Import libraries and read the file

In [1]:

```
# import libraries

import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
# Read the census data
census_data = pd.read_csv('census_data.csv')
```

Study the dataset

In [3]:

```
#Check the number of rows and columns
census_data.shape
```

Out[3]:

```
(8646, 11)
```

In [4]:

```
# print out the first five rows to have an overview of the dataset
census_data.head()
```

Out[4]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
0	1	George Avenue	Harry	James	60	Head	Single	Male	Unemployed	None	None
1	2	George Avenue	Anne	Johnson	34	Head	Married	Female	Corporate treasurer	None	None
2	2	George Avenue	Jack	Johnson	36	Husband	Married	Male	Product/process development scientist	None	None
3	2	George Avenue	Guy	Johnson	12	Son	NaN	Male	Student	None	NaN
4	3	George Avenue	Simon	Smith	79	Head	Single	Male	Retired Tour manager	Physical Disability	Jewish

In [5]:

```
# Displays the data type, non null count and number of data entry
census_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8646 entries, 0 to 8645
Data columns (total 11 columns):
 #   Column                         Non-Null Count  Dtype 
---  ------                         --------------  ----- 
 0   House Number                   8646 non-null   int64 
 1   Street                         8646 non-null   object
 2   First Name                     8646 non-null   object
 3   Surname                        8646 non-null   object
 4   Age                            8646 non-null   object
 5   Relationship to Head of House  8646 non-null   object
 6   Marital Status                 6419 non-null   object
 7   Gender                         8646 non-null   object
 8   Occupation                     8646 non-null   object
 9   Infirmity                      8646 non-null   object
 10  Religion                       6373 non-null   object
dtypes: int64(1), object(10)
memory usage: 743.1+ KB
```

In [6]:

```python
# Check the total number of missing values
census_data.isna().sum()
```

Out[6]:

```
House Number                      0
Street                            0
First Name                        0
Surname                           0
Age                               0
Relationship to Head of House     0
Marital Status                 2227
Gender                            0
Occupation                        0
Infirmity                         0
Religion                       2273
dtype: int64
```

In [7]:

```python
# Check for duplicate
duplicate = census_data.duplicated()
census_data[duplicate]
```

Out[7]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
7309	1	Leedsbox Crescent	Ashleigh	Osborne	15	Daughter	NaN	Female	Student	None	NaN

In [8]:

```python
# Drop the duplicate
census_data = census_data.drop(7309)
```

Data Cleaning

Age

In [9]:

```python
# Check the unique entries for age
print(census_data['Age'].unique())
```

```
['60' '34' '36' '12' '79' '35' '61' '24' '3' '75' '52' '14' '11' '42' '25'
 '28' '40' '57' '55' '22' '18' '43' '51' '0' '21' '45' '17' '16' '13' '9'
 '65' '32' '31' '8' '56' '39' '7' '41' '27' '78' '30' '29' '15' '54' '19'
 '84' '38' '33' '6' '1' '48' '10' '5' '49' '46' '26' '50' '53' '63' '4'
 '44' '47' '2' '23' '64' '37' '58' '66' '67' '71' '72' '20' '62' '68' '73'
 '74' '69' '81' '70' '59' '89' '105' '87' '80' '77' '76' ' ' '82' '88'
 '49.16040882016717' '54.16040882016717' '3.0' '85' '99' '101' '83'
 '69.13036593215614' '67.13036593215614' '103' '90' '93' '86' '96'
 '85.66111048772531' '87.66111048772531' '34.0' '30.0' '26.0' '91' '102'
 '83.52432893335205' '26.999999999999993' '23.999999999999993'
 '21.999999999999993' '16.999999999999993' '92' '97' '69.13473801820774'
 '15.000000000000007' '13.000000000000007' '10.000000000000007' '98'
 '50.53760781824045' '53.53760781824045' '0.0']
```

In [10]:

```python
# Check for blank entries in Age
census_data[census_data['Age'] == ' ']
```

Out[10]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
460	18	Smith Gateway	Dominic	Griffiths		Son	NaN	Male	Student	None	NaN

In [11]:

```python
# Drops the line, since it is just a row, and we cannot predict the age of this person
census_data = census_data.drop(460)
```

In [12]:

```python
# Convert the Age column to integer
census_data['Age'] = census_data['Age'].astype(float).round(0).astype(int)
```

Religion

In [13]:

```python
# Check the unique entries in Religion
print(census_data['Religion'].unique())
```

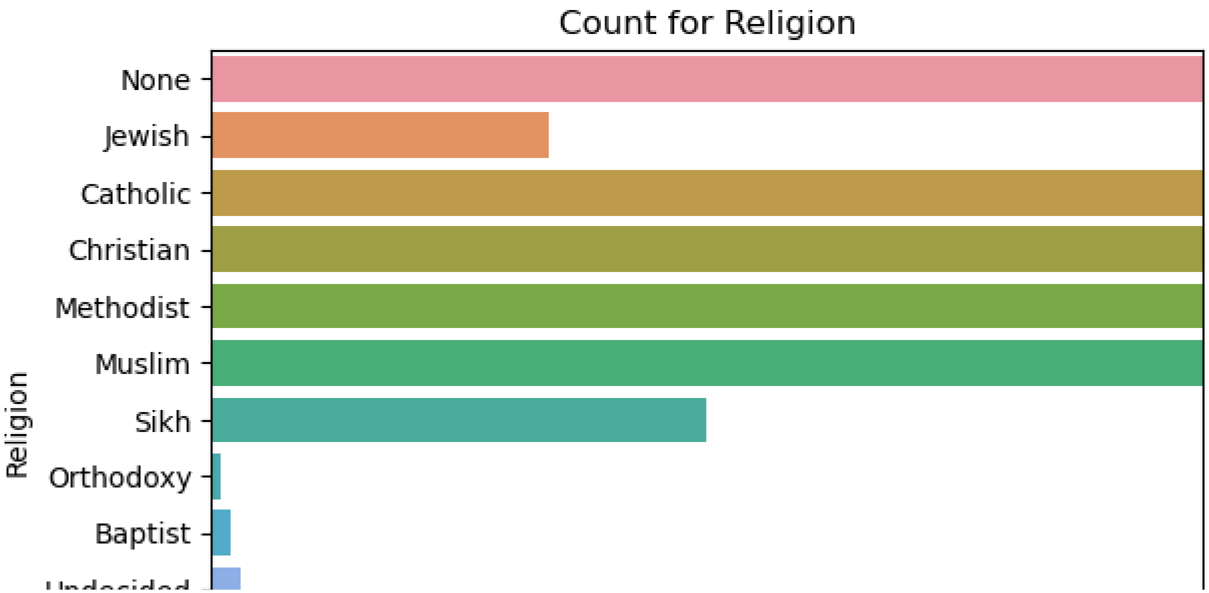
```
['None' nan 'Jewish' 'Catholic' 'Christian' 'Methodist' 'Muslim' 'Sikh'
 'Orthodoxy' 'Baptist' 'Undecided' 'Buddist' ' ' 'Nope']
```

In [14]:

```
# Countplot for religion to visualise the data
religion_bar = sns.countplot(census_data, y = census_data['Religion'])

# sets the x-axis limit from 0-100, to easily visualised the entries with small values.
plt.xlim(0,100)

# Title of the plot
religion_bar.set(title = 'Count for Religion')
```

Out[14]:

```
[Text(0.5, 1.0, 'Count for Religion')]
```



In [15]:

```python
# Check for blank entries in Religion
census_data[census_data['Religion'] == ' ']
```

Out[15]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
7069	4	Parsons Stream	Neil	Hall	34	Husband	Married	Male	Mining engineer	None	
7809	1	Cox Drive	Valerie	Arnold	57	Head	Single	Female	Unemployed	None	
8385	57	George Lane	Debra	Davies	31	Head	Married	Female	Barista	None	
8433	67	George Lane	Ashleigh	Martin	38	Lodger	Single	Female	Minerals surveyor	None	

In [16]:

```python
# Check the range of each entry to find a family tie
census_data[7806:7810]
```

Out[16]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
7808	4	Kelly Mountain	Denise	Thompson	10	Daughter	NaN	Female	Student	None	NaN
7809	1	Cox Drive	Valerie	Arnold	57	Head	Single	Female	Unemployed	None	
7810	1	Cox Drive	Katie	Arnold	23	Daughter	Single	Female	Health promotion specialist	None	Christian
7811	2	Cox Drive	Kyle	Perkins	72	Head	Widowed	Male	Retired Film/video editor	None	Christian

In [17]:

```python
# Replace row 7809 with Christian since the daughter in row 7810 is a Christian
census_data.loc[7809,'Religion'] = 'Christian'
```

In [18]:

```python
# Replace the remaining blanks to 'None' since they are adult
census_data['Religion'].replace(' ', 'None', regex = False, inplace = True)
```

In [19]:

```
# Replace the entry 'Nope' to 'None' for consistency
census_data['Religion'].replace('Nope', 'None', regex = True, inplace = True)

# Replace the entry 'Buddist' to 'Buddhist' for consistency
census_data['Religion'].replace('Buddist', 'Buddhist', regex = True, inplace = True)
```

Replace Methodist, Baptist and Orthodoxy to Christian, as these are not religion

In [20]:

```
# Replace Methodist, Baptist and Orthodoxy to Cristian
census_data['Religion'].replace('Methodist', 'Christian', regex = True, inplace = True)
census_data['Religion'].replace('Baptist', 'Christian', regex = True, inplace = True)
census_data['Religion'].replace('Orthodoxy', 'Christian', regex = True, inplace = True)
```

Fill the missing value of religion with the 'Head' of house, since children can adopt their parents religion

In [21]:

```
# Filter age less than 18 from religion missing values
religion_age_data = census_data[(census_data['Religion'].isna()) & (census_data['Age'] < 18)]

'# Iterate through the data
for index, row in religion_age_data.iterrows():
    # Find the row of each head of house
    head_of_house = census_data[(census_data['House Number'] == row['House Number']) &
                                (census_data['Street'] == row['Street']) &
                                (census_data['Relationship to Head of House'] == 'Head')]

    # If head of house is not empty, replace the missing values with it
    if not head_of_house.empty:
        census_data.loc[index, 'Religion'] = head_of_house.iloc[0]['Religion']
```

```
  File "C:\Users\rosem\AppData\Local\Temp\ipykernel_11532\571342830.py", line 4
    '# Iterate through the data
                               ^
SyntaxError: EOL while scanning string literal
```

In [22]:

```python
# Replace other missing values with None
census_data['Religion'] = census_data['Religion'].fillna('None')
```

Marital Status

In [23]:

```python
# print the unique entries in Marital Status
print(census_data['Marital Status'].unique())
```

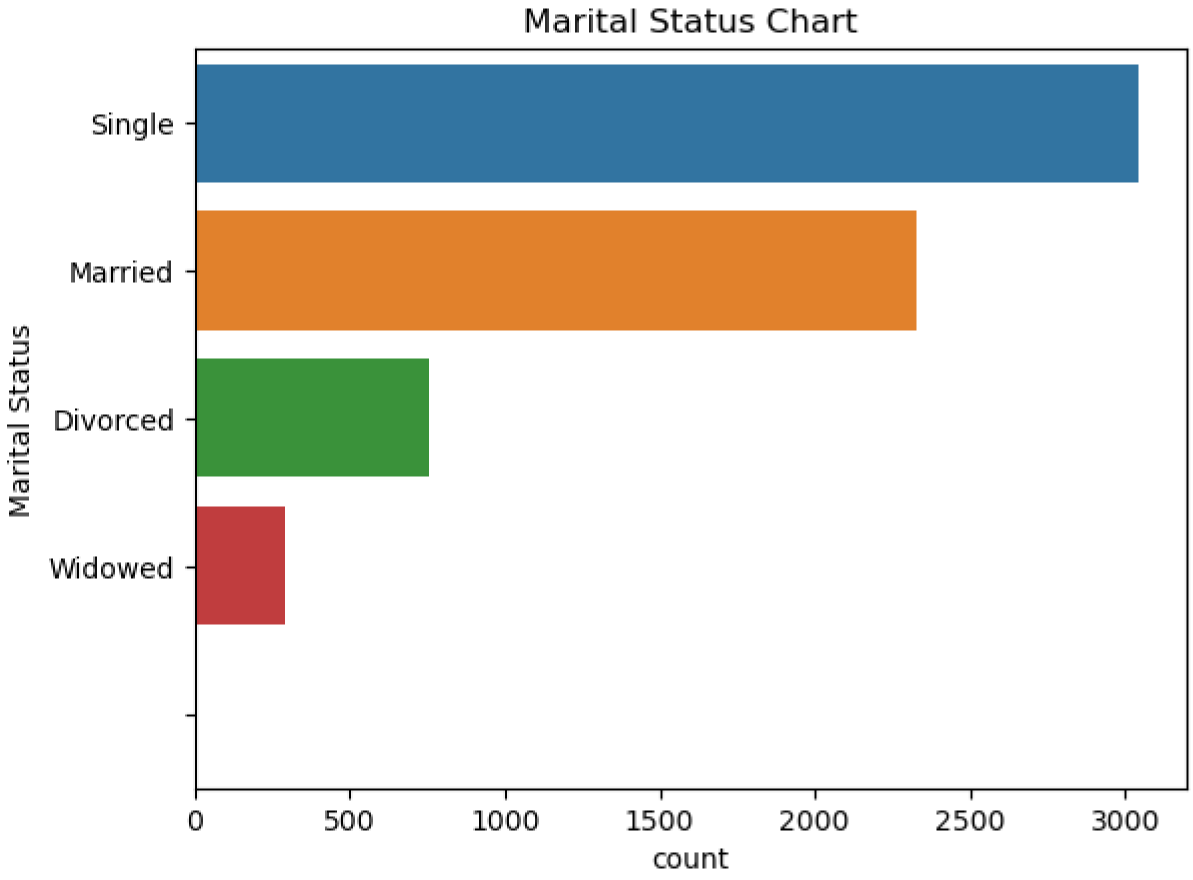
```
['Single' 'Married' nan 'Divorced' 'Widowed' ' ']
```

In [24]:

```
# Plot a chart for Marital Status
marital_status_count = sns.countplot(census_data, y = census_data['Marital Status'])

# Title the plot
marital_status_count.set(title='Marital Status Chart')
```

Out[24]:

```
[Text(0.5, 1.0, 'Marital Status Chart')]
```



In [25]:

```python
# Check blank entries for Marital Status
census_data[census_data['Marital Status'] == ' ']
```

Out[25]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
3205	43	Morgan Fords	Diana	Robinson	39	Wife		Female	Hospital pharmacist	None	None

In [26]:

```python
# Check the range of each entry to find a family tie
census_data[3203:3209]
```

Out[26]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
3204	43	Morgan Fords	Peter	Robinson	42	Head	Married	Male	Designer, interior/spatial	None	Catholic
3205	43	Morgan Fords	Diana	Robinson	39	Wife		Female	Hospital pharmacist	None	None
3206	43	Morgan Fords	Wayne	Robinson	12	Son	NaN	Male	Student	None	None
3207	43	Morgan Fords	Dale	Robinson	5	Son	NaN	Male	Student	None	None
3208	43	Morgan Fords	Charles	Robinson	3	Son	NaN	Male	Child	None	None
3209	44	Morgan Fords	Beverley	Williams	39	Head	Divorced	Female	Copywriter, advertising	None	None

In [27]:

```python
# Replace row 3205 with Married, since the husband who is the head of house is Married
census_data.loc[3205,'Marital Status'] = 'Married'
```

Change children marital status to 'Not Available' because they do not have a marital status yet until they are 18 years old

In [28]:

```python
# Find missing values in marital status for children
children_missing_value = (census_data['Age'] < 18) & (census_data['Marital Status'].isna())

# Replace missing values with 'Not Available'
census_data.loc[children_missing_value, 'Marital Status'] = 'Not Available'
```

Gender

In [29]:

```python
census_data['Gender'].unique()
```

Out[29]:

```
array(['Male', 'Female', ' '], dtype=object)
```

In [30]:

```python
# checks for blank entries in Gender
census_data[census_data['Gender'] == ' ']
```

Out[30]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
6013	9	Lime Street	Elizabeth	Dobson	4	Daughter	Not Available		Child	None	None
7538	37	Leedsbox Crescent	Liam	Yates	66	Head	Married		Television production assistant	None	Christian

In [31]:

```
# Check the range of each entry to find a family tie
census_data[7535:7540]
```

Out[31]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
7537	36	Leedsbox Crescent	Danielle	Palmer	69	Head	Widowed	Female	Retired Engineer, mining	None	Christian
7538	37	Leedsbox Crescent	Liam	Yates	66	Head	Married		Television production assistant	None	Christian
7539	37	Leedsbox Crescent	Hayley	Yates	66	Wife	Married	Female	Solicitor, Scotland	None	Christian
7540	37	Leedsbox Crescent	Dylan	Yates	41	Son	Single	Male	Materials engineer	None	None
7541	37	Leedsbox Crescent	Terry	Yates	38	Son	Single	Male	Designer, fashion/clothing	None	Christian

In [32]:

```
# Replace with Female, since her status is 'Daughter'
census_data.loc[6013,'Gender'] = 'Female'
```

In [33]:

```
# Replace with Male, since he is the head and has a wife
census_data.loc[7538,'Gender'] = 'Male'
```

Infirmity

In [34]:

```
# Check for unique entries in Infirmity
census_data['Infirmity'].unique()
```

Out[34]:

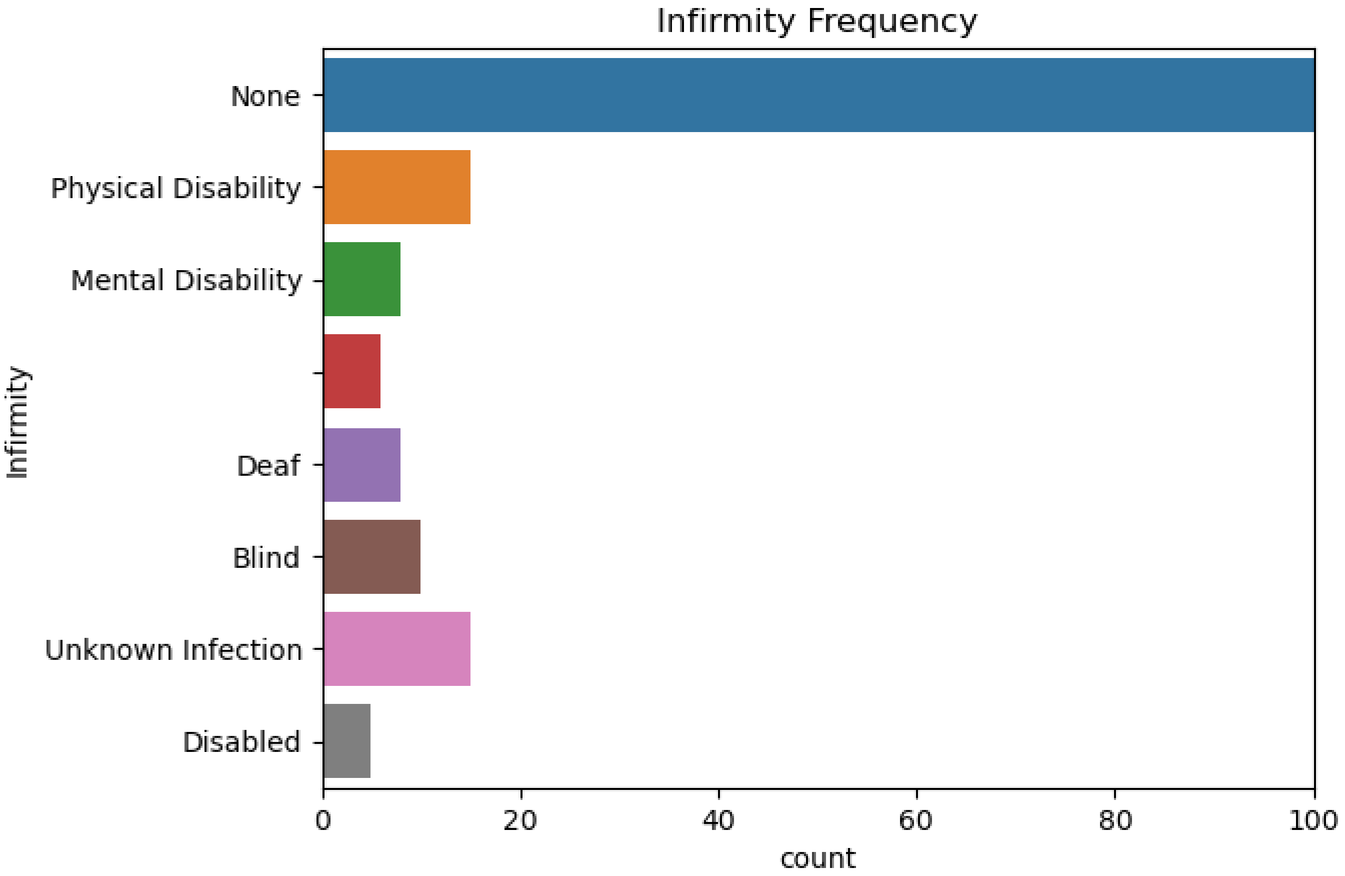
```
array(['None', 'Physical Disability', 'Mental Disability', ' ', 'Deaf',
       'Blind', 'Unknown Infection', 'Disabled'], dtype=object)
```

In [35]:

```
# Count plot for infirmity
infirmity_count = sns.countplot(census_data, y = census_data['Infirmity'])

# limit the count to 100
plt.xlim(0,100)

# Title the plot
infirmity_count.set(title='Infirmity Frequency')
```

Out[35]:

```
[Text(0.5, 1.0, 'Infirmity Frequency')]
```



In [36]:

```python
# Check for blank entires in Infirmity
census_data[census_data['Infirmity']== ' ']
```

Out[36]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
556	1	Morgan View	Sean	Howe	15	Son	Not Available	Male	Student		None
909	15	Newfound Station	Lynda	Murphy	24	Head	Single	Female	Public affairs consultant		Christian
1120	24	Palmer Crescent	Garry	Burns	52	Husband	Married	Male	Actuary		None
4244	47	Madridgate Drive	Fiona	Lloyd	79	Wife	Married	Female	Retired Contractor		Christian
6047	12	Graham Road	Caroline	Bruce	46	Head	Divorced	Female	Chartered management accountant		Catholic
7727	9	Salmon Lane	Holly	Francis	40	Head	Single	Female	Programmer, systems		Catholic

In [37]:

```python
# Replace all blank entries with None
census_data['Infirmity']= census_data['Infirmity'].replace(' ', 'None')
```

Surname

In [38]:

```python
# Check for blank entries in Surname
census_data[census_data['Surname']== ' ']
```

Out[38]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
2123	24	Morley Lodge	Simon		56	None	Single	Male	Information officer	None	Christian
3168	33	Morgan Fords	Stephanie		8	Daughter	Not Available	Female	Student	None	None

In [39]:

```python
# Check the range to identify family relationship
census_data[2121:2126]

# There is no family member associated with 2123, I will ignore the data and continue.
# A blank surname will not affect our analysis
```

Out[39]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
2122	24	Morley Lodge	Caroline	Barber	38	None	Single	Female	Pharmacologist	None	Christian
2123	24	Morley Lodge	Simon		56	None	Single	Male	Information officer	None	Christian
2124	25	Morley Lodge	Dylan	Griffiths	34	Head	Single	Male	Financial manager	None	Catholic
2125	25	Morley Lodge	Eleanor	Griffiths	43	Cousin	Single	Female	Teacher, secondary school	None	Catholic
2126	26	Morley Lodge	Lindsey	Smith	44	Head	Single	Female	Unemployed	None	Catholic

In [40]:

```python
# Check the range to identify family relationship
census_data[3165:3170]
```

Out[40]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
3166	33	Morgan Fords	Lorraine	Griffin	31	Head	Married	Female	Unemployed	None	None
3167	33	Morgan Fords	Henry	Griffin	31	Husband	Married	Male	Acupuncturist	None	None
3168	33	Morgan Fords	Stephanie		8	Daughter	Not Available	Female	Student	None	None
3169	33	Morgan Fords	Francis	Griffin	4	Son	Not Available	Male	Child	None	None
3170	33	Morgan Fords	Kathryn	Dobson	2	Daughter	Not Available	Female	Child	None	None

In [41]:

```
# Input surname for 3168 with Griffin since they are family
census_data.loc[3168, 'Surname']='Griffin'
```

First Name

In [42]:

```
# Check for blank cells in First Name
census_data[census_data['First Name']== ' ']

#First Name is a unique value, I will ignore it and continue, as this will not affect our further anlysis
```

Out[42]:

	House Number	Street	First Name	Surname	Age	Relationship to Head of House	Marital Status	Gender	Occupation	Infirmity	Religion
618	19	Morgan View		Ali	8	Son	Not Available	Male	Student	None	None
3266	4	Simmons Course		Wong	9	Son	Not Available	Male	Student	None	None
3916	6	ExcaliburBells Road		Doyle	5	Son	Not Available	Male	Student	None	None

House Number, Street

In [43]:

```
len(census_data[census_data['House Number']== ' '])
```

Out[43]:

```
0
```

In [44]:

```
len(census_data[census_data['Street'] ==' '])
```

Out[44]:

```
0
```

In [45]:

```
len(census_data[census_data['Relationship to Head of House']== ' '])
```

Out[45]:

```
0
```

In [46]:

```
len(census_data[census_data['Occupation']== ' '])
```

Out[46]:

```
0
```

Discussion / Analysis

In [47]:

```
# Save the cleaned data to a csv file
census_data.to_csv('cleaned_data.csv', index = False)
```

In [48]:

```
# Read the cleaned csv file
new_data = pd.read_csv('cleaned_data.csv')
```

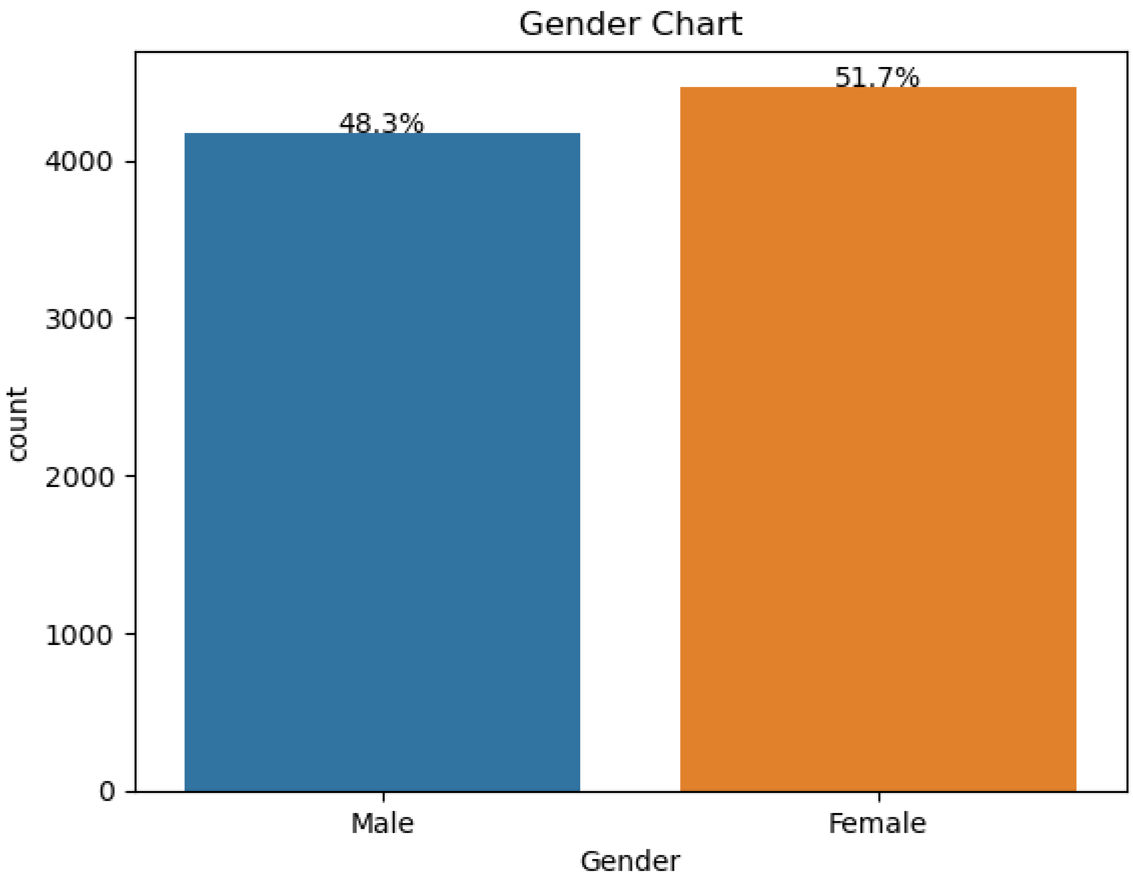
Age Distribution

In [49]:

```
# Create a plot for the Gender column
gender_plot = sns.countplot(data=new_data, x='Gender')

# Calculate the total count of gender
total_frequency = len(new_data['Gender'])

# Loop through each bar in the plot
for p in gender_plot.patches:
    # Get the percentage value for the bar
    percentage = p.get_height() / total_frequency * 100

    # Add the percentage as text above the bar
    gender_plot.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va=

# Give the plot a title and name the x-axis
gender_plot.set_title('Gender Chart')
gender_plot.set_xlabel('Gender')

```

Out[49]:

```
Text(0.5, 0, 'Gender')
```



Population Pyramid

In [50]:

```
# Create a list of age group with bin width 10, and add it to the existing new_data

bin_width = [0,9,19,29,39,49,59,69,79,89,99,105]
new_data['Age Group'] = pd.cut(new_data['Age'],bin_width,
                               labels = ['0-9','10-19','20-29','30-39','40-49','50-59','60-69','70-79','80-89','9
```

In [51]:

```
# Group age by gender
group_age_bygender = new_data.groupby(["Age Group", "Gender"]).size().unstack()
```

In [52]:

```
# Create a variable name for male age after multiplying it by 1
male_age = group_age_bygender["Male"] * -1

# Create a variable name for female age
female_age = group_age_bygender["Female"]

# Use female data as the age group
age_group = group_age_bygender.index
```

In [53]:

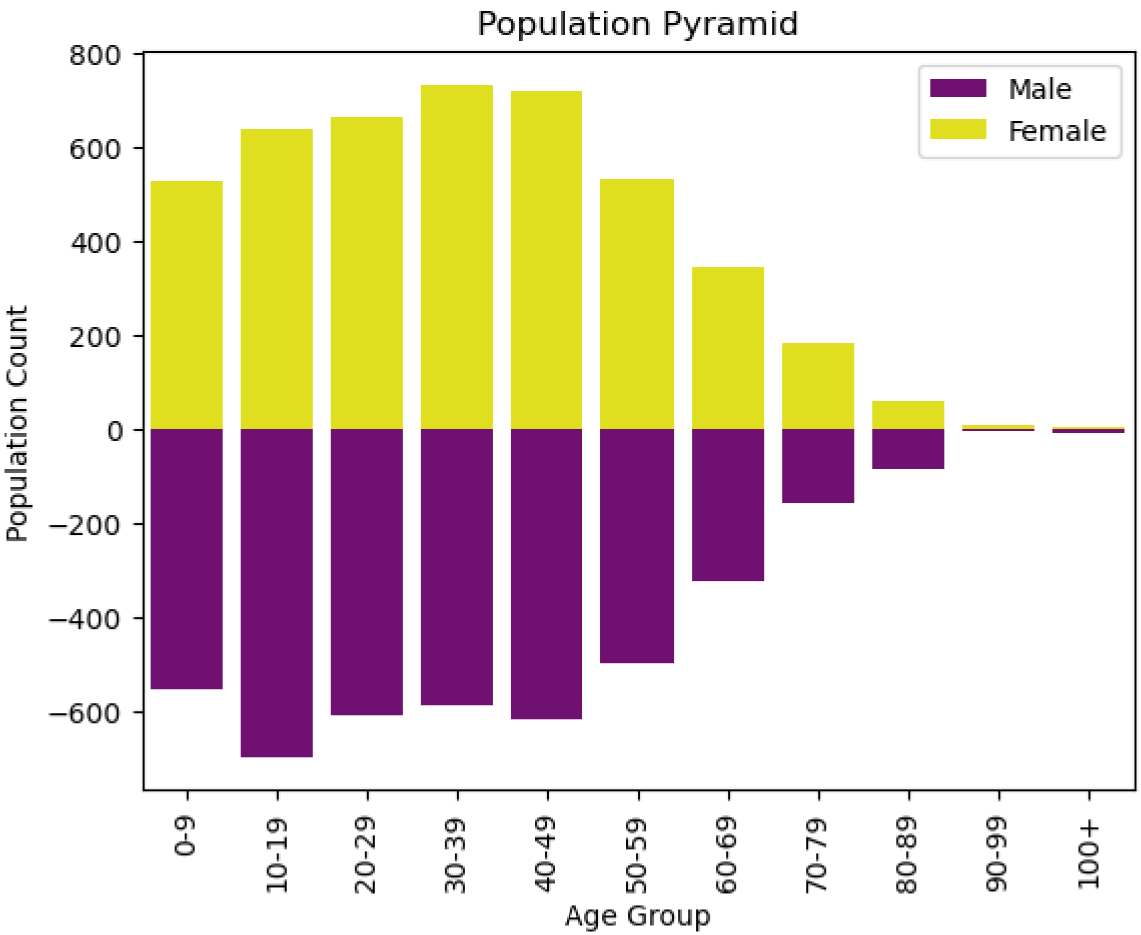
```
# Create the dataframe for Age
age_df = pd.DataFrame({'Age': age_group,
'Male': male_age,
'Female': female_age})
```

In [54]:

```python
# Create a barplot for the male population
age_pyramid = sns.barplot(y='Male', x='Age', data=age_df,  color=('purple'), label='Male')

# Create a second barplot on top of the first one for the female population
age_pyramid = sns.barplot(y='Female', x='Age', data=age_df,  color=('yellow'), label='Female')

# Add a legend to the plot to differentiate between male and female bars
age_pyramid.legend()

# Rotate x-labels by 90 degrees
age_pyramid.set_xticklabels(age_pyramid.get_xticklabels(), rotation=90)

# Title the plot, x and y axis
age_pyramid.set(ylabel='Population Count', xlabel='Age Group')
plt.title('Population Pyramid')
```

Out[54]:

```
Text(0.5, 1.0, 'Population Pyramid')
```

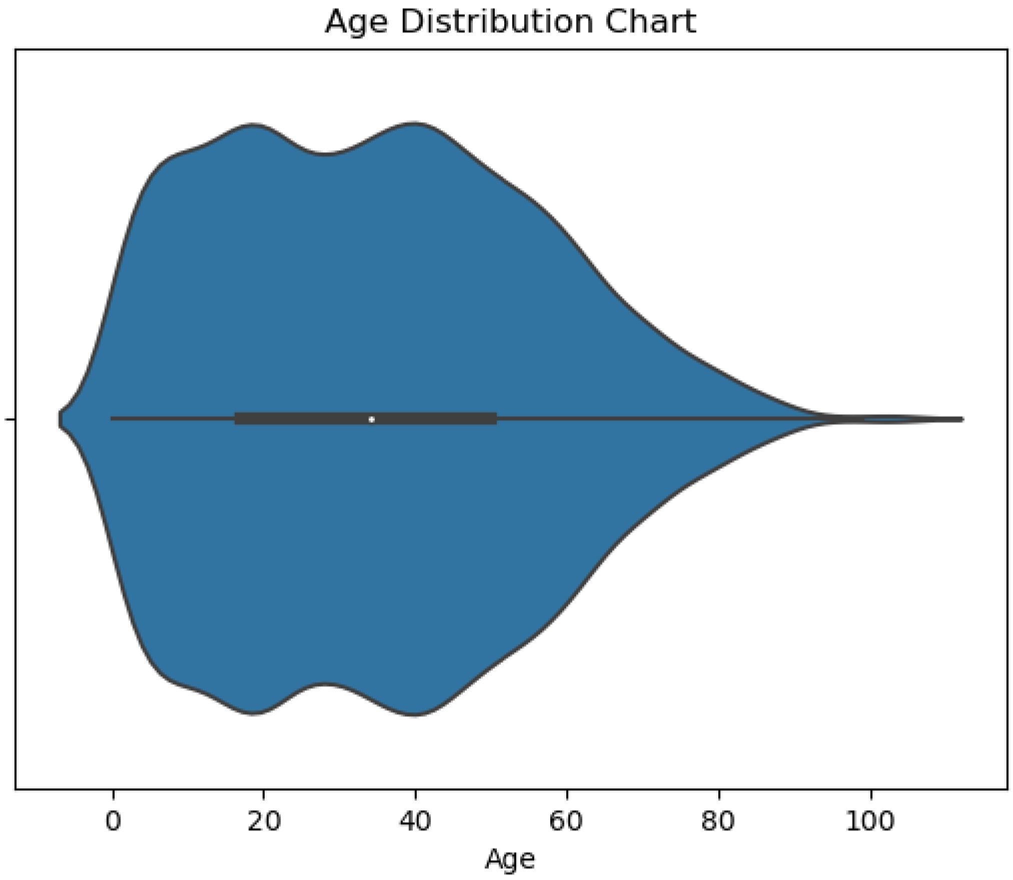

Population Pyramid
Population Count
800
600
400
200
0
−200
−400
−600
Male
Female
0-9
10-19
20-29
30-39
40-49
50-59
60-69
70-79
80-89
90-99
100+
Age Group

In [55]:

```
# Violin plot for Age distribution
violin_age_plot = sns.violinplot(new_data, x = 'Age')

# Title the plot
violin_age_plot.set(title= 'Age Distribution Chart')
```

Out[55]:

```
[Text(0.5, 1.0, 'Age Distribution Chart')]
```

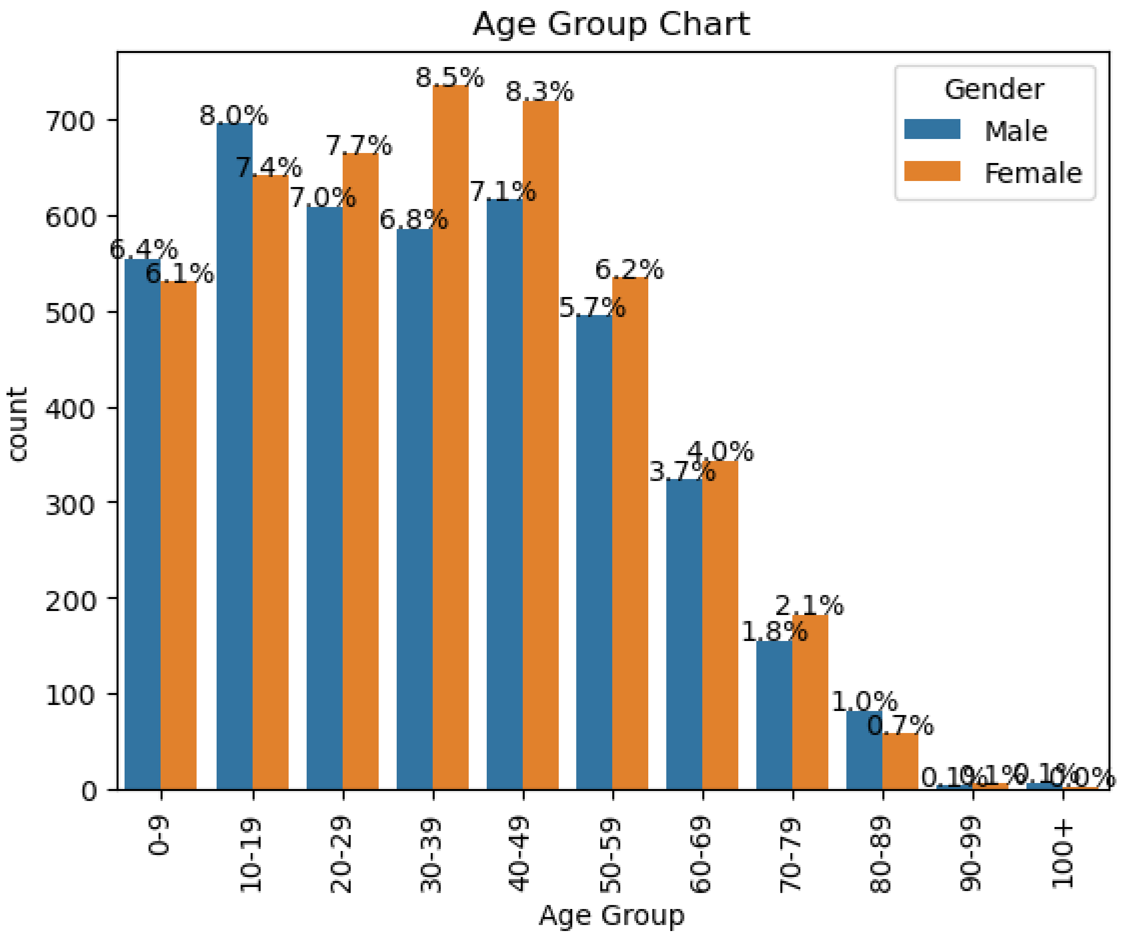


In [56]:

```
# Create a plot for the Gender column
age_group_plot = sns.countplot(data=new_data, x='Age Group', hue = 'Gender')

# Calculate the total count of age group
total_frequency = len(new_data['Age Group'])

# Loop through each bar in the plot
for p in age_group_plot.patches:
    # Get the percentage value for the bar
    percentage = p.get_height() / total_frequency * 100

    # Add the percentage as text above the bar
    age_group_plot.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center',

# Rotate the x-axis label
age_group_plot.set_xticklabels(age_group_plot.get_xticklabels(), rotation=90)

# Give the plot a title and name the x-axis
age_group_plot.set(title = 'Age Group Chart')
age_group_plot.set_xlabel('Age Group')
```

Out[56]:

```
Text(0.5, 0, 'Age Group')
```



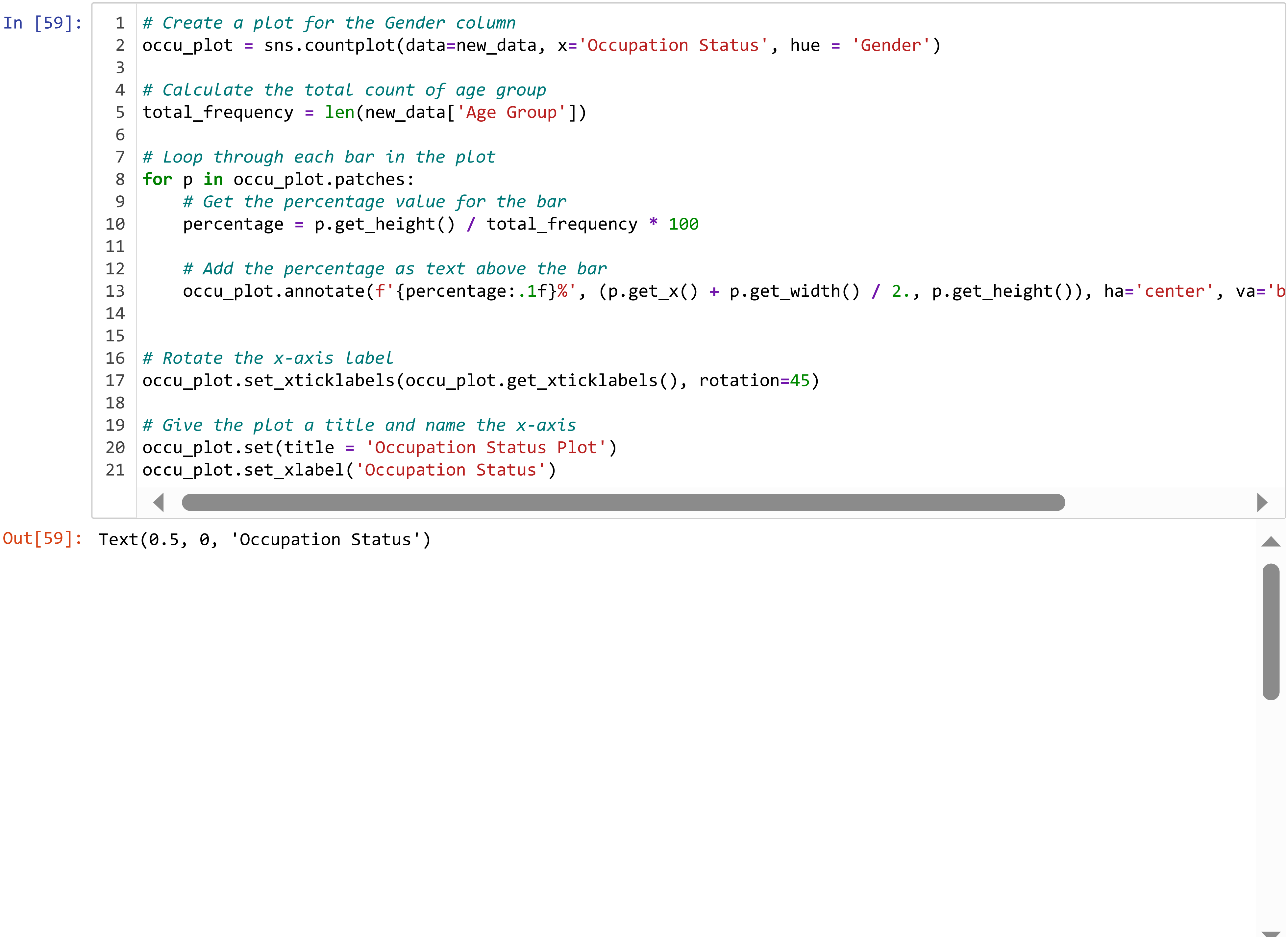
Occupation Level

In [57]:

```python
def occupation_status(occupation):

    if 'University Student'in occupation:
        return 'University Student'
    if 'PhD Student'in occupation:
        return 'PhD Student'
    if 'Student' in occupation:
        return 'Students'
    elif 'Retired' in occupation:
        return 'Retired'
    elif 'Child' in occupation and not ("Child psychotherapist" in occupation or "Nurse,children's" in occupation
        return 'Child'
    elif 'Unemployed' in occupation:
        return 'Unemployed'
    else:
        return 'Employed'
```

In [58]:

```python
# Create a new column called Occupation Status
new_data['Occupation Status']= new_data['Occupation'].apply(occupation_status)
```

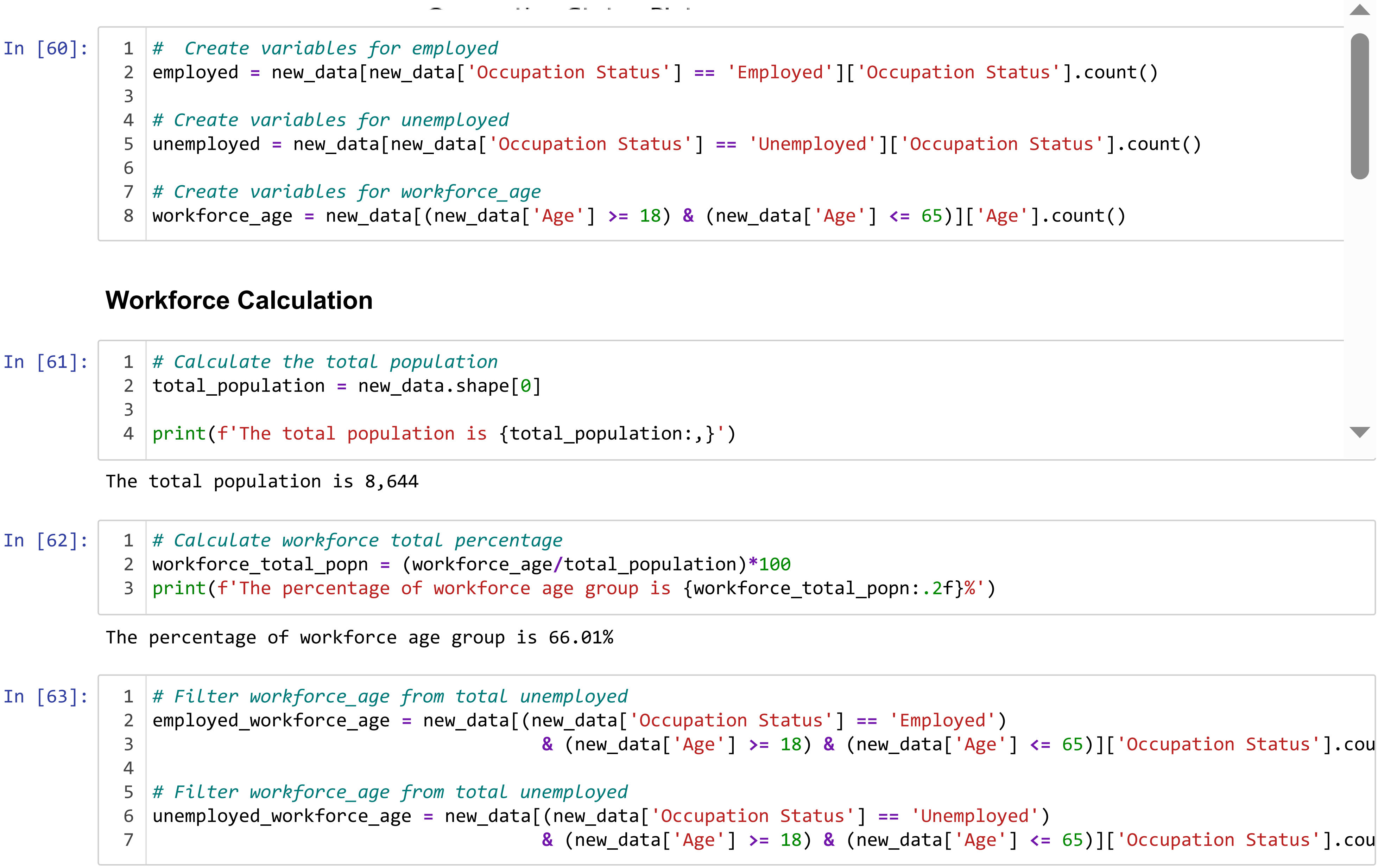


In [59]:

```python
# Create a plot for the Gender column
occu_plot = sns.countplot(data=new_data, x='Occupation Status', hue = 'Gender')

# Calculate the total count of age group
total_frequency = len(new_data['Age Group'])

# Loop through each bar in the plot
for p in occu_plot.patches:
    # Get the percentage value for the bar
    percentage = p.get_height() / total_frequency * 100

    # Add the percentage as text above the bar
    occu_plot.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2., p.get_height()), ha='center', va='b


# Rotate the x-axis label
occu_plot.set_xticklabels(occu_plot.get_xticklabels(), rotation=45)

# Give the plot a title and name the x-axis
occu_plot.set(title = 'Occupation Status Plot')
occu_plot.set_xlabel('Occupation Status')
```

Out[59]:

```
Text(0.5, 0, 'Occupation Status')
```



In [60]:

```
#  Create variables for employed
employed = new_data[new_data['Occupation Status'] == 'Employed']['Occupation Status'].count()

# Create variables for unemployed
unemployed = new_data[new_data['Occupation Status'] == 'Unemployed']['Occupation Status'].count()

# Create variables for workforce_age
workforce_age = new_data[(new_data['Age'] >= 18) & (new_data['Age'] <= 65)]['Age'].count()
```

Workforce Calculation

In [61]:

```
# Calculate the total population
total_population = new_data.shape[0]

print(f'The total population is {total_population:,}')
```

```
The total population is 8,644
```

In [62]:

```
# Calculate workforce total percentage
workforce_total_popn = (workforce_age/total_population)*100
print(f'The percentage of workforce age group is {workforce_total_popn:.2f}%')
```

```
The percentage of workforce age group is 66.01%
```

In [63]:

```
# Filter workforce_age from total unemployed
employed_workforce_age = new_data[(new_data['Occupation Status'] == 'Employed')
                                    & (new_data['Age'] >= 18) & (new_data['Age'] <= 65)]['Occupation Status'].cou

# Filter workforce_age from total unemployed
unemployed_workforce_age = new_data[(new_data['Occupation Status'] == 'Unemployed')
                                    & (new_data['Age'] >= 18) & (new_data['Age'] <= 65)]['Occupation Status'].cou
```

In [64]:

```python
# Check for employed that is within the workforce age
employed_workforce = employed_workforce_age/(unemployed_workforce_age + employed_workforce_age)

# Multiply by 100 to get the rate
employed_workforce_rate = employed_workforce * 100

# Print the result
print(f'The employed workforce rate is {employed_workforce_rate:.2f}%')
```

```
The employed workforce rate is 89.43%
```

In [65]:

```python
# Check for unemployed that is within the workforce age
unemployed_workforce = unemployed_workforce_age/(unemployed_workforce_age + employed_workforce_age)

# Multiply by 100 to get the rate
unemployed_workforce_rate = unemployed_workforce * 100

# Print the result
print(f'The unemployed workforce rate is {unemployed_workforce_rate:.2f}%')
```

```
The unemployed workforce rate is 10.57%
```

Population Growth

Birth Rate Calculation

In [66]:

```
# Find the number of infant = 0 year old
infant = new_data['Age'] == 0
infant_num = len(new_data[infant])

# Filter women at age 25-29 years
age_25_29 = new_data[(new_data['Age'] >= 25) & (new_data['Age'] <= 29)]
female_age_25_29= len(age_25_29[age_25_29['Gender']=='Female'])

# Find the total birth of women between the age of 25_29 years
num_birth_25_29 = (infant_num/female_age_25_29)*100000
birth_rate_25_29 = (num_birth_25_29)/total_population

print(f'The birth rate /annum of women between 25 and 29 years is {birth_rate_25_29:.2f}%','of 100,000 population
```

```
The birth rate /annum of women between 25 and 29 years is 3.55% of 100,000 population:
```

In [67]:

```
# Find the number of babies = 4 years old
babies = new_data['Age'] == 4
babies_num = len(new_data[babies])

# Filter women at age 30-34 years
age_30_34 = new_data[(new_data['Age'] >= 30) & (new_data['Age'] <= 34)]
fermale_age_30_34= len(age_30_34[age_30_34['Gender']=='Female'])

# Find the birth rate of women between the age of 30-34 years
num_birth_30_34 = (babies_num/fermale_age_30_34)*100000
birth_rate_30_34 = (num_birth_30_34)/total_population

print(f'The birth rate/annum of women between 30 and 34 years is {birth_rate_30_34:.2f}%','of 100,000 population:
```

```
The birth rate/annum of women between 30 and 34 years is 3.60% of 100,000 population:
```

Death Rate Calculation

In [68]:

```python
# Filter the Age of persons between 50-59years
aged_50_59 = new_data[(new_data['Age'] >= 50) & (new_data['Age'] <= 59)]

# Filter the Age of persons between 60-69years
aged_60_69 = new_data[(new_data['Age'] >= 60) & (new_data['Age'] <= 69)]

# The difference between the two age groups
death_1= len(aged_50_59) - len(aged_60_69)

# Base population = total population
total_popn_1 =len(aged_50_59)

# Find the death rate per annum per 100,000 population
no_of_death1 = (death_1/total_popn_1)*100000

# Number of death/annum
annual_death_1 = int(no_of_death1/10)

# Print the result
print(f'The death/annum of people betwwen 50 to 69 years is {annual_death_1:,}', 'of 100,000 population')
```

```
The death/annum of people betwwen 50 to 69 years is 3,517 of 100,000 population
```

In [69]:

```
# Filter the Age of persons between 70-79years
aged_70_79 = new_data[(new_data['Age'] >= 70) & (new_data['Age'] <= 79)]

# Filter the Age of persons between 80-89years
aged_80_89 = new_data[(new_data['Age'] >= 80) & (new_data['Age'] <= 89)]

death_2 = len(aged_70_79)- len(aged_80_89)
total_popn_2 = len(aged_70_79)

no_of_death2 = (death_2/total_popn_2)*100000
annual_death_2 = int(no_of_death2/10)

# Print the result
print(f'The death/annum of people betwwen 70 to 89 years is {annual_death_2:,}', 'of 100,000 population')
```

```
The death/annum of people betwwen 70 to 89 years is 5,811 of 100,000 population
```

In [70]:

```
# Filter the Age of persons between 90-99years
aged_90_99 = new_data[(new_data['Age'] >= 90) & (new_data['Age'] <= 99)]

# Filter the Age of persons between 100-110years
aged_100_110 = new_data[(new_data['Age'] >= 100) & (new_data['Age'] <= 109)]

death_3 = len(aged_90_99) - len(aged_100_110)
total_popn_3 = len(aged_90_99)

no_of_death3 = (death_3/total_popn_3)*100000
annual_death_3 = int(no_of_death3/10)

# Print the result
print(f'The death/annum of people betwwen 90 to 109 years is {annual_death_3:,}', 'of 100,000 population')
```

```
The death/annum of people betwwen 90 to 109 years is 833 of 100,000 population
```

In [71]:

```python
# Total death of 100,000 population
total_death = annual_death_1 + annual_death_2 + annual_death_3

# Total death rate of 100,000 population
total_death_rate = total_death/total_population
print(f'The total_death_rate/annum is {total_death_rate:.2f}%', 'of 100,000 population')
```

```
The total_death_rate/annum is 1.18% of 100,000 population
```

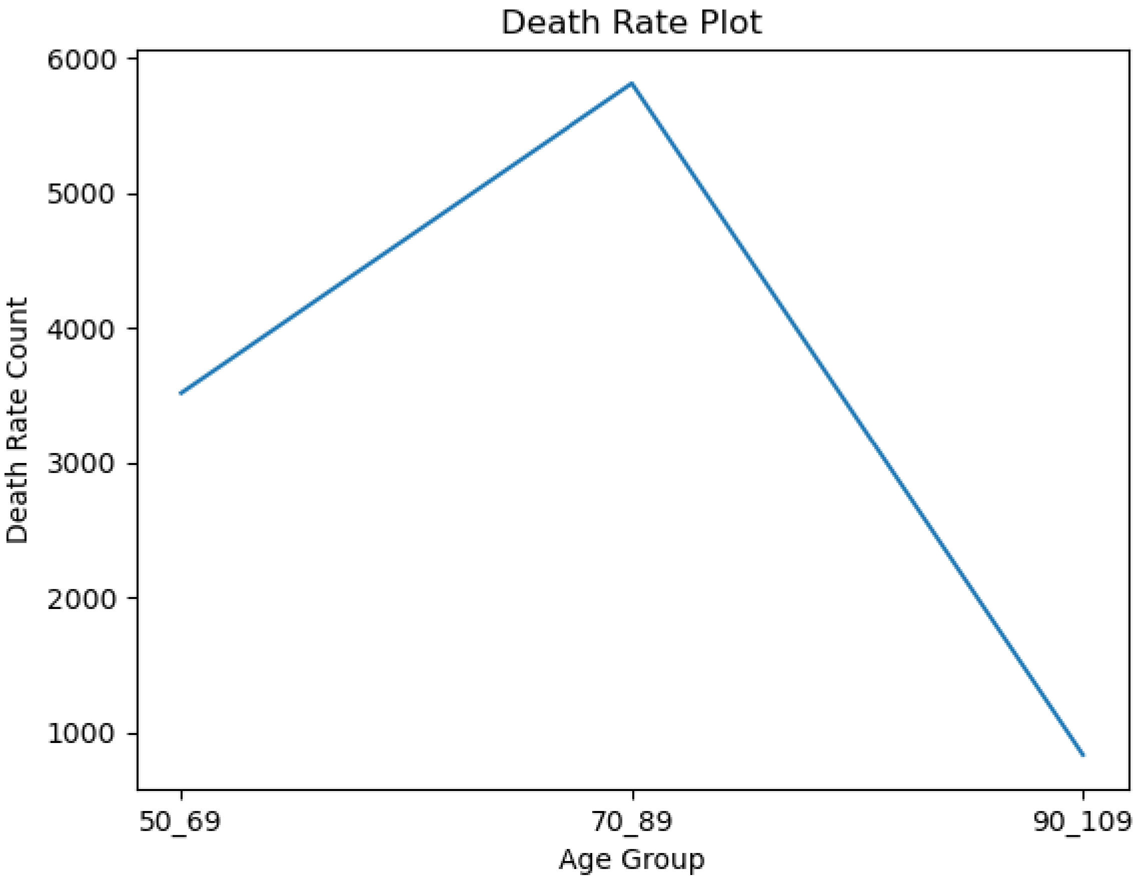
Death Rate Plot

In [72]:

```
# Line Plot for death rate
death_rate_plot = sns.lineplot(x = ['50_69', '70_89', '90_109'], y = [annual_death_1, annual_death_2, annual_deat

# Title the plot, x and y axis
death_rate_plot.set(title = 'Death Rate Plot')
death_rate_plot.set_xlabel('Age Group')
death_rate_plot.set_ylabel('Death Rate Count')
```

Out[72]:

```
Text(0, 0.5, 'Death Rate Count')
```



Migration Calculation

In [73]:

```
# create a variable names for lodgers and visitors
lodgers = new_data[new_data['Relationship to Head of House'] == 'Lodger']['Relationship to Head of House'].count(
visitors = new_data[new_data['Relationship to Head of House'] == 'Visitor']['Relationship to Head of House'].coun

# create a variable names for uni_students and employed
uni_students = new_data[new_data['Occupation'] == 'University Student']['Occupation'].count()

# create a variable name for divorced
divorced = new_data[new_data['Marital Status'] == 'Divorced']['Marital Status'].count()
```

Possible Immigrant

In [74]:

```
# Filter lodgers from total employed
employed_lodgers = new_data[(new_data['Occupation Status'] == 'Employed')
                              & (new_data['Relationship to Head of House'] == 'Lodger')]['Occupation Status'].co

# Filter visitors from total employed
employed_visitors = new_data[(new_data['Occupation Status'] == 'Employed')
                              & (new_data['Relationship to Head of House'] == 'Visitor')]['Occupation Status'].c

# Filter visitors from divorced
divorced_visitors = new_data[(new_data['Marital Status'] == 'Divorced')
                              & (new_data['Relationship to Head of House'] == 'Visitor')]['Marital Status'].coun

```

In [75]:

```
# Total number of possible immigrant
immigrant = (employed_lodgers+ employed_visitors+ divorced_visitors)
immigration_rate = immigrant/total_population

#Print the result
print(f'The Immigration Rate/annum is {immigration_rate:.2f}%')
```

```
The Immigration Rate/annum is 0.04%
```

Possible Emmigrant

In [76]:

```
# Filter divorced from total unemployed
unemployed_divorced = new_data[(new_data['Occupation Status'] == 'Unemployed')
                               & (new_data['Marital Status'] == 'Divorced')]['Occupation Status'].count()

# Filter workforce_age from total unemployed
unemployed_workforce_age = new_data[(new_data['Occupation Status'] == 'Unemployed')
                                    & (new_data['Age'] >= 18) & (new_data['Age'] <= 65)]['Occupation Status'].cou

# Filter divorced from lodgers
divorced_lodgers = new_data[(new_data['Marital Status'] == 'Divorced')
                            & (new_data['Relationship to Head of House'] == 'Lodger')]['Marital Status'].count()
```

In [77]:

```
#Total number of possible emmigrant
emigrant = unemployed_divorced + unemployed_workforce_age + divorced_lodgers
emigration_rate = emigrant/total_population

#Print the result
print(f'The Emmigration Rate/annum is {emigration_rate:.2f}%')
```

```
The Emmigration Rate/annum is 0.08%
```

Migration Rate

In [78]:

```
# Calculate the migration rate
migration_rate = immigration_rate - emigration_rate

# Print the result
print(f'The migration_rate/annum is {migration_rate:.2f}%', 'of 100,000 apopulation')
```

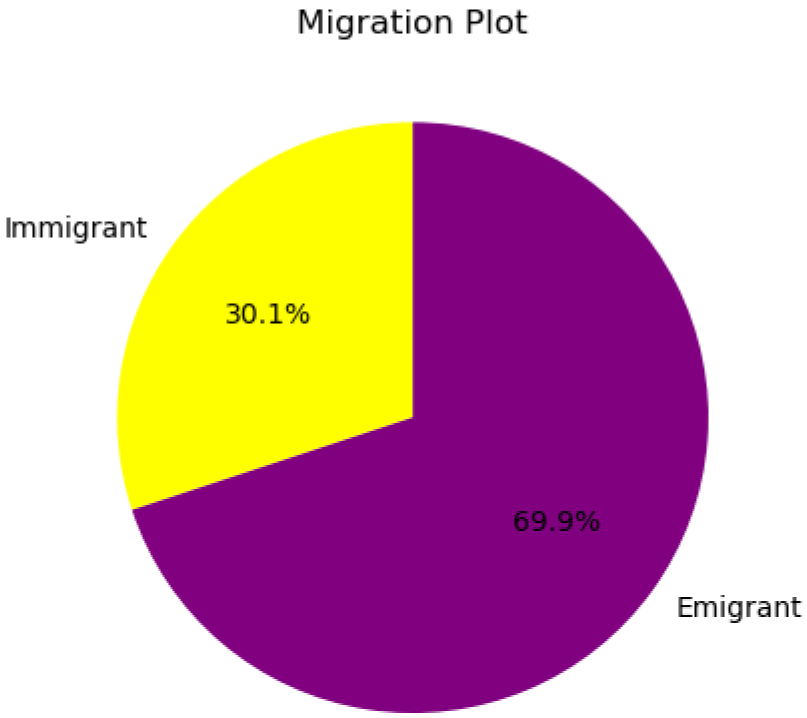
```
The migration_rate/annum is -0.05% of 100,000 apopulation
```

In [79]:

```python
# define the data
labels = ['Immigrant', 'Emigrant']
sizes = [immigration_rate, emigration_rate]
colors = ['yellow','purple']

# create the pie chart
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=90)

# title the plot
plt.title('Migration Plot')
```

Out[79]:

```
Text(0.5, 1.0, 'Migration Plot')
```



Population Growth Rate

In [80]:

```python
# Define the Crude death and Crude birth rate
crude_birth_rate = birth_rate_25_29
crude_death_rate = total_death_rate

# Calculate population growth rate
population_growth_rate = (crude_birth_rate + immigration_rate) - (crude_death_rate + emigration_rate)

# Print the result
print(f'population_growth_rate/annum is {population_growth_rate:.2f}%', 'of 100,000 population')
```

```
population_growth_rate/annum is 2.32% of 100,000 population
```

Occupancy Level

In [81]:

```python
# Find the number of houses, grouped by 'House Number and Street
num_of_houses = new_data.groupby(['House Number', 'Street']).size().reset_index(name = 'Actual Occupant')

# Exclude house number 1 (It is assumed to be an hotel)
num_of_houses = num_of_houses[~(num_of_houses['House Number'] == 1)]
num_of_houses
```

Out[81]:

	House Number	Street	Actual Occupant
105	2	Albans Pines	5
106	2	Albionfold Road	4
107	2	Archer Drives	6
108	2	Arrows Camp	5
109	2	Atkinson Meadows	3
...	...	...	...
2900	242	Hughes Camp	2
2901	243	Hughes Camp	2
2902	244	Hughes Camp	2
2903	245	Hughes Camp	2
2904	246	Hughes Camp	2

In [82]:

```python
# Output the total number of houses
total_houses = num_of_houses.shape[0]

print(f"The number of houses are {total_houses:.0f}")
```

```
The number of houses are 2800
```

In [83]:

```python
# Find the average occupant staying in each house
average_occupant = num_of_houses['Actual Occupant'].mean()

print(f"The Average Occupant is {average_occupant:.0f}")
```

```
The Average Occupant is 3
```

In [84]:

```python
# Number of houses whose occupant are less than or equal to the average occupant
occupant_less_than_3 = num_of_houses[num_of_houses['Actual Occupant']<= 3].shape[0]

print(f"The Occupant less than or equal to 3 is {occupant_less_than_3:.0f}")
```

```
The Occupant less than or equal to 3 is 1806
```

In [85]:

```python
# Number of houses whose occupant are greater than average occupant
occupant_greater_than_3 = num_of_houses[num_of_houses['Actual Occupant']> 3].shape[0]

print(f"The Occupant greater than 3 is {occupant_greater_than_3:.0f}")
```

```
The Occupant greater than 3 is 994
```

In [86]:

```python
# Percentage of houses greater than average occupant
houses_overused = (occupant_greater_than_3/total_houses)*100

print(f"The percentage of houses overused are: {houses_overused:.0f}%")
```

```
The percentage of houses overused are: 36%
```

Possible Commuters

In [87]:

```python
phd_students = new_data['Occupation'].str.contains('phd', case=False)
phd_students = phd_students.sum()
```

In [88]:

```
lecturers = new_data['Occupation'].str.contains('lecturer', case=False) & ~new_data['Occupation'].str.contains('r
lecturers = lecturers.sum()
```

In [89]:

```
professors = new_data['Occupation'].str.contains('professor', case=False) & ~new_data['Occupation'].str.contains(
professors = professors.sum()
```

In [90]:

```
academic_staff = new_data['Occupation'].str.contains('academic', case=False) & ~new_data['Occupation'].str.contai
academic_staff = academic_staff.sum()
```

In [91]:

```
# Total number of commuters
commuters = uni_students + phd_students + lecturers + professors + academic_staff

print(f'The total number of commuters is {commuters:.0f}')
```

```
The total number of commuters is 661
```

In [92]:

```
# Percentage of commuters
perc_commuters = (commuters/total_population) *100

# Print the result
print(f'The percentage of commuters is {perc_commuters:.2f}%')
```

```
The percentage of commuters is 7.65%
```

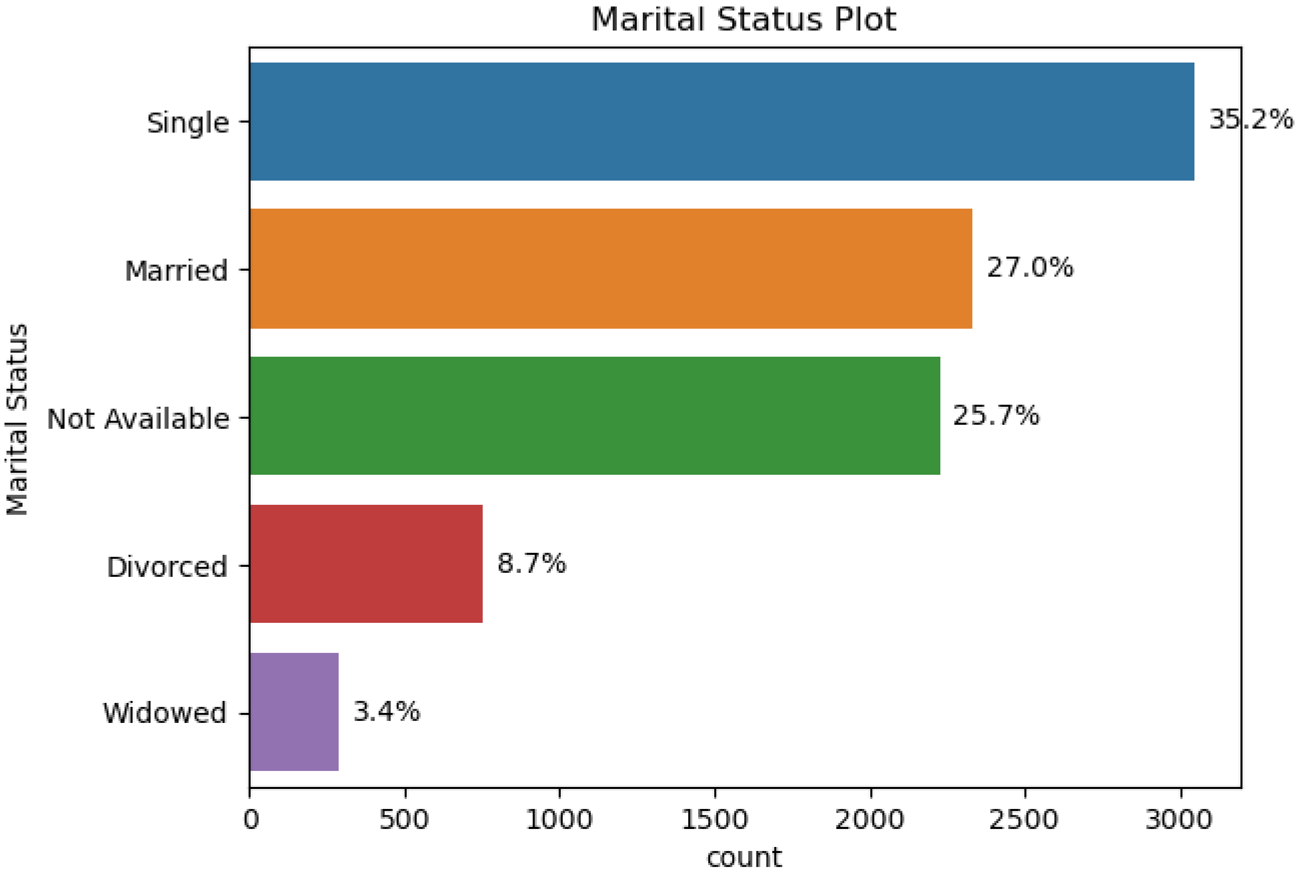
Marital Status

In [93]:

```
# Create a plot for the Gender column
marital_plot = sns.countplot(data=new_data, y='Marital Status')

# Get the total frequency of each category
total_frequency = len(new_data['Marital Status'])

# Loop through each bar in the plot
for p in marital_plot.patches:
    # Get the percentage value for the bar
    percentage = p.get_width() / total_frequency * 100

    # Add the percentage as text next to the bar
    marital_plot.annotate(f'{percentage:.1f}%', (p.get_width(), p.get_y() + p.get_height() / 2.), xytext=(5, 0),

# Rotate the y-axis label
marital_plot.set_yticklabels(marital_plot.get_yticklabels(), rotation=0)

# Give the plot a title and name the y-axis
marital_plot.set(title='Marital Status Plot')
marital_plot.set_ylabel('Marital Status')
```

Out[93]:

```
Text(0, 0.5, 'Marital Status')
```

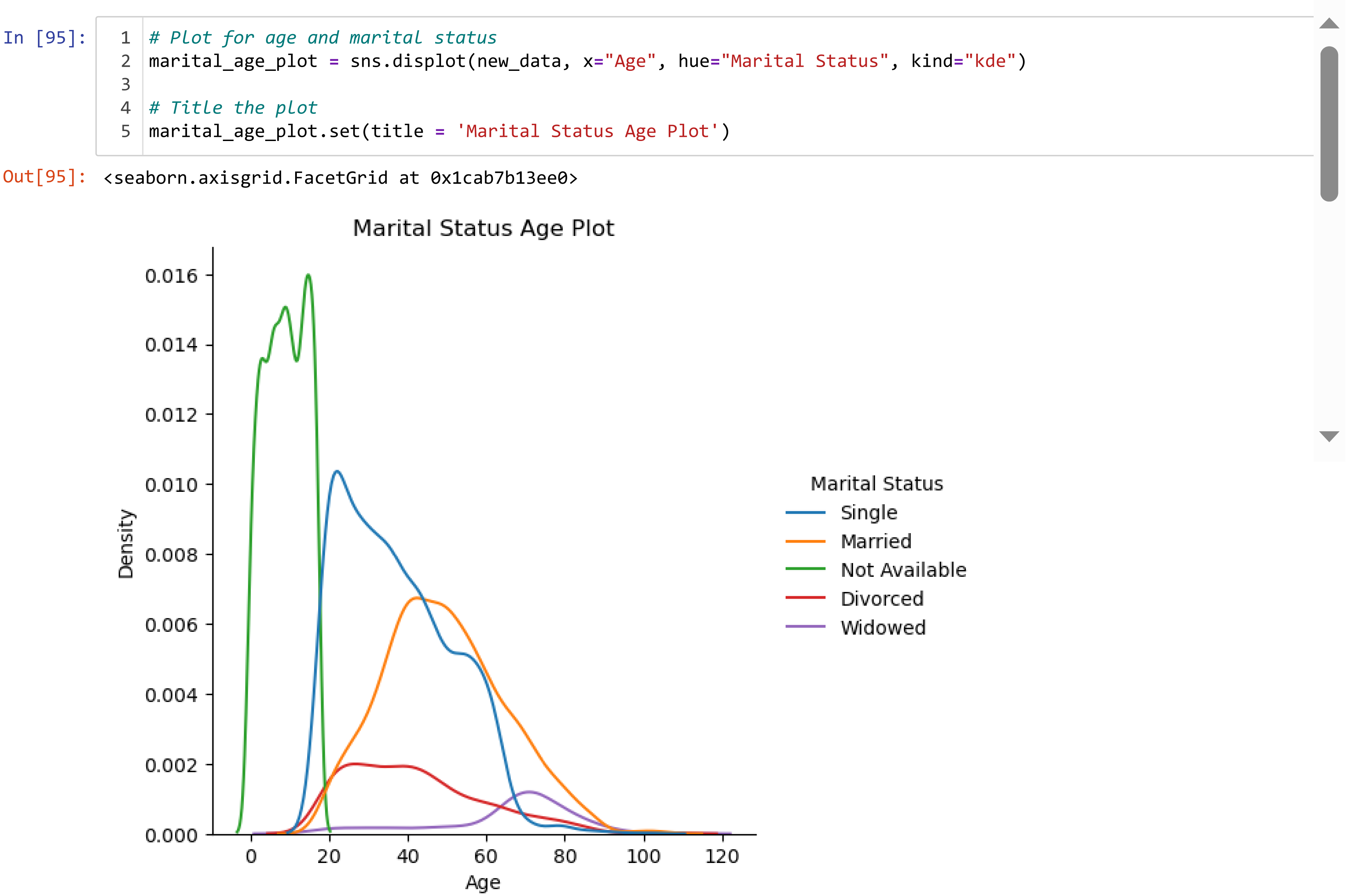

Marital Status Plot
Marital Status
Single
Married
Not Available
Divorced
Widowed
35.2%
27.0%
25.7%
8.7%
3.4%
0
500
1000
1500
2000
2500
3000
count

In [94]:

```python
# Create a plot for the Gender column
marital_plot = sns.countplot(data=new_data, y='Marital Status', hue='Gender')

# Get the total frequency of each category
total_frequency = len(new_data['Marital Status'])

# Loop through each bar in the plot
for p in marital_plot.patches:
    # Get the percentage value for the bar
    percentage = p.get_width() / total_frequency * 100

    # Add the percentage as text next to the bar
    marital_plot.annotate(f'{percentage:.1f}%', (p.get_width(), p.get_y() + p.get_height() / 2.), xytext=(5, 0),

# Rotate the y-axis label
marital_plot.set_yticklabels(marital_plot.get_yticklabels(), rotation=0)

# Give the plot a title and name the y-axis
marital_plot.set(title='Marital Status_Gender Plot')
marital_plot.set_ylabel('Marital Status')
```

Out[94]:

```
Text(0, 0.5, 'Marital Status')
```



In [95]:

```
# Plot for age and marital status
marital_age_plot = sns.displot(new_data, x="Age", hue="Marital Status", kind="kde")

# Title the plot
marital_age_plot.set(title = 'Marital Status Age Plot')
```

Out[95]:

```
<seaborn.axisgrid.FacetGrid at 0x1cab7b13ee0>
```

In [96]:

```python
# Calculate divorce rate against the total population
divorced_rate = (divorced/total_population)*100

# Print the result
print(f'The divorce rate is {divorced_rate:.2f}%')
```

```
The divorce rate is 8.72%
```

In [97]:

```python
# create a variable for married people
married = new_data[new_data['Marital Status'] == 'Married']['Marital Status'].count()

# Calculate divorce rate against the total population
married_rate = (married/total_population)*100

# Print the resultbirth
print(f'The married rate is {married_rate:.2f}%')
```

```
The married rate is 26.96%
```

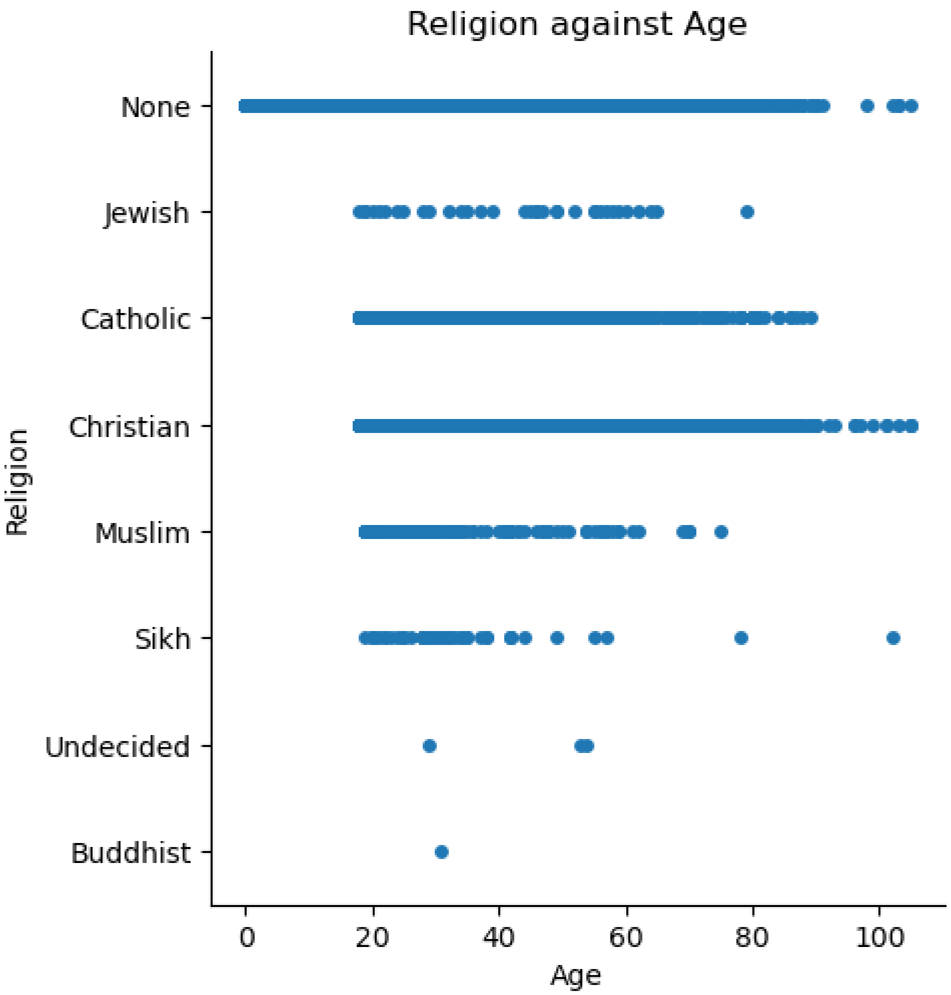
Religious Affliations

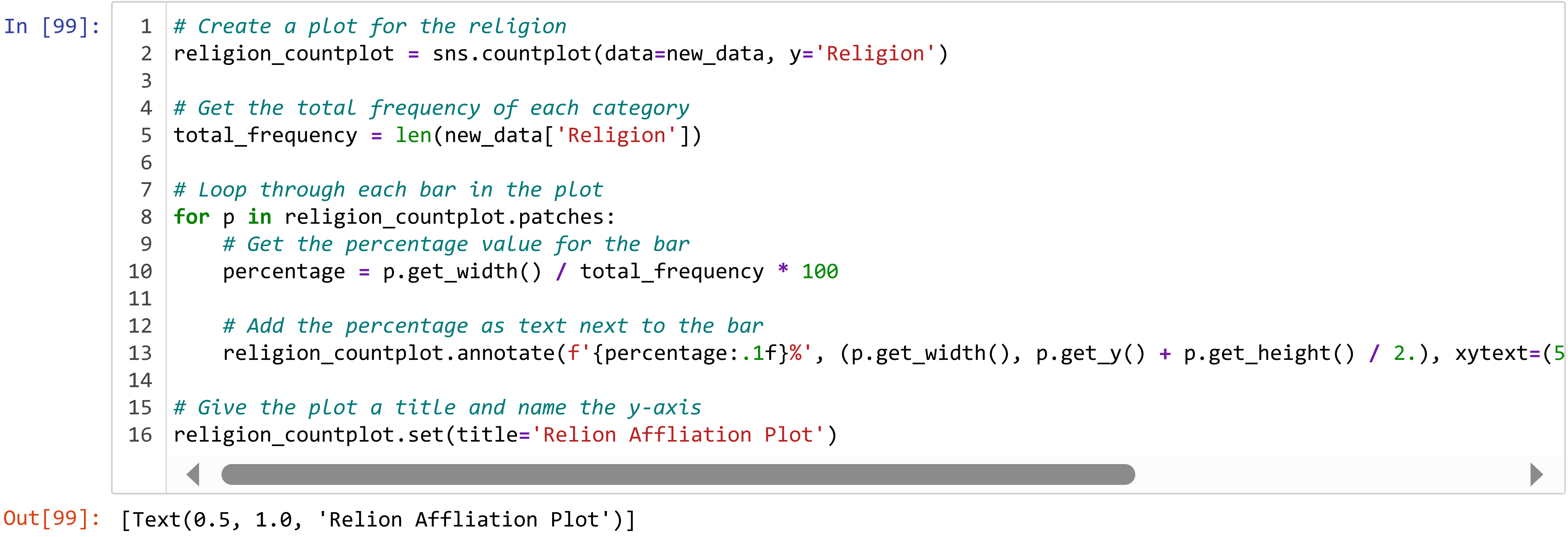
In [98]:

```
# Plot for religion with age
religion_catplot = sns.catplot(new_data, x="Age", y="Religion", jitter=False)

# Title the plot
religion_catplot.set(title = 'Religion against Age')
```

Out[98]:

```
<seaborn.axisgrid.FacetGrid at 0x1cab909d400>
```


Religion against Age
Religion
None
Jewish
Catholic
Christian
Muslim
Sikh
Undecided
Buddhist
0
20
40
60
80
100
Age

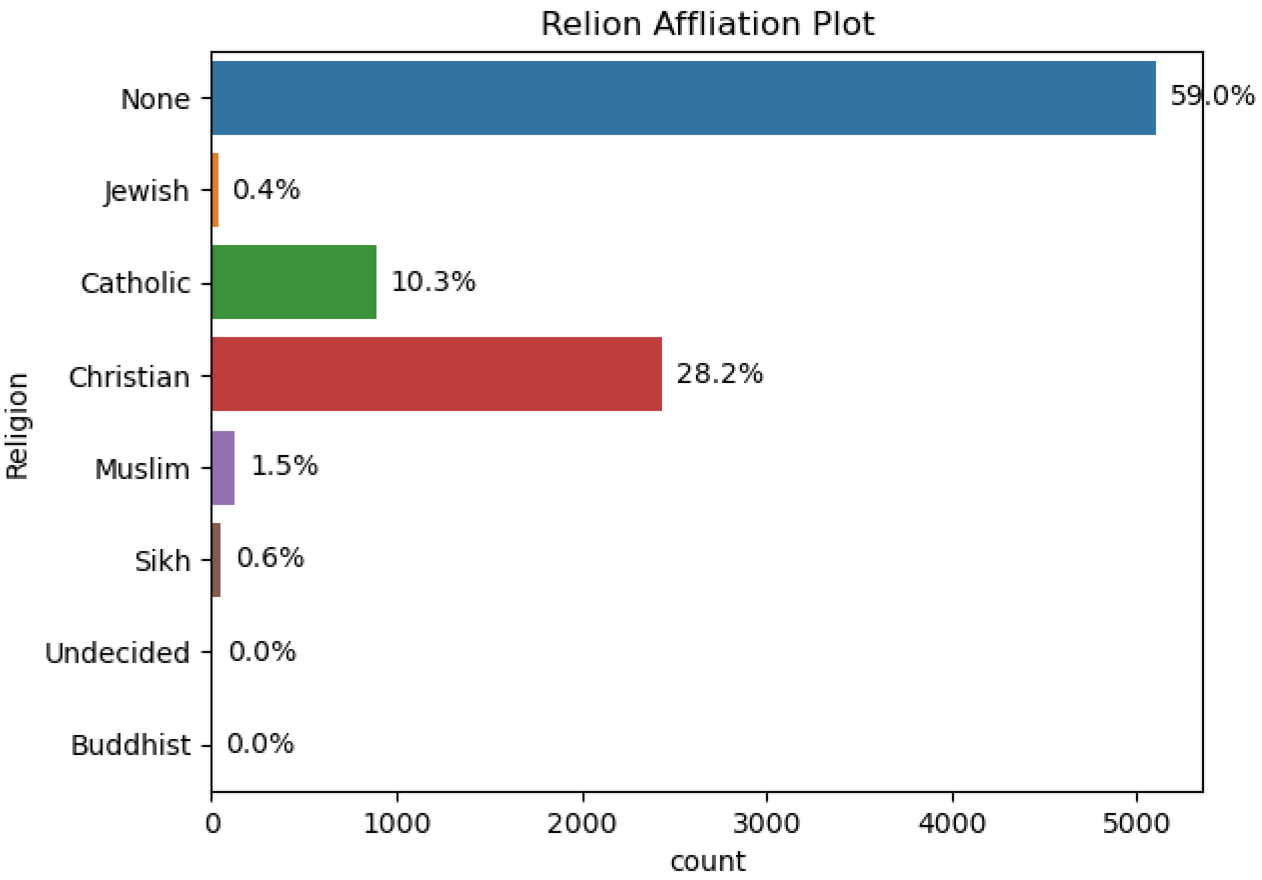


In [99]:

```
# Create a plot for the religion
religion_countplot = sns.countplot(data=new_data, y='Religion')

# Get the total frequency of each category
total_frequency = len(new_data['Religion'])

# Loop through each bar in the plot
for p in religion_countplot.patches:
    # Get the percentage value for the bar
    percentage = p.get_width() / total_frequency * 100

    # Add the percentage as text next to the bar
    religion_countplot.annotate(f'{percentage:.1f}%', (p.get_width(), p.get_y() + p.get_height() / 2.), xytext=(5

# Give the plot a title and name the y-axis
religion_countplot.set(title='Relion Affliation Plot')
```

Out[99]:

```
[Text(0.5, 1.0, 'Relion Affliation Plot')]
```


Relion Affliation Plot
Religion
None
Jewish
Catholic
Christian
Muslim
Sikh
Undecided
Buddhist
59.0%
0.4%
10.3%
28.2%
1.5%
0.6%
0.0%
0.0%
0
1000
2000
3000
4000
5000
count