

Dataset URL

- <https://www.kaggle.com/datasets/rishikeshkonapure/home-loan-approval>
- Please note when you download the dataset, you will find 2 files "train" and "test" datasets. I used in this assignment the train dataset "loan_sanction_train.csv" but I renamed it to be loan_sanction_data.csv

```
In [1]: # --- Imports and Setup ---
from IPython.display import Image, display
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
In [2]: ### Load Dataset #####
df = pd.read_csv('loan_sanction_data.csv')
```

```
In [4]: ### Show the first 5th rows
df.head()
```

```
Out[4]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIn
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	

```
In [5]: # --- Check feature types ---
df.dtypes
```

```
Out[5]: Loan_ID          object
Gender          object
Married         object
Dependents      object
Education       object
Self_Employed  object
ApplicantIncome    int64
CoapplicantIncome float64
LoanAmount       float64
Loan_Amount_Term float64
Credit_History   float64
Property_Area    object
Loan_Status      object
dtype: object
```

```
In [6]: # --- Print all column names ---
df.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]: # --- Count missing values ---
df.isnull().sum()
```

```
Out[7]: 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 [8]: missing_percentage = df.isnull().mean() * 100
```

```
In [9]: print(missing_percentage)
```

```
Loan_ID          0.000000
Gender           2.117264
Married          0.488599
Dependents       2.442997
Education        0.000000
Self_Employed    5.211726
ApplicantIncome  0.000000
CoapplicantIncome 0.000000
LoanAmount       3.583062
Loan_Amount_Term 2.280130
Credit_History   8.143322
Property_Area    0.000000
Loan_Status      0.000000
dtype: float64
```

```
In [14]: ##### Note #####
#         I tottaly understand these lines of code and I did a test by setti
#         and I got thess deleted colums (Columns dropped: ['Self_Employed',

threshold = 30
columns_to_drop = missing_percentage[missing_percentage > threshold].index
df = df.drop(columns=columns_to_drop)

print(f"Columns dropped: {list(columns_to_drop)}"+"\\n The dataset I am us
print(f"Remaining columns: {df.columns}")
```

Columns dropped: []

The dataset I am using doesn't have any feature with more than 30% missing data

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

```
In [15]: # Encode categorical columns using LabelEncoder
# this part to change the alphabet to number ex: Yes , No to 1 , 0
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

for column in df.select_dtypes(include=['object']).columns:
    df[column] = label_encoder.fit_transform(df[column])
```

```
In [16]: # print the df to make sure LabelEncoder is work or not
df.head()
```

```
Out[16]:
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantInc
0	0	1	0	0	0	0	5
1	1	1	1	1	0	0	4
2	2	1	1	0	0	1	3
3	3	1	1	0	1	0	2
4	4	1	0	0	0	0	6

```
In [17]: # Fill missing values based on type
# Fill numerical columns with mean and categorical with mode

for column in df.select_dtypes(include=['float64']).columns:
    mean_value = df[column].mean()
    df[column] = df[column].fillna(mean_value)

for column in df.select_dtypes(include=['object']).columns:
    mode_value = df[column].mode()[0]
    df[column] = df[column].fillna(mode_value)
```

```
In [61]: df.describe()
```

Out [61]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employ
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.0000
mean	306.500000	0.838762	0.657980	0.842020	0.218241	0.2377
std	177.390811	0.421752	0.484971	1.120531	0.413389	0.5347
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	153.250000	1.000000	0.000000	0.000000	0.000000	0.0000
50%	306.500000	1.000000	1.000000	0.000000	0.000000	0.0000
75%	459.750000	1.000000	1.000000	2.000000	0.000000	0.0000
max	613.000000	2.000000	2.000000	4.000000	1.000000	2.0000

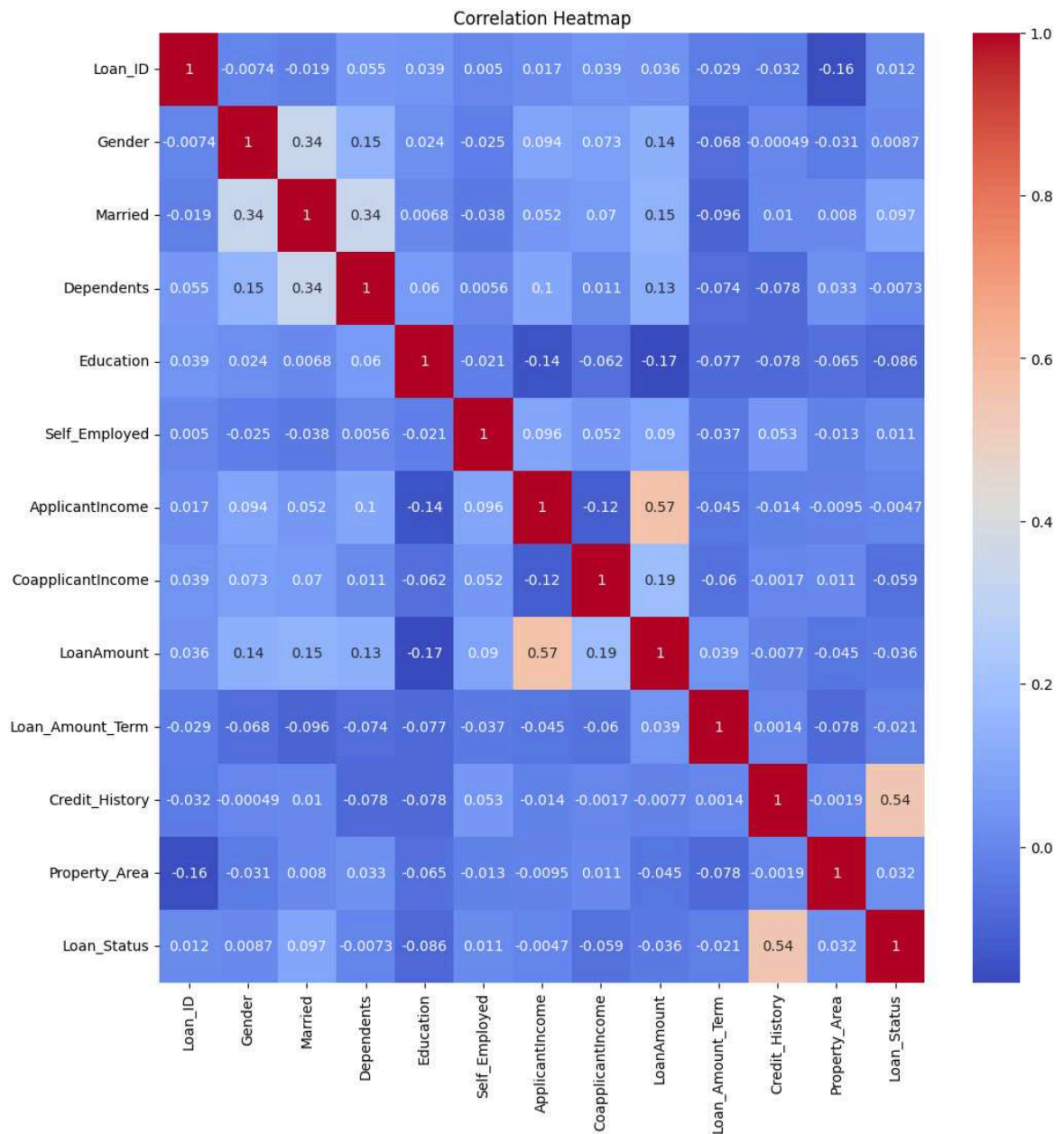
In [60]: `df.isnull().sum()`

Out [60]:

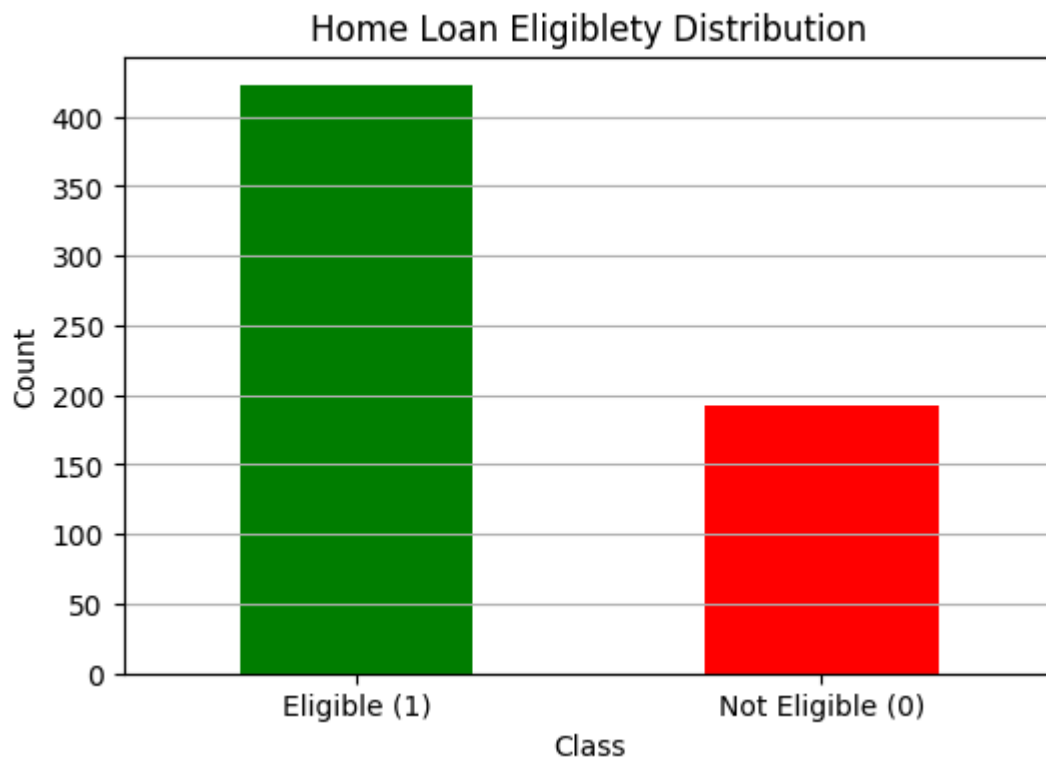
Loan_ID	0
Gender	0
Married	0
Dependents	0
Education	0
Self_Employed	0
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	0
Loan_Amount_Term	0
Credit_History	0
Property_Area	0
Loan_Status	0
dtype:	int64

In [23]:

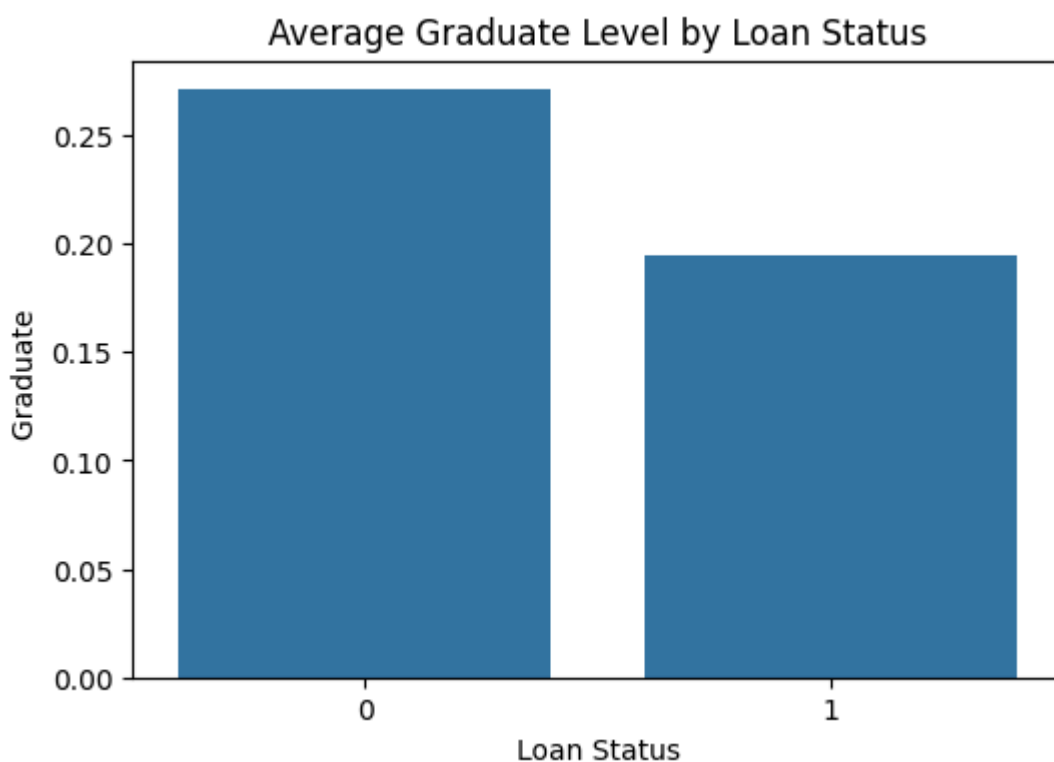
```
# Correlation Heatmap
plt.figure(figsize=(12,12)) ## this is the size of the figure not related
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```
In [27]: # Distribution of Home loan Classification
plt.figure(figsize=(6, 4))
df['Loan_Status'].value_counts().plot(kind='bar', color=['green', 'red'])
plt.xticks([0, 1], ['Eligible (1)', 'Not Eligible (0)'], rotation=0)
plt.title('Home Loan Eligibility Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.grid(axis='y')
plt.show()
```

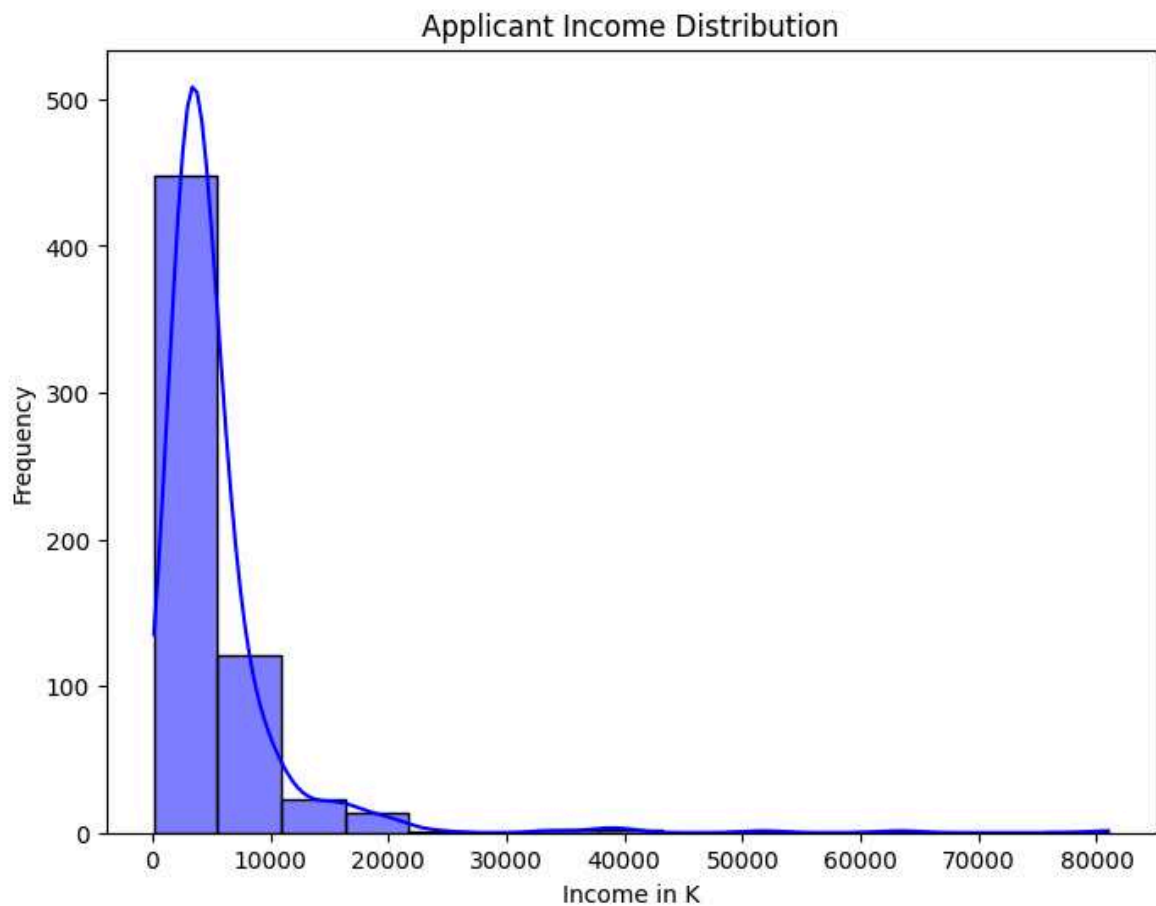


```
In [31]: # Average Hemoglobin by CKD Status
plt.figure(figsize=(6, 4))
sns.barplot(x='Loan_Status', y='Education', data=df, estimator='mean', ci
plt.title("Average Graduate Level by Loan Status")
plt.xlabel("Loan Status")
plt.ylabel("Graduate ")
plt.show()
```

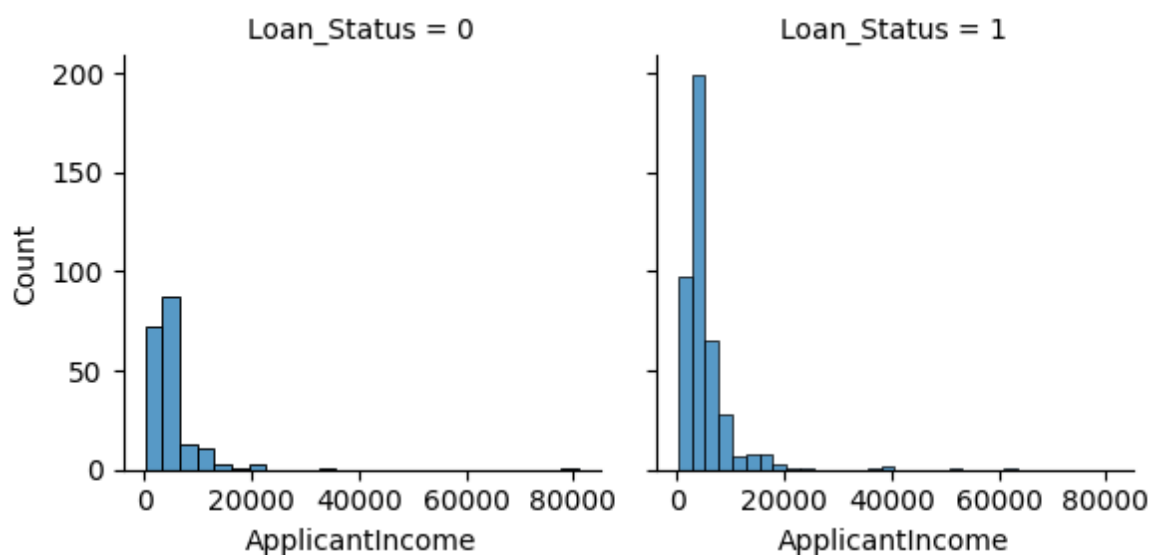


```
In [34]: # Applicant Income Distribution
plt.figure(figsize=(8, 6))
sns.histplot(df['ApplicantIncome'], bins=15, kde=True, color='blue')
plt.title('Applicant Income Distribution')
```

```
plt.xlabel('Income in K')
plt.ylabel('Frequency')
plt.show()
```



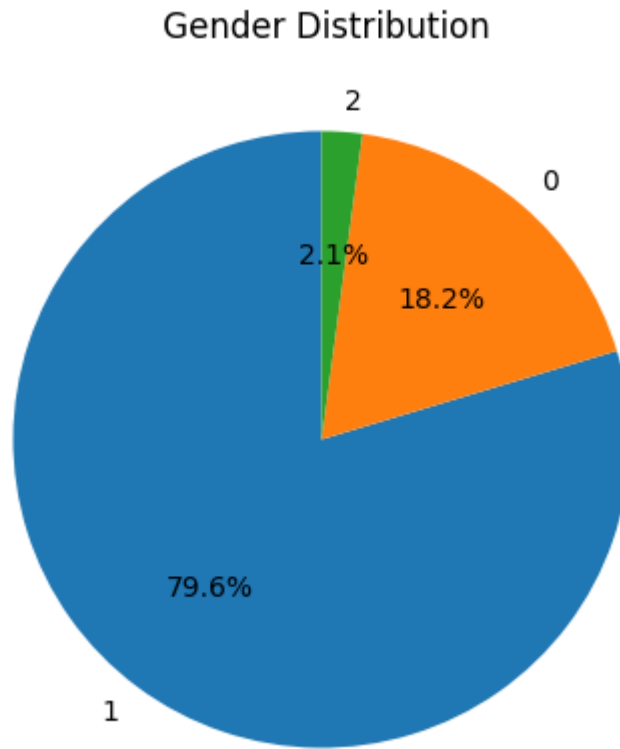
```
In [36]: g = sns.FacetGrid(df, col="Loan_Status")
g.map(sns.histplot, "ApplicantIncome", bins=25)
plt.show()
```



```
In [53]: # Gender distribution (Pie Chart)
gender_counts = df['Gender'].value_counts()
plt.figure(figsize=(5, 5))
plt.pie(gender_counts, labels=gender_counts.index, autopct='%1.1f%%', sta
plt.title(" Gender Distribution")
```

```
print(gender_counts.index)
plt.show()
```

```
Index([1, 0, 2], dtype='int64', name='Gender')
```



```
In [55]: # Crosstab Bar Chart
ax = pd.crosstab(df.Gender, df.Loan_Status).plot(
    kind='bar', figsize=(8, 5),
    color=['skyblue', 'lightcoral'], edgecolor='black', alpha=0.85
)
for rect in ax.patches:
    height = rect.get_height()
    ax.text(rect.get_x() + rect.get_width() / 2, height + 1, int(height),

plt.suptitle('Home Loan Distribution by Gender', fontsize=16, fontweight=
plt.xlabel('Gender (Sex)')
plt.ylabel('Total')
plt.xticks([0, 1, 2], ['Female', 'Male', 'Other'])
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.legend(labels=['Not Eligible', 'Eligible'], title='Target')
plt.tight_layout()
plt.show()
```

