Notebook

March 17, 2025

1 *Merged Jupyter Notebook*

from file: PMRP_1

2 OM CHOKSI 23AIML010 PMRP DAY 1 ASSIGNMENT 1

2.1 Question 1

Separate the given list based on the data types. List1 = [“Aakash”, 90, 77, “B”, 3.142,12]

[2]:
```
List1 = ["Aakash", 90, 77, "B", 3.142,12]
string = []
inte = []
flo = []
for i in List1:
    if type(i) == str:
        string.append(i)
    elif type(i) == int:
        inte.append(i)
    else:
        flo.append(i)
print(f"strings are {string}\ninteger are {inte}\nfloat are {flo}")
```

```
strings are ['Aakash', 'B']
integer are [90, 77, 12]
float are [3.142]
```

2.2 Question 2

Consider you are collecting data from students on their heights (in cms) containing numbers as 140,145,153, etc. Use Numpy library and randomly generate 50 such numbers in the range 150 to 180. Which data type would you use list or array to store such data? Calculate measures of central tendency of this data stored in list as well as array.

[3]:
```
import numpy as np
import statistics as st

height = np.random.randint(150,180,50)
```

```
print(height)
print(type(height))


mean=np.mean(height)
median=np.median(height)
mode=st.mode(height)
std=np.std(height)
var=np.var(height)

print(f"Mean of given data: {mean}")
print(f"Median of given data: {median}")
print(f"Mode of given data: {mode}")
print(f"Standard deviation of given data: {std}")
print(f"Variance of given data: {var}")
```

```
[155 153 155 177 169 174 154 156 161 158 153 158 156 155 153 155 170 164
 165 160 173 155 152 175 172 175 172 170 163 153 161 177 160 177 165 151
 152 171 156 169 153 178 173 172 157 171 172 170 155 152]
<class 'numpy.ndarray'>
Mean of given data: 163.06
Median of given data: 161.0
Mode of given data: 155
Standard deviation of given data: 8.755364070100113
Variance of given data: 76.6564
```

2.3 OPTIONAL QUESTION

find mode of given range using maths basic formulas

[5]:
```
import numpy as np

height = np.random.randint(150, 180, 50)


freq = {}
mode = None
max_count = 0

for h in height:
    if h in freq:
        freq[h] += 1
    else:
        freq[h] = 1

for key in freq:
    if freq[key] > max_count:
        max_count = freq[key]
```

```
        mode = key

print("Mode:", mode)
```

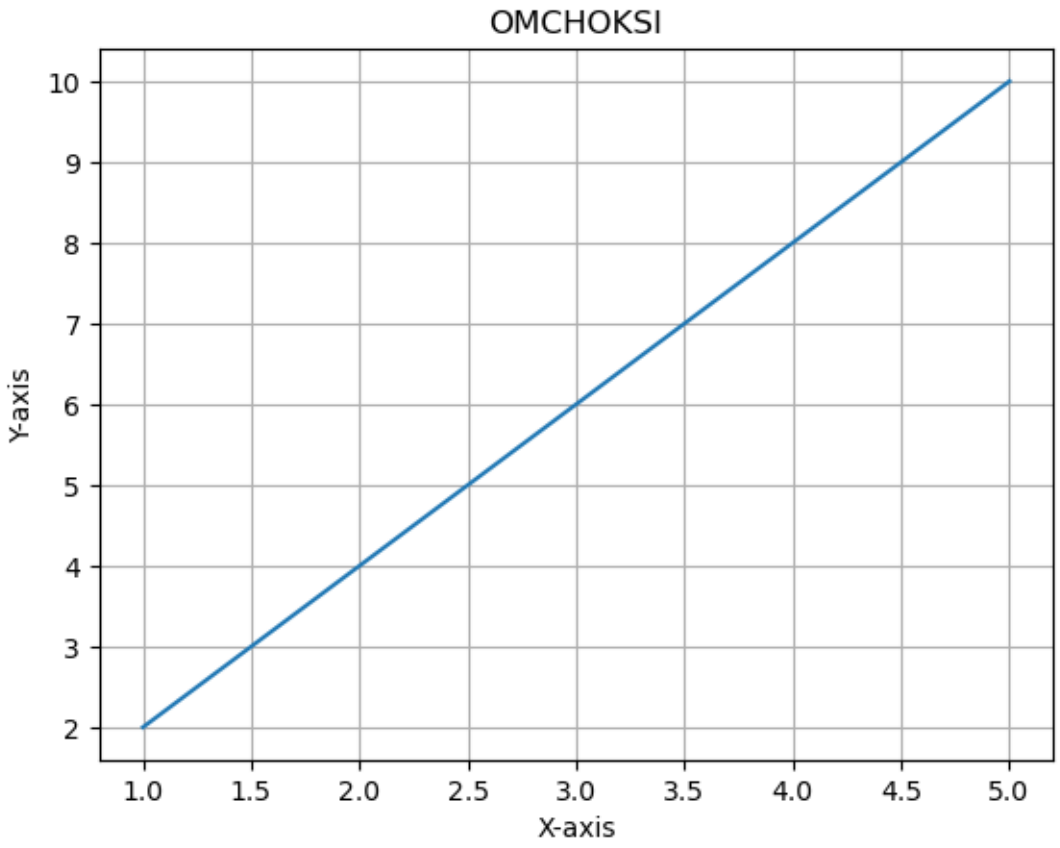
```
Mode: 154
```

2.4 Question 3

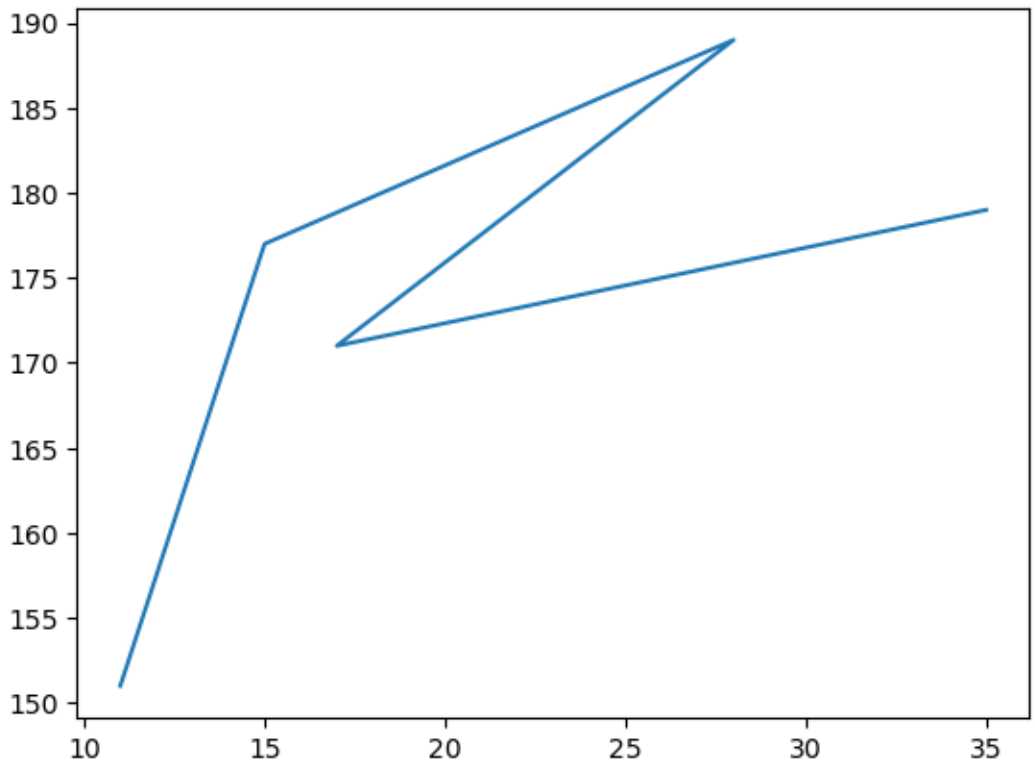
Part 1:-

Create the function that will plot simple line chart for any given data.

[7]:
```
import matplotlib.pyplot as plt
def linePLOT(x,y):
  plt.plot(x,y)
  plt.xlabel("X-axis")
  plt.ylabel("Y-axis")
  plt.title("OMCHOKSI")
  plt.grid(True)
  plt.show()

x=[1,2,3,4,5]
y=[2,4,6,8,10]
linePLOT(x,y)

data1 = np.random.randint(1, 50, 5)
data2 = np.random.randint(150, 200, 5)
plt.plot(data1, data2)
```



[7]: [<matplotlib.lines.Line2D at 0x19632b65d90>]



2.5 Question 3

Part 2:-

Create the recursive function for finding out factorial of a given number

[10]:
```
def fact(n):
    if n == 1:
        return 1
    return n * fact(n - 1)

n = int(input())
print(fact(n))
```

```
10
```

```
3628800
```

2.6 Question 3

Part 3:-

Create generator function for Fibonacci series and print out first 10 numbers.

[13]:
```
def fibonacci_generator(n):
    x, y = 0, 1
    for _ in range(n):
        yield x
        x,y=y,x+y
fib_gen = fibonacci_generator(10)
for num in fib_gen:
    print(num)
```

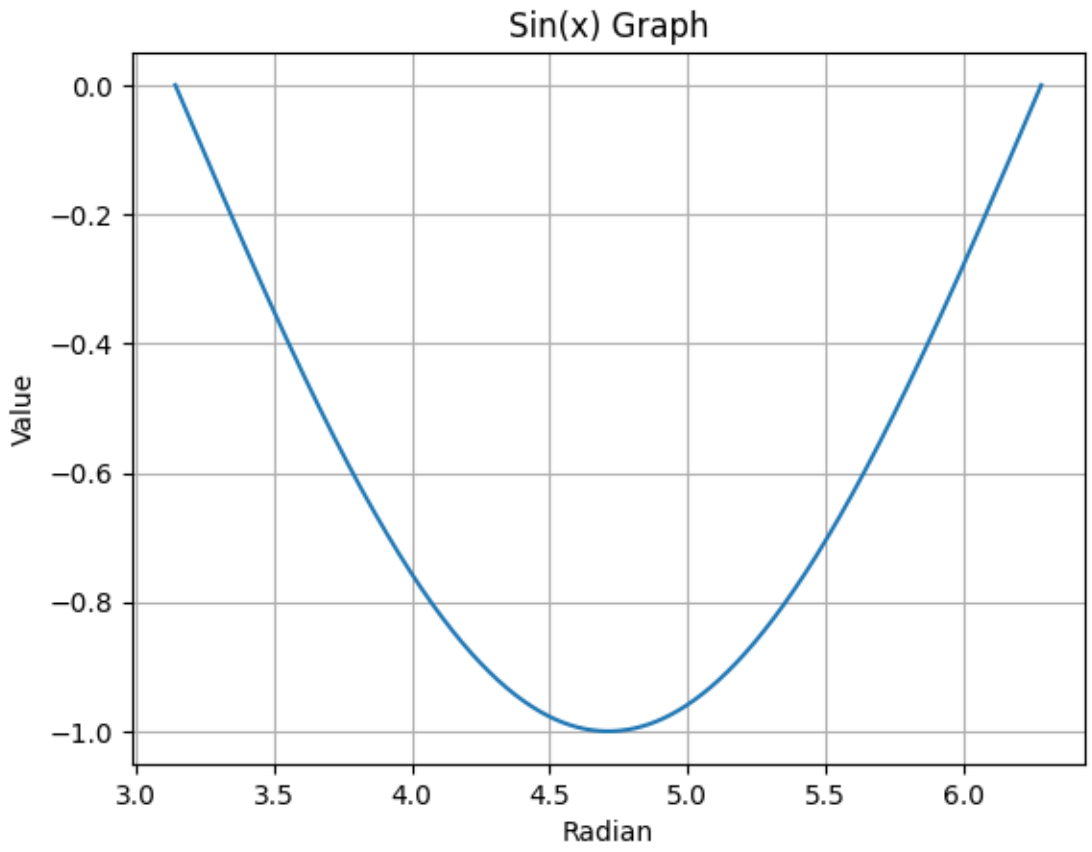
```
0
1
1
2
3
5
8
13
21
34
```

2.7 Question 3

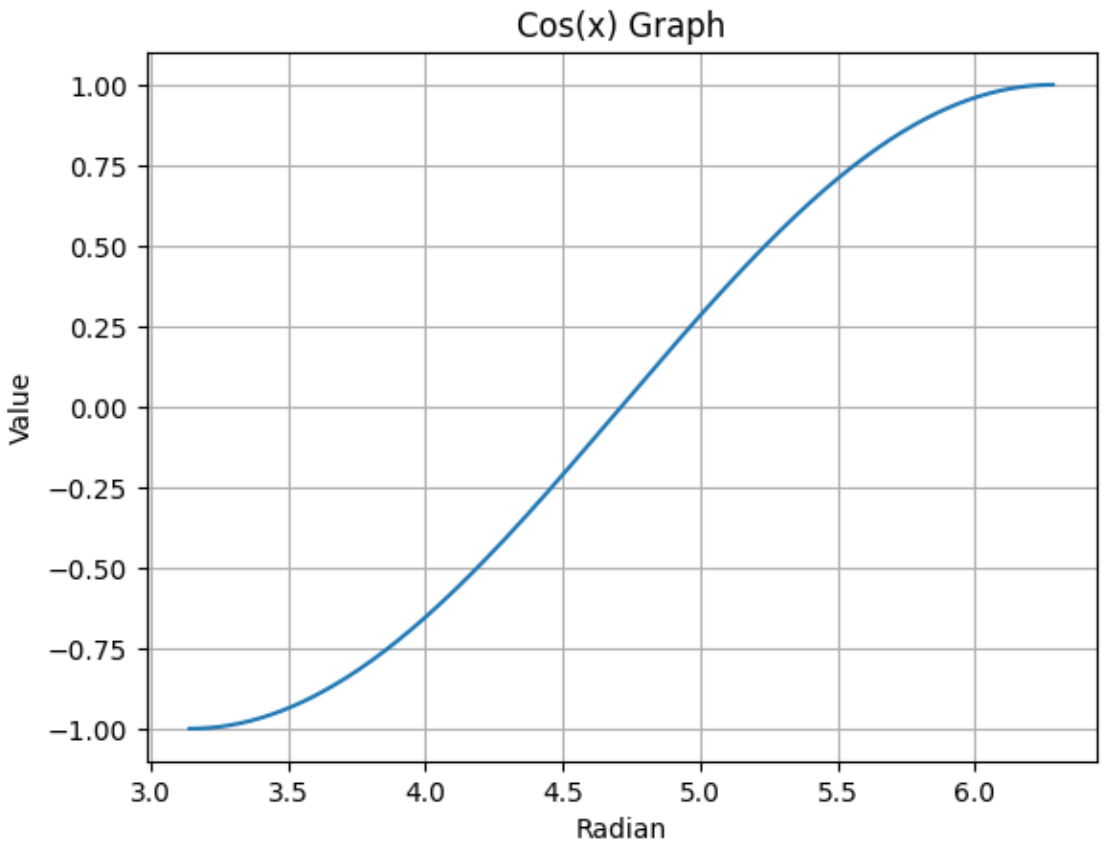
Part 4:-

Plot the graphs for trigonometric functions sin, cos, tan, cot, sec & cosec for the values pi to 2pi.

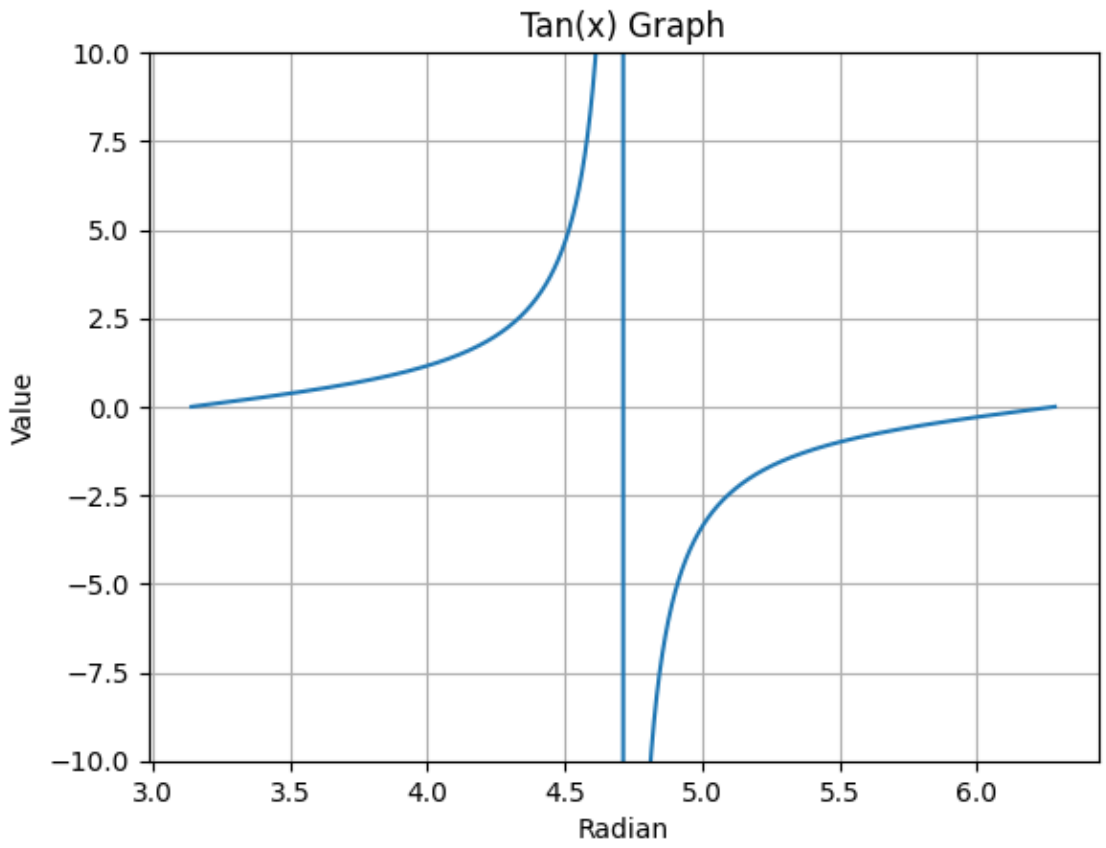
[57]:
```
import math
x = np.linspace(math.pi, 2 * math.pi, 10000)
y = np.sin(x)
plt.grid()
plt.xlabel("Radian")
plt.ylabel("Value")
plt.title("Sin(x) Graph")
plt.plot(x, y)
plt.show()
```



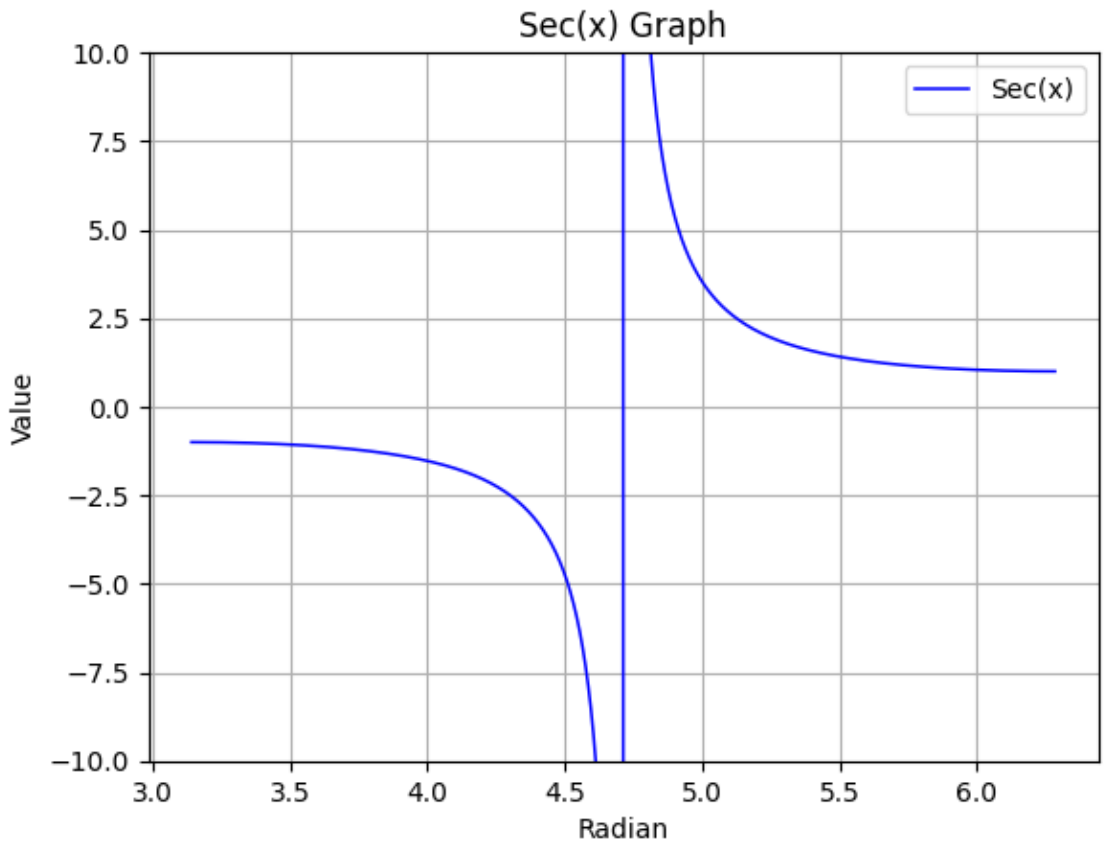
[58]:
```
import math
x = np.linspace(math.pi, 2 * math.pi, 10000)
y = np.cos(x)
plt.grid()
plt.xlabel("Radian")
plt.ylabel("Value")
plt.title("Cos(x) Graph")
plt.plot(x, y)
plt.show()
```



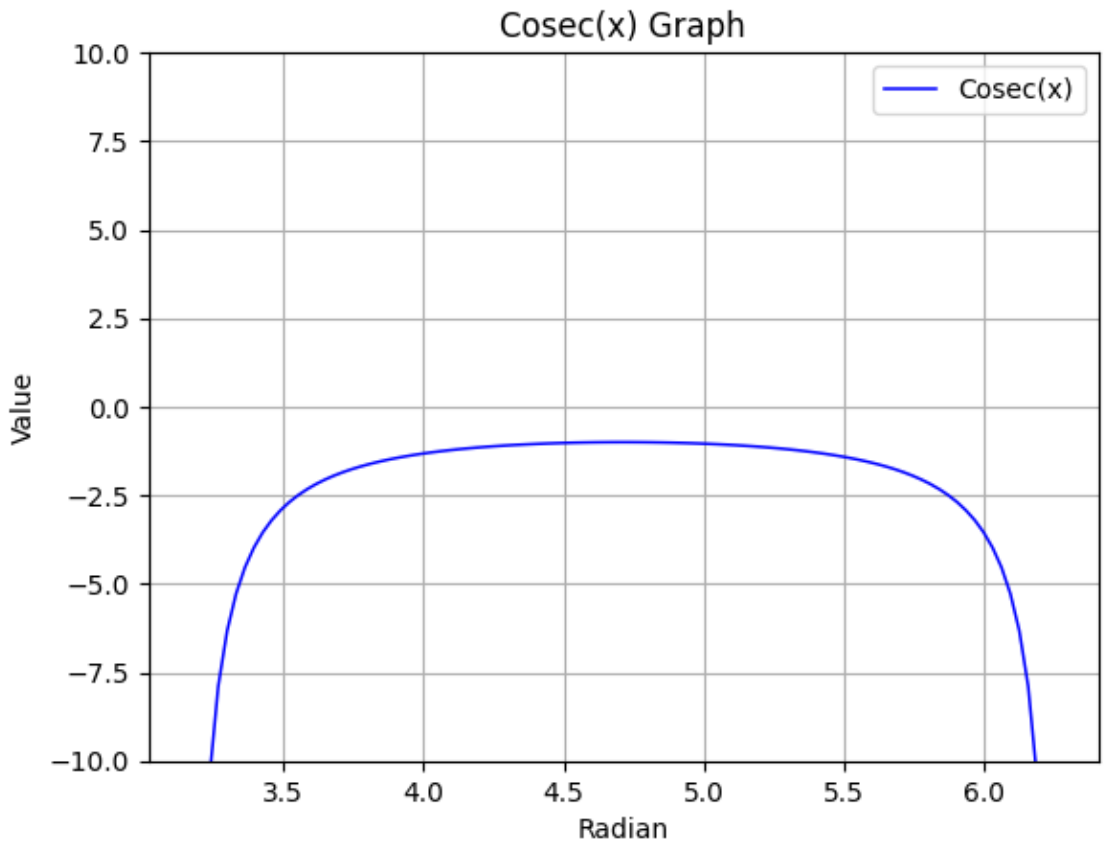
[59]:
```
import math
x = np.linspace(math.pi, 2 * math.pi, 10000)
y = np.tan(x)
plt.grid()
plt.xlabel("Radian")
plt.ylabel("Value")
plt.title("Tan(x) Graph")
plt.ylim(-10, 10)
plt.plot(x, y)
plt.show()
```



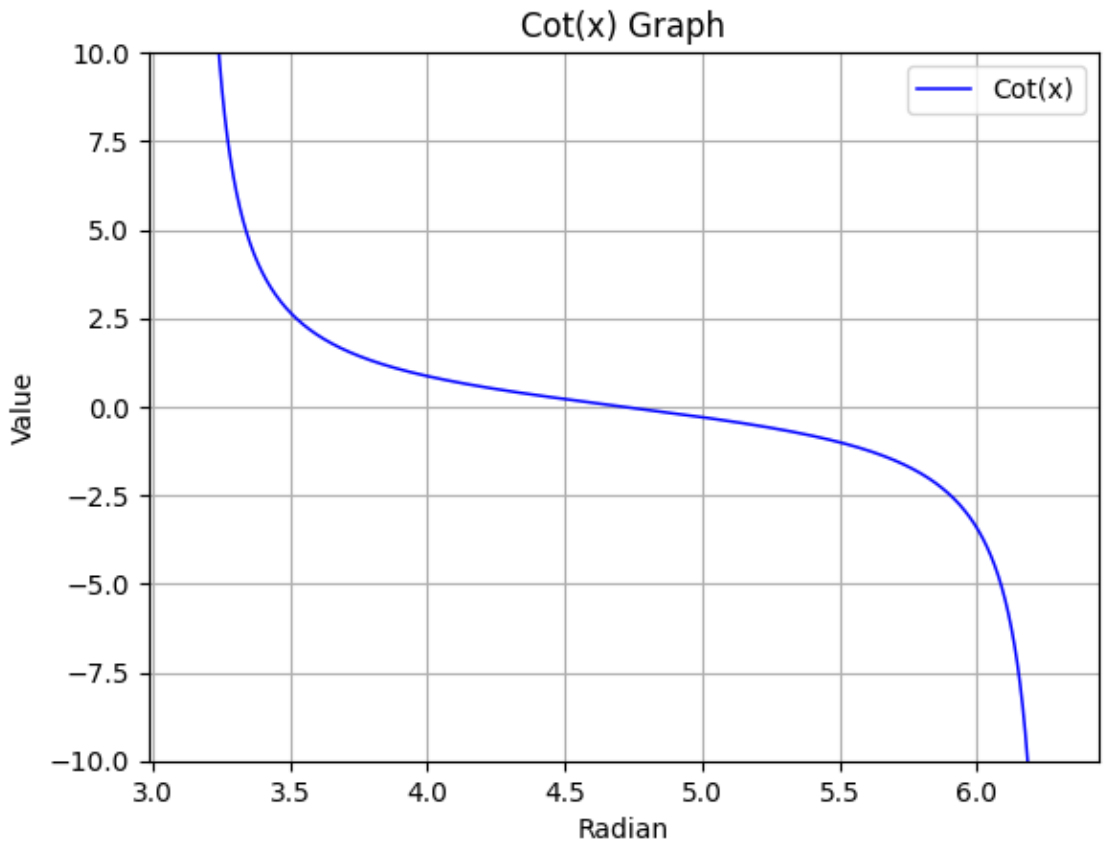
[60]:
```
import math
x = np.linspace(math.pi, 2 * math.pi, 10000)
y = 1 / np.cos(x)
y[np.abs(np.cos(x)) < 1e-5] = np.nan
plt.grid()
plt.xlabel("Radian")
plt.ylabel("Value")
plt.title("Sec(x) Graph")
plt.plot(x, y, linewidth = 1.1, color = 'blue', label = "Sec(x)")
plt.legend()
plt.ylim(-10, 10)
plt.show()
```



[61]:
```
import math
x = np.linspace(math.pi, 2 * math.pi, 100)
y = 1 / np.sin(x)
y[np.abs(np.sin(x)) < 1e-5] = np.nan
plt.grid()
plt.xlabel("Radian")
plt.ylabel("Value")
plt.title("Cosec(x) Graph")
plt.plot(x, y, linewidth = 1.1, color = 'blue', label = "Cosec(x)")
plt.legend()
plt.ylim(-10, 10)
plt.show()
```



[62]:
```
import math
x = np.linspace(math.pi, 2 * math.pi, 10000)
y = 1 / np.tan(x)
y[np.abs(np.tan(x)) < 1e-5] = np.nan
plt.grid()
plt.xlabel("Radian")
plt.ylabel("Value")
plt.title("Cot(x) Graph")
plt.plot(x, y, linewidth = 1.1, color = 'blue', label = "Cot(x)")
plt.legend()
plt.ylim(-10, 10)
plt.show()
```



2.8 Question 4

Consider you want create dataset with ages of people in your surroundings. Use input method to ask user their age, store those ages in appropriate data type. Apply error handling that will not accept more than 130 or less than 0 inputs, raise appropriate prompts to guide users.

[63]:
```
def coll():
    ages = []

    while True:
        usr = input("Enter Age of the person\nEnter q to exit")

        if usr == 'q':
            break
        try:
            usr = int(usr)
            if(usr < 0 or usr > 130):
                print("Invalid input")
            else:
                ages.append(usr)
```

```
        except ValueError:
            print("Invallid input")
    return ages
use = coll()
print(use)
```

```
Enter Age of the person
Enter q to exit 5
Enter Age of the person
Enter q to exit q
```

```
[5]
```

2.9 Question 5

Create class as Employees with inputs as name, department and salary. Salary should be encapsulated.

[64]:
```
class Employees:
    def __init__(self, name, department, salary):
        self.name = name
        self.department = department
        self.__salary = salary
    def setsalary(self, slary):
        self.__salary = slary
    def getsalary(self):
        return self.__salary
    def print(self):
        print(f"The Employee name is {self.name}\nThe department is {self.
 ↪department}\nThe salary is {self.__salary}")
e1 = Employees("Yash", "AIML", 100000000)
e1.print()
```

```
The Employee name is Yash
The department is AIML
The salary is 100000000
```

2.10 Question 6

Create two 3d arrays as matrices. Perform matrix operations (Addition, Multiplication, dot product, inverse, determinant) on those matrices. Explain identity matrix, multiply each matrix with identity matrix and record the observation. (All operations should be done with Numpy library)

[65]:
```
import numpy as np

arr1 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])
arr2 = np.array([[[9, 10], [11, 12]], [[13, 14], [15, 16]]])

print("Matrix Addition:\n", arr1 + arr2)
```

```
print("\nElement-wise Multiplication:\n", arr1 * arr2)
print("\nDot Product:\n", np.matmul(arr1, arr2))
print("\nInverse of arr1:\n", np.linalg.inv(arr1))
print("\nDeterminants of arr1:\n", np.linalg.det(arr1))
print("\nIdentity Matrix:\n", np.eye(2))
print("\narr1 multiplied with Identity Matrix:\n", np.array([np.dot(np.eye(2),␣
 ↪mat) for mat in arr1]))
```

```
Matrix Addition:
 [[[10 12]
  [14 16]]

 [[18 20]
  [22 24]]]

Element-wise Multiplication:
 [[[  9  20]
  [ 33  48]]

 [[ 65  84]
  [105 128]]]

Dot Product:
 [[[ 31  34]
  [ 71  78]]

 [[155 166]
  [211 226]]]

Inverse of arr1:
 [[[-2.   1. ]
  [ 1.5 -0.5]]

 [[-4.   3. ]
  [ 3.5 -2.5]]]

Determinants of arr1:
 [-2. -2.]

Identity Matrix:
 [[1. 0.]
 [0. 1.]]

arr1 multiplied with Identity Matrix:
 [[[1. 2.]
  [3. 4.]]

 [[5. 6.]
```

```
[7. 8.]]]
```

from file: PMRP_2

23AIML010 OM CHOKSI PMRP ASSIGNMENT 2

[46]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Find out the outliers in each numerical column
2. Find out gender distribution in this data.
3. What is average daily usage of data? Explore gender wise and device wise variation in average usage of data.
4. Which device have highest popularity based on Age and Gender?

WE WILL LOAD CSV FILE AND GET BASIC DESCRIPTION OF DATA

[47]:
```
data1 =  pd.read_csv("data.csv")
data.head(),data.describe(),data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

[47]:
```
(   Age Sex      BP Cholesterol  Na_to_K   Drug
 0   23   F    HIGH        HIGH   25.355  drugY
 1   47   M     LOW        HIGH   13.093  drugC
 2   47   M     LOW        HIGH   10.114  drugC
 3   28   F  NORMAL        HIGH    7.798  drugX
 4   61   F     LOW        HIGH   18.043  drugY,
                Age     Na_to_K
 count  200.000000  200.000000
 mean    44.315000   16.084485
 std     16.544315    7.223956
 min     15.000000    6.269000
 25%     31.000000   10.445500
 50%     45.000000   13.936500
 75%     58.000000   19.380000
```

```
 max     74.000000   38.247000,
 None)
```

Find out the outliers in each numerical column using pandas

[48]:
```
def detect_outliers(data1, column):
    Q1 = data1[column].quantile(0.25)
    Q3 = data1[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = data1[(data1[column] < lower_bound) | (data1[column] >␣
 ↪upper_bound)]
    return outliers


numerical_columns = data1.select_dtypes(include=np.number).columns

outliers_dict = {col: detect_outliers(data1, col) for col in numerical_columns}

for col, outliers in outliers_dict.items():
    print(f"Outliers in {col}:\n", outliers)
```

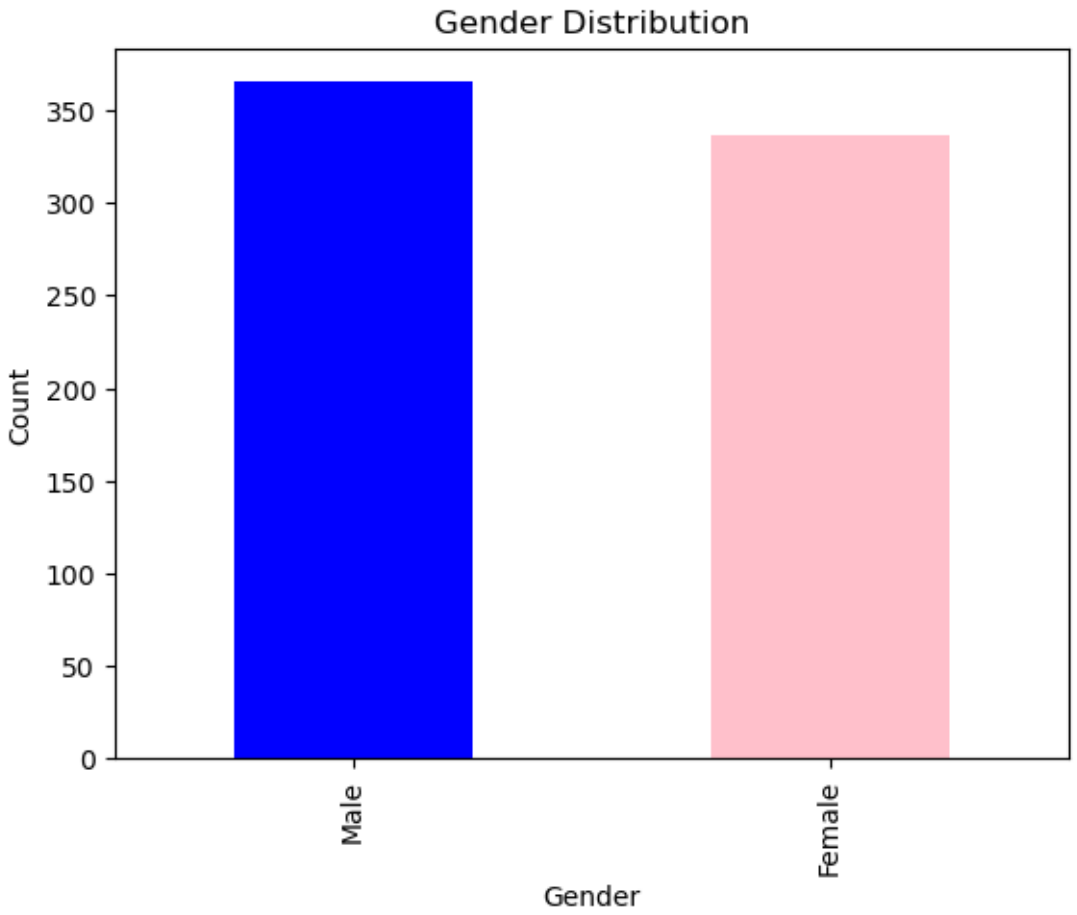
```
Outliers in User ID:
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
Outliers in App Usage Time (min/day):
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
Outliers in Screen On Time (hours/day):
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
Outliers in Battery Drain (mAh/day):
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
Outliers in Number of Apps Installed:
```

```
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
Outliers in Data Usage (MB/day):
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
Outliers in Age:
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
Outliers in User Behavior Class:
 Empty DataFrame
Columns: [User ID, Device Model, Operating System, App Usage Time (min/day),
Screen On Time (hours/day), Battery Drain (mAh/day), Number of Apps Installed,
Data Usage (MB/day), Age, Gender, User Behavior Class]
Index: []
```

2. Find out gender distribution in this data.

[49]:
```
gender_distribution = data1['Gender'].value_counts()
print("Gender Distribution:")
print(gender_distribution)

gender_distribution.plot(kind='bar', color=['blue', 'pink'], title='Gender␣
 ↪Distribution')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()
```

```
Gender Distribution:
Gender
Male      365
Female    336
Name: count, dtype: int64
```

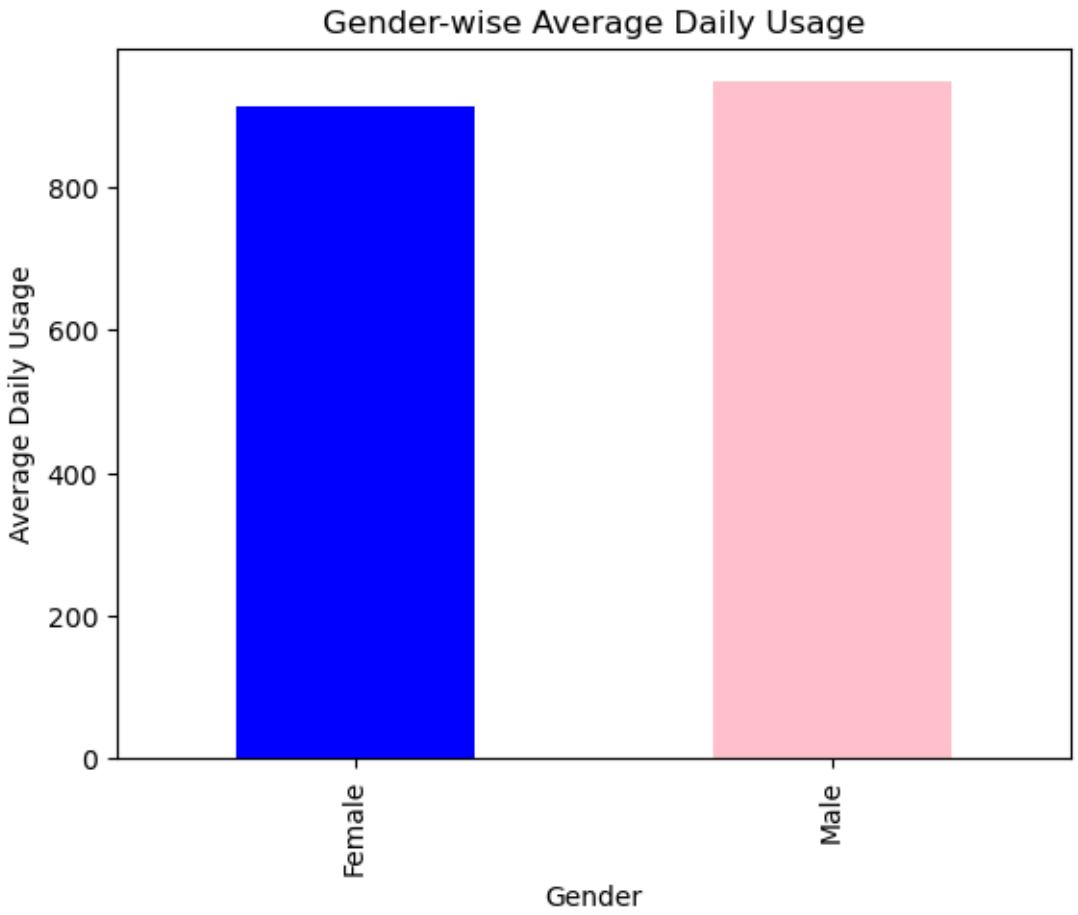


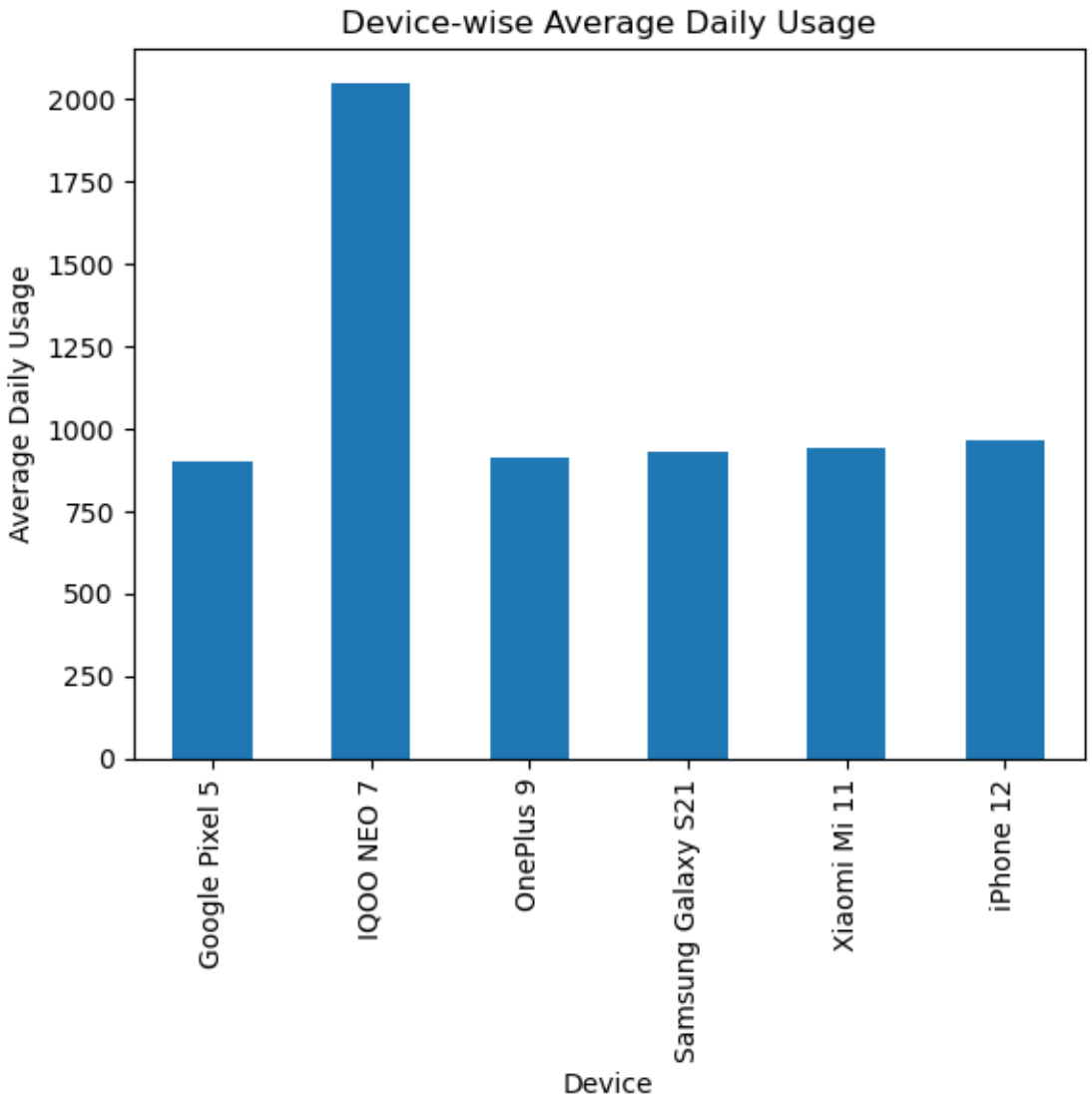
3. What is average daily usage of data? Explore gender wise and device wise variation in average usage of data.

[50]:
```
average_daily_usage = data1['Data Usage (MB/day)'].mean()
print(f"Overall Average Daily Usage: {average_daily_usage}")

gender_avg_usage = data1.groupby('Gender')['Data Usage (MB/day)'].mean()
device_avg_usage = data1.groupby('Device Model')['Data Usage (MB/day)'].mean()

print("Gender-wise Average Daily Usage:")
print(gender_avg_usage)

print("Device-wise Average Daily Usage:")
print(device_avg_usage)

gender_avg_usage.plot(kind='bar', color=['blue', 'pink'], title='Gender-wise␣
 ↪Average Daily Usage')
plt.xlabel('Gender')
```

```
plt.ylabel('Average Daily Usage')
plt.show()

device_avg_usage.plot(kind='bar', title='Device-wise Average Daily Usage')
plt.xlabel('Device')
plt.ylabel('Average Daily Usage')
plt.show()
```

```
Overall Average Daily Usage: 931.3380884450785
Gender-wise Average Daily Usage:
Gender
Female    914.321429
Male      947.002740
Name: Data Usage (MB/day), dtype: float64
Device-wise Average Daily Usage:
Device Model
Google Pixel 5         897.704225
IQOO NEO 7            2048.000000
OnePlus 9              911.120301
Samsung Galaxy S21     931.872180
Xiaomi Mi 11           940.164384
iPhone 12              965.506849
Name: Data Usage (MB/day), dtype: float64
```


Gender-wise Average Daily Usage
Average Daily Usage
0
200
400
600
800
Female
Male
Gender

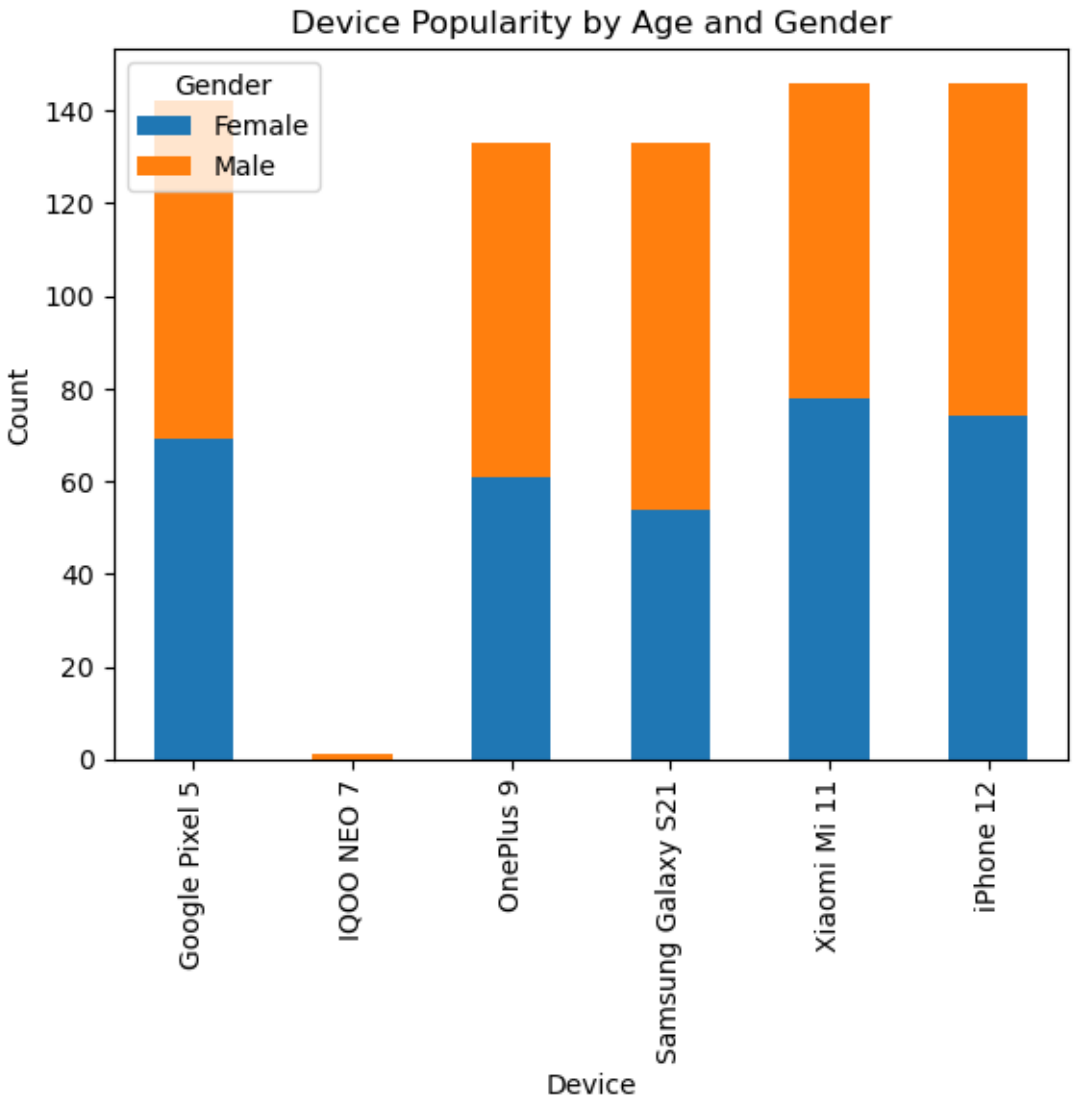


4. Which device have highest popularity based on Age and Gender?

[51]:
```
pAgedevice = data1.groupby(['Age', 'Gender'])['Device Model'].agg(lambda x: x.
  ↪mode().iloc[0])
pAgedevice

device_popularity = data1.groupby(['Device Model', 'Gender'])['Age'].count()
highest_popularity = device_popularity.idxmax()
print(f"Device with highest popularity based on Age and Gender:␣
  ↪{highest_popularity}")

popularity_pivot = data1.pivot_table(index='Device Model', columns='Gender',␣
  ↪values='Age', aggfunc='count')
```

```
popularity_pivot.plot(kind='bar', stacked=True, title='Device Popularity by Age␣
↪and Gender')
plt.xlabel('Device')
plt.ylabel('Count')
plt.show()
```

```
Device with highest popularity based on Age and Gender: ('Samsung Galaxy S21',
'Male')
```



CLASSWORK PMRP

Plot Distribution curve for Age along with histogram. Calculate Q1,Q2,Q3 and IQR without using np.percentile function. Calculate lower and upper bound values. Plot box plot as well for Age. Calculate frequency table as well for age column. Ranges for this can be in multiple of 10,

e.g. 10-20,20-30,etc..

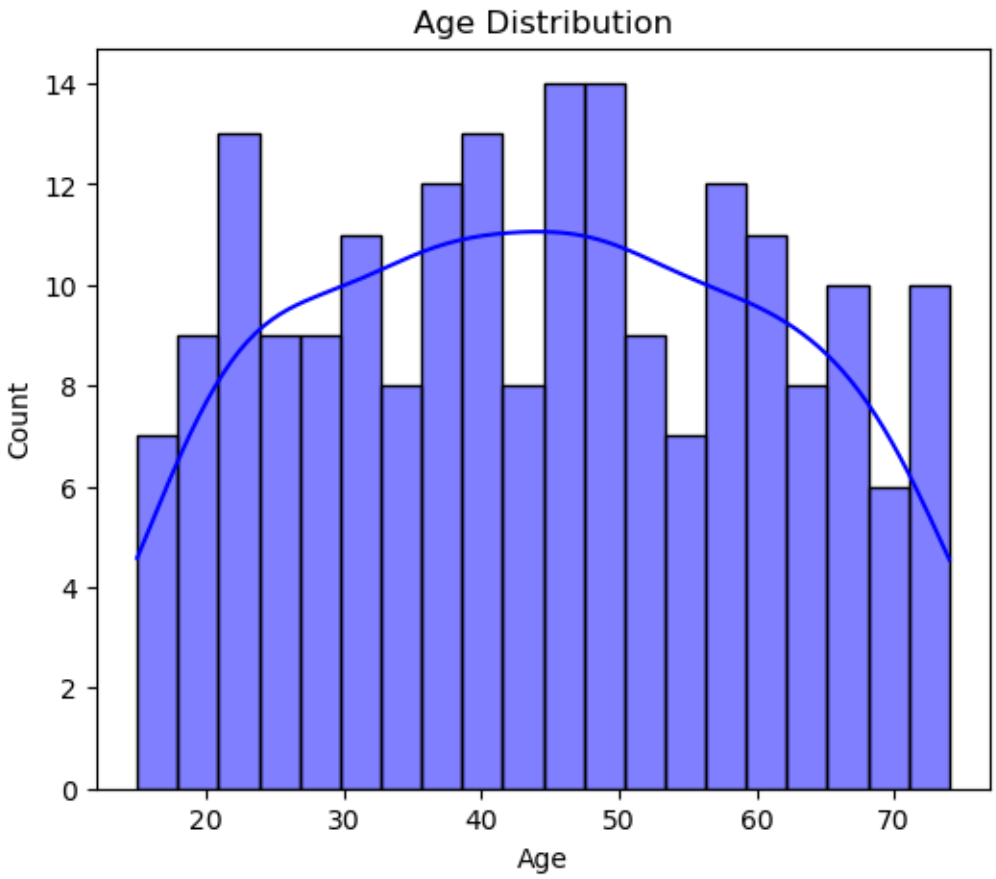
[52]:
```
data2=pd.read_csv("drug200.csv")
data2.head(),data.info(),data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
```

[52]:
```
(   Age Sex      BP Cholesterol  Na_to_K   Drug
 0   23   F    HIGH        HIGH   25.355  drugY
 1   47   M     LOW        HIGH   13.093  drugC
 2   47   M     LOW        HIGH   10.114  drugC
 3   28   F  NORMAL        HIGH    7.798  drugX
 4   61   F     LOW        HIGH   18.043  drugY,
 None,
               Age     Na_to_K
 count  200.000000  200.000000
 mean    44.315000   16.084485
 std     16.544315    7.223956
 min     15.000000    6.269000
 25%     31.000000   10.445500
 50%     45.000000   13.936500
 75%     58.000000   19.380000
 max     74.000000   38.247000)
```

1.Plot Distribution curve for Age along with histogram.

[53]:
```
plt.figure(figsize=(6, 5))
sns.histplot(data2["Age"], kde=True, color='blue',bins=20)
plt.title('Age Distribution')
plt.show()
```



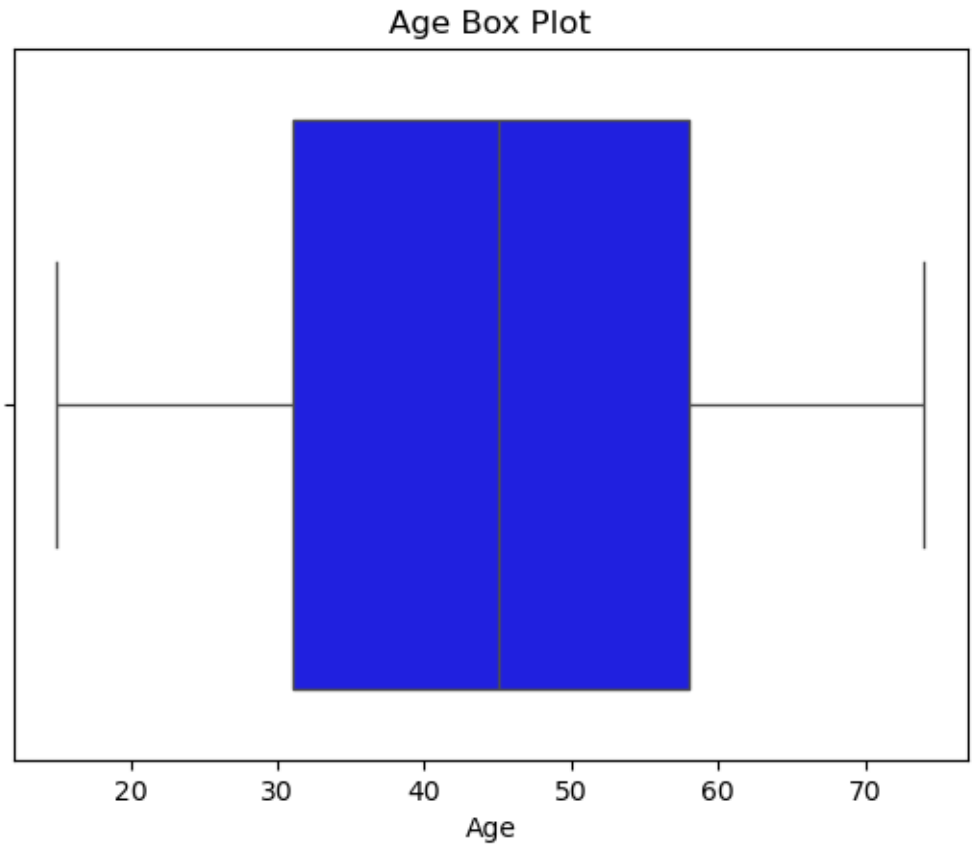
2.Calculate Q1,Q2,Q3 and IQR without using np.percentile function. Calculate lower and upper bound values.

[54]:
```
def q_vals(col):
    q1 = col.sort_values().iloc[len(col) // 4]
    q2 = col.median()
    q3 = col.sort_values().iloc[(len(col) * 3) // 4]
    iqr = q3 - q1
    lb = q1 - 1.5 * iqr
    ub = q3 + 1.5 * iqr
    return q1, q2, q3, iqr, lb, ub

q1, q2, q3, iqr, lb, ub = q_vals(data2["Age"])

print(f"Q1: {q1}, Q2: {q2}, Q3: {q3}, IQR: {iqr}, LB: {lb}, UB: {ub}")
```

```
Q1: 31, Q2: 45.0, Q3: 58, IQR: 27, LB: -9.5, UB: 98.5
```

Plot box plot as well for Age.

[55]:
```
sns.boxplot(x=data2["Age"], color='blue')
plt.title('Age Box Plot')
plt.show()
```



4.Calculate frequency table as well for age column. Ranges for this can be in multiple of 10, e.g. 10-20,20-30,etc..

[56]:
```
bins = list(range(10, data2["Age"].max() + 10, 10))
age_freq = pd.cut(data2["Age"], bins=bins).value_counts().sort_index()
print(age_freq)
```

```
Age
(10, 20]    16
(20, 30]    32
(30, 40]    39
(40, 50]    40
(50, 60]    33
(60, 70]    30
(70, 80]    10
Name: count, dtype: int64
```

SECOND DRAFT

1. What is a Gender distribution of data?
2. What percent of total population have high cholesterol & high BP?
3. What are the unique values of Drugs given in data? (df[“Drug”].unique)
4. How many people have high cholesterol before age of 30?

1. What is a Gender distribution of data?

[58]:
```
g_dist = data2['Sex'].value_counts()
print(g_dist)
```

```
Sex
M    104
F     96
Name: count, dtype: int64
```

2. What percent of total population have high cholesterol & high BP?

[59]:
```
hc_hbp = len(data2[(data2['Cholesterol'] == 'HIGH') & (data2['BP'] == 'HIGH')])
total = len(data2)
pct_hc_hbp = (hc_hbp / total) * 100
print(f"{pct_hc_hbp:.2f}%")
```

```
17.50%
```

3. What are the unique values of Drugs given in data? (df[“Drug”].unique)

[61]:
```
d_vals = data2['Drug'].unique()
print(d_vals)
```

```
['drugY' 'drugC' 'drugX' 'drugA' 'drugB']
```

4. How many people have high cholesterol before age of 30?

[62]:
```
hc_under_30 = len(data2[(data2['Cholesterol'] == 'HIGH') & (data2['Age'] < 30)])
print(hc_under_30)
```

```
26
```

[ ]:
```

```

from file: PMRP_3

OM CHOKSI 23AIML010 PMRP CLASSROOM WORK + ASSIGNMENT 3

GOOGLE CLASSROOM WORK

Load ‘tips’ dataset from seaborn library with .load_dataset(‘tips’) 1) check info about dataset using .info() 2) check statistical measures using .describe(). Write explanation about each value in notebook as comments. 3) Plot histogram for each column, find out kind of skewness. 4) What are the different ways to reduce skewness? Implement any one method and plot histogram 5) Generate covariance matrix, correlation matrix and heatmap for the dataset. Explain all values present in matrix in jupyter notebook. 6) Plot cumulative frequency polygon for ‘total bill’ column and find

out median value from graph 7) Find unique values and their value counts for each column. 8) Find out if there are any null records in data. 9) How to replace null records? 10) Drop unnecessary columns from the data.

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
# load tips dataset
tips = sns.load_dataset('tips')
tips
```

[3]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
..	…	…	…	…	…	…	…
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

[244 rows x 7 columns]

Question 1 and Question 2

check info about dataset using .info()

check statistical measures using .describe(). Write explanation about each value in notebook as comments.

[4]:
```
tips.info(),tips.describe()


'''

count: The number of non-missing values for each column.
mean: The average value of each column.
std: The standard deviation, a measure of spread or variability in the data.
min: The smallest value in each column.
25%: The first quartile (25th percentile), a measure of the lower bound of the␣
 ↪interquartile range.
50%: The median (50th percentile) of the column values.
75%: The third quartile (75th percentile), a measure of the upper bound of the␣
 ↪interquartile range.
```

```
max: The largest value in each column.

'''
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

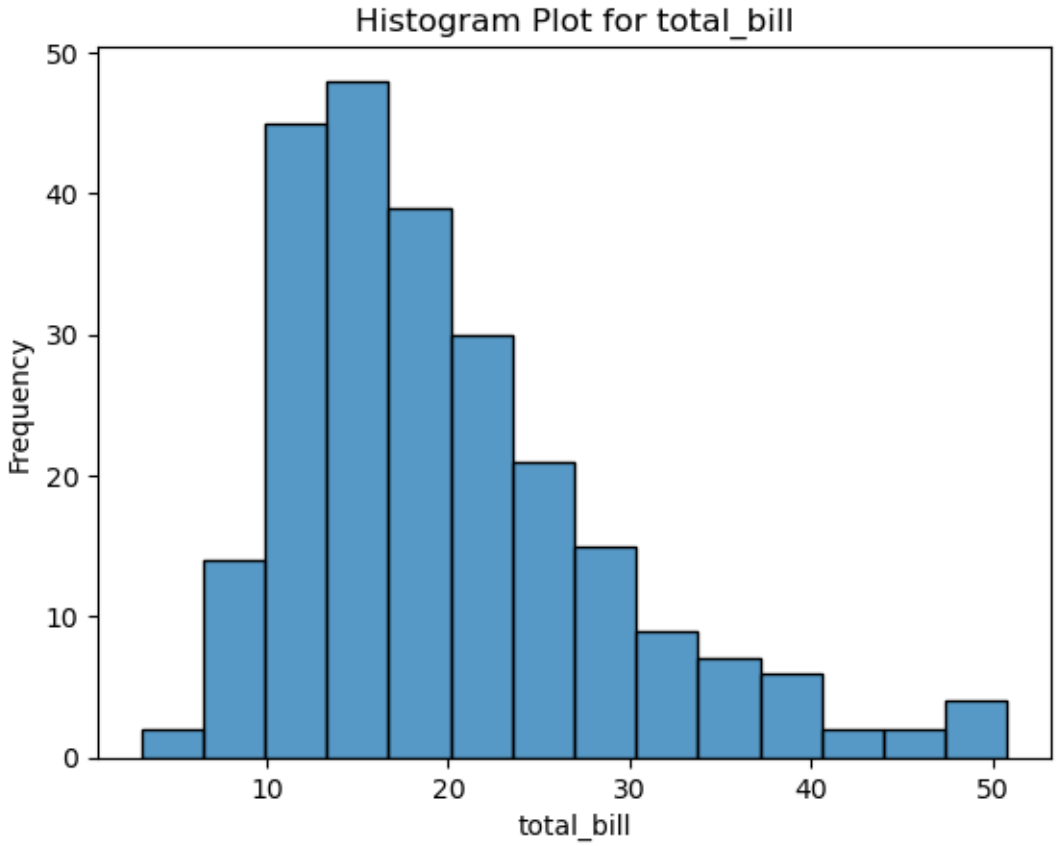
[4]:
```
'\n\ncount: The number of non-missing values for each column.\nmean: The average
value of each column.\nstd: The standard deviation, a measure of spread or
variability in the data.\nmin: The smallest value in each column.\n25%: The
first quartile (25th percentile), a measure of the lower bound of the
interquartile range.\n50%: The median (50th percentile) of the column
values.\n75%: The third quartile (75th percentile), a measure of the upper bound
of the interquartile range.\nmax: The largest value in each column.\n\n'
```

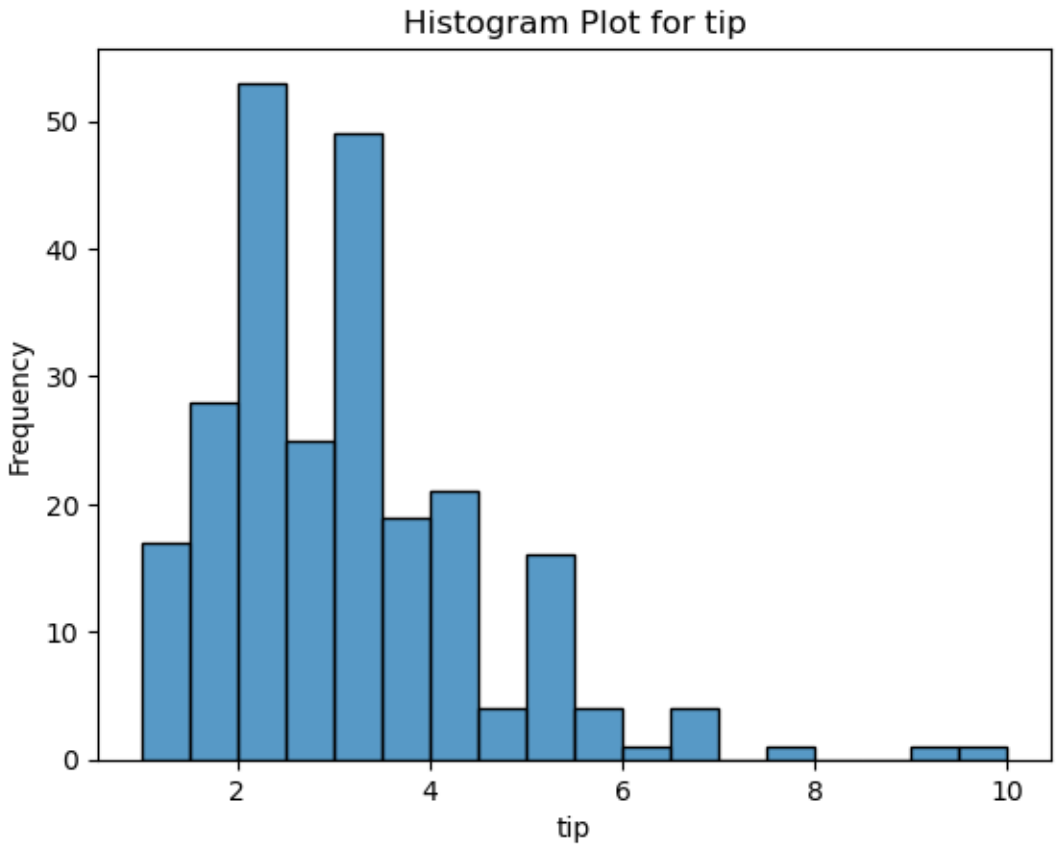
Question 3

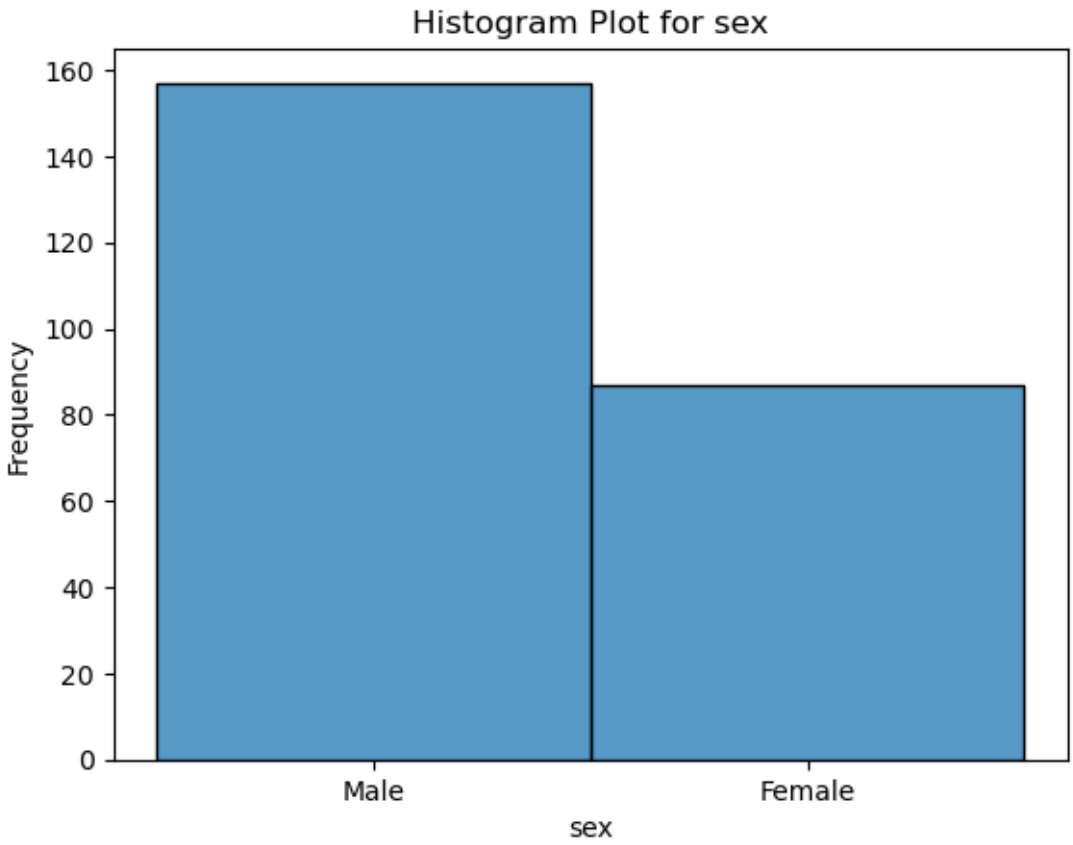
Plot histogram for each column, find out kind of skewness.

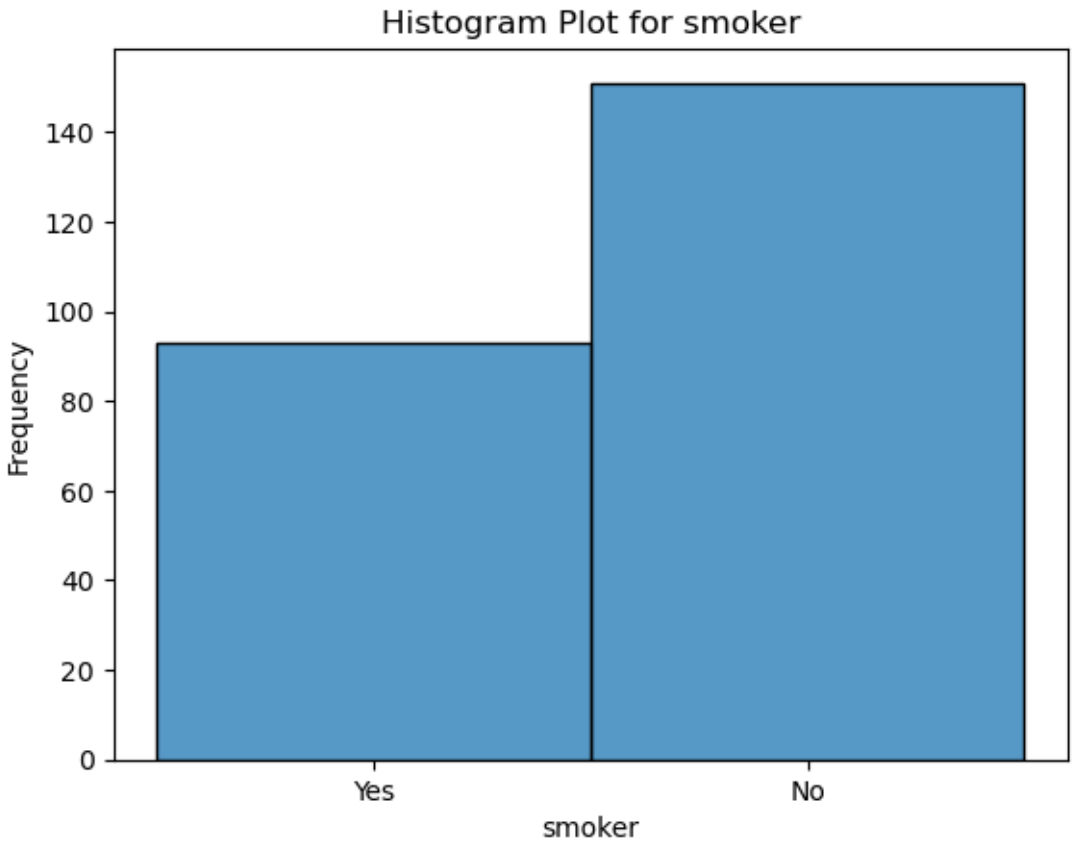
[5]:
```
Column = tips.columns
for column in Column:

    sns.histplot(tips[column])


    plt.title(f'Histogram Plot for {column}')


    plt.xlabel(column)


    plt.ylabel('Frequency')


    plt.show()
```

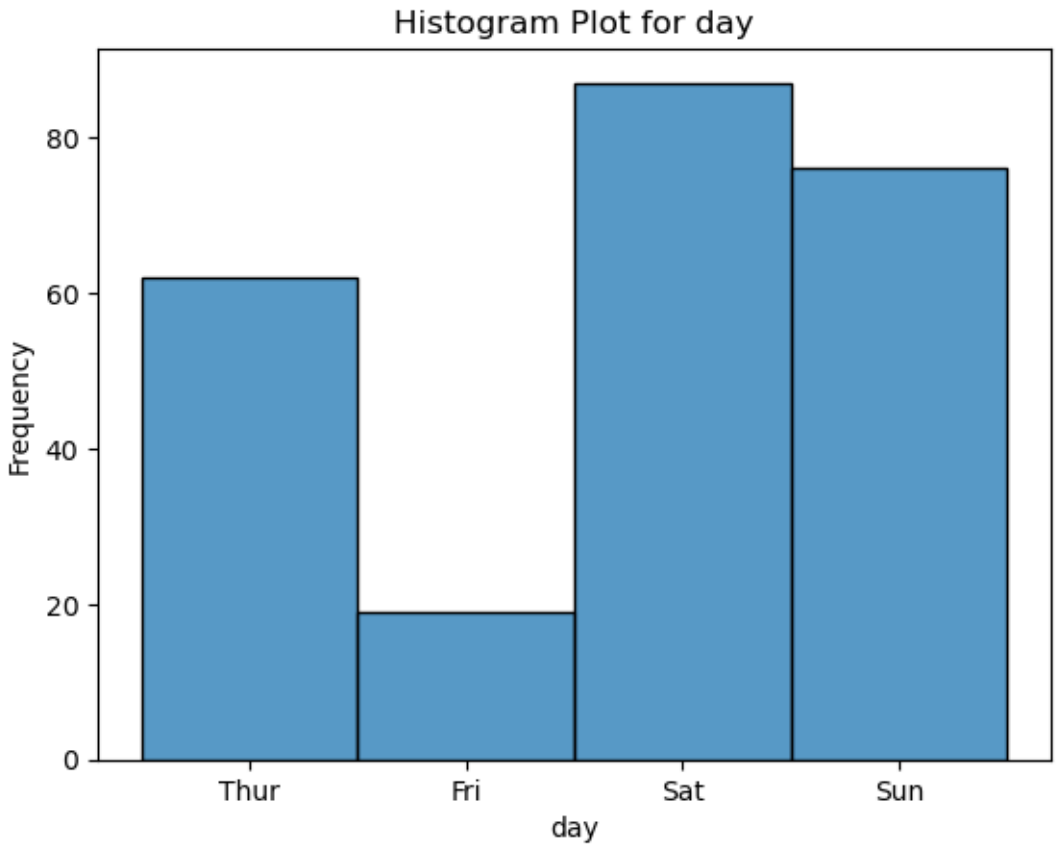
```
tips.hist(bins=15, figsize=(12, 8))
plt.tight_layout()
plt.show()
```

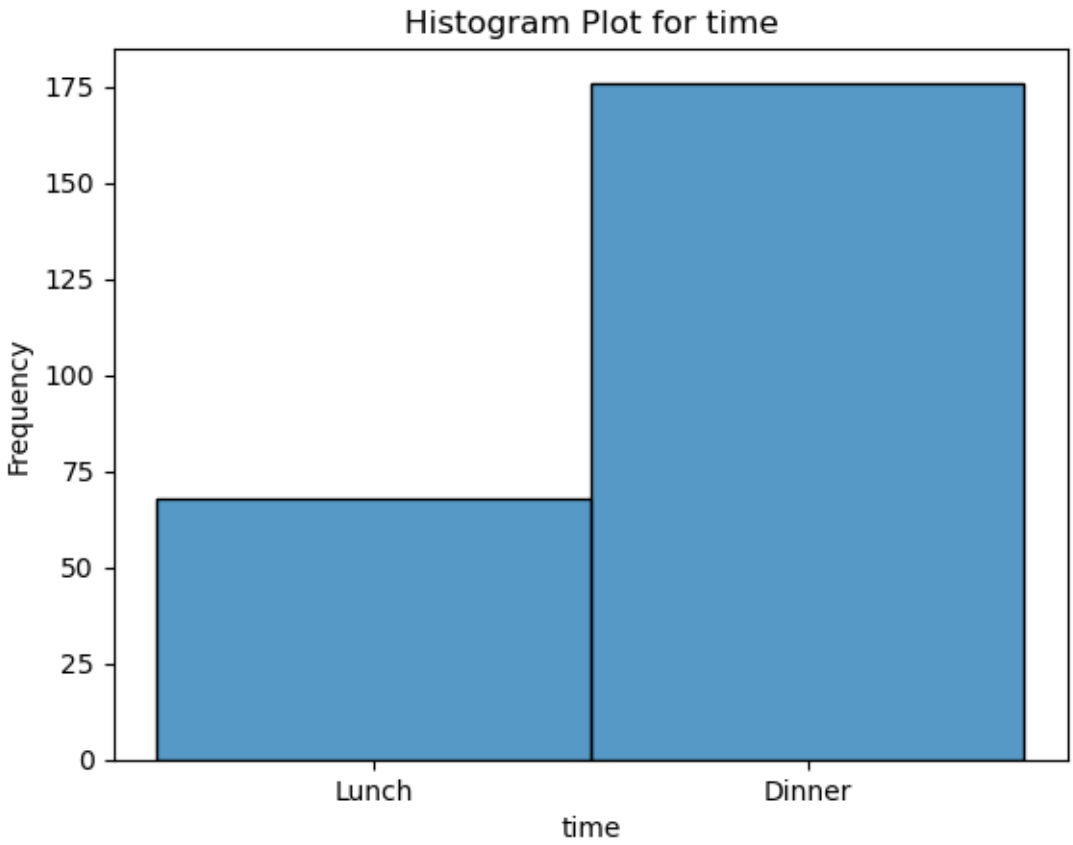


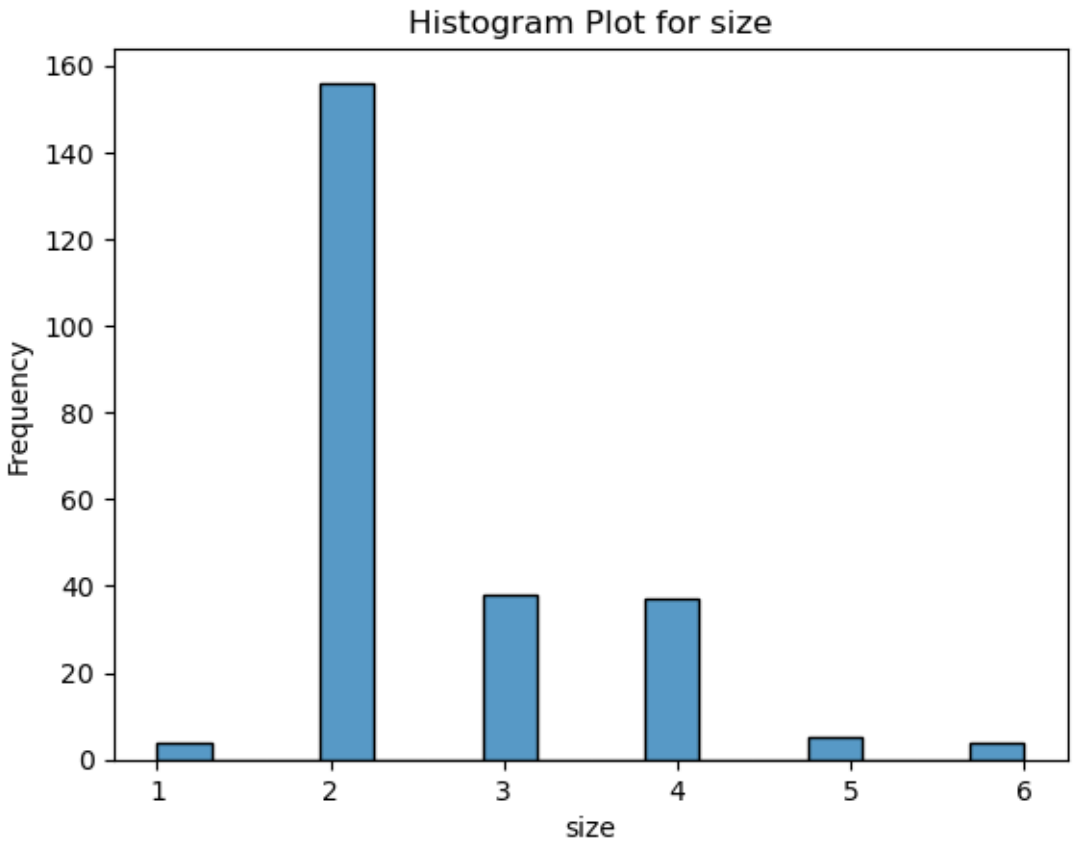

Histogram Plot for tip
Frequency
0
10
20
30
40
50
2
4
6
8
10
tip

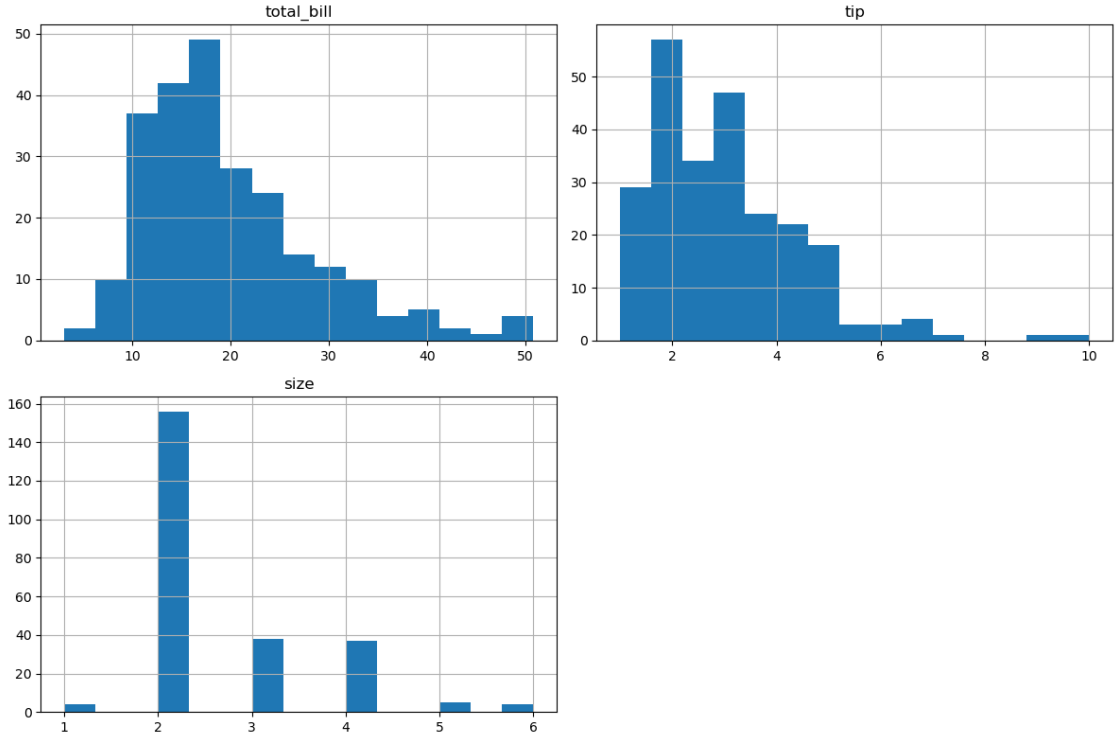

Histogram Plot for sex
160
140
120
100
80
60
40
20
0
Frequency
Male
Female
sex


Histogram Plot for smoker
Frequency
0
20
40
60
80
100
120
140
Yes
No
smoker


Histogram Plot for day
Frequency
0
20
40
60
80
Thur
Fri
Sat
Sun
day


Histogram Plot for time
175
150
125
100
75
50
25
0
Frequency
Lunch
Dinner
time

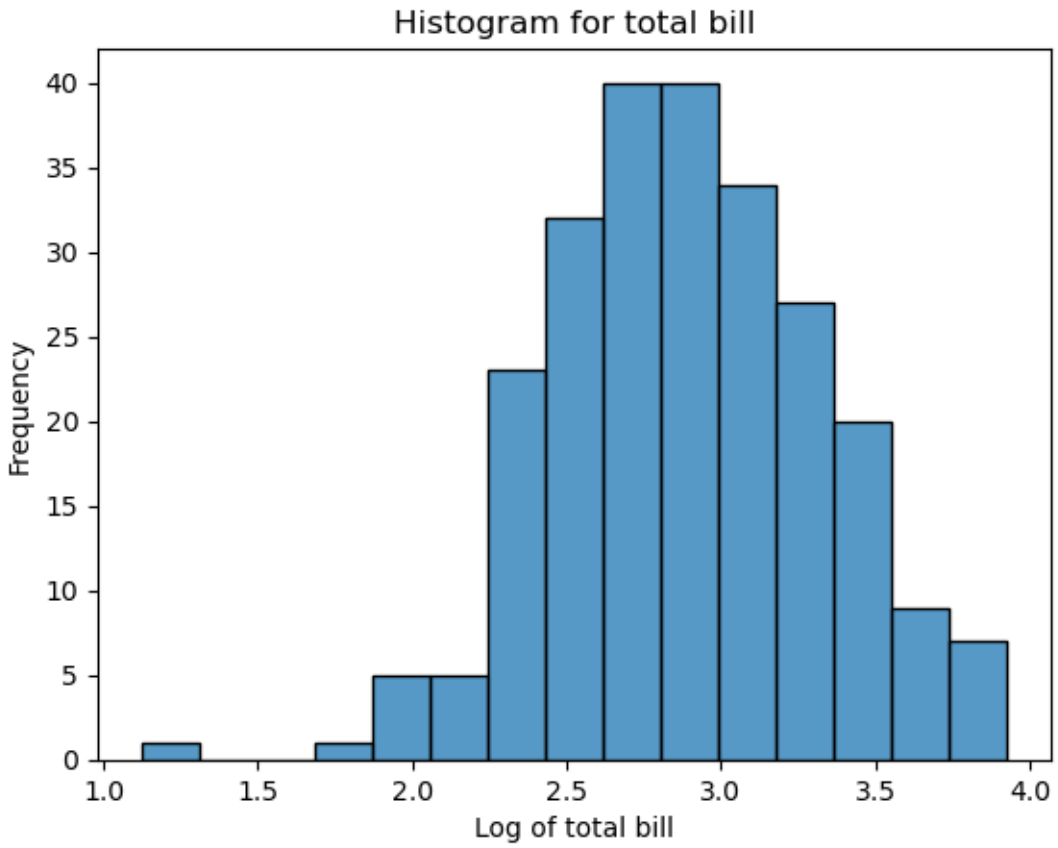

Histogram Plot for size
Frequency
size
0
20
40
60
80
100
120
140
160
1
2
3
4
5
6


total_bill
tip
size

Question 4

What are the different ways to reduce skewness? Implement any one method and plot histogram

[6]:
```
# Q4
tips['total_bill_log'] = np.log(tips['total_bill'])
sns.histplot(tips['total_bill_log'])
plt.title('Histogram for total bill')
plt.xlabel('Log of total bill')
plt.ylabel('Frequency')
plt.show()



'''
Skewness explanation:
 Positive skew: Tail is longer on the right.
 Negative skew: Tail is longer on the left.
 Symmetrical distribution: Balanced tails on both sides.


Methods to reduce skewness
Log Transformation: Reduces positive skewness by compressing large values
Formula: y = log(x)

Square Root Transformation: Compresses larger values, less aggressive than log␣
 ↪transformation
Formula: y = sqrt(x)

Cube Root Transformation: Reduces both positive and negative skewness
Formula: y = x^(1/3)

Box-Cox Transformation: Finds the best power transformation (lambda) for␣
 ↪reducing skewness
Formula: y = (x^lambda - 1) / lambda (for lambda != 0)

Reciprocal Transformation: Dramatically reduces large values, suitable for␣
 ↪highly skewed data
Formula: y = 1/x

Exponential Transformation: Expands small values, useful for negative skewness
Formula: y = x^p, where p > 1


 '''
```



[6]:
```
'\nSkewness explanation:\n Positive skew: Tail is longer on the right.\n
Negative skew: Tail is longer on the left.\n Symmetrical distribution: Balanced
tails on both sides.\n\n\nMethods to reduce skewness\nLog Transformation:
Reduces positive skewness by compressing large values\nFormula: y =
log(x)\n\nSquare Root Transformation: Compresses larger values, less aggressive
than log transformation\nFormula: y = sqrt(x)\n\nCube Root Transformation:
Reduces both positive and negative skewness\nFormula: y = x^(1/3)\n\nBox-Cox
Transformation: Finds the best power transformation (lambda) for reducing
skewness\nFormula: y = (x^lambda - 1) / lambda (for lambda != 0)\n\nReciprocal
Transformation: Dramatically reduces large values, suitable for highly skewed
data\nFormula: y = 1/x\n\nExponential Transformation: Expands small values,
useful for negative skewness\nFormula: y = x^p, where p > 1\n\n \n '
```

Question 5

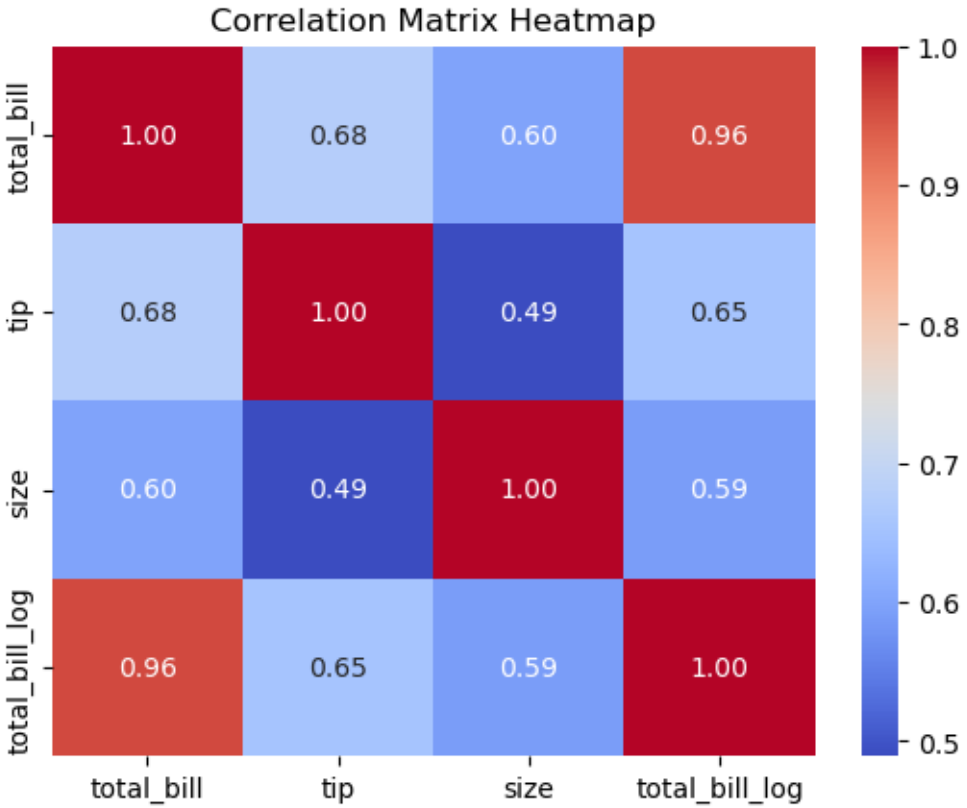
Generate covariance matrix, correlation matrix and heatmap for the dataset.

Explain all values present in matrix in jupyter notebook.

[7]:
```
# Q5
numeric_tips = tips.select_dtypes(include=['float64', 'int64'])
```

```
cov = numeric_tips.cov()
corr = numeric_tips.corr()
print(cov)
print(corr)
plt.figure()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix Heatmap')
plt.show()
```

```
                total_bill       tip      size  total_bill_log
total_bill       79.252939  8.323502  5.065983        3.738577
tip               8.323502  1.914455  0.643906        0.397362
size              5.065983  0.643906  0.904591        0.246124
total_bill_log    3.738577  0.397362  0.246124        0.192612
                total_bill       tip      size  total_bill_log
total_bill        1.000000  0.675734  0.598315        0.956879
tip               0.675734  1.000000  0.489299        0.654368
size              0.598315  0.489299  1.000000        0.589640
total_bill_log    0.956879  0.654368  0.589640        1.000000
```



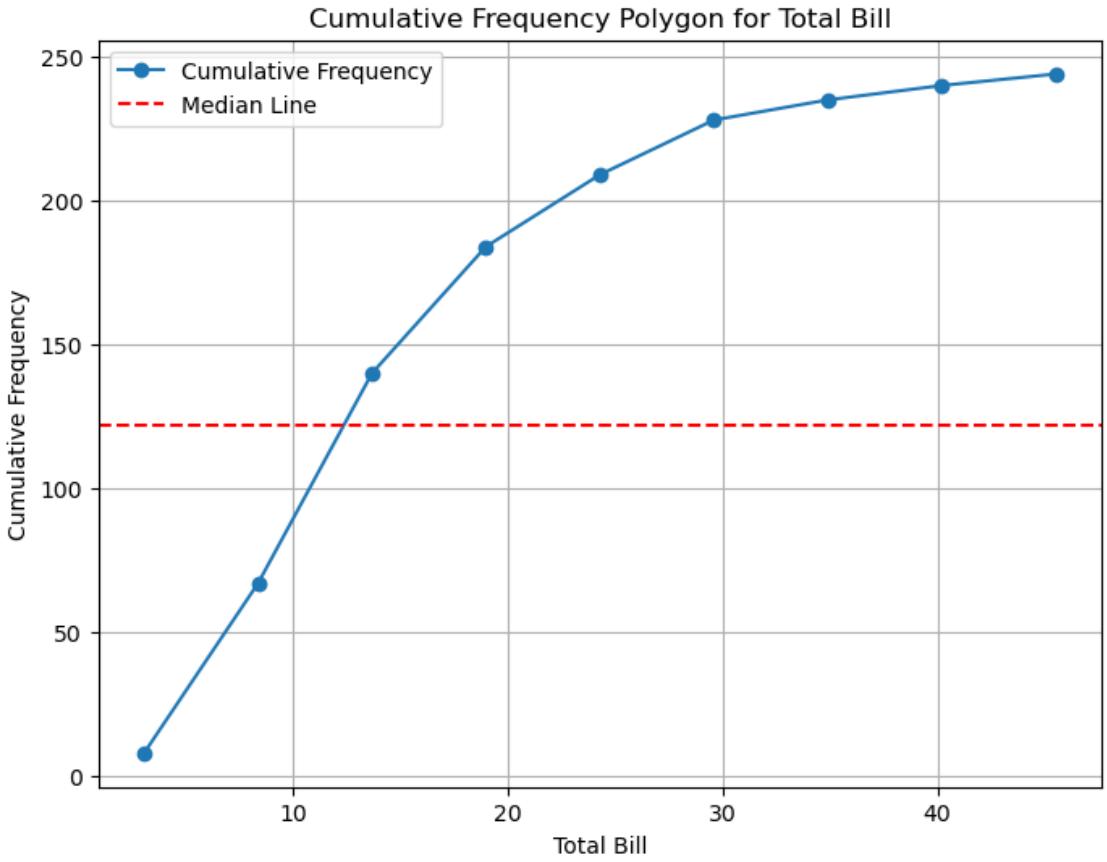
[8]:
```
'''
Explanation :-

Covariance and Correlation Matrices:
Covariance shows how two things change together (uses units).
Correlation makes those values between -1 and 1.
Close to 1/-1 = strong positive/negative link.
Close to 0 = weak or not.

'''
```

[8]:
```
'\nExplanation :-\n\nCovariance and Correlation Matrices:  \nCovariance shows
how two things change together (uses units).  \nCorrelation makes those values
between -1 and 1.  \nClose to 1/-1 = strong positive/negative link.  \nClose to
0 = weak or not.\n\n'
```

Question 6

Plot cumulative frequency polygon for 'total bill' column and find out median value from graph

[9]:
```
# Q6
total_bill = tips['total_bill'].sort_values()
bins = np.linspace(total_bill.min(), total_bill.max(), 10)
freq, bin_edges = np.histogram(total_bill, bins=bins)
cumulative_freq = np.cumsum(freq)

plt.figure(figsize=(8, 6))
plt.plot(bin_edges[:-1], cumulative_freq, marker='o', label='Cumulative␣
 ↪Frequency')
plt.axhline(y=0.5 * cumulative_freq[-1], color='r', linestyle='--',␣
 ↪label='Median Line')
plt.title('Cumulative Frequency Polygon for Total Bill')
plt.xlabel('Total Bill')
plt.ylabel('Cumulative Frequency')
plt.legend()
plt.grid()
plt.show()

median_bin_index = np.argmax(cumulative_freq >= 0.5 * cumulative_freq[-1])
median_value = bin_edges[median_bin_index]
```



Question 7

Find unique values and their value counts for each column.

[10]:
```
# Q7
for column in tips.columns:
    print(f"Unique values and their counts for {column}:")
    print(tips[column].value_counts())
    print(f"Unique values in {column}: {tips[column].unique()}")
    print("\n")
```

```
Unique values and their counts for total_bill:
total_bill
13.42    3
13.81    2
15.98    2
17.92    2
10.07    2
        ..
24.71    1
21.16    1
```

```
28.97    1
22.49    1
18.78    1
Name: count, Length: 229, dtype: int64
Unique values in total_bill: [16.99 10.34 21.01 23.68 24.59 25.29  8.77 26.88
15.04 14.78 10.27 35.26
 15.42 18.43 14.83 21.58 10.33 16.29 16.97 20.65 17.92 20.29 15.77 39.42
 19.82 17.81 13.37 12.69 21.7  19.65  9.55 18.35 15.06 20.69 17.78 24.06
 16.31 16.93 18.69 31.27 16.04 17.46 13.94  9.68 30.4  18.29 22.23 32.4
 28.55 18.04 12.54 10.29 34.81  9.94 25.56 19.49 38.01 26.41 11.24 48.27
 13.81 11.02 17.59 20.08 16.45  3.07 20.23 15.01 12.02 17.07 26.86 25.28
 14.73 10.51 27.2  22.76 17.29 19.44 16.66 10.07 32.68 15.98 34.83 13.03
 18.28 24.71 21.16 28.97 22.49  5.75 16.32 22.75 40.17 27.28 12.03 12.46
 11.35 15.38 44.3  22.42 20.92 15.36 20.49 25.21 18.24 14.31 14.    7.25
 38.07 23.95 25.71 17.31 29.93 10.65 12.43 24.08 11.69 13.42 14.26 15.95
 12.48 29.8   8.52 14.52 11.38 22.82 19.08 20.27 11.17 12.26 18.26  8.51
 14.15 16.   13.16 17.47 34.3  41.19 27.05 16.43  8.35 18.64 11.87  9.78
  7.51 14.07 13.13 17.26 24.55 19.77 29.85 48.17 25.   13.39 16.49 21.5
 12.66 16.21 17.51 24.52 20.76 31.71 10.59 10.63 50.81 15.81 31.85 16.82
 32.9  17.89 14.48  9.6  34.63 34.65 23.33 45.35 23.17 40.55 20.9  30.46
 18.15 23.1  15.69 19.81 28.44 15.48 16.58  7.56 43.11 13.   13.51 18.71
 12.74 16.4  20.53 16.47 26.59 38.73 24.27 12.76 30.06 25.89 48.33 13.27
 28.17 12.9  28.15 11.59  7.74 30.14 12.16  8.58 16.27 10.09 20.45 13.28
 22.12 24.01 11.61 10.77 15.53 12.6  32.83 35.83 29.03 27.18 22.67 17.82
 18.78]


Unique values and their counts for tip:
tip
2.00    33
3.00    23
4.00    12
5.00    10
2.50    10
        ..
4.34     1
1.56     1
5.20     1
2.60     1
1.75     1
Name: count, Length: 123, dtype: int64
Unique values in tip: [ 1.01  1.66  3.5   3.31  3.61  4.71  2.    3.12  1.96
3.23  1.71  5.
  1.57  3.    3.02  3.92  1.67  3.71  3.35  4.08  2.75  2.23  7.58  3.18
  2.34  4.3   1.45  2.5   2.45  3.27  3.6   3.07  2.31  2.24  2.54  3.06
  1.32  5.6   6.    2.05  2.6   5.2   1.56  4.34  3.51  1.5   1.76  6.73
  3.21  1.98  3.76  2.64  3.15  2.47  1.    2.01  2.09  1.97  3.14  2.2
  1.25  3.08  4.    2.71  3.4   1.83  2.03  5.17  5.85  3.25  4.73  3.48
```

```
  1.64  4.06  4.29  2.55  5.07  1.8   2.92  1.68  2.52  4.2   1.48  2.18
  2.83  6.7   2.3   1.36  1.63  1.73  2.74  5.14  3.75  2.61  4.5   1.61
 10.    3.16  5.15  3.11  3.55  3.68  5.65  6.5   4.19  2.56  2.02  1.44
  3.41  5.16  9.    1.1   3.09  1.92  1.58  2.72  2.88  3.39  1.47  1.17
  4.67  5.92  1.75]


Unique values and their counts for sex:
sex
Male      157
Female     87
Name: count, dtype: int64
Unique values in sex: ['Female', 'Male']
Categories (2, object): ['Male', 'Female']


Unique values and their counts for smoker:
smoker
No     151
Yes     93
Name: count, dtype: int64
Unique values in smoker: ['No', 'Yes']
Categories (2, object): ['Yes', 'No']


Unique values and their counts for day:
day
Sat     87
Sun     76
Thur    62
Fri     19
Name: count, dtype: int64
Unique values in day: ['Sun', 'Sat', 'Thur', 'Fri']
Categories (4, object): ['Thur', 'Fri', 'Sat', 'Sun']


Unique values and their counts for time:
time
Dinner    176
Lunch      68
Name: count, dtype: int64
Unique values in time: ['Dinner', 'Lunch']
Categories (2, object): ['Lunch', 'Dinner']


Unique values and their counts for size:
size
2    156
```

```
3     38
4     37
5      5
1      4
6      4
Name: count, dtype: int64
Unique values in size: [2 3 4 1 6 5]


Unique values and their counts for total_bill_log:
total_bill_log
2.596746    3
2.625393    2
2.771338    2
2.885917    2
2.309561    2
           ..
3.207208    1
3.052113    1
3.366261    1
3.113071    1
2.932792    1
Name: count, Length: 229, dtype: int64
Unique values in total_bill_log: [2.83262494 2.33601987 3.04499851 3.16463081
3.20233986 3.23040906
 2.17133681 3.29138252 2.71071332 2.69327492 2.32922702 3.56274918
 2.73566537 2.91397977 2.69665216 3.07176696 2.33505228 2.79055142
 2.83144708 3.02771532 2.88591741 3.01012815 2.7581094  3.6742733
 2.98669153 2.8797601  2.59301339 2.54081428 3.07731226 2.97807734
 2.25654115 2.90962957 2.71204222 3.02965049 2.87807423 3.18055071
 2.79177842 2.8290872  2.92798862 3.44265917 2.7750856  2.85991255
 2.63476241 2.2700619  3.41444261 2.90635446 3.10144273 3.47815842
 3.35165694 2.89259151 2.52892354 2.33117255 3.5499047  2.29656702
 3.24102863 2.96990151 3.63784928 3.27374273 2.41947884 3.87681025
 2.62539297 2.3997118  2.86733056 2.99972429 2.80032548 1.12167756
 3.00716665 2.70871665 2.48657193 2.83732254 3.29063819 3.23001357
 2.68988623 2.35232718 3.30321697 3.12500461 2.8501283  2.9673328
 2.81301064 2.30956071 3.48676327 2.77133794 3.55047908 2.56725439
 2.90580757 3.20720802 3.05211261 3.36626081 3.11307077 1.74919985
 2.79239135 3.12456515 3.69312045 3.30615383 2.48740353 2.52252351
 2.42921774 2.73306796 3.79098468 3.10995342 3.04070564 2.73176673
 3.01993696 3.22724074 2.90361698 2.66095859 2.63905733 1.98100147
 3.63942657 3.17596832 3.24688002 2.85128437 3.39886132 2.36555989
 2.52011291 3.18138162 2.45873378 2.59674613 2.65745841 2.76945883
 2.52412736 3.39450839 2.14241634 2.67552701 2.43185743 3.12763734
 2.94864067 3.00914196 2.41323161 2.50634193 2.90471288 2.14124194
 2.64971462 2.77258872 2.57718193 2.86048512 3.53514535 3.71819551
 3.29768701 2.79910893 2.12226154 2.92530981 2.47401421 2.28033948
```

```
2.01623547 2.64404487 2.57489969 2.84839169 3.20071185 2.98416564
3.39618484 3.87473642 3.21887582 2.59450816 2.80275414 3.06805294
2.53844742 2.78562834 2.86277215 3.19948911 3.03302806 3.45663209
2.35991016 2.36368019 3.92809319 2.76064265 3.46103738 2.82256865
3.49347266 2.8842419  2.67276839 2.2617631  3.54472036 3.54529773
3.14974009 3.81441018 3.14285834 3.70253578 3.03974916 3.41641435
2.89867056 3.13983262 2.75302357 2.98618686 3.3477966  2.73954887
2.80819715 2.02287119 3.76375499 2.56494936 2.60343015 2.92905814
2.54474665 2.79728133 3.02188723 2.80154054 3.28053521 3.65661449
3.18924102 2.54631528 3.40319538 3.25385679 3.87805249 2.58550585
3.33825758 2.55722731 3.33754735 2.45014266 2.04640169 3.40585319
2.49815188 2.14943391 2.78932292 2.31154483 3.01798288 2.58625914
3.09648218 3.17847041 2.4518668  2.37676449 2.74277364 2.53369681
3.49134273 3.57878553 3.36832978 3.30248141 3.12104246 2.88032142
2.93279247]
```

Question 8

Find out if there are any null records in data.

[11]:
```
# Q8
null_records = tips.isnull().sum()
print("Null records in each column:")
print(null_records)
```

```
Null records in each column:
total_bill        0
tip               0
sex               0
smoker            0
day               0
time              0
size              0
total_bill_log    0
dtype: int64
```

Question 8

How to replace null records?

[12]:
```
'''
 Q9. How to replace null records?
Ans: Replace null values in numerical columns with the median

'''
```

[12]:
```
'\n Q9. How to replace null records?\nAns: Replace null values in numerical
columns with the median\n \n'
```

Question 10

Drop unnecessary columns from the data.

[13]:
```
'''
Q10
 Drop unnecessary columns from the dataset (e.g., 'day' and 'time')

'''
tips.drop(columns=['day', 'time'], inplace=True)
tips
```

[13]:

	total_bill	tip	sex	smoker	size	total_bill_log
0	16.99	1.01	Female	No	2	2.832625
1	10.34	1.66	Male	No	3	2.336020
2	21.01	3.50	Male	No	3	3.044999
3	23.68	3.31	Male	No	2	3.164631
4	24.59	3.61	Female	No	4	3.202340
..	…	…	…	…	…	…
239	29.03	5.92	Male	No	3	3.368330
240	27.18	2.00	Female	Yes	2	3.302481
241	22.67	2.00	Male	Yes	2	3.121042
242	17.82	1.75	Male	No	2	2.880321
243	18.78	3.00	Female	No	2	2.932792

[244 rows x 6 columns]

CLASSROOM IS COMPLETED SUCCESSFULLY….

ASSIGNMENT 3 WORK

1) Find out count of unique records in each column.
2) Find if any outliers in data.
3) Plot heatmap of correlation matrix and covariance matrix for the given dataset.
4) Remove unnecessary or empty columns as well as any rows if required from the dataset.
5) Plot histograms for each column and remove any skewness using transformations.
6) Plot Yearly records for numerical columns (e.g. runs, trophies)

QUESTION 1

1) Find out count of unique records in each column.

[18]:
```
df = pd.read_csv("matches.csv")    #we load a dataset
df.head()
unique_counts = df.nunique()
print("Unique counts in each column:\n", unique_counts)
```

```
Unique counts in each column:
 id                 1095
season               17
```

```
city                36
date               823
match_type           8
player_of_match    291
venue               58
team1               19
team2               19
toss_winner         19
toss_decision        2
winner              19
result               4
result_margin       98
target_runs        170
target_overs        15
super_over           2
method               1
umpire1             62
umpire2             62
dtype: int64
```

QUESTION 2

2) Find if any outliers in data.

[20]:
```
numerical_cols = df.select_dtypes(include=[np.number]).columns
print("Summary statistics for numerical columns:\n", df[numerical_cols].
  ↪describe())
```

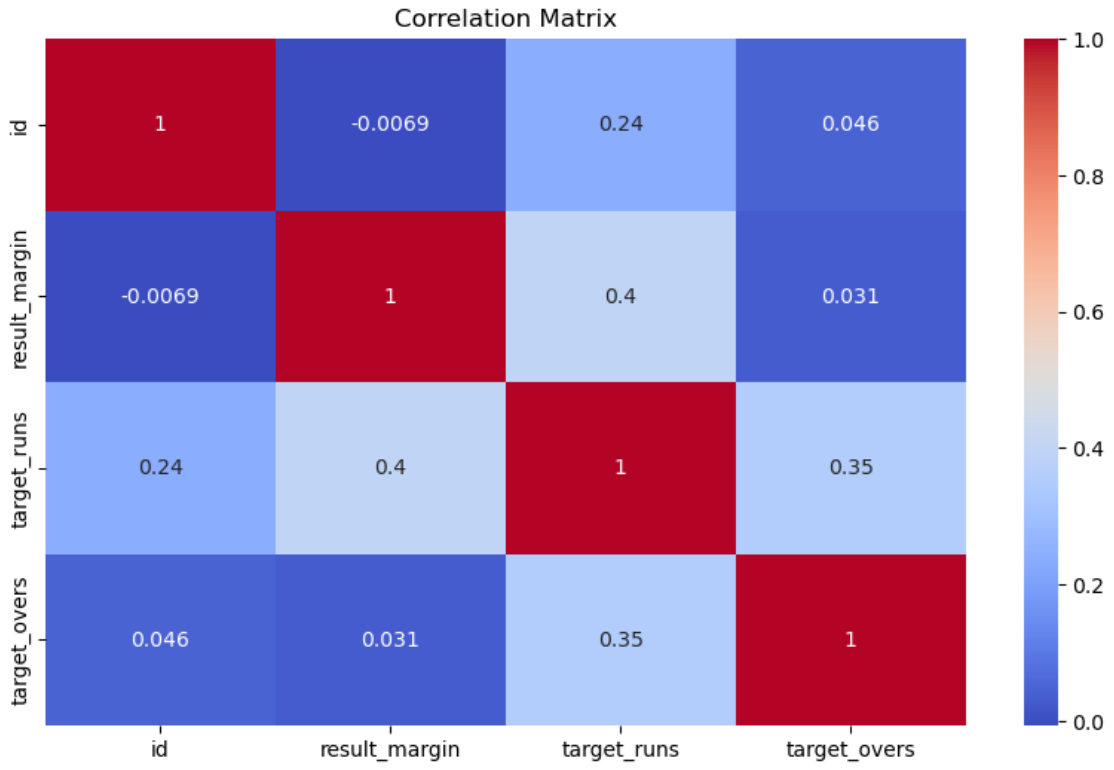
```
Summary statistics for numerical columns:
                  id  result_margin  target_runs  target_overs
count  1.095000e+03    1076.000000  1092.000000   1092.000000
mean   9.048283e+05      17.259294   165.684066     19.759341
std    3.677402e+05      21.787444    33.427048      1.581108
min    3.359820e+05       1.000000    43.000000      5.000000
25%    5.483315e+05       6.000000   146.000000     20.000000
50%    9.809610e+05       8.000000   166.000000     20.000000
75%    1.254062e+06      20.000000   187.000000     20.000000
max    1.426312e+06     146.000000   288.000000     20.000000
```

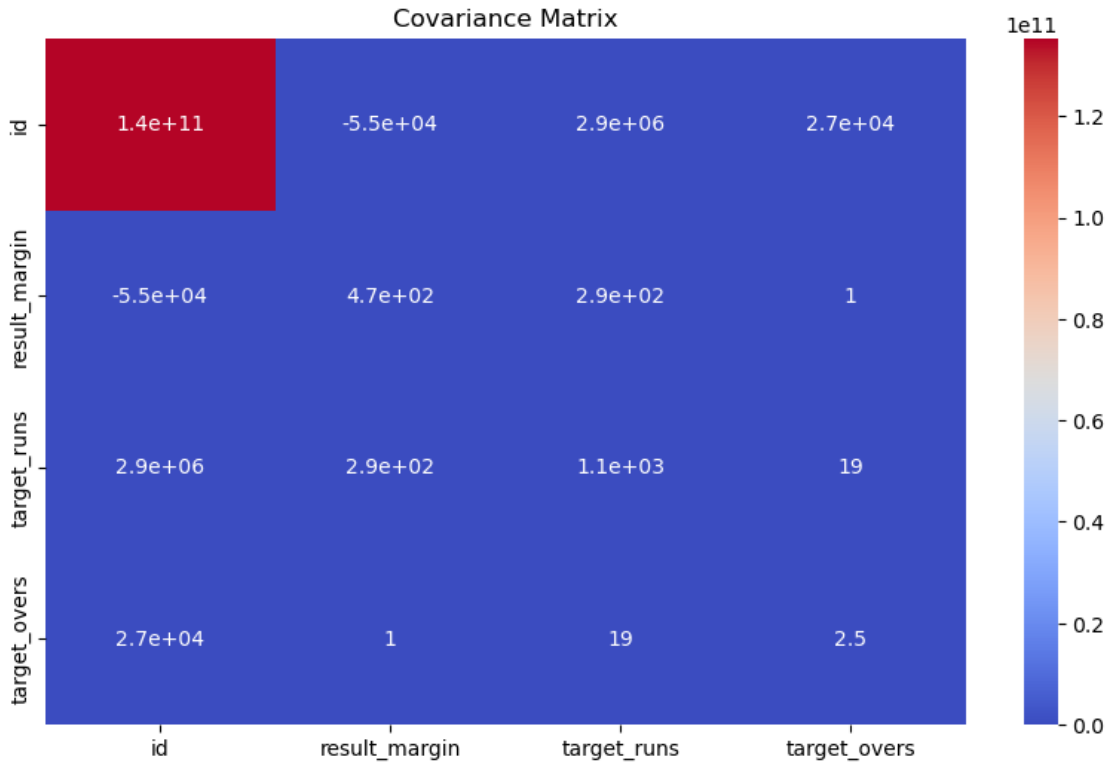
QUESTION 3

3) Plot heatmap of correlation matrix and covariance matrix for the given dataset.

[26]:
```
numerical_cols = df.select_dtypes(include=['float64', 'int64']).columns

correlation_matrix = df[numerical_cols].corr()
covariance_matrix = df[numerical_cols].cov()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm")
```

```
plt.title("Correlation Matrix")
plt.show()

plt.figure(figsize=(10, 6))
sns.heatmap(covariance_matrix, annot=True, cmap="coolwarm")
plt.title("Covariance Matrix")
plt.show()
```





QUESTION 4

4) Remove unnecessary or empty columns as well as any rows if required from the dataset.

[28]:

```
df_cleaned = df.dropna(axis=1, how='all')

df_cleaned = df_cleaned.dropna()

print("Cleaned Dataset Info:")
print(df_cleaned.info())
df
```

```
Cleaned Dataset Info:
<class 'pandas.core.frame.DataFrame'>
Index: 21 entries, 38 to 1023
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id               21 non-null     int64
 1   season           21 non-null     object
 2   city             21 non-null     object
 3   date             21 non-null     object
 4   match_type       21 non-null     object
```

```
 5   player_of_match  21 non-null     object
 6   venue            21 non-null     object
 7   team1            21 non-null     object
 8   team2            21 non-null     object
 9   toss_winner      21 non-null     object
 10  toss_decision    21 non-null     object
 11  winner           21 non-null     object
 12  result           21 non-null     object
 13  result_margin    21 non-null     float64
 14  target_runs      21 non-null     float64
 15  target_overs     21 non-null     float64
 16  super_over       21 non-null     object
 17  method           21 non-null     object
 18  umpire1          21 non-null     object
 19  umpire2          21 non-null     object
dtypes: float64(3), int64(1), object(16)
memory usage: 3.4+ KB
None
```

[28]:

	id	season	city	date	match_type	player_of_match	venue	team1	team2
0	335982	2007/08	Bangalore	2008-04-18	League	BB McCullum	M Chinnaswamy Stadium	Royal Challengers Bangalore	Kolkata Knight Riders
1	335983	2007/08	Chandigarh	2008-04-19	League	MEK Hussey	Punjab Cricket Association Stadium, Mohali	…	…
2	335984	2007/08	Delhi	2008-04-19	League	MF Maharoof	Feroz Shah Kotla	…	…
3	335985	2007/08	Mumbai	2008-04-20	League	MV Boucher	Wankhede Stadium	…	…
4	335986	2007/08	Kolkata	2008-04-20	League	DJ Hussey	Eden Gardens	…	…
…	…	…	…	…	…	…	…	…	…
1090	1426307	2024	Hyderabad	2024-05-19	League	Abhishek Sharma	Rajiv Gandhi International Stadium, Uppal, Hyd…	…	…
1091	1426309	2024	Ahmedabad	2024-05-21	Qualifier 1	MA Starc	Narendra Modi Stadium, Ahmedabad	…	…
1092	1426310	2024	Ahmedabad	2024-05-22	Eliminator	R Ashwin	Narendra Modi Stadium, Ahmedabad	…	…
1093	1426311	2024	Chennai	2024-05-24	Qualifier 2	Shahbaz Ahmed	MA Chidambaram Stadium, Chepauk, Chennai	…	…
1094	1426312	2024	Chennai	2024-05-26	Final	MA Starc	MA Chidambaram Stadium, Chepauk, Chennai	…	…

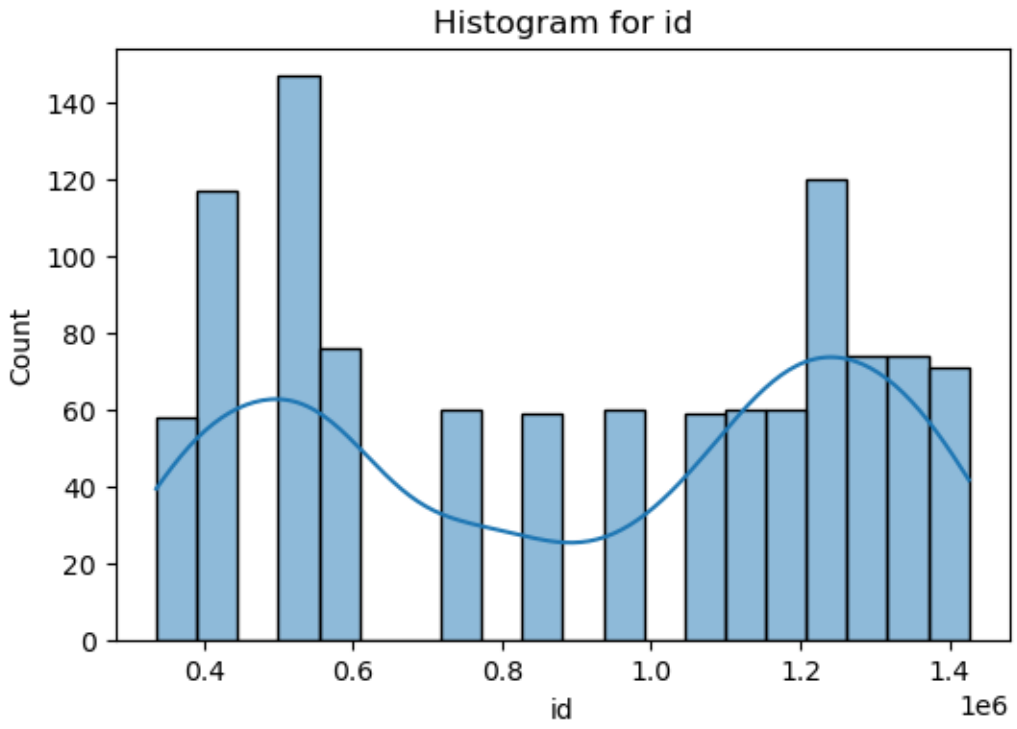
```
1                 Kings XI Punjab          Chennai Super Kings
2               Delhi Daredevils             Rajasthan Royals
3                 Mumbai Indians  Royal Challengers Bangalore
4           Kolkata Knight Riders              Deccan Chargers
…                             …                           …
1090                Punjab Kings          Sunrisers Hyderabad
1091          Sunrisers Hyderabad        Kolkata Knight Riders
1092  Royal Challengers Bengaluru             Rajasthan Royals
1093          Sunrisers Hyderabad             Rajasthan Royals
1094          Sunrisers Hyderabad        Kolkata Knight Riders

                     toss_winner toss_decision                        winner  \
0     Royal Challengers Bangalore         field        Kolkata Knight Riders
1             Chennai Super Kings           bat          Chennai Super Kings
2                Rajasthan Royals           bat             Delhi Daredevils
3                  Mumbai Indians           bat  Royal Challengers Bangalore
4                 Deccan Chargers           bat        Kolkata Knight Riders
…                             …           …                           …
1090                Punjab Kings           bat          Sunrisers Hyderabad
1091          Sunrisers Hyderabad           bat        Kolkata Knight Riders
1092             Rajasthan Royals         field             Rajasthan Royals
1093             Rajasthan Royals         field          Sunrisers Hyderabad
1094          Sunrisers Hyderabad           bat        Kolkata Knight Riders

       result  result_margin  target_runs  target_overs super_over method  \
0        runs          140.0        223.0          20.0          N    NaN
1        runs           33.0        241.0          20.0          N    NaN
2     wickets            9.0        130.0          20.0          N    NaN
3     wickets            5.0        166.0          20.0          N    NaN
4     wickets            5.0        111.0          20.0          N    NaN
…       …             …          …            …         …    …
1090  wickets            4.0        215.0          20.0          N    NaN
1091  wickets            8.0        160.0          20.0          N    NaN
1092  wickets            4.0        173.0          20.0          N    NaN
1093     runs           36.0        176.0          20.0          N    NaN
1094  wickets            8.0        114.0          20.0          N    NaN

                    umpire1               umpire2
0                 Asad Rauf            RE Koertzen
1                 MR Benson             SL Shastri
2                 Aleem Dar         GA Pratapkumar
3                  SJ Davis              DJ Harper
4                 BF Bowden            K Hariharan
…                       …                     …
1090            Nitin Menon             VK Sharma
1091          AK Chaudhary              R Pandit
1092  KN Ananthapadmanabhan  MV Saidharshan Kumar
```

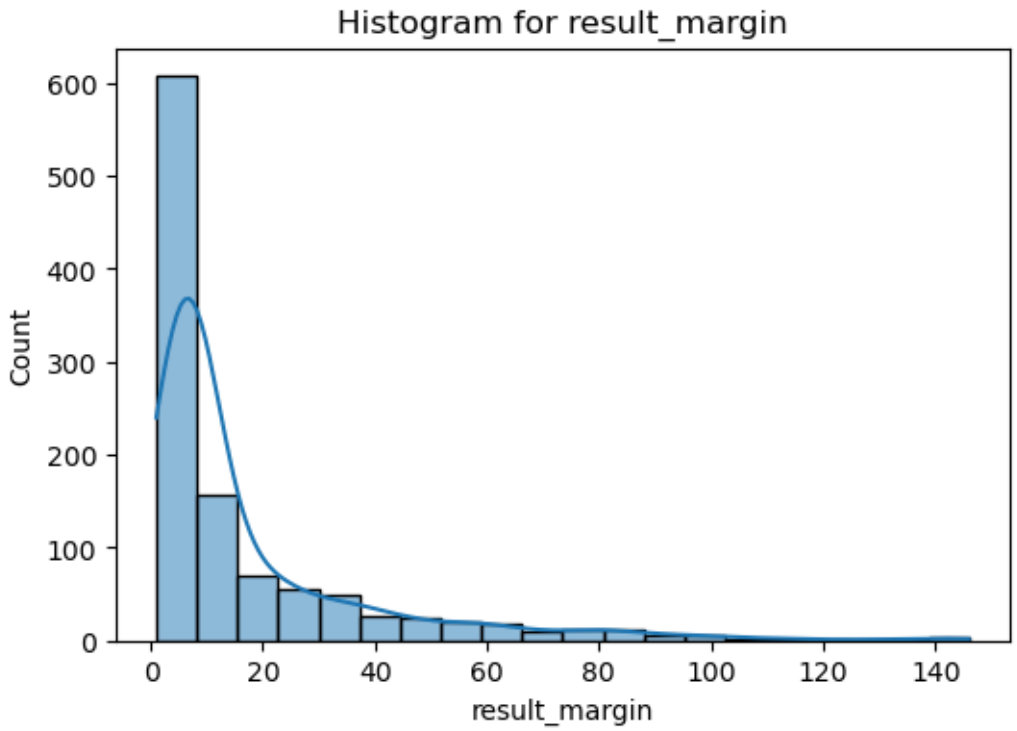
```
1093            Nitin Menon             VK Sharma
1094          J Madanagopal           Nitin Menon

[1095 rows x 20 columns]
```

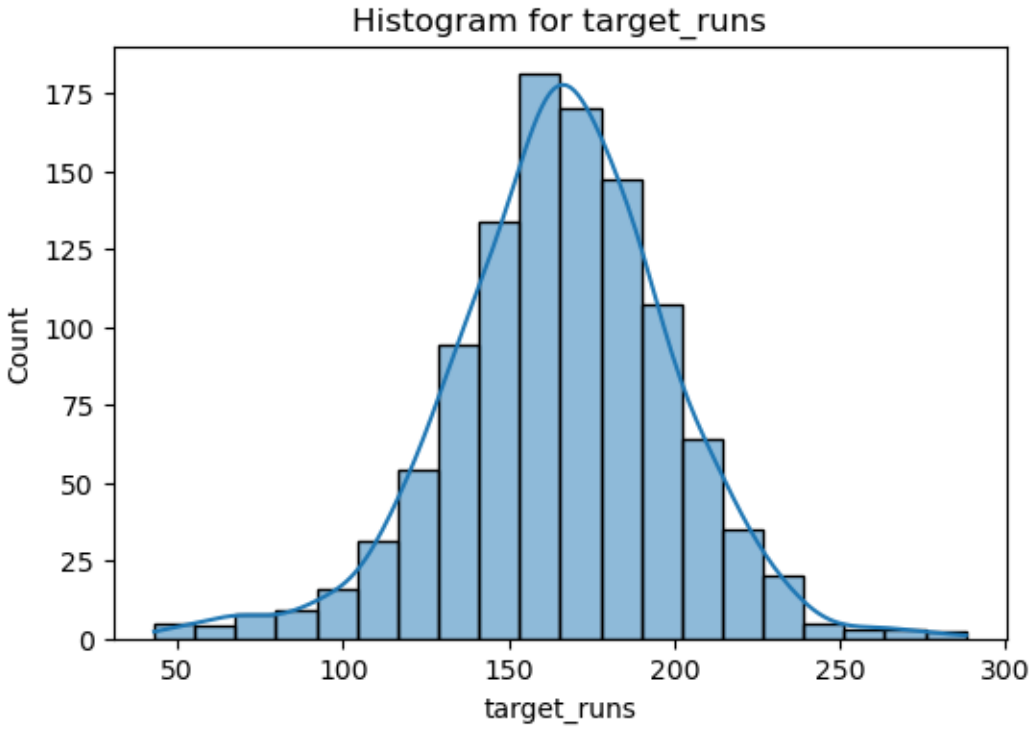
QUESTION 5

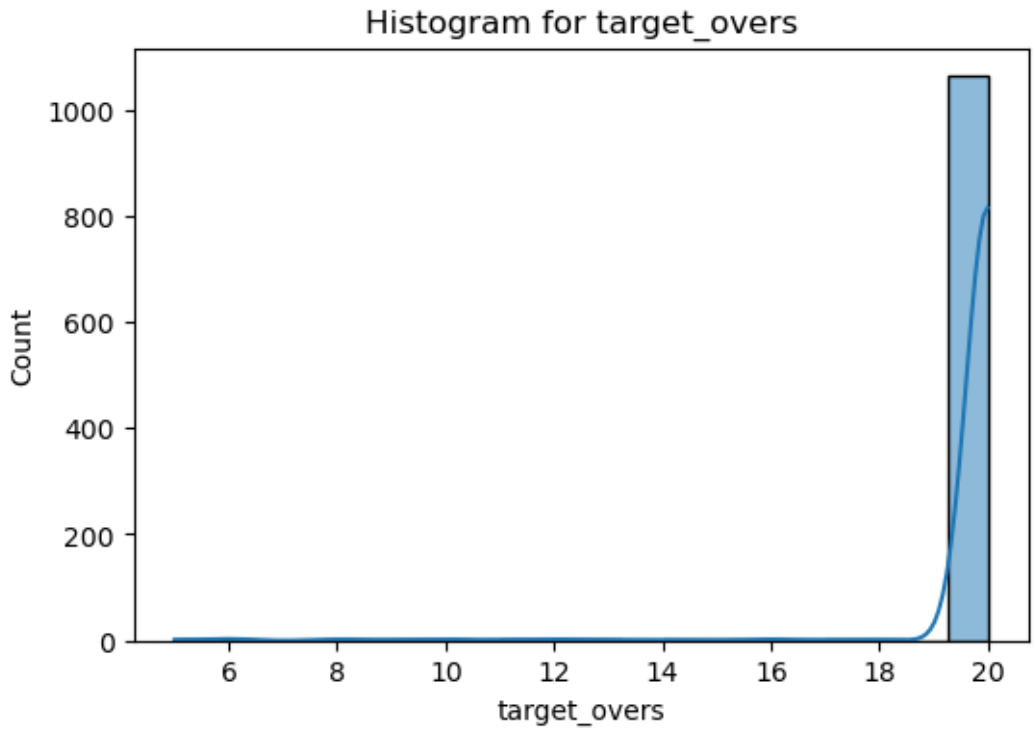
5) Plot histograms for each column and remove any skewness using transformations.

[29]:
```
for col in numerical_cols:
    plt.figure(figsize=(6, 4))
    sns.histplot(df[col], kde=True, bins=20)
    plt.title(f"Histogram for {col}")
    plt.show()


# Example: Log Transformation
df['log_result_margin'] = np.log1p(df['result_margin'])  # Log transformation␣
 ↪for skewed column
```




Histogram for result_margin
Count
result_margin
0
100
200
300
400
500
600
0
20
40
60
80
100
120
140


Histogram for target_runs
Count
target_runs
0
25
50
75
100
125
150
175
50
100
150
200
250
300

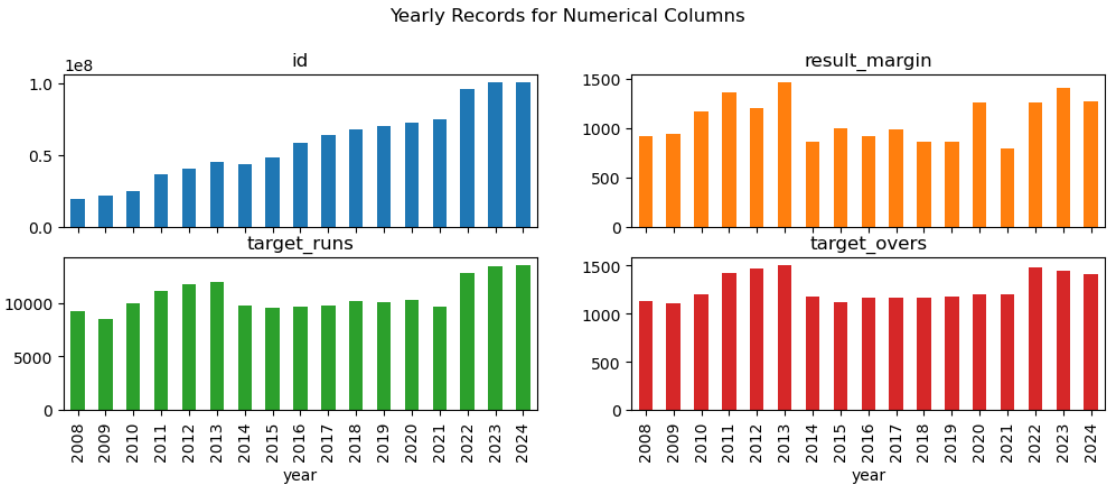


QUESTION 6

6) Plot Yearly records for numerical columns (e.g. runs, trophies)

[30]:

```
df['year'] = pd.to_datetime(df['date']).dt.year


yearly_data = df.groupby('year')[numerical_cols].sum()


yearly_data.plot(kind='bar', figsize=(12, 6), subplots=True, layout=(3, 2),␣
↪legend=False)
plt.suptitle("Yearly Records for Numerical Columns")
plt.show()
```



ASSIGNMENT 3 WORK COMPELTED

---

from file: PMRP_4

OM CHOKSI 23AIML010 CLASSROOM ASSIGNMENT 4 TASK

CLASS TASK TWO EXAMPLES

1) A factory produces light bulbs, and 5% of the bulbs produced are defective. There is a test that is used to check the bulbs:

- It correctly identifies defective bulbs 98% of the time (True Positive Rate).
- It incorrectly identifies non-defective bulbs as defective 3% of the time (False Positive Rate).

If a bulb tests positive for being defective, what is the probability that it is actually defective?

2) Consider an email spam filter that classifies emails as either “spam” or “not spam.” The filter is trained on a dataset where:

- 10% of emails are spam.
- The filter correctly identifies spam emails 90% of the time (True Positive Rate).
- The filter incorrectly identifies non-spam emails as spam 5% of the time (False Positive Rate).

If an email is flagged as spam, what is the probability that it is actually spam?

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Question 1

[2]:
```
'''
```

```
1) A factory produces light bulbs, and 5% of the bulbs produced are defective.␣
 ↪There is a test that is used to check the bulbs:
- It correctly identifies defective bulbs 98% of the time (True Positive Rate).
- It incorrectly identifies non-defective bulbs as defective 3% of the time␣
 ↪(False Positive Rate).

'''



P_D = 0.05
P_not_D = 0.95
P_T_D = 0.98
P_T_not_D = 0.03


P_T = (P_T_D * P_D) + (P_T_not_D * P_not_D)


P_D_T = (P_T_D * P_D) / P_T

print(f"The probability that the bulb is actually defective given it tests␣
 ↪positive is: {P_D_T:.4f}")
```

```
The probability that the bulb is actually defective given it tests positive is:
0.6323
```

Question 2

[ ]:
```
'''
2) Consider an email spam filter that classifies emails as either "spam" or␣
 ↪"not spam." The filter is trained on a dataset where:
- 10% of emails are spam.
- The filter correctly identifies spam emails 90% of the time (True Positive␣
 ↪Rate).
- The filter incorrectly identifies non-spam emails as spam 5% of the time␣
 ↪(False Positive Rate).

If an email is flagged as spam, what is the probability that it is actually␣
 ↪spam?

'''



P_S = 0.1
P_not_S = 0.9
P_F_given_S = 0.9
P_F_given_not_S = 0.05
```

```
P_F = (P_F_given_S * P_S) + (P_F_given_not_S * P_not_S)


P_S_given_F = (P_F_given_S * P_S) / P_F


print(f"The probability that the email is actually spam given it is flagged as␣
 ↪spam is: {P_S_given_F:.4f}")
```

```
The probability that the email is actually spam given it is flagged as spam is:
0.6667
```

CLASSWORK ASSIGNMENT 4

Use this dataset: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Apply Bayes theorem on the dataset to calculate probability of fraudulent transaction if there is high amount transaction. Threshold for higher amount is 100. Calculate all the required probabilities from dataset. Perform necessary data cleaning part (V1 to V8 columns are not required) and perform all calculations in .py or .ipynb file.

[ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
file_path = '/content/my-drive/creditcard.csv'
df = pd.read_csv(file_path)
df
```

[ ]:
```
      Time        V1        V2        V3        V4        V5        V6  \
0        0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1        0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2        1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3        1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4        2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
…     …       …       …       …       …       …       …
5969  6634 -1.611463  0.190648  0.901715  1.531254 -1.535865  0.799245
5970  6635 -1.420272  1.449354  1.320110 -1.894320  0.913695  0.454601
5971  6637 -1.206696  0.284728  2.152053 -2.850437 -0.437285 -0.238376
5972  6644  1.067611  0.091006 -0.153917  0.704233  0.113894 -0.826866
5973  6645 -0.535272 -0.132299  2.180041  1.018303 -1.498819  0.529570

            V7        V8        V9  …       V21       V22       V23  \
0     0.239599  0.098698  0.363787  … -0.018307  0.277838 -0.110474
1    -0.078803  0.085102 -0.255425  … -0.225775 -0.638672  0.101288
```

```
2     0.791461  0.247676 -1.514654  …  0.247998  0.771679  0.909412
3     0.237609  0.377436 -1.387024  … -0.108300  0.005274 -0.190321
4     0.592941 -0.270533  0.817739  … -0.009431  0.798278 -0.137458
…          …        …        …  …        …         …         …
5969  1.513786  0.495829  0.200390  …  0.211223  0.007477  1.026272
5970  0.894179 -0.385450  2.433841  … -0.529027 -0.368394 -0.247773
5971 -0.333341  0.334679  2.870542  …  0.039460  0.464476 -0.457193
5972  0.567690 -0.464181  0.957295  … -0.476723 -1.410090 -0.037550
5973  0.420147  0.045445  1.543919  …       NaN       NaN       NaN

            V24       V25       V26       V27       V28  Amount  Class
0     0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62    0.0
1    -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69    0.0
2    -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66    0.0
3    -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50    0.0
4     0.141267 -0.206010  0.502292  0.219422  0.215153   69.99    0.0
…          …        …        …        …         …     …    …
5969  0.057628 -0.024955 -0.368263  0.081684  0.140669  458.92    0.0
5970 -1.189156 -0.126040  0.701487  0.277333 -0.222694    0.77    0.0
5971 -0.556105  0.517579  0.008006  0.366054  0.185008   14.00    0.0
5972 -0.177773  0.321810  0.114930 -0.109640  0.023205  139.90    0.0
5973       NaN       NaN       NaN       NaN       NaN     NaN    NaN

[5974 rows x 31 columns]
```

[ ]:
```
df.head(),df.tail(),df.describe()
```

[ ]:
```
(   Time        V1        V2        V3        V4        V5        V6        V7
\
 0     0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
 1     0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
 2     1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
 3     1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
 4     2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

          V8        V9  …       V21       V22       V23       V24       V25  \
 0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
 1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
 2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
 3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
 4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

         V26       V27       V28  Amount  Class
 0 -0.189115  0.133558 -0.021053  149.62    0.0
 1  0.125895 -0.008983  0.014724    2.69    0.0
 2 -0.139097 -0.055353 -0.059752  378.66    0.0
 3 -0.221929  0.062723  0.061458  123.50    0.0
```

```
4  0.502292  0.219422  0.215153   69.99    0.0

[5 rows x 31 columns],
      Time        V1        V2        V3        V4        V5        V6  \
5969  6634 -1.611463  0.190648  0.901715  1.531254 -1.535865  0.799245
5970  6635 -1.420272  1.449354  1.320110 -1.894320  0.913695  0.454601
5971  6637 -1.206696  0.284728  2.152053 -2.850437 -0.437285 -0.238376
5972  6644  1.067611  0.091006 -0.153917  0.704233  0.113894 -0.826866
5973  6645 -0.535272 -0.132299  2.180041  1.018303 -1.498819  0.529570

           V7        V8        V9  …       V21       V22       V23  \
5969  1.513786  0.495829  0.200390  …  0.211223  0.007477  1.026272
5970  0.894179 -0.385450  2.433841  … -0.529027 -0.368394 -0.247773
5971 -0.333341  0.334679  2.870542  …  0.039460  0.464476 -0.457193
5972  0.567690 -0.464181  0.957295  … -0.476723 -1.410090 -0.037550
5973  0.420147  0.045445  1.543919  …       NaN       NaN       NaN

          V24       V25       V26       V27       V28  Amount  Class
5969  0.057628 -0.024955 -0.368263  0.081684  0.140669  458.92    0.0
5970 -1.189156 -0.126040  0.701487  0.277333 -0.222694    0.77    0.0
5971 -0.556105  0.517579  0.008006  0.366054  0.185008   14.00    0.0
5972 -0.177773  0.321810  0.114930 -0.109640  0.023205  139.90    0.0
5973       NaN       NaN       NaN       NaN       NaN     NaN    NaN

[5 rows x 31 columns],
              Time            V1            V2            V3            V4  \
count  5974.000000  5974.000000  5974.000000  5974.000000  5974.000000
mean   2677.615501    -0.266159     0.285505     0.844231     0.104200
std    1765.025532     1.395405     1.208867     1.031448     1.442339
min       0.000000   -12.168192   -15.732974   -12.389545    -4.657545
25%    1162.250000    -1.015749    -0.280054     0.295701    -0.839417
50%    2537.000000    -0.420703     0.346083     0.882882     0.161767
75%    3781.750000     1.115402     0.941548     1.504158     1.071412
max    6645.000000     1.685314     7.467017     4.101716     6.013346

                V5            V6            V7            V8            V9  …  \
count  5974.000000  5974.000000  5974.000000  5974.000000  5974.000000  …
mean      0.000709     0.194948     0.018324    -0.039006     0.396916  …
std       1.185900     1.365525     1.059870     1.304005     1.047749  …
min     -32.092129    -7.465603   -12.968670   -23.632502    -3.336805  …
25%      -0.609206    -0.677720    -0.492968    -0.189736    -0.264280  …
50%      -0.083983    -0.142606     0.041761     0.037831     0.360826  …
75%       0.441406     0.605784     0.566306     0.343067     0.961662  …
max      10.658654    21.393069    34.303177     3.877662     9.272376  …

               V21           V22           V23           V24           V25  \
count  5973.000000  5973.000000  5973.000000  5973.000000  5973.000000
```

```
mean     -0.043098    -0.161548    -0.036483     0.028960     0.089873
std       0.883330     0.646380     0.373210     0.619810     0.407680
min     -11.468435    -8.454599    -7.996811    -2.512377    -2.322906
25%      -0.260507    -0.594625    -0.187108    -0.350226    -0.152744
50%      -0.111701    -0.177197    -0.046772     0.094946     0.106290
75%       0.059809     0.273148     0.088154     0.435670     0.355157
max      22.580675     4.393846     4.095021     3.200201     1.972515

                V26          V27          V28       Amount        Class
count  5973.000000  5973.000000  5973.000000  5973.000000  5973.000000
mean     -0.040197     0.025234     0.006116    65.061811     0.000502
std       0.488284     0.364482     0.265131   192.490314     0.022407
min      -1.338556    -7.976100    -2.909294     0.000000     0.000000
25%      -0.399334    -0.049681    -0.017776     4.450000     0.000000
50%      -0.079583     0.015976     0.019417    15.620000     0.000000
75%       0.245560     0.155281     0.082701    56.660000     0.000000
max       3.463246     3.852046     4.860769  7712.430000     1.000000

[8 rows x 31 columns])
```

[ ]:
```python
df = df.drop(columns=[f"V{i}" for i in range(1, 9)])
```

[ ]:
```python
df = df.dropna()
```

[ ]:
```python
threshold = 100


total_transactions = len(df)
fraud_transactions = df[df['Class'] == 1]
high_amount_transactions = df[df['Amount'] > threshold]
```

[ ]:
```python
p_fraud = len(fraud_transactions) / total_transactions

p_high_amount = len(high_amount_transactions) / total_transactions

print("Probability Of Fraud : ",p_fraud)
print("Probability Of high Amount  : ",p_high_amount)
```

```
Probability Of Fraud :  0.0005022601707684581
Probability Of high Amount  :  0.1523522517997656
```

[ ]:
```python
# P(High Amount | Fraud)
high_amount_fraud = fraud_transactions[fraud_transactions['Amount'] > threshold]
p_high_amount_given_fraud = len(high_amount_fraud) / len(fraud_transactions)
print(f"P(High Amount | Fraud): {p_high_amount_given_fraud:.4f}")
```

```
P(High Amount | Fraud): 0.6667
```

[ ]:
```
# P(Fraud | High Amount)
if p_high_amount > 0:
    p_fraud_given_high_amount = (p_high_amount_given_fraud * p_fraud) /␣
 ↪p_high_amount
else:
    p_fraud_given_high_amount = 0

print(f"P(Fraud | High Amount): {p_fraud_given_high_amount:.4f}")
```

```
P(Fraud | High Amount): 0.0022
```

ANOTHER LAB TASK CODE

Generate 15 random numbers from 1 to 50 and plot Q-Q plot for the points.

Plot Q-Q plot for earning column of this dataset and notedown your inference

[4]:
```
data2=pd.read_csv("G:\SEM 4\PMRP\RAW_CODE\PMRP_DAY_10\Forbes Richest Atheletes␣
 ↪(Forbes Richest Athletes 1990-2020).csv")
data2
```

```
<>:1: SyntaxWarning: invalid escape sequence '\S'
<>:1: SyntaxWarning: invalid escape sequence '\S'
C:\Users\omcho\AppData\Local\Temp\ipykernel_14844\3117694017.py:1:
SyntaxWarning: invalid escape sequence '\S'
  data2=pd.read_csv("G:\SEM 4\PMRP\RAW_CODE\PMRP_DAY_10\Forbes Richest Atheletes
(Forbes Richest Athletes 1990-2020).csv")
```

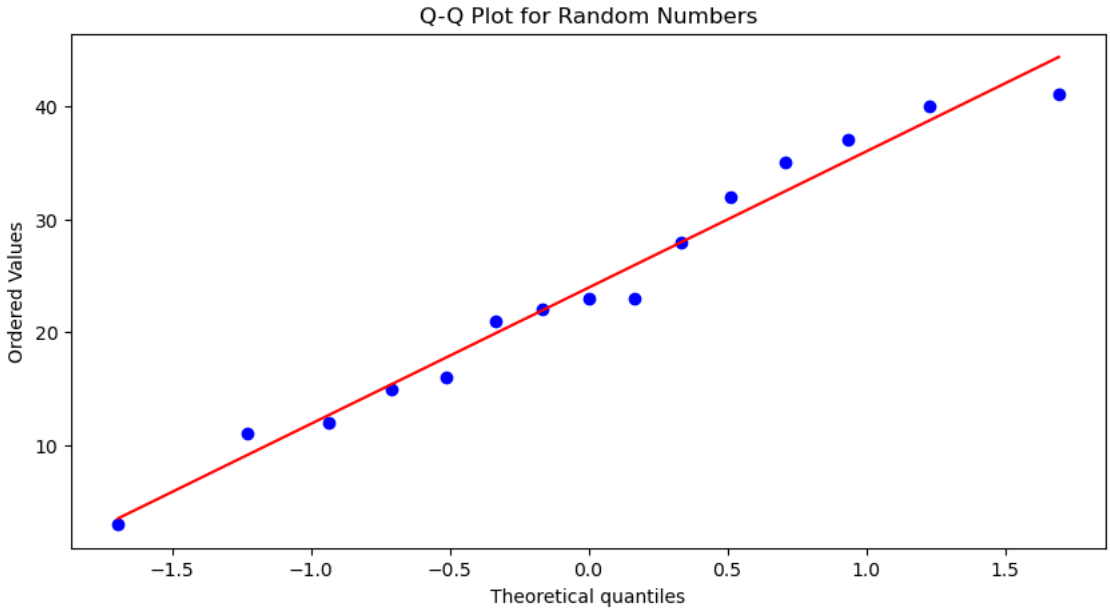
[4]:

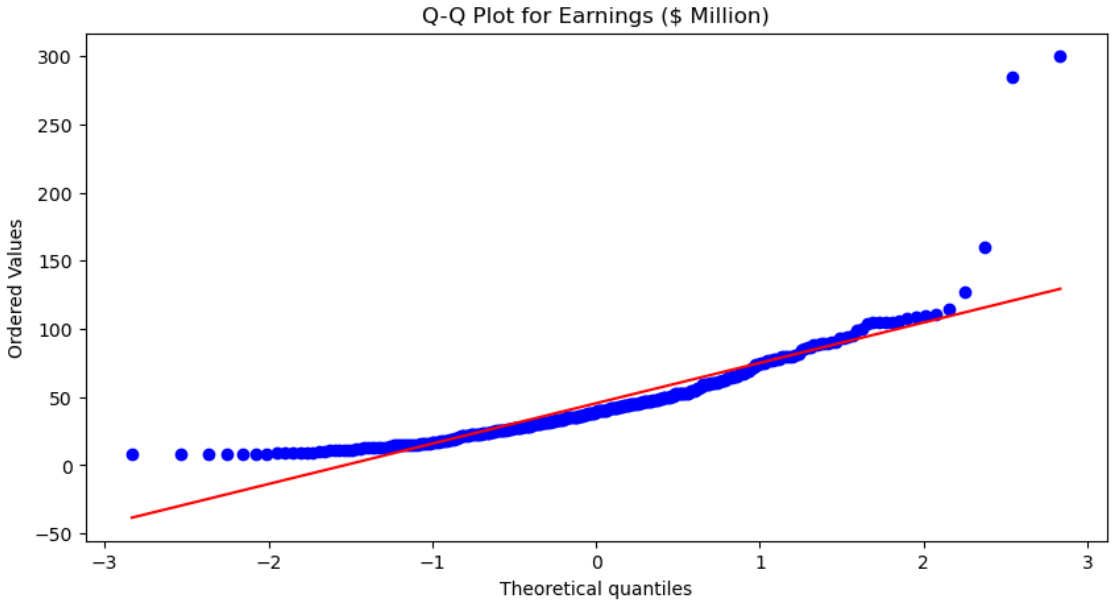
	S.NO	Name	Nationality	Current Rank	Previous Year Rank	\
0	1	Mike Tyson	USA	1	NaN	
1	2	Buster Douglas	USA	2	NaN	
2	3	Sugar Ray Leonard	USA	3	NaN	
3	4	Ayrton Senna	Brazil	4	NaN	
4	5	Alain Prost	France	5	NaN	
..	…	…	…	…	…	
296	297	Stephen Curry	USA	6	9	
297	298	Kevin Durant	USA	7	10	
298	299	Tiger Woods	USA	8	11	
299	300	Kirk Cousins	USA	9	>100	
300	301	Carson Wentz	USA	10	>100	

	Sport	Year	earnings ($ million)
0	boxing	1990	28.6
1	boxing	1990	26.0
2	boxing	1990	13.0
3	auto racing	1990	10.0
4	auto racing	1990	9.0
..	…	…	…
296	Basketball	2020	74.4

```
297         Basketball  2020                 63.9
298               Golf  2020                 62.3
299  American Football  2020                 60.5
300  American Football  2020                 59.1

[301 rows x 8 columns]
```

[5]:
```
import scipy.stats as stats

random_numbers = np.random.randint(1, 51, 15)


plt.figure(figsize=(10, 5))
stats.probplot(random_numbers, dist="norm", plot=plt)
plt.title("Q-Q Plot for Random Numbers")
plt.show()


data2['earnings ($ million)'] = pd.to_numeric(data2['earnings ($ million)'],␣
 ↪errors='coerce')
earnings = data2['earnings ($ million)'].dropna()

plt.figure(figsize=(10, 5))
stats.probplot(earnings, dist="norm", plot=plt)
plt.title("Q-Q Plot for Earnings ($ Million)")
plt.show()


print("Inference:")
print("- The Q-Q plot for random numbers will show whether they follow a normal␣
 ↪distribution.")
print("- The Q-Q plot for earnings will indicate whether the earnings data␣
 ↪aligns with a normal distribution.")
```





Inference:
- The Q-Q plot for random numbers will show whether they follow a normal distribution.
- The Q-Q plot for earnings will indicate whether the earnings data aligns with a normal distribution.

[ ]:

from file: PMRP_5

23AIML010 OM CHOKSI PMRP ASSIGNMENT 5 + CLASSWORK

CLASSWORK QUESTIONS

IPL DATA ANALYTICS

1. Calculate the total number of matches played in each season
2. Find the most successful team (team with the most wins)
3. Find the average margin of victory by wickets and by runs
4. Which player won the most 'Player of the Match awards?
5. Find the number of matches where the toss winner won the match
6. Calculate the total number of runs scored in all matches for each team
7. Determine the average number of wickets taken by the winning team in each match
8. How many matches were decided by a Super Over?
9. Find the distribution of match results (runs vs wickets)
10. Find the top 5 venues with the most matches played
11. Find the match with the highest margin of victory (by wickets or runs)
12. Calculate the win percentage for each team
13. Find the average number of overs played in all matches
14. Find the most common match outcome (runs, wickets, or no result)
15. Find the total number of matches played at each venue by year
16. Analyze the win margin distribution by year
17. Calculate the total number of 'no result' matches and their impact on the tournament
18. How many matches were won by teams batting first vs. batting second?
19. Find out the average number of runs scored by the winning team
20. Identify the most successful captain (team with the most wins under a captain)

[77]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv('D:/SEM4/PMRP/RAW_CODE/PMRP_DAY_13/matches.csv')
df.head,df.tail,df.describe,df.info
```

[77]:
```
(<bound method NDFrame.head of            id   season        city        date
match_type  player_of_match  \
 0      335982  2007/08   Bangalore  2008-04-18       League      BB McCullum
```

```
1      335983  2007/08  Chandigarh  2008-04-19       League       MEK Hussey
2      335984  2007/08       Delhi  2008-04-19       League      MF Maharoof
3      335985  2007/08      Mumbai  2008-04-20       League       MV Boucher
4      335986  2007/08     Kolkata  2008-04-20       League        DJ Hussey
…         …       …          …          …           …              …
1090  1426307     2024   Hyderabad  2024-05-19       League  Abhishek Sharma
1091  1426309     2024   Ahmedabad  2024-05-21  Qualifier 1         MA Starc
1092  1426310     2024   Ahmedabad  2024-05-22   Eliminator         R Ashwin
1093  1426311     2024     Chennai  2024-05-24  Qualifier 2    Shahbaz Ahmed
1094  1426312     2024     Chennai  2024-05-26        Final         MA Starc

                                                venue  \
0                                  M Chinnaswamy Stadium
1               Punjab Cricket Association Stadium, Mohali
2                                       Feroz Shah Kotla
3                                       Wankhede Stadium
4                                           Eden Gardens
…                                                     …
1090  Rajiv Gandhi International Stadium, Uppal, Hyd…
1091                    Narendra Modi Stadium, Ahmedabad
1092                    Narendra Modi Stadium, Ahmedabad
1093            MA Chidambaram Stadium, Chepauk, Chennai
1094            MA Chidambaram Stadium, Chepauk, Chennai

                            team1                         team2  \
0     Royal Challengers Bangalore          Kolkata Knight Riders
1                 Kings XI Punjab            Chennai Super Kings
2                Delhi Daredevils               Rajasthan Royals
3                  Mumbai Indians    Royal Challengers Bangalore
4            Kolkata Knight Riders                Deccan Chargers
…                              …                             …
1090                 Punjab Kings            Sunrisers Hyderabad
1091           Sunrisers Hyderabad          Kolkata Knight Riders
1092  Royal Challengers Bengaluru               Rajasthan Royals
1093           Sunrisers Hyderabad               Rajasthan Royals
1094           Sunrisers Hyderabad          Kolkata Knight Riders

                      toss_winner toss_decision                        winner  \
0     Royal Challengers Bangalore         field          Kolkata Knight Riders
1             Chennai Super Kings           bat            Chennai Super Kings
2                Rajasthan Royals           bat               Delhi Daredevils
3                  Mumbai Indians           bat    Royal Challengers Bangalore
4                  Deccan Chargers           bat          Kolkata Knight Riders
…                              …           …                             …
1090                 Punjab Kings           bat            Sunrisers Hyderabad
1091           Sunrisers Hyderabad           bat          Kolkata Knight Riders
1092              Rajasthan Royals         field               Rajasthan Royals
```

```
1093             Rajasthan Royals         field          Sunrisers Hyderabad
1094           Sunrisers Hyderabad           bat        Kolkata Knight Riders

       result  result_margin  target_runs  target_overs super_over method  \
0        runs          140.0        223.0          20.0          N    NaN
1        runs           33.0        241.0          20.0          N    NaN
2     wickets            9.0        130.0          20.0          N    NaN
3     wickets            5.0        166.0          20.0          N    NaN
4     wickets            5.0        111.0          20.0          N    NaN
…       …             …          …            …        …    …
1090  wickets            4.0        215.0          20.0          N    NaN
1091  wickets            8.0        160.0          20.0          N    NaN
1092  wickets            4.0        173.0          20.0          N    NaN
1093     runs           36.0        176.0          20.0          N    NaN
1094  wickets            8.0        114.0          20.0          N    NaN

                    umpire1              umpire2
0                 Asad Rauf           RE Koertzen
1                 MR Benson            SL Shastri
2                 Aleem Dar        GA Pratapkumar
3                  SJ Davis             DJ Harper
4                 BF Bowden           K Hariharan
…                       …                    …
1090             Nitin Menon             VK Sharma
1091            AK Chaudhary              R Pandit
1092  KN Ananthapadmanabhan  MV Saidharshan Kumar
1093             Nitin Menon             VK Sharma
1094           J Madanagopal           Nitin Menon

[1095 rows x 20 columns]>,
<bound method NDFrame.tail of            id   season        city        date
match_type  player_of_match  \
0      335982  2007/08   Bangalore  2008-04-18       League      BB McCullum
1      335983  2007/08  Chandigarh  2008-04-19       League       MEK Hussey
2      335984  2007/08       Delhi  2008-04-19       League      MF Maharoof
3      335985  2007/08      Mumbai  2008-04-20       League       MV Boucher
4      335986  2007/08     Kolkata  2008-04-20       League        DJ Hussey
…       …      …          …          …           …              …
1090  1426307     2024   Hyderabad  2024-05-19       League  Abhishek Sharma
1091  1426309     2024   Ahmedabad  2024-05-21  Qualifier 1         MA Starc
1092  1426310     2024   Ahmedabad  2024-05-22   Eliminator         R Ashwin
1093  1426311     2024     Chennai  2024-05-24  Qualifier 2    Shahbaz Ahmed
1094  1426312     2024     Chennai  2024-05-26        Final         MA Starc

                                              venue  \
0                              M Chinnaswamy Stadium
1           Punjab Cricket Association Stadium, Mohali
```

```
2                                      Feroz Shah Kotla
3                                      Wankhede Stadium
4                                          Eden Gardens
…                                                   …
1090  Rajiv Gandhi International Stadium, Uppal, Hyd…
1091                    Narendra Modi Stadium, Ahmedabad
1092                    Narendra Modi Stadium, Ahmedabad
1093            MA Chidambaram Stadium, Chepauk, Chennai
1094            MA Chidambaram Stadium, Chepauk, Chennai

                            team1                        team2  \
0     Royal Challengers Bangalore         Kolkata Knight Riders
1                  Kings XI Punjab           Chennai Super Kings
2                Delhi Daredevils              Rajasthan Royals
3                  Mumbai Indians  Royal Challengers Bangalore
4            Kolkata Knight Riders               Deccan Chargers
…                                …                            …
1090                 Punjab Kings           Sunrisers Hyderabad
1091           Sunrisers Hyderabad         Kolkata Knight Riders
1092  Royal Challengers Bengaluru              Rajasthan Royals
1093           Sunrisers Hyderabad              Rajasthan Royals
1094           Sunrisers Hyderabad         Kolkata Knight Riders

                      toss_winner toss_decision                        winner  \
0     Royal Challengers Bangalore         field         Kolkata Knight Riders
1             Chennai Super Kings           bat           Chennai Super Kings
2                Rajasthan Royals           bat              Delhi Daredevils
3                  Mumbai Indians           bat  Royal Challengers Bangalore
4                 Deccan Chargers           bat         Kolkata Knight Riders
…                              …           …                             …
1090                Punjab Kings           bat           Sunrisers Hyderabad
1091          Sunrisers Hyderabad           bat         Kolkata Knight Riders
1092             Rajasthan Royals         field              Rajasthan Royals
1093             Rajasthan Royals         field           Sunrisers Hyderabad
1094          Sunrisers Hyderabad           bat         Kolkata Knight Riders

       result  result_margin  target_runs  target_overs super_over method  \
0        runs          140.0        223.0          20.0          N    NaN
1        runs           33.0        241.0          20.0          N    NaN
2     wickets            9.0        130.0          20.0          N    NaN
3     wickets            5.0        166.0          20.0          N    NaN
4     wickets            5.0        111.0          20.0          N    NaN
…       …            …          …           …        …    …
1090  wickets            4.0        215.0          20.0          N    NaN
1091  wickets            8.0        160.0          20.0          N    NaN
1092  wickets            4.0        173.0          20.0          N    NaN
1093     runs           36.0        176.0          20.0          N    NaN
```

```
1094  wickets            8.0        114.0          20.0          N    NaN

                    umpire1              umpire2
0                 Asad Rauf           RE Koertzen
1                 MR Benson            SL Shastri
2                 Aleem Dar        GA Pratapkumar
3                  SJ Davis             DJ Harper
4                 BF Bowden           K Hariharan
…                       …                    …
1090            Nitin Menon             VK Sharma
1091          AK Chaudhary              R Pandit
1092  KN Ananthapadmanabhan  MV Saidharshan Kumar
1093            Nitin Menon             VK Sharma
1094         J Madanagopal           Nitin Menon

[1095 rows x 20 columns]>,
<bound method NDFrame.describe of            id   season        city
date   match_type  player_of_match  \
0      335982  2007/08   Bangalore  2008-04-18       League      BB McCullum
1      335983  2007/08  Chandigarh  2008-04-19       League       MEK Hussey
2      335984  2007/08       Delhi  2008-04-19       League      MF Maharoof
3      335985  2007/08      Mumbai  2008-04-20       League       MV Boucher
4      335986  2007/08     Kolkata  2008-04-20       League        DJ Hussey
…       …      …           …          …           …              …
1090  1426307     2024   Hyderabad  2024-05-19       League  Abhishek Sharma
1091  1426309     2024   Ahmedabad  2024-05-21  Qualifier 1         MA Starc
1092  1426310     2024   Ahmedabad  2024-05-22   Eliminator         R Ashwin
1093  1426311     2024     Chennai  2024-05-24  Qualifier 2    Shahbaz Ahmed
1094  1426312     2024     Chennai  2024-05-26        Final         MA Starc

                                              venue  \
0                                 M Chinnaswamy Stadium
1            Punjab Cricket Association Stadium, Mohali
2                                     Feroz Shah Kotla
3                                     Wankhede Stadium
4                                         Eden Gardens
…                                                  …
1090  Rajiv Gandhi International Stadium, Uppal, Hyd…
1091                    Narendra Modi Stadium, Ahmedabad
1092                    Narendra Modi Stadium, Ahmedabad
1093           MA Chidambaram Stadium, Chepauk, Chennai
1094           MA Chidambaram Stadium, Chepauk, Chennai

                           team1                        team2  \
0     Royal Challengers Bangalore        Kolkata Knight Riders
1                 Kings XI Punjab          Chennai Super Kings
2               Delhi Daredevils             Rajasthan Royals
```

```
3                  Mumbai Indians  Royal Challengers Bangalore
4           Kolkata Knight Riders              Deccan Chargers
…                             …                           …
1090                Punjab Kings          Sunrisers Hyderabad
1091          Sunrisers Hyderabad        Kolkata Knight Riders
1092  Royal Challengers Bengaluru             Rajasthan Royals
1093          Sunrisers Hyderabad             Rajasthan Royals
1094          Sunrisers Hyderabad        Kolkata Knight Riders

                      toss_winner toss_decision                        winner  \
0     Royal Challengers Bangalore         field         Kolkata Knight Riders
1             Chennai Super Kings           bat           Chennai Super Kings
2                Rajasthan Royals           bat              Delhi Daredevils
3                  Mumbai Indians           bat  Royal Challengers Bangalore
4                 Deccan Chargers           bat         Kolkata Knight Riders
…                             …           …                            …
1090                Punjab Kings           bat           Sunrisers Hyderabad
1091          Sunrisers Hyderabad           bat         Kolkata Knight Riders
1092             Rajasthan Royals         field              Rajasthan Royals
1093             Rajasthan Royals         field           Sunrisers Hyderabad
1094          Sunrisers Hyderabad           bat         Kolkata Knight Riders

       result  result_margin  target_runs  target_overs super_over method  \
0        runs          140.0        223.0          20.0          N    NaN
1        runs           33.0        241.0          20.0          N    NaN
2     wickets            9.0        130.0          20.0          N    NaN
3     wickets            5.0        166.0          20.0          N    NaN
4     wickets            5.0        111.0          20.0          N    NaN
…       …            …          …           …        …    …
1090  wickets            4.0        215.0          20.0          N    NaN
1091  wickets            8.0        160.0          20.0          N    NaN
1092  wickets            4.0        173.0          20.0          N    NaN
1093     runs           36.0        176.0          20.0          N    NaN
1094  wickets            8.0        114.0          20.0          N    NaN

                    umpire1              umpire2
0                 Asad Rauf           RE Koertzen
1                 MR Benson            SL Shastri
2                 Aleem Dar        GA Pratapkumar
3                  SJ Davis             DJ Harper
4                 BF Bowden           K Hariharan
…                       …                     …
1090           Nitin Menon             VK Sharma
1091          AK Chaudhary              R Pandit
1092  KN Ananthapadmanabhan  MV Saidharshan Kumar
1093           Nitin Menon             VK Sharma
1094         J Madanagopal           Nitin Menon
```

```
 [1095 rows x 20 columns]>,
 <bound method DataFrame.info of            id   season        city        date
match_type  player_of_match  \
 0      335982  2007/08   Bangalore  2008-04-18       League      BB McCullum
 1      335983  2007/08  Chandigarh  2008-04-19       League       MEK Hussey
 2      335984  2007/08       Delhi  2008-04-19       League      MF Maharoof
 3      335985  2007/08      Mumbai  2008-04-20       League       MV Boucher
 4      335986  2007/08     Kolkata  2008-04-20       League        DJ Hussey
 …         …       …           …           …           …              …
 1090  1426307     2024   Hyderabad  2024-05-19       League  Abhishek Sharma
 1091  1426309     2024   Ahmedabad  2024-05-21  Qualifier 1         MA Starc
 1092  1426310     2024   Ahmedabad  2024-05-22   Eliminator         R Ashwin
 1093  1426311     2024     Chennai  2024-05-24  Qualifier 2    Shahbaz Ahmed
 1094  1426312     2024     Chennai  2024-05-26        Final         MA Starc

                                                venue  \
 0                                  M Chinnaswamy Stadium
 1            Punjab Cricket Association Stadium, Mohali
 2                                       Feroz Shah Kotla
 3                                       Wankhede Stadium
 4                                           Eden Gardens
 …                                                    …
 1090  Rajiv Gandhi International Stadium, Uppal, Hyd…
 1091                    Narendra Modi Stadium, Ahmedabad
 1092                    Narendra Modi Stadium, Ahmedabad
 1093           MA Chidambaram Stadium, Chepauk, Chennai
 1094           MA Chidambaram Stadium, Chepauk, Chennai

                             team1                        team2  \
 0     Royal Challengers Bangalore        Kolkata Knight Riders
 1                  Kings XI Punjab          Chennai Super Kings
 2                Delhi Daredevils             Rajasthan Royals
 3                  Mumbai Indians  Royal Challengers Bangalore
 4            Kolkata Knight Riders              Deccan Chargers
 …                              …                            …
 1090                 Punjab Kings          Sunrisers Hyderabad
 1091           Sunrisers Hyderabad        Kolkata Knight Riders
 1092  Royal Challengers Bengaluru             Rajasthan Royals
 1093           Sunrisers Hyderabad             Rajasthan Royals
 1094           Sunrisers Hyderabad        Kolkata Knight Riders

                      toss_winner toss_decision                       winner  \
 0     Royal Challengers Bangalore         field        Kolkata Knight Riders
 1             Chennai Super Kings           bat          Chennai Super Kings
 2                Rajasthan Royals           bat             Delhi Daredevils
 3                  Mumbai Indians           bat  Royal Challengers Bangalore
```

```
4                 Deccan Chargers           bat        Kolkata Knight Riders
…                             …           …                           …
1090                Punjab Kings            bat          Sunrisers Hyderabad
1091          Sunrisers Hyderabad           bat        Kolkata Knight Riders
1092             Rajasthan Royals         field             Rajasthan Royals
1093             Rajasthan Royals         field          Sunrisers Hyderabad
1094          Sunrisers Hyderabad           bat        Kolkata Knight Riders

       result  result_margin  target_runs  target_overs super_over method  \
0        runs          140.0        223.0          20.0          N    NaN
1        runs           33.0        241.0          20.0          N    NaN
2     wickets            9.0        130.0          20.0          N    NaN
3     wickets            5.0        166.0          20.0          N    NaN
4     wickets            5.0        111.0          20.0          N    NaN
…       …            …          …           …        …    …
1090  wickets            4.0        215.0          20.0          N    NaN
1091  wickets            8.0        160.0          20.0          N    NaN
1092  wickets            4.0        173.0          20.0          N    NaN
1093     runs           36.0        176.0          20.0          N    NaN
1094  wickets            8.0        114.0          20.0          N    NaN

                    umpire1              umpire2
0                 Asad Rauf           RE Koertzen
1                 MR Benson            SL Shastri
2                 Aleem Dar        GA Pratapkumar
3                  SJ Davis             DJ Harper
4                 BF Bowden           K Hariharan
…                       …                    …
1090            Nitin Menon            VK Sharma
1091          AK Chaudhary             R Pandit
1092  KN Ananthapadmanabhan  MV Saidharshan Kumar
1093            Nitin Menon            VK Sharma
1094          J Madanagopal           Nitin Menon

[1095 rows x 20 columns]>)
```

1.Calculate the total number of Matches Played in Each Session

[78]:
```
matches_per_season = df['season'].value_counts().sort_index()
print(matches_per_season)
```

```
season
2007/08    58
2009       57
2009/10    60
2011       73
2012       74
2013       76
```

```
2014        60
2015        59
2016        60
2017        59
2018        60
2019        60
2020/21     60
2021        60
2022        74
2023        74
2024        71
Name: count, dtype: int64
```

2. Find the Most Successfull team (team with most runs)

[79]:
```
# runs_df = df[df['result'] == 'runs']


# most_successful_team = runs_df.groupby('winner')['result_margin'].sum().
  ↪idxmax()
# print(f"The most successful team (team with most runs) is:␣
  ↪{most_successful_team}")

df["winner"].value_counts()
```

[79]:
```
winner
Mumbai Indians                 144
Chennai Super Kings            138
Kolkata Knight Riders          131
Royal Challengers Bangalore    116
Rajasthan Royals               112
Kings XI Punjab                 88
Sunrisers Hyderabad             88
Delhi Daredevils                67
Delhi Capitals                  48
Deccan Chargers                 29
Gujarat Titans                  28
Lucknow Super Giants            24
Punjab Kings                    24
Gujarat Lions                   13
Pune Warriors                   12
Rising Pune Supergiant          10
Royal Challengers Bengaluru      7
Kochi Tuskers Kerala             6
Rising Pune Supergiants          5
Name: count, dtype: int64
```

3. Find the average margin of victory by wickets and runs

[80]:
```
average_runs_margin = df[df['result'] == 'runs']['result_margin'].mean()
average_wickets_margin = df[df['result'] == 'wickets']['result_margin'].mean()
print(f'Average margin of victory by runs: {average_runs_margin}')
print(f'Average margin of victory by wickets: {average_wickets_margin}')
```

```
Average margin of victory by runs: 30.104417670682732
Average margin of victory by wickets: 6.192041522491349
```

4. Which player won the most 'Player of the Match awards?

[81]:
```
most_player_of_match = df['player_of_match'].value_counts().idxmax()
print(f"The player who won the most 'Player of the Match' awards is:␣
↪{most_player_of_match}")
```

```
The player who won the most 'Player of the Match' awards is: AB de Villiers
```

5. Find the number of matches where the toss winner won the match

[82]:
```
toss_winner_matches = df[df['toss_winner'] == df['winner']].shape[0]
print(f"The number of matches where the toss winner won the match:␣
↪{toss_winner_matches}")
```

```
The number of matches where the toss winner won the match: 554
```

6. Calculate the total number of runs scored in all matches for each team

[83]:
```
total_runs_per_team = df.groupby('team1')['target_runs'].sum() + df.
↪groupby('team2')['target_runs'].sum()
print(total_runs_per_team)
```

```
team1
Chennai Super Kings            39503.0
Deccan Chargers                12047.0
Delhi Capitals                 15930.0
Delhi Daredevils               25492.0
Gujarat Lions                   5077.0
Gujarat Titans                  7865.0
Kings XI Punjab                31391.0
Kochi Tuskers Kerala            2014.0
Kolkata Knight Riders          40557.0
Lucknow Super Giants            7835.0
Mumbai Indians                 43728.0
Pune Warriors                   6950.0
Punjab Kings                    9787.0
Rajasthan Royals               36250.0
Rising Pune Supergiant          2571.0
Rising Pune Supergiants         1993.0
Royal Challengers Bangalore    39807.0
Royal Challengers Bengaluru     2986.0
Sunrisers Hyderabad            30071.0
Name: target_runs, dtype: float64
```

7. Determine the average number of wickets taken by the winning team in each match

[84]:
```
average_wickets_taken = df[df['result'] == 'wickets']['result_margin'].mean()
print(f'The average number of wickets taken by the winning team in each match␣
↪is: {average_wickets_taken}')
```

```
The average number of wickets taken by the winning team in each match is:
6.192041522491349
```

8. How many matches were decided by a Super Over?

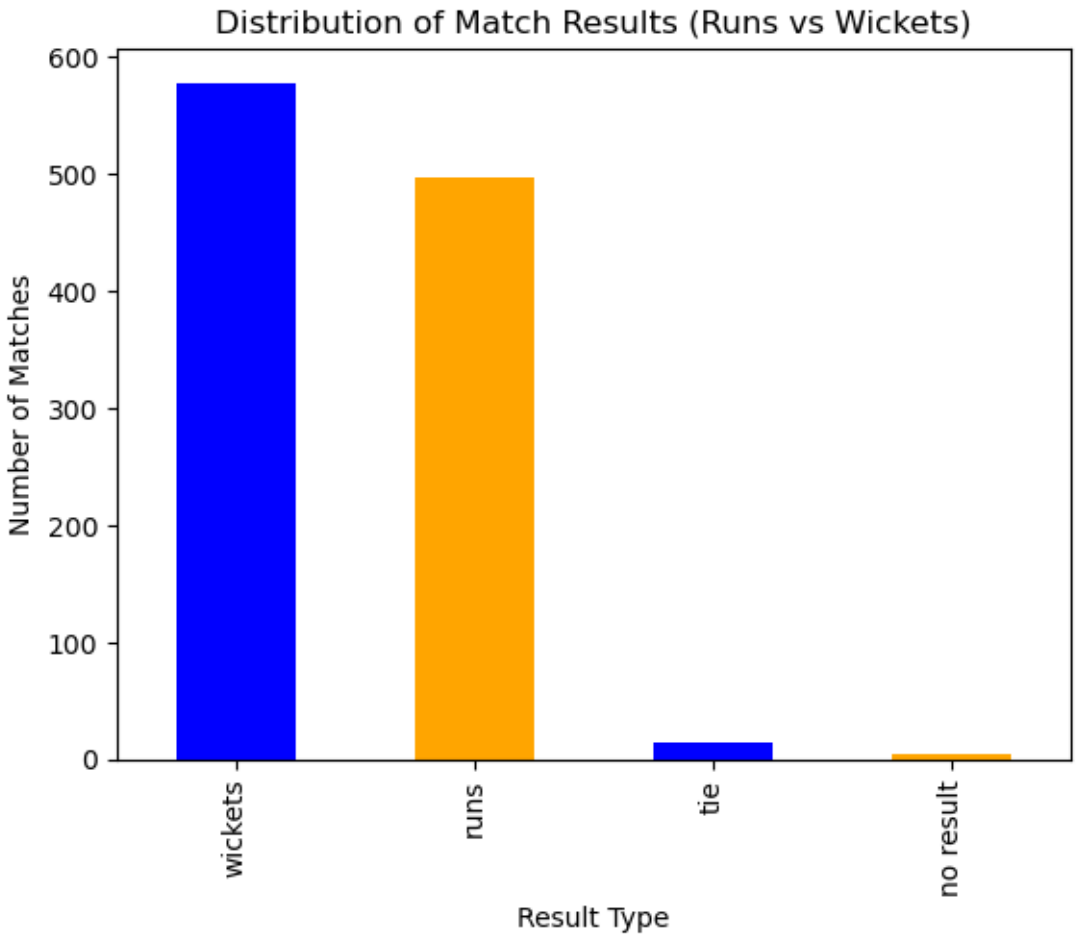
[85]:
```
super_over_matches = df[df['super_over'] == 'Y'].shape[0]
print(f"The number of matches decided by a Super Over: {super_over_matches}")
```

```
The number of matches decided by a Super Over: 14
```

9. Find the distribution of match results (runs vs wickets)

[86]:
```
result_distribution = df['result'].value_counts()
print(result_distribution)

# Plotting the distribution
result_distribution.plot(kind='bar', color=['blue', 'orange'])
plt.title('Distribution of Match Results (Runs vs Wickets)')
plt.xlabel('Result Type')
plt.ylabel('Number of Matches')
plt.show()
```

```
result
wickets      578
runs         498
tie           14
no result      5
Name: count, dtype: int64
```



10. Find the top 5 venues with the most matches played

[87]:
```
top_venues = df['venue'].value_counts().head(5)
print(top_venues)

# # Plotting the top 5 venues
# top_venues.plot(kind='bar', color='green')
# plt.title('Top 5 Venues with the Most Matches Played')
# plt.xlabel('Venue')
# plt.ylabel('Number of Matches')
# plt.show()
```

```
venue
Eden Gardens                                77
Wankhede Stadium                            73
M Chinnaswamy Stadium                       65
Feroz Shah Kotla                            60
Rajiv Gandhi International Stadium, Uppal   49
```

```
Name: count, dtype: int64
```

11. Find the match with the highest margin of victory (by wickets or runs)

[88]:
```
df[df['result_margin']==df['result_margin'].max()]
```

[88]:
```
           id season   city        date match_type player_of_match  \
620  1082635   2017  Delhi  2017-05-06     League     LMP Simmons

                venue             team1           team2       toss_winner  \
620  Feroz Shah Kotla  Delhi Daredevils  Mumbai Indians  Delhi Daredevils

    toss_decision          winner result  result_margin  target_runs  \
620         field  Mumbai Indians   runs          146.0        213.0

     target_overs super_over method      umpire1    umpire2
620          20.0          N    NaN  Nitin Menon  CK Nandan
```

[89]:
```
# Find the match with the highest margin of victory (by wickets or runs)
df_wickets=df[df['result']=='wickets']
df_runs=df[df['result']=='runs']

max_margin_wicket=df_wickets.loc[df_wickets['result_margin'].idxmax()]

max_margin_run=df_runs.loc[df_runs['result_margin'].idxmax()]

max_margin_run,max_margin_wicket
```

[89]:
```
(id                         1082635
 season                        2017
 city                         Delhi
 date                    2017-05-06
 match_type                  League
 player_of_match         LMP Simmons
 venue              Feroz Shah Kotla
 team1              Delhi Daredevils
 team2                Mumbai Indians
 toss_winner        Delhi Daredevils
 toss_decision                 field
 winner               Mumbai Indians
 result                          runs
 result_margin                146.0
 target_runs                  213.0
 target_overs                  20.0
 super_over                       N
 method                         NaN
 umpire1                Nitin Menon
 umpire2                  CK Nandan
```

```
Name: 620, dtype: object,
id                                    335994
season                                2007/08
city                                   Mumbai
date                               2008-04-27
match_type                             League
player_of_match                 AC Gilchrist
venue              Dr DY Patil Sports Academy
team1                           Mumbai Indians
team2                          Deccan Chargers
toss_winner                    Deccan Chargers
toss_decision                           field
winner                         Deccan Chargers
result                                 wickets
result_margin                             10.0
target_runs                              155.0
target_overs                              20.0
super_over                                   N
method                                     NaN
umpire1                            Asad Rauf
umpire2                           SL Shastri
Name: 12, dtype: object)
```

12. Calculate the win percentage for each team

[90]:
```
matches_played = df['team1'].value_counts() + df['team2'].value_counts()

matches_won = df['winner'].value_counts()
win_percentage = (matches_won / matches_played) * 100

print(win_percentage)
```

```
Chennai Super Kings            57.983193
Deccan Chargers                38.666667
Delhi Capitals                 52.747253
Delhi Daredevils               41.614907
Gujarat Lions                  43.333333
Gujarat Titans                 62.222222
Kings XI Punjab                46.315789
Kochi Tuskers Kerala           42.857143
Kolkata Knight Riders          52.191235
Lucknow Super Giants           54.545455
Mumbai Indians                 55.172414
Pune Warriors                  26.086957
Punjab Kings                   42.857143
Rajasthan Royals               50.678733
Rising Pune Supergiant         62.500000
Rising Pune Supergiants        35.714286
```

```
Royal Challengers Bangalore    48.333333
Royal Challengers Bengaluru    46.666667
Sunrisers Hyderabad            48.351648
Name: count, dtype: float64
```

13. Find the average number of overs played in all matches

[91]:
```
average_overs_played = df['target_overs'].mean()

print(f'The average number of overs played in all matches is:␣
 ↪{average_overs_played}')
```

```
The average number of overs played in all matches is: 19.75934065934066
```

14. Find the most common match outcome (runs, wickets, or no result)

[92]:
```
most_common_outcome = result_distribution.idxmax()
print(f'The most common match outcome is: {most_common_outcome}')
```

```
The most common match outcome is: wickets
```

15. Find the total number of matches played at each venue by year

[93]:
```
matches_per_venue_year = df.groupby(['season','venue']).size()
print(matches_per_venue_year)
```

```
season   venue
2007/08  Dr DY Patil Sports Academy                                    4
         Eden Gardens                                                  7
         Feroz Shah Kotla                                              6
         M Chinnaswamy Stadium                                         7
         MA Chidambaram Stadium, Chepauk                               7
                                                                      ..
2024     Maharaja Yadavindra Singh International Cricket Stadium, Mullanpur    5
         Narendra Modi Stadium, Ahmedabad                              8
         Rajiv Gandhi International Stadium, Uppal, Hyderabad          6
         Sawai Mansingh Stadium, Jaipur                                5
         Wankhede Stadium, Mumbai                                      7
Length: 175, dtype: int64
```

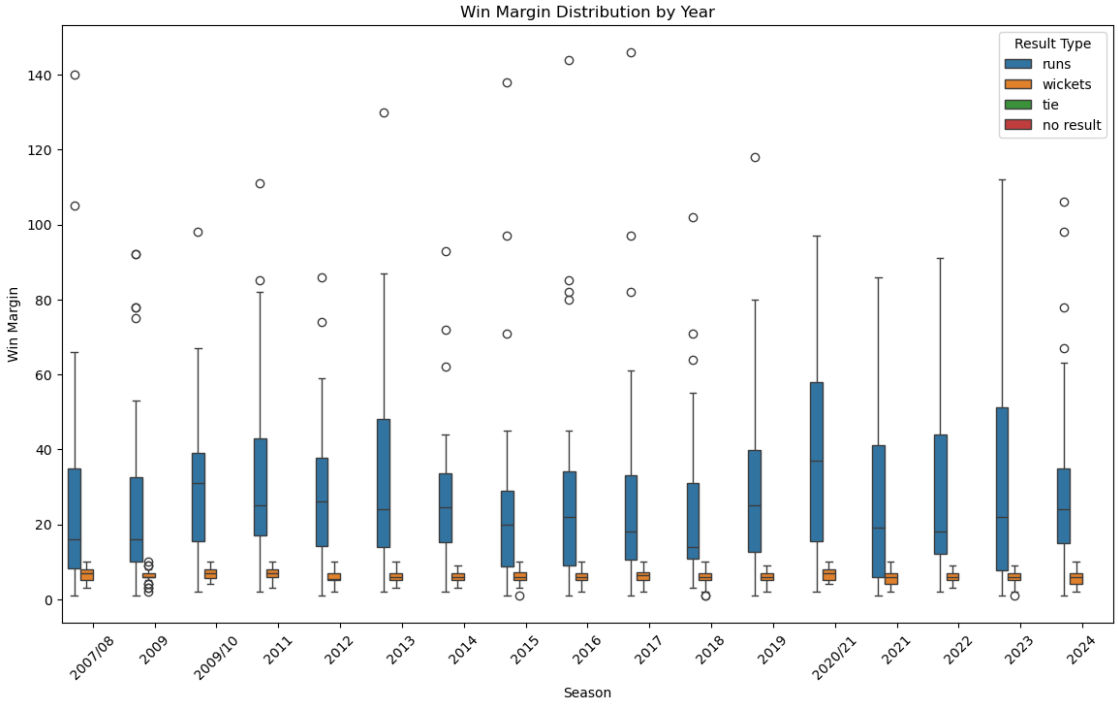
16. Analyze the win margin distribution by year

[94]:
```
# Grouping the data by season and result type
win_margin_by_year = df.groupby(['season', 'result'])['result_margin'].
 ↪describe()
print(win_margin_by_year)
# Plotting the win margin distribution by year
plt.figure(figsize=(14, 8))
sns.boxplot(x='season', y='result_margin', hue='result', data=df)
plt.title('Win Margin Distribution by Year')
plt.xlabel('Season')
```

```
plt.ylabel('Win Margin')
plt.xticks(rotation=45)
plt.legend(title='Result Type')
plt.show()
```

		count	mean	std	min	25%	50%	75%	max
season	result								
2007/08	runs	24.0	29.375000	34.291351	1.0	8.25	16.0	35.00	140.0
	wickets	34.0	6.500000	2.078024	3.0	5.00	7.0	8.00	10.0
2009	runs	27.0	28.296296	28.894789	1.0	10.00	16.0	32.50	92.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	29.0	6.206897	1.820112	2.0	6.00	6.0	7.00	10.0
2009/10	runs	31.0	31.483871	20.990269	2.0	15.50	31.0	39.00	98.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	28.0	6.785714	1.571909	4.0	5.75	7.0	8.00	10.0
2011	no result	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	runs	33.0	33.272727	26.081929	2.0	17.00	25.0	43.00	111.0
	wickets	39.0	6.794872	1.794428	3.0	6.00	7.0	8.00	10.0
2012	runs	34.0	28.235294	19.645431	1.0	14.25	26.0	37.75	86.0
	wickets	40.0	6.025000	1.716996	2.0	5.00	5.5	7.00	10.0
2013	runs	37.0	33.540541	28.657551	2.0	14.00	24.0	48.00	130.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	37.0	6.135135	1.669367	3.0	5.00	6.0	7.00	10.0
2014	runs	22.0	29.272727	22.416367	2.0	15.25	24.5	33.50	93.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	37.0	6.081081	1.516179	3.0	5.00	6.0	7.00	9.0
2015	no result	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	runs	32.0	26.562500	28.598373	1.0	8.75	20.0	29.00	138.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	24.0	6.166667	2.219805	1.0	5.00	6.0	7.25	10.0
2016	runs	21.0	32.190476	36.347791	1.0	9.00	22.0	34.00	144.0
	wickets	39.0	6.256410	1.772865	2.0	5.00	6.0	7.00	10.0
2017	runs	26.0	30.307692	33.638988	1.0	10.50	18.0	33.00	146.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	32.0	6.375000	1.896516	2.0	5.00	6.5	7.25	10.0
2018	runs	28.0	24.107143	23.850366	3.0	10.75	14.0	31.00	102.0
	wickets	32.0	5.812500	2.206113	1.0	5.00	6.0	7.00	10.0
2019	no result	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	runs	22.0	30.227273	27.194068	1.0	12.50	25.0	39.75	118.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	35.0	5.771429	1.646488	2.0	5.00	6.0	7.00	9.0
2020/21	runs	27.0	39.370370	26.716673	2.0	15.50	37.0	58.00	97.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	29.0	6.965517	1.762360	4.0	5.00	7.0	8.00	10.0
2021	runs	22.0	26.454545	24.039110	1.0	6.00	19.0	41.00	86.0
	tie	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	wickets	37.0	5.918919	2.019053	2.0	4.00	6.0	7.00	10.0
2022	runs	37.0	27.945946	23.085525	2.0	12.00	18.0	44.00	91.0

	wickets	37.0	6.000000	1.615893	3.0	5.00	6.0	7.00	9.0
2023	no result	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
	runs	40.0	30.400000	27.554887	1.0	7.75	22.0	51.25	112.0
	wickets	33.0	5.727273	1.908414	1.0	5.00	6.0	7.00	9.0
2024	runs	35.0	30.142857	25.994505	1.0	15.00	24.0	35.00	106.0
	wickets	36.0	5.944444	1.999206	2.0	4.00	6.0	7.00	10.0



17. Calculate the total number of 'no result' matches and their impact on the tournament

[95]:
```
# Calculate the total number of 'no result' matches
no_result_matches = df[df['result'] == 'no result'].shape[0]
print(f"The total number of 'no result' matches: {no_result_matches}")

# Analyze the distribution of 'no result' matches by season
no_result_by_season = df[df['result'] == 'no result']['season'].value_counts().
 ↪sort_index()
print("Distribution of 'no result' matches by season:")
print(no_result_by_season)

# Analyze the distribution of 'no result' matches by team
no_result_by_team = df[df['result'] == 'no result']['team1'].value_counts() +␣
 ↪df[df['result'] == 'no result']['team2'].value_counts()
print("Distribution of 'no result' matches by team:")
print(no_result_by_team)
```

```
The total number of 'no result' matches: 5
Distribution of 'no result' matches by season:
season
2011    1
2015    2
2019    1
2023    1
Name: count, dtype: int64
Distribution of 'no result' matches by team:
Chennai Super Kings            NaN
Delhi Daredevils               2.0
Lucknow Super Giants           NaN
Pune Warriors                  NaN
Rajasthan Royals               NaN
Royal Challengers Bangalore    NaN
Name: count, dtype: float64
```

18. How many matches were won by teams batting first vs. batting second?

[96]:
```
# Matches won by teams batting first
batting_first_wins = df[(df['toss_decision'] == 'bat') & (df['toss_winner'] ==␣
 ↪df['winner'])].shape[0] + \
                      df[(df['toss_decision'] == 'field') & (df['toss_winner'] !
 ↪= df['winner'])].shape[0]

# Matches won by teams batting second
batting_second_wins = df[(df['toss_decision'] == 'field') & (df['toss_winner']␣
 ↪== df['winner'])].shape[0] + \
                       df[(df['toss_decision'] == 'bat') & (df['toss_winner'] !=␣
 ↪df['winner'])].shape[0]

print(f"Matches won by teams batting first: {batting_first_wins}")
print(f"Matches won by teams batting second: {batting_second_wins}")
```

```
Matches won by teams batting first: 504
Matches won by teams batting second: 591
```

19. Find out the average number of runs scored by the winning team

[97]:
```
average_runs_scored_by_winning_team = df[df['result'] == 'runs']['target_runs'].
 ↪mean()
print(f'The average number of runs scored by the winning team is:␣
 ↪{average_runs_scored_by_winning_team}')
```

```
The average number of runs scored by the winning team is: 179.69678714859438
```

20. Identify the most unsuccessful team (team with lowest wins)

[98]:
```
most_unsuccessful_team = matches_won.idxmin()
print(f"The most unsuccessful team (team with the lowest wins) is:␣
↪{most_unsuccessful_team}")
```

```
The most unsuccessful team (team with the lowest wins) is: Rising Pune
Supergiants
```

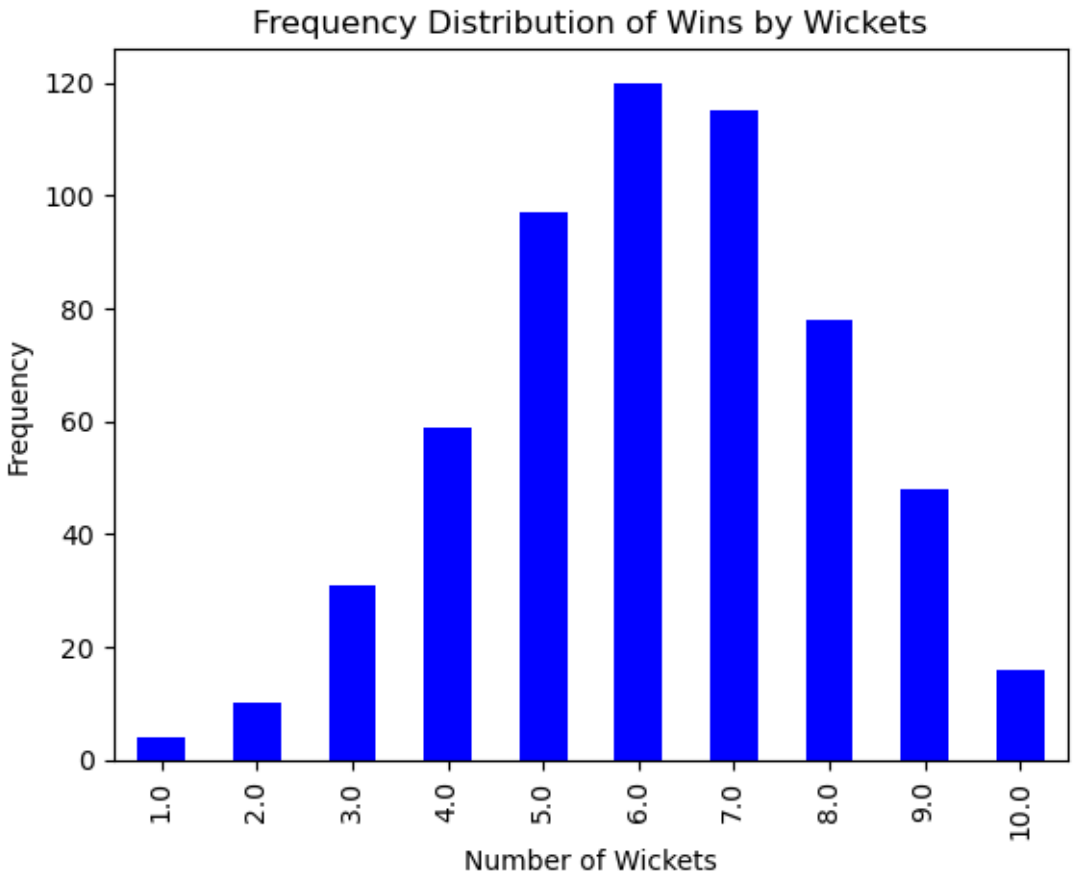
ASSIGNMENT QUESTIONS

Explore following for given dataset and also perform EDA. 1. Frequency Distribution of Wins by Wickets 2. Relative Frequency Distribution 3. Cumulative Relative Frequency Graph 4. Probability of Winning by 6 Wickets or Less 5. Normal Distribution of Wins by Wickets 6. Mean, Standard Deviation, and Percentile Calculation 7. Find out outliers for the selective columns for lower range outliers will be lower than mu - 2sigma, similarly for upper range outliers will be greater than mu+2sigma.

1. Frequency Distribution of Wins by Wickets

[107]:
```
# Frequency distribution of wins by wickets
wins_by_wickets = df_wickets['result_margin'].value_counts().sort_index()
print(wins_by_wickets)

# Plotting the frequency distribution
wins_by_wickets.plot(kind='bar', color='blue')
plt.title('Frequency Distribution of Wins by Wickets')
plt.xlabel('Number of Wickets')
plt.ylabel('Frequency')
plt.show()
```

```
result_margin
1.0       4
2.0      10
3.0      31
4.0      59
5.0      97
6.0     120
7.0     115
8.0      78
9.0      48
10.0     16
Name: count, dtype: int64
```

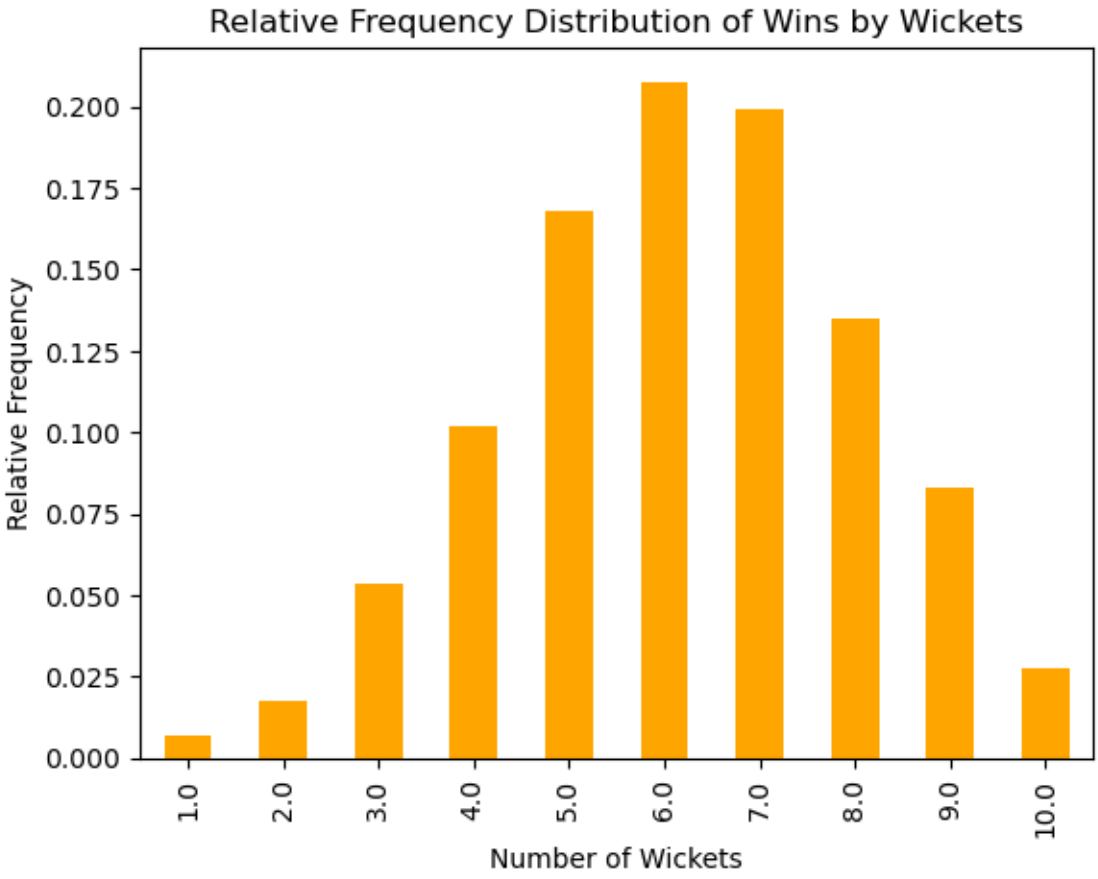


2. Relative Frequency Distribution

[109]:
```
relative_frequency_wins_by_wickets = wins_by_wickets / wins_by_wickets.sum()
print(relative_frequency_wins_by_wickets)

# Plotting the relative frequency distribution
relative_frequency_wins_by_wickets.plot(kind='bar', color='orange')
plt.title('Relative Frequency Distribution of Wins by Wickets')
plt.xlabel('Number of Wickets')
plt.ylabel('Relative Frequency')
plt.show()
```

```
result_margin
1.0     0.006920
2.0     0.017301
3.0     0.053633
4.0     0.102076
5.0     0.167820
6.0     0.207612
7.0     0.198962
```

```
8.0     0.134948
9.0     0.083045
10.0    0.027682
Name: count, dtype: float64
```

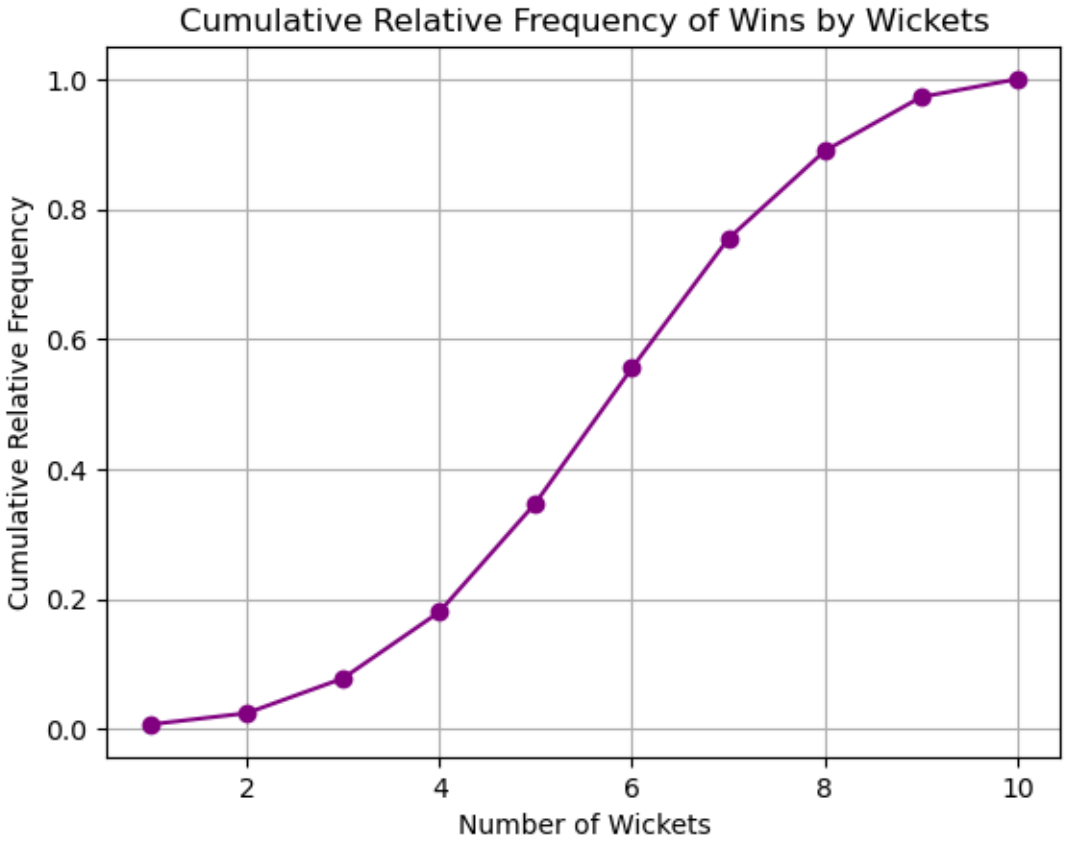


3. Cumulative Relative Frequency Graph

[110]:
```
# Calculate the cumulative relative frequency
cumulative_relative_frequency = relative_frequency_wins_by_wickets.cumsum()
print(cumulative_relative_frequency)

# Plotting the cumulative relative frequency graph
cumulative_relative_frequency.plot(kind='line', marker='o', color='purple')
plt.title('Cumulative Relative Frequency of Wins by Wickets')
plt.xlabel('Number of Wickets')
plt.ylabel('Cumulative Relative Frequency')
plt.grid(True)
plt.show()
```

```
result_margin
```

```
1.0     0.006920
2.0     0.024221
3.0     0.077855
4.0     0.179931
5.0     0.347751
6.0     0.555363
7.0     0.754325
8.0     0.889273
9.0     0.972318
10.0    1.000000
Name: count, dtype: float64
```



4. Probability of Winning by 6 Wickets or Less

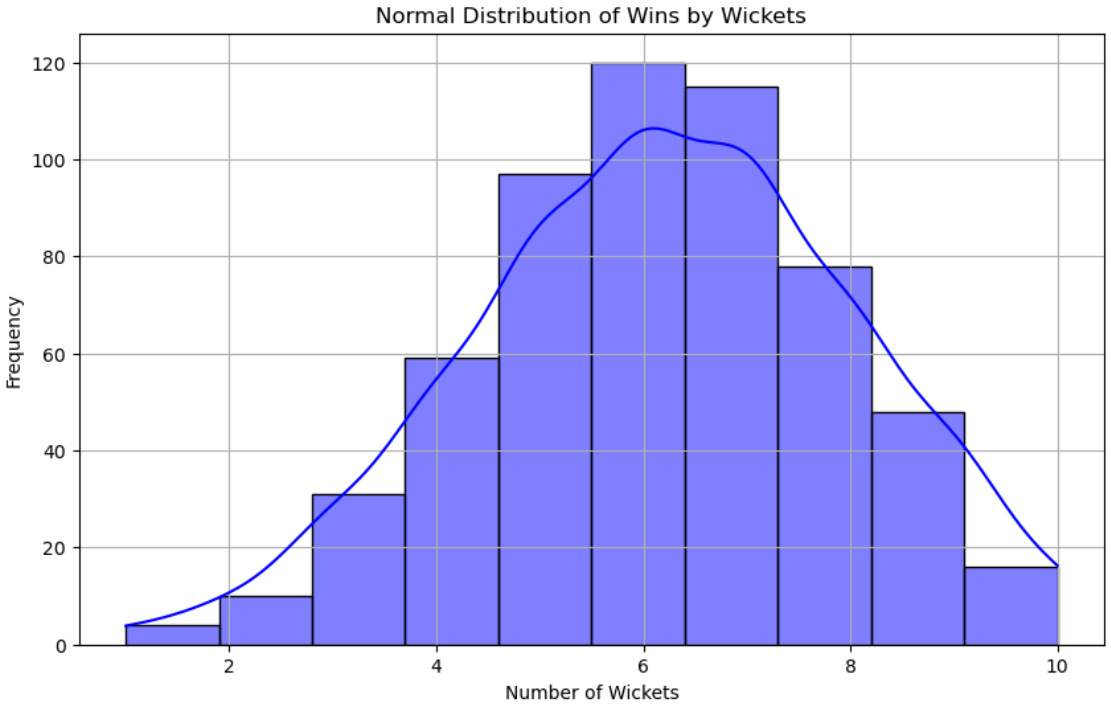
[111]:
```
# Calculate the total number of wins by wickets
total_wins_by_wickets = wins_by_wickets.sum()

# Calculate the number of wins by 6 wickets or less
wins_by_6_or_less = wins_by_wickets[wins_by_wickets.index <= 6].sum()
```

```
# Calculate the probability
probability_wins_by_6_or_less = wins_by_6_or_less / total_wins_by_wickets
print(f'The probability of winning by 6 wickets or less is:␣
  ↪{probability_wins_by_6_or_less}')
```

```
The probability of winning by 6 wickets or less is: 0.5553633217993079
```

5. Normal Distribution of Wins by Wickets

[112]:
```
# Plotting the normal distribution of wins by wickets
plt.figure(figsize=(10, 6))
sns.histplot(df_wickets['result_margin'], kde=True, bins=10, color='blue')
plt.title('Normal Distribution of Wins by Wickets')
plt.xlabel('Number of Wickets')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```



6. Mean, Standard Deviation, and Percentile Calculation

[116]:
```
print(df.describe())
```

```
                id  result_margin  target_runs  target_overs
count  1.095000e+03    1076.000000  1092.000000   1092.000000
mean   9.048283e+05      17.259294   165.684066     19.759341
std    3.677402e+05      21.787444    33.427048      1.581108
```

```
min    3.359820e+05       1.000000    43.000000      5.000000
25%    5.483315e+05       6.000000   146.000000     20.000000
50%    9.809610e+05       8.000000   166.000000     20.000000
75%    1.254062e+06      20.000000   187.000000     20.000000
max    1.426312e+06     146.000000   288.000000     20.000000
```

7. Find out outliers for the selective columns for lower range outliers will be lower than mu - 2sigma, similarly for upper range outliers will be greater than mu+2sigma.

[118]:
```
# Calculate the mean and standard deviation for the result_margin column
mu = df['result_margin'].mean()
sigma = df['result_margin'].std()

# Calculate the lower and upper bounds for outliers
lower_bound = mu - 2 * sigma
upper_bound = mu + 2 * sigma

# Find the outliers
outliers = df[(df['result_margin'] < lower_bound) | (df['result_margin'] >␣
 ↪upper_bound)]
print(outliers)
```

```
           id   season           city        date  match_type player_of_match  \
0      335982  2007/08      Bangalore  2008-04-18      League     BB McCullum
9      335991  2007/08     Chandigarh  2008-04-25      League   KC Sangakkara
39     336023  2007/08         Jaipur  2008-05-17      League        GC Smith
55     336038  2007/08         Mumbai  2008-05-30  Semi Final       SR Watson
59     392182     2009      Cape Town  2009-04-18      League        R Dravid
…         …       …              …           …           …              …
1030  1422125     2024        Chennai  2024-03-26      League          S Dube
1039  1422134     2024  Visakhapatnam  2024-04-03      League       SP Narine
1058  1426273     2024          Delhi  2024-04-20      League         TM Head
1069  1426284     2024        Chennai  2024-04-28      League      RD Gaikwad
1077  1426292     2024        Lucknow  2024-05-05      League       SP Narine

                                                 venue  \
0                                M Chinnaswamy Stadium
9            Punjab Cricket Association Stadium, Mohali
39                               Sawai Mansingh Stadium
55                                    Wankhede Stadium
59                                           Newlands
…                                                …
1030           MA Chidambaram Stadium, Chepauk, Chennai
1039  Dr. Y.S. Rajasekhara Reddy ACA-VDCA Cricket St…
1058                          Arun Jaitley Stadium, Delhi
1069           MA Chidambaram Stadium, Chepauk, Chennai
1077  Bharat Ratna Shri Atal Bihari Vajpayee Ekana C…
```

```
                           team1                        team2  \
0     Royal Challengers Bangalore        Kolkata Knight Riders
9                  Kings XI Punjab              Mumbai Indians
39               Rajasthan Royals  Royal Challengers Bangalore
55               Delhi Daredevils            Rajasthan Royals
59    Royal Challengers Bangalore            Rajasthan Royals
…                             …                           …
1030           Chennai Super Kings              Gujarat Titans
1039         Kolkata Knight Riders              Delhi Capitals
1058           Sunrisers Hyderabad              Delhi Capitals
1069           Chennai Super Kings         Sunrisers Hyderabad
1077         Kolkata Knight Riders        Lucknow Super Giants

                     toss_winner toss_decision                       winner  \
0     Royal Challengers Bangalore         field       Kolkata Knight Riders
9                   Mumbai Indians         field             Kings XI Punjab
39    Royal Challengers Bangalore         field            Rajasthan Royals
55                Delhi Daredevils         field            Rajasthan Royals
59    Royal Challengers Bangalore           bat  Royal Challengers Bangalore
…                             …           …                         …
1030              Gujarat Titans         field          Chennai Super Kings
1039         Kolkata Knight Riders           bat       Kolkata Knight Riders
1058                Delhi Capitals         field          Sunrisers Hyderabad
1069           Sunrisers Hyderabad         field          Chennai Super Kings
1077          Lucknow Super Giants         field       Kolkata Knight Riders

     result  result_margin  target_runs  target_overs super_over method  \
0      runs          140.0        223.0          20.0          N    NaN
9      runs           66.0        183.0          20.0          N    NaN
39     runs           65.0        198.0          20.0          N    NaN
55     runs          105.0        193.0          20.0          N    NaN
59     runs           75.0        134.0          20.0          N    NaN
…     …             …          …           …        …    …
1030   runs           63.0        207.0          20.0          N    NaN
1039   runs          106.0        273.0          20.0          N    NaN
1058   runs           67.0        267.0          20.0          N    NaN
1069   runs           78.0        213.0          20.0          N    NaN
1077   runs           98.0        236.0          20.0          N    NaN

                  umpire1              umpire2
0               Asad Rauf           RE Koertzen
9               Aleem Dar             AM Saheba
39              BF Bowden            SL Shastri
55              BF Bowden           RE Koertzen
59            BR Doctrove             RB Tiffin
…                       …                     …
1030             AG Wharf          Tapan Sharma
1039              A Totre             UV Gandhe
```

```
1058         J Madanagopal          Navdeep Singh
1069               R Pandit  MV Saidharshan Kumar
1077  MV Saidharshan Kumar               YC Barde

[65 rows x 20 columns]
```

from file: PMRP_6

OM CHOKSI 23AIML010 PMRP ASSIGNMENT 6 WITH CONCLUSION

CLASSWORK

QUESTIONS:- ->General Population and Gender Distribution

What is the total population in each county, and how does it vary by state? What is the gender distribution (Men vs. Women) across different counties? What is the average population size for census tracts in each state? How does the population of each race (White, Black, Hispanic, etc.) differ across states? What is the proportion of the male population compared to the female population in each census tract?

->Ethnicity and Race

What is the distribution of Hispanic population across various counties and states? How do different racial groups (White, Black, Native, etc.) vary in terms of percentage of total population in different counties? Which states have the highest percentage of Black or Hispanic populations?

->Employment and Work Type

What is the employment rate (Employed vs. Unemployed) for each census tract? How does the rate of self-employed individuals compare to those working in private/public sectors across different states? What percentage of the population works from home, and how does it vary by county and state? How does the unemployment rate vary across different states and counties? What is the distribution of employed individuals working in private vs. public sectors?

->Commuting and Transportation

What is the average commuting time across counties and states, and how does it differ for employed individuals? What modes of transportation are most commonly used for commuting in different states (e.g., car, public transportation, walking)? How does the percentage of people commuting via walking or public transportation vary between urban and rural areas?

->Income and Housing

What is the average income (or median household income) in each state and county? How does the distribution of housing type (e.g., owner-occupied vs. renter-occupied) vary across different counties? How does the cost of living compare across different states based on average income and housing costs?

-> Social Characteristics

What is the relationship between education levels (e.g., percentage with a high school diploma, bachelor's degree) and employment types across different states?

3 General Population and Gender Distribution

What is the total population in each county, and how does it vary by state?

What is the gender distribution (Men vs. Women) across different counties?

What is the average population size for census tracts in each state?

How does the population of each race (White, Black, Hispanic, etc.) differ across states?

What is the proportion of the male population compared to the female population in each census tract?

[65]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

df=pd.read_csv('acs2017_census_tract_data.csv')
df,df.head(),df.tail(),df.describe(),df.info(),df.columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74001 entries, 0 to 74000
Data columns (total 37 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   TractId           74001 non-null  int64
 1   State             74001 non-null  object
 2   County            74001 non-null  object
 3   TotalPop          74001 non-null  int64
 4   Men               74001 non-null  int64
 5   Women             74001 non-null  int64
 6   Hispanic          73305 non-null  float64
 7   White             73305 non-null  float64
 8   Black             73305 non-null  float64
 9   Native            73305 non-null  float64
 10  Asian             73305 non-null  float64
 11  Pacific           73305 non-null  float64
 12  VotingAgeCitizen  74001 non-null  int64
 13  Income            72885 non-null  float64
 14  IncomeErr         72885 non-null  float64
 15  IncomePerCap      73256 non-null  float64
 16  IncomePerCapErr   73256 non-null  float64
 17  Poverty           73159 non-null  float64
 18  ChildPoverty      72891 non-null  float64
 19  Professional      73190 non-null  float64
 20  Service           73190 non-null  float64
 21  Office            73190 non-null  float64
 22  Construction      73190 non-null  float64
 23  Production        73190 non-null  float64
 24  Drive             73200 non-null  float64
```

```
 25  Carpool           73200 non-null  float64
 26  Transit           73200 non-null  float64
 27  Walk              73200 non-null  float64
 28  OtherTransp       73200 non-null  float64
 29  WorkAtHome        73200 non-null  float64
 30  MeanCommute       73055 non-null  float64
 31  Employed          74001 non-null  int64
 32  PrivateWork       73190 non-null  float64
 33  PublicWork        73190 non-null  float64
 34  SelfEmployed      73190 non-null  float64
 35  FamilyWork        73190 non-null  float64
 36  Unemployment      73191 non-null  float64
dtypes: float64(29), int64(6), object(2)
memory usage: 20.9+ MB
```

[65]: (

	TractId	State	County	TotalPop	Men	Women	\
0	1001020100	Alabama	Autauga County	1845	899	946	
1	1001020200	Alabama	Autauga County	2172	1167	1005	
2	1001020300	Alabama	Autauga County	3385	1533	1852	
3	1001020400	Alabama	Autauga County	4267	2001	2266	
4	1001020500	Alabama	Autauga County	9965	5054	4911	
…	…	…	…	…	…	…	
73996	72153750501	Puerto Rico	Yauco Municipio	6011	3035	2976	
73997	72153750502	Puerto Rico	Yauco Municipio	2342	959	1383	
73998	72153750503	Puerto Rico	Yauco Municipio	2218	1001	1217	
73999	72153750601	Puerto Rico	Yauco Municipio	4380	1964	2416	
74000	72153750602	Puerto Rico	Yauco Municipio	3001	1343	1658	

	Hispanic	White	Black	Native	…	Walk	OtherTransp	WorkAtHome	\
0	2.4	86.3	5.2	0.0	…	0.5	0.0	2.1	
1	1.1	41.6	54.5	0.0	…	0.0	0.5	0.0	
2	8.0	61.4	26.5	0.6	…	1.0	0.8	1.5	
3	9.6	80.3	7.1	0.5	…	1.5	2.9	2.1	
4	0.9	77.5	16.4	0.0	…	0.8	0.3	0.7	
…	…	…	…	…	…	…	…	…	
73996	99.7	0.3	0.0	0.0	…	0.5	0.0	3.6	
73997	99.1	0.9	0.0	0.0	…	0.0	0.0	1.3	
73998	99.5	0.2	0.0	0.0	…	3.4	0.0	3.4	
73999	100.0	0.0	0.0	0.0	…	0.0	0.0	0.0	
74000	99.2	0.8	0.0	0.0	…	4.9	0.0	8.9	

	MeanCommute	Employed	PrivateWork	PublicWork	SelfEmployed	\
0	24.5	881	74.2	21.2	4.5	
1	22.2	852	75.9	15.0	9.0	
2	23.1	1482	73.3	21.1	4.8	
3	25.9	1849	75.8	19.7	4.5	
4	21.0	4787	71.4	24.1	4.5	

```
…            …       …           …          …           …
73996         26.9      1576         59.2        33.8           7.0
73997         25.3       666         58.4        35.4           6.2
73998         23.5       560         57.5        34.5           8.0
73999         24.1      1062         67.7        30.4           1.9
74000         21.6       759         75.9        19.1           5.0

       FamilyWork  Unemployment
0             0.0           4.6
1             0.0           3.4
2             0.7           4.7
3             0.0           6.1
4             0.0           2.3
…             …             …
73996         0.0          20.8
73997         0.0          26.3
73998         0.0          23.0
73999         0.0          29.5
74000         0.0          17.9

[74001 rows x 37 columns],
      TractId    State          County  TotalPop   Men  Women  Hispanic  \
0  1001020100  Alabama  Autauga County      1845   899    946       2.4
1  1001020200  Alabama  Autauga County      2172  1167   1005       1.1
2  1001020300  Alabama  Autauga County      3385  1533   1852       8.0
3  1001020400  Alabama  Autauga County      4267  2001   2266       9.6
4  1001020500  Alabama  Autauga County      9965  5054   4911       0.9

   White  Black  Native  …  Walk  OtherTransp  WorkAtHome  MeanCommute  \
0   86.3    5.2     0.0  …   0.5          0.0         2.1         24.5
1   41.6   54.5     0.0  …   0.0          0.5         0.0         22.2
2   61.4   26.5     0.6  …   1.0          0.8         1.5         23.1
3   80.3    7.1     0.5  …   1.5          2.9         2.1         25.9
4   77.5   16.4     0.0  …   0.8          0.3         0.7         21.0

   Employed  PrivateWork  PublicWork  SelfEmployed  FamilyWork  Unemployment
0       881         74.2        21.2           4.5         0.0           4.6
1       852         75.9        15.0           9.0         0.0           3.4
2      1482         73.3        21.1           4.8         0.7           4.7
3      1849         75.8        19.7           4.5         0.0           6.1
4      4787         71.4        24.1           4.5         0.0           2.3

[5 rows x 37 columns],
           TractId        State           County  TotalPop   Men  Women  \
73996  72153750501  Puerto Rico  Yauco Municipio      6011  3035   2976
73997  72153750502  Puerto Rico  Yauco Municipio      2342   959   1383
73998  72153750503  Puerto Rico  Yauco Municipio      2218  1001   1217
```

```
73999  72153750601  Puerto Rico  Yauco Municipio      4380  1964   2416
74000  72153750602  Puerto Rico  Yauco Municipio      3001  1343   1658

       Hispanic  White  Black  Native  …  Walk  OtherTransp  WorkAtHome  \
73996      99.7    0.3    0.0     0.0  …   0.5          0.0         3.6
73997      99.1    0.9    0.0     0.0  …   0.0          0.0         1.3
73998      99.5    0.2    0.0     0.0  …   3.4          0.0         3.4
73999     100.0    0.0    0.0     0.0  …   0.0          0.0         0.0
74000      99.2    0.8    0.0     0.0  …   4.9          0.0         8.9

       MeanCommute  Employed  PrivateWork  PublicWork  SelfEmployed  \
73996         26.9      1576         59.2        33.8           7.0
73997         25.3       666         58.4        35.4           6.2
73998         23.5       560         57.5        34.5           8.0
73999         24.1      1062         67.7        30.4           1.9
74000         21.6       759         75.9        19.1           5.0

       FamilyWork  Unemployment
73996         0.0          20.8
73997         0.0          26.3
73998         0.0          23.0
73999         0.0          29.5
74000         0.0          17.9

[5 rows x 37 columns],
             TractId      TotalPop           Men         Women      Hispanic  \
count   7.400100e+04  74001.000000  74001.000000  74001.000000  73305.000000
mean    2.839113e+10   4384.716017   2157.710707   2227.005311     17.265444
std     1.647593e+10   2228.936729   1120.560504   1146.240218     23.073811
min     1.001020e+09      0.000000      0.000000      0.000000      0.000000
25%     1.303901e+10   2903.000000   1416.000000   1465.000000      2.600000
50%     2.804700e+10   4105.000000   2007.000000   2082.000000      7.400000
75%     4.200341e+10   5506.000000   2707.000000   2803.000000     21.100000
max     7.215375e+10  65528.000000  32266.000000  33262.000000    100.000000

              White        Black        Native         Asian       Pacific  \
count  73305.000000  73305.00000  73305.000000  73305.000000  73305.000000
mean      61.309043     13.28910      0.734047      4.753691      0.147341
std       30.634461     21.60118      4.554247      8.999888      1.029250
min        0.000000      0.00000      0.000000      0.000000      0.000000
25%       38.000000      0.80000      0.000000      0.200000      0.000000
50%       70.400000      3.80000      0.000000      1.500000      0.000000
75%       87.700000     14.60000      0.400000      5.000000      0.000000
max      100.000000    100.00000    100.000000    100.000000     71.900000

       …          Walk   OtherTransp    WorkAtHome   MeanCommute  \
count  …  73200.000000  73200.000000  73200.000000  73055.000000
```

```
mean   …      3.042825      1.894605      4.661466     26.056594
std    …      5.805753      2.549374      4.014940      7.124524
min    …      0.000000      0.000000      0.000000      1.000000
25%    …      0.400000      0.400000      2.000000     21.100000
50%    …      1.400000      1.200000      3.800000     25.400000
75%    …      3.300000      2.500000      6.300000     30.300000
max    …    100.000000    100.000000    100.000000     73.900000

           Employed   PrivateWork    PublicWork  SelfEmployed    FamilyWork  \
count  74001.000000  73190.000000  73190.000000  73190.000000  73190.000000
mean    2049.152052     79.494222     14.163342      6.171484      0.171164
std     1138.865457      8.126383      7.328680      3.932364      0.456580
min        0.000000      0.000000      0.000000      0.000000      0.000000
25%     1276.000000     75.200000      9.300000      3.500000      0.000000
50%     1895.000000     80.600000     13.000000      5.500000      0.000000
75%     2635.000000     85.000000     17.600000      8.000000      0.000000
max    28945.000000    100.000000    100.000000    100.000000     22.300000

       Unemployment
count  73191.000000
mean       7.246738
std        5.227624
min        0.000000
25%        3.900000
50%        6.000000
75%        9.000000
max      100.000000

[8 rows x 35 columns],
None,
Index(['TractId', 'State', 'County', 'TotalPop', 'Men', 'Women', 'Hispanic',
       'White', 'Black', 'Native', 'Asian', 'Pacific', 'VotingAgeCitizen',
       'Income', 'IncomeErr', 'IncomePerCap', 'IncomePerCapErr', 'Poverty',
       'ChildPoverty', 'Professional', 'Service', 'Office', 'Construction',
       'Production', 'Drive', 'Carpool', 'Transit', 'Walk', 'OtherTransp',
       'WorkAtHome', 'MeanCommute', 'Employed', 'PrivateWork', 'PublicWork',
       'SelfEmployed', 'FamilyWork', 'Unemployment'],
      dtype='object'))
```

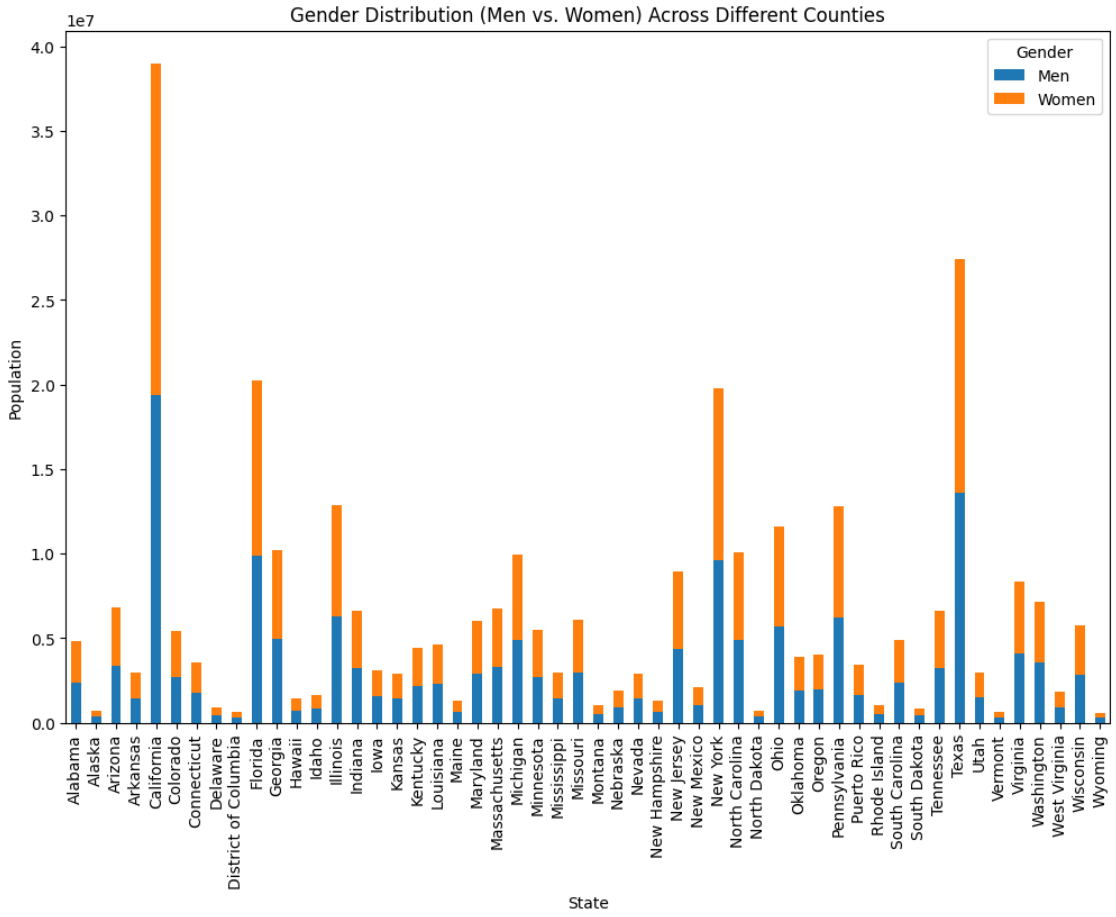
[66]:
```
#What is the total population in each county, and how does it vary by state?
total_population_by_county = df.groupby(['State', 'County'])['TotalPop'].sum().
 ↪reset_index()
print(total_population_by_county)

total_population_by_state = df.groupby('State')['TotalPop'].sum().reset_index()
print(total_population_by_state)
```

```
       State             County  TotalPop
```

```
0     Alabama     Autauga County     55036
1     Alabama     Baldwin County    203360
2     Alabama     Barbour County     26201
3     Alabama        Bibb County     22580
4     Alabama      Blount County     57667
…       …                …       …
3215  Wyoming  Sweetwater County     44527
3216  Wyoming       Teton County     22923
3217  Wyoming       Uinta County     20758
3218  Wyoming    Washakie County      8253
3219  Wyoming      Weston County      7117

[3220 rows x 3 columns]
                   State  TotalPop
0                Alabama   4850771
1                 Alaska    738565
2                Arizona   6809946
3               Arkansas   2977944
4             California  38982847
5               Colorado   5436519
6            Connecticut   3594478
7               Delaware    943732
8   District of Columbia    672391
9                Florida  20278447
10               Georgia  10201635
11                Hawaii   1421658
12                 Idaho   1657375
13              Illinois  12854526
14               Indiana   6614418
15                  Iowa   3118102
16                Kansas   2903820
17              Kentucky   4424376
18             Louisiana   4663461
19                 Maine   1330158
20              Maryland   5996079
21         Massachusetts   6789319
22              Michigan   9925568
23             Minnesota   5490726
24           Mississippi   2986220
25              Missouri   6075300
26               Montana   1029862
27              Nebraska   1893921
28                Nevada   2887725
29         New Hampshire   1331848
30            New Jersey   8960161
31            New Mexico   2084828
32              New York  19798228
33        North Carolina  10052564
```

```
34          North Dakota    745475
35                  Ohio  11609756
36              Oklahoma   3896251
37                Oregon   4025127
38          Pennsylvania  12790505
39           Puerto Rico   3468963
40          Rhode Island   1056138
41        South Carolina   4893444
42          South Dakota    855444
43             Tennessee   6597381
44                 Texas  27419612
45                  Utah   2993941
46               Vermont    624636
47              Virginia   8365952
48            Washington   7169967
49         West Virginia   1836843
50             Wisconsin   5763217
51               Wyoming    583200
```

[67]:
```
#What is the gender distribution (Men vs. Women) across different counties?

gender_distribution_by_county = df.groupby(['State', 'County'])[['Men',␣
 ↪'Women']].sum().reset_index()


fig, ax = plt.subplots(figsize=(12, 8))
gender_distribution_by_county.groupby('State')[['Men', 'Women']].sum().
 ↪plot(kind='bar', stacked=True, ax=ax)
ax.set_title('Gender Distribution (Men vs. Women) Across Different Counties')
ax.set_xlabel('State')
ax.set_ylabel('Population')
plt.legend(title='Gender')
plt.show()
```



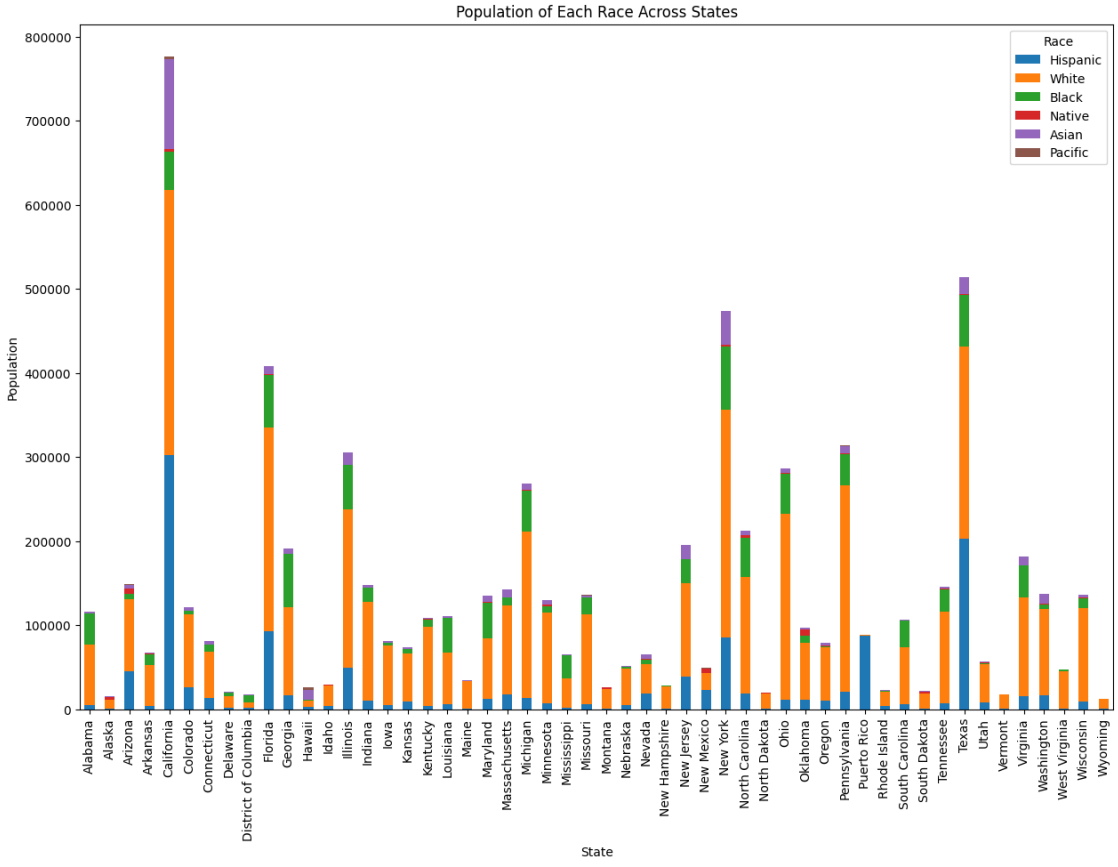
[68]:
```
#What is the average population size for census tracts in each state?

average_population_by_state = df.groupby('State')['TotalPop'].mean().
 ↪reset_index()
# print(average_population_by_state)
average_population_by_state.head()
```

[68]:

	State	TotalPop
0	Alabama	4107.342083
1	Alaska	4422.544910
2	Arizona	4462.612058
3	Arkansas	4341.026239
4	California	4838.382400

[69]:
```
#How does the population of each race (White, Black, Hispanic, etc.) differ␣
 ↪across states?
```

```
race_population_by_state = df.groupby('State')[['Hispanic', 'White', 'Black',␣
 ↪'Native', 'Asian', 'Pacific']].sum().reset_index()
# print(race_population_by_state)
race_population_by_state

# Plot the population of each race across states
race_population_by_state.set_index('State').plot(kind='bar', stacked=True,␣
 ↪figsize=(15, 10))
plt.title('Population of Each Race Across States')
plt.xlabel('State')
plt.ylabel('Population')
plt.legend(title='Race')
plt.show()
```



[70]:
```
#What is the proportion of the male population compared to the female␣
 ↪population in each census tract?

df['MaleToFemaleRatio'] = df['Men'] / df['Women']
```

```
# Display the first few rows to verify the calculation
df[['State', 'County', 'TractId', 'Men', 'Women', 'MaleToFemaleRatio']].head()
```

[70]:

	State	County	TractId	Men	Women	MaleToFemaleRatio
0	Alabama	Autauga County	1001020100	899	946	0.950317
1	Alabama	Autauga County	1001020200	1167	1005	1.161194
2	Alabama	Autauga County	1001020300	1533	1852	0.827754
3	Alabama	Autauga County	1001020400	2001	2266	0.883054
4	Alabama	Autauga County	1001020500	5054	4911	1.029118

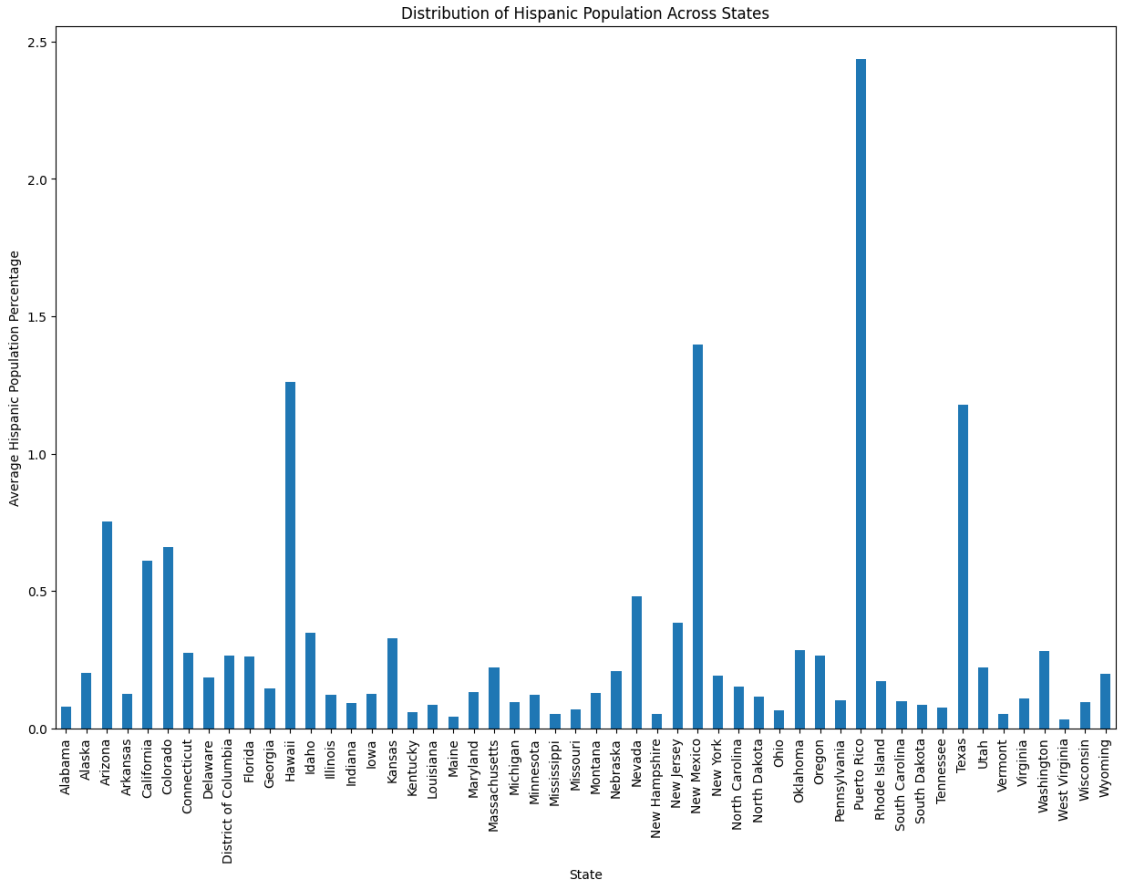
4 Ethnicity and Race

What is the distribution of Hispanic population across various counties and states?

How do different racial groups (White, Black, Native, etc.) vary in terms of percentage of total population in different counties?

Which states have the highest percentage of Black or Hispanic populations?

[71]:
```
#What is the distribution of Hispanic population across various counties and␣
 ↪states?

df['Hispanic_Percentage'] = (df['Hispanic'] / df['TotalPop']) * 100


hispanic_by_county = df.groupby(['State', 'County'])[['Hispanic', 'TotalPop']].
 ↪sum().reset_index()
hispanic_by_county['Hispanic_Percentage'] = (hispanic_by_county['Hispanic'] /␣
 ↪hispanic_by_county['TotalPop']) * 100


hispanic_state_data = hispanic_by_county.
 ↪groupby('State')['Hispanic_Percentage'].mean()

hispanic_state_data.plot(kind='bar', figsize=(15, 10))
plt.title('Distribution of Hispanic Population Across States')
plt.xlabel('State')
plt.ylabel('Average Hispanic Population Percentage')
plt.show()
```



[72]:
```
#How do different racial groups (White, Black, Native, etc.) vary in terms of␣
 ↪percentage of total population in different counties?
# Calculate the percentage of each racial group in each county
df['White_Percentage'] = (df['White'] / df['TotalPop']) * 100
df['Black_Percentage'] = (df['Black'] / df['TotalPop']) * 100
df['Native_Percentage'] = (df['Native'] / df['TotalPop']) * 100
df['Asian_Percentage'] = (df['Asian'] / df['TotalPop']) * 100
df['Pacific_Percentage'] = (df['Pacific'] / df['TotalPop']) * 100

racial_percentage_by_county = df.groupby(['State',␣
 ↪'County'])[['White_Percentage', 'Black_Percentage', 'Native_Percentage',␣
 ↪'Asian_Percentage', 'Pacific_Percentage']].mean().reset_index()

racial_percentage_by_county
```

[72]:

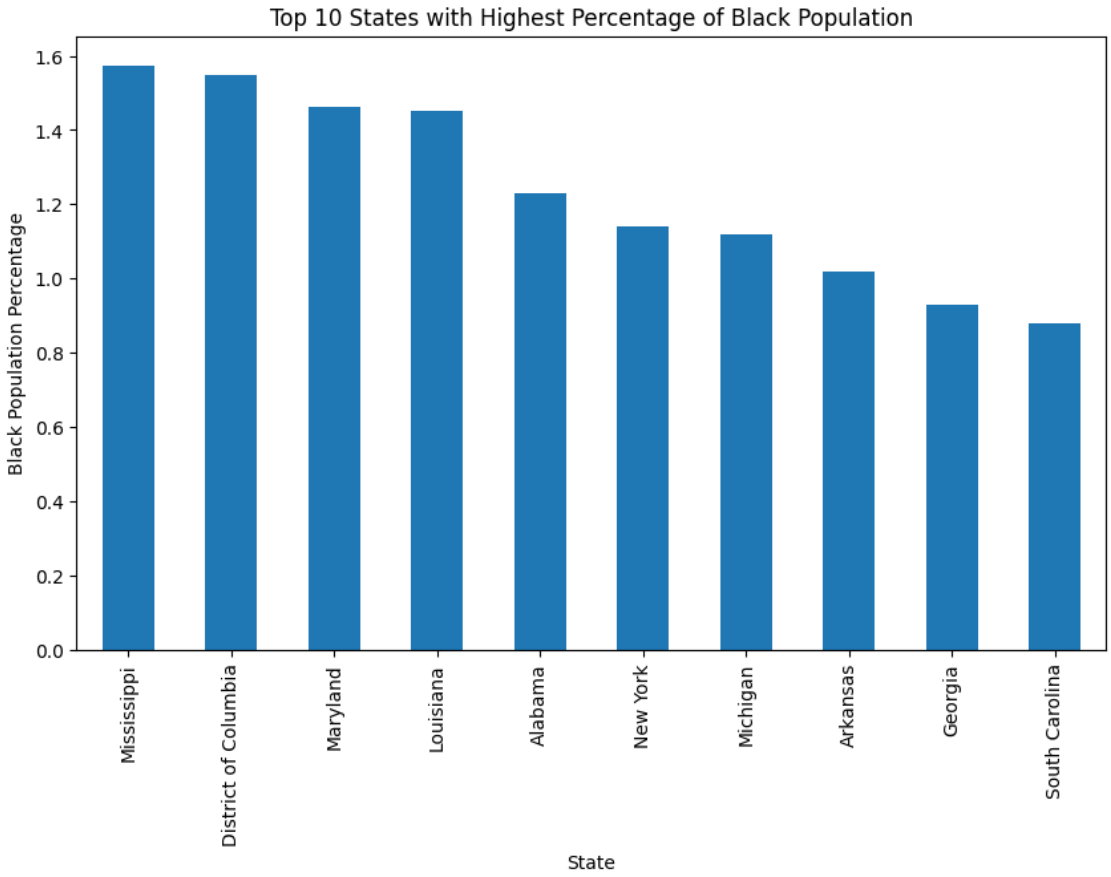
	State	County	White_Percentage	Black_Percentage	\
0	Alabama	Autauga County	2.033985	0.677915	
1	Alabama	Baldwin County	1.837739	0.193102	
2	Alabama	Barbour County	1.749863	1.845252	

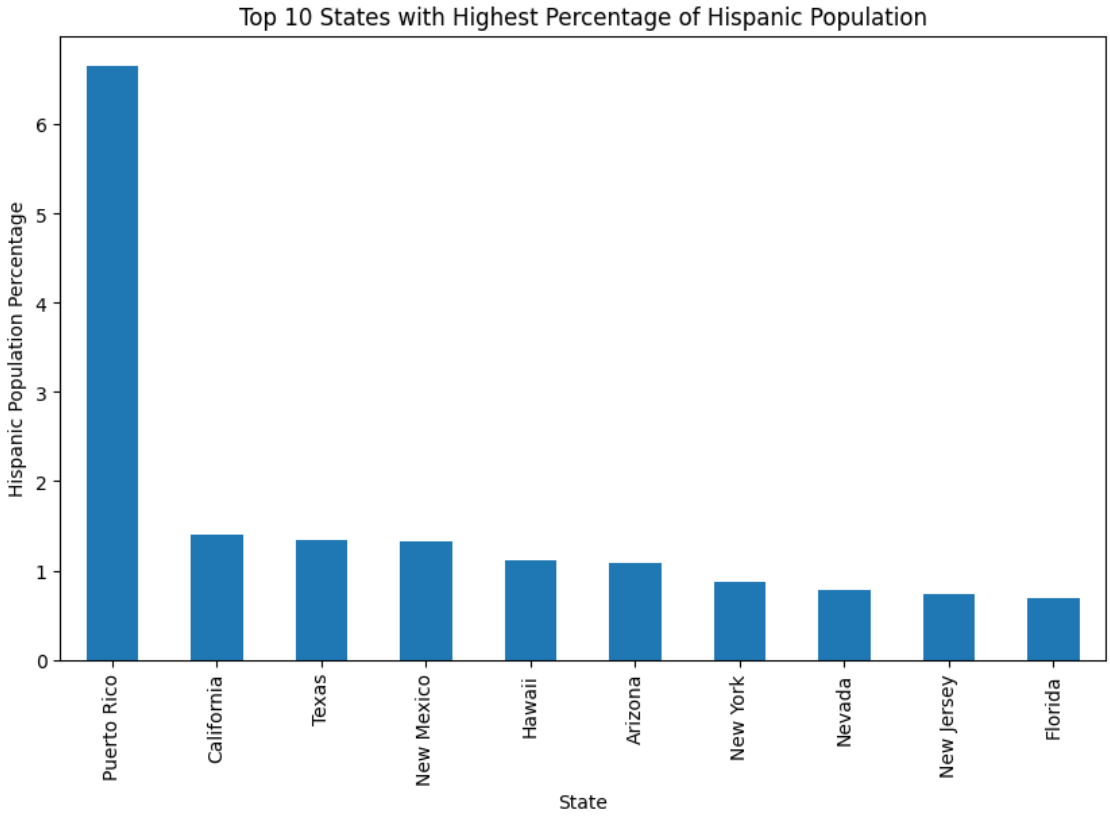
```
3     Alabama         Bibb County          1.787693          0.285017
4     Alabama       Blount County          1.496798          0.018945
…       …                 …                …                 …
3215  Wyoming   Sweetwater County          2.699432          0.017665
3216  Wyoming        Teton County          1.701364          0.008142
3217  Wyoming        Uinta County          1.283898          0.001929
3218  Wyoming     Washakie County          2.996604          0.008328
3219  Wyoming       Weston County          2.578226          0.015020

      Native_Percentage  Asian_Percentage  Pacific_Percentage
0              0.011672          0.015104            0.000985
1              0.017746          0.007720            0.000000
2              0.004166          0.014984            0.000000
3              0.006752          0.000000            0.000000
4              0.006233          0.002023            0.000000
…                 …                 …                 …
3215           0.018119          0.021793            0.009755
3216           0.005138          0.037086            0.000000
3217           0.011570          0.001302            0.000000
3218           0.013837          0.005118            0.000000
3219           0.002822          0.116336            0.000000

[3220 rows x 7 columns]
```

[73]:
```
#Which states have the highest percentage of Black or Hispanic populations?

# Calculate the average percentage of Black and Hispanic populations in each␣
 ↪state
black_hispanic_percentage_by_state = df.groupby('State')[['Black_Percentage',␣
 ↪'Hispanic_Percentage']].mean().reset_index()

# the highest percentage of Black population
highest_black_percentage_states = black_hispanic_percentage_by_state.
 ↪sort_values(by='Black_Percentage', ascending=False).head(10)
print("States with the highest percentage of Black population:")
print(highest_black_percentage_states)

# the highest percentage of Hispanic population
highest_hispanic_percentage_states = black_hispanic_percentage_by_state.
 ↪sort_values(by='Hispanic_Percentage', ascending=False).head(10)
print("States with the highest percentage of Hispanic population:")
print(highest_hispanic_percentage_states)

# Plot Black population
highest_black_percentage_states.plot(x='State', y='Black_Percentage',␣
 ↪kind='bar', figsize=(10, 6), legend=False)
plt.title('Top 10 States with Highest Percentage of Black Population')
```

```
plt.xlabel('State')
plt.ylabel('Black Population Percentage')
plt.show()

# Plot Hispanic population
highest_hispanic_percentage_states.plot(x='State', y='Hispanic_Percentage',␣
 ↪kind='bar', figsize=(10, 6), legend=False)
plt.title('Top 10 States with Highest Percentage of Hispanic Population')
plt.xlabel('State')
plt.ylabel('Hispanic Population Percentage')
plt.show()
```

```
States with the highest percentage of Black population:
                   State  Black_Percentage  Hispanic_Percentage
24           Mississippi          1.574405             0.113420
8   District of Columbia          1.548670             0.280445
20               Maryland          1.462464             0.235107
18              Louisiana          1.452869             0.165938
0                 Alabama          1.229608             0.114788
32               New York          1.139241             0.865029
22               Michigan          1.118413             0.183960
3                Arkansas          1.020047             0.162479
10                Georgia          0.930425             0.196465
41         South Carolina          0.878945             0.145437
States with the highest percentage of Hispanic population:
          State  Black_Percentage  Hispanic_Percentage
39  Puerto Rico          0.003747             6.660837
4    California          0.249553             1.394659
44        Texas          0.325855             1.339024
31   New Mexico          0.050507             1.325957
11       Hawaii          0.077773             1.115713
2       Arizona          0.103129             1.080010
32     New York          1.139241             0.865029
28       Nevada          0.223116             0.787030
30   New Jersey          0.459594             0.731697
9       Florida          0.757573             0.685121
```


Top 10 States with Highest Percentage of Black Population
Black Population Percentage
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Mississippi
District of Columbia
Maryland
Louisiana
Alabama
New York
Michigan
Arkansas
Georgia
South Carolina
State



5 Employment and Work Type

What is the employment rate (Employed vs. Unemployed) for each census tract?

How does the rate of self-employed individuals compare to those working in private/public sectors across different states?

What percentage of the population works from home, and how does it vary by county and state?

How does the unemployment rate vary across different states and counties?

What is the distribution of employed individuals working in private vs. public sectors?

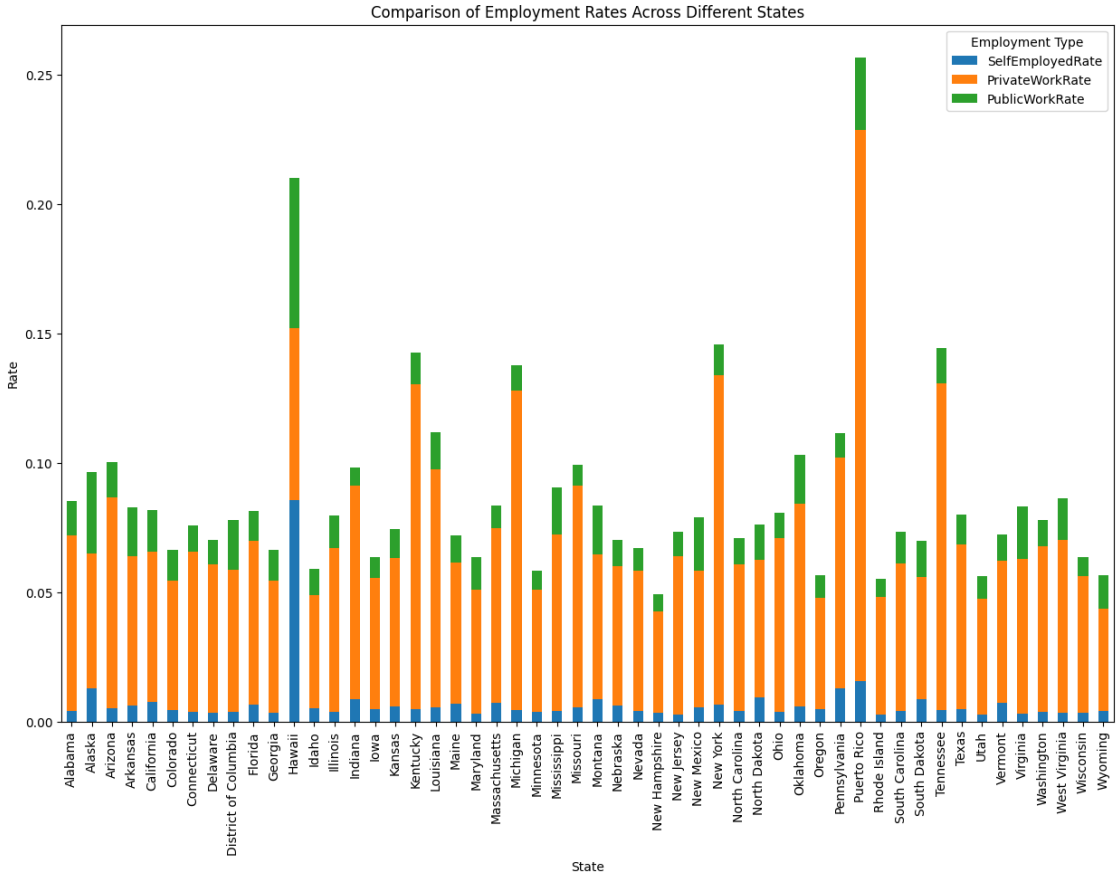
[74]:
```
#What is the employment rate (Employed vs. Unemployed) for each census tract?
df['EmploymentRate'] = df['Employed'] / df['TotalPop']
df['UnemploymentRate'] = df['Unemployment'] / df['TotalPop']
df[['State', 'County', 'TractId', 'Employed', 'Unemployment', 'EmploymentRate',␣
↪'UnemploymentRate']]
```

[74]:

	State	County	TractId	Employed	Unemployment	\
0	Alabama	Autauga County	1001020100	881	4.6	
1	Alabama	Autauga County	1001020200	852	3.4	

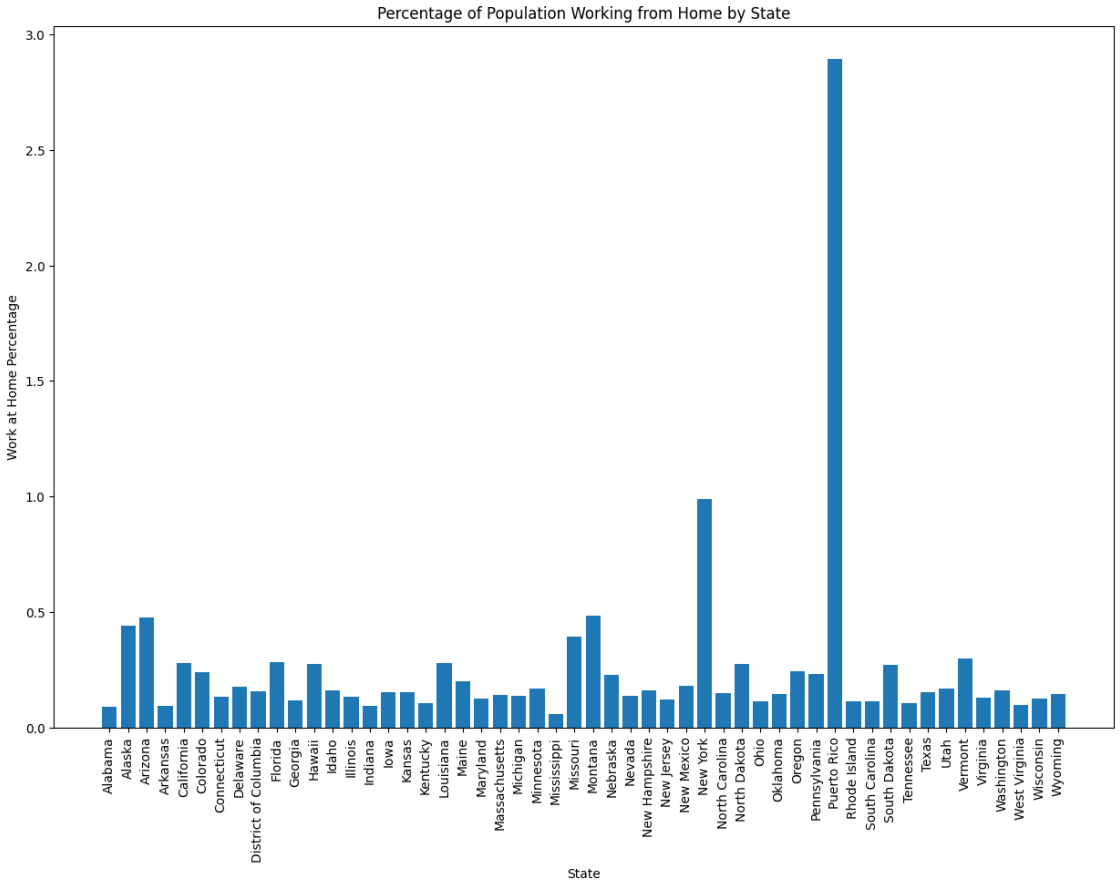
```
2          Alabama   Autauga County   1001020300      1482           4.7
3          Alabama   Autauga County   1001020400      1849           6.1
4          Alabama   Autauga County   1001020500      4787           2.3
…              …              …          …       …          …
73996  Puerto Rico  Yauco Municipio  72153750501      1576          20.8
73997  Puerto Rico  Yauco Municipio  72153750502       666          26.3
73998  Puerto Rico  Yauco Municipio  72153750503       560          23.0
73999  Puerto Rico  Yauco Municipio  72153750601      1062          29.5
74000  Puerto Rico  Yauco Municipio  72153750602       759          17.9

       EmploymentRate  UnemploymentRate
0            0.477507          0.002493
1            0.392265          0.001565
2            0.437814          0.001388
3            0.433326          0.001430
4            0.480381          0.000231
…                 …                 …
73996        0.262186          0.003460
73997        0.284372          0.011230
73998        0.252480          0.010370
73999        0.242466          0.006735
74000        0.252916          0.005965

[74001 rows x 7 columns]
```

[75]:
```
#How does the rate of self-employed individuals compare to those working in␣
 ↪private/public sectors across different states?
df['SelfEmployedRate'] = df['SelfEmployed'] / df['Employed']
df['PrivateWorkRate'] = df['PrivateWork'] / df['Employed']
df['PublicWorkRate'] = df['PublicWork'] / df['Employed']


employment_rates_by_state = df.groupby('State')[['SelfEmployedRate',␣
 ↪'PrivateWorkRate', 'PublicWorkRate']].mean().reset_index()

employment_rates_by_state.head()


employment_rates_by_state.set_index('State').plot(kind='bar', stacked=True,␣
 ↪figsize=(15, 10))
plt.title('Comparison of Employment Rates Across Different States')
plt.xlabel('State')
plt.ylabel('Rate')
plt.legend(title='Employment Type')
plt.show()
```

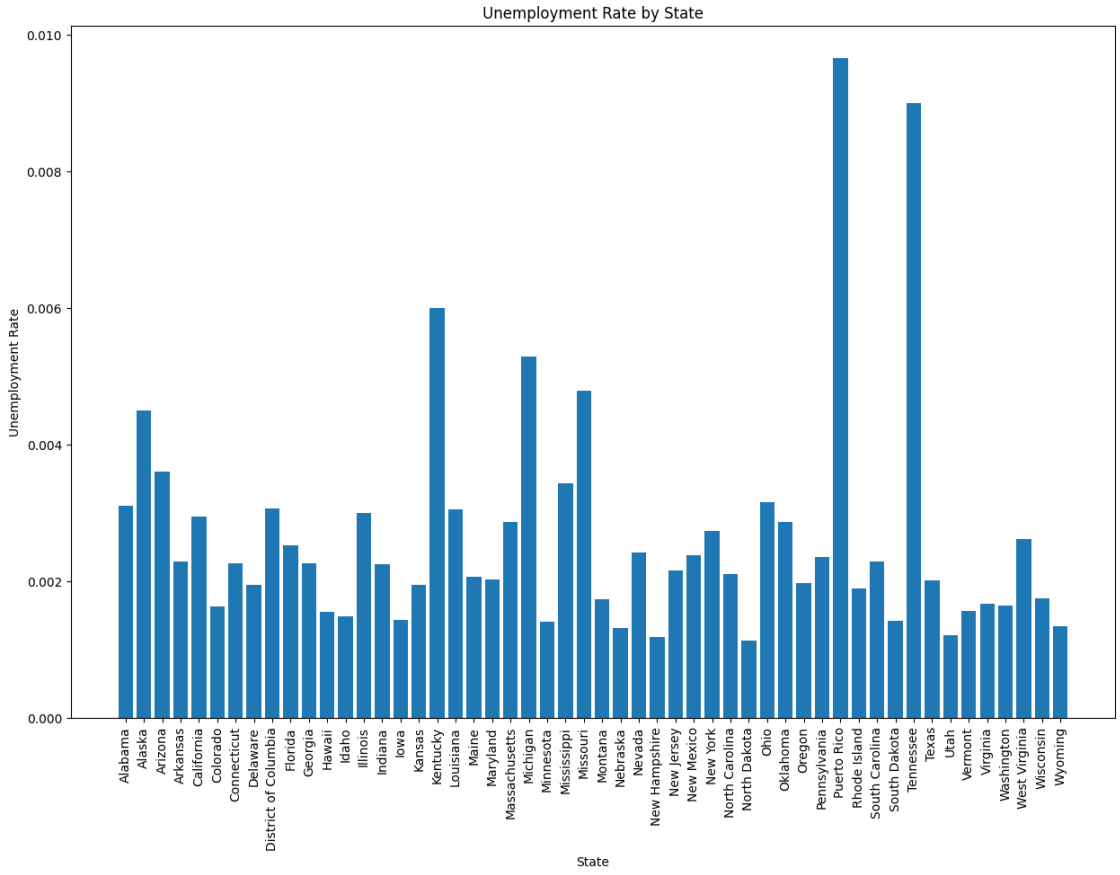


[76]:
```
#What percentage of the population works from home, and how does it vary by␣
 ↪county and state?

df['WorkAtHomePercentage'] = (df['WorkAtHome'] / df['TotalPop']) * 100


work_at_home_by_county = df.groupby(['State',␣
 ↪'County'])['WorkAtHomePercentage'].mean().reset_index()
work_at_home_by_state = df.groupby('State')['WorkAtHomePercentage'].mean().
 ↪reset_index()


plt.figure(figsize=(15, 10))
plt.bar(work_at_home_by_state['State'],␣
 ↪work_at_home_by_state['WorkAtHomePercentage'])
plt.title('Percentage of Population Working from Home by State')
plt.xlabel('State')
plt.ylabel('Work at Home Percentage')
plt.xticks(rotation=90)
```

```
plt.show()
```



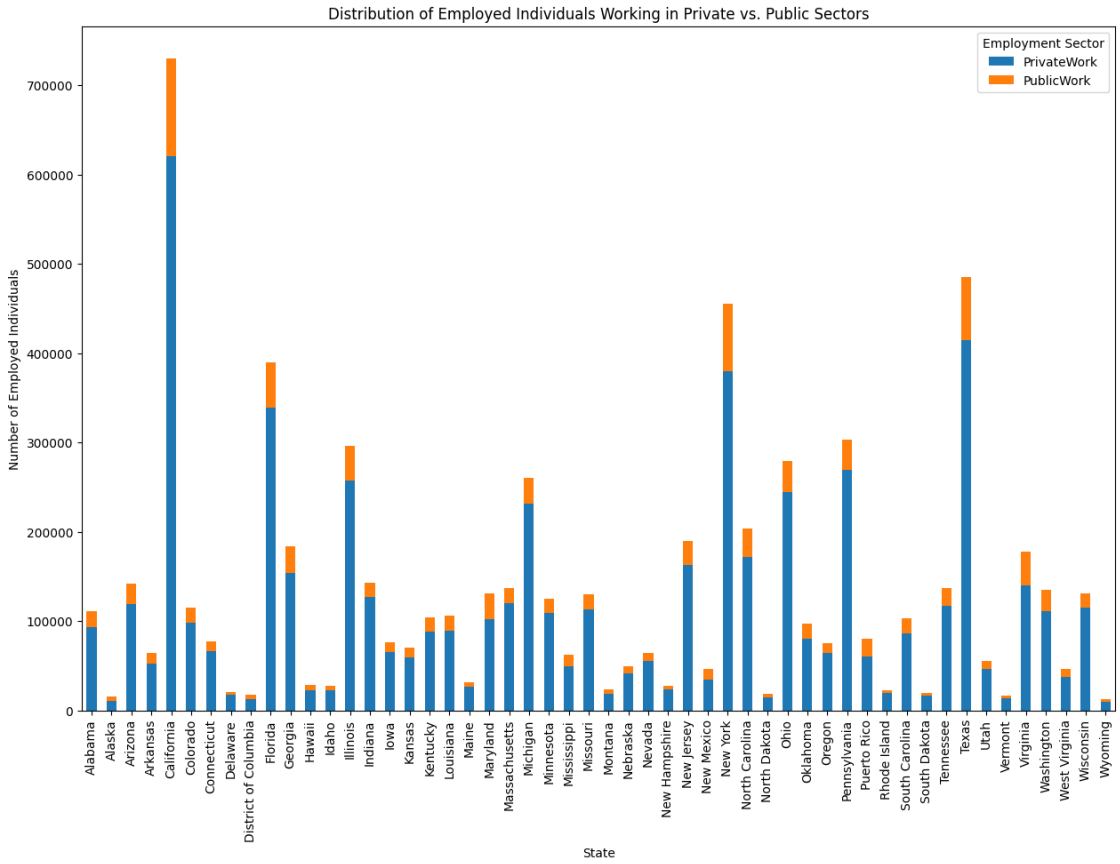
[77]:
```
#How does the unemployment rate vary across different states and counties?

unemployment_rate_by_county = df.groupby(['State',␣
 ↪'County'])['UnemploymentRate'].mean().reset_index()
unemployment_rate_by_state = df.groupby('State')['UnemploymentRate'].mean().
 ↪reset_index()


plt.figure(figsize=(15, 10))
plt.bar(unemployment_rate_by_state['State'],␣
 ↪unemployment_rate_by_state['UnemploymentRate'])
plt.title('Unemployment Rate by State')
plt.xlabel('State')
plt.ylabel('Unemployment Rate')
plt.xticks(rotation=90)
plt.show()
```



[78]:
```
#What is the distribution of employed individuals working in private vs. public␣
 ↪sectors?
employment_distribution_by_sector = df.groupby('State')[['PrivateWork',␣
 ↪'PublicWork']].sum().reset_index()
print(employment_distribution_by_sector)

employment_distribution_by_sector.set_index('State').plot(kind='bar',␣
 ↪stacked=True, figsize=(15, 10))
plt.title('Distribution of Employed Individuals Working in Private vs. Public␣
 ↪Sectors')
plt.xlabel('State')
plt.ylabel('Number of Employed Individuals')
plt.legend(title='Employment Sector')
plt.show()
```

	State	PrivateWork	PublicWork
0	Alabama	93013.8	18161.0
1	Alaska	10828.5	4559.1
2	Arizona	119351.9	22220.5
3	Arkansas	52857.1	11189.5

4	California	620398.1	109397.2
5	Colorado	97844.1	17353.8
6	Connecticut	66745.7	10529.6
7	Delaware	17486.0	3008.6
8	District of Columbia	12648.6	4495.2
9	Florida	339459.8	50039.8
10	Georgia	154053.7	30032.9
11	Hawaii	22185.5	6871.9
12	Idaho	22544.6	4713.8
13	Illinois	257640.6	38630.2
14	Indiana	127561.2	15807.4
15	Iowa	65720.6	10575.5
16	Kansas	59007.9	11671.1
17	Kentucky	87831.4	16510.3
18	Louisiana	88930.0	17008.1
19	Maine	26940.6	4891.5
20	Maryland	101853.0	29729.1
21	Massachusetts	119845.6	17640.1
22	Michigan	231237.5	29144.3
23	Minnesota	109311.0	15853.7
24	Mississippi	49870.3	12135.6
25	Missouri	113203.8	17254.2
26	Montana	18926.0	5122.4
27	Nebraska	41686.2	7305.3
28	Nevada	55822.1	8199.3
29	New Hampshire	23277.0	3879.4
30	New Jersey	162630.5	27280.9
31	New Mexico	34674.1	11583.9
32	New York	379901.5	75897.8
33	North Carolina	172143.5	31173.4
34	North Dakota	14928.3	3435.3
35	Ohio	244297.7	34850.6
36	Oklahoma	80218.8	17329.3
37	Oregon	64121.0	11690.7
38	Pennsylvania	269785.3	33345.0
39	Puerto Rico	60283.9	19938.7
40	Rhode Island	19867.9	2902.2
41	South Carolina	86280.9	16694.5
42	South Dakota	16150.8	3726.8
43	Tennessee	116793.3	20576.8
44	Texas	414609.8	70444.0
45	Utah	46872.6	8617.6
46	Vermont	13926.9	2583.8
47	Virginia	140180.9	37739.8
48	Washington	111172.3	24186.2
49	West Virginia	37205.6	9021.3
50	Wisconsin	114720.4	16816.3
51	Wyoming	9333.9	2849.7



6 Commuting and Transportation

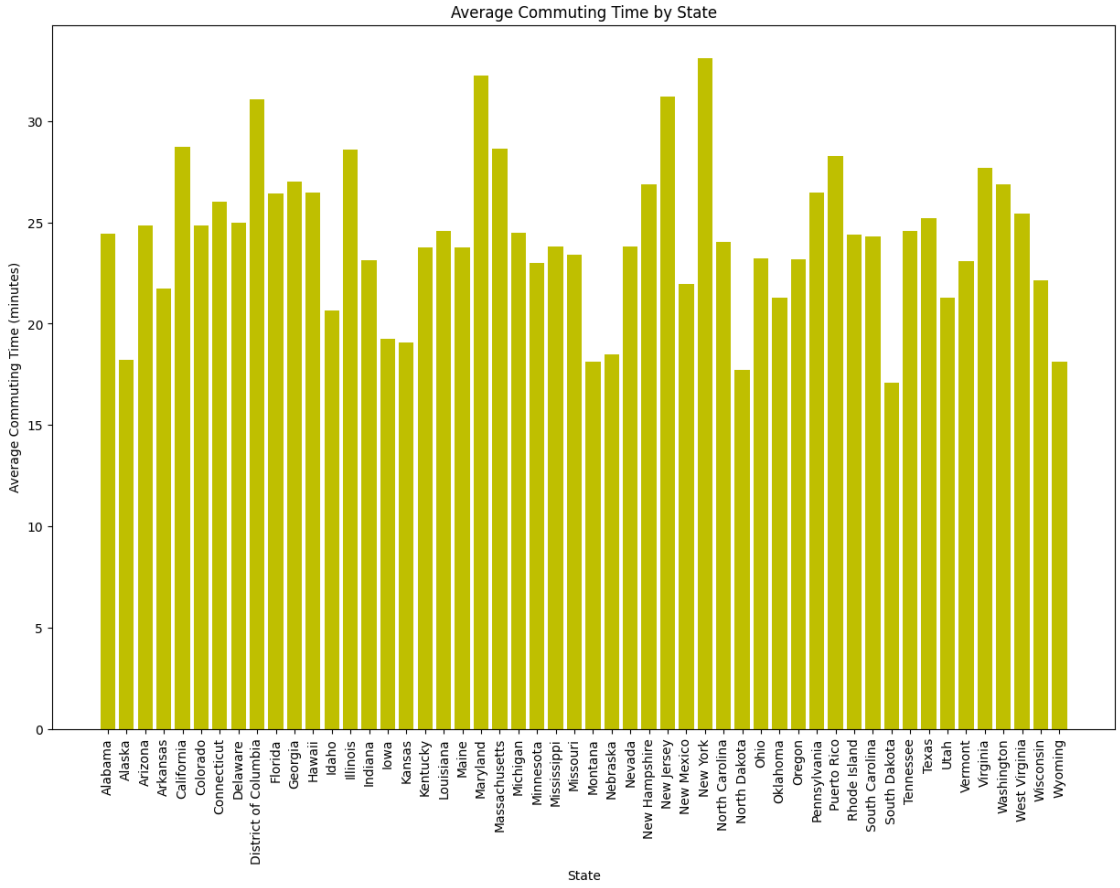
What is the average commuting time across counties and states, and how does it differ for employed individuals?

What modes of transportation are most commonly used for commuting in different states (e.g., car, public transportation, walking)?

How does the percentage of people commuting via walking or public transportation vary between urban and rural areas?

[79]:
```
#What is the average commuting time across counties and states, and how does it␣
 ↪differ for employed individuals?

average_commute_by_county = df.groupby(['State', 'County'])['MeanCommute'].
 ↪mean().reset_index()
average_commute_by_state = df.groupby('State')['MeanCommute'].mean().
 ↪reset_index()

#see how differ
```

```
plt.figure(figsize=(15, 10))
plt.bar(average_commute_by_state['State'],␣
  ↪average_commute_by_state['MeanCommute'],color='y')
plt.title('Average Commuting Time by State')
plt.xlabel('State')
plt.ylabel('Average Commuting Time (minutes)')
plt.xticks(rotation=90)
plt.show()


print(average_commute_by_county)
print(average_commute_by_state)
```



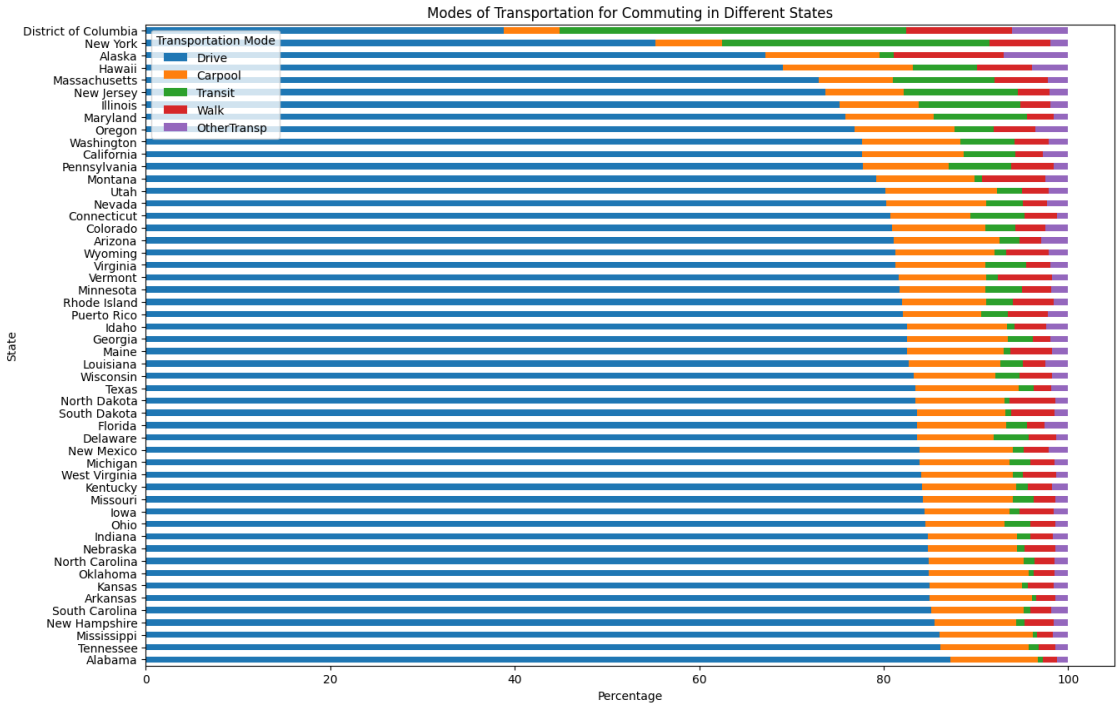
	State	County	MeanCommute
0	Alabama	Autauga County	25.766667
1	Alabama	Baldwin County	27.054839
2	Alabama	Barbour County	22.744444
3	Alabama	Bibb County	31.200000
4	Alabama	Blount County	35.011111

```
…       …                 …          …
3215  Wyoming  Sweetwater County    20.708333
3216  Wyoming       Teton County    14.450000
3217  Wyoming       Uinta County    20.233333
3218  Wyoming    Washakie County    14.533333
3219  Wyoming      Weston County    26.000000

[3220 rows x 3 columns]
```

	State	MeanCommute
0	Alabama	24.458638
1	Alaska	18.209639
2	Arizona	24.833444
3	Arkansas	21.739824
4	California	28.720396
5	Colorado	24.836812
6	Connecticut	26.018909
7	Delaware	24.965421
8	District of Columbia	31.087640
9	Florida	26.436147
10	Georgia	27.015984
11	Hawaii	26.475641
12	Idaho	20.638721
13	Illinois	28.584158
14	Indiana	23.149035
15	Iowa	19.248967
16	Kansas	19.061133
17	Kentucky	23.754430
18	Louisiana	24.555298
19	Maine	23.745584
20	Maryland	32.228974
21	Massachusetts	28.636557
22	Michigan	24.467458
23	Minnesota	22.994670
24	Mississippi	23.791450
25	Missouri	23.416955
26	Montana	18.123792
27	Nebraska	18.464839
28	Nevada	23.829056
29	New Hampshire	26.895890
30	New Jersey	31.191014
31	New Mexico	21.963454
32	New York	33.084997
33	North Carolina	24.020849
34	North Dakota	17.738537
35	Ohio	23.213692
36	Oklahoma	21.298177
37	Oregon	23.183981
38	Pennsylvania	26.470801

```
39          Puerto Rico    28.281087
40         Rhode Island    24.409167
41       South Carolina    24.292173
42         South Dakota    17.077477
43            Tennessee    24.576626
44                Texas    25.205923
45                 Utah    21.286770
46              Vermont    23.095082
47             Virginia    27.695833
48           Washington    26.888989
49        West Virginia    25.428306
50            Wisconsin    22.135396
51              Wyoming    18.145802
```

[80]:
```
#What modes of transportation are most commonly used for commuting in different␣
 ↪states (e.g., car, public transportation, walking)?

#Calcultions
transportation_modes_by_state = df.groupby('State')[['Drive', 'Carpool',␣
 ↪'Transit', 'Walk', 'OtherTransp']].sum().reset_index()

# Normalize the data to get percentages
transportation_modes_by_state[['Drive', 'Carpool', 'Transit', 'Walk',␣
 ↪'OtherTransp']] = transportation_modes_by_state[['Drive', 'Carpool',␣
 ↪'Transit', 'Walk', 'OtherTransp']].
 ↪div(transportation_modes_by_state[['Drive', 'Carpool', 'Transit', 'Walk',␣
 ↪'OtherTransp']].sum(axis=1), axis=0) * 100

# Sort the data by the 'Drive' column in descending order
transportation_modes_by_state = transportation_modes_by_state.
 ↪sort_values(by='Drive', ascending=False)

# HoriZontal Plot of the data
transportation_modes_by_state.set_index('State').plot(kind='barh',␣
 ↪stacked=True, figsize=(15, 10))
plt.title('Modes of Transportation for Commuting in Different States')
plt.xlabel('Percentage')
plt.ylabel('State')
plt.legend(title='Transportation Mode')
plt.show()


print(transportation_modes_by_state)
```



	State	Drive	Carpool	Transit	Walk	\
0	Alabama	87.237800	9.464023	0.586360	1.483553	
43	Tennessee	86.129360	9.594789	1.083404	1.773425	
24	Mississippi	86.086598	10.119360	0.424133	1.671821	
29	New Hampshire	85.520991	8.814976	0.895838	3.211881	
41	South Carolina	85.126841	10.013474	0.761312	2.265327	
3	Arkansas	85.009745	11.042141	0.500057	2.015314	
16	Kansas	84.992725	10.007144	0.596194	2.837533	
36	Oklahoma	84.907412	10.795471	0.583061	2.210284	
33	North Carolina	84.901295	10.254153	1.212449	2.128705	
27	Nebraska	84.767305	9.650356	0.817301	3.337992	
14	Indiana	84.761873	9.662309	1.482523	2.462258	
35	Ohio	84.513781	8.589462	2.775086	2.761280	
15	Iowa	84.454221	9.198036	1.067967	3.695741	
25	Missouri	84.286308	9.674766	2.303347	2.313148	
17	Kentucky	84.152657	10.172520	1.336560	2.625019	
49	West Virginia	84.103448	9.925722	1.045823	3.613250	
22	Michigan	83.925573	9.733940	2.229668	2.609457	
31	New Mexico	83.907252	10.063634	1.180954	2.694058	
7	Delaware	83.632857	8.266754	3.799419	2.954340	
9	Florida	83.585773	9.656995	2.252973	1.895705	
42	South Dakota	83.578985	9.608715	0.613847	4.691166	
34	North Dakota	83.455019	9.613547	0.561868	5.015786	
44	Texas	83.448608	11.148666	1.682065	1.891720	
50	Wisconsin	83.272902	8.829600	2.647031	3.519228	

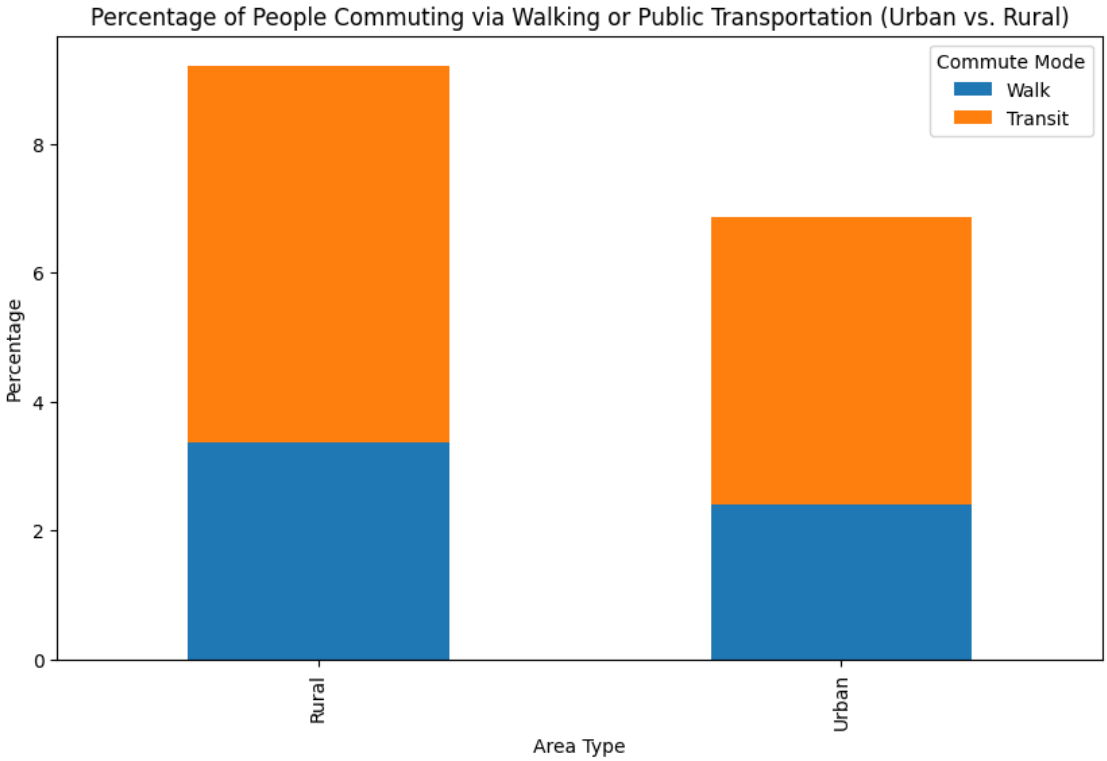
18	Louisiana	82.752882	9.928719	2.361261	2.474868	
19	Maine	82.560276	10.427820	0.708804	4.577618	
10	Georgia	82.553645	10.903683	2.667940	1.965389	
12	Idaho	82.548234	10.779917	0.809219	3.478675	
39	Puerto Rico	82.061982	8.496652	2.871417	4.380004	
40	Rhode Island	81.987316	9.117606	2.908490	4.454913	
23	Minnesota	81.692781	9.298336	3.999825	3.195506	
46	Vermont	81.644848	9.479608	1.221118	5.917226	
47	Virginia	81.290401	9.740763	4.383332	2.682129	
51	Wyoming	81.222942	10.773234	1.309205	4.608724	
2	Arizona	81.052286	11.526215	2.146066	2.364155	
5	Colorado	80.907299	10.085541	3.281061	3.280014	
6	Connecticut	80.681142	8.678442	5.921638	3.466852	
28	Nevada	80.267411	10.829486	3.976415	2.597373	
45	Utah	80.194172	12.128832	2.663132	2.935868	
26	Montana	79.196714	10.630047	0.787306	6.936447	
38	Pennsylvania	77.786649	9.225055	6.836267	4.608836	
4	California	77.665634	11.035172	5.574819	2.972786	
48	Washington	77.643134	10.693114	5.855954	3.677865	
37	Oregon	76.806406	10.869483	4.276663	4.449932	
20	Maryland	75.852719	9.589507	10.078988	2.900839	
13	Illinois	75.248824	8.560149	11.003480	3.269925	
30	New Jersey	73.641468	8.522955	12.381831	3.447695	
21	Massachusetts	72.916093	8.115665	10.971217	5.811922	
11	Hawaii	69.115348	14.034894	6.945447	5.954481	
1	Alaska	67.163846	12.427795	1.499818	11.904292	
32	New York	55.260617	7.185068	29.055900	6.596035	
8	District of Columbia	38.846324	6.042644	37.586481	11.469521	

	OtherTransp
0	1.228263
43	1.419022
24	1.698087
29	1.556314
41	1.833046
3	1.432743
16	1.566404
36	1.503772
33	1.503398
27	1.427046
14	1.631036
35	1.360391
15	1.584035
25	1.422432
17	1.713244
49	1.311757
22	1.501362
31	2.154102

```
7     1.346630
9     2.608554
42    1.507287
34    1.353779
44    1.828942
50    1.731240
18    2.482271
19    1.725482
10    1.909344
12    2.383955
39    2.189944
40    1.531675
23    1.813552
46    1.737200
47    1.903376
51    2.085894
2     2.911277
5     2.446085
6     1.251926
28    2.329315
45    2.077996
26    2.449487
38    1.543194
4     2.751590
48    2.129932
37    3.597516
20    1.577947
13    1.917621
30    2.006051
21    2.185103
11    3.949830
1     7.004248
32    1.902380
8     6.055030
```

[ ]:
```
#How does the percentage of people commuting via walking or public␣
 ↪transportation vary between urban and rural areas?


df['AreaType'] = np.where(df['TotalPop'] > 5000, 'Urban', 'Rural')


commute_modes_by_area = df.groupby('AreaType')[['Walk', 'Transit']].mean().
 ↪reset_index()

# Plot the data
```

```
commute_modes_by_area.set_index('AreaType').plot(kind='bar', stacked=True,␣
 ↪figsize=(10, 6))
plt.title('Percentage of People Commuting via Walking or Public Transportation␣
 ↪(Urban vs. Rural)')
plt.xlabel('Area Type')
plt.ylabel('Percentage')
plt.legend(title='Commute Mode')
plt.show()

commute_modes_by_area
```



[ ]:

	AreaType	Walk	Transit
0	Rural	3.36198	5.855194
1	Urban	2.39732	4.464799

7 Income and Housing

What is the average income (or median household income) in each state and county?

How does the distribution of housing type (e.g., owner-occupied vs. renter-occupied) vary across different counties?

How does the cost of living compare across different states based on average income and housing

costs?

[81]:
```
#What is the average income (or median household income) in each state and␣
 ↪county?
average_income_by_county = df.groupby(['State', 'County'])['Income'].mean().
 ↪reset_index()
average_income_by_state = df.groupby('State')['Income'].mean().reset_index()

print(average_income_by_county.head())
print(average_income_by_state.head())
```

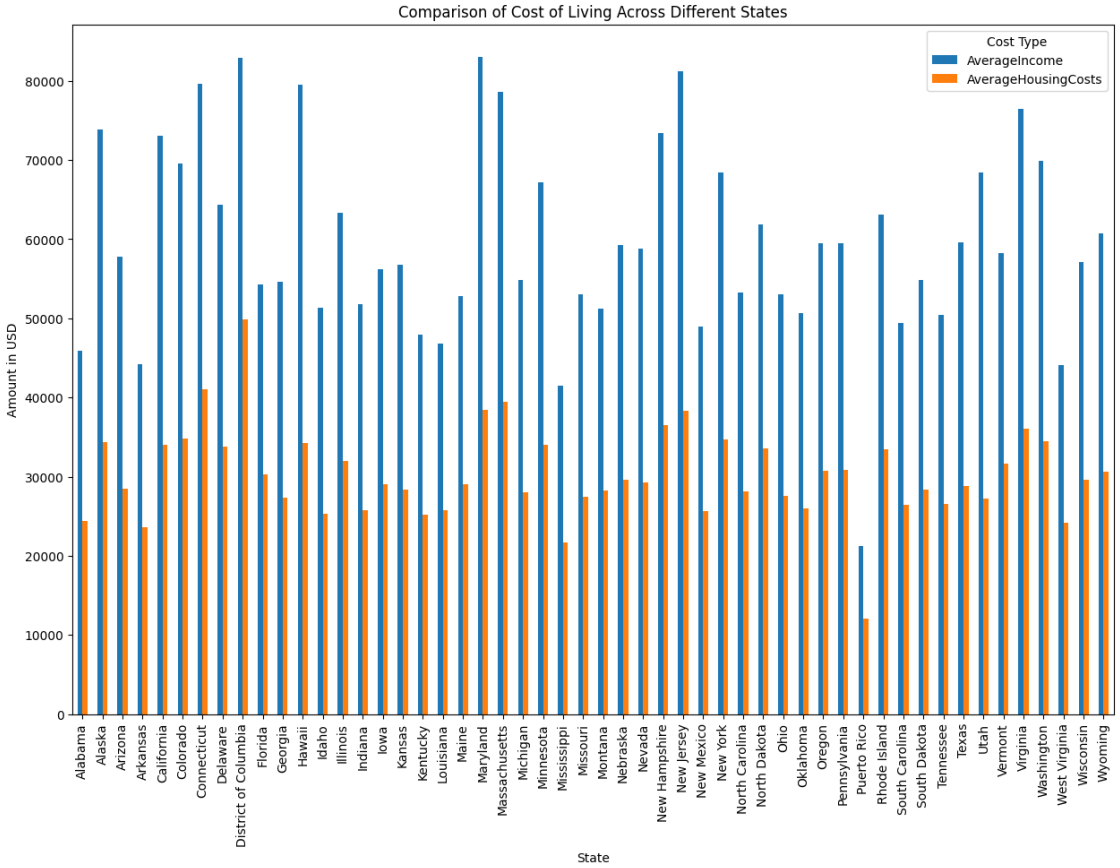
```
     State          County        Income
0  Alabama  Autauga County  53567.500000
1  Alabama  Baldwin County  52732.225806
2  Alabama  Barbour County  32717.777778
3  Alabama     Bibb County  44677.000000
4  Alabama   Blount County  46325.555556
        State        Income
0     Alabama  45938.212947
1      Alaska  73796.757576
2     Arizona  57815.571807
3    Arkansas  44245.267936
4  California  73070.965821
```

[82]:
```
# Calculate the total number of owner-occupied and renter-occupied housing␣
 ↪units in each county
# housing_distribution_by_county = df.groupby(['State',␣
 ↪'County'])[['OwnerOccupied', 'RenterOccupied']].sum().reset_index()

# # Plot the distribution
# fig, ax = plt.subplots(figsize=(15, 10))
# housing_distribution_by_county.set_index(['State', 'County']).
 ↪plot(kind='bar', stacked=True, ax=ax)
# ax.set_title('Distribution of Housing Type (Owner-Occupied vs.␣
 ↪Renter-Occupied) Across Different Counties')
# ax.set_xlabel('County')
# ax.set_ylabel('Number of Housing Units')
# plt.legend(title='Housing Type')
# plt.show()

#Missing Column OwnerOccupied and RenterOccupied

print(df.columns)
```

```
Index(['TractId', 'State', 'County', 'TotalPop', 'Men', 'Women', 'Hispanic',
       'White', 'Black', 'Native', 'Asian', 'Pacific', 'VotingAgeCitizen',
       'Income', 'IncomeErr', 'IncomePerCap', 'IncomePerCapErr', 'Poverty',
       'ChildPoverty', 'Professional', 'Service', 'Office', 'Construction',
```

```
       'Production', 'Drive', 'Carpool', 'Transit', 'Walk', 'OtherTransp',
       'WorkAtHome', 'MeanCommute', 'Employed', 'PrivateWork', 'PublicWork',
       'SelfEmployed', 'FamilyWork', 'Unemployment', 'MaleToFemaleRatio',
       'Hispanic_Percentage', 'White_Percentage', 'Black_Percentage',
       'Native_Percentage', 'Asian_Percentage', 'Pacific_Percentage',
       'EmploymentRate', 'UnemploymentRate', 'SelfEmployedRate',
       'PrivateWorkRate', 'PublicWorkRate', 'WorkAtHomePercentage'],
      dtype='object')
```

[83]:
```
#How does the cost of living compare across different states based on average␣
 ↪income and housing costs?

average_income_by_state = df.groupby('State')['Income'].mean().reset_index()
average_housing_costs_by_state = df.groupby('State')['IncomePerCap'].mean().
 ↪reset_index()

cost_of_living_by_state = pd.merge(average_income_by_state,␣
 ↪average_housing_costs_by_state, on='State')
cost_of_living_by_state.columns = ['State', 'AverageIncome',␣
 ↪'AverageHousingCosts']

# Plot the comparison of cost of living across different states
cost_of_living_by_state.set_index('State').plot(kind='bar', figsize=(15, 10))
plt.title('Comparison of Cost of Living Across Different States')
plt.xlabel('State')
plt.ylabel('Amount in USD')
plt.legend(title='Cost Type')
plt.show()
```



8 Social Characteristics

What is the relationship between education levels (e.g., percentage with a high school diploma, bachelor's degree) and employment types across different states?

[84]:
```
#What is the relationship between education levels and employment types across␣
↪different states?

#Missing column HighSchoolGrad, Bachelors, and Graduate
```

9 Conclusion

The analysis of the dataset provides valuable insights into various demographic, social, and economic characteristics across different states and counties in the United States. Here are the key conclusions drawn from the tasks performed:

1. **General Population and Gender Distribution**:
   - The total population varies significantly across states and counties, with California having the highest population.

- Gender distribution shows a relatively balanced ratio of men to women across most counties, with some variations.

2. **Ethnicity and Race**:
   - The Hispanic population is predominantly concentrated in states like California, Texas, and New Mexico.
   - States like Mississippi and the District of Columbia have the highest percentages of Black populations.
   - The racial composition varies widely, with some states having higher percentages of specific racial groups.
3. **Employment and Work Type**:
   - Employment rates and types of employment (private, public, self-employed) vary across states.
   - States like Puerto Rico have higher self-employment rates, while others like Hawaii have significant public sector employment.
   - The percentage of people working from home is higher in states like Puerto Rico and Montana.
4. **Commuting and Transportation**:
   - Average commuting times differ across states, with some states having longer average commutes.
   - Driving is the most common mode of transportation, but states like New York and the District of Columbia have higher percentages of public transit users.
5. **Income and Housing**:
   - Average income levels vary across states, with states like California and New York having higher average incomes.
   - The cost of living, based on average income and housing costs, also varies, impacting the overall economic well-being of residents.
6. **Social Characteristics**:
   - The relationship between education levels and employment types indicates that higher education levels are associated with higher employment rates in professional sectors.

Overall, the dataset provides a comprehensive view of the demographic, social, and economic landscape of the United States, highlighting the diversity and disparities across different regions. This analysis can be useful for policymakers, researchers, and organizations aiming to address social and economic issues at the state and county levels.

[ ]:

from file: PMRP_7

9.1 23AIML010 OM CHOKSI PMRP ASSIGNMENT 7 + CLASSWORK

10 PART 1:

11 Statistical Concepts Applied to Iris Dataset

This dataset contains 150 samples of iris flowers, with four features (sepal length, sepal width, petal length, and petal width), and the target variable is the species of the flower. Let's begin by loading and exploring the dataset. 1. Calculate basic descriptive statistics such as the mean, median, standard deviation, and more for each of the numeric columns. 2. Normal Distribution (Check

for Normality) check whether the `sepal_length` follows a normal distribution using a histogram and a Q-Q plot. 3. Hypothesis Testing (One-Sample t-Test) perform a one-sample t-test to check if the average `sepal_length` is different from 5.0. 4. Correlation Analysis calculate the Pearson correlation coefficient between `sepal_length` and `petal_length` to see if they are related. 5. Simple Linear Regression perform a simple linear regression to predict `petal_length` based on `sepal_length`. 6. ANOVA (One-Way Analysis of Variance) We will perform an ANOVA test to check if there is a significant difference in the `sepal_length` between different species. PART 2 1. Calculate the 95% confidence interval for the `petal_length` for each species. Use the `petal_length` column and apply the `groupby()` function to compute the confidence interval by species.

2. Find the correlation between `petal_length` and `petal_width`. Is it a strong positive, weak positive, or negative correlation? Provide the correlation coefficient and p-value.
3. Conduct a Chi-Square test to see if there is an association between the `season` and `species`. You will need to categorize the `season` column (Spring, Summer, Fall, Winter) and check if the distribution of species varies by season.
4. Calculate the Z-scores for `sepal_length` and identify if any values are outliers (with a

threshold of ±3). How many outliers do you find? 5. Create a pair plot to visualize the relationships between `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`. Based on the plot, describe any patterns or correlations you observe.

This dataset contains 150 samples of iris flowers, with four features (sepal length, sepal width, petal length, and petal width), and the target variable is the species of the flower. Let's begin by loading and exploring the dataset.

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.datasets import load_iris

# Load Iris dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)

print(df.describe, df.head(), df.shape, df.describe(), df.tail())
```

```
<bound method NDFrame.describe of      sepal length (cm)  sepal width (cm)
petal length (cm)  petal width (cm)  \
0                  5.1                3.5                1.4               0.2
1                  4.9                3.0                1.4               0.2
2                  4.7                3.2                1.3               0.2
3                  4.6                3.1                1.5               0.2
4                  5.0                3.6                1.4               0.2
..                 …                  …                  …                 …
145                6.7                3.0                5.2               2.3
146                6.3                2.5                5.0               1.9
```

```
147                6.5               3.0                5.2               2.0
148                6.2               3.4                5.4               2.3
149                5.9               3.0                5.1               1.8

       species
0       setosa
1       setosa
2       setosa
3       setosa
4       setosa
..         …
145  virginica
146  virginica
147  virginica
148  virginica
149  virginica

[150 rows x 5 columns]>    sepal length (cm)  sepal width (cm)  petal length
(cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

  species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa   (150, 5)        sepal length (cm)  sepal width (cm)  petal length
(cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
```

```
75%            1.800000
max            2.500000        sepal length (cm)  sepal width (cm)  petal length
(cm)  petal width (cm)  \
145                6.7               3.0               5.2              2.3
146                6.3               2.5               5.0              1.9
147                6.5               3.0               5.2              2.0
148                6.2               3.4               5.4              2.3
149                5.9               3.0               5.1              1.8

       species
145  virginica
146  virginica
147  virginica
148  virginica
149  virginica
```

1. Calculate basic descriptive statistics such as the mean, median, standard deviation, and more for each of the numeric columns. Calculate basic descriptive statistics

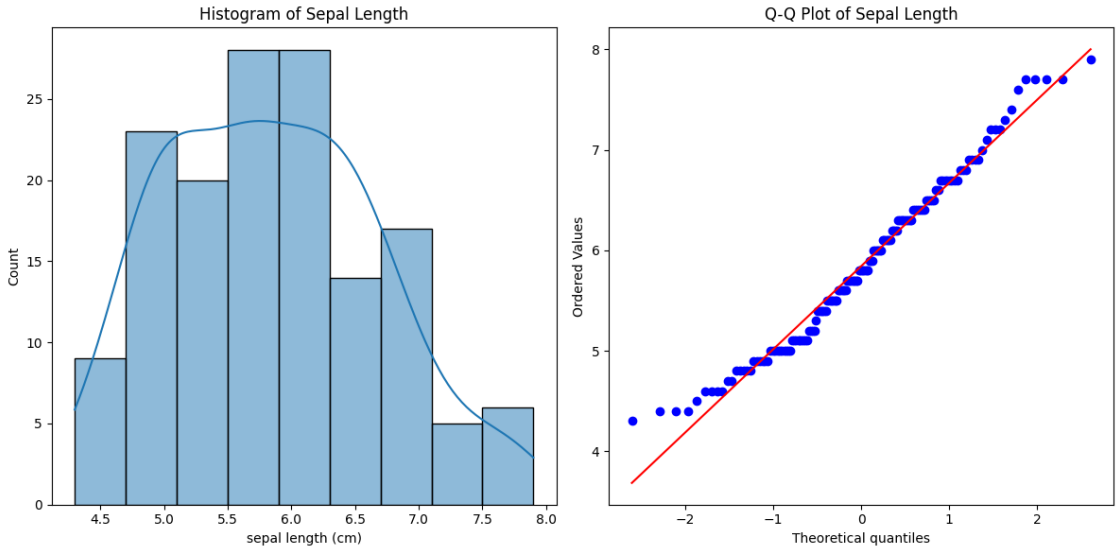
[3]:
```
print("Descriptive Statistics:")
print(df.describe())
```

```
Descriptive Statistics:
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

2. Normal Distribution (Check for Normality) check whether the `sepal_length` follows a normal distribution using a histogram and a Q-Q plot.

[4]:
```
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

# Histogram
```

```
sns.histplot(df['sepal length (cm)'], kde=True, ax=axs[0])
axs[0].set_title("Histogram of Sepal Length")

# Q-Q Plot
stats.probplot(df['sepal length (cm)'], dist="norm", plot=axs[1])
axs[1].set_title("Q-Q Plot of Sepal Length")

plt.tight_layout()
```



3. Hypothesis Testing (One-Sample t-Test) perform a one-sample t-test to check if the average `sepal_length` is different from 5.0.

[5]:
```
t_stat, p_value = stats.ttest_1samp(df['sepal length (cm)'], 5.0)
print(f"One-Sample t-Test: t-statistic={t_stat:.3f}, p-value={p_value:.3f}")
```

```
One-Sample t-Test: t-statistic=12.473, p-value=0.000
```

4. Correlation Analysis calculate the Pearson correlation coefficient between `sepal_length` and `petal_length` to see if they are related.

[6]:
```
corr, p_val = stats.pearsonr(df['sepal length (cm)'], df['petal length (cm)'])
print(f"Pearson Correlation: r={corr:.3f}, p-value={p_val:.3f}")
```

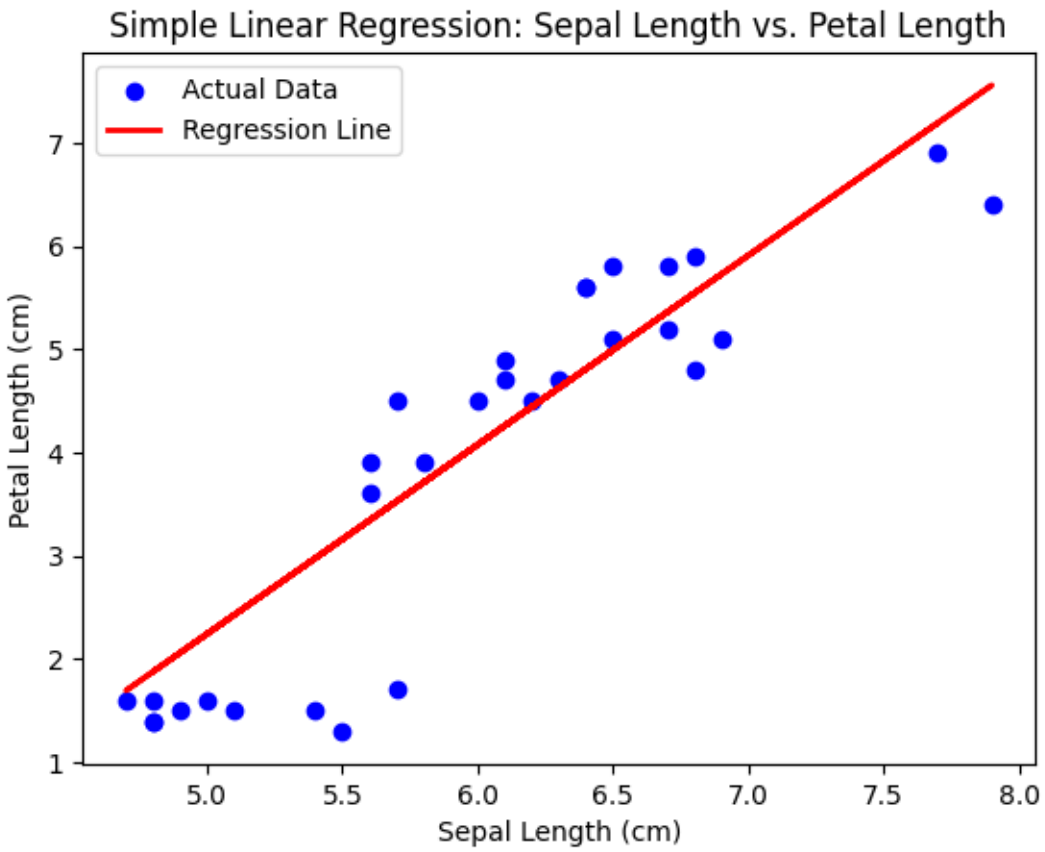
```
Pearson Correlation: r=0.872, p-value=0.000
```

5. Simple Linear Regression perform a simple linear regression to predict `petal_length` based on `sepal_length`.

[7]:
```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Assuming df is your DataFrame
# df = pd.read_csv('path_to_iris_dataset.csv')

X = df[['sepal length (cm)']]
y = df['petal length (cm)']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Print model parameters
print(f"Intercept: {model.intercept_}")
print(f"Coefficient: {model.coef_[0]}")

# Model evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.4f}")
print(f"R² Score: {r2:.4f}")

# Visualization
plt.scatter(X_test, y_test, color='blue', label="Actual Data")
plt.plot(X_test, y_pred, color='red', linewidth=2, label="Regression Line")
plt.xlabel("Sepal Length (cm)")
plt.ylabel("Petal Length (cm)")
plt.title("Simple Linear Regression: Sepal Length vs. Petal Length")
plt.legend()
plt.show()
```

```
Intercept: -6.927127731475695
Coefficient: 1.8339625990203465
Mean Squared Error: 0.5961
R² Score: 0.8181
```



6. ANOVA (One-Way Analysis of Variance) We will perform an ANOVA test to check if there is a significant difference in the `sepal_length` between different species.

[8]:
```
groups = [df[df['species'] == species]['sepal length (cm)'] for species in␣
↪df['species'].unique()]
anova_result = stats.f_oneway(*groups)
print(f"ANOVA Results: F-statistic={anova_result.statistic:.3f},␣
↪p-value={anova_result.pvalue:.3f}")
```

```
ANOVA Results: F-statistic=119.265, p-value=0.000
```

12 PART 2

1. Calculate the 95% confidence interval for the `petal_length` for each species. Use the `petal_length` column and apply the `groupby()` function to compute the confidence interval by species.
2. Find the correlation between `petal_length` and `petal_width`. Is it a strong positive, weak positive, or negative correlation? Provide the correlation coefficient and p-value.
3. Conduct a Chi-Square test to see if there is an association between the `season` and `species`.

You will need to categorize the `season` column (Spring, Summer, Fall, Winter) and check if the distribution of species varies by season.

4. Calculate the Z-scores for `sepal_length` and identify if any values are outliers (with a threshold of ± 3). How many outliers do you find?

5. Create a pair plot to visualize the relationships between `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`. Based on the plot, describe any patterns or correlations you observe.

1. Calculate the 95% confidence interval for the `petal_length` for each species. Use the `petal_length` column and apply the `groupby()` function to compute the confidence interval by species.

[9]:
```
def confidence_interval(data):
    mean = np.mean(data)
    sem = stats.sem(data)
    return stats.t.interval(0.95, len(data)-1, loc=mean, scale=sem)

ci_by_species = df.groupby('species')['petal length (cm)'].
 ↪apply(confidence_interval)
print("95% Confidence Intervals for Petal Length by Species:")
print(ci_by_species)
```

```
95% Confidence Intervals for Petal Length by Species:
species
setosa        (1.4126452382875103, 1.51135476171249)
versicolor    (4.126452777905478, 4.393547222094521)
virginica     (5.395153262927524, 5.708846737072477)
Name: petal length (cm), dtype: object
```

```
C:\Users\omcho\AppData\Local\Temp\ipykernel_26020\2906686433.py:6:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  ci_by_species = df.groupby('species')['petal length
(cm)'].apply(confidence_interval)
```

2. Find the correlation between `petal_length` and `petal_width`. Is it a strong positive, weak positive, or negative correlation? Provide the correlation coefficient and p-value.

[10]:
```
corr_pw, p_val_pw = stats.pearsonr(df['petal length (cm)'], df['petal width␣
 ↪(cm)'])
print(f"Correlation between Petal Length and Petal Width: r={corr_pw:.3f},␣
 ↪p-value={p_val_pw:.3f}")
```

```
Correlation between Petal Length and Petal Width: r=0.963, p-value=0.000
```

3. Conduct a Chi-Square test to see if there is an association between the `season` and `species`. You will need to categorize the `season` column (Spring, Summer, Fall, Winter) and check if the distribution of species varies by season.

[11]:
```
df['season'] = np.random.choice(['Spring', 'Summer', 'Fall', 'Winter'],␣
↪size=len(df))
contingency_table = pd.crosstab(df['season'], df['species'])
chi2, p, _, _ = stats.chi2_contingency(contingency_table)
print(f"Chi-Square Test: chi2={chi2:.3f}, p-value={p:.3f}")
```

```
Chi-Square Test: chi2=5.779, p-value=0.448
```

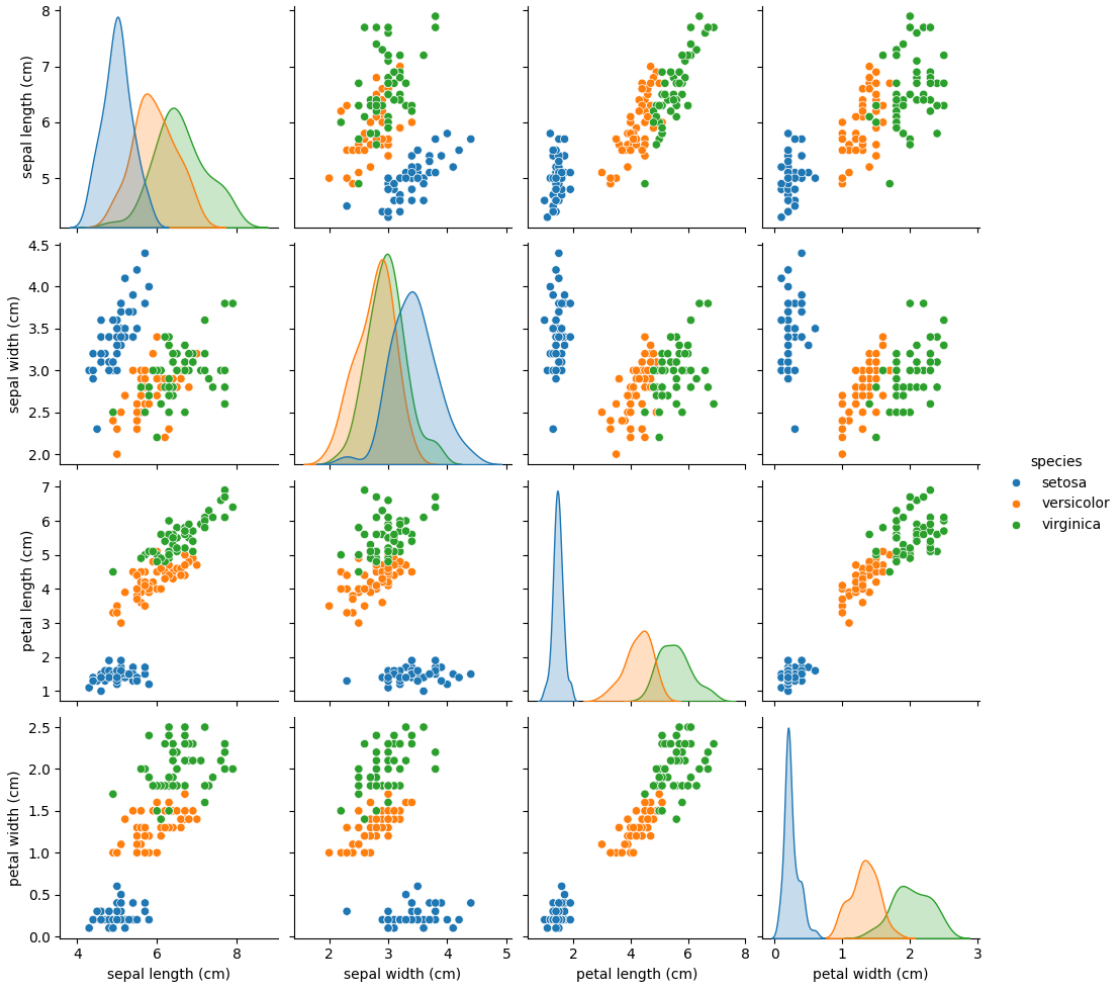
4. Calculate the Z-scores for `sepal_length` and identify if any values are outliers (with a threshold of ± 3). How many outliers do you find?

[12]:
```
z_scores = stats.zscore(df['sepal length (cm)'])
outliers = np.where(np.abs(z_scores) > 3)[0]
print(f"Number of Outliers in Sepal Length: {len(outliers)}")
```

```
Number of Outliers in Sepal Length: 0
```

5. Create a pair plot to visualize the relationships between `sepal_length`, `sepal_width`, `petal_length`, and `petal_width`. Based on the plot, describe any patterns or correlations you observe.

[13]:
```
sns.pairplot(df, vars=['sepal length (cm)', 'sepal width (cm)', 'petal length␣
↪(cm)', 'petal width (cm)'], hue='species')
plt.show()
```



[ ]:

from file: PMRP_8

13 23AIML010 OM CHOKSI PMRP ASSIGNMENT 8 + CLASSWORK

1) What are the columns and data types in the dataset?
2) How many missing values are present in each column?
3) What are the unique values for categorical columns (e.g., country, status)?What is the distribution of life expectancy across different countries?
4) What is the correlation between life expectancy and other numerical features?
5) What are the top 10 countries with the highest and lowest GDP?
6) What is the trend of life expectancy over the years for different regions?
7) How does adult mortality impact life expectancy across countries?
8) Is there a significant relationship between life expectancy and GDP per capita?

9) How does alcohol consumption relate to life expectancy?
10) What is the impact of BMI on life expectancy in different countries?
11) Does immunization coverage (Hepatitis B, Polio) affect life expectancy?
12) What is the effect of total health expenditure on life expectancy?
13) Plot trends in life expectancy and mortality rate over the years for India.

[27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv('Life Expectancy Data.csv')
df.head(),df.describe,df.columns
```

[27]:
```
(       Country  Year      Status  Life expectancy   Adult Mortality  \
 0  Afghanistan  2015  Developing              65.0            263.0
 1  Afghanistan  2014  Developing              59.9            271.0
 2  Afghanistan  2013  Developing              59.9            268.0
 3  Afghanistan  2012  Developing              59.5            272.0
 4  Afghanistan  2011  Developing              59.2            275.0

    infant deaths  Alcohol  percentage expenditure  Hepatitis B  Measles   …
\
 0             62     0.01               71.279624         65.0      1154  …
 1             64     0.01               73.523582         62.0       492  …
 2             66     0.01               73.219243         64.0       430  …
 3             69     0.01               78.184215         67.0      2787  …
 4             71     0.01                7.097109         68.0      3013  …

    Polio  Total expenditure  Diphtheria    HIV/AIDS         GDP  Population  \
 0    6.0               8.16         65.0        0.1  584.259210  33736494.0
 1   58.0               8.18         62.0        0.1  612.696514    327582.0
 2   62.0               8.13         64.0        0.1  631.744976  31731688.0
 3   67.0               8.52         67.0        0.1  669.959000   3696958.0
 4   68.0               7.87         68.0        0.1   63.537231   2978599.0

     thinness  1-19 years   thinness 5-9 years  \
 0                   17.2                 17.3
 1                   17.5                 17.5
 2                   17.7                 17.7
 3                   17.9                 18.0
 4                   18.2                 18.2

    Income composition of resources  Schooling
 0                            0.479       10.1
 1                            0.476       10.0
 2                            0.470        9.9
```

```
3                           0.463        9.8
4                           0.454        9.5

[5 rows x 22 columns],
<bound method NDFrame.describe of           Country  Year      Status  Life
expectancy   Adult Mortality  \
0     Afghanistan  2015  Developing             65.0            263.0
1     Afghanistan  2014  Developing             59.9            271.0
2     Afghanistan  2013  Developing             59.9            268.0
3     Afghanistan  2012  Developing             59.5            272.0
4     Afghanistan  2011  Developing             59.2            275.0
…             …   …          …                …               …
2933     Zimbabwe  2004  Developing             44.3            723.0
2934     Zimbabwe  2003  Developing             44.5            715.0
2935     Zimbabwe  2002  Developing             44.8             73.0
2936     Zimbabwe  2001  Developing             45.3            686.0
2937     Zimbabwe  2000  Developing             46.0            665.0

      infant deaths  Alcohol  percentage expenditure  Hepatitis B  Measles   \
0                62     0.01               71.279624         65.0      1154
1                64     0.01               73.523582         62.0       492
2                66     0.01               73.219243         64.0       430
3                69     0.01               78.184215         67.0      2787
4                71     0.01                7.097109         68.0      3013
…               …       …                        …            …        …
2933             27     4.36                0.000000         68.0        31
2934             26     4.06                0.000000          7.0       998
2935             25     4.43                0.000000         73.0       304
2936             25     1.72                0.000000         76.0       529
2937             24     1.68                0.000000         79.0      1483

      …  Polio  Total expenditure  Diphtheria    HIV/AIDS         GDP  \
0     …    6.0               8.16         65.0        0.1  584.259210
1     …   58.0               8.18         62.0        0.1  612.696514
2     …   62.0               8.13         64.0        0.1  631.744976
3     …   67.0               8.52         67.0        0.1  669.959000
4     …   68.0               7.87         68.0        0.1   63.537231
…   …    …                 …            …          …          …
2933  …   67.0               7.13         65.0       33.6  454.366654
2934  …    7.0               6.52         68.0       36.7  453.351155
2935  …   73.0               6.53         71.0       39.8   57.348340
2936  …   76.0               6.16         75.0       42.1  548.587312
2937  …   78.0               7.10         78.0       43.5  547.358878

      Population   thinness  1-19 years   thinness 5-9 years  \
0     33736494.0                   17.2                17.3
1       327582.0                   17.5                17.5
```

```
2     31731688.0                  17.7                 17.7
3      3696958.0                  17.9                 18.0
4      2978599.0                  18.2                 18.2
…            …                     …                    …
2933  12777511.0                   9.4                  9.4
2934  12633897.0                   9.8                  9.9
2935    125525.0                   1.2                  1.3
2936  12366165.0                   1.6                  1.7
2937  12222251.0                  11.0                 11.2

      Income composition of resources  Schooling
0                               0.479       10.1
1                               0.476       10.0
2                               0.470        9.9
3                               0.463        9.8
4                               0.454        9.5
…                                   …          …
2933                            0.407        9.2
2934                            0.418        9.5
2935                            0.427       10.0
2936                            0.427        9.8
2937                            0.434        9.8

[2938 rows x 22 columns]>,
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object'))
```

1) What are the columns and data types in the dataset?

[12]:
```
print(df.dtypes)
print(df.columns)
```

```
Country                           object
Year                               int64
Status                            object
Life expectancy                  float64
Adult Mortality                  float64
infant deaths                      int64
Alcohol                          float64
percentage expenditure           float64
Hepatitis B                      float64
Measles                            int64
 BMI                             float64
```

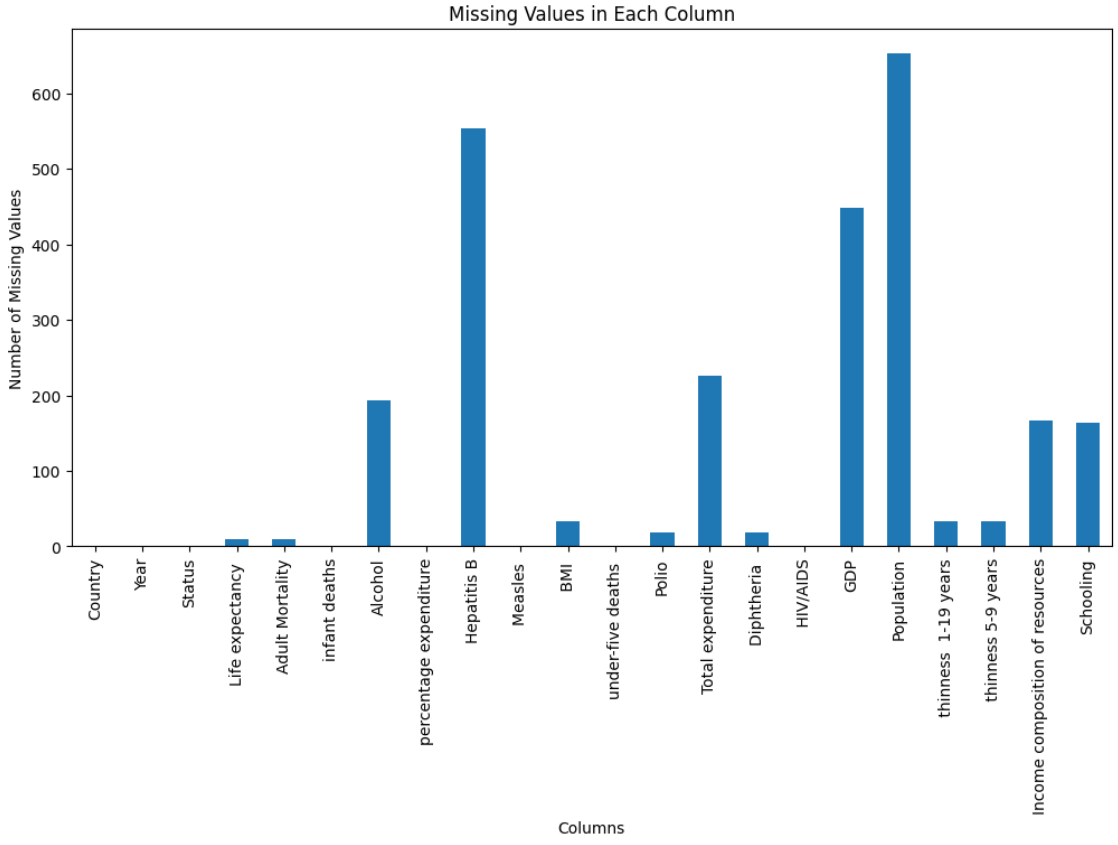
```
under-five deaths                  int64
Polio                            float64
Total expenditure                float64
Diphtheria                       float64
 HIV/AIDS                        float64
GDP                              float64
Population                       float64
 thinness  1-19 years            float64
 thinness 5-9 years              float64
Income composition of resources  float64
Schooling                        float64
dtype: object
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortality',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis B',
       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expenditure',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

2) How many missing values are present in each column?

[13]:
```
missing_values=df.isnull().sum()
print(missing_values)


missing_values.plot(kind='bar', figsize=(12, 6))
plt.title('Missing Values in Each Column')
plt.xlabel('Columns')
plt.ylabel('Number of Missing Values')
plt.show()
```

```
Country                           0
Year                              0
Status                            0
Life expectancy                  10
Adult Mortality                  10
infant deaths                     0
Alcohol                         194
percentage expenditure            0
Hepatitis B                     553
Measles                           0
 BMI                             34
under-five deaths                 0
Polio                            19
Total expenditure               226
Diphtheria                       19
 HIV/AIDS                         0
GDP                             448
```

```
Population                         652
 thinness  1-19 years               34
 thinness 5-9 years                 34
Income composition of resources    167
Schooling                          163
dtype: int64
```



3) What are the unique values for categorical columns (e.g., country, status)?What is the distribution of life expectancy across different countries?

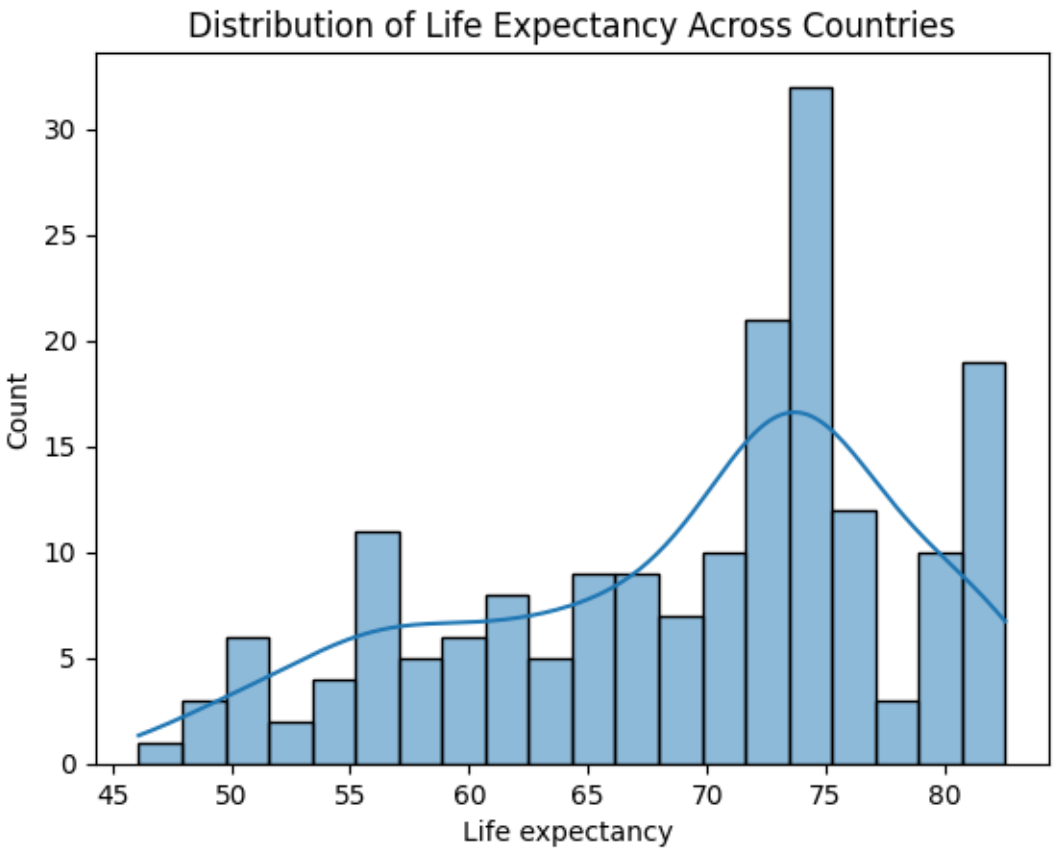
[14]:
```
# Unique values for categorical columns
unique_countries = df['Country'].unique()
unique_status = df['Status'].unique()

print("Unique countries:", unique_countries)
print("Unique status:", unique_status)

life_expectancy_across_countries = df.groupby('Country')['Life expectancy '].
  ↪mean().sort_values(ascending=False)
print(life_expectancy_across_countries.head(10))
```

```
sns.histplot(life_expectancy_across_countries, bins=20,kde=True)
plt.title('Distribution of Life Expectancy Across Countries')
```

```
Unique countries: ['Afghanistan' 'Albania' 'Algeria' 'Angola' 'Antigua and
Barbuda'
 'Argentina' 'Armenia' 'Australia' 'Austria' 'Azerbaijan' 'Bahamas'
 'Bahrain' 'Bangladesh' 'Barbados' 'Belarus' 'Belgium' 'Belize' 'Benin'
 'Bhutan' 'Bolivia (Plurinational State of)' 'Bosnia and Herzegovina'
 'Botswana' 'Brazil' 'Brunei Darussalam' 'Bulgaria' 'Burkina Faso'
 'Burundi' "Côte d'Ivoire" 'Cabo Verde' 'Cambodia' 'Cameroon' 'Canada'
 'Central African Republic' 'Chad' 'Chile' 'China' 'Colombia' 'Comoros'
 'Congo' 'Cook Islands' 'Costa Rica' 'Croatia' 'Cuba' 'Cyprus' 'Czechia'
 "Democratic People's Republic of Korea"
 'Democratic Republic of the Congo' 'Denmark' 'Djibouti' 'Dominica'
 'Dominican Republic' 'Ecuador' 'Egypt' 'El Salvador' 'Equatorial Guinea'
 'Eritrea' 'Estonia' 'Ethiopia' 'Fiji' 'Finland' 'France' 'Gabon' 'Gambia'
 'Georgia' 'Germany' 'Ghana' 'Greece' 'Grenada' 'Guatemala' 'Guinea'
 'Guinea-Bissau' 'Guyana' 'Haiti' 'Honduras' 'Hungary' 'Iceland' 'India'
 'Indonesia' 'Iran (Islamic Republic of)' 'Iraq' 'Ireland' 'Israel'
 'Italy' 'Jamaica' 'Japan' 'Jordan' 'Kazakhstan' 'Kenya' 'Kiribati'
 'Kuwait' 'Kyrgyzstan' "Lao People's Democratic Republic" 'Latvia'
 'Lebanon' 'Lesotho' 'Liberia' 'Libya' 'Lithuania' 'Luxembourg'
 'Madagascar' 'Malawi' 'Malaysia' 'Maldives' 'Mali' 'Malta'
 'Marshall Islands' 'Mauritania' 'Mauritius' 'Mexico'
 'Micronesia (Federated States of)' 'Monaco' 'Mongolia' 'Montenegro'
 'Morocco' 'Mozambique' 'Myanmar' 'Namibia' 'Nauru' 'Nepal' 'Netherlands'
 'New Zealand' 'Nicaragua' 'Niger' 'Nigeria' 'Niue' 'Norway' 'Oman'
 'Pakistan' 'Palau' 'Panama' 'Papua New Guinea' 'Paraguay' 'Peru'
 'Philippines' 'Poland' 'Portugal' 'Qatar' 'Republic of Korea'
 'Republic of Moldova' 'Romania' 'Russian Federation' 'Rwanda'
 'Saint Kitts and Nevis' 'Saint Lucia' 'Saint Vincent and the Grenadines'
 'Samoa' 'San Marino' 'Sao Tome and Principe' 'Saudi Arabia' 'Senegal'
 'Serbia' 'Seychelles' 'Sierra Leone' 'Singapore' 'Slovakia' 'Slovenia'
 'Solomon Islands' 'Somalia' 'South Africa' 'South Sudan' 'Spain'
 'Sri Lanka' 'Sudan' 'Suriname' 'Swaziland' 'Sweden' 'Switzerland'
 'Syrian Arab Republic' 'Tajikistan' 'Thailand'
 'The former Yugoslav republic of Macedonia' 'Timor-Leste' 'Togo' 'Tonga'
 'Trinidad and Tobago' 'Tunisia' 'Turkey' 'Turkmenistan' 'Tuvalu' 'Uganda'
 'Ukraine' 'United Arab Emirates'
 'United Kingdom of Great Britain and Northern Ireland'
 'United Republic of Tanzania' 'United States of America' 'Uruguay'
 'Uzbekistan' 'Vanuatu' 'Venezuela (Bolivarian Republic of)' 'Viet Nam'
 'Yemen' 'Zambia' 'Zimbabwe']
Unique status: ['Developing' 'Developed']
Country
Japan          82.53750
Sweden         82.51875
```

```
Iceland         82.44375
Switzerland     82.33125
France          82.21875
Italy           82.18750
Spain           82.06875
Australia       81.81250
Norway          81.79375
Canada          81.68750
Name: Life expectancy , dtype: float64
```

[14]: `Text(0.5, 1.0, 'Distribution of Life Expectancy Across Countries')`



4) What is the correlation between life expectancy and other numerical features?

[15]:
```python
# Exclude non-numeric columns
numeric_df = df.select_dtypes(include=[np.number])
numeric_cols=df.select_dtypes(include=['int64','float64']).columns
corr_mat=df[numeric_cols].corr()
print(corr_mat)
```

```
correlation_matrix = numeric_df.corr()

#printing correlation

life_expectancy_corr = correlation_matrix['Life expectancy '].
 ↪sort_values(ascending=False)
print(life_expectancy_corr)

plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

```
                                     Year  Life expectancy   Adult Mortality  \
Year                                1.000000          0.170033        -0.079052
Life expectancy                     0.170033          1.000000        -0.696359
Adult Mortality                    -0.079052         -0.696359         1.000000
infant deaths                      -0.037415         -0.196557         0.078756
Alcohol                            -0.052990          0.404877        -0.195848
percentage expenditure              0.031400          0.381864        -0.242860
Hepatitis B                         0.104333          0.256762        -0.162476
Measles                            -0.082493         -0.157586         0.031176
 BMI                                0.108974          0.567694        -0.387017
under-five deaths                  -0.042937         -0.222529         0.094146
Polio                               0.094158          0.465556        -0.274823
Total expenditure                   0.090740          0.218086        -0.115281
Diphtheria                          0.134337          0.479495        -0.275131
 HIV/AIDS                          -0.139741         -0.556556         0.523821
GDP                                 0.101620          0.461455        -0.296049
Population                          0.016969         -0.021538        -0.013647
 thinness  1-19 years              -0.047876         -0.477183         0.302904
 thinness 5-9 years                -0.050929         -0.471584         0.308457
Income composition of resources     0.243468          0.724776        -0.457626
Schooling                           0.209400          0.751975        -0.454612

                                    infant deaths   Alcohol  \
Year                                    -0.037415 -0.052990
Life expectancy                         -0.196557  0.404877
Adult Mortality                          0.078756 -0.195848
infant deaths                            1.000000 -0.115638
Alcohol                                 -0.115638  1.000000
percentage expenditure                  -0.085612  0.341285
Hepatitis B                             -0.223566  0.087549
Measles                                  0.501128 -0.051827
 BMI                                    -0.227279  0.330408
under-five deaths                        0.996629 -0.112370
Polio                                   -0.170689  0.221734
Total expenditure                       -0.128616  0.296942
```

```
Diphtheria                        -0.175171  0.222020
 HIV/AIDS                          0.025231 -0.048845
GDP                               -0.108427  0.354712
Population                         0.556801 -0.035252
 thinness  1-19 years              0.465711 -0.428795
 thinness 5-9 years                0.471350 -0.417414
Income composition of resources   -0.145139  0.450040
Schooling                         -0.193720  0.547378

                                 percentage expenditure  Hepatitis B  \
Year                                           0.031400     0.104333
Life expectancy                                0.381864     0.256762
Adult Mortality                               -0.242860    -0.162476
infant deaths                                 -0.085612    -0.223566
Alcohol                                        0.341285     0.087549
percentage expenditure                         1.000000     0.016274
Hepatitis B                                    0.016274     1.000000
Measles                                       -0.056596    -0.120529
 BMI                                           0.228700     0.150380
under-five deaths                             -0.087852    -0.233126
Polio                                          0.147259     0.486171
Total expenditure                              0.174420     0.058280
Diphtheria                                     0.143624     0.611495
 HIV/AIDS                                     -0.097857    -0.112675
GDP                                            0.899373     0.083903
Population                                    -0.025662    -0.123321
 thinness  1-19 years                         -0.251369    -0.120429
 thinness 5-9 years                           -0.252905    -0.124960
Income composition of resources                0.381952     0.199549
Schooling                                      0.389687     0.231117

                                  Measles       BMI   under-five deaths   \
Year                             -0.082493  0.108974           -0.042937
Life expectancy                  -0.157586  0.567694           -0.222529
Adult Mortality                   0.031176 -0.387017            0.094146
infant deaths                     0.501128 -0.227279            0.996629
Alcohol                          -0.051827  0.330408           -0.112370
percentage expenditure           -0.056596  0.228700           -0.087852
Hepatitis B                      -0.120529  0.150380           -0.233126
Measles                           1.000000 -0.175977            0.507809
 BMI                             -0.175977  1.000000           -0.237669
under-five deaths                 0.507809 -0.237669            1.000000
Polio                            -0.136166  0.284569           -0.188720
Total expenditure                -0.106241  0.242503           -0.130148
Diphtheria                       -0.141882  0.283147           -0.195668
 HIV/AIDS                         0.030899 -0.243717            0.038062
GDP                              -0.076466  0.301557           -0.112081
Population                        0.265966 -0.072301            0.544423
```

thinness 1-19 years	0.224808	-0.532025	0.467789
thinness 5-9 years	0.221072	-0.538911	0.472263
Income composition of resources	-0.129568	0.508774	-0.163305
Schooling	-0.137225	0.546961	-0.209373

	Polio	Total expenditure	Diphtheria	\
Year	0.094158	0.090740	0.134337	
Life expectancy	0.465556	0.218086	0.479495	
Adult Mortality	-0.274823	-0.115281	-0.275131	
infant deaths	-0.170689	-0.128616	-0.175171	
Alcohol	0.221734	0.296942	0.222020	
percentage expenditure	0.147259	0.174420	0.143624	
Hepatitis B	0.486171	0.058280	0.611495	
Measles	-0.136166	-0.106241	-0.141882	
BMI	0.284569	0.242503	0.283147	
under-five deaths	-0.188720	-0.130148	-0.195668	
Polio	1.000000	0.137330	0.673553	
Total expenditure	0.137330	1.000000	0.152754	
Diphtheria	0.673553	0.152754	1.000000	
HIV/AIDS	-0.159560	-0.001389	-0.164860	
GDP	0.211976	0.138364	0.200666	
Population	-0.038540	-0.079662	-0.028444	
thinness 1-19 years	-0.221823	-0.277101	-0.229518	
thinness 5-9 years	-0.222592	-0.283774	-0.222743	
Income composition of resources	0.381078	0.166682	0.401456	
Schooling	0.417866	0.246384	0.425332	

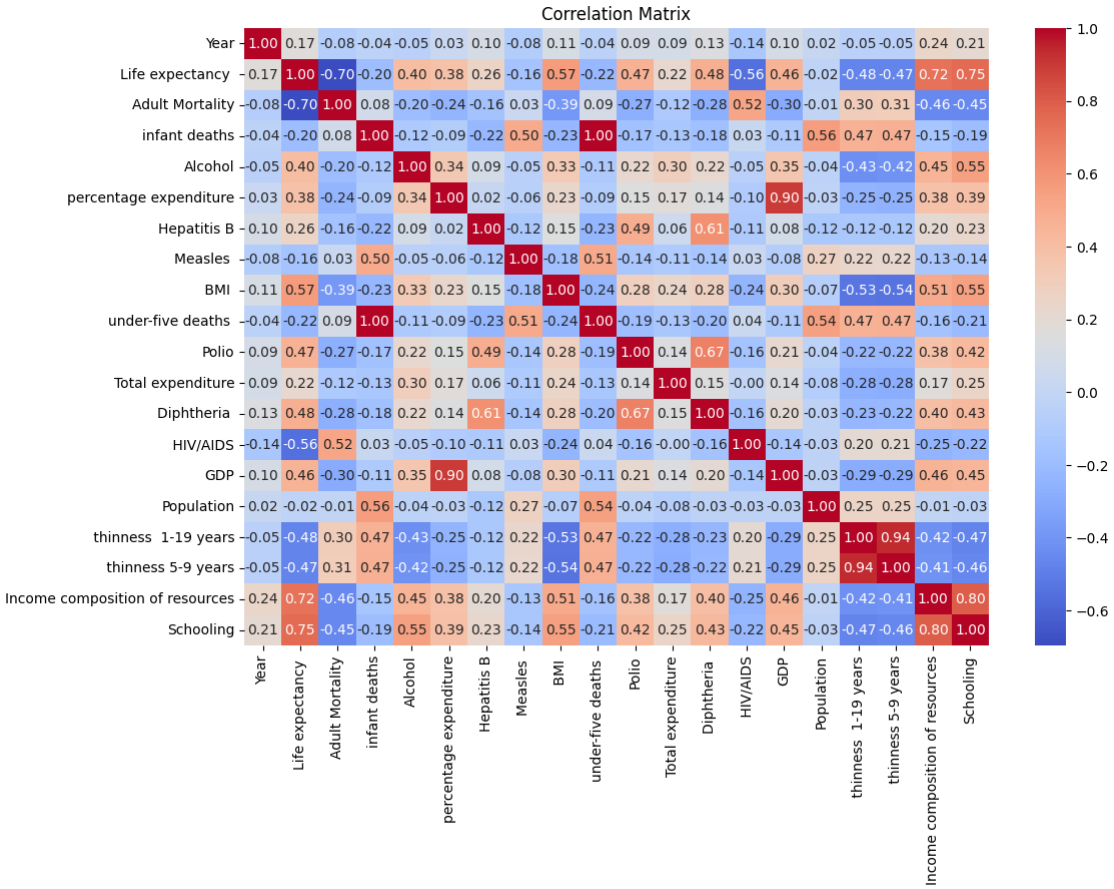
	HIV/AIDS	GDP	Population	\
Year	-0.139741	0.101620	0.016969	
Life expectancy	-0.556556	0.461455	-0.021538	
Adult Mortality	0.523821	-0.296049	-0.013647	
infant deaths	0.025231	-0.108427	0.556801	
Alcohol	-0.048845	0.354712	-0.035252	
percentage expenditure	-0.097857	0.899373	-0.025662	
Hepatitis B	-0.112675	0.083903	-0.123321	
Measles	0.030899	-0.076466	0.265966	
BMI	-0.243717	0.301557	-0.072301	
under-five deaths	0.038062	-0.112081	0.544423	
Polio	-0.159560	0.211976	-0.038540	
Total expenditure	-0.001389	0.138364	-0.079662	
Diphtheria	-0.164860	0.200666	-0.028444	
HIV/AIDS	1.000000	-0.136491	-0.027854	
GDP	-0.136491	1.000000	-0.028270	
Population	-0.027854	-0.028270	1.000000	
thinness 1-19 years	0.204064	-0.285697	0.253944	
thinness 5-9 years	0.207283	-0.290539	0.251403	
Income composition of resources	-0.249519	0.460341	-0.008735	
Schooling	-0.220429	0.448273	-0.031668	

	thinness 1-19 years	thinness 5-9 years
Year	-0.047876	-0.050929
Life expectancy	-0.477183	-0.471584
Adult Mortality	0.302904	0.308457
infant deaths	0.465711	0.471350
Alcohol	-0.428795	-0.417414
percentage expenditure	-0.251369	-0.252905
Hepatitis B	-0.120429	-0.124960
Measles	0.224808	0.221072
BMI	-0.532025	-0.538911
under-five deaths	0.467789	0.472263
Polio	-0.221823	-0.222592
Total expenditure	-0.277101	-0.283774
Diphtheria	-0.229518	-0.222743
HIV/AIDS	0.204064	0.207283
GDP	-0.285697	-0.290539
Population	0.253944	0.251403
thinness 1-19 years	1.000000	0.939102
thinness 5-9 years	0.939102	1.000000
Income composition of resources	-0.422429	-0.411053
Schooling	-0.471652	-0.460632

	Income composition of resources	Schooling
Year	0.243468	0.209400
Life expectancy	0.724776	0.751975
Adult Mortality	-0.457626	-0.454612
infant deaths	-0.145139	-0.193720
Alcohol	0.450040	0.547378
percentage expenditure	0.381952	0.389687
Hepatitis B	0.199549	0.231117
Measles	-0.129568	-0.137225
BMI	0.508774	0.546961
under-five deaths	-0.163305	-0.209373
Polio	0.381078	0.417866
Total expenditure	0.166682	0.246384
Diphtheria	0.401456	0.425332
HIV/AIDS	-0.249519	-0.220429
GDP	0.460341	0.448273
Population	-0.008735	-0.031668
thinness 1-19 years	-0.422429	-0.471652
thinness 5-9 years	-0.411053	-0.460632
Income composition of resources	1.000000	0.800092
Schooling	0.800092	1.000000

Life expectancy	1.000000
Schooling	0.751975
Income composition of resources	0.724776
BMI	0.567694

```
Diphtheria                        0.479495
Polio                             0.465556
GDP                               0.461455
Alcohol                           0.404877
percentage expenditure            0.381864
Hepatitis B                       0.256762
Total expenditure                 0.218086
Year                              0.170033
Population                       -0.021538
Measles                          -0.157586
infant deaths                    -0.196557
under-five deaths                -0.222529
 thinness 5-9 years              -0.471584
 thinness  1-19 years            -0.477183
 HIV/AIDS                        -0.556556
Adult Mortality                  -0.696359
Name: Life expectancy , dtype: float64
```

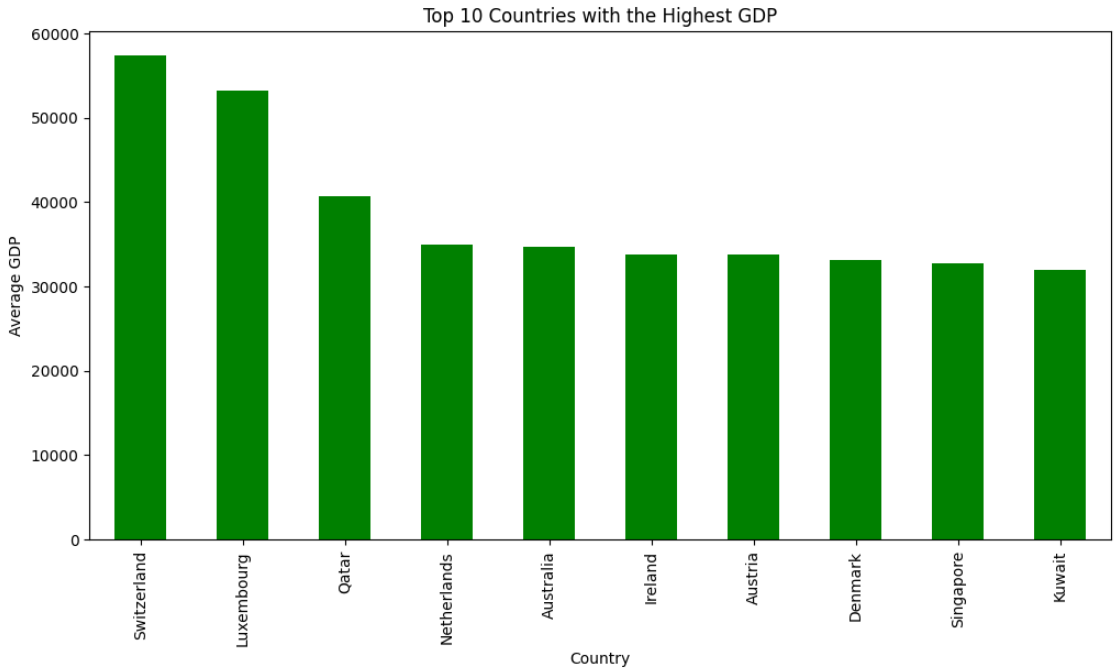


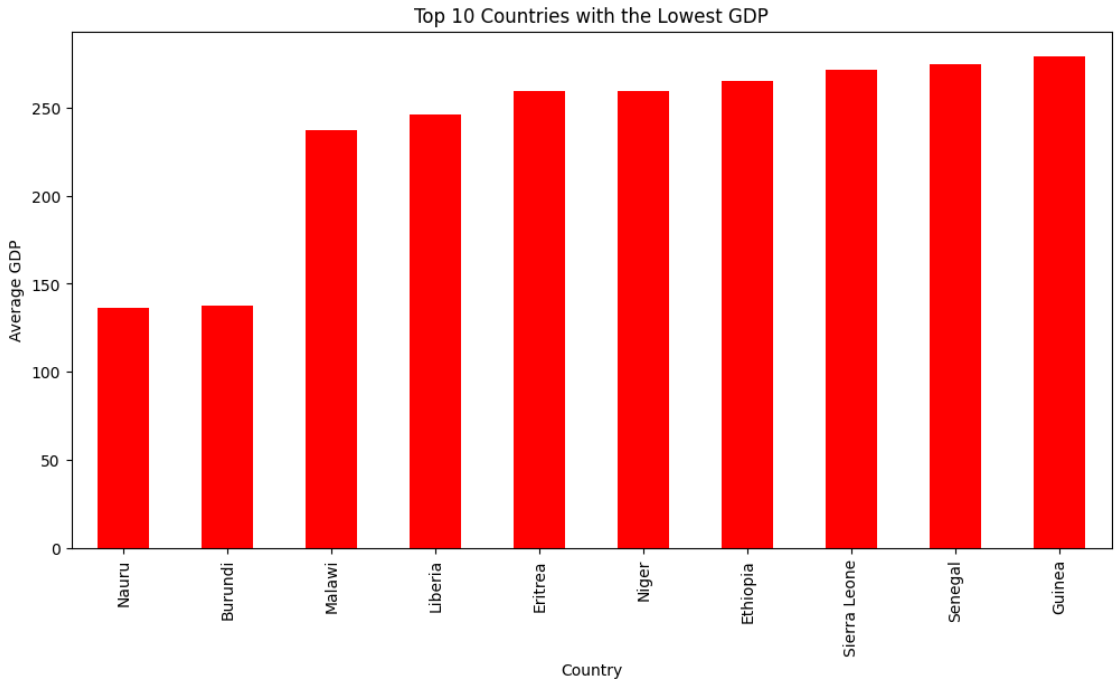
5) What are the top 10 countries with the highest and lowest GDP?

[16]:
```
highest_GDP=df.groupby('Country')['GDP'].mean().sort_values(ascending=False)
print("Highest GDP TOP 10 Countries : ")
print(highest_GDP.head(10), end="\n\n")



lowest_GDP=df.groupby('Country')['GDP'].mean().sort_values(ascending=True)
print("Lowest GDP TOP 10 Countries : ")
print(lowest_GDP.head(10))


plt.figure(figsize=(12, 6))
highest_GDP.head(10).plot(kind='bar', color='green')
plt.title('Top 10 Countries with the Highest GDP')
plt.xlabel('Country')
plt.ylabel('Average GDP')
plt.show()


plt.figure(figsize=(12, 6))
lowest_GDP.head(10).plot(kind='bar', color='red')
plt.title('Top 10 Countries with the Lowest GDP')
plt.xlabel('Country')
plt.ylabel('Average GDP')
plt.show()
```

```
Highest GDP TOP 10 Countries :
Country
Switzerland    57362.874601
Luxembourg     53257.012741
Qatar          40748.444104
Netherlands    34964.719797
Australia      34637.565047
Ireland        33835.272005
Austria        33827.476309
Denmark        33067.407916
Singapore      32790.105907
Kuwait         31914.378339
Name: GDP, dtype: float64

Lowest GDP TOP 10 Countries :
Country
Nauru           136.183210
Burundi         137.815321
Malawi          237.504042
Liberia         246.281748
Eritrea         259.395356
Niger           259.782441
```

```
Ethiopia         264.970950
Sierra Leone     271.505561
Senegal          274.611166
Guinea           279.464798
Name: GDP, dtype: float64
```



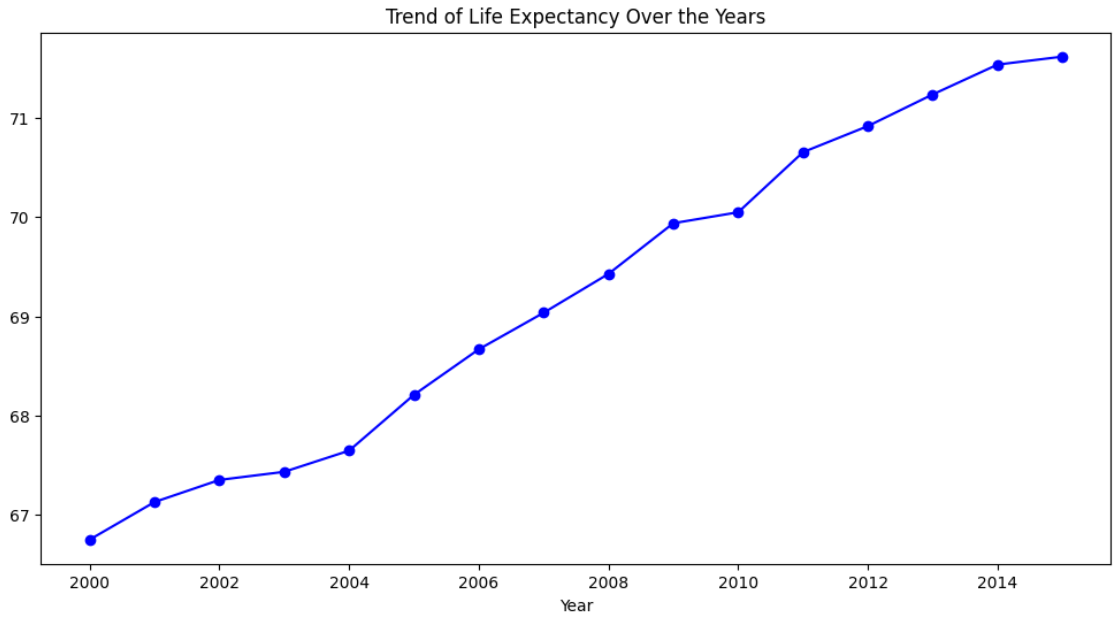


6) What is the trend of life expectancy over the years for different regions?

[17]:
```
trend_life_expectancy = df.groupby('Year')['Life expectancy '].mean()
print(trend_life_expectancy)

trend_life_expectancy.plot(kind='line', figsize=(12, 6), marker='o',␣
 ↪color='blue')
plt.title('Trend of Life Expectancy Over the Years')
```

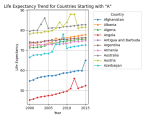
```
Year
2000    66.750273
2001    67.128962
2002    67.351366
2003    67.433333
2004    67.646448
2005    68.209290
2006    68.667760
2007    69.036066
2008    69.427869
2009    69.938251
2010    70.048634
2011    70.654098
2012    70.916940
2013    71.236066
2014    71.536612
2015    71.616940
Name: Life expectancy , dtype: float64
```

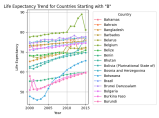
[17]:
```
Text(0.5, 1.0, 'Trend of Life Expectancy Over the Years')
```

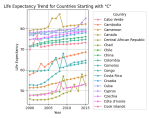


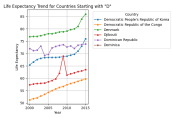
[19]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import string
import math

# Grouping the data by Year and Country, then calculating the mean life␣
 ↪expectancy
region_life_expectancy = df.groupby(['Year', 'Country'])['Life expectancy '].
 ↪mean().reset_index()

# Create a plot for each group of countries starting with the same letter
# Calculate the number of rows required (2 columns per row)
num_plots = len(string.ascii_uppercase)  # One plot for each letter A-Z
num_rows = math.ceil(num_plots / 2)  # Two plots per row

# Create subplots (dynamic number of rows and 2 columns)
fig, axes = plt.subplots(num_rows, 2, figsize=(15, num_rows * 6))  # Adjust␣
 ↪height based on number of rows
axes = axes.flatten()  # Flatten to make indexing easier

# Iterate over each letter in the alphabet and plot the life expectancy trends
for i, letter in enumerate(string.ascii_uppercase):
    # Filter countries starting with the current letter
    countries_starting_with_letter =␣
 ↪region_life_expectancy[region_life_expectancy['Country'].str.
 ↪startswith(letter)]
```

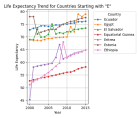
```
    # If there are countries starting with this letter, plot them
    if not countries_starting_with_letter.empty:
        ax = axes[i]  # Select the subplot for the current plot
        sns.lineplot(x='Year', y='Life expectancy ', hue='Country',␣
↪data=countries_starting_with_letter, marker='o', ax=ax)
        ax.set_title(f'Life Expectancy Trend for Countries Starting with␣
↪"{letter}"')
        ax.set_xlabel('Year')
        ax.set_ylabel('Life Expectancy')
        ax.grid(True)
        ax.legend(title='Country', bbox_to_anchor=(1.05, 1), loc='upper left')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```


Life Expectancy Trend for Countries Starting with "A"
Life Expectancy
Year
Country
Afghanistan
Albania
Algeria
Angola
Antigua and Barbuda
Argentina
Armenia
Australia
Austria
Azerbaijan

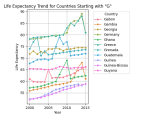

Life Expectancy Trend for Countries Starting with "B"
Life Expectancy
Year
Country
Bahamas
Bahrain
Bangladesh
Barbados
Belarus
Belgium
Belize
Benin
Bhutan
Bolivia (Plurinational State of)
Bosnia and Herzegovina
Botswana
Brazil
Brunei Darussalam
Bulgaria
Burkina Faso
Burundi


Life Expectancy Trend for Countries Starting with "C"
Life Expectancy
Year
Country
Cabo Verde
Cambodia
Cameroon
Canada
Central African Republic
Chad
Chile
China
Colombia
Comoros
Congo
Costa Rica
Croatia
Cuba
Cyprus
Czechia
Côte d'Ivoire
Cook Islands


Life Expectancy Trend for Countries Starting with "D"
Life Expectancy
Year
Country
Democratic People's Republic of Korea
Democratic Republic of the Congo
Denmark
Djibouti
Dominican Republic
Dominica

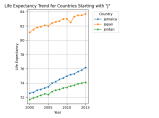

Life Expectancy Trend for Countries Starting with "E"
Life Expectancy
Year
Country
Ecuador
Egypt
El Salvador
Equatorial Guinea
Eritrea
Estonia
Ethiopia


Life Expectancy Trend for Countries Starting with "F"
Life Expectancy
Year
Country
Fiji
Finland
France


Life Expectancy Trend for Countries Starting with "G"
Life Expectancy
Year
Country
Gabon
Gambia
Georgia
Germany
Ghana
Greece
Grenada
Guatemala
Guinea
Guinea-Bissau
Guyana


Life Expectancy Trend for Countries Starting with "H"
Life Expectancy
Year
Country
Haiti
Honduras
Hungary


Life Expectancy Trend for Countries Starting with "I"
Life Expectancy
Year
Country
Iceland
India
Indonesia
Iran (Islamic Republic of)
Iraq
Ireland
Israel
Italy

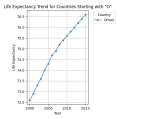

Life Expectancy Trend for Countries Starting with "J"
Life Expectancy
Year
Country
Jamaica
Japan
Jordan


Life Expectancy Trend for Countries Starting with "K"
Life Expectancy
Year
Country
Kazakhstan
Kenya
Kiribati
Kuwait
Kyrgyzstan

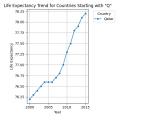

Life Expectancy Trend for Countries Starting with "L"
Life Expectancy
Year
Country
Lao People's Democratic Republic
Latvia
Lebanon
Lesotho
Liberia
Libya
Lithuania
Luxembourg


Life Expectancy Trend for Countries Starting with "M"
Life Expectancy
Year
Country
Madagascar
Malawi
Malaysia
Maldives
Mali
Malta
Mauritania
Mauritius
Mexico
Micronesia (Federated States of)
Mongolia
Montenegro
Morocco
Mozambique
Myanmar
Marshall Islands
Monaco


Life Expectancy Trend for Countries Starting with "N"
Life Expectancy
Year
Country
Namibia
Nepal
Netherlands
New Zealand
Nicaragua
Niger
Nigeria
Norway
Nauru
Niue

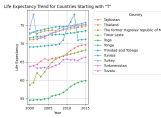

Life Expectancy Trend for Countries Starting with "O"
Life Expectancy
Year
Country
Oman

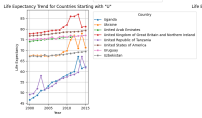

Life Expectancy Trend for Countries Starting with "P"
Life Expectancy
Year
Country
Pakistan
Panama
Papua New Guinea
Paraguay
Peru
Philippines
Poland
Portugal
Palau

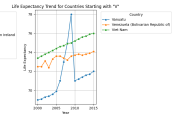

Life Expectancy Trend for Countries Starting with "Q"
Life Expectancy
Year
Country
Qatar

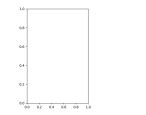

Life Expectancy Trend for Countries Starting with "R"
Life Expectancy
Year
Country
Republic of Korea
Republic of Moldova
Romania
Russian Federation
Rwanda

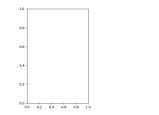

Life Expectancy Trend for Countries Starting with "S"
Life Expectancy
Year
Country
Saint Kitts and Nevis
Saint Vincent and the Grenadines
Samoa
Sao Tome and Principe
Saudi Arabia
Senegal
Serbia
Seychelles
Sierra Leone
Singapore
Slovakia
Slovenia
Solomon Islands
Somalia
South Africa
South Sudan
Spain
Sri Lanka
Sudan
Suriname
Swaziland
Sweden
Switzerland
Syrian Arab Republic
San Marino

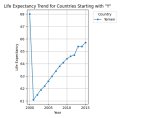

Life Expectancy Trend for Countries Starting with "T"
Life Expectancy
Year
Country
Tajikistan
Thailand
The former Yugoslav republic of Macedonia
Timor-Leste
Togo
Tonga
Trinidad and Tobago
Tunisia
Turkey
Turkmenistan
Tuvalu

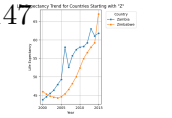

Life Expectancy Trend for Countries Starting with "U"
Life Expectancy
Year
Country
Uganda
Ukraine
United Arab Emirates
United Kingdom of Great Britain and Northern Ireland
United Republic of Tanzania
United States of America
Uruguay
Uzbekistan


Life Expectancy Trend for Countries Starting with "V"
Life Expectancy
Year
Country
Vanuatu
Venezuela (Bolivarian Republic of)
Viet Nam


0.0
0.2
0.4
0.6
0.8
1.0


0.0
0.2
0.4
0.6
0.8
1.0

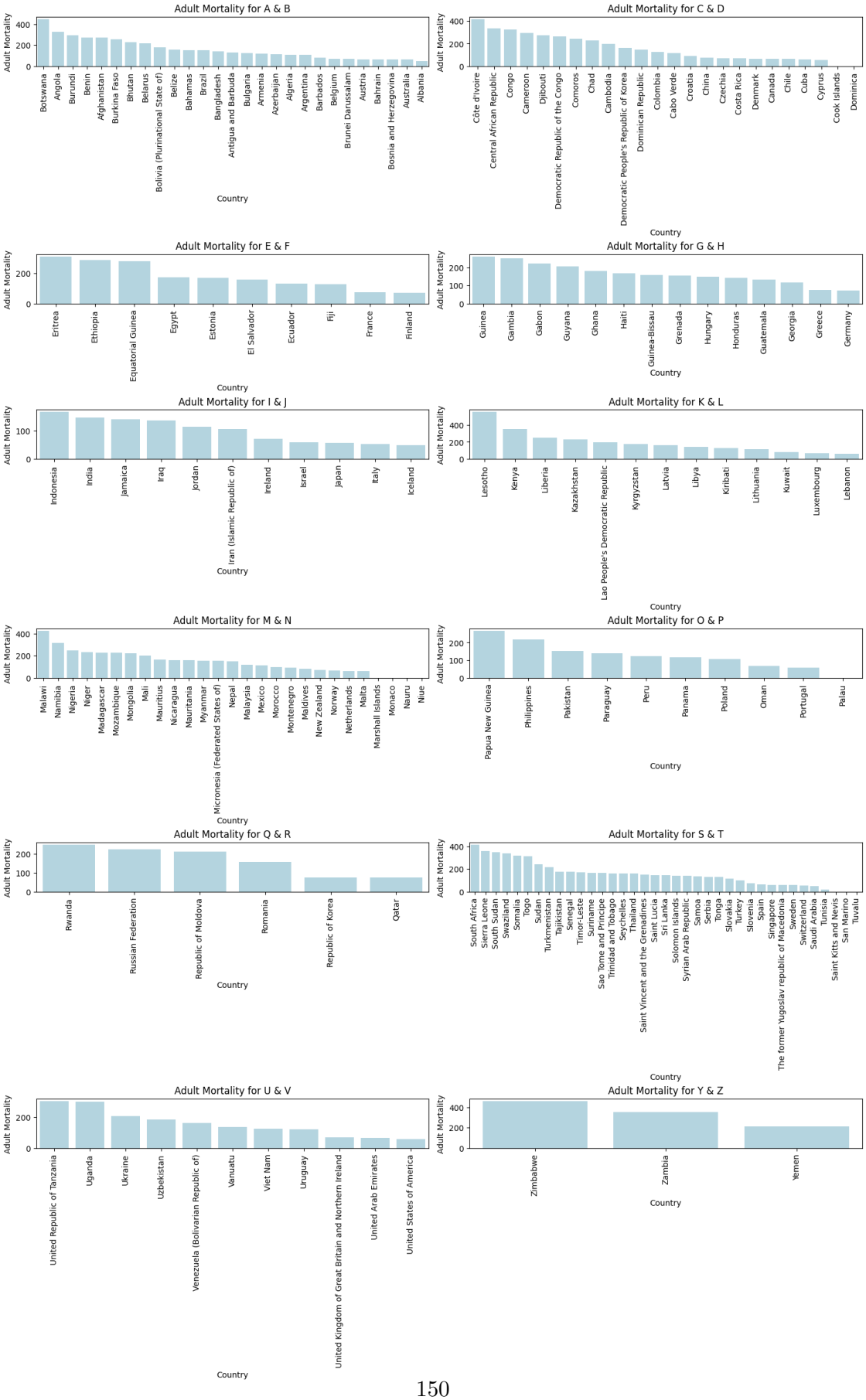

Life Expectancy Trend for Countries Starting with "Y"
Life Expectancy
Year
Country
Yemen


Life Expectancy Trend for Countries Starting with "Z"
Life Expectancy
Year
Country
Zambia
Zimbabwe

7) How does adult mortality impact life expectancy across countries?

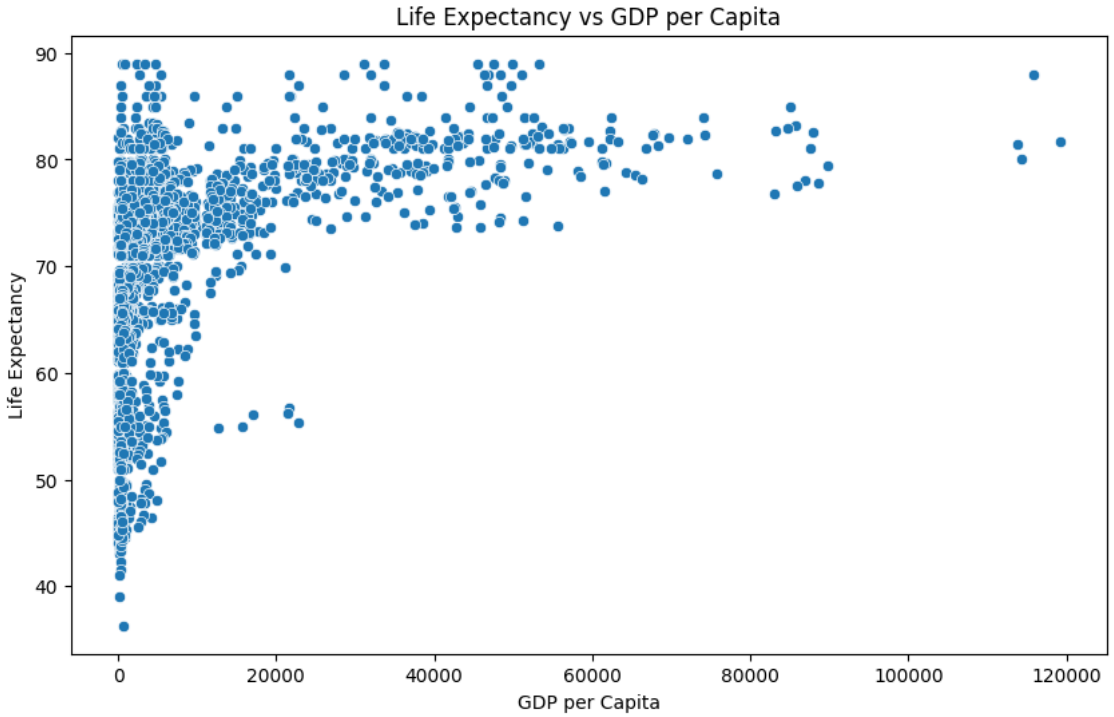
[20]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import string
import math

# Sample DataFrame (Replace with actual dataset)
# df = pd.read_csv("your_data.csv")

# Grouping data by country and calculating the mean adult mortality
adult_mortality_impact = df.groupby('Country')['Adult Mortality'].mean().
 ↪sort_values(ascending=False)

# Split countries into pairs of letters (A-B, C-D, etc.)
country_groups = [(string.ascii_uppercase[i], string.ascii_uppercase[i+1]) for␣
 ↪i in range(0, len(string.ascii_uppercase), 2)]

# Filter only groups that exist in data
valid_groups = [group for group in country_groups if any(adult_mortality_impact.
 ↪index.str.startswith(tuple(group)))]

# Calculate the number of rows needed
num_rows = math.ceil(len(valid_groups) / 2)

# Create subplots with adjustable layout
fig, axes = plt.subplots(num_rows, 2, figsize=(15, num_rows * 4),␣
 ↪constrained_layout=True)
axes = axes.flatten()

# Create a plot for each group of countries
for i, group in enumerate(valid_groups):
    # Filter countries that start with the current pair of letters
    filtered_countries = adult_mortality_impact[adult_mortality_impact.index.
 ↪str.startswith(tuple(group))]

    if not filtered_countries.empty:
        sns.barplot(x=filtered_countries.index, y=filtered_countries.values,␣
 ↪color='lightblue', ax=axes[i])
        axes[i].set_title(f'Adult Mortality for {group[0]} & {group[1]}')
        axes[i].set_xlabel('Country')
        axes[i].set_ylabel('Adult Mortality')
        axes[i].tick_params(axis='x', rotation=90)
```

```
# Hide unused subplots
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.show()
```


Adult Mortality for A & B
Adult Mortality
Botswana
Angola
Burundi
Benin
Afghanistan
Burkina Faso
Bhutan
Belarus
Bolivia (Plurinational State of)
Belize
Bahamas
Brazil
Bangladesh
Antigua and Barbuda
Bulgaria
Armenia
Azerbaijan
Algeria
Argentina
Barbados
Belgium
Brunei Darussalam
Austria
Bahrain
Bosnia and Herzegovina
Australia
Albania
Country
Adult Mortality for C & D
Adult Mortality
Côte d'Ivoire
Central African Republic
Congo
Cameroon
Djibouti
Democratic Republic of the Congo
Comoros
Chad
Cambodia
Democratic People's Republic of Korea
Dominican Republic
Colombia
Cabo Verde
Croatia
China
Czechia
Costa Rica
Denmark
Canada
Chile
Cuba
Cyprus
Cook Islands
Dominica
Country
Adult Mortality for E & F
Adult Mortality
Eritrea
Ethiopia
Equatorial Guinea
Egypt
Estonia
El Salvador
Ecuador
Fiji
France
Finland
Country
Adult Mortality for G & H
Adult Mortality
Guinea
Gambia
Gabon
Guyana
Ghana
Haiti
Guinea-Bissau
Grenada
Hungary
Honduras
Guatemala
Georgia
Greece
Germany
Country
Adult Mortality for I & J
Adult Mortality
Indonesia
India
Jamaica
Iraq
Jordan
Iran (Islamic Republic of)
Ireland
Israel
Japan
Italy
Iceland
Country
Adult Mortality for K & L
Adult Mortality
Lesotho
Kenya
Liberia
Kazakhstan
Lao People's Democratic Republic
Kyrgyzstan
Latvia
Libya
Kiribati
Lithuania
Kuwait
Luxembourg
Lebanon
Country
Adult Mortality for M & N
Adult Mortality
Malawi
Namibia
Nigeria
Niger
Madagascar
Mozambique
Mongolia
Mali
Mauritius
Nicaragua
Mauritania
Myanmar
Micronesia (Federated States of)
Nepal
Malaysia
Mexico
Morocco
Montenegro
Maldives
New Zealand
Norway
Netherlands
Malta
Marshall Islands
Monaco
Nauru
Niue
Country
Adult Mortality for O & P
Adult Mortality
Papua New Guinea
Philippines
Pakistan
Paraguay
Peru
Panama
Poland
Oman
Portugal
Palau
Country
Adult Mortality for Q & R
Adult Mortality
Rwanda
Russian Federation
Republic of Moldova
Romania
Republic of Korea
Qatar
Country
Adult Mortality for S & T
Adult Mortality
South Africa
Sierra Leone
South Sudan
Swaziland
Somalia
Togo
Sudan
Turkmenistan
Tajikistan
Senegal
Timor-Leste
Suriname
Sao Tome and Principe
Trinidad and Tobago
Seychelles
Thailand
Saint Vincent and the Grenadines
Saint Lucia
Sri Lanka
Solomon Islands
Syrian Arab Republic
Samoa
Serbia
Tonga
Slovakia
Turkey
Slovenia
Spain
Singapore
The former Yugoslav republic of Macedonia
Sweden
Switzerland
Saudi Arabia
Tunisia
Saint Kitts and Nevis
San Marino
Tuvalu
Country
Adult Mortality for U & V
Adult Mortality
United Republic of Tanzania
Uganda
Ukraine
Uzbekistan
Venezuela (Bolivarian Republic of)
Vanuatu
Viet Nam
Uruguay
United Kingdom of Great Britain and Northern Ireland
United Arab Emirates
United States of America
Country
Adult Mortality for Y & Z
Adult Mortality
Zimbabwe
Zambia
Yemen
Country

8) Is there a significant relationship between life expectancy and GDP per capita?

[21]:
```
# Scatter plot of Life Expectancy vs GDP
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='GDP', y='Life expectancy ')
plt.title('Life Expectancy vs GDP per Capita')
plt.xlabel('GDP per Capita')
plt.ylabel('Life Expectancy')
plt.show()

# Calculate the correlation coefficient
correlation = df['GDP'].corr(df['Life expectancy '])
print(f'Correlation coefficient between Life Expectancy and GDP per Capita:␣
↪{correlation}')
```

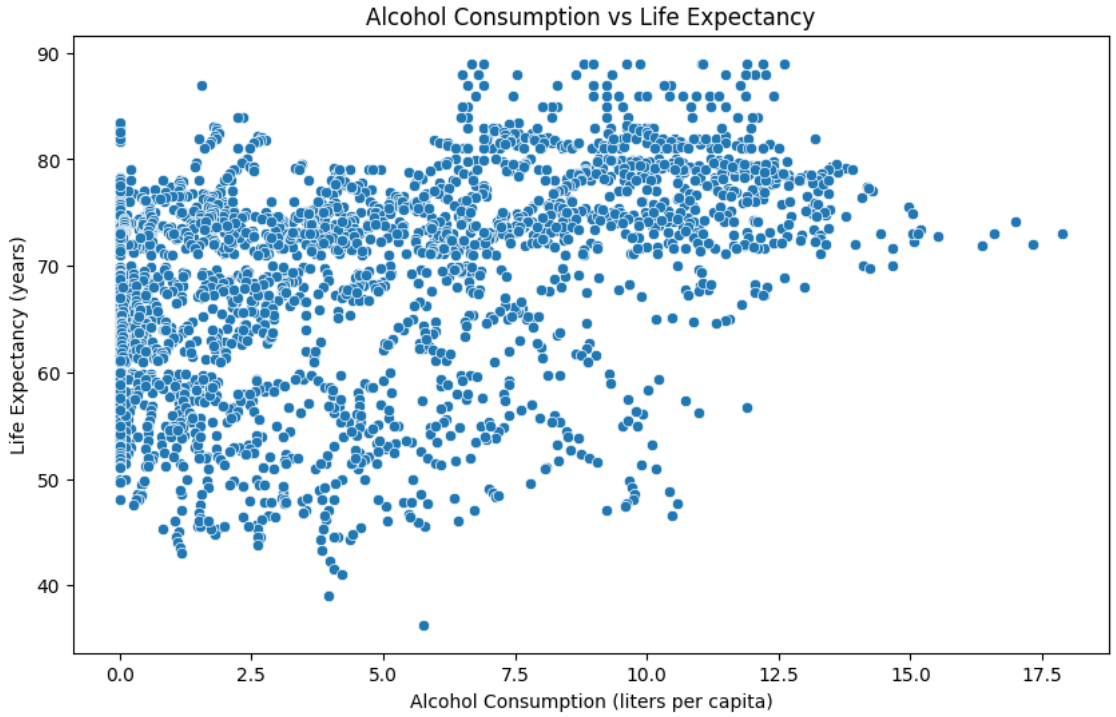



```
Correlation coefficient between Life Expectancy and GDP per Capita:
0.4614551926207382
```

9) How does alcohol consumption relate to life expectancy?

[22]:
```
# Scatter plot of Alcohol consumption vs Life expectancy
plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(data=df, x='Alcohol', y='Life expectancy ')
plt.title('Alcohol Consumption vs Life Expectancy')
plt.xlabel('Alcohol Consumption (liters per capita)')
plt.ylabel('Life Expectancy (years)')
plt.show()

# Calculate the correlation between Alcohol consumption and Life expectancy
correlation_alcohol_life_expectancy = df['Alcohol'].corr(df['Life expectancy '])
print(f'Correlation between Alcohol Consumption and Life Expectancy:␣
  ↪{correlation_alcohol_life_expectancy}')
```

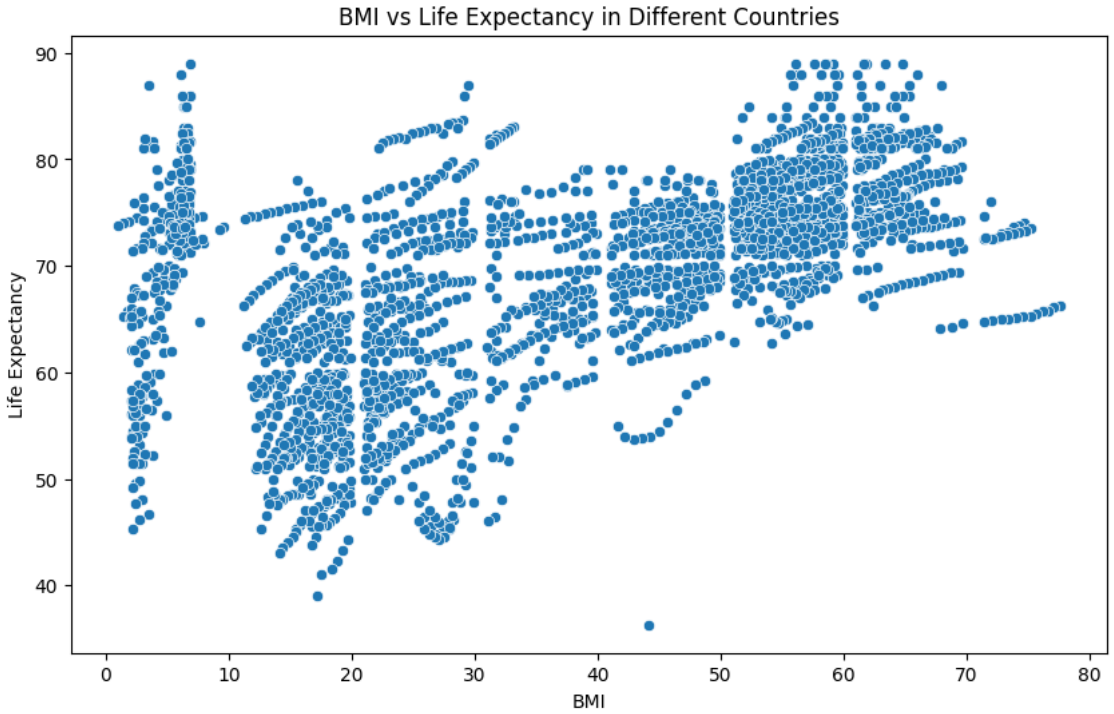



```
Correlation between Alcohol Consumption and Life Expectancy: 0.4048767611266021
```

10) What is the impact of BMI on life expectancy in different countries?

[23]:
```
# Scatter plot of BMI vs Life expectancy
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x=' BMI ', y='Life expectancy ')
plt.title('BMI vs Life Expectancy in Different Countries')
plt.xlabel('BMI')
plt.ylabel('Life Expectancy')
plt.show()

# Calculate the correlation between BMI and Life expectancy
```

```
correlation_bmi_life_expectancy = df[' BMI '].corr(df['Life expectancy '])
print(f'Correlation between BMI and Life Expectancy:␣
  ↪{correlation_bmi_life_expectancy}')
```

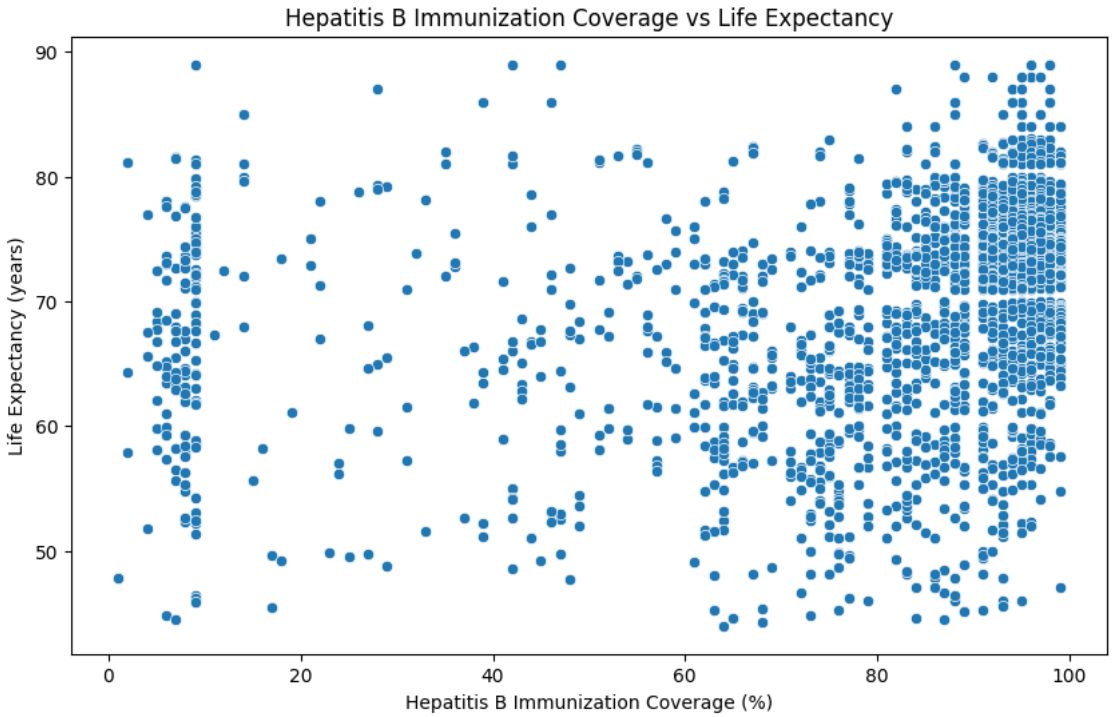



```
Correlation between BMI and Life Expectancy: 0.5676935475459862
```

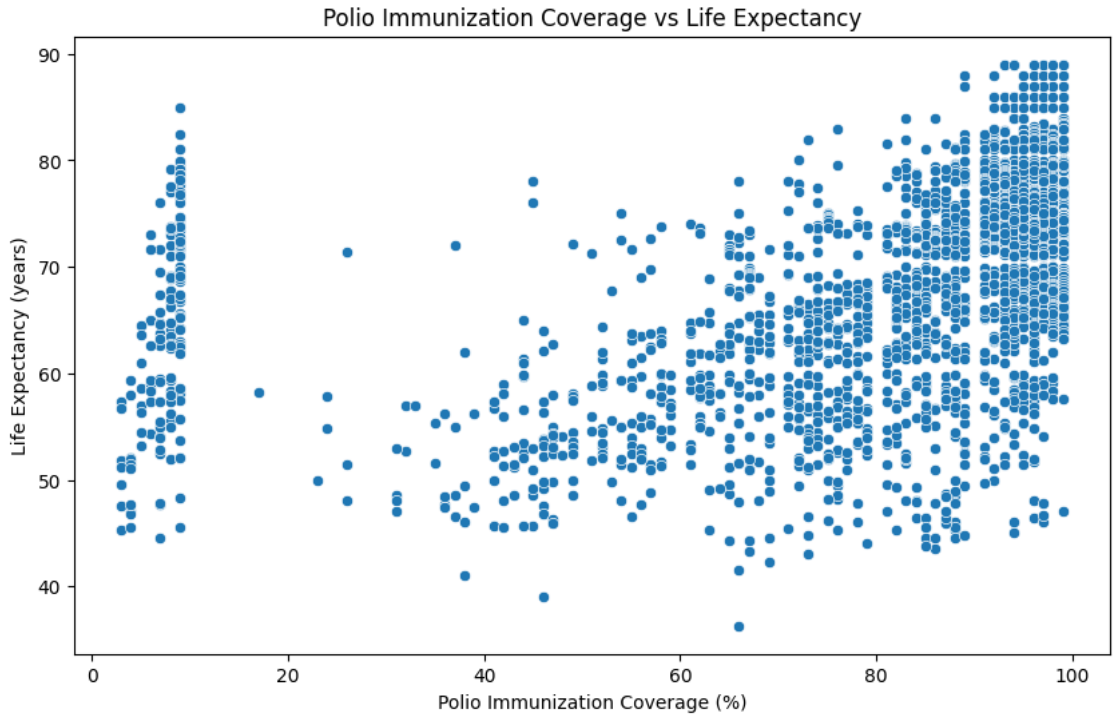
11) Does immunization coverage (Hepatitis B, Polio) affect life expectancy?

[24]:
```
# Scatter plot of Hepatitis B immunization coverage vs Life expectancy
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Hepatitis B', y='Life expectancy ')
plt.title('Hepatitis B Immunization Coverage vs Life Expectancy')
plt.xlabel('Hepatitis B Immunization Coverage (%)')
plt.ylabel('Life Expectancy (years)')
plt.show()

# Calculate the correlation between Hepatitis B immunization coverage and Life␣
  ↪expectancy
correlation_hepatitisB_life_expectancy = df['Hepatitis B'].corr(df['Life␣
  ↪expectancy '])
print(f'Correlation between Hepatitis B Immunization Coverage and Life␣
  ↪Expectancy: {correlation_hepatitisB_life_expectancy}')

# Scatter plot of Polio immunization coverage vs Life expectancy
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Polio', y='Life expectancy ')
plt.title('Polio Immunization Coverage vs Life Expectancy')
plt.xlabel('Polio Immunization Coverage (%)')
plt.ylabel('Life Expectancy (years)')
plt.show()

# Calculate the correlation between Polio immunization coverage and Life␣
 ↪expectancy
correlation_polio_life_expectancy = df['Polio'].corr(df['Life expectancy '])
print(f'Correlation between Polio Immunization Coverage and Life Expectancy:␣
 ↪{correlation_polio_life_expectancy}')
```



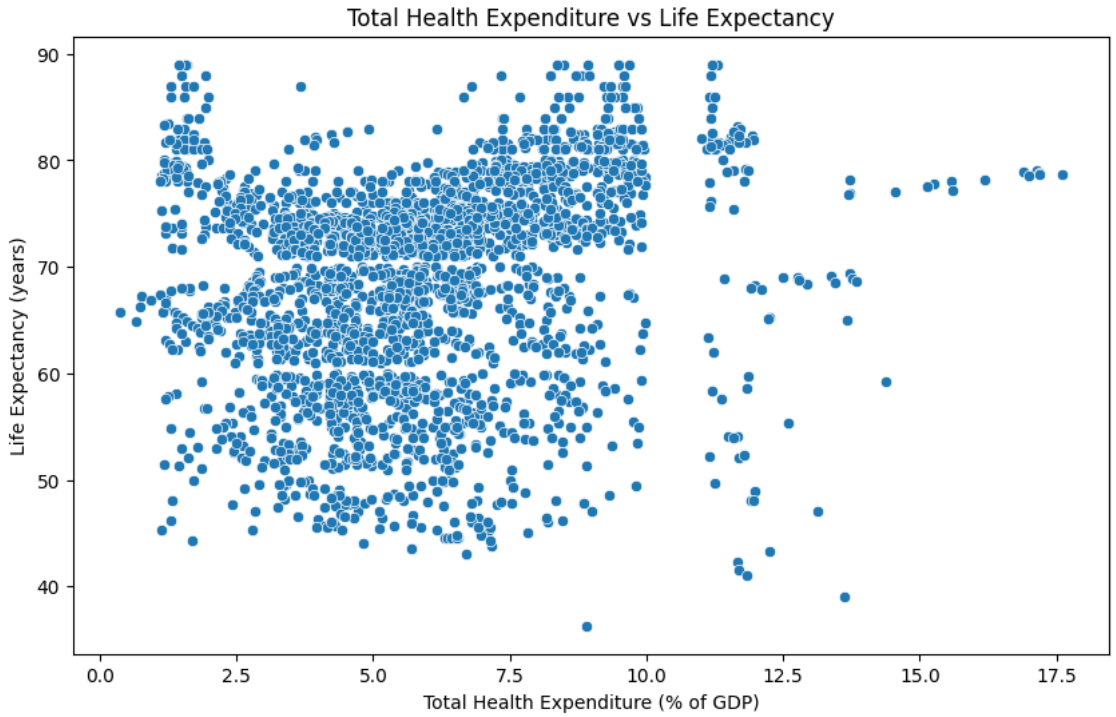

```
Correlation between Hepatitis B Immunization Coverage and Life Expectancy:
0.25676194760492443
```




```
Correlation between Polio Immunization Coverage and Life Expectancy:
0.4655558059771988
```

12) What is the effect of total health expenditure on life expectancy?

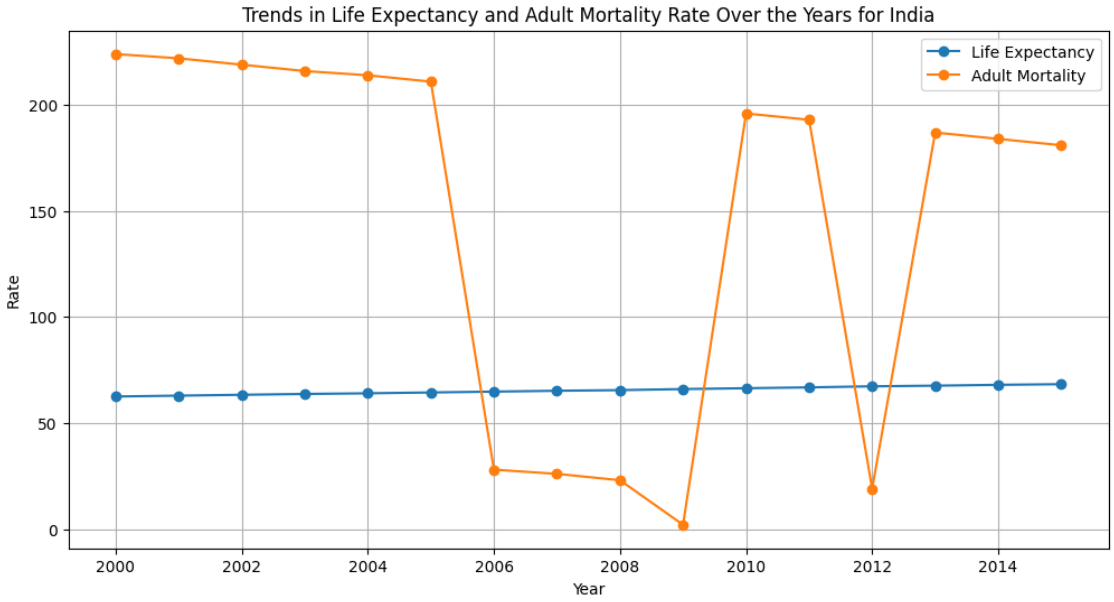
[25]:
```
# Scatter plot of Total Health Expenditure vs Life Expectancy
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Total expenditure', y='Life expectancy ')
plt.title('Total Health Expenditure vs Life Expectancy')
plt.xlabel('Total Health Expenditure (% of GDP)')
plt.ylabel('Life Expectancy (years)')
plt.show()


correlation_total_expenditure_life_expectancy = df['Total expenditure'].
 ↪corr(df['Life expectancy '])
print(f'Correlation between Total Health Expenditure and Life Expectancy:␣
 ↪{correlation_total_expenditure_life_expectancy}')
```




```
Correlation between Total Health Expenditure and Life Expectancy:
0.2180863736916713
```

13) Plot trends in life expectancy and mortality rate over the years for India.

[26]:
```
# Filter data for India
india_data = df[df['Country'] == 'India']

# Plotting Life Expectancy trend
plt.figure(figsize=(12, 6))
plt.plot(india_data['Year'], india_data['Life expectancy '], marker='o',␣
 ↪label='Life Expectancy')
plt.plot(india_data['Year'], india_data['Adult Mortality'], marker='o',␣
 ↪label='Adult Mortality')
plt.title('Trends in Life Expectancy and Adult Mortality Rate Over the Years␣
 ↪for India')
plt.xlabel('Year')
plt.ylabel('Rate')
plt.legend()
plt.grid(True)
plt.show()
```



13.1 Conlcusion

13.2 Conclusion

In this analysis, we explored various aspects of the Life Expectancy dataset to uncover insights and relationships between different factors and life expectancy across countries. Here are the key observations:

1. **Data Overview**: We examined the columns and data types, and identified missing values in the dataset.
2. **Categorical Analysis**: We explored unique values for categorical columns such as countries and status, and analyzed the distribution of life expectancy across different countries.
3. **Correlation Analysis**: We investigated the correlation between life expectancy and other numerical features, identifying significant relationships.
4. **GDP Analysis**: We identified the top 10 countries with the highest and lowest GDP, and visualized these findings.
5. **Trend Analysis**: We analyzed the trend of life expectancy over the years for different regions, providing insights into regional improvements or declines.
6. **Impact of Adult Mortality**: We examined how adult mortality impacts life expectancy across countries, highlighting significant differences.
7. **Life Expectancy and GDP**: We found a positive correlation between life expectancy and GDP per capita, indicating that higher economic prosperity is associated with longer life expectancy.
8. **Alcohol Consumption**: We observed the relationship between alcohol consumption and life expectancy, finding a negative correlation.
9. **BMI Impact**: We analyzed the impact of BMI on life expectancy, revealing a moderate correlation.
10. **Immunization Coverage**: We explored the effect of immunization coverage (Hepatitis B,

Polio) on life expectancy, finding positive correlations.

11. **Health Expenditure**: We concluded that higher total health expenditure is positively correlated with increased life expectancy.

These insights provide a comprehensive understanding of the factors influencing life expectancy and can guide policymakers and researchers in making informed decisions to improve public health outcomes globally.

from file: PMRP_9

14 *Merged Jupyter Notebook*

from file: 23AIML010_PR-09_CLEANING

15 23AIML010 OM CHOKSI PMRP ASSIGNMENT 9 + CLASSWORK

This is housing dataset, Find trends in this data. What is price fluctuation according to state? Find any outliers in data for each column. Generate 5-7 conclusive insights on this dataset.

`Housing.csv`

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
```

[2]:
```
data=pd.read_csv('Housing.csv')
data
```

[2]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	\
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	
..	…	…	…	…	…	…	…		
540	1820000	3000	2	1	1	yes	no	yes	
541	1767150	2400	3	1	1	no	no	no	
542	1750000	3620	2	1	1	yes	no	no	
543	1750000	2910	3	1	1	no	no	no	
544	1750000	3850	3	1	2	yes	no	no	

	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	no	yes	2	yes	furnished
1	no	yes	3	no	furnished
2	no	no	2	yes	semi-furnished

```
3                no             yes          3      yes        furnished
4                no             yes          2       no        furnished
..              ...             ...        ...      ...              ...
540              no              no          2       no      unfurnished
541              no              no          0       no   semi-furnished
542              no              no          0       no      unfurnished
543              no              no          0       no        furnished
544              no              no          0       no      unfurnished

[545 rows x 13 columns]
```

[3]:
```
data.describe(),data.drop_duplicates(),data.dropna,data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             545 non-null    int64
 1   area              545 non-null    int64
 2   bedrooms          545 non-null    int64
 3   bathrooms         545 non-null    int64
 4   stories           545 non-null    int64
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

[3]:
```
(              price          area    bedrooms   bathrooms     stories  \
 count  5.450000e+02    545.000000  545.000000  545.000000  545.000000
 mean   4.766729e+06   5150.541284    2.965138    1.286239    1.805505
 std    1.870440e+06   2170.141023    0.738064    0.502470    0.867492
 min    1.750000e+06   1650.000000    1.000000    1.000000    1.000000
 25%    3.430000e+06   3600.000000    2.000000    1.000000    1.000000
 50%    4.340000e+06   4600.000000    3.000000    1.000000    2.000000
 75%    5.740000e+06   6360.000000    3.000000    2.000000    2.000000
 max    1.330000e+07  16200.000000    6.000000    4.000000    4.000000

            parking
 count  545.000000
 mean     0.693578
 std      0.861586
```

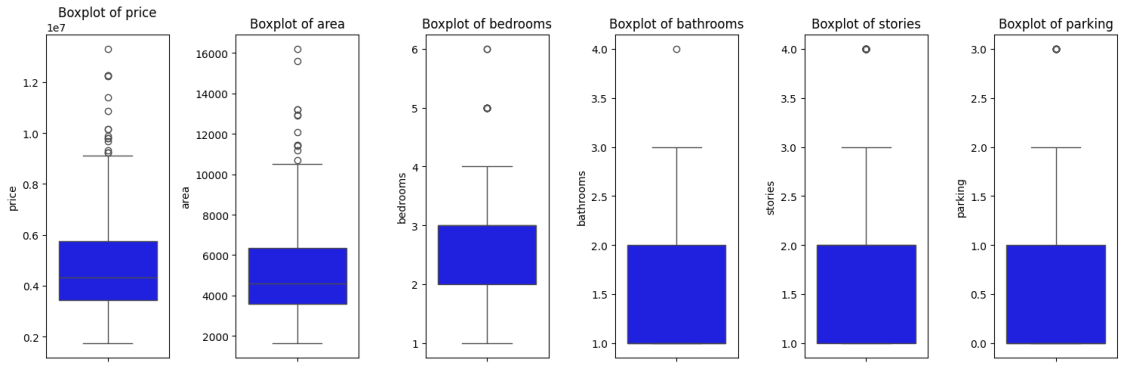
```
 min      0.000000
 25%      0.000000
 50%      0.000000
 75%      1.000000
 max      3.000000  ,
        price  area  bedrooms  bathrooms  stories mainroad guestroom basement
\
 0    13300000  7420         4          2        3      yes        no       no
 1    12250000  8960         4          4        4      yes        no       no
 2    12250000  9960         3          2        2      yes        no      yes
 3    12215000  7500         4          2        2      yes        no      yes
 4    11410000  7420         4          1        2      yes       yes      yes
 ..        …   …        …         …       …      …       …       …
 540   1820000  3000         2          1        1      yes        no      yes
 541   1767150  2400         3          1        1       no        no       no
 542   1750000  3620         2          1        1      yes        no       no
 543   1750000  2910         3          1        1       no        no       no
 544   1750000  3850         3          1        2      yes        no       no

     hotwaterheating airconditioning  parking prefarea furnishingstatus
 0                no             yes        2      yes        furnished
 1                no             yes        3       no        furnished
 2                no              no        2      yes   semi-furnished
 3                no             yes        3      yes        furnished
 4                no             yes        2       no        furnished
 ..              …              …      …      …              …
 540              no              no        2       no      unfurnished
 541              no              no        0       no   semi-furnished
 542              no              no        0       no      unfurnished
 543              no              no        0       no        furnished
 544              no              no        0       no      unfurnished

 [545 rows x 13 columns],
 <bound method DataFrame.dropna of         price  area  bedrooms  bathrooms
stories mainroad guestroom basement  \
 0    13300000  7420         4          2        3      yes        no       no
 1    12250000  8960         4          4        4      yes        no       no
 2    12250000  9960         3          2        2      yes        no      yes
 3    12215000  7500         4          2        2      yes        no      yes
 4    11410000  7420         4          1        2      yes       yes      yes
 ..        …   …        …         …       …      …       …       …
 540   1820000  3000         2          1        1      yes        no      yes
 541   1767150  2400         3          1        1       no        no       no
 542   1750000  3620         2          1        1      yes        no       no
 543   1750000  2910         3          1        1       no        no       no
 544   1750000  3850         3          1        2      yes        no       no
```

```
    hotwaterheating airconditioning  parking prefarea furnishingstatus
0                no              yes        2      yes        furnished
1                no              yes        3       no        furnished
2                no               no        2      yes   semi-furnished
3                no              yes        3      yes        furnished
4                no              yes        2       no        furnished
..              ...              ...      ...      ...              ...
540              no               no        2       no      unfurnished
541              no               no        0       no   semi-furnished
542              no               no        0       no      unfurnished
543              no               no        0       no        furnished
544              no               no        0       no      unfurnished

[545 rows x 13 columns]>,
None)
```

The code above reads the housing dataset from a CSV file and displays the first few rows of the

[4]:
```
df = pd.read_csv("Housing.csv")

print(df.head())

plt.figure(figsize=(15, 5))

for i, column in enumerate(df.select_dtypes(include=["number"]).columns, 1):
    plt.subplot(1, len(df.select_dtypes(include=["number"]).columns), i)
    sns.boxplot(y=df[column], color="blue")
    plt.title(f"Boxplot of {column}")
    plt.ylabel(column)

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

```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no              yes        2      yes        furnished
1              no              yes        3       no        furnished
2              no               no        2      yes   semi-furnished
3              no              yes        3      yes        furnished
4              no              yes        2       no        furnished
```



The code above reads the housing dataset from a CSV file and defines a function `remove_outliers_iqr` to remove outliers using the Interquartile Range (IQR) method. The function iterates through each numerical column in the dataframe, calculates the IQR, and filters out rows that fall outside the lower and upper bounds. The cleaned dataset is then saved to a new CSV file named `cleaned_dataset.csv`. The shapes of the original and cleaned datasets are printed, along with the first few rows of the cleaned dataset. ‘‘‘

[5]:
```
df = pd.read_csv("Housing.csv")

def remove_outliers_iqr(df):
    cleaned_df = df.copy()
    for column in df.select_dtypes(include=["number"]).columns:
        Q1 = df[column].quantile(0.25)
        Q3 = df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        cleaned_df = cleaned_df[(cleaned_df[column] >= lower_bound) &␣
 ↪(cleaned_df[column] <= upper_bound)]

    return cleaned_df
df_cleaned = remove_outliers_iqr(df)

print(f"Original dataset shape: {df.shape}")
print(f"Cleaned dataset shape: {df_cleaned.shape}")

df_cleaned.to_csv("cleaned_dataset.csv", index=False)

print(df_cleaned.head())
```

```
Original dataset shape: (545, 13)
Cleaned dataset shape: (463, 13)
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
```

```
15  9100000  6000         4          1        2      yes        no      yes
16  9100000  6600         4          2        2      yes       yes      yes
18  8890000  4600         3          2        2      yes       yes       no
19  8855000  6420         3          2        2      yes        no       no
20  8750000  4320         3          1        2      yes        no      yes

   hotwaterheating airconditioning  parking prefarea furnishingstatus
15              no              no        2       no   semi-furnished
16              no             yes        1      yes      unfurnished
18              no             yes        2       no        furnished
19              no             yes        1      yes   semi-furnished
20             yes              no        2       no   semi-furnished
```

[6]:
```
file_path = "cleaned_dataset.csv"
df = pd.read_csv(file_path)

df.info()
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 463 entries, 0 to 462
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   price             463 non-null    int64
 1   area              463 non-null    int64
 2   bedrooms          463 non-null    int64
 3   bathrooms         463 non-null    int64
 4   stories           463 non-null    int64
 5   mainroad          463 non-null    object
 6   guestroom         463 non-null    object
 7   basement          463 non-null    object
 8   hotwaterheating   463 non-null    object
 9   airconditioning   463 non-null    object
 10  parking           463 non-null    int64
 11  prefarea          463 non-null    object
 12  furnishingstatus  463 non-null    object
dtypes: int64(6), object(7)
memory usage: 47.2+ KB
     price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  9100000  6000         4          1        2      yes        no      yes
1  9100000  6600         4          2        2      yes       yes      yes
2  8890000  4600         3          2        2      yes       yes       no
3  8855000  6420         3          2        2      yes        no       no
4  8750000  4320         3          1        2      yes        no      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no              no        2       no   semi-furnished
```

```
1              no             yes        1      yes      unfurnished
2              no             yes        2       no        furnished
3              no             yes        1      yes   semi-furnished
4             yes              no        2       no   semi-furnished
```

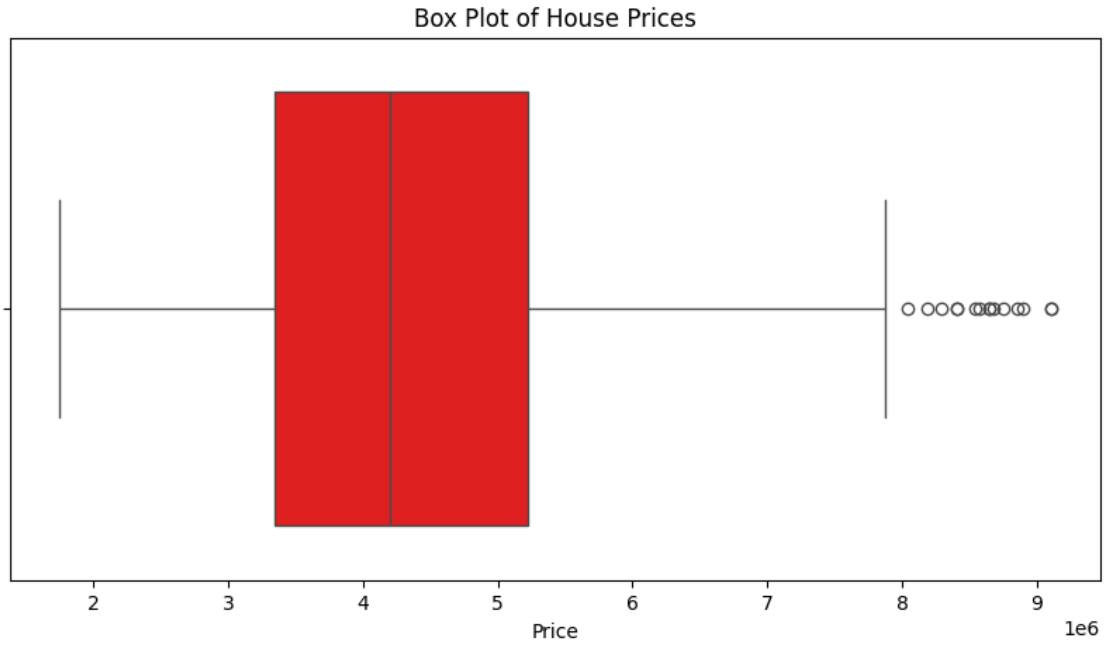
15.1 Price Distribution

The following plot shows the distribution of house prices in the dataset. The histogram provides a visual representation of the frequency of different price ranges, while the KDE plot overlays a smooth curve to indicate the density of the data points.

[13]:
```
plt.figure(figsize=(10, 5))
sns.histplot(df["price"], bins=30, color="pink", kde=True)
plt.title("Price Distribution")
plt.xlabel("Price")
plt.ylabel("Frequency")
plt.show()
```

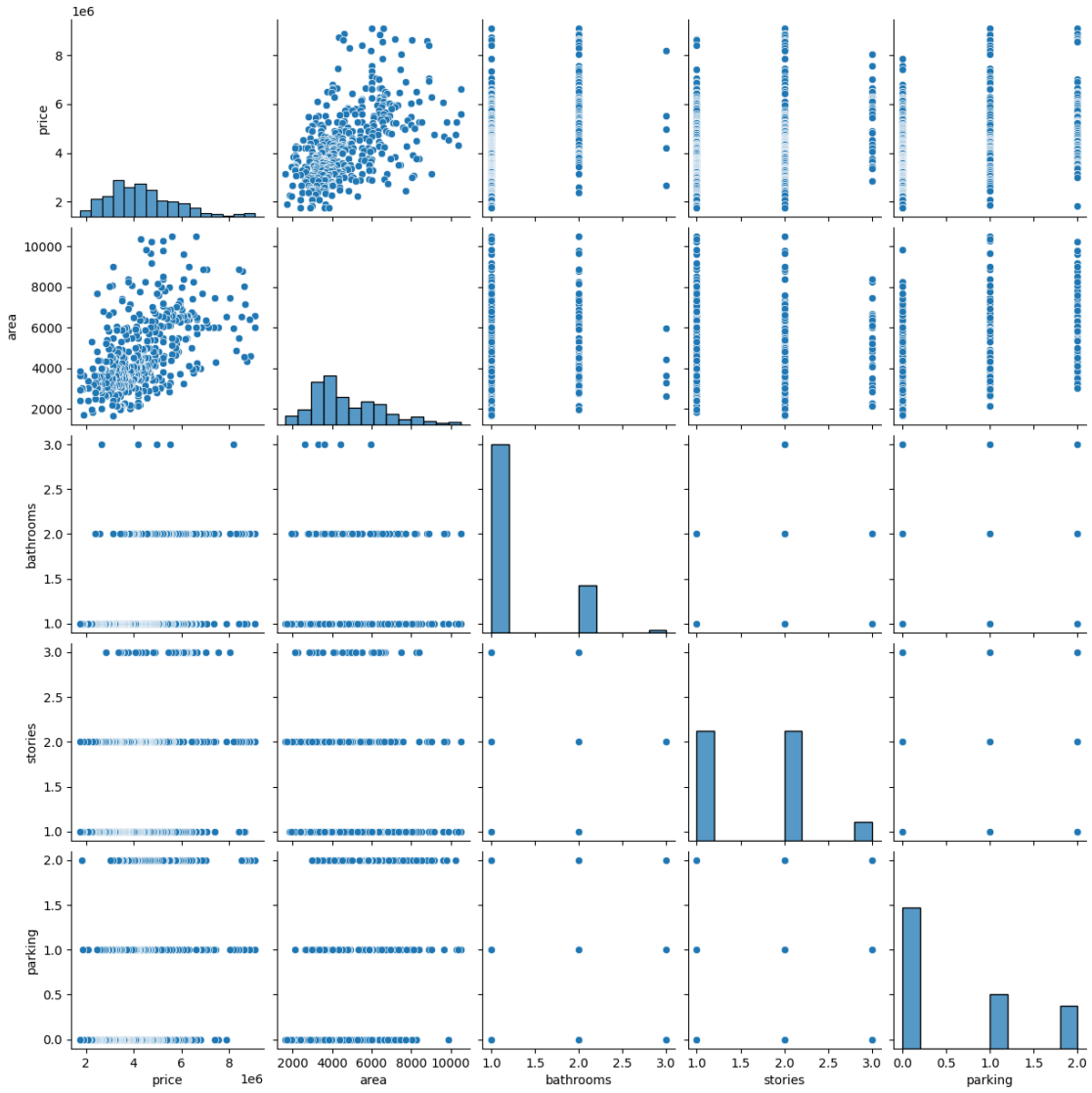
The following code creates a box plot to visualize the distribution of house prices in the dataset. The box plot helps identify the spread of the data, the median price, and any potential outliers.

[14]:
```
plt.figure(figsize=(10, 5))
sns.boxplot(x=df["price"], color="red")
plt.title("Box Plot of House Prices")
plt.xlabel("Price")
plt.show()
```



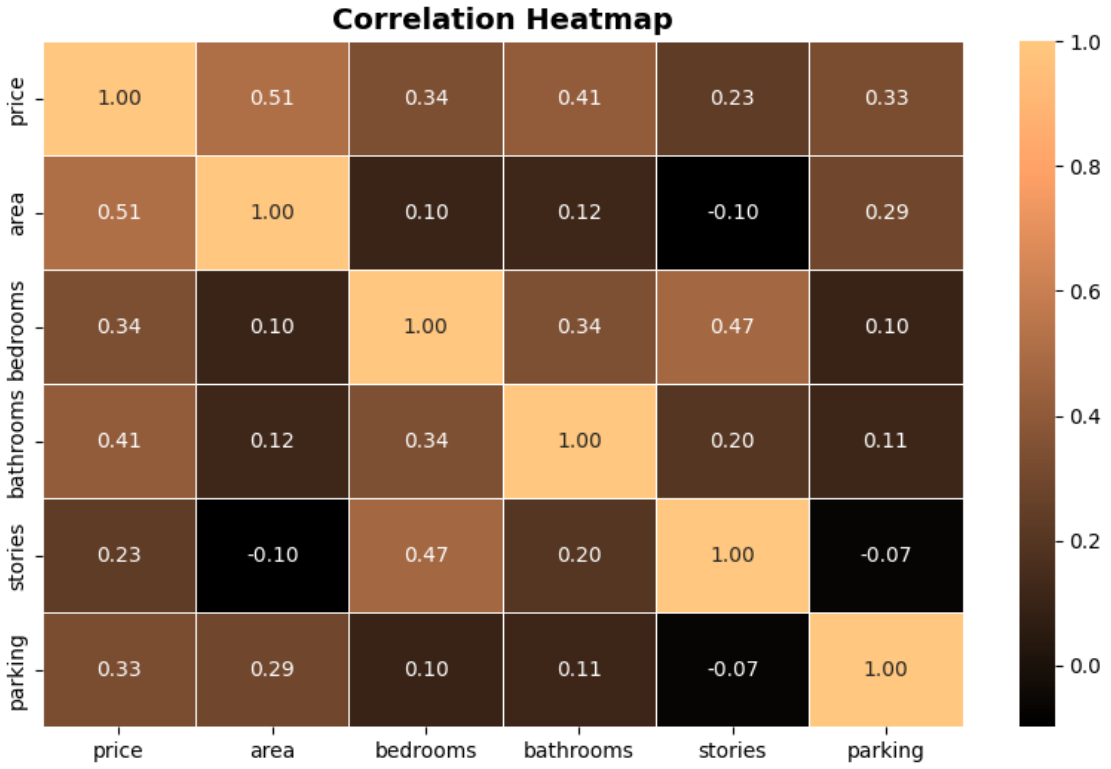
The code in the markdown cell at index 17 is used to create a pair plot for the features with high correlation in the dataset. The `sns.pairplot` function from the Seaborn library is used to visualize the pairwise relationships between the specified features: `price`, `area`, `bathrooms`, `stories`, and `parking`. The `plt.show()` function is used to display the plot.

[9]:
```
high_corr_features = ["price", "area", "bathrooms", "stories", "parking"]
sns.pairplot(df[high_corr_features])
plt.show()
```



Creates a correlation heatmap showing relationships between numeric variables in the dataset Values range from -1 (perfect negative correlation) to +1 (perfect positive correlation) Uses India's national colors for a patriotic visualization theme The annotations make it easy to read exact correlation values

[15]:
```
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='copper', fmt=".2f",␣
 ↪linewidths=0.5)

plt.title("Correlation Heatmap", fontsize=14, fontweight="bold")
plt.show()
```



[31]:
```
insights = [
    "1. Price is strongly correlated with area and number of bathrooms.",
    "2. Outliers exist in price, area, and bathrooms.",
    "3. Most houses have 2-4 bedrooms and 1-2 bathrooms.",
    "4. Features like air conditioning and guest rooms impact price.",
    "5. Parking and stories have less impact on price than area and bathrooms."
]
```

Check columns with “yes” or “no” values in dataset. for example “airconditioning” column, find out for all such columns if there is significant difference between prices of house with or without these facilities. Write python code for the same.

[29]:
```
from scipy.stats import ttest_ind
# Identify columns with "yes" or "no" values
yes_no_columns = [col for col in df.columns if df[col].isin(['yes', 'no']).
  ↪all()]

# Analyze price difference for each "yes/no" column
for column in yes_no_columns:
    yes_prices = df[df[column] == 'yes']['price']
```

```
    no_prices = df[df[column] == 'no']['price']

    # Perform t-test
    t_statistic, p_value = ttest_ind(yes_prices, no_prices)

    print(f"Analysis for {column}:")
    print(f"  T-statistic: {t_statistic}")
    print(f"  P-value: {p_value}")

    # Interpret results
    alpha = 0.05  # Significance level
    if p_value < alpha:
        print(f"  Significant price difference between houses with and without␣
↪{column}.")
    else:
        print(f"  No significant price difference between houses with and␣
↪without {column}.")
    print("-" * 20)
```

```
Analysis for mainroad:
  T-statistic: 6.667988980058728
  P-value: 7.435571010623803e-11
  Significant price difference between houses with and without mainroad.
--------------------
Analysis for guestroom:
  T-statistic: 7.460676622564867
  P-value: 4.317658041875818e-13
  Significant price difference between houses with and without guestroom.
--------------------
Analysis for basement:
  T-statistic: 6.470117383859754
  P-value: 2.505613937158052e-10
  Significant price difference between houses with and without basement.
--------------------
Analysis for hotwaterheating:
  T-statistic: 1.8258958344544944
  P-value: 0.06851266363257677
  No significant price difference between houses with and without
hotwaterheating.
--------------------
Analysis for airconditioning:
  T-statistic: 9.696308174857993
  P-value: 2.3503401532885515e-20
  Significant price difference between houses with and without airconditioning.
--------------------
Analysis for prefarea:
  T-statistic: 7.915433716172065
```

```
  P-value: 1.8516526002697993e-14
  Significant price difference between houses with and without prefarea.
--------------------
```

from file: 23AIML010_PR-09_QUES

16 23AIML010 OM CHOKSI PMRP 7 QUESTIONS ANSWERS ASSIGNMENT 9

16.1 7 QUESTIONS ANSWER

1) A researcher wants to test whether the average height of male students in a university is different from the national average of 175 cm. A random sample of 50 students has a mean height of 178 cm with a standard deviation of 10 cm. Using a Z-test at a 5% significance level, what is the Z-value for this hypothesis test, and what conclusion can be drawn?

2) A company claims that a new diet plan helps people lose 5 kg in a month. A random sample of 10 individuals following this diet had an average weight loss of 4.2 kg with a sample standard deviation of 1.5 kg. Using a T-test at a 5% significance level, should the company's claim be rejected? Assume a two-tailed test.

3) A hypothesis test is conducted where the null hypothesis states that the mean weight of a certain fruit is 200 grams. The test results in a p-value of 0.03. If the significance level ( ) is 0.05, what decision should be made regarding the null hypothesis? What if =0.01?

4) A company collects the monthly salaries (in thousands) of 12 employees: [30, 32, 35, 37, 40, 42, 45, 47, 50, 55, 60, 120] Use both Standard Deviation and IQR methods to detect outliers in the dataset. Compare the results.

5) The recorded temperatures (in °C) over 14 days are: [22, 24, 23, 25, 26, 24, 23, 22, 23, 24, 30, 31, 40, 42] Use the IQR method to detect any temperature anomalies.

6) A real estate agent collects the prices (in thousands of dollars) of 20 houses in a locality: [150, 160, 170, 180, 190, 200, 210, 220, 225, 230, 235, 240, 250, 260, 275, 280, 290, 295, 1000, 1200] Detect outliers using both Standard Deviation and IQR methods. What would be your conclusion about the real estate market in this locality?

7) For a chemical plant, two catalysts have been analyzed to test the mean yield of a chemical process. The catalyst 1 is currently in use but the process engineer wants to test whether catalyst 2, that is cheaper, could be adopted. The test is conducted in the pilot plant and the mean yield for each catalyst is as follows

Catalyst1,Catalyst2

91.5,89.19

94.18,90.95

92.18,90.46

95.39,93.21

91.79,97.19

89.07,97.04

94.72,91.07

89.21,92.75

Is there any difference in mean yield. Use =0.05 and assume equal variance

1.Z-test for Mean Height of Male Students

[2]:
```
import numpy as np
import scipy.stats as stats

sample_mean = 178
pop_mean = 175
sample_std = 10
n = 50


z_score = (sample_mean - pop_mean) / (sample_std / np.sqrt(n))


p_value = 2 * (1 - stats.norm.cdf(abs(z_score)))


alpha = 0.05
z_test_result = "Reject H0" if p_value < alpha else "Fail to reject H0"

print(f"Z-test for Height:")
print(f"   Z-score: {z_score:.2f}")
print(f"   P-value: {p_value:.4f}")
print(f"   Conclusion: {z_test_result}\n")
```

```
Z-test for Height:
   Z-score: 2.12
   P-value: 0.0339
   Conclusion: Reject H0
```

2. T-test for Weight Loss Diet

[3]:
```
sample_mean = 4.2
pop_mean = 5
sample_std = 1.5
n = 10


t_score = (sample_mean - pop_mean) / (sample_std / np.sqrt(n))
```

```
p_value = 2 * (1 - stats.t.cdf(abs(t_score), df=n-1))


t_test_result = "Reject H0" if p_value < alpha else "Fail to reject H0"

print(f" T-test for Weight Loss:")
print(f"   T-score: {t_score:.2f}")
print(f"   P-value: {p_value:.4f}")
print(f"   Conclusion: {t_test_result}\n")
```

```
T-test for Weight Loss:
  T-score: -1.69
  P-value: 0.1260
  Conclusion: Fail to reject H0
```

3.Decision Based on P-value

[4]:
```
p_value = 0.03

decision_alpha_005 = "Reject H0" if p_value < 0.05 else "Fail to reject H0"
decision_alpha_001 = "Reject H0" if p_value < 0.01 else "Fail to reject H0"

print(f" Hypothesis Decision:")
print(f"   At   = 0.05: {decision_alpha_005}")
print(f"   At   = 0.01: {decision_alpha_001}\n")
```

```
Hypothesis Decision:
  At   = 0.05: Reject H0
  At   = 0.01: Fail to reject H0
```

4. Outliers using Standard Deviation and IQR

[5]:
```
salaries = np.array([30, 32, 35, 37, 40, 42, 45, 47, 50, 55, 60, 120])

# Standard Deviation Method
mean_salary = np.mean(salaries)
std_salary = np.std(salaries, ddof=1)
threshold = 3
outliers_std = salaries[(salaries > mean_salary + threshold * std_salary) |␣
 ↪(salaries < mean_salary - threshold * std_salary)]

# IQR Method
Q1 = np.percentile(salaries, 25)
Q3 = np.percentile(salaries, 75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
```

```
upper_bound = Q3 + 1.5 * IQR
outliers_iqr = salaries[(salaries < lower_bound) | (salaries > upper_bound)]

print(f"Salary Outliers Detection:")
print(f"   Outliers (Std Dev method): {outliers_std}")
print(f"   Outliers (IQR method): {outliers_iqr}\n")
```

```
Salary Outliers Detection:
   Outliers (Std Dev method): []
   Outliers (IQR method): [120]
```

5) Temperature Anomalies using IQR

[6]:
```
temperatures = np.array([22, 24, 23, 25, 26, 24, 23, 22, 23, 24, 30, 31, 40,␣
 ↪42])

Q1 = np.percentile(temperatures, 25)
Q3 = np.percentile(temperatures, 75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers_temp = temperatures[(temperatures < lower_bound) | (temperatures >␣
 ↪upper_bound)]

print(f"Temperature Anomalies:")
print(f"   Outliers: {outliers_temp}\n")
```

```
Temperature Anomalies:
   Outliers: [40 42]
```

16.1.1 6) Outliers in House Prices using Std Dev and IQR

[7]:
```
house_prices = np.array([150, 160, 170, 180, 190, 200, 210, 220, 225, 230,
                         235, 240, 250, 260, 275, 280, 290, 295, 1000, 1200])

# Standard Deviation Method
mean_price = np.mean(house_prices)
std_price = np.std(house_prices, ddof=1)
outliers_std = house_prices[(house_prices > mean_price + threshold * std_price)␣
 ↪| (house_prices < mean_price - threshold * std_price)]

# IQR Method
Q1 = np.percentile(house_prices, 25)
Q3 = np.percentile(house_prices, 75)
IQR = Q3 - Q1
```

```
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers_iqr = house_prices[(house_prices < lower_bound) | (house_prices >␣
 ↪upper_bound)]

print(f"  House Price Outliers:")
print(f"   Outliers (Std Dev method): {outliers_std}")
print(f"   Outliers (IQR method): {outliers_iqr}\n")
```

```
 House Price Outliers:
  Outliers (Std Dev method): [1200]
  Outliers (IQR method): [1000 1200]
```

16.1.2 7) T-test for Catalysts (Assuming Equal Variance)

[8]:
```
catalyst_1 = np.array([91.5, 94.18, 92.18, 95.39, 91.79, 89.07, 94.72, 89.21])
catalyst_2 = np.array([89.19, 90.95, 90.46, 93.21, 97.19, 97.04, 91.07, 92.75])

# Perform independent t-test (equal variance assumed)
t_score, p_value = stats.ttest_ind(catalyst_1, catalyst_2, equal_var=True)

# Conclusion
t_test_result = "Reject H0" if p_value < alpha else "Fail to reject H0"

print(f"7) T-test for Catalysts:")
print(f"   T-score: {t_score:.2f}")
print(f"   P-value: {p_value:.4f}")
print(f"   Conclusion: {t_test_result}")
```

```
7) T-test for Catalysts:
   T-score: -0.35
   P-value: 0.7289
   Conclusion: Fail to reject H0
```

[ ]:
```

```

This notebook was converted with convert.ploomber.io