Loan Status Prediction

Description:This Dataset contains informations about loab applicants,including their personal and financial details and whether they were approved for a loan.It's ideal for building a logistic regression model to predict loan approval.

Importing the Libraries.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as pltM
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_r
import warnings
warnings.filterwarnings("ignore")
```

Loading the dataset

In [3]:
```
data=pd.read_csv("loan.csv")
```

In [4]:
```
data
```

Out[4]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	Appl
0	LP001002	Male	No	0	Graduate	No	
1	LP001003	Male	Yes	1	Graduate	No	
2	LP001005	Male	Yes	0	Graduate	Yes	
3	LP001006	Male	Yes	0	Not Graduate	No	
4	LP001008	Male	No	0	Graduate	No	
...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	
610	LP002979	Male	Yes	3+	Graduate	No	
611	LP002983	Male	Yes	1	Graduate	No	
612	LP002984	Male	Yes	2	Graduate	No	
613	LP002990	Female	No	0	Graduate	Yes	

614 rows × 13 columns

In [5]:
```python
# Drop Loan_ID column at the very beginning
if 'Loan_ID' in data.columns:
    data = data.drop('Loan_ID', axis=1)

# Check columns
print(data.columns)
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Statu
s'],
      dtype='object')
```

Overviewing the Data.

In [6]:
```python
#Checking Missing Value
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Gender             601 non-null    object 
 1   Married            611 non-null    object 
 2   Dependents         599 non-null    object 
 3   Education          614 non-null    object 
 4   Self_Employed      582 non-null    object 
 5   ApplicantIncome    614 non-null    int64  
 6   CoapplicantIncome  614 non-null    float64
 7   LoanAmount         592 non-null    float64
 8   Loan_Amount_Term   600 non-null    float64
 9   Credit_History     564 non-null    float64
 10  Property_Area      614 non-null    object 
 11  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(7)
memory usage: 57.7+ KB
```

In [7]:
```python
#Check the missing values
data.isnull().sum()
```

Out[7]:
```
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 [8]:
```
#Summary Statistics
data.describe()
```

Out[8]:

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

Data Cleaning.

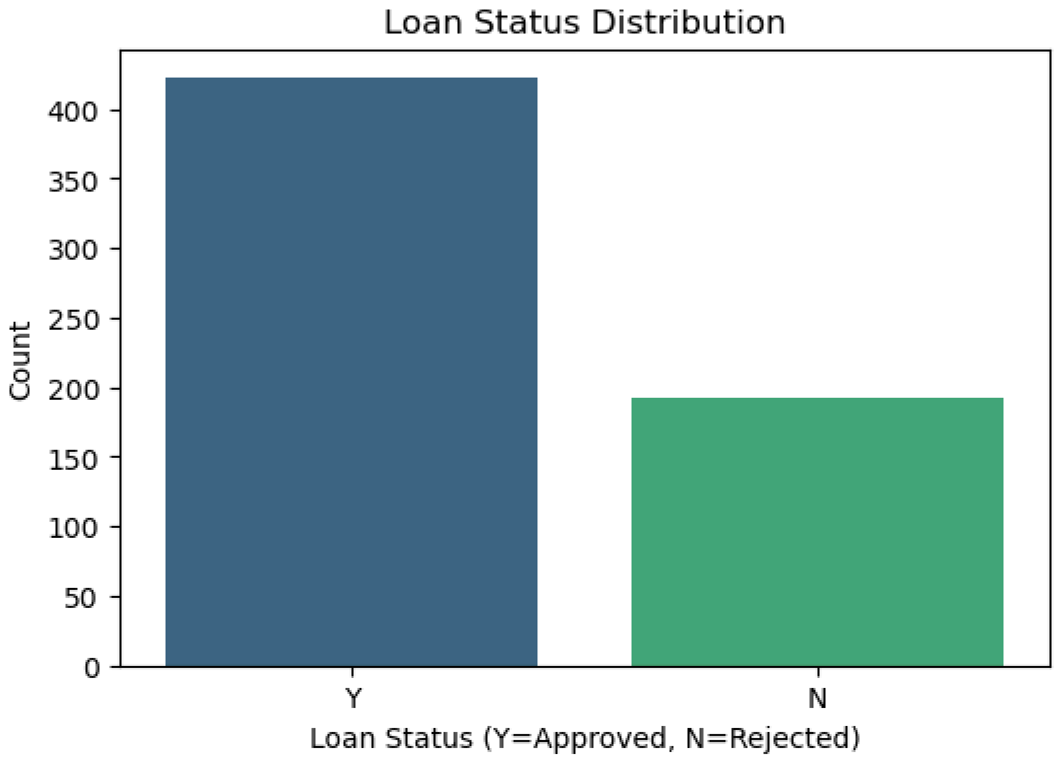
In [9]:
```
# Fill missing categorical values with mode
for col in ['Gender','Married','Dependents','Self_Employed']:
    data[col].fillna(data[col].mode()[0], inplace=True)

# Fill missing numeric values
data['LoanAmount'].fillna(data['LoanAmount'].median(), inplace=True)
data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0], inplace=
data['Credit_History'].fillna(data['Credit_History'].mode()[0], inplace=True

# Verify missing values are handled
data.isnull().sum()
```

Out[9]:
```
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
```

Exploratory Data Analysis.

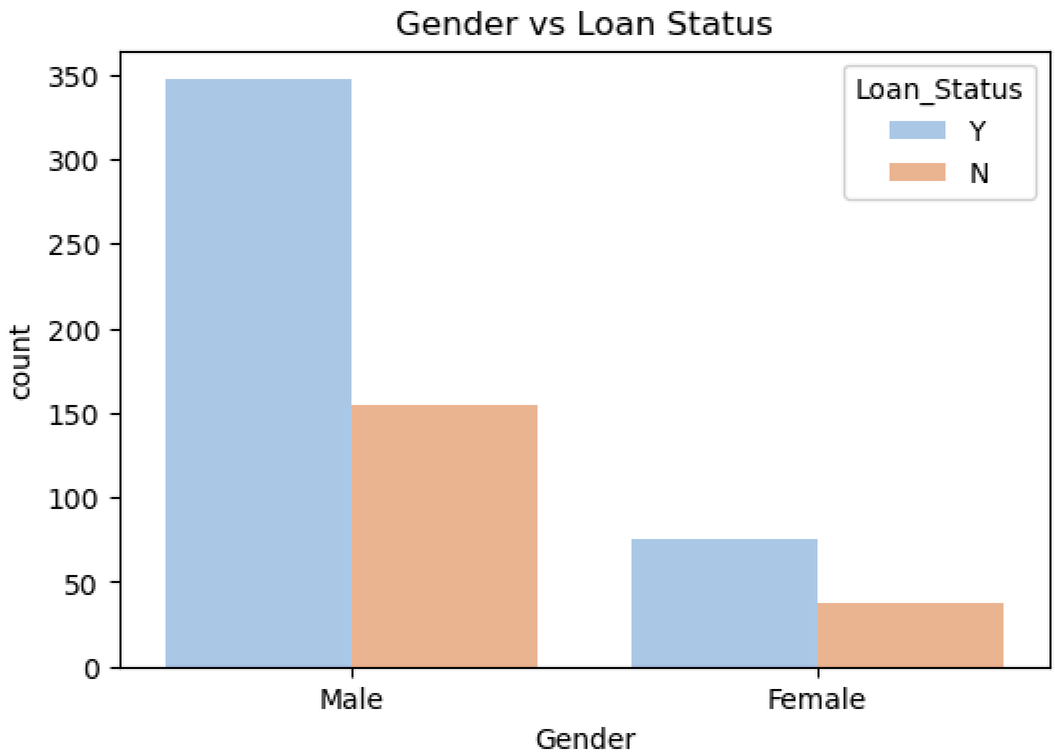
Loan Status Distribution

In [10]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Loan_Status', data=data, palette='viridis')
plt.title("Loan Status Distribution")
plt.xlabel("Loan Status (Y=Approved, N=Rejected)")
plt.ylabel("Count")
plt.show()
```



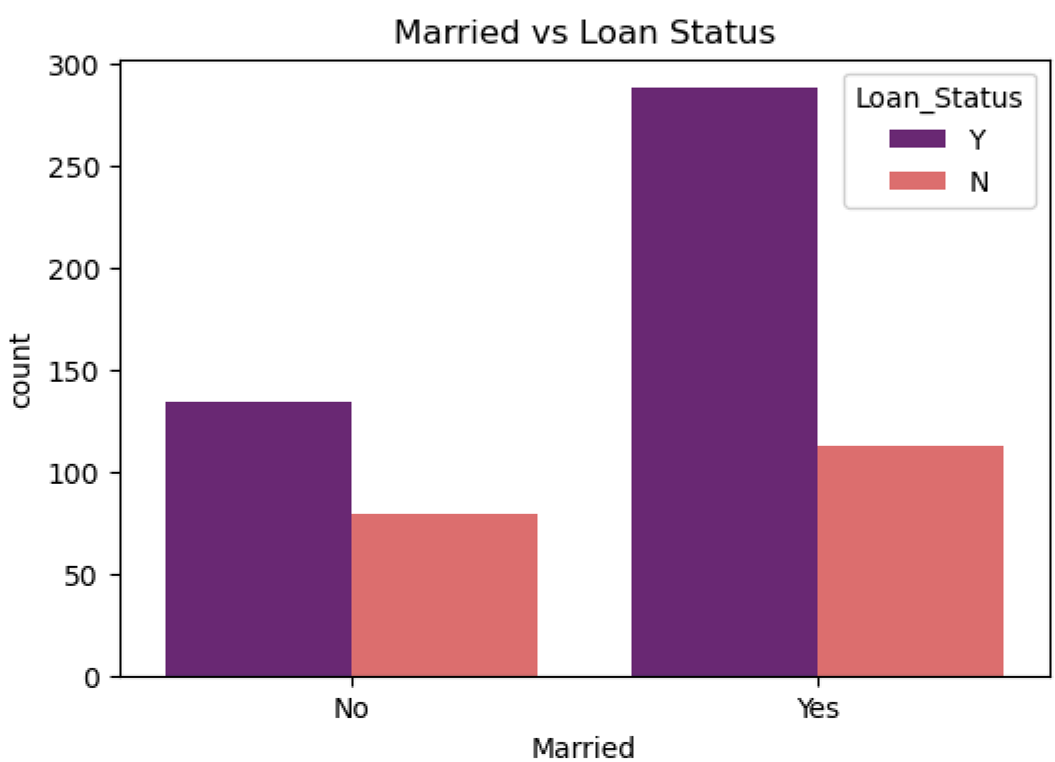
Gender VS Loan Status.

In [11]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Gender', hue='Loan_Status', data=data, palette='pastel')
plt.title("Gender vs Loan Status")
plt.show()
```



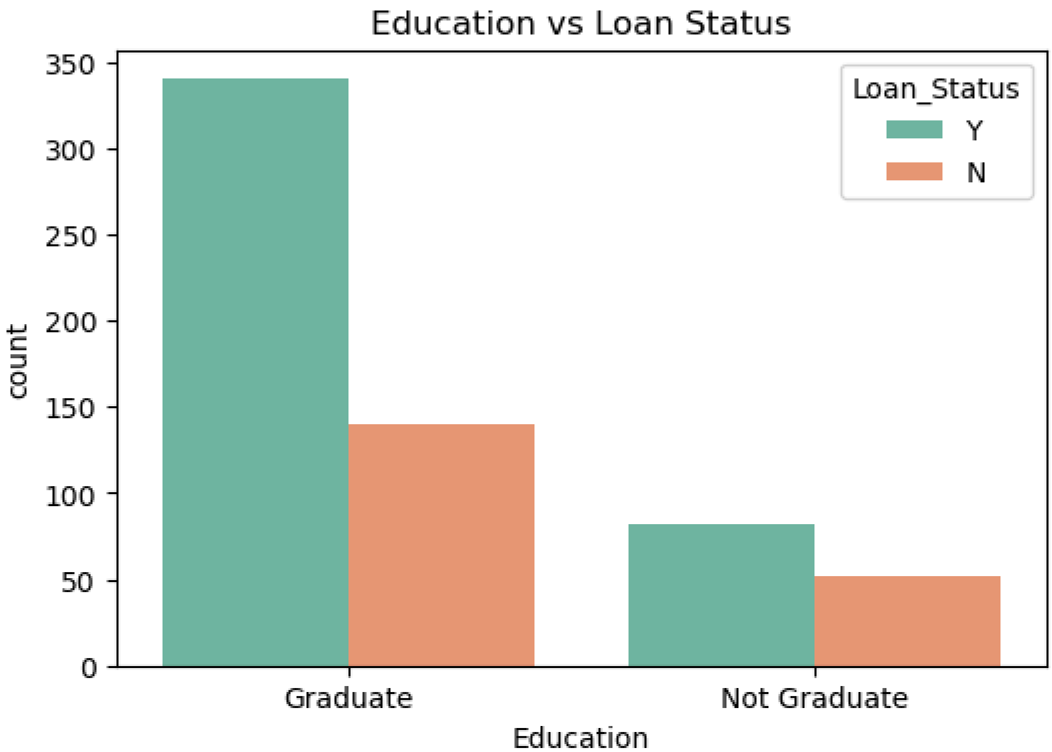
Married Vs Loan Status.

In [12]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Married', hue='Loan_Status', data=data, palette='magma')
plt.title("Married vs Loan Status")
plt.show()
```



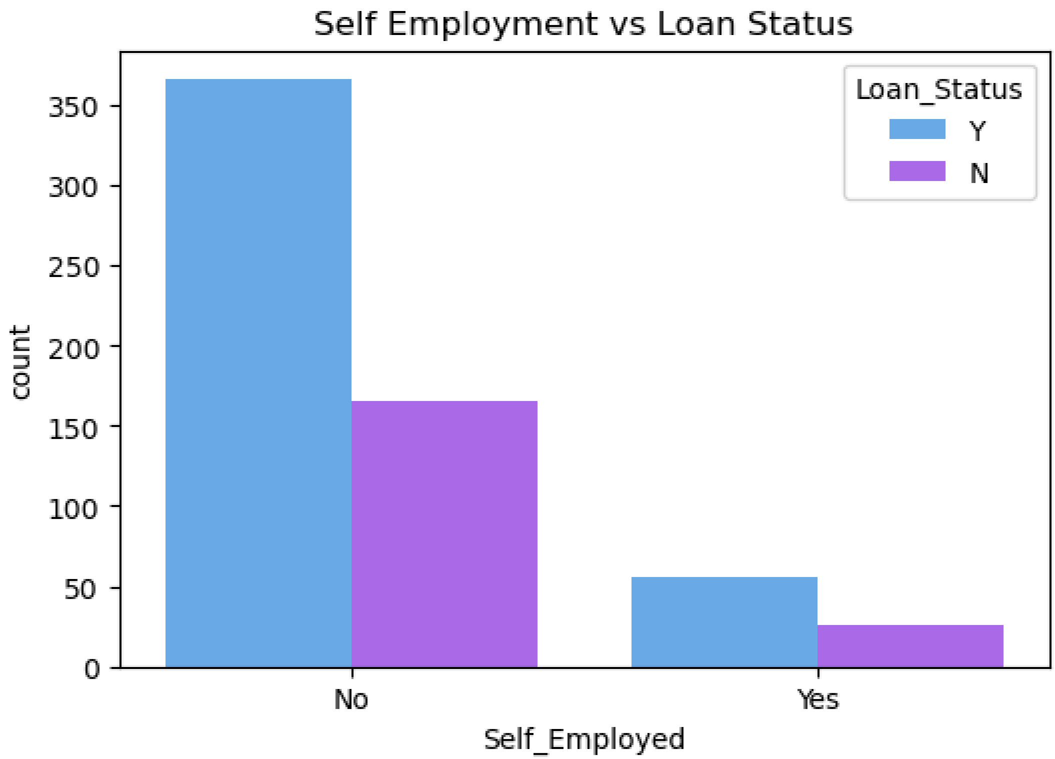
Education vs Loan Status.

In [13]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Education', hue='Loan_Status', data=data, palette='Set2')
plt.title("Education vs Loan Status")
plt.show()
```



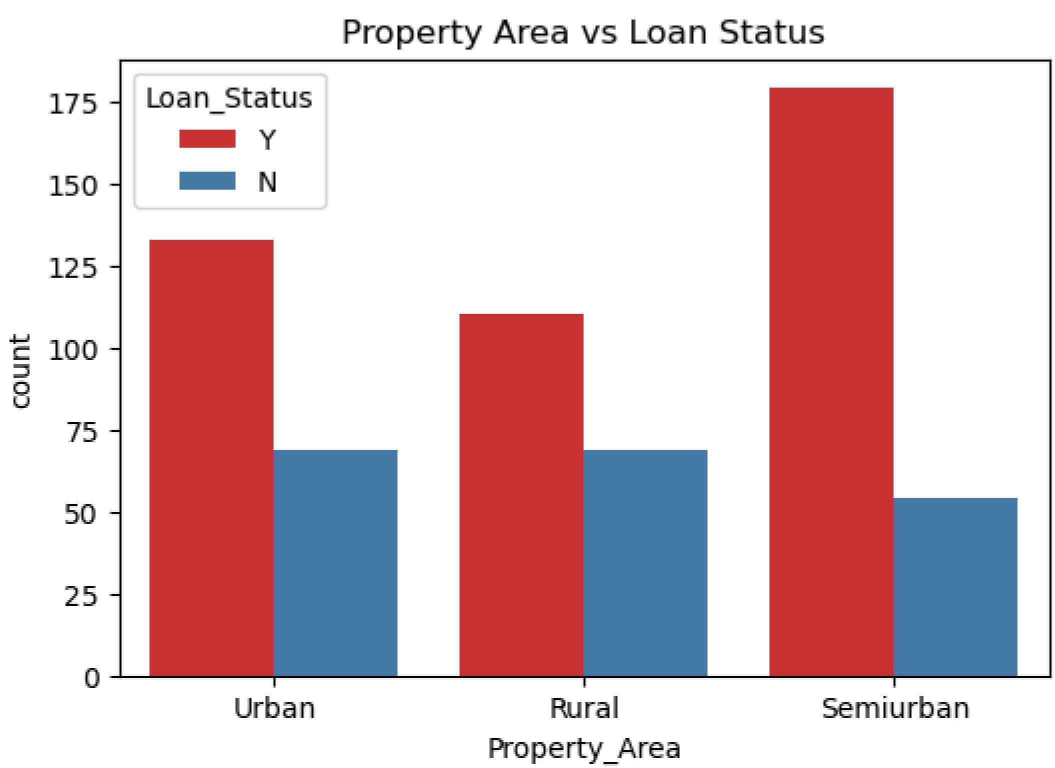
Self Employment vs Loan Status.

In [14]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Self_Employed', hue='Loan_Status', data=data, palette='cool
plt.title("Self Employment vs Loan Status")
plt.show()
```



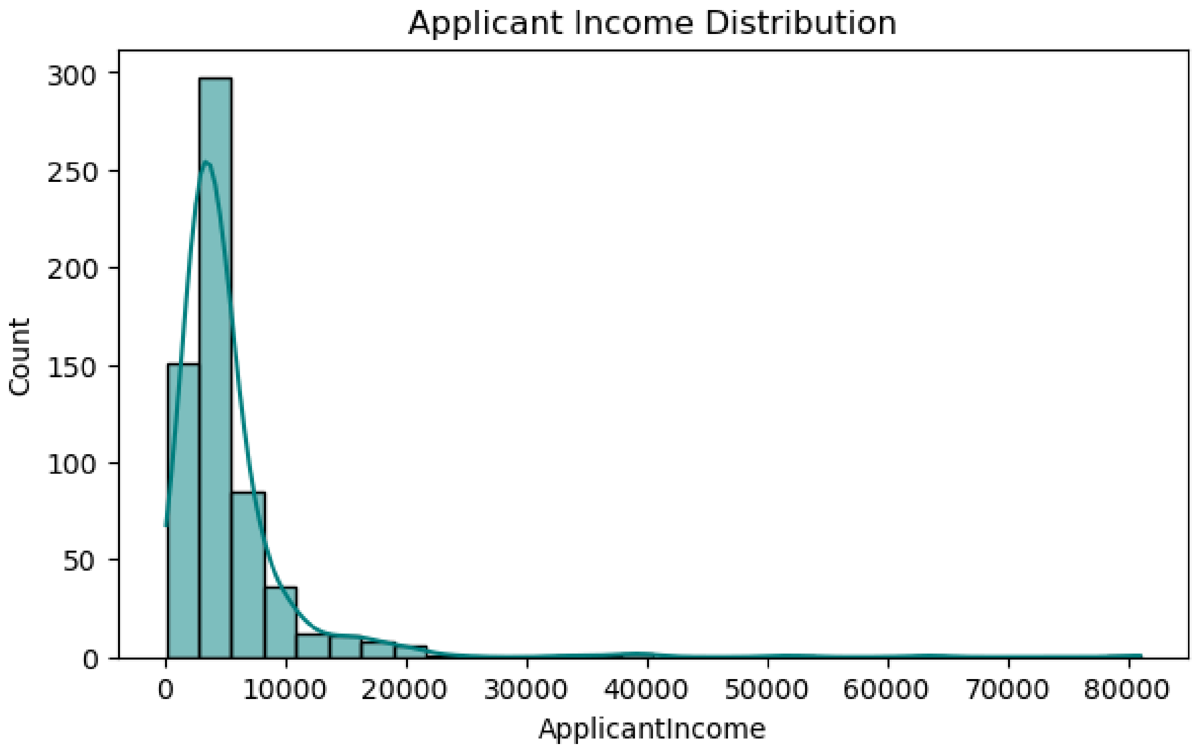
Property Area Vs Loan Status.

In [15]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Property_Area', hue='Loan_Status', data=data, palette='Set1
plt.title("Property Area vs Loan Status")
plt.show()
```



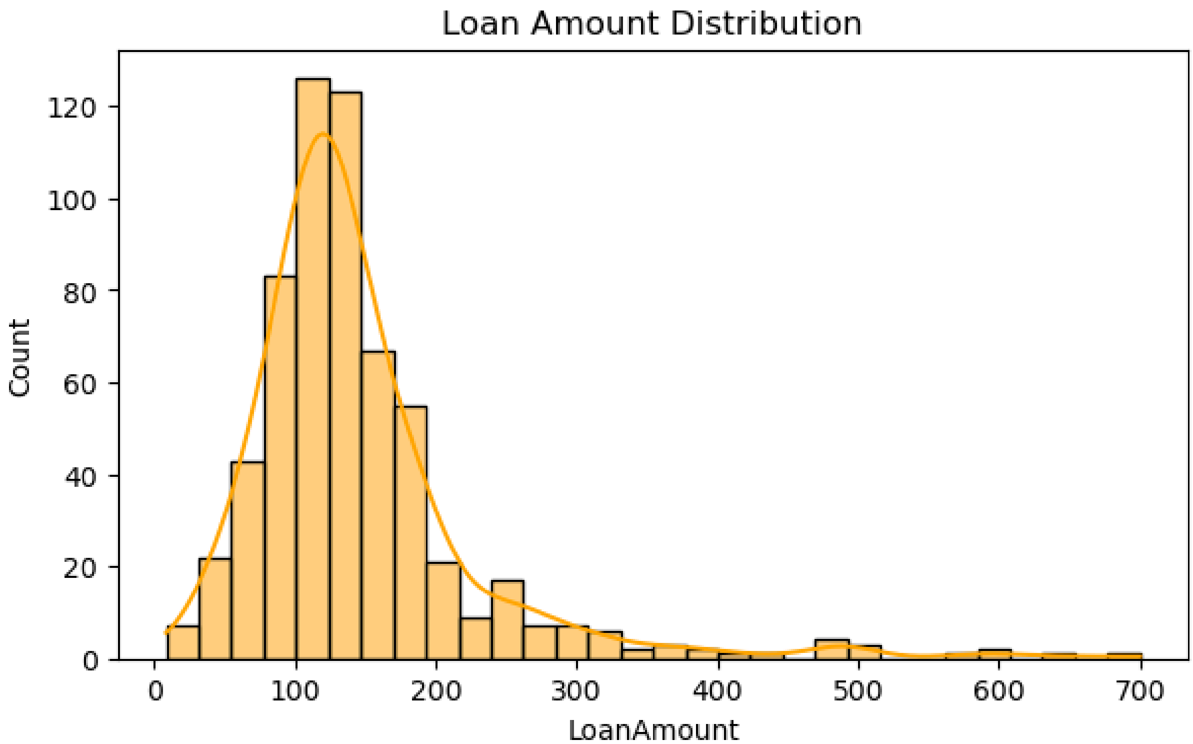
Applicant Income Distribution.

In [16]:
```
plt.figure(figsize=(7,4))
sns.histplot(data['ApplicantIncome'], bins=30, kde=True, color='teal')
plt.title("Applicant Income Distribution")
plt.show()
```



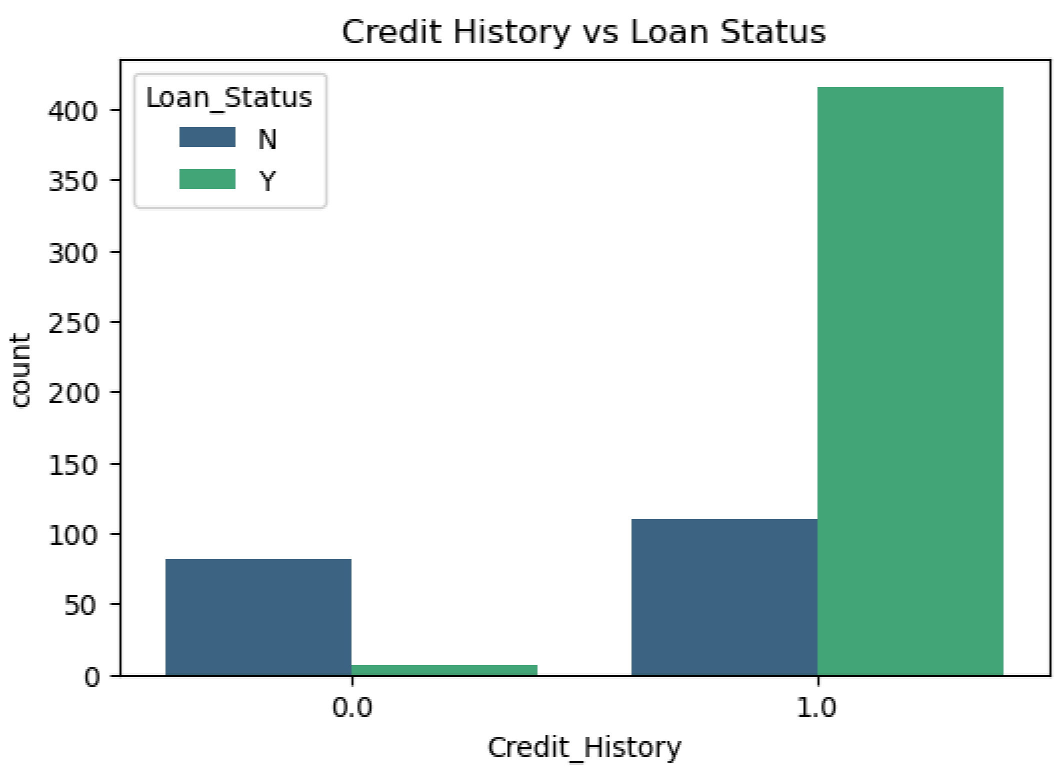
Loan Amount Distribution.

In [17]:
```
plt.figure(figsize=(7,4))
sns.histplot(data['LoanAmount'], bins=30, kde=True, color='orange')
plt.title("Loan Amount Distribution")
plt.show()
```



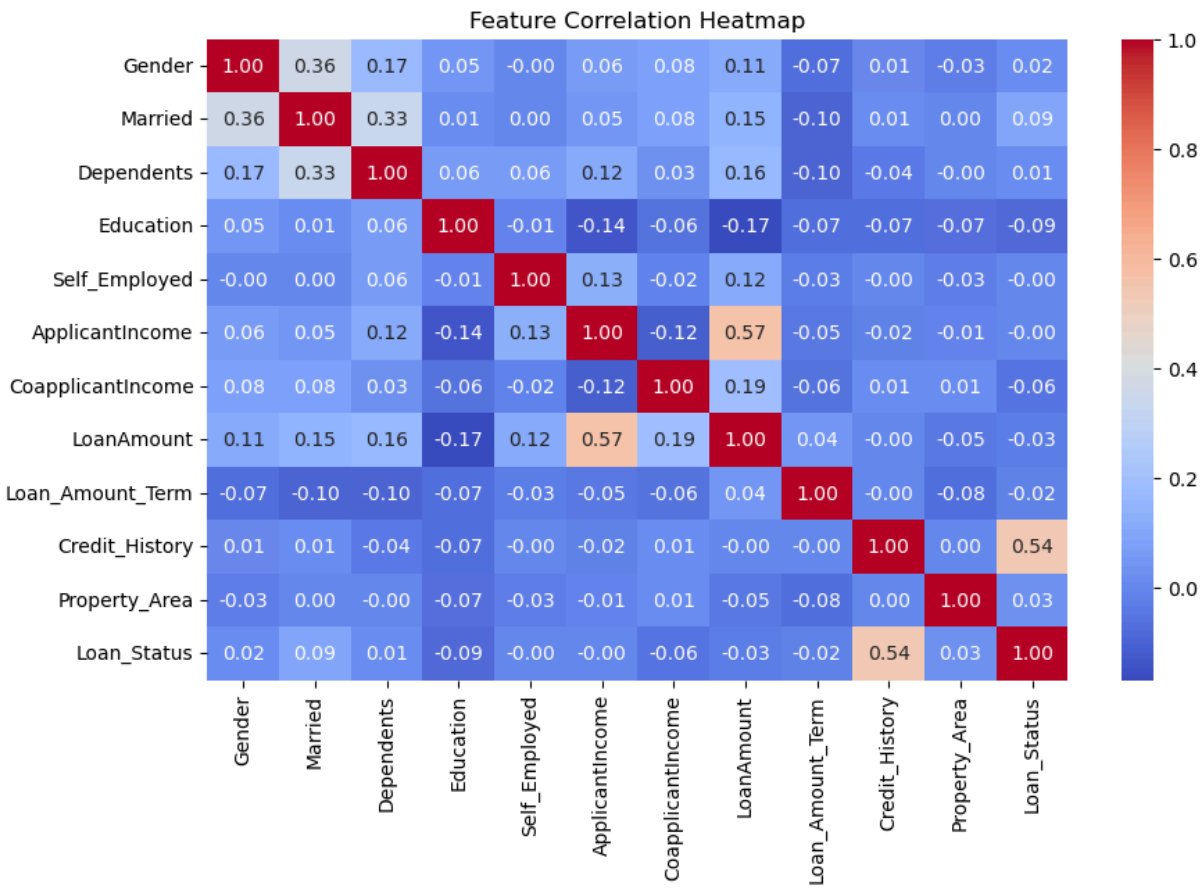
Credit History vs Loan Status.

In [18]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='Credit_History', hue='Loan_Status', data=data, palette='vir
plt.title("Credit History vs Loan Status")
plt.show()
```



Correlation Heatmap.

In [19]:
```
# Encode categorical variables temporarily for correlation
temp_df = data.copy()
le = LabelEncoder()
for col in temp_df.select_dtypes(include='object').columns:
    temp_df[col] = le.fit_transform(temp_df[col])

plt.figure(figsize=(10,6))
sns.heatmap(temp_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```



	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
Gender	1.00	0.36	0.17	0.05	-0.00	0.06	0.08	0.11	-0.07	0.01	-0.03	0.02
Married	0.36	1.00	0.33	0.01	0.00	0.05	0.08	0.15	-0.10	0.01	0.00	0.09
Dependents	0.17	0.33	1.00	0.06	0.06	0.12	0.03	0.16	-0.10	-0.04	-0.00	0.01
Education	0.05	0.01	0.06	1.00	-0.01	-0.14	-0.06	-0.17	-0.07	-0.07	-0.07	-0.09
Self_Employed	-0.00	0.00	0.06	-0.01	1.00	0.13	-0.02	0.12	-0.03	-0.00	-0.03	-0.00
ApplicantIncome	0.06	0.05	0.12	-0.14	0.13	1.00	-0.12	0.57	-0.05	-0.02	-0.01	-0.00
CoapplicantIncome	0.08	0.08	0.03	-0.06	-0.02	-0.12	1.00	0.19	-0.06	0.01	0.01	-0.06
LoanAmount	0.11	0.15	0.16	-0.17	0.12	0.57	0.19	1.00	0.04	-0.00	-0.05	-0.03
Loan_Amount_Term	-0.07	-0.10	-0.10	-0.07	-0.03	-0.05	-0.06	0.04	1.00	-0.00	-0.08	-0.02
Credit_History	0.01	0.01	-0.04	-0.07	-0.00	-0.02	0.01	-0.00	-0.00	1.00	0.00	0.54
Property_Area	-0.03	0.00	-0.00	-0.07	-0.03	-0.01	0.01	-0.05	-0.08	0.00	1.00	0.03
Loan_Status	0.02	0.09	0.01	-0.09	-0.00	-0.00	-0.06	-0.03	-0.02	0.54	0.03	1.00

Encode Categorical Variables.

In [20]:
```
# Encode categorical variables using LabelEncoder
le = LabelEncoder()
categorical_cols = ['Gender','Married','Dependents','Education','Self_Employ

for col in categorical_cols:
    data[col] = le.fit_transform(data[col])

# Check the first few rows after encoding
data.head()
```

Out[20]:

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	1	0	0	0	0	5849
1	1	1	1	0	0	4583
2	1	1	0	0	1	3000
3	1	1	0	1	0	2583
4	1	0	0	0	0	6000

Split Dataset into features and target.

In [21]:
```
# Target variable
y = data['Loan_Status']

# Feature variables
X = data.drop('Loan_Status', axis=1)

# Split into training and test sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (491, 11)
Test set shape: (123, 11)
```

Train Logistic Regression Model.

In [22]:
```
# Initialize Logistic Regression
model = LogisticRegression(max_iter=200)

# Train the model
model.fit(X_train, y_train)

# Make predictions on test set
y_pred = model.predict(X_test)
```

Evaluate the Model.

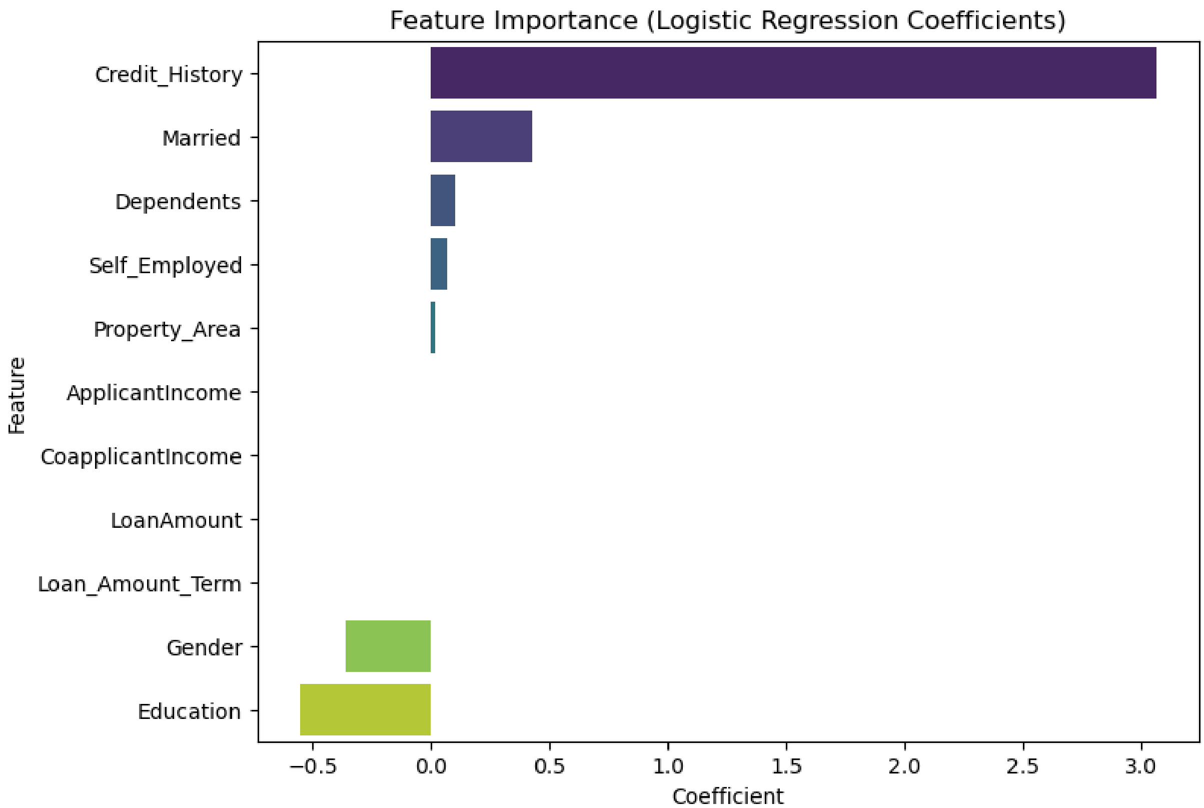
In [23]:
```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print("✅ Accuracy:", accuracy)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print("\n□ Confusion Matrix:\n", cm)
```

```
# Classification Report
cr = classification_report(y_test, y_pred)
print("\n Classification Report:\n", cr)
```

```
✅ Accuracy: 0.7886178861788617

 Confusion Matrix:
 [[18 25]
 [ 1 79]]

 Classification Report:
               precision    recall  f1-score   support

           0       0.95      0.42      0.58        43
           1       0.76      0.99      0.86        80

    accuracy                           0.79       123
   macro avg       0.85      0.70      0.72       123
weighted avg       0.83      0.79      0.76       123
```

Feature Importance.

In [24]:
```
# Logistic Regression coefficients
feature_importance = pd.DataFrame({'Feature': X.columns, 'Coefficient': mode
feature_importance = feature_importance.sort_values(by='Coefficient', ascend

# Plot
plt.figure(figsize=(8,6))
sns.barplot(x='Coefficient', y='Feature', data=feature_importance, palette='
plt.title("Feature Importance (Logistic Regression Coefficients)")
plt.show()
```



Predict Loan Status (User Input).

In [29]:

```
def predict_loan_status():
    print("\n□ Let's Predict the Loan Status!")
    Gender = input("Enter Gender (Male/Female): ")
    Married = input("Married (Yes/No): ")
    Dependents = input("Number of Dependents (0/1/2/3+): ")
    Education = input("Education (Graduate/Not Graduate): ")
    Self_Employed = input("Self Employed (Yes/No): ")
    ApplicantIncome = float(input("Applicant Income: "))
    CoapplicantIncome = float(input("Coapplicant Income: "))
    LoanAmount = float(input("Loan Amount: "))
    Loan_Amount_Term = float(input("Loan Amount Term: "))
    Credit_History = float(input("Credit History (1 or 0): "))
    Property_Area = input("Property Area (Urban/Rural/Semiurban): ")

    # Create a dataframe for the input
    user_data = pd.DataFrame({
        'Gender': [Gender],
        'Married': [Married],
        'Dependents': [Dependents],
        'Education': [Education],
        'Self_Employed': [Self_Employed],
        'ApplicantIncome': [ApplicantIncome],
        'CoapplicantIncome': [CoapplicantIncome],
        'LoanAmount': [LoanAmount],
        'Loan_Amount_Term': [Loan_Amount_Term],
        'Credit_History': [Credit_History],
```

```python
        'Property_Area': [Property_Area]
    })

    # Encode user data
    for col in ['Gender', 'Married', 'Education', 'Self_Employed', 'Property
        user_data[col] = le.fit_transform(user_data[col])

    # Make prediction
    prediction = model.predict(user_data)
    result = '✅ Loan Approved' if prediction[0] == 1 else '❌ Loan Rejected'
    print("\n□ Prediction Result:", result)

# Run the prediction function
predict_loan_status()
```

```
□ Let's Predict the Loan Status!
□ Prediction Result: ❌ Loan Rejected
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: