural	Y
611	LP002983	Male	Yes	Graduate	No	Urban	Y
612	LP002984	Male	Yes	Graduate	No	Urban	Y
613	LP002990	Female	No	Graduate	Yes	Semiurban	N

614 rows × 7 columns

Dropping Loan_ID column

The Loan_ID column is dropped because it contains unique values with no predictive value, introduces noise, lacks patterns or trends related to the target variable.

In [56]:
```
categorical_columns.drop('Loan_ID', axis=1, inplace=True)
```

In [57]:
```
for i in categorical_columns:
    print(categorical_columns[i].value_counts())
    print("="*30)
```

```
Gender
Male      502
Female    112
Name: count, dtype: int64
==============================
Married
Yes    401
No     213
Name: count, dtype: int64
==============================
Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64
==============================
Self_Employed
No     532
Yes     82
Name: count, dtype: int64
==============================
Property_Area
Semiurban    233
Urban        202
Rural        179
Name: count, dtype: int64
==============================
Loan_Status
Y    422
N    192
Name: count, dtype: int64
==============================
```

Data Encoding

In [59]:
```
#Label encoding
from sklearn.preprocessing import LabelEncoder
```

In [60]:
```
lb = LabelEncoder()
```

In [61]:
```
columns = ['Gender', 'Married', 'Education', 'Self_Employed', 'Loan_Status']
```

In [62]:
```
for i in columns:
    categorical_columns[i]=lb.fit_transform(categorical_columns[i])
```

In [63]:
```
categorical_columns.head()
```

Out[63]:

	Gender	Married	Education	Self_Employed	Property_Area	Loan_Status
0	1	0	0	0	Urban	1
1	1	1	0	0	Rural	0
2	1	1	0	1	Urban	1
3	1	1	1	0	Urban	1
4	1	0	0	0	Urban	1

In [64]:

```
#one-hot Encoding(get dummies)
categorical_columns = pd.get_dummies(categorical_columns, columns=['Property_Area']).astype(int)
```

In [65]:

```
data = pd.concat([numerical_columns, categorical_columns], axis=1)
data.head()
```

Out[65]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender	Married	Educatio
0	-0.737806	0.595849	-1.107783	-0.016022	-0.463951	1	1	0	(
1	0.253470	0.167494	0.782158	-0.016022	-0.463951	1	1	1	(
2	-0.737806	-0.576603	-1.107783	-1.531848	-0.463951	1	1	1	(
3	-0.737806	-0.839386	0.897526	-0.164156	-0.463951	1	1	1	'
4	-0.737806	0.640612	-1.107783	0.206139	-0.463951	1	1	0	(

In [66]:

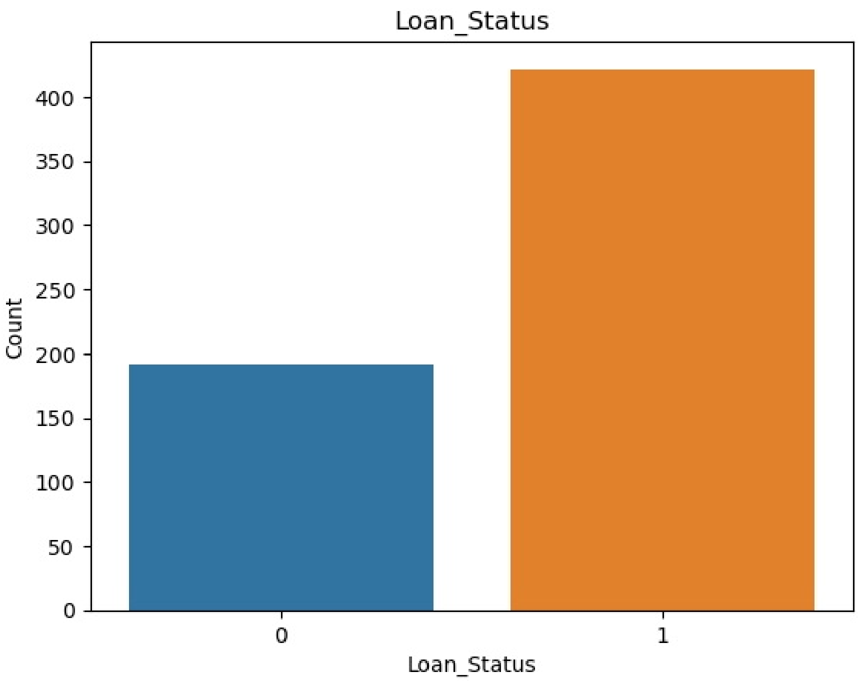
```
print(dict(enumerate(lb.classes_)))
```

```
{0: 'N', 1: 'Y'}
```

In [67]:

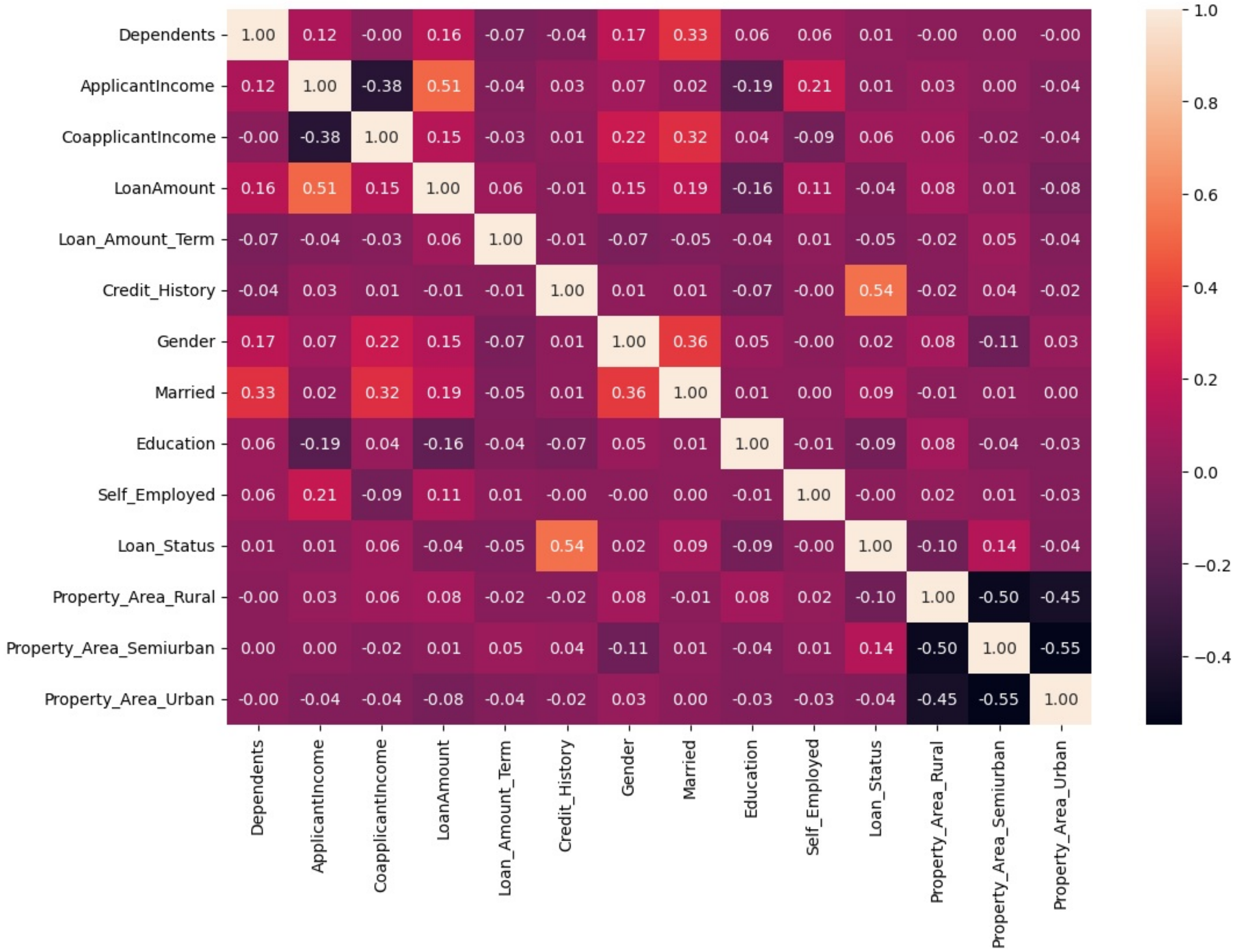
```
print(dict(enumerate(lb.classes_)))
palette=sns.color_palette()
sns.countplot(x=data['Loan_Status'],palette=palette)
plt.title('Loan_Status')
plt.xlabel('Loan_Status')
plt.ylabel('Count')
plt.show()
```

```
{0: 'N', 1: 'Y'}
```



In [68]:

```
plt.figure(figsize=(12,8))
sns.heatmap(data.corr(),fmt = '.2f',annot = True)
plt.show()
```



The Self_Employed column is removed from the dataset because its correlation with the target variable is -0.00, indicating no meaningful relationship or contribution to the predictive model.

In [70]:
```
data.drop('Self_Employed', axis=1, inplace=True)
```

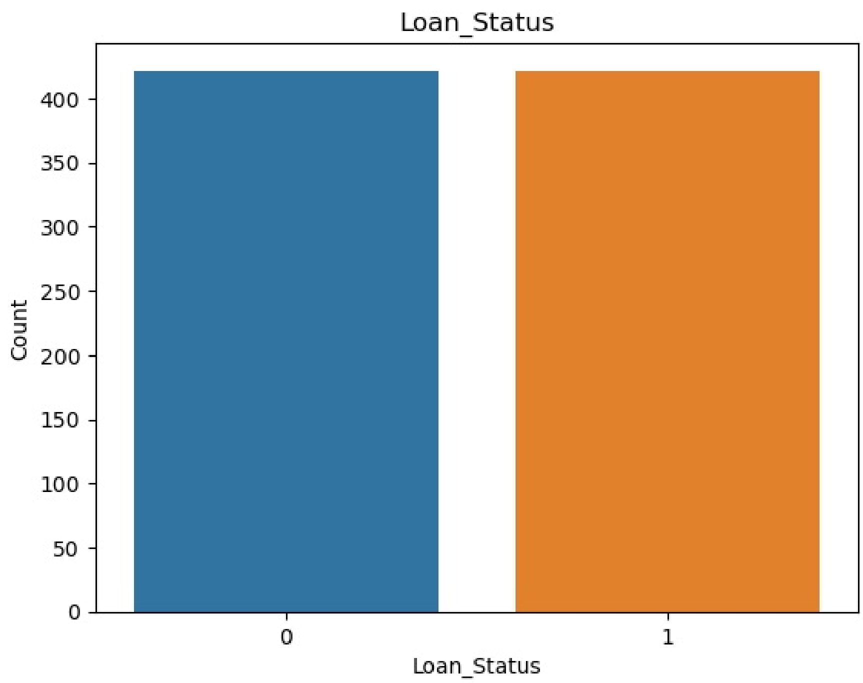
Handling imbalanced data

In [72]:
```
from imblearn.over_sampling import SMOTE
x = data.drop('Loan_Status', axis=1)
y = data['Loan_Status']

smt = SMOTE(random_state = 10)
x_sample, y_sample = smt.fit_resample(x,y)

x = x_sample
y = y_sample
```

In [73]:
```
palette=sns.color_palette()
sns.countplot(x=y,palette=palette)
plt.title('Loan_Status')
plt.xlabel('Loan_Status')
plt.ylabel('Count')
plt.show()
```



In [74]:
```
#Train-Test-Split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2, random_state=11)
```

Logistic Regression Model

In [76]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

In [77]:
```
lr_model = LogisticRegression()
lr_model.fit(x_train,y_train)
```

Out[77]:
```
▾ LogisticRegression
LogisticRegression()
```

In [78]:
```
y_pred_lr_test = lr_model.predict(x_test)

accuracy_lr_test = accuracy_score(y_test, y_pred_lr_test)*100

print("Accuracy score testing:", accuracy_lr_test)
```

```
Accuracy score testing: 78.69822485207101
```

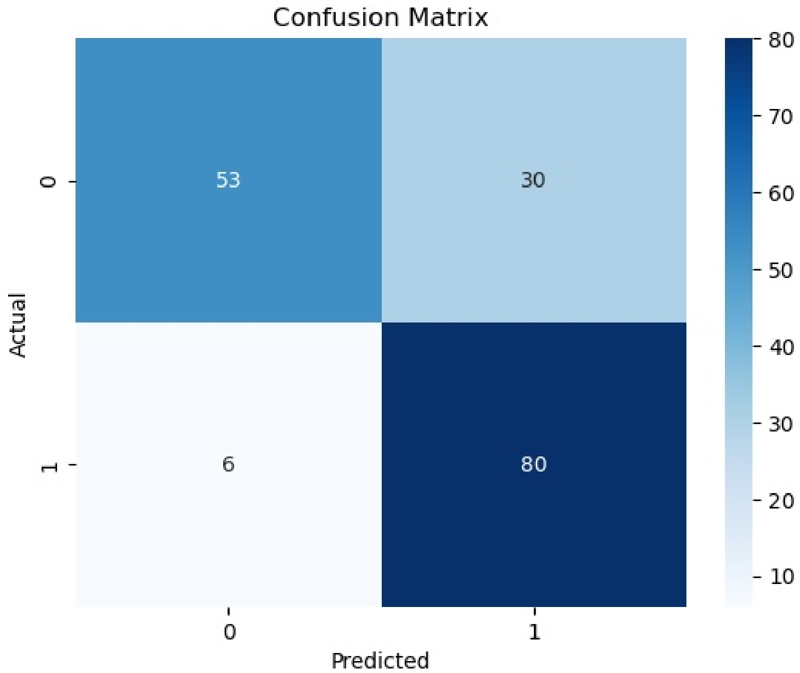
In [79]:
```
y_pred_lr_train=lr_model.predict(x_train)

accuracy_lr_train=accuracy_score(y_train,y_pred_lr_train)*100

print("Accuracy score Training:",accuracy_lr_train)
```

```
Accuracy score Training: 78.66666666666666
```

In [80]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
# Plot confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_lr_test)
sns.heatmap(conf_matrix, annot=True, cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [81]:
```
report_lr = classification_report(y_test, y_pred_lr_test)
print(report_lr)
```

```
              precision    recall  f1-score   support

           0       0.90      0.64      0.75        83
           1       0.73      0.93      0.82        86

    accuracy                           0.79       169
   macro avg       0.81      0.78      0.78       169
weighted avg       0.81      0.79      0.78       169
```

Random Forest Model

In [83]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [84]:
```
rf_model=RandomForestClassifier()
rf_model.fit(x_train,y_train)
```

Out[84]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [85]:
```
y_pred_rf_test=rf_model.predict(x_test)

accuracy_rf=accuracy_score(y_test,y_pred_rf_test)*100

print("Accuracy score testing:",accuracy_rf)
```

```
Accuracy score testing: 82.24852071005917
```

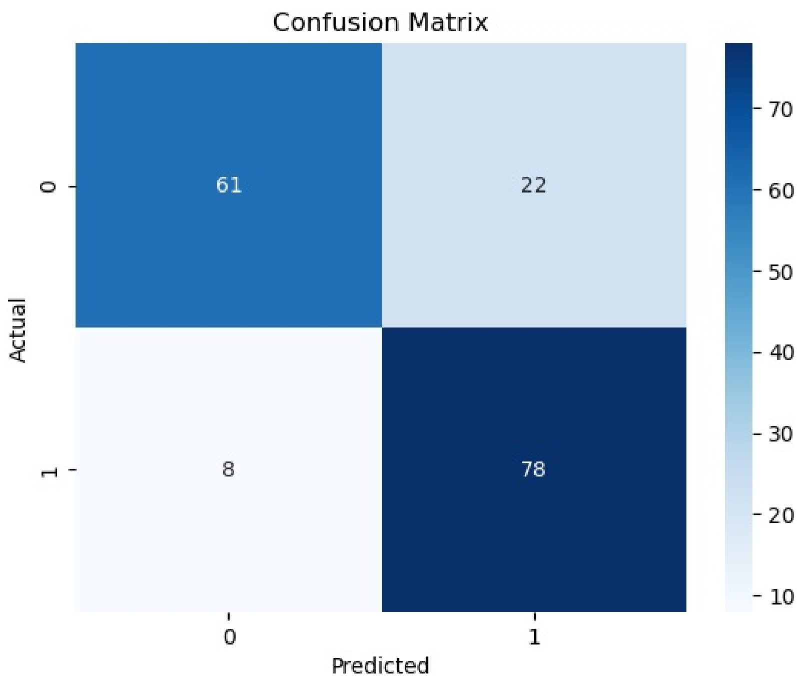
In [86]:
```
y_pred_rf_train=rf_model.predict(x_train)

accuracy_rf_train=accuracy_score(y_train,y_pred_rf_train)

print("Accuracy score Training:",accuracy_rf_train*100)
```

```
Accuracy score Training: 100.0
```

In [87]:
```
conf_matrix = confusion_matrix(y_test, y_pred_rf_test)
sns.heatmap(conf_matrix, annot=True, cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [88]:
```
report_lr = classification_report(y_test, y_pred_rf_test)
print(report_lr)
```

```
              precision    recall  f1-score   support

           0       0.88      0.73      0.80        83
           1       0.78      0.91      0.84        86

    accuracy                           0.82       169
   macro avg       0.83      0.82      0.82       169
weighted avg       0.83      0.82      0.82       169
```

AdaBoost Model

In [90]:
```
from sklearn.ensemble import AdaBoostClassifier
adb_classify=AdaBoostClassifier(random_state=10)
adb_classify.fit(x_sample,y_sample)
```

Out[90]:
```
▾ AdaBoostClassifier
AdaBoostClassifier(random_state=10)
```

In [91]:
```
y_pred_train_adb = adb_classify.predict(x_train)
accuracy_ada_train = accuracy_score(y_train, y_pred_train_adb)

print("Accuracy score traning:",accuracy_ada_train*100)
```

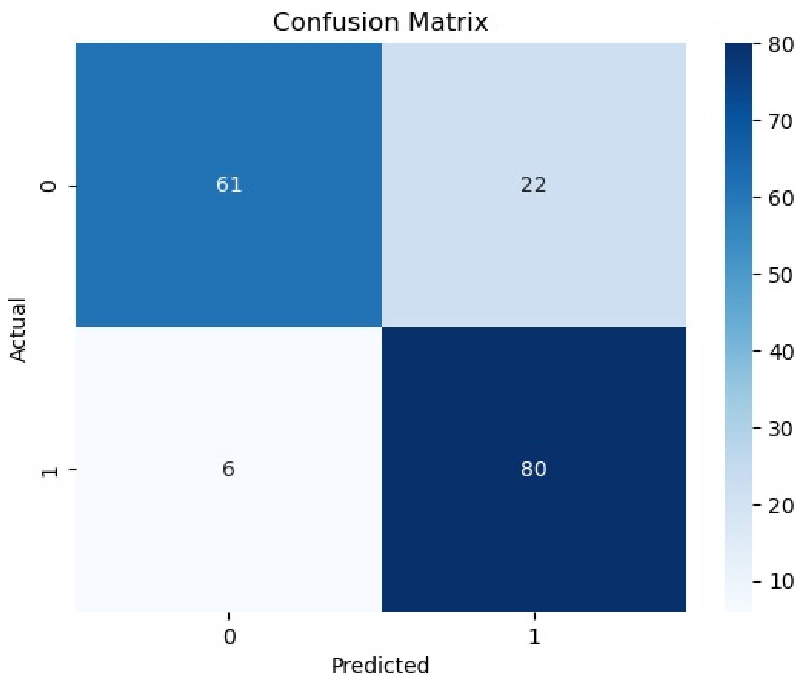
```
Accuracy score traning: 82.37037037037037
```

In [92]:
```
y_pred_test_adb = adb_classify.predict(x_test)
accuracy_ada_test = accuracy_score(y_test, y_pred_test_adb)*100

print("Accuracy score testing:",accuracy_ada_test)
```

```
Accuracy score testing: 83.4319526627219
```

In [93]:
```
conf_matrix = confusion_matrix(y_test, y_pred_test_adb)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



In [94]:
```
report_lr = classification_report(y_test, y_pred_test_adb)
print(report_lr)
```

```
              precision    recall  f1-score   support

           0       0.91      0.73      0.81        83
           1       0.78      0.93      0.85        86

    accuracy                           0.83       169
   macro avg       0.85      0.83      0.83       169
weighted avg       0.85      0.83      0.83       169
```

In [95]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.5],
    'algorithm': ['SAMME.R', 'SAMME']
}

grid_search_ada = GridSearchCV(estimator=adb_classify, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search_ada.fit(x_train, y_train)
print("Best Parameters:", grid_search_ada.best_params_)
```

```
Best Parameters: {'algorithm': 'SAMME.R', 'learning_rate': 0.1, 'n_estimators': 150}
```

In [96]:
```
y_pred_train_adb = grid_search_ada.predict(x_train)
accuracy = accuracy_score(y_train, y_pred_train_adb)

print("Accuracy score traning:",accuracy*100)
```

```
Accuracy score traning: 79.4074074074074
```

In [97]:
```
y_pred_test_adb = grid_search_ada.predict(x_test)
accuracy = accuracy_score(y_test, y_pred_test_adb)

print("Accuracy score testing:",accuracy*100)
```

```
Accuracy score testing: 77.51479289940828
```

In [98]:
```
Dict={'Model':['Logistic_Regression', 'Random_Forest', 'AdaBoost'],
      'Accuracy':[accuracy_lr_test, accuracy_rf, accuracy_ada_test]}
```

In [99]:
```
model_accuracy=pd.DataFrame(Dict,columns=['Model','Accuracy'])
model_accuracy
```

Out[99]:

	Model	Accuracy
0	Logistic_Regression	78.698225
1	Random_Forest	82.248521
2	AdaBoost	83.431953