Loan_Eligibility_Predictions

November 26, 2023

1 Extract data from the source

[39]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import ( confusion_matrix, accuracy_score,␣
↪precision_score, recall_score, f1_score)
```

[2]:
```
loan_predictions = pd.read_csv('Loan Eligibility Predictions.csv')
```

[3]:
```
loan_predictions.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
```

[4]:
```
loan_predictions.head()
```

[4]:
```
    Loan_ID Gender Married Dependents     Education Self_Employed  \
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

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0             5849                0.0         NaN             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

   Credit_History Property_Area Loan_Status
0             1.0         Urban           Y
1             1.0         Rural           N
2             1.0         Urban           Y
3             1.0         Urban           Y
4             1.0         Urban           Y
```

2 Exploratory Data Analysis (EDA)

2.0.1 Identify descriptive statistics on variables/features

[5]:
```python
# Descriptive statistics for numerical columns
loan_predictions.describe()
```

[5]:
```
       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

       Credit_History
count      564.000000
mean         0.842199
std          0.364878
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

[6]:
```python
# Count for categorical columns
categorical_columns = ['Gender', 'Married', 'Dependents', 'Education',␣
↪'Self_Employed', 'Credit_History', 'Property_Area', 'Loan_Status']

for column in categorical_columns:
    print(loan_predictions[column].value_counts())
    print("\n")
```

```
Gender
Male      489
Female    112
Name: count, dtype: int64



Married
Yes    398
No     213
Name: count, dtype: int64



Dependents
0     345
1     102
2     101
3+     51
Name: count, dtype: int64



Education
Graduate        480
Not Graduate    134
Name: count, dtype: int64



Self_Employed
No     500
Yes     82
Name: count, dtype: int64



Credit_History
1.0    475
0.0     89
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
```

[7]:
```python
loan_predictions.select_dtypes(include=['object']).describe()
```

[7]:
```
        Loan_ID Gender Married Dependents Education Self_Employed  \
count       614    601     611        599       614           582
unique      614      2       2          4         2             2
top    LP001002   Male     Yes          0  Graduate            No
freq          1    489     398        345       480           500

       Property_Area Loan_Status
count            614         614
unique             3           2
top        Semiurban           Y
freq             233         422
```

2.0.2 Determine correlation between variables/features

[8]:
```python
# Determine correlation between variables/features (R2)
numeric_columns = loan_predictions.select_dtypes(include=[np.number])
comatrix = numeric_columns.corr()
comatrix
```

[8]:
```
                   ApplicantIncome  CoapplicantIncome  LoanAmount  \
ApplicantIncome           1.000000          -0.116605    0.570909
CoapplicantIncome        -0.116605           1.000000    0.188619
LoanAmount                0.570909           0.188619    1.000000
Loan_Amount_Term         -0.045306          -0.059878    0.039447
Credit_History           -0.014715          -0.002056   -0.008433

                   Loan_Amount_Term  Credit_History
ApplicantIncome           -0.045306       -0.014715
CoapplicantIncome         -0.059878       -0.002056
LoanAmount                 0.039447       -0.008433
Loan_Amount_Term           1.000000        0.001470
Credit_History             0.001470        1.000000
```

2.0.3 Identify and handle NULL values

[9]:
```
# Check for missing values in the entire DataFrame
loan_predictions.isnull().sum()
```

[9]:
```
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
```

[10]:
```
loan_predictions.fillna({'Gender': 'Male', 'Married': 'Yes', 'Dependents': '0',␣
↪'Self_Employed': 'No', 'LoanAmount': loan_predictions['LoanAmount'].
↪median(), 'Loan_Amount_Term': loan_predictions['Loan_Amount_Term'].
↪mode()[0], 'Credit_History': 1}, inplace=True)
loan_predictions.isnull().sum()
```

[10]:
```
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
```

2.0.4 Identify any outliers

[11]:
```
# Filter to select only numeric columns
numeric_columns = loan_predictions.select_dtypes(include=np.number)

# Calculate IQR for each numeric column
Q1 = numeric_columns.quantile(0.25)
```

```
Q3 = numeric_columns.quantile(0.75)
IQR = Q3 - Q1

# Identify potential outliers using the IQR
outliers = (numeric_columns < (Q1 - 1.5 * IQR)) | (numeric_columns > (Q3 + 1.5␣
↪* IQR))
outliers.sum()
```

[11]:
```
ApplicantIncome      50
CoapplicantIncome    18
LoanAmount           41
Loan_Amount_Term     88
Credit_History       89
dtype: int64
```

2.0.5 Handling outliers

[12]:
```
# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Replace outliers with the nearest non-outlier value
for column in numeric_columns.columns:
    numeric_columns.loc[numeric_columns[column] < lower_bound[column], column]␣
↪= lower_bound[column]
    numeric_columns.loc[numeric_columns[column] > upper_bound[column], column]␣
↪= upper_bound[column]
```

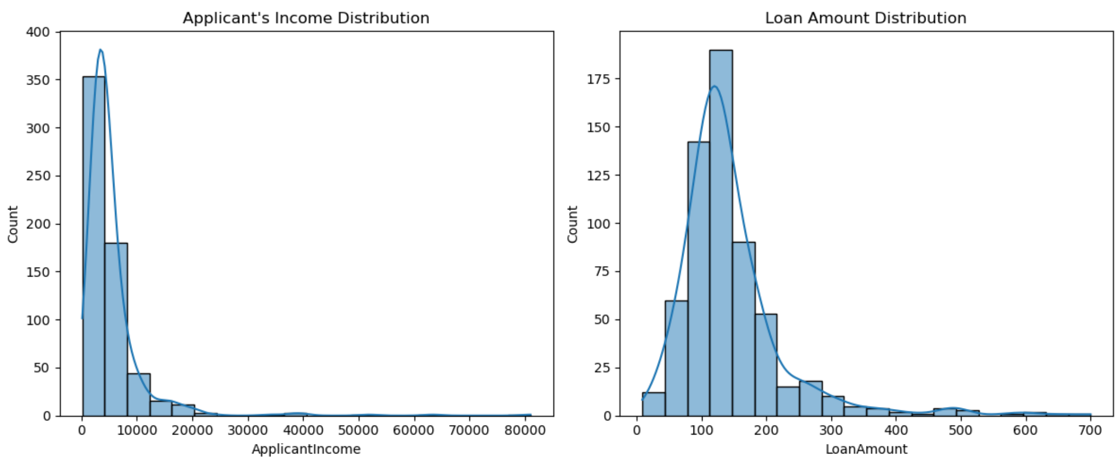
[13]:
```
outliers_after_handling = (numeric_columns < (Q1 - 1.5 * IQR)) |␣
↪(numeric_columns > (Q3 + 1.5 * IQR))
outliers_count = outliers_after_handling.sum()
outliers_count
```

[13]:
```
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
dtype: int64
```

3 Visualizations

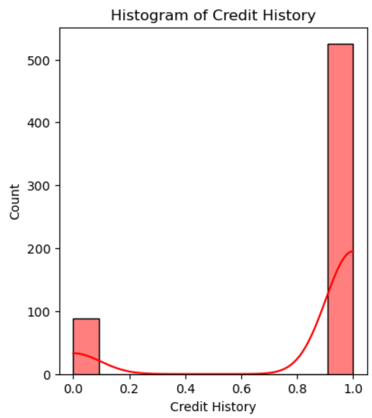
3.0.1 Histograms of relevant variables

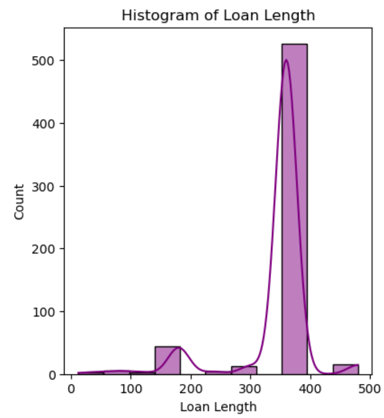
[14]:
```
# Histograms of applicants income and loan amount
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

sns.histplot(loan_predictions['ApplicantIncome'], bins=20, kde=True, ax=axes[0])
axes[0].set_title('Applicant\'s Income Distribution')

sns.histplot(loan_predictions['LoanAmount'], bins=20, kde=True, ax=axes[1])
axes[1].set_title('Loan Amount Distribution')

plt.tight_layout()
plt.show()
```



[15]:
```
# Histogram of credit history and loan length
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
sns.histplot(data=loan_predictions, x='Credit_History', kde=True, color='red')
plt.title("Histogram of Credit History")
plt.xlabel("Credit History")

plt.subplot(1, 3, 3)
sns.histplot(data=loan_predictions, x='Loan_Amount_Term', kde=True,
    color='purple')
plt.title("Histogram of Loan Length")
plt.xlabel("Loan Length")
```

[15]: Text(0.5, 0, 'Loan Length')

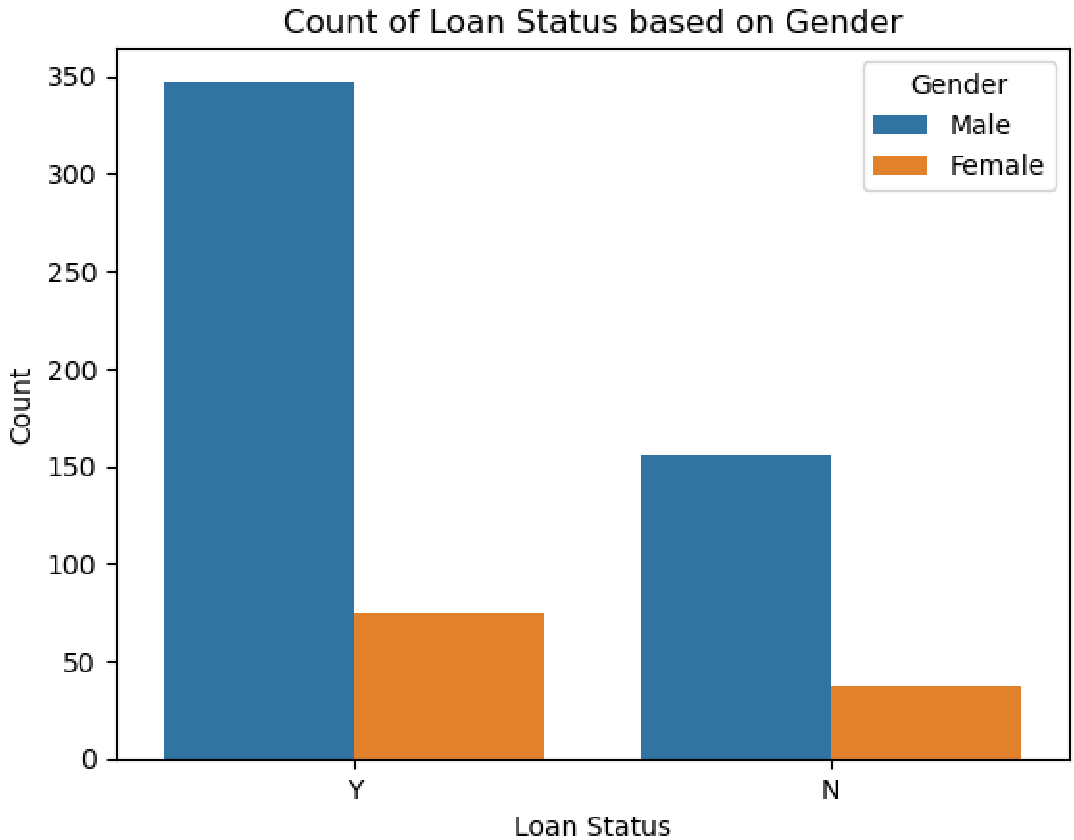




3.0.2 Different types of plots

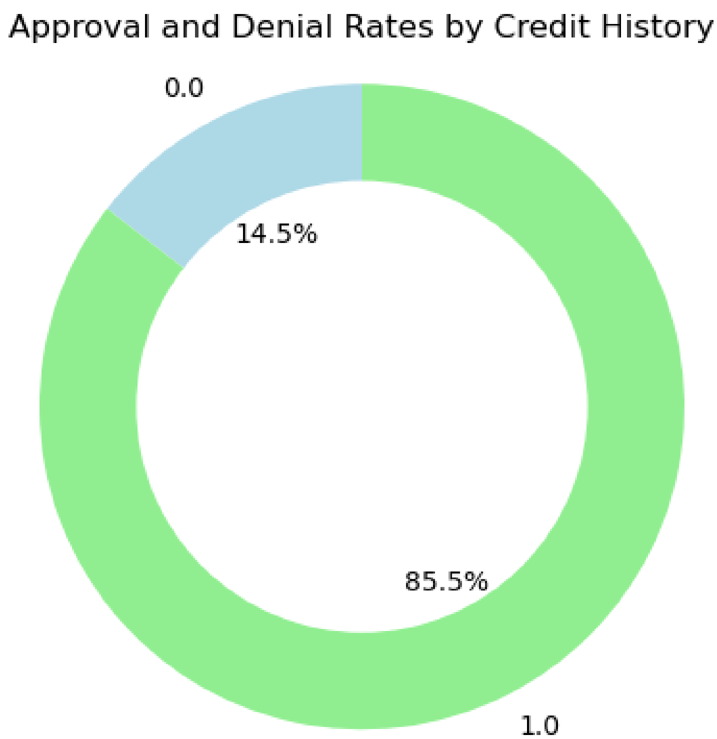
Loan Status with respect to Gender

[16]:
```
# Count plot for 'Loan_Status' with respect to 'Gender'
sns.countplot(data=loan_predictions, x='Loan_Status', hue='Gender')
plt.title('Count of Loan Status based on Gender')
plt.xlabel('Loan Status')
plt.ylabel('Count')
plt.show()
```



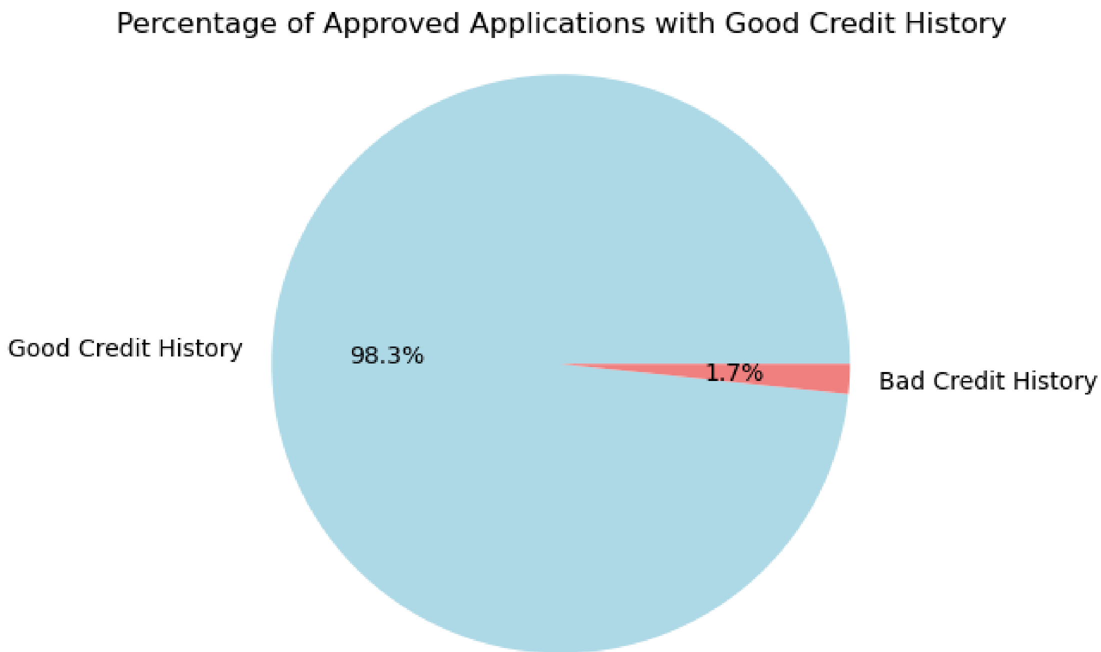
Approval and Denial rates based on credit history

[17]:
```
# Calculate approval and denial counts based on credit history
credit_history_counts = loan_predictions.groupby(['Credit_History',␣
↪'Loan_Status']).size().unstack()

# Calculate the total counts of approvals and denials
total_counts = credit_history_counts.sum(axis=1)

# Plotting the donut chart
fig, ax = plt.subplots()
ax.pie(total_counts, labels=total_counts.index, autopct='%1.1f%%',␣
↪startangle=90, colors=['lightblue', 'lightgreen'])
centre_circle = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
ax.axis('equal')
plt.title('Approval and Denial Rates by Credit History')
plt.show()
```



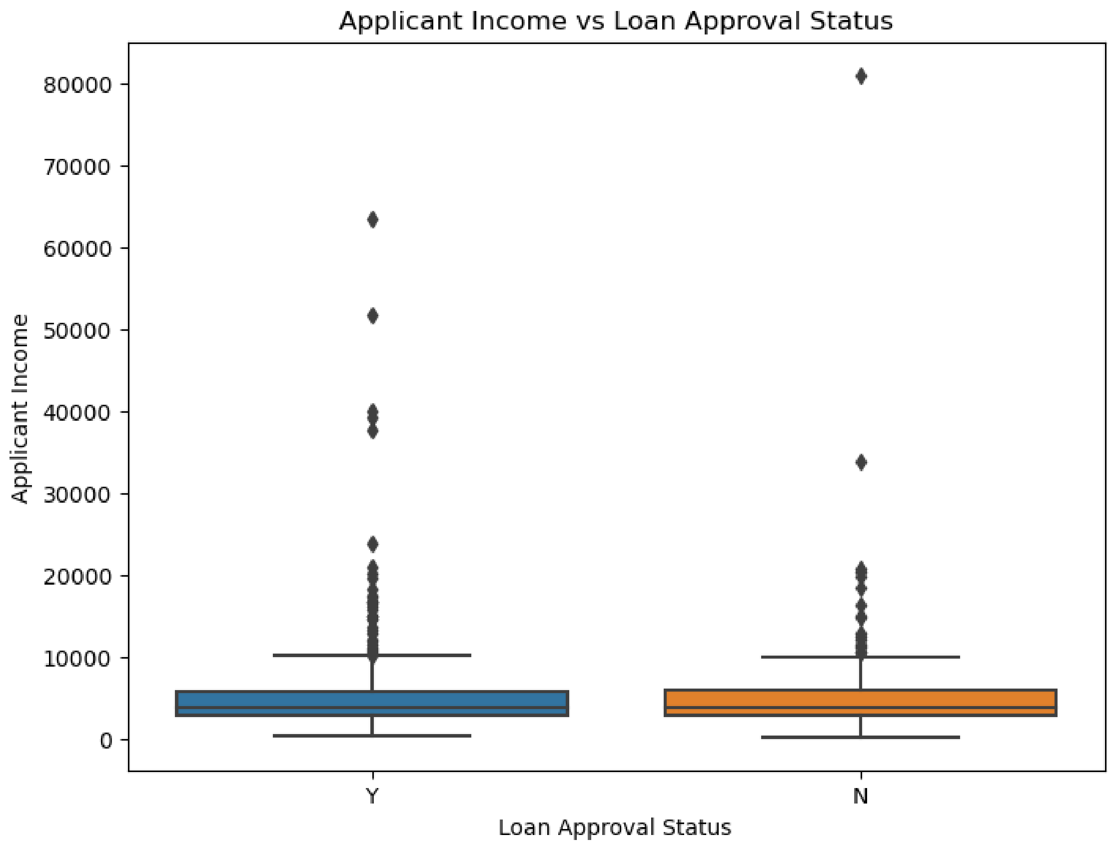
What percentage of approved applications have a good credit history?

[18]:
```
# Calculate the number of approved applications
approved_applications = loan_predictions[loan_predictions['Loan_Status'] == 'Y']

# Calculate the percentage of approved applications with a good credit history
good_credit_approved =␣
↪approved_applications[approved_applications['Credit_History'] == 1]
percentage_good_credit_approved = (len(good_credit_approved) /␣
↪len(approved_applications)) * 100

# Calculate the percentage of approved applications with a bad credit history
percentage_bad_credit_approved = 100 - percentage_good_credit_approved

# Plotting
plt.pie([percentage_good_credit_approved, percentage_bad_credit_approved],␣
↪labels=['Good Credit History', 'Bad Credit History'], colors=['lightblue',␣
↪'lightcoral'], autopct='%1.1f%%')
plt.axis('equal')
plt.title('Percentage of Approved Applications with Good Credit History')
plt.show()
```



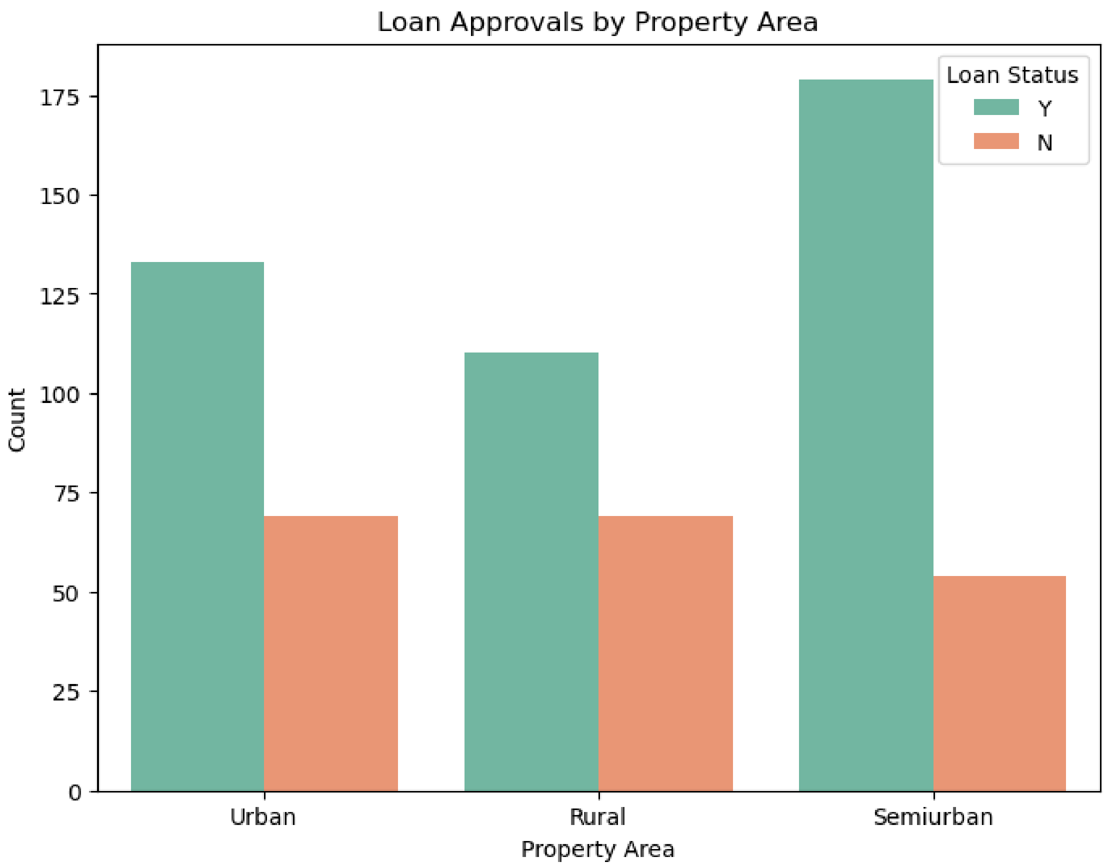
Are there income thresholds that significantly impact loan approval?

[19]:
```
# Create a boxplot of ApplicantIncome for approved and denied loans
plt.figure(figsize=(8, 6))
sns.boxplot(x='Loan_Status', y='ApplicantIncome', data=loan_predictions)
plt.title('Applicant Income vs Loan Approval Status')
plt.xlabel('Loan Approval Status')
plt.ylabel('Applicant Income')
plt.show()
```



Loan approval by property area

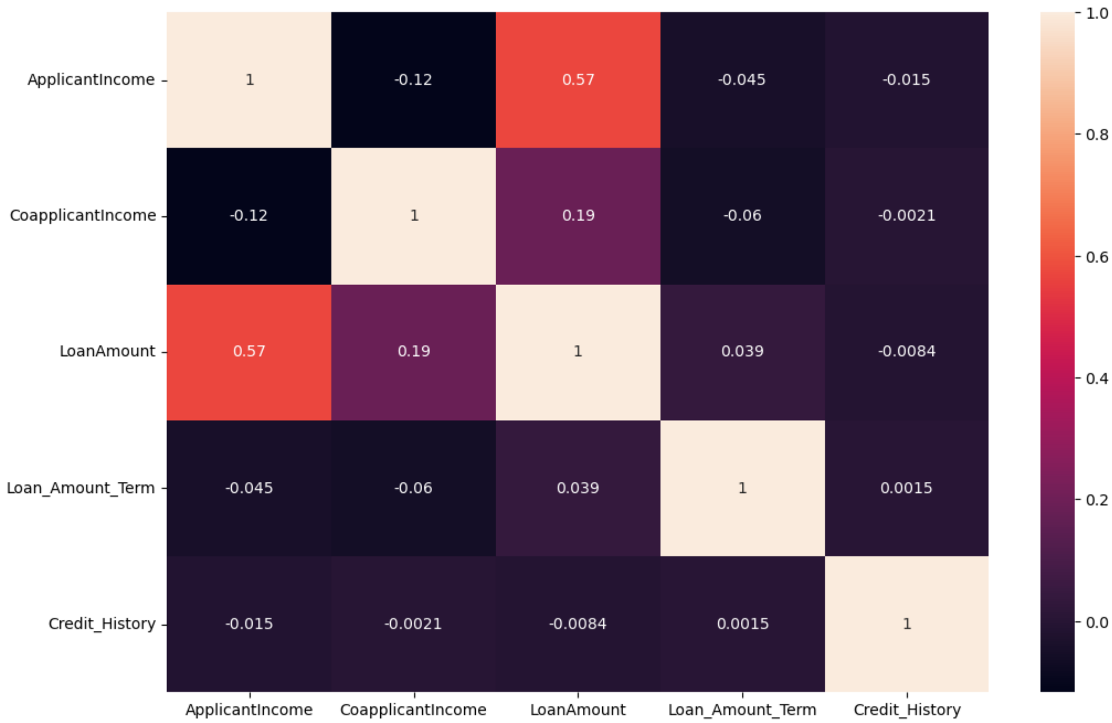
[20]:
```
# Plotting loan approvals by Property_Area
plt.figure(figsize=(8, 6))
sns.countplot(x='Property_Area', hue='Loan_Status', data=loan_predictions,␣
 ↪palette='Set2')
plt.title('Loan Approvals by Property Area')
plt.xlabel('Property Area')
plt.ylabel('Count')
plt.legend(title='Loan Status', loc='upper right')
plt.show()
```



Correlations between variables

[33]:
```
# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(comatrix, annot=True)
```

[33]: <Axes: >



4 Model Development using KNN

[34]:
```
X = loan_predictions[['Credit_History', 'ApplicantIncome', 'CoapplicantIncome',␣
↪'LoanAmount', 'Loan_Amount_Term']]
y = loan_predictions['Loan_Status']
```

[35]:
```
y = y.map({'N': 0, 'Y': 1})
```

[36]:
```
# Step 1: Split the data into training and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[37]:
```
# Step 2: Standardize data to the same scale
standardizer = StandardScaler()
x_standardized = standardizer.fit_transform(X_train)
x_test_std = standardizer.fit_transform(X_test)
```

[40]:
```
# Step 3: Create model (fit the training data)
knn = KNeighborsClassifier(n_neighbors=17).fit(x_standardized, y_train)
```

[41]:
```
# Step 4: Use model to predict the class of test data
y_predicted= knn.predict(x_test_std)
```

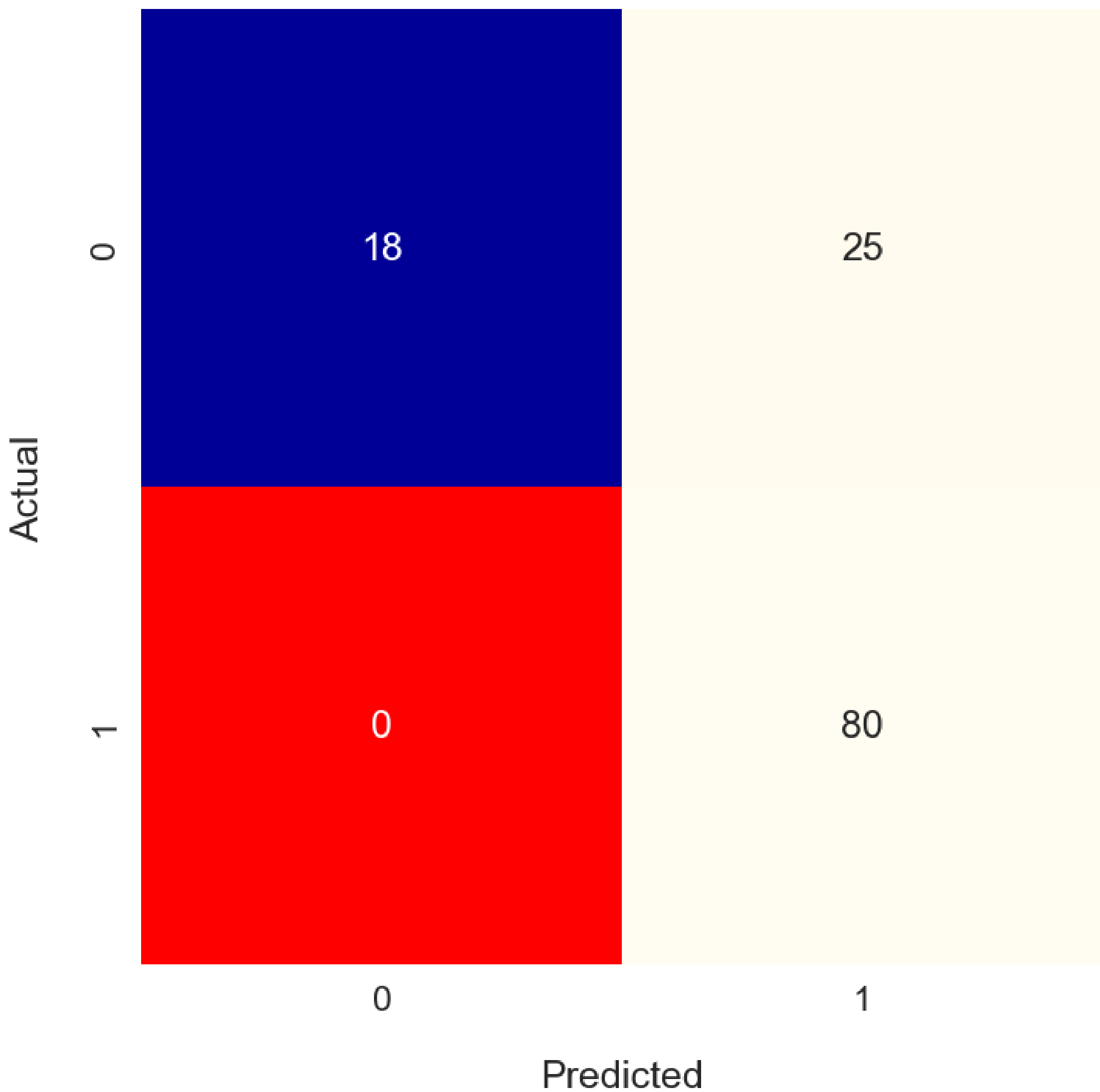
[42]:
```
# Step 5: Model Evaluation
print(accuracy_score(y_test, y_predicted))
print(precision_score(y_test, y_predicted, average= 'macro'))
print(recall_score(y_test, y_predicted, average= 'macro'))
print(f1_score(y_test, y_predicted, average= 'macro'))
```

```
0.7967479674796748
0.8809523809523809
0.7093023255813954
0.7275143996455471
```

[43]:
```
# Step 6: Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_predicted)
conf_matrix
```

[43]:
```
array([[18, 25],
       [ 0, 80]], dtype=int64)
```

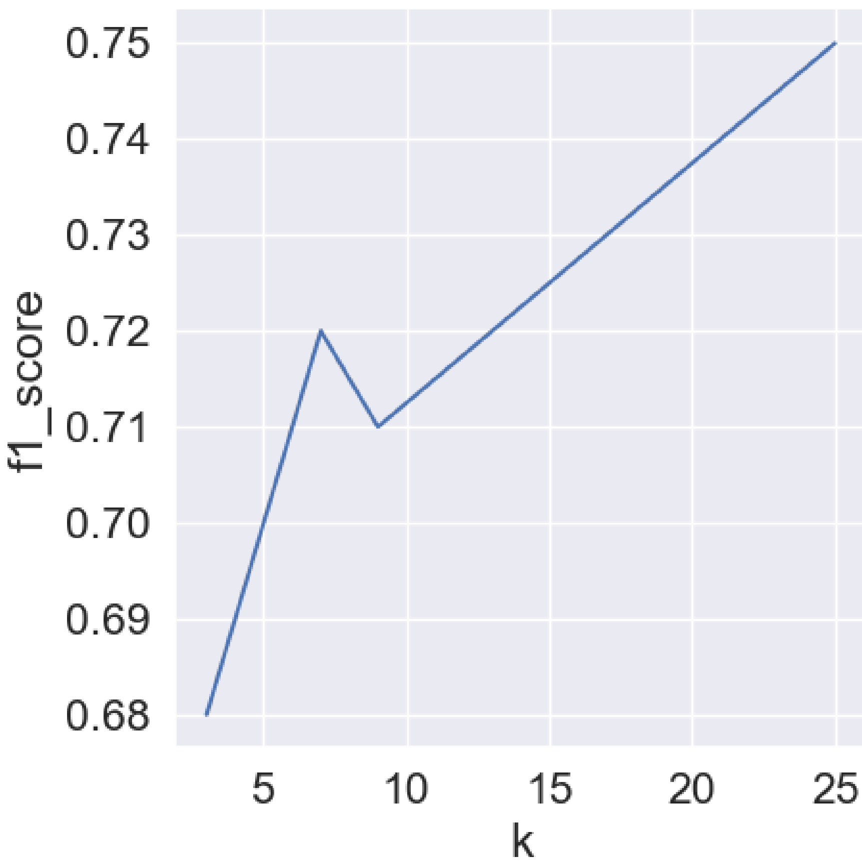
[44]:
```
# Plot confusion matrix
plt.figure(figsize=(8,8))
sns.set(font_scale = 1.5)
ax = sns.heatmap(
conf_matrix,
annot=True,
fmt='d',
cbar=False,
cmap='flag',
vmax=175
)
ax.set_xlabel("Predicted", labelpad=20)
ax.set_ylabel("Actual", labelpad=20)
plt.show()
```



[45]:
```
plot_data= {'k' : [9,25,3,5,7], 'f1_score': [.71, .75, .68, .70, .72]}
sns.relplot(data=plot_data, kind='line', x='k', y='f1_score')
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:
The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

[45]: <seaborn.axisgrid.FacetGrid at 0x1c6c036a590>



[ ]:
```
# Bend is observed at 9, Optimal K is 9
```