Problem Statement 1:

Write a python program to print all the prime numbers that fall between two numbers taken as input from the user (include both the numbers also).

Description:

The task is to create a Python program that takes two integer numbers as input from the user and then prints all the prime numbers that fall between (inclusive of) those two input numbers.

Algorithm:

```
1.  START
2.      INPUT lower as an integer
3.      INPUT upper as an integer
4.      IF lower > upper
5.          SWAP lower and upper
6.      END IF
7.      PRINT "The prime numbers between ", lower, " and ", upper, " are:"
8.      WHILE lower <= upper
9.          IF lower < 2
10.             lower <- 2
11.             CONTINUE
12.         ELSE IF lower <= 3
13.             PRINT lower, " , "
14.             lower <- lower + 1
15.         ELSE
16.             prime <- True
17.             i <- 2
18.             WHILE i < lower
19.                 IF lower modulo i equals 0
20.                     prime <- False
21.                     BREAK
22.                 END IF
23.                 i <- i + 1
24.             END WHILE
25.             IF prime is True
26.                 PRINT lower, " , "
27.             END IF
28.             lower <- lower + 1
29.         END IF
30.     END WHILE
31. END
```

Python Source Code:

```
lower = int(input("Enter the lower bound: "))
upper = int(input("Enter the upper bound: "))

if lower > upper:
   lower, upper = upper, lower

print("The prime numbers between ", lower, " and ", upper, " are:")
while lower <= upper:
   if lower < 2:
      lower = 2
      continue
   elif lower <= 3:
      print(lower, end=', ')
      lower += 1
   else:
      prime = True

      i = 2
      while i < lower:
         if lower % i == 0:
            False
            break
         i += 1

      if prime:
         print(lower, end=', ')
      lower += 1
```

Output:

Enter the lower bound: 1

Enter the upper bound: 100

The prime numbers between 1 and 100 are:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,

Enter the lower bound: 57

Enter the upper bound: -100

The prime numbers between -100 and 57 are:

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57,

Problem Statement 2:

Write a python program to find the largest and smallest digit in a number taken as input from the user.

Description:

Develop a Python program that takes a number as input from the user and then determines the largest and smallest digits within that number.

Algorithm:

```
1.  START
2.      INPUT number as an integer
3.      largest <- 0
4.      smallest <- 10
5.
6.      IF number is not 0
7.          WHILE number > 0
8.              digit <- number modulo 10
9.              largest <- maximum(largest, digit)
10.             smallest <- minimum(smallest, digit)
11.             number <- number // 10
12.         END WHILE
13.     ELSE
14.         smallest <- 0
15.     END IF
16.
17.     PRINT "Largest digit:", largest
18.     PRINT "Smallest digit:", smallest
19. END
```

Python Source Code:

```
number = int(input("Enter a number: "))

largest = 0
smallest = 10

# in case number is 00000 or sometrhing like that
if not number:
    smallest = 0

while number > 0:
```

```
    digit = number % 10
    largest = max(largest, digit)
    smallest = min(smallest, digit)
    number //= 10

print("Largest digit:", largest)
print("Smallest digit:", smallest)
```

Output:

```
Enter a number: 1230
Largest digit: 3
Smallest digit: 0
```

```
Enter a number: 00000
Largest digit: 0
Smallest digit: 0
```

Problem Statement 3:

Write a python program to print numbers from 1 to 100 except multiples of 2 & 5.

Description:

Create a Python program to display numbers in the range of 1 to 100, excluding those that are multiples of 2 and 5.

Algorithm:

```
1.  START
2.      num <- 1
3.
4.      PRINT "Numbers between 1 and 100 not divisible by 2 and 5 are:"
5.      WHILE num < 101
6.          IF NOT (num modulo 2 equals 0 AND num modulo 5 equals 0)
7.              PRINT num, " , "
8.          END IF
9.          num <- num + 1
10.     END WHILE
11. END
```

Python Source Code:

```
num = 1

print("Numbers between 1 and 100 not divisible by 2 and 5 are:")
while num < 101:
    if not (num % 2 == 0 and num % 5 == 0):
        print(num, end=", ")
    num += 1
```

Output:

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28,
29, 31, 32, 33, 34, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 48, 49, 51, 52, 53, 54,
55, 56, 57, 58, 59, 61, 62, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 74, 75, 76, 77, 78, 79,
81, 82, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 98, 99,
```

Problem Statement 4:

Write a python program to take number input from the user till user enters -1. Display the sum and average of all the numbers.

Description:

Design a Python program that prompts the user for number inputs repeatedly until the user enters -1. Calculate and exhibit the sum and average of all the entered numbers.

Algorithm:

```
1.  START
2.      summ <- 0
3.      count <- 0
4.
5.      PRINT "Enter a number (-1 to stop): "
6.
7.      WHILE True
8.          num <- INPUT as an integer
9.          IF num equals -1
10.             BREAK
11.         END IF
12.         summ <- summ + num
13.         count <- count + 1
14.     END WHILE
15.
16.     IF summ is not 0
17.         average <- summ / count
18.         PRINT "Sum of entered numbers:", summ
19.         PRINT "Average of entered numbers:", average
20.     ELSE
21.         PRINT "No numbers entered."
22.     END IF
23. END
```

Python Source Code:

```
summ = 0
count = 0

print("Enter a number (-1 to stop): ")

while True:
   num = int(input())
```

```
    if num == -1:
        break
    summ += num
    count += 1

if summ:
    average = summ / count
    print("Sum of entered numbers:", summ)
    print("Average of entered numbers:", average)
else:
    print("No numbers entered.")
```

Output:

```
Enter a number (-1 to stop):
1
2
3
4
5
-1
Sum of entered numbers: 15
Average of entered numbers: 3.0
```

Problem Statement 5:

Write a python program to find the mean, median and mode of n numbers taken as input.

Description:

Create a Python program that accepts 'n' numbers as input and calculates the mean, median, and mode of the entered numbers.

Algorithm:

```
START
    n <- INPUT "Enter the number of values: "
    numbers <- empty list
    total <- 0

    i <- 0
    WHILE i < n
        num <- INPUT as a float "Enter value " + (i+1)
        ADD num to numbers
        total <- total + num

        i <- i + 1
    END WHILE

    mean <- total / i

    SORT numbers

    IF n modulo 2 equals 0
        median <- (numbers[n//2 - 1] + numbers[n//2]) / 2
    ELSE
        median <- numbers[n//2]
    END IF

    maxx_count <- 0
    mode <- numbers[0]

    i <- 1
    WHILE i < n
        count <- 1
        WHILE i < n AND numbers[i] equals numbers[i-1]
            count <- count + 1
            i <- i + 1
        END WHILE
```

```
    IF count > maxx_count
        maxx_count <- count
        mode <- numbers[i-1]
    END IF

    i <- i + 1
END WHILE

PRINT "Mean:", mean
PRINT "Median:", median
PRINT "Mode:", mode
END
```

Python Source Code:

```
n = int(input("Enter the number of values: "))
numbers = []
total = 0

i = 0
while i < n:
    num = float(input(f"Enter value {i+1}: "))
    numbers.append(num)
    total += num

    i += 1

# mean
mean = total / i

#median
numbers.sort()

if n % 2 == 0:
    median = (numbers[n//2 - 1] + numbers[n//2]) / 2
else:
    median = numbers[n//2]

# mode
maxx_count = 0
mode = numbers[0]

i = 1
while i < n:
    count = 1
    while i < n  and numbers[i] == numbers[i-1]:
```

```
        count += 1
        i += 1
```

```
    if count > maxx_count:
        maxx_count = count
        mode = numbers[i-1]

    i += 1

print("Mean:", mean)
print("Median:", median)
print("Mode:", mode)
```

Output:

```
Enter the number of values: 3
Enter value 1: 1
Enter value 2: 2
Enter value 3: 2
Mean: 1.6666666666666667
Median: 2.0
Mode: 2.0
```

```
Enter the number of values: 11
Enter value 1: 1
Enter value 2: 2
Enter value 3: 5
Enter value 4: 5
Enter value 5: 3
Enter value 6: 8
Enter value 7: 8
Enter value 8: 8
Enter value 9: 1
Enter value 10: 2
Enter value 11: 5
Mean: 4.363636363636363
Median: 5.0
Mode: 5.0
```

Problem Statement 6:
Write a program to display the number names of the digits of a number.
For e.g. 12 : One Two

Description:

This problem requires us to write a program that accepts an integer as input and outputs the names of each digit in the number. The digits from 0 to 9 should be mapped to their corresponding names, "Zero" to "Nine".

Algorithm:

1. Define a dictionary with keys as digits (0-9) and values as their corresponding names ("Zero" to "Nine").
2. Convert the input number into a string.
3. For each digit in the string, look up its name in the dictionary and print it.

Python Code:

```
digit_names = {
      '0': 'Zero',
      '1': 'One',
      '2': 'Two',
      '3': 'Three',
      '4': 'Four',
      '5': 'Five',
      '6': 'Six',
      '7': 'Seven',
      '8': 'Eight',
      '9': 'Nine'
}

def display_number_names(number):
   number_string = str(number)

   for digit in number_string:
      print(digit_names[digit], end=' ')

display_number_names(input("Enter number:\n"))
```

Output:

```
Enter number:
186

One Eight Six
```

Problem Statement 7:
Write a program to convert decimal to binary, octal and hexadecimal values.

Description:
The decimal number is a base-10 number system that uses digits from 0 to 9. On the other hand, binary, octal, and hexadecimal are base-2, base-8, and base-16 number systems respectively.

The conversion from decimal to binary, octal, and hexadecimal involves dividing the decimal number by the base of the target number system and keeping track of the remainders. The remainders form the digits of the target number system.

Algorithm:
1. Divide the decimal number by the base of the target number system (2 for binary, 8 for octal, and 16 for hexadecimal).
2. Keep track of the remainder.
3. Repeat steps 1 and 2 until the decimal number becomes 0.
4. The digits of the target number system are the remainders obtained in reverse order.

Python Code:

```
def decimal_to_binary(n):
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n = n // 2
    return binary

def decimal_to_octal(n):
    octal = ""
    while n > 0:
        octal = str(n % 8) + octal
        n = n // 8
    return octal

def decimal_to_hexadecimal(n):
    hexadecimal = ""
    while n > 0:
        remainder = n % 16
        if remainder < 10:
            hexadecimal = str(remainder) + hexadecimal
        else:
            hexadecimal = chr(ord('A') + remainder - 10) + hexadecimal
        n = n // 16
    return hexadecimal

n = input(“Enter number:\n”)
print(decimal_to_binary(n))
print(decimal_to_octal(n))
print(decimal_to_hexadecimal(n))
```

Output:

Enter number:

1697

11010100001

3241

6A1

Problem Statement 8:

Write a program to add first n terms of the following series:

1/1! + 1/2! + 1/3! …….. 1/n!

Description:

The series is a series of fractions where the numerator is 1 and the denominator is the factorial of the term number. The factorial of a number is the product of all positive integers less than or equal to that number.

Algorithm:

1. Initialize a variable sum to 0. This variable will hold the sum of the series.
2. For each term from 1 to n:
    a. Calculate the factorial of the term number.
    b. Add 1 divided by the factorial of the term number to sum.
3. Return sum.

Python Code:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

def sum_of_series(n):
    sum = 0
    for i in range(1, n+1):
        sum += 1 / factorial(i)
    return sum

print(sum_of_series(int(input("Enter n:\n"))))
```

Output:

```
Enter n:
156
1.7182818284590455
```

Problem Statement 9:
Write a NumPy program to compute sum of all elements, sum of each column and sum of each row of a given array.

Description:
This program will print a NumPy program to compute sum of all elements, sum of each column and sum of each row of a given array.

Algorithm:

1. Creates a 2D array 'x' with the shape (2, 2) and elements.
2. In the ‘print(np.sum(x))’ statement, np.sum() function.
3. The sum of all elements in the array 'x'.
4. In the ‘print(np.sum(x, axis=0))’ statement.
5. The result is a 1D array with the sum of each column:
6. Finally in ‘print(np.sum(x, axis=1)).
7. The result is a 1D array with the sum of each row.
8. The output is then printed. Step 10 = Stop.

Python Code:

```
import numpy as np
arr = np.array([[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]])
total_sum = np.sum(arr)
column_sums = np.sum(arr, axis=0)
row_sums = np.sum(arr, axis=1)
print("Original Array:")
print(arr)
print("\nSum of All Elements:", total_sum)
print("\nSum of Each Column:")
print(column_sums)
print("\nSum of Each Row:")
print(row_sums)
```

Output:

```
Original Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Sum of All
Elements: 45
Sum of Each
Column: [12
15 18]
Sum of
Each
Row: [
6 15 24
```

Problem Statement 10:
Write a NumPy program to find the union of two arrays. Union will return the unique, sorted array of values that are in either of the two input arrays

Description:
This program will print a NumPy program to find the union of two arrays. Union will return the unique, sorted array of values that are in either of the two input arrays.

Algorithm:
Step 1 = Start.
Step 2 = To find the union of two NumPy arrays
Step 3 = Using the union1d() function,
Step 4 = First import the NumPy library.
Step 5 = Then create two arrays arr1 and arr2.
Step 6 = Finally, we use the union1d() function to find
Step 7 = The union of these two arrays
Step 8 = Stop.

Python Code:

```
import numpy as np
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([4, 5, 6, 7, 8])

union_result = np.union1d(array1, array2)

print("Array 1:", array1)
print("Array 2:", array2)
print("Union of Array 1 and Array 2:", union_result)
```

Output:
Array 1: [1 2 3 4 5]
Array 2: [4 5 6 7 8]
Union of Array 1 and Array 2: [1 2 3 4 5 6 7 8]

Problem Statement 11:
Write a program to implement pandas datatype (both series and data frame datatype) by converting a list.

Description: This a program to implement pandas datatype (both series and data frame datatype) by converting a list.

Algorithm:

Step 1 = Start.
Step 2 = Import the Pandas library.
Step 3 = Define or obtain the list of data that you want to convert into a Series and DataFrame.
Step 4 = Create a Pandas Series:
- Use the `pd.Series()` constructor.
  - Pass the list as an argument to `pd.Series()` to create the Series. Step 5 = Create a Pandas DataFrame:
- Use the `pd.DataFrame()` constructor.
  - Pass the list as a dictionary with a column name as the key and the list as the value to create the DataFrame.

Step 6 = Print or utilize the Pandas Series and DataFrame as needed.

Python Code:

```
import pandas as pd
data_list = [1, 2, 3, 4, 5]

series_data = pd.Series(data_list)
df_data = pd.DataFrame({'Column_Name': data_list})
print("Pandas Series:")
print(series_data)

print("\nPandas DataFrame:")
print(df_data)
```

Output:

```
Pandas Series:
0    1
1    2
2    3
3    4
4    5
dtype: int64

Pandas DataFrame:
   Column_Name
0            1
1            2
2            3
3            4
4            5
```

Problem Statement 12:
Write a Pandas program to get the information of the DataFrame (stocks.csv file)including data types and memory usage.

Description:
This program will print a Pandas program to get the information of the DataFrame (stocks.csv file)including data types and memory usage.

Algorithm:

Step 1 = Start.
Step 2 = Import the Pandas library.
Step 3 = Read the CSV file into a Pandas DataFrame using the `pd.read_csv()` function.
Step 4 = Use the `info()` method of the DataFrame to get information about the DataFrame, including data types and memory usage.
Step 5 = Print or display the information.
Step 6 = Stop.

Python Code:

```
import pandas as pd
df = pd.read_csv('stocks.csv') # Replace 'stocks.csv' with your file path
df_info = df.info()
print(df_info)
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 5 columns):
 #   Column     Non-Null Count  Dtype
--  ----       ---------       ----
 0 Date        1000 non-null   object
 1 Open        1000 non-null   float64
 2 High        1000 non-null   float64
 3 Low         1000 non-null   float64

dtypes: float64(3), object(1)
memory usage: 39.2+ KB
```

Problem Statement 13:
Write a Pandas program to display the movies (title, runtime) longer than 30 minutes and shorter than 360 minutes. (use movies_metadata.csv file).

Description:
This program will print a Pandas program to display the movies (title, runtime) longer than 30 minutes and shorter than 360 minutes. (use movies_metadata.csv file).

Algorithm:

Step 1 = Start.
Step 2 = Import the Pandas library.
Step 3 = Read the CSV file ('movies_metadata.csv') into a Pandas DataFrame using the `pd.read_csv()` function.
Step 4 = Filter the DataFrame based on the runtime criteria:
- Use boolean indexing to select rows where the runtime is greater than 30 and less than 360.
Step 5 = Create a new DataFrame containing only the 'title' and 'runtime' columns. Step 6 = Display the filtered DataFrame.
Step 7 = Stop.

Code:

```
import pandas as pd

df = pd.read_csv('movies_metadata.csv', low_memory=False)

filtered_movies = df[(df['runtime'] > 30) & (df['runtime'] < 360)]


result = filtered_movies[['title', 'runtime']]
print(result)
```

Output:

	Title	runtime
0	Movie 1	105
2	Movie 2	127
5	Movie 3	92
...		

Problem Statement 14: Automobile Data

1. From the given dataset print the first and last five rows

```
import pandas as pd

csv = r"C:\Users\lucky\Downloads\Automobile_data.csv"
df = pd.read_csv(csv)

df.head(5)
df.tail(5)
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
0	0	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	13495.0
1	1	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	16500.0
2	2	alfa-romero	hatchback	94.5	171.2	ohcv	six	154	19	16500.0
3	3	audi	sedan	99.8	176.6	ohc	four	102	24	13950.0
4	4	audi	sedan	99.4	176.6	ohc	five	115	18	17450.0

```
df.tail(5)
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
56	81	volkswagen	sedan	97.3	171.7	ohc	four	85	27	7975.0
57	82	volkswagen	sedan	97.3	171.7	ohc	four	52	37	7995.0
58	86	volkswagen	sedan	97.3	171.7	ohc	four	100	26	9995.0
59	87	volvo	sedan	104.3	188.8	ohc	four	114	23	12940.0
60	88	volvo	wagon	104.3	188.8	ohc	four	114	23	13415.0

2. Clean the dataset and update the CSV file

Replace all column values which contain : ? , na, n.a. or NaN

```
print(df[df.isnull().any(axis=1)])
df.dropna(inplace=True)
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
22	31	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
23	32	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
47	63	porsche	hatchback	98.4	175.7	dohcv	eight	288	17	NaN

3. Find the most expensive car company name

```
max_p = df.price.max()
df[df['price']==max_p].company
df.loc[df['price'].idxmax(), 'company']
```

```
35    mercedes-benz
Name: company, dtype: object

'mercedes-benz'
```

4. Print All Toyota Cars details

```
df[df['company']=='toyota']
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
48	66	toyota	hatchback	95.7	158.7	ohc	four	62	35	5348.0
49	67	toyota	hatchback	95.7	158.7	ohc	four	62	31	6338.0
50	68	toyota	hatchback	95.7	158.7	ohc	four	62	31	6488.0
51	69	toyota	wagon	95.7	169.7	ohc	four	62	31	6918.0
52	70	toyota	wagon	95.7	169.7	ohc	four	62	27	7898.0
53	71	toyota	wagon	95.7	169.7	ohc	four	62	27	8778.0
54	79	toyota	wagon	104.5	187.8	dohc	six	156	19	15750.0

5. Count total cars per company

```
df.groupby('company').size()
```

```
company
alfa-romero      3
audi             4
bmw              6
chevrolet        3
dodge            2
honda            3
isuzu            3
jaguar           3
mazda            5
mercedes-benz    4
mitsubishi       4
nissan           5
porsche          3
toyota           7
volkswagen       4
volvo            2
dtype: int64
```

6. Find each company’s Highest price car

```
df.loc[df.groupby('company')['price'].idxmax()]
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
1	1	alfa-romero	convertible	88.6	168.8	dohc	four	111	21	16500.0
6	6	audi	wagon	105.8	192.7	ohc	five	110	19	18920.0
11	14	bmw	sedan	103.5	193.8	ohc	six	182	16	41315.0
15	18	chevrolet	sedan	94.5	158.8	ohc	four	70	38	6575.0
16	19	dodge	hatchback	93.7	157.3	ohc	four	68	31	6377.0
19	28	honda	sedan	96.5	175.4	ohc	four	101	24	12945.0
21	30	isuzu	sedan	94.3	170.7	ohc	four	78	24	6785.0
26	35	jaguar	sedan	102.0	191.7	ohcv	twelve	262	13	36000.0
31	43	mazda	sedan	104.9	175.0	ohc	four	72	31	18344.0
35	47	mercedes-benz	hardtop	112.0	199.2	ohcv	eight	184	14	45400.0
39	52	mitsubishi	sedan	96.3	172.4	ohc	four	88	25	8189.0
44	57	nissan	sedan	100.4	184.6	ohcv	six	152	19	13499.0
46	62	porsche	convertible	89.5	168.9	ohcf	six	207	17	37028.0
54	79	toyota	wagon	104.5	187.8	dohc	six	156	19	15750.0
58	86	volkswagen	sedan	97.3	171.7	ohc	four	100	26	9995.0
60	88	volvo	wagon	104.3	188.8	ohc	four	114	23	13415.0

7. Find the average mileage of each car making company

```
df.groupby('company')[['average-mileage']].mean()
```

company	average-mileage
alfa-romero	20.333333
audi	20.000000
bmw	19.000000
chevrolet	41.000000
dodge	31.000000
honda	26.333333
isuzu	33.333333
jaguar	14.333333
mazda	28.000000
mercedes-benz	18.000000
mitsubishi	29.500000
nissan	31.400000
porsche	17.000000
toyota	28.714286
volkswagen	31.750000
volvo	23.000000

8. Sort all cars by Price column

```
df.sort_values('price', ascending=False)
```

	index	company	body-style	wheel-base	length	engine-type	num-of-cylinders	horsepower	average-mileage	price
35	47	mercedes-benz	hardtop	112.0	199.2	ohcv	eight	184	14	45400.0
11	14	bmw	sedan	103.5	193.8	ohc	six	182	16	41315.0
34	46	mercedes-benz	sedan	120.9	208.1	ohcv	eight	184	14	40960.0
46	62	porsche	convertible	89.5	168.9	ohcf	six	207	17	37028.0
12	15	bmw	sedan	110.0	197.0	ohc	six	182	15	36880.0
...	...	...	...	...	...	...	...	...	...	...
27	36	mazda	hatchback	93.1	159.1	ohc	four	68	30	5195.0
13	16	chevrolet	hatchback	88.4	141.1	l	three	48	47	5151.0
22	31	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
23	32	isuzu	sedan	94.5	155.9	ohc	four	70	38	NaN
47	63	porsche	hatchback	98.4	175.7	dohcv	eight	288	17	NaN

61 rows × 10 columns

Problem Statement 15:

Using the given dataset perform the following :

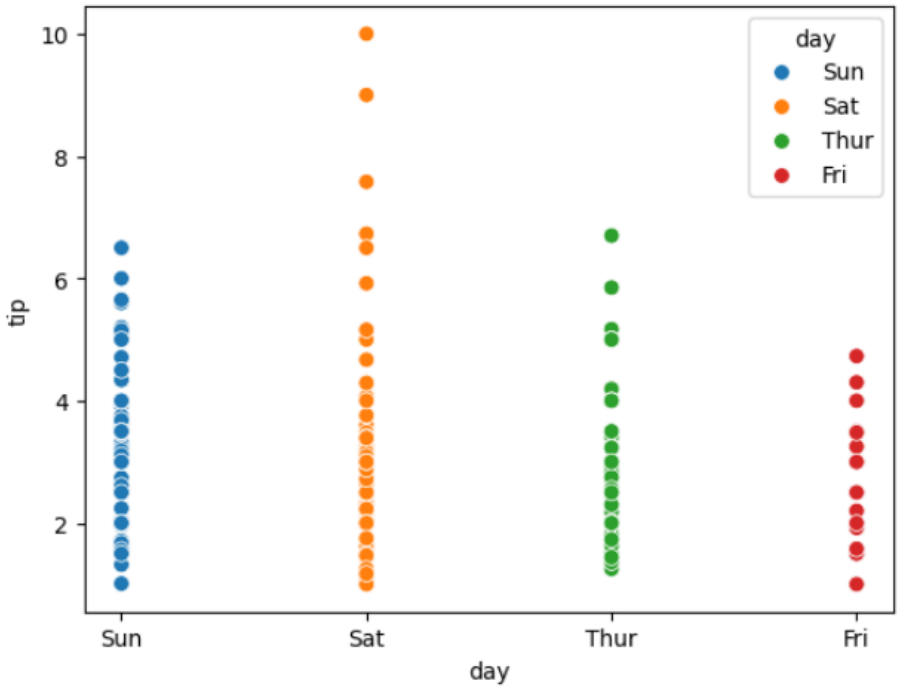
1. Display the top 5 rows of the dataset

```
df.head(5)
```

	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

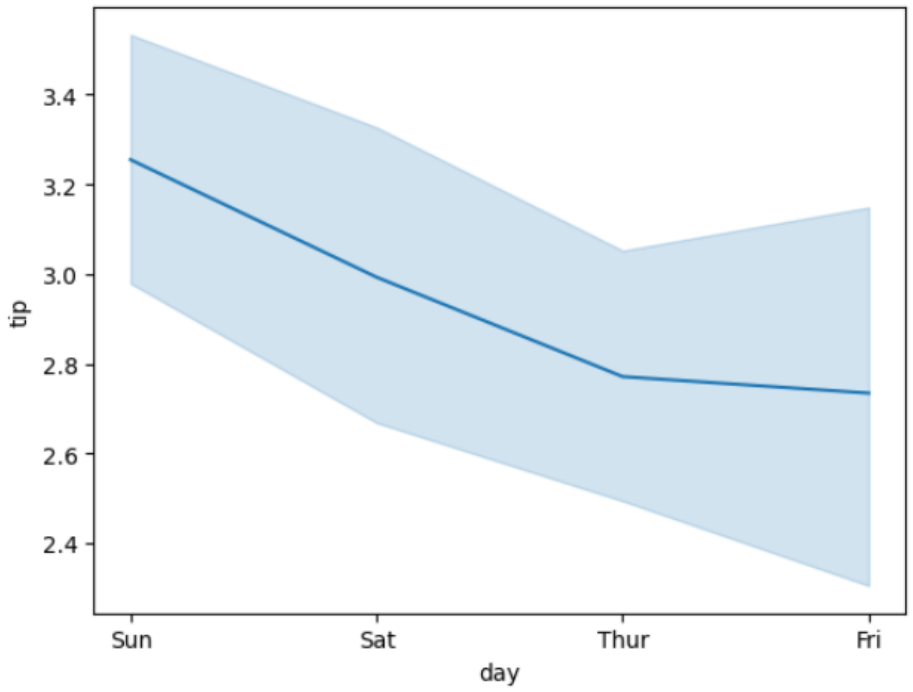
2. Create Scatter plot with day against the tip.

```
sns.scatterplot(x='day', y='tip', data=df, hue='day', s=50)
plt.show()
```



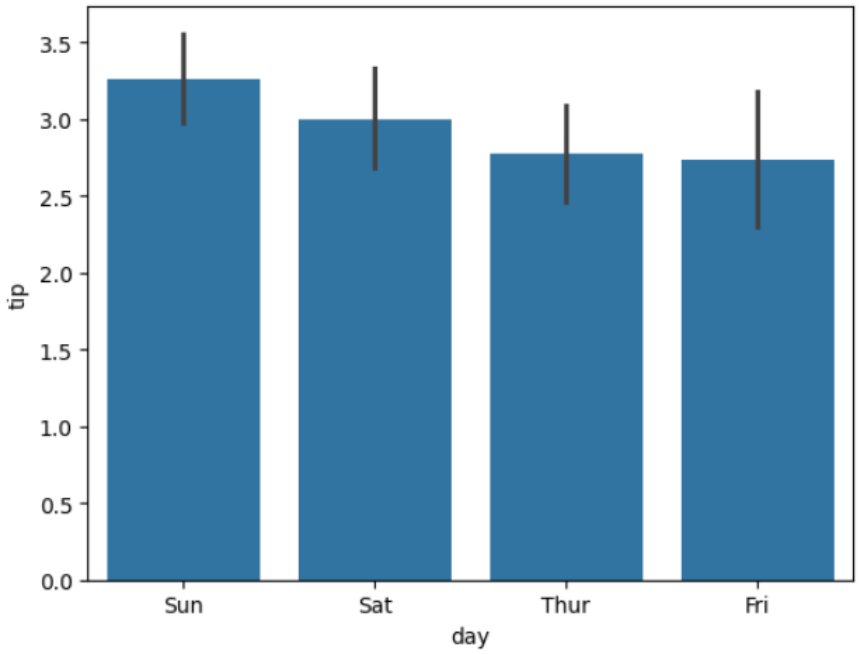
3. Create Line plot with day against tip.

```
sns.lineplot(x='day', y='tip', data=df)
plt.show()
```



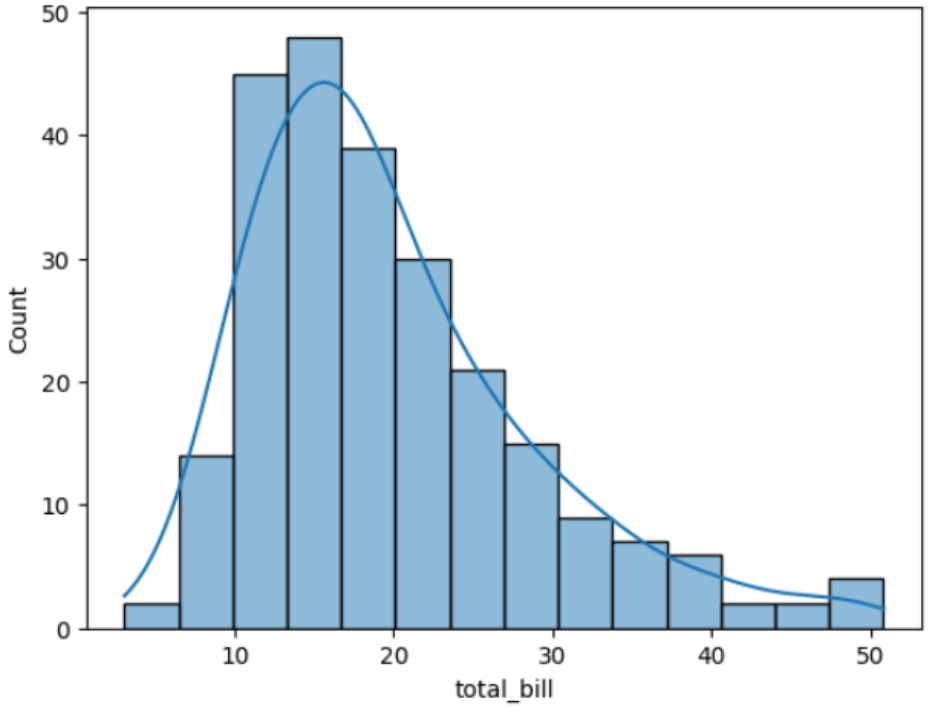
4. Create Bar chart with day against tip.

```
sns.barplot(data=df, x='day', y='tip')
plt.show()
```



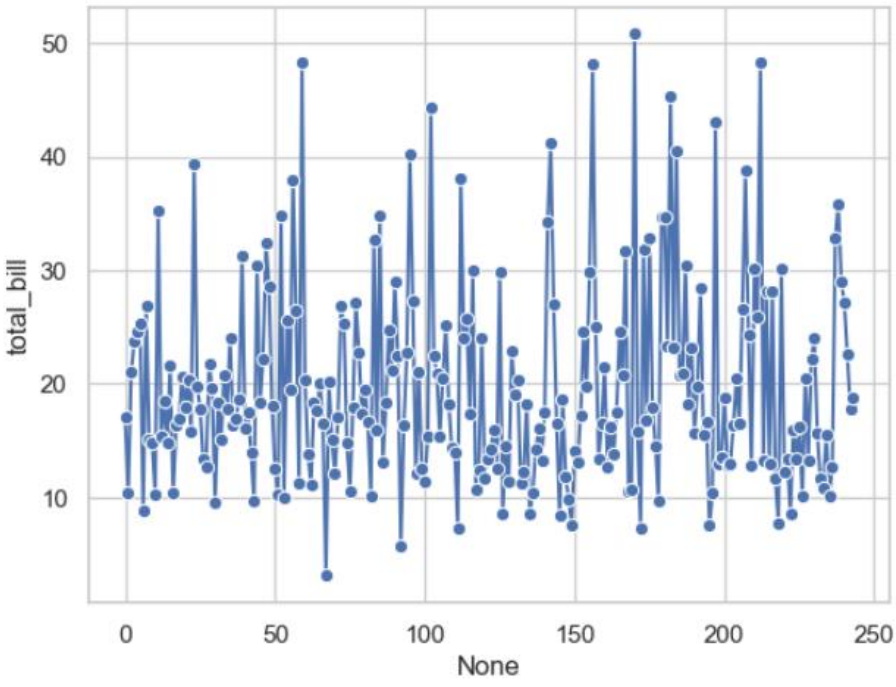
5. Create histogram of total_bills

```
sns.histplot(data=df, x='total_bill', kde=True)
plt.show()
```



6. Draw line plot for total_bills

```
sns.set(style="whitegrid")
sns.lineplot(data=df, x=df.index, y='total_bill', marker='o')
plt.show()
```



Problem Statement 16:

Case Study 1:

Coronavirus disease 2019 (COVID-19) is an infectious disease caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The disease was first identified in 2019 in Wuhan, China, and has since spread globally, resulting in the 2019–20 coronavirus pandemic. Common symptoms include fever, cough and shortness of breath. Muscle pain, sputum production and sore throat are less common. The rate of deaths per number of diagnosed cases is on average 3.4%, ranging from 0.2% in those less than 20 to approximately 15% in those over 80 years old.

Link for dataset : https://github.com/CSSEGISandData/COVID-19/tree/master/csse_covid_19_data/csse_covid_19_daily_reports

1. Write a Python program to display first 5 rows from COVID-19 dataset. Also print the dataset information and check the missing values.

```
import pandas as pd

df = pd.read_csv(r"C:\Users\lucky\Downloads\04-30-2021.csv")

df.head()
df.info()
df.isnull().sum()
```

	FIPS	Admin2	Province_State	Country_Region	Last_Update	Lat	Long_	Confirmed	Deaths	Recovered	Active	Combined_Key	Incident_Rate	Case_Fatali
0	NaN	NaN	NaN	Afghanistan	2021-05-01 04:20:47	33.93911	67.709953	59745	2625	53206.0	3914.0	Afghanistan	153.474303	
1	NaN	NaN	NaN	Albania	2021-05-01 04:20:47	41.15330	20.168300	131085	2394	109338.0	19353.0	Albania	4555.042046	
2	NaN	NaN	NaN	Algeria	2021-05-01 04:20:47	28.03390	1.659600	122108	3253	85108.0	33747.0	Algeria	278.460879	
3	NaN	NaN	NaN	Andorra	2021-05-01 04:20:47	42.50630	1.521800	13232	125	12684.0	423.0	Andorra	17125.477254	
4	NaN	NaN	NaN	Angola	2021-05-01 04:20:47	-11.20270	17.873900	26652	596	23876.0	2180.0	Angola	81.092262	

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4014 entries, 0 to 4013
Data columns (total 14 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   FIPS                 3266 non-null   float64
 1   Admin2               3271 non-null   object
 2   Province_State       3835 non-null   object
 3   Country_Region       4014 non-null   object
 4   Last_Update          4014 non-null   object
 5   Lat                  3924 non-null   float64
 6   Long_                3924 non-null   float64
 7   Confirmed            4014 non-null   int64
 8   Deaths               4014 non-null   int64
 9   Recovered            737 non-null    float64
 10  Active               737 non-null    float64
 11  Combined_Key         4014 non-null   object
 12  Incident_Rate        3924 non-null   float64
 13  Case_Fatality_Ratio  3967 non-null   float64
dtypes: float64(7), int64(2), object(5)
memory usage: 439.2+ KB
```

```
FIPS                  748
Admin2                743
Province_State        179
Country_Region          0
Last_Update             0
Lat                    90
Long_                  90
Confirmed               0
Deaths                  0
Recovered            3277
Active               3277
Combined_Key            0
Incident_Rate          90
Case_Fatality_Ratio    47
dtype: int64
```

2. Get the latest number of confirmed, deaths, recovered and active cases of Novel Coronavirus (COVID-19) Country wise.

```
df.groupby(['Country_Region']).first()[['Confirmed', 'Deaths', 'Recovered', 'Active']]
```

Country_Region	Confirmed	Deaths	Recovered	Active
Afghanistan	59745	2625	53206.0	3914.0
Albania	131085	2394	109338.0	19353.0
Algeria	122108	3253	85108.0	33747.0
Andorra	13232	125	12684.0	423.0
Angola	26652	596	23876.0	2180.0
...	...	...	...	...
West Bank and Gaza	296462	3249	272333.0	20880.0
Winter Olympics 2022	0	0	0.0	0.0
Yemen	6317	1226	2751.0	2340.0
Zambia	91586	1251	89933.0	402.0
Zimbabwe	38257	1567	35612.0	1078.0

201 rows × 4 columns

3. get the latest number of confirmed deaths and recovered people of Novel Coronavirus (COVID-19) cases Country/Region - Province/State wise.

```
df.groupby(['Country_Region', 'Province_State']).first()[['Confirmed', 'Deaths', 'Recovered', 'Active']]
```

Country_Region	Province_State	Confirmed	Deaths	Recovered	Active
Australia	Australian Capital Territory	124	3	120.0	1.0
	New South Wales	5481	54	0.0	5427.0
	Northern Territory	166	0	115.0	51.0
	Queensland	1561	7	1507.0	47.0
	South Australia	724	4	687.0	33.0
...	...	...	...	...	...
United Kingdom	Saint Helena, Ascension and Tristan da Cunha	4	0	4.0	0.0
	Scotland	226052	10116	0.0	218393.0
	Turks and Caicos Islands	2388	17	2343.0	28.0
	Unknown	0	184	0.0	0.0
	Wales	211494	7914	0.0	205944.0

616 rows × 4 columns

4. get the top 10 countries data (Last Update, Country/Region, Confirmed, Deaths, Recovered) of Novel Coronavirus (COVID-19).

```
df.sort_values('Confirmed', ascending=False).head(10)[['Last_Update', 'Country_Region', 'Confirmed', 'Deaths', 'Recovered']]
```

	Last_Update	Country_Region	Confirmed	Deaths	Recovered
216	2021-05-01 04:20:47	France	5571804	103815	305692.0
650	2021-05-01 04:20:47	Turkey	4820591	40131	4323897.0
269	2021-05-01 04:20:47	India	4602472	68813	3868976.0
3962	2021-05-01 04:20:47	United Kingdom	3858882	132632	0.0
6	2021-05-01 04:20:47	Argentina	2977363	63865	2634306.0
65	2021-05-01 04:20:47	Brazil	2903709	96191	2552653.0
490	2021-05-01 04:20:47	Poland	2792142	67502	2496810.0
287	2021-05-01 04:20:47	Iran	2499077	71758	1954321.0
286	2021-05-01 04:20:47	Indonesia	1668368	45521	1522634.0
187	2021-05-01 04:20:47	Czechia	1630758	29267	1549439.0

5. list countries with no cases of Novel Coronavirus (COVID-19) recovered.

```
df[df['Recovered'] == 0]['Country_Region'].unique()
```

```
array(['Australia', 'Belgium', 'Canada', 'Colombia', 'Germany', 'India',
       'Mexico', 'Netherlands', 'Peru', 'Serbia', 'Spain', 'Sweden',
       'United Kingdom', 'Kiribati', 'Palau', 'New Zealand',
       'Summer Olympics 2020', 'Malaysia', 'Tonga',
       'Winter Olympics 2022', 'Antarctica', 'Korea, North', 'Ukraine',
       'Nauru', 'Tuvalu'], dtype=object)
```

6. get the latest country wise deaths cases of Novel Coronavirus (COVID- 19).

```
df[df['Deaths'] != 0].groupby(['Country_Region']).first()[['Deaths']].sort_values('Deaths', ascending=False)
```

Country_Region	Deaths
Iran	71758
Poland	67502
Argentina	63865
South Africa	54350
Indonesia	45521
...	...
MS Zaandam	2
Denmark	1
Vanuatu	1
Grenada	1
Bhutan	1

184 rows × 1 columns

Problem Statement 17:

The contemporary Olympic Games, sometimes known as the Olympics, are major international sporting events that feature summer and winter sports contests in which thousands of participants from all over the world compete in a range of disciplines. With over 200 nations competing, the Olympic Games are regarded as the world's premier sporting event. We have two CSV files when dealing with Olympic data. One detailing the total sports-related expenses of all Olympic Games. Another has information on athletes from all years who competed with information.

1. Write a python code to merge both datasets.

```
import pandas as pd


a = r"C:\Users\lucky\Downloads\athlete_events.csv"
b = r"C:\Users\lucky\Downloads\datasets_31029_40943_noc_regions.csv"

df1, df2 = pd.read_csv(a), pd.read_csv(b)
```

```
df = pd.merge(df1, df2, on='NOC')
```

```
df
```

2. Create a new data frame including only gold medallists.

```
df[df['Medal'] == 'Gold']
```

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games
68	17294	Cai Yalin	M	23.0	174.0	60.0	China	CHN	2000 Summer
77	17299	Cai Yun	M	32.0	181.0	68.0	China-1	CHN	2012 Summer
87	17995	Cao Lei	F	24.0	168.0	75.0	China	CHN	2008 Summer
104	18005	Cao Yuan	M	17.0	160.0	42.0	China	CHN	2012 Summer
105	18005	Cao Yuan	M	21.0	160.0	42.0	China	CHN	2016 Summer
...	...	...	...	...	...	...	...	...	...
269895	129665	Hannelore "Hanni" Wenzel (-Weirather)	F	23.0	165.0	57.0	Liechtenstein	LIE	1980 Winter

3. List medals we have only for women in the recent history of the Summer Games.

```
df[df['Medal'].notna()][(df['Sex'] == 'F') & (df['Year'] >= 2000)]
```

```
S:\Temp\ipykernel_22000\1489553856.py:1: UserWarning: Boolean Series key will be reindexed to match D
  df[df['Medal'].notna()][(df['Sex'] == 'F') & (df['Year'] >= 2000)]
```

	ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Season	City	
33	7597	Bao Yingying	F	24.0	172.0	67.0	China	CHN	2008 Summer	2008	Summer	Beijing	
63	17289	Cai Tongtong	F	18.0	168.0	48.0	China	CHN	2008 Summer	2008	Summer	Beijing	
87	17995	Cao Lei	F	24.0	168.0	75.0	China	CHN	2008 Summer	2008	Summer	Beijing	\
118	19779	Chang Si	F	25.0	170.0	56.0	China	CHN	2012 Summer	2012	Summer	London	S
186	20220	Chen Jing	F	28.0	182.0	75.0	China	CHN	2004 Summer	2004	Summer	Athina	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
269996	106233	Maja Savi	F	36.0	176.0	66.0	Montenegro	MNE	2012 Summer	2012	Summer	London	

4. Display top 5 Gold medallist countries

```
gold_medals_df = df[df['Medal'] == 'Gold']
gold_medals_df.groupby('NOC').size().sort_values(ascending=False).head(5)
```

```
NOC
USA    2638
URS    1082
GER     745
GBR     678
ITA     575
dtype: int64
```

5. count the medals per discipline for country taken as input

```
medals_per_discipline = df.groupby(['NOC', 'Event']).size()
medals_per_discipline_df = medals_per_discipline.reset_index(name='Count')
```

```
medals_per_discipline_df
```

	NOC	Event	Count
0	AFG	Athletics Men's 100 metres	7
1	AFG	Athletics Men's 110 metres Hurdles	1
2	AFG	Athletics Men's 200 metres	1
3	AFG	Athletics Men's 4 x 100 metres Relay	4
4	AFG	Athletics Men's 400 metres	1
...	...	...	...
28242	ZIM	Tennis Men's Singles	5
28243	ZIM	Tennis Women's Doubles	2
28244	ZIM	Tennis Women's Singles	4
28245	ZIM	Triathlon Men's Olympic Distance	3
28246	ZIM	Weightlifting Men's Heavyweight I	1

28247 rows × 3 columns

6. What is the median height/weight of an Olympic medallist?

```
df[df['Medal'].notna()]['Height'].median()
```

```
178.0
```

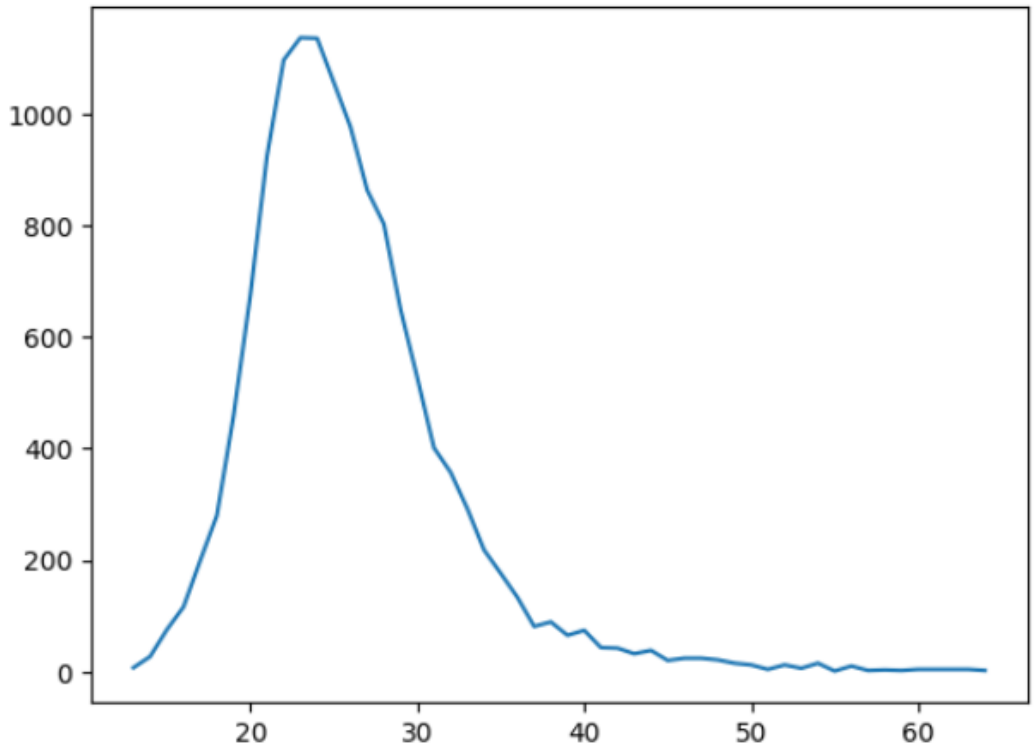
```
df[df['Medal'].notna()]['Weight'].median()
```

```
73.0
```

7. make a graph showing the number of gold medals in relation to age.

```
import matplotlib.pyplot as plt

gold_df = df[df['Medal'] == 'Gold']
medals_by_age = gold_df.groupby('Age').size()
plt.plot(medals_by_age)
plt.show()
```



Problem Statement 18:
From the given data set perform the following :

Read Data:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv(r"S:\Everything\mca3\nba.csv")
```

1. Remove all the missing values and NaN from the dataset.

```
df.dropna(inplace=True)
```

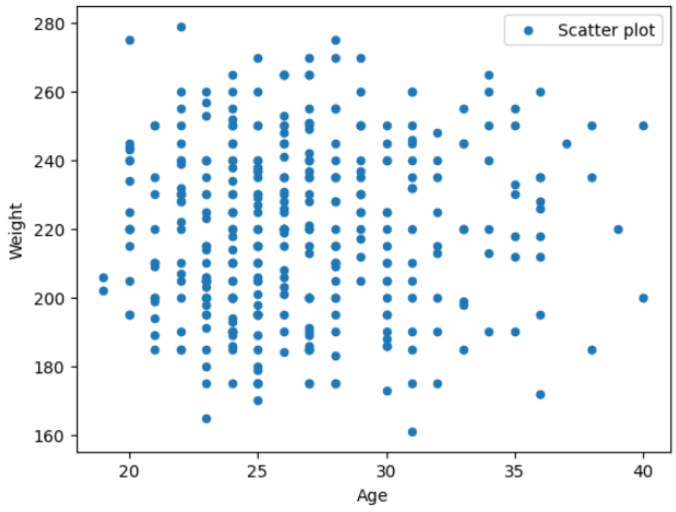
2. Find the average age for SG position.

```
df[df['Position'] == 'SG']['Age'].mean()
```

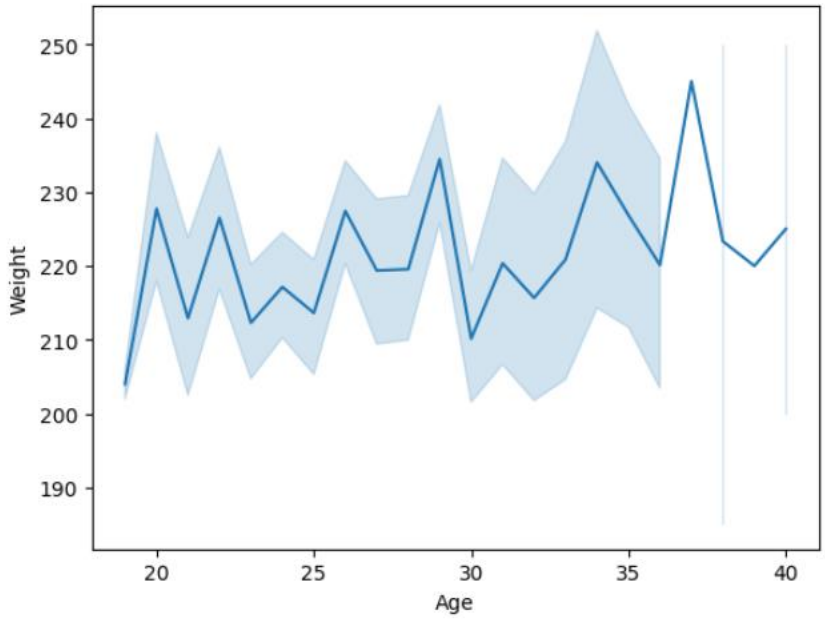
```
26.20689655172414
```

3. Plot the age vs weight graph using line and scatter plot

```
df.plot(x='Age', y='Weight', kind='scatter', label='Scatter plot')
plt.legend()
plt.show()
```

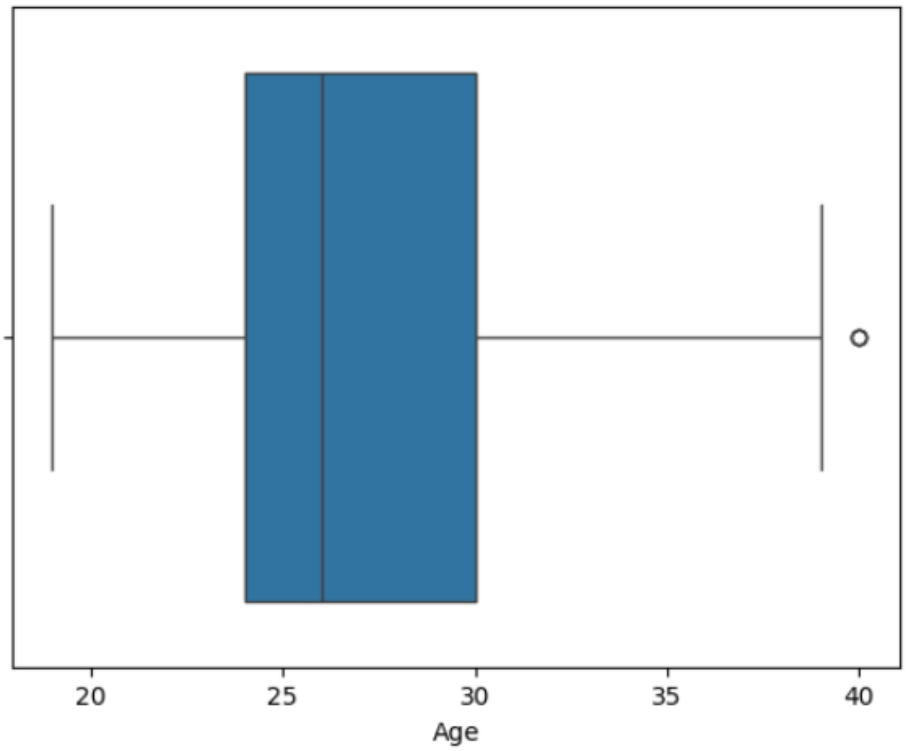


```
sns.lineplot(x='Age', y='Weight', data=df)
plt.show()
```



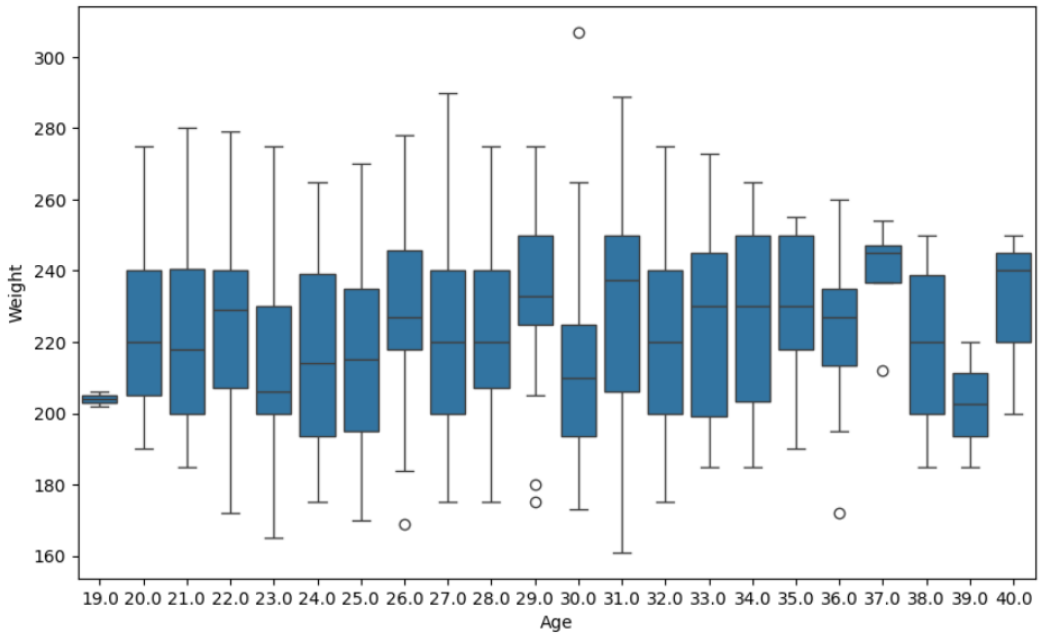
4. Display the box plot for age value using seaborn and pyplot

```
sns.boxplot(x=df['Age'])
plt.show()
```



5. Display the box plot for age and weight value using seaborn and pyplot

```
plt.figure(figsize=(10, 6))
sns.boxplot(x=df['Age'], y=df['Weight'])
plt.show()
```



Problem Statement 19:

From the given data set perform the following :

1. Write a Pandas program to find the sum, mean, max, min value of 'Production (short tons)' column of coalpublic2013.xlsx file.

```
import pandas as pd

df = pd.read_excel(r"S:\Everything\mca3\304 BIG DATA\coalpublic2013.xls", skiprows=3)

print("Sum: ",df["Production (short tons)"].sum())
print("Mean: ",df["Production (short tons)"].mean())
print("Maximum: ",df["Production (short tons)"].max())
print("Minimum: ",df["Production (short tons)"].min())
```

```
Sum:  984841779
Mean:  679201.2268965517
Maximum:  111005549
Minimum:  0
```

2. Write a Pandas program to import some excel data (coalpublic2013.xlsx) skipping first twenty rows into a Pandas dataframe.

```
pd.read_excel(r"S:\Everything\mca3\304 BIG DATA\coalpublic2013.xls", skiprows=20)
```

3. Write a Pandas program to create a subtotal of "Labor Hours" against MSHA ID from the given excel data

```
subtotal = df.groupby('MSHA ID')['Labor Hours'].sum()
subtotal
```

```
MSHA ID
100329      144002
100347      215295
100515        6240
100759      474784
100851     1001809
            ...
4801353    2811138
4801429     161270
4801645      35687
4801646     661265
5000030     286079
Name: Labor Hours, Length: 1321, dtype: int64
```

4. Write a Pandas program to import excel data (coalpublic2013.xlsx) into a dataframe and find details where "Mine Name" starts with "P".

```
df[df['Mine Name'].str.startswith('P').fillna(False)].head()
```

	Year	MSHA ID	Mine Name	Mine State	Mine County	Mine Status	Mine Type	Company Type	Operati Ty
13	2013	103332	Powhatan Mine	Alabama	Jefferson	Active	Surface	Indepedent Producer Operator	Mine or
18	2013	102976	Piney Woods Preparation Plant	Alabama	Shelby	Active	Surface	Indepedent Producer Operator	Preparati Pla
19	2013	102976	Piney Woods Preparation Plant	Alabama	Shelby	Active	Underground	Indepedent Producer Operator	Preparati Pla
46	2013	103321	Poplar Springs	Alabama	Winston	Active	Surface	Indepedent Producer Operator	Mine or
50	2013	301569	Penny #1	Arkansas	Sebastian	Temporarily closed	Surface	Indepedent Producer Operator	Mine or

Problem Statement 20:

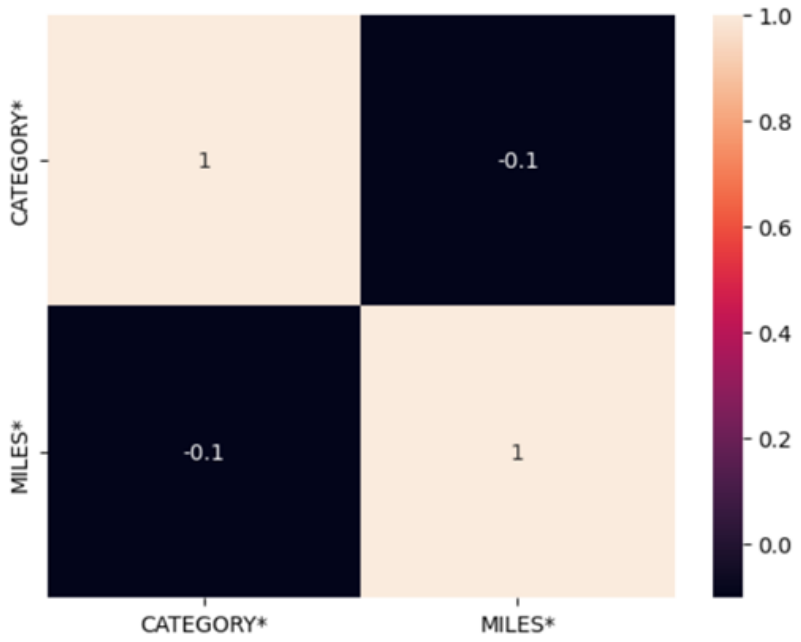
Uber Technologies, Inc., commonly known as Uber, is an American multinational ride-hailing company offering services that include peer-to-peer ridesharing, ride service hailing, food delivery (Uber Eats), and a micromobility system with electric bikes and scooters. The company is based in San Francisco and has operations in over 785 metropolitan areas worldwide. Its platforms can be accessed via its websites and mobile apps. There are more than 75 million active Uber riders accross the world. Uber is available in more than 80 countries worldwide. Uber has completed more than 5 billion rides till now. Over 3 million people drive for Uber. In the United States, Uber fulfills 40 million rides per month. The average Uber driver earns $364 per month.

https://www.kaggle.com/code/mohamed08/exploratory-data-analysis-for-uber-trips/input

This dataset contains 7 columns and 1156 entries.

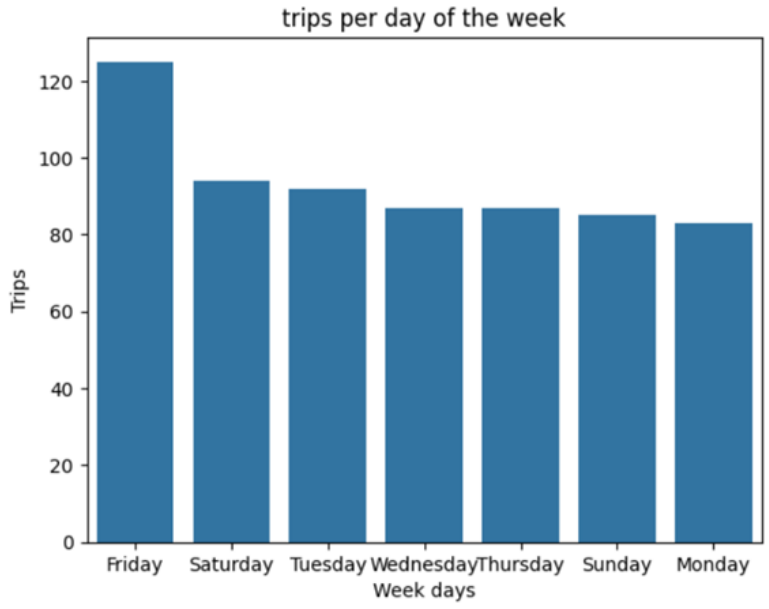
1. Drop/Remove the null values from the dataset and represent it in a heatmap.

```
df = pd.read_csv('My Uber Drives - 2016.csv')
df.dropna(inplace=True)
df['START_DATE*'] = pd.to_datetime(df['START_DATE*'], format="%m/%d/%Y %H:%M").dt.strftime('%Y/%m/%d %H:%M')
df['END_DATE*'] = pd.to_datetime(df['END_DATE*'], format='%m/%d/%Y %H:%M').dt.strftime('%Y/%m/%d %H:%M')
df['START_DATE*'] = pd.to_datetime(df['START_DATE*'], format='%Y/%m/%d %H:%M')
df['END_DATE*'] = pd.to_datetime(df['END_DATE*'], format='%Y/%m/%d %H:%M')
df['CATEGORY*'].replace({'Business' : 1, 'Personal' : 0}, inplace=True)
sns.heatmap(df.corr(numeric_only=True), annot=True)
```



2. Plot the trips per day of the week.

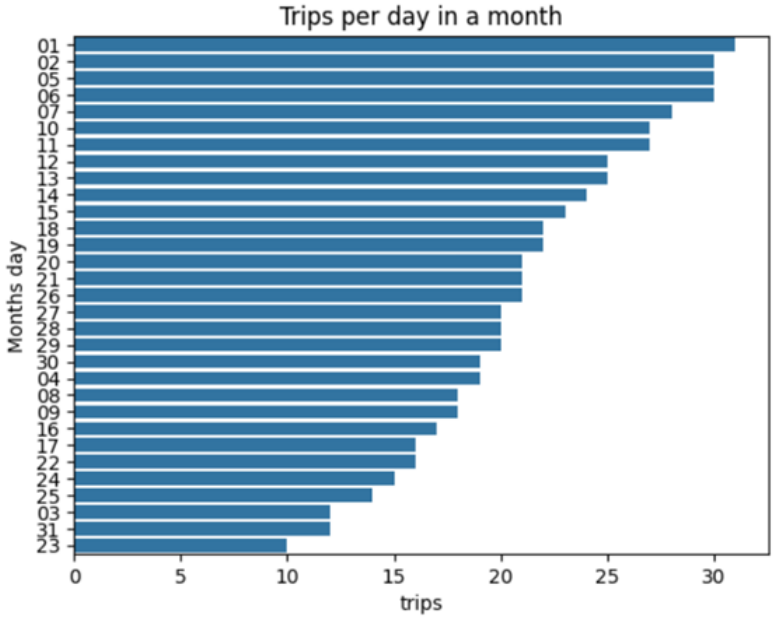
```
df['month'] = df['START_DATE*'].dt.month_name()
df['week'] = df['START_DATE*'].dt.day_name()
sns.barplot(x=df['week'].unique(), y=df['week'].value_counts()).set(xlabel='Week days', ylabel='Trips', title='trips per day of the week')
```



3. Plot the trips per day in a month.

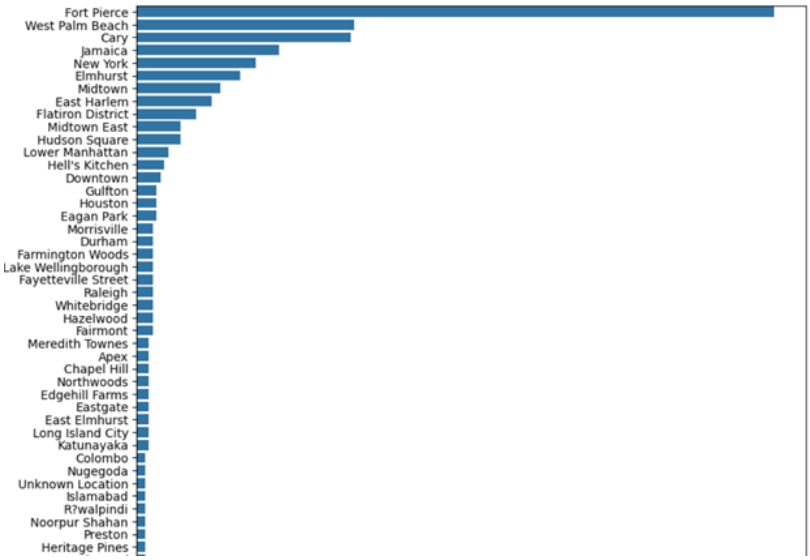
df['day'] = df['START_DATE*'].dt.strftime('%d')

sns.barplot(y=df['day'].unique(), x=df['day'].value_counts()).set(xlabel='trips', ylabel='Months day', title='Trips per day in a month')



4. Plotting the starting point of the trip

```
plt.figure(figsize=(10,25))
sns.barplot(y=df['START*'].unique(), x=df['START*'].value_counts())
```



5. Comparing the overall purpose with miles, hour, day of week, month, travel tim and speed.

```
df['travel(min)'] = (df['END_DATE*']-df['START_DATE*']).dt.total_seconds()/60
```

```
df['speed(Mph)'] = df['MILES*']/(df['travel(min)']/60)
df.head()
df['hour'] = df['START_DATE*'].dt.strftime('%H')
df.head()
df.groupby('PURPOSE*')[['MILES*', 'hour', 'week', 'month', 'travel(min)',
'speed(Mph)']].agg({'MILES*' : 'mean',  'hour' : 'unique',  'week' : 'unique',  'month' :
'unique',   'travel(min)' : 'sum', 'speed(Mph)' : 'mean'}).reset_index()
```

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*	month	week	day	travel(min)	speed(Mph)
0	2016-01-01 21:11:00	2016-01-01 21:17:00	1	Fort Pierce	Fort Pierce	5.1	Meal/Entertain	January	Friday	01	6.0	51.000000
2	2016-01-02 20:25:00	2016-01-02 20:38:00	1	Fort Pierce	Fort Pierce	4.8	Errand/Supplies	January	Saturday	02	13.0	22.153846
3	2016-01-05 17:31:00	2016-01-05 17:45:00	1	Fort Pierce	Fort Pierce	4.7	Meeting	January	Tuesday	05	14.0	20.142857
4	2016-01-06 14:42:00	2016-01-06 15:49:00	1	Fort Pierce	West Palm Beach	63.7	Customer Visit	January	Wednesday	06	67.0	57.044776
5	2016-01-06 17:15:00	2016-01-06 17:19:00	1	West Palm Beach	West Palm Beach	4.3	Meal/Entertain	January	Wednesday	06	4.0	64.500000

	START_DATE*	END_DATE*	CATEGORY*	START*	STOP*	MILES*	PURPOSE*	month	week	day	travel(min)	speed(Mph)	hour
0	2016-01-01 21:11:00	2016-01-01 21:17:00	1	Fort Pierce	Fort Pierce	5.1	Meal/Entertain	January	Friday	01	6.0	51.000000	21
2	2016-01-02 20:25:00	2016-01-02 20:38:00	1	Fort Pierce	Fort Pierce	4.8	Errand/Supplies	January	Saturday	02	13.0	22.153846	20
3	2016-01-05 17:31:00	2016-01-05 17:45:00	1	Fort Pierce	Fort Pierce	4.7	Meeting	January	Tuesday	05	14.0	20.142857	17
4	2016-01-06 14:42:00	2016-01-06 15:49:00	1	Fort Pierce	West Palm Beach	63.7	Customer Visit	January	Wednesday	06	67.0	57.044776	14
5	2016-01-06 17:15:00	2016-01-06 17:19:00	1	West Palm Beach	West Palm Beach	4.3	Meal/Entertain	January	Wednesday	06	4.0	64.500000	17

	PURPOSE*	MILES*	hour	week	month	travel(min)	speed(Mph)
0	Airport/Travel	5.500000	[10, 15, 13]	[Sunday, Saturday, Thursday]	[August, December]	78.0	13.531575
1	Between Offices	10.944444	[16, 15, 14, 21, 13, 17, 22, 01, 09, 12, 19, 1...	[Saturday, Monday, Tuesday, Sunday, Wednesday,...	[February, March, April, May, June, November, ...	459.0	25.622915
2	Charity ($)	15.100000	[11]	[Sunday]	[July]	27.0	33.555556
3	Commute	180.200000	[12]	[Sunday]	[July]	185.0	58.443243
4	Customer Visit	20.688119	[14, 12, 15, 16, 13, 10, 09, 18, 11, 03, 21, 1...	[Wednesday, Sunday, Tuesday, Thursday, Friday,...	[January, February, March, April, May, June, J...	3375.0	30.294273
5	Errand/Supplies	3.968750	[20, 11, 14, 15, 21, 00, 12, 16, 17, 18, 13, 0...	[Saturday, Monday, Tuesday, Thursday, Friday, ...	[January, February, March, April, May, June, J...	1661.0	19.463559
6	Meal/Entertain	5.698125	[21, 17, 13, 14, 12, 11, 16, 10, 09, 15, 19, 2...	[Friday, Wednesday, Monday, Tuesday, Thursday,...	[January, February, March, April, May, June, J...	2580.0	21.759906