VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

WEEK - 1
BASIC PYTHON PROGRAMS

1. Write a python program to display all the common characters between two strings. Return -1 if there are no matching characters.
Note: ignore blank spaces if there are any. Perform case sensitive string comparison whenever necessary.

PROGRAM:

```
def common_characters(str1, str2):
    x = set(str1.replace(" ", ""))
    y = set(str2.replace(" ", ""))
    common_chars = x.intersection(y)
    if common_chars:
        return sorted(list(common_chars))
    else:
        return -1

s1 = input("Enter the first string: ")
s2 = input("Enter the second string: ")
res = common_characters(s1, s2)

if res == -1:
    print("There are no common characters.")
else:
    print("Common characters:", res)

```

OUTPUT:

```
Enter the first string: HI HELLO
Enter the second string: HELLO
Common characters: ['E', 'H', 'L', 'O']

Process finished with exit code 0
```

2.Represent a small bilingual (English-swedish) glossary given below as python dictionary

{“merry”:”god”, ”Christmas”:”jul”, “and”:’och”, “happy”:”got”, “new”:”nytt”, “year”:”ar”}

And use to translate your Christmas wishes from English to Swedish.That is, write a python function translate() that accepts the bilingual dictionary and a list of English words(your Christmas wish) and returns a list of equivalent Swedish word.

PROGRAM:

```
def translate(glossary, english):
    result = ""
    for i in english:
        result += glossary[i]+" "
    return result

if __name__ == '__main__':
    glossary = {
        "merry":"god",
        "Christmas":"jul",
        "and":"och",
        "happy":"got",
        "new":"nytt",
        "year":"ar"
    }
    english = ["merry","Christmas"]
    swedish = translate(glossary,english)
    print(swedish)
```

OUTPUT:

```
god jul

Process finished with exit code 0
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

3.By using list comprehension write a program for the following: Print the list after removing numbers which are divisible by 3 and 5 in [5,10,30,7,5,3,31,12,101].

Print the list after removing the 0th 2nd 4th 6th numbers in [5,10,40,7,53,31,12,10].

Generate a 3*4 2-d list whose elements are initialized to zero.

Generate 2*3*5 3-d list whose elements are initialized to 0

PROGRAM:

```
if __name__ == '__main__':
    l = [5,10,30,7,5,3,31,12,101]
    l = [_ for _ in l if not(_ % 3 == 0 and _ % 5 == 0)]
    print(l)
    l1 = [[0 for _ in range(4)] for i in range(3)]
    print('3*4 2-D list : ',l1)
    l2 = [[[0 for _ in range(5)] for _ in range(3)] for i in range(2)]
    print('2*3*5 3-D list : ',l2 )
```

OUTPUT:

```
[5, 10, 7, 5, 3, 31, 12, 101]
3*4 2-D list :  [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
2*3*5 3-D list :  [[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]], [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]]

Process finished with exit code 0
```

4.Given a list of integer values. Write a python program to check whether it contains same number in adjacent positions. Display the count of such adjacent occurrences.

PROGRAM:

```
def adjacent_occurances(nums):
    count = 0
    for i in range(1,len(nums)):
        if nums[i] == nums[i-1]:
            count += 1
    return count

if __name__ == '__main__':
    nums = [1,2,2,3,4,4,5,5]
    adj_count = adjacent_occurances(nums)
    print('count of adjacent occurences : ' , adj_count)
```

OUTPUT:

```
count of adjacent occurences :  3

Process finished with exit code 0
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

5.Given a string containing uppercase characters, compress the string using RunLength Encoding. Repetition of character has to be replaced by storing the length os that run. Write a python function which performs the run length encoding for a given string and returns the run length encoded string.

PROGRAM:

```
def encoding(s):
    if len(s) == 0:
        return ""
    if len(s) == 1:
        return "1"+s==[0]
    result = ""
    count = 0
    for i in range(len(s)-1):
        count += 1
        if s[i] != s[i+1]:
            result += str(count)
            result += s[i]
            count = 0
    if s[i+1] == s[i]:
        count += 1
    else:
        count = 1
    result += str(count)
    result += s[i+1]
    return result

print('Enter the string : ')
s = input()
print('Run length of encoded string is : ', encoding(s))
```

OUTPUT:

```
Enter the string :
AAAAAABBBBBBBCCCCCCRRRRRRR
Run length of encoded string is :  6A7B6C7R
```

VNR VJIET

Name of the Laboratory:_____ **Name of the Experiment: _____**

Experiment No:_____Date: _____

WEEK 2

1)

Given a string containing uppercase characters (A-Z), compress the string using Run Length encoding. Repetition of character has to be replaced by storing the length of that run.

Write a python function which performs the run length encoding for a given String and returns the run length encoded String.

Provide different String values and test your program

Sample Input	Expected Output
AAAABBBBCCCCCCCC	4A4B8C

Program

```
C: > Users > LENOVO > ml1.py > ...
1   def res(s):
2       cnt=""
3       l = list(s)
4       for i in l:
5           if i not in cnt:
6               cnt += str(l.count(i))
7               cnt += i
8       return cnt
9   s=input("Enter a string:\n")
10  print("Output is:",res(s))
11
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE

PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/LENOVO/ml1.py
Enter a string:
AAAABBBBCCCDDD
Output is: 4A4B3C3D
PS C:\Users\LENOVO>
```

VNR VJIET

Name of the Laboratory:_____ **Name of the Experiment: _____**

Experiment No:_____Date: _____

2)

Write a python function, create_largest_number(), which accepts a list of numbers and returns the largest number possible by concatenating the list of numbers.☐
Note: Assume that all the numbers are two digit numbers.

Sample Input	Expected Output
23,34,55	553423

Program

```
C: > Users > LENOVO > ml1.py > ...
1   def res(l):
2       l.sort(reverse=True)
3       s=""
4       l=[str(i) for i in l]
5       return s.join((l))
6   print("Enter the numbers:")
7   l=list(map(int,input().split()))
8   print("Ouput is:",int(res(l)))
9
```

Output:

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/LENOVO/ml1.py
Enter the numbers:
23 35 55
Ouput is: 553523
PS C:\Users\LENOVO>
```

VNR VJIET

Name of the Laboratory:_____ **Name of the Experiment: _____**

Experiment No:_____Date: _____

3)

Care hospital wants to know the medical speciality visited by the maximum number of patients. Assume that the patient id of the patient along with the medical speciality visited by the patient is stored in a list. The details of the medical specialities are stored in a dictionary as follows:

```
{
"P":"Pediatrics",
"O":"Orthopedics",
"E":"ENT
}
```

Write a function to find the medical speciality visited by the maximum number of patients and return the name of the speciality.

Note:

1. Assume that there is always only one medical speciality which is visited by maximum number of patients.
2. Perform case sensitive string comparison wherever necessary.

Sample Input	Expected Output
[101,P,102,O,302,P,305,P]	Pediatrics
[101,O,102,O,302,P,305,E,401,O,656,O]	Orthopedics
[101,O,102,E,302,P,305,P,401,E,656,O,987,E]	ENT

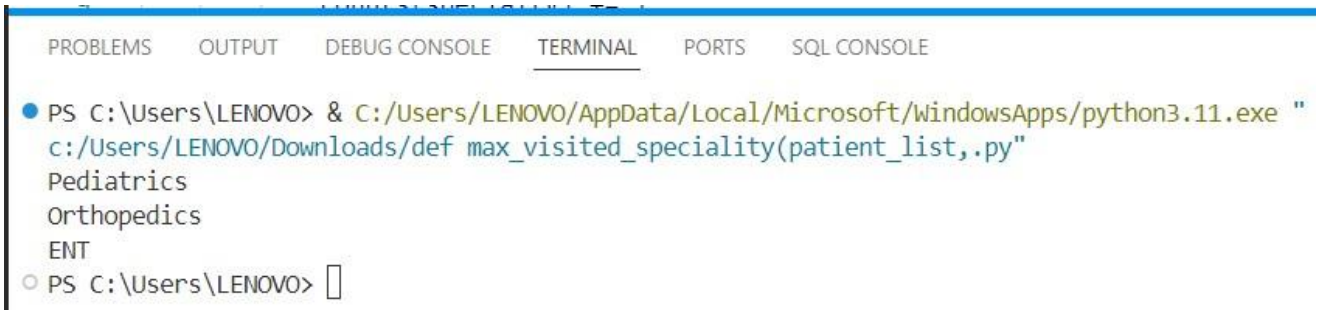
Program:

def max_visited_speciality(patient_list,.py

C: > Users > LENOVO > Downloads > def max_visited_speciality(patient_list,.py > ...

```python
def max_visited_speciality(patient_list, specialties):
    counts = {}
    for i in range(1, len(patient_list), 2):
        patient_id = patient_list[i]
        if patient_id in specialties:
            specialty = specialties[patient_id]
            if specialty not in counts:
                counts[specialty] = 0
            counts[specialty] += 1

    max_specialty = max(counts, key=counts.get)
    return max_specialty
# Input
patient_list_1 = [101, "P", 102, 0, 302, "P", 305, "P"]
patient_list_2 = [101, 0, 102, "E", 302, "P", 305, "O", 401, "O", 656, 0, 987, "O"]
patient_list_3 = [101, 0, 102, "E", 302, "P", 305, "P", 401, "E", 656, 0, 987, "E"]
specialties = {"P": "Pediatrics", "O": "Orthopedics", "E": "ENT"}
# Output
print(max_visited_speciality(patient_list_1, specialties))
print(max_visited_speciality(patient_list_2, specialties))
print(max_visited_speciality(patient_list_3, specialties))
```

Output:



4)

Write a python function, **find_correct()** which accepts a dictionary and returns a list as per the rules mentioned below.
The input dictionary will contain correct spelling of a word as key and the spelling provided by a contestant as the value.

The function should identify the degree of correctness as mentioned below:
CORRECT, if it is an exact match
ALMOST CORRECT, if no more than 2 letters are wrong
WRONG, if more than 2 letters are wrong or if length (correct spelling versus spelling given by contestant) mismatches.

and return a list containing the number of CORRECT answers, number of ALMOST CORRECT answers and number of WRONG answers.
Assume that the words contain only uppercase letters and the maximum word length is 10.

Sample Input	Expected Output
{"THEIR": "THEIR", "BUSINESS": "BISINESS","WINDOWS":"WINDMILL","WERE":"WEAR","SAMPLE":"SAMPLE"}	[2, 2, 1]

Program:

word count.py

C: > Users > LENOVO > Downloads > word count.py > ...

```python
def find_correct(word_dict):
    correct = 0
    almost_correct = 0
    wrong = 0
    for correct_word, contestant_word in word_dict.items():
        if correct_word == contestant_word:
            correct += 1
        elif len(correct_word) == len(contestant_word):
            diff_count = sum(c1 != c2 for c1, c2 in zip(correct_word, contestant_word))
            if diff_count <= 2:
                almost_correct += 1
            else:
                wrong += 1
        else:
            wrong += 1
    return [correct, almost_correct, wrong]
word_dict = {"THEIR": "THEIR", "BUSINESS": "BISINESS",
             "WINDOWS": "WINDMILL", "WERE": "WEAR", "SAMPLE": "SAMPLE"}
print("Input is:",word_dict)
print("Ouput is:",find_correct(word_dict))

```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE

PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/LENOVO/Downloads/word count.py"
Input is: {'THEIR': 'THEIR', 'BUSINESS': 'BISINESS', 'WINDOWS': 'WINDMILL', 'WERE': 'WEAR', 'SAMPLE': 'SAMPLE'}
Ouput is: [2, 2, 1]
PS C:\Users\LENOVO>
```

5)

Write a python function, **encrypt_sentence()** which accepts a message and encrypts it based on rules given below and returns the encrypted message.☐
Words at odd position -> Reverse It
Words at even position -> Rearrange the characters so that all consonants appear before the vowels and their order should not change

Note:

1. Assume that the sentence would begin with a word and there will be only a single space between the words.
2. Perform case sensitive string operations wherever necessary.

Sample Input	Expected Output
the sun rises in the east	eht snu sesir ni eht stea

VNR VJIET

Name of the Laboratory:_____ **Name of the Experiment:** _____

Experiment No:_____**Date:** _____

Program:

word count.py

C: > Users > LENOVO > Downloads > word count.py > ...

```
def encrypt_sentence(message):
    def rearrange_word(word):
        vowels = "aeiouAEIOU"
        consonants = "".join([c for c in word if c not in vowels])
        vowels = "".join([c for c in word if c in vowels])
        return consonants + vowels
    words = message.split()
    encrypted_words = []
    for i, word in enumerate(words):
        if i % 2 == 0:
            encrypted_words.append(word[::-1])
        else:
            encrypted_words.append(rearrange_word(word))
    return " ".join(encrypted_words)
message = input("Enter the input:")
print("Input is:",message)
print("Output is:",encrypt_sentence(message))
```

Output:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL CONSOLE

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe "
c:/Users/LENOVO/Downloads/word count.py"
Enter the input:the sun rises in the east
Input is: the sun rises in the east
Output is: eht snu sesir ni eht stea
PS C:\Users\LENOVO>
```

Name of the Laboratory:_____ **Name of the Experiment:** _____

Experiment No:_____ **Date:** _____

6)

The flight ticket rates for a round-trip (Mumbai->Dubai) were as follows:
Rate per Adult: Rs. 37550.0
Rate per Child: 1/3rd of the rate per adult
Service Tax: 7% of the ticket amount (including all passengers)
As it was a holiday season, the airline also offered 10% discount on the final ticket cost (after inclusion of the service tax).
Find and display the total ticket cost for a group which had adults and children.

Test the program with different input values for number of adults and children.

Sample Input		Expected Output
Number of adults	Number of children	
5	2	Total Ticket Cost: 204910.35
3	1	Total Ticket Cost: 120535.5

Program:

word count.py

C: > Users > LENOVO > Downloads > word count.py > ...

```
def calculate_ticket_cost(num_adults, num_children):
    rate_per_adult = 37550.0
    rate_per_child = rate_per_adult / 3
    service_tax_rate = 0.07
    discount_rate = 0.10
    total_adult_cost = rate_per_adult * num_adults
    total_child_cost = rate_per_child * num_children
    subtotal = total_adult_cost + total_child_cost
    service_tax = subtotal * service_tax_rate
    total_cost = subtotal + service_tax
    discount_amount = total_cost * discount_rate
    final_cost = total_cost - discount_amount
    return final_cost
num_adults = int(input("Number of adults: "))
num_children = int(input("Number of children: "))
total_cost = calculate_ticket_cost(num_adults, num_children)
print("Total Ticket Cost: {:.2f}".format(total_cost))
```

VNR VJIET

Name of the Laboratory:_____ **Name of the Experiment:** _____

Experiment No:_____**Date:** _____

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SQL CONSOLE

PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe "
c:/Users/LENOVO/Downloads/word count.py"
Number of adults: 5
Number of children: 2
Total Ticket Cost: 204910.35
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe "
c:/Users/LENOVO/Downloads/word count.py"
Number of adults: 2
Number of children: 1
Total Ticket Cost: 84374.85
PS C:\Users\LENOVO>
```

7)

Write a python program to solve a classic ancient Chinese puzzle.

We count 35 heads and 94 legs among the chickens and rabbits in a farm. How many rabbits and how many chickens do we have?

Sample Input	Expected Output
heads-150 legs-400	100 50
heads-3 legs-11	No solution
heads-3 legs-12	0 3
heads-5 legs-10	5 0

Program:

word count.py

C: > Users > LENOVO > Downloads > word count.py > ...

```
def solve_puzzle(total_heads, total_legs):
    for num_chickens in range(total_heads + 1):
        num_rabbits = total_heads - num_chickens
        if (2 * num_chickens + 4 * num_rabbits) == total_legs:
            return num_chickens, num_rabbits
    return None, None
inputs = [(150, 400),(3, 11),(3, 12),(5, 10)]
print("Input is:",inputs)
print("Output is:")
for total_heads, total_legs in inputs:
    print("heads-{} legs-{}".format(total_heads, total_legs))
    chickens, rabbits = solve_puzzle(total_heads, total_legs)
    if chickens is not None and rabbits is not None:
        print(chickens, rabbits)
    else:
        print("No solution")

```

Output:

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe "
c:/Users/LENOVO/Downloads/word count.py"
Input is: [(150, 400), (3, 11), (3, 12), (5, 10)]
Output is:
heads-150 legs-400
100 50
heads-3 legs-11
No solution
heads-3 legs-12
0 3
heads-5 legs-10
5 0
```

8)

Write a python program which finds the maximum number from num1 to num2 (num2 inclusive) based on the following rules.

1. Always num1 should be less than num2
2. Consider each number from num1 to num2 (num2 inclusive). Populate the number into a list, if the below conditions are satisfied
   a. Sum of the digits of the number is a multiple of 3
   b. Number has only two digits
   c. Number is a multiple of 5
3. Display the maximum element from the list

In case of any invalid data or if the list is empty, display -1.

Program:

C: > Users > LENOVO > Downloads > word count.py > ...

```python
def find_max_number(num1, num2):
    if num1 >= num2:
        return -1
    numbers = []
    for num in range(num1, num2 + 1):
        if num % 5 == 0 and 10 <= num <= 99:
            digit_sum = sum(int(digit) for digit in str(num))
            if digit_sum % 3 == 0:
                numbers.append(num)
    if not numbers:
        return -1
    return max(numbers)
num1 = int(input("Enter num1: "))
num2 = int(input("Enter num2: "))
print("Input is:",num1,num2)
max_number = find_max_number(num1, num2)
print("Maximum number satisfying the conditions:", max_number)
```

VNR VJIET

Name of the Laboratory:_____ **Name of the Experiment:** _____

Experiment No:_____**Date:** _____

Output:

```
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe "
c:/Users/LENOVO/Downloads/word count.py"
Enter num1: 30
Enter num2: 50
Input is: 30 50
Maximum number satisfying the conditions: 45
PS C:\Users\LENOVO> & C:/Users/LENOVO/AppData/Local/Microsoft/WindowsApps/python3.11.exe "
c:/Users/LENOVO/Downloads/word count.py"
Enter num1: 100
Enter num2: 36
Input is: 100 36
Maximum number satisfying the conditions: -1
PS C:\Users\LENOVO>
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

WEEK-3

A) Pandas Basic Functions

```
import pandas as pd
import numpy as np
pd.__version__
```

```
'1.5.3'
```

1) Create a series of data using pandas.

Program-

```
data=pd.Series([5,2,0,-1,6])
data
```

Output-

```
0    5
1    2
2    0
3   -1
4    6
dtype: int64
```

2) Convert a dictionary into a series.

Program-

```
population_dict={'Hyderabad':12345,'Bangalore':67890,'Mumbai':13457,'Pune':24678,'Noida':854631}
population=pd.Series(population_dict)
population
```

Output-

```
Hyderabad      12345
Bangalore      67890
Mumbai         13457
Pune           24678
Noida         854631
dtype: int64
```

3) Find the null values in the series.

Program-

```
[ ]  data.isnull()
```

Output-

```
0    False
1    False
2    False
3    False
4    False
dtype: bool
```

4) Find the total number of null values.

Program-

```
data.isnull().sum()
```

Output-

```
0
```

5) Drop null values in the given data.

Program-

```
data2=data.dropna()
data2
```

Output-

```
0    1.0
1    2.0
2    3.0
3    4.0
4    5.0
8    9.0
dtype: float64
```

6) Fill Null values with 0.

Program-

```
data.fillna(0) #fill nans with zeroes
```

Output-

```
0    1.0
1    2.0
2    3.0
3    4.0
4    5.0
5    0.0
6    0.0
7    0.0
8    9.0
dtype: float64
```

7) Find the cumulative sum of the series.

Program-

```
data.cumsum() # cummulative sum
```

Output-

```
0     1.0
1     3.0
2     6.0
3    10.0
4    15.0
5     NaN
6     NaN
7     NaN
8    24.0
dtype: float64
```

8) Describe the data using pandas.

Program-

```
data.describe()
```

Output-

```
count    6.000000
mean     4.000000
std      2.828427
min      1.000000
25%      2.250000
50%      3.500000
75%      4.750000
max      9.000000
dtype: float64
```

9) Find the Dimensions of the data.

Program-

```
[ ]  data.ndim
```

Output-

```
1
```

10) Change the indexes of the data into alphabets.

Program-

```
data.index=['a','b','c','d','e']
data
```

Output-

```
a    5
b    2
c    0
d   -1
e    6
dtype: int64
```

B) Importing Dataset and Working on it

1) Import a dataset using pandas.

Program-

```
[ ] import pandas as pd
    ds=pd.read_csv('C:/Users/B314-CSE-SYS18/Downloads/deepak sir dataset.csv')
```

```
ds
```

Output-

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0
...	...	...	...	...	...	...	...	...	...
762	10	101	76	48	180	32.9	0.171	63	0
763	2	122	70	27	0	36.8	0.340	27	0
764	5	121	72	23	112	26.2	0.245	30	0
765	1	126	60	0	0	30.1	0.349	47	1
766	1	93	70	31	0	30.4	0.315	23	0

767 rows × 9 columns

2) Describe the no of rows and columns.

Program-

```
[ ] ds.shape #describe no. of rows and columns
```

Output-

```
(767, 9)
```

3) Display the first 5 rows.

Program-

```
ds.head(5) #first 5 rows
```

Output-

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

4) Display the last 5 rows

Program-

```
[ ] ds.tail(5) #last 5 rows
```

Output-

	6	148	72	35	0	33.6	0.627	50	1
762	10	101	76	48	180	32.9	0.171	63	0
763	2	122	70	27	0	36.8	0.340	27	0
764	5	121	72	23	112	26.2	0.245	30	0
765	1	126	60	0	0	30.1	0.349	47	1
766	1	93	70	31	0	30.4	0.315	23	0

5) Display column labels of the data frame

Program-

```
[ ] ds.columns
```

Output-

```
Index(['6', '148', '72', '35', '0', '33.6', '0.627', '50', '1'], dtype='object')
```

6) Display the data in the first cell

Program-

```
[ ] rainfall.iloc[0,0]
```

Output-

```
'ANDAMAN And NICOBAR ISLANDS'
```

7) Display the data in the first column.

Program-

```
rainfall.iloc[:,0]
```

Output-

```
0      ANDAMAN And NICOBAR ISLANDS
1      ANDAMAN And NICOBAR ISLANDS
2      ANDAMAN And NICOBAR ISLANDS
3                ARUNACHAL PRADESH
4                ARUNACHAL PRADESH
                   ...
636                        KERALA
637                        KERALA
638                        KERALA
639                        KERALA
640                    LAKSHADWEEP
Name: STATE_UT_NAME, Length: 641, dtype: object
```

8) Display the data between the rows 5 and 13.

Program-

```
[ ] rainfall.iloc[5:13]
```

Output-

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
5	ARUNACHAL PRADESH	SUBANSIRI F.D	28.0	48.3	85.3	101.5	140.5	228.4	217.4	182.8	159.8	75.9	20.9	11.6	1300.4	76.3	327.3	788.4	108.4
6	ARUNACHAL PRADESH	TIRAP	42.2	72.7	141.0	316.9	328.7	614.7	851.9	500.6	418.3	218.7	42.9	22.9	3571.5	114.9	786.6	2385.5	284.5
7	ARUNACHAL PRADESH	ANJAW (LOHIT)	42.2	80.8	176.4	358.5	306.4	447.0	660.1	427.8	313.6	167.1	34.1	29.8	3043.8	123.0	841.3	1848.5	231.0
8	ARUNACHAL PRADESH	LOWER DIBANG	83.7	153.9	303.5	383.6	268.0	374.2	272.0	160.5	266.7	167.2	64.0	56.0	2553.3	237.6	955.1	1073.4	287.2
9	ARUNACHAL PRADESH	CHANGLANG	70.3	170.9	367.9	554.4	334.2	526.2	460.8	291.5	353.6	275.0	64.9	74.2	3543.9	241.2	1256.5	1632.1	414.1
10	ARUNACHAL PRADESH	PAPUM PARE	33.5	67.8	106.1	226.9	453.0	640.5	609.5	503.4	492.3	214.7	19.2	11.3	3378.2	101.3	786.0	2245.7	245.2
11	ARUNACHAL PRADESH	LOW SUBANSIRI	97.5	109.3	92.4	204.3	266.2	284.1	248.9	270.5	192.7	78.5	49.5	27.2	1921.1	206.8	562.9	996.2	155.2
12	ARUNACHAL PRADESH	UPPER SIANG	74.3	176.7	362.6	397.5	408.7	801.9	653.0	417.9	686.0	264.9	86.9	71.7	4402.1	251.0	1168.8	2558.8	423.5

9) Display the count of each state in the data frame.

Program-

```
rainfall[['STATE_UT_NAME']].value_counts()
```

Output-

```
STATE_UT_NAME
UTTAR PRADESH                  71
MADHYA PRADESH                 50
BIHAR                          38
MAHARASHTRA                    35
RAJASTHAN                      33
TAMIL NADU                     32
KARNATAKA                      30
ORISSA                         30
ASSAM                          27
GUJARAT                        26
JHARKHAND                      24
ANDHRA PRADESH                 23
JAMMU AND KASHMIR              22
HARYANA                        21
PUNJAB                         20
WEST BENGAL                    19
CHATISGARH                     18
ARUNACHAL PRADESH              16
KERALA                         14
UTTARANCHAL                    13
HIMACHAL                       12
NAGALAND                       11
MANIPUR                         9
MIZORAM                         9
DELHI                           9
MEGHALAYA                       7
PONDICHERRY                     4
SIKKIM                          4
TRIPURA                         4
ANDAMAN And NICOBAR ISLANDS     3
GOA                             2
DAMAN AND DUI                   2
LAKSHADWEEP                     1
DADAR NAGAR HAVELI              1
CHANDIGARH                      1
dtype: int64
```

10) Display Rainfall data of Arunachal Pradesh.

Program-

```
[ ]  rainfall.loc[rainfall['STATE_UT_NAME']=='ARUNACHAL PRADESH']
```

Output-

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	ANNUAL	Jan-Feb	Mar-May	Jun-Sep	Oct-Dec
3	ARUNACHAL PRADESH	LOHIT	42.2	80.8	176.4	358.5	306.4	447.0	660.1	427.8	313.6	167.1	34.1	29.8	3043.8	123.0	841.3	1848.5	231.0
4	ARUNACHAL PRADESH	EAST SIANG	33.3	79.5	105.9	216.5	323.0	738.3	990.9	711.2	568.0	206.9	29.5	31.7	4034.7	112.8	645.4	3008.4	268.1
5	ARUNACHAL PRADESH	SUBANSIRI F.D	28.0	48.3	85.3	101.5	140.5	228.4	217.4	182.8	159.8	75.9	20.9	11.6	1300.4	76.3	327.3	788.4	108.4
6	ARUNACHAL PRADESH	TIRAP	42.2	72.7	141.0	316.9	328.7	614.7	851.9	500.6	418.3	218.7	42.9	22.9	3571.5	114.9	786.6	2385.5	284.5
7	ARUNACHAL PRADESH	ANJAW (LOHIT)	42.2	80.8	176.4	358.5	306.4	447.0	660.1	427.8	313.6	167.1	34.1	29.8	3043.8	123.0	841.3	1848.5	231.0
8	ARUNACHAL PRADESH	LOWER DIBANG	83.7	153.9	303.5	383.6	268.0	374.2	272.0	160.5	266.7	167.2	64.0	56.0	2553.3	237.6	955.1	1073.4	287.2
9	ARUNACHAL PRADESH	CHANGLANG	70.3	170.9	367.9	554.4	334.2	526.2	460.8	291.5	353.6	275.0	64.9	74.2	3543.9	241.2	1256.5	1632.1	414.1
10	ARUNACHAL PRADESH	PAPUM PARE	33.5	67.8	106.1	226.9	453.0	640.5	609.5	503.4	492.3	214.7	19.2	11.3	3378.2	101.3	786.0	2245.7	245.2
11	ARUNACHAL PRADESH	LOW SUBANSIRI	97.5	109.3	92.4	204.3	266.2	284.1	248.9	270.5	192.7	78.5	49.5	27.2	1921.1	206.8	562.9	996.2	155.2
12	ARUNACHAL PRADESH	UPPER SIANG	74.3	176.7	362.6	397.5	408.7	801.9	653.0	417.9	686.0	264.9	86.9	71.7	4402.1	251.0	1168.8	2558.8	423.5
13	ARUNACHAL PRADESH	WEST SIANG	26.0	66.7	76.8	229.2	239.5	416.6	592.4	312.4	291.1	126.8	33.7	29.5	2440.7	92.7	545.5	1612.5	190.0
14	ARUNACHAL PRADESH	DIBANG VALLEY	83.7	153.9	303.5	383.6	268.0	374.2	272.0	160.5	266.7	167.2	64.0	56.0	2553.3	237.6	955.1	1073.4	287.2
15	ARUNACHAL PRADESH	WEST KAMENG	35.2	43.5	58.9	134.3	341.1	665.3	749.9	579.1	490.9	233.9	40.3	27.0	3399.4	78.7	534.3	2485.2	301.2
16	ARUNACHAL PRADESH	EAST KAMENG	49.0	74.4	96.5	156.9	208.0	345.7	368.5	256.2	275.9	138.2	34.4	27.2	2030.9	123.4	461.4	1246.3	199.8
17	ARUNACHAL PRADESH	TAWANG(W KAME	35.2	43.5	58.9	134.3	341.1	665.3	749.9	579.1	490.9	233.9	40.3	27.0	3399.4	78.7	534.3	2485.2	301.2
18	ARUNACHAL PRADESH	KURUNG KUMEY	82.7	70.0	128.2	245.7	271.4	292.7	404.0	276.3	283.5	92.3	32.3	42.4	2221.5	152.7	645.3	1256.5	167.0

C) Handling Missing Values

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

```
[ ]  df = pd.read_csv('C:/Users/pvrao/deepak sir dataset.csv')
```

1) If there are missing values: examine them, then fill them in with 0

Program-

```
#fill with zeros
df.fillna(0,inplace=True)
#df_filled_zeros = df2018.fillna(0)
```

Output-

```
[ ]  features_with_na = [features for features in df.columns if df[features].isnull().sum() > 1]
     print("Total number of features with missing values:",len(features_with_na))

     Total number of features with missing values: 0
```

2) Fill the missing values using forward filling

Program-

```
[ ]  #filling with before values in terms of columns
     df = df.fillna(method='ffill', axis=1)# check with axis = 0 in terms of row
```

Output-

```
[ ]  features_with_na = [features for features in df.columns if df[features].isnull().sum() > 1]
     print("Total number of features with missing values:",len(features_with_na))

     Total number of features with missing values: 1
```

3) Fill the missing values using backward filling

Program

```
[ ]  #filling with before values in terms of columns
     df = df.fillna(method='bfill', axis=1)# check with axis = 0 in terms of row
```

Output-

```
[ ]  features_with_na = [features for features in df.columns if df[features].isnull().sum() > 1]
     print("Total number of features with missing values:",len(features_with_na))

     Total number of features with missing values: 1
```

4) Fill the missing values in column-1 using mean of column-1

Program-

```
[ ]  df['c1'].fillna(df['c1'].mean(),inplace= True)
```

```
#check all missing values of Age column are filled
df.isna().sum()
```

Output-

```
c1      0
c2      5
c3     35
c4    227
c5    373
c6     11
c7      0
c8      0
c9    500
dtype: int64
```

5) Fill the missing values in column-2 using median of column-2

Program-

```
[ ]  df['c2'].fillna(df['c2'].median(),inplace= True)
```

```
#check all missing values of Age column are filled
df.isna().sum()
```

Output-

```
c1      0
c2      0
c3     35
c4    227
c5    373
c6     11
c7      0
c8      0
c9    500
dtype: int64
```

6) Fill the missing values in column-3 using median of column-3

Program-

```
[ ]  #Imputing missing values of 'c3':
     df['c3'].mode()
```

```
0    70.0
dtype: float64
```

```
[ ]  df['c3'].fillna(df['c3'].mode()[0],inplace= True)
```

```
#check all missing values of Age column are filled
df.isna().sum()
```

Output-

```
c1      0
c2      0
c3      0
c4    227
c5    373
c6     11
c7      0
c8      0
c9    500
dtype: int64
```

7) Fill the missing values using KNN imputer

Program-

```
[ ]  impute = KNNImputer()
```

```
[ ]  df = pd.DataFrame(impute.fit_transform(df), columns = df.columns)
```

```
features_with_na = [features for features in df.columns if df[features].isnull().sum() > 1]
print("Total number of features with missing values:",len(features_with_na))
```

Output-

```
Total number of features with missing values: 0
```

8) Fill the missing values using simple imputer

Program-

```
from sklearn.impute import SimpleImputer
```

```
impute = SimpleImputer()
```

```
df = pd.DataFrame(impute.fit_transform(df), columns = df.columns)
```

```
features_with_na = [features for features in df.columns if df[features].isnull().sum() > 1]
print("Total number of features with missing values:",len(features_with_na))
```

Output-

```
Total number of features with missing values: 0
```

9) Fill the missing values using missing indicator

Program-

```
from sklearn.impute import MissingIndicator
```

```
impute = MissingIndicator()
```

```
df = pd.DataFrame(impute.fit_transform(df))
```

```
features_with_na = [features for features in df.columns if df[features].isnull().sum() > 1]
print("Total number of features with missing values:",len(features_with_na))
```

Output-

```
Total number of features with missing values: 0
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

10) Fill the missing values using Iterative Imputer

Program-

```
from sklearn.impute import IterativeImputer
```

```
impute = IterativeImputer()
```

```
df = pd.DataFrame(impute.fit_transform(df), columns = df.columns)
```

```
features_with_na = [features for features in df.columns if df[features].isnull().sum() > 1]
print("Total number of features with missing values:",len(features_with_na))
```

Output-

```
Total number of features with missing values: 0
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

WEEK - 4

WORKING WITH DATASETS:

```
import pandas as pd
import numpy as np
df = pd.read_csv('auto_mpg.csv')
df
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
...	...	...	...	...	...	...	...	...	...
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford ranger
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s-10

1) USING HEAD AND TAIL FUNCTIONS:

PROGRAM:

```
df.head()
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

PROGRAM:

```
[ ]  df.tail()
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford ranger
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s-10

2) GIVE THE SUMMARY OF THE DATASET:

PROGRAM:

```
df.describe()
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

3) WRITE PROGRAM FOR DROPPING NULL VALUES?

PROGRAM:

```
[ ]  df.dropna(inplace = True)
     df.info()
```

OUTPUT:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        392 non-null    int64
 5   acceleration  392 non-null    float64
 6   model year    392 non-null    int64
 7   origin        392 non-null    int64
 8   car name      392 non-null    object
dtypes: float64(4), int64(4), object(1)
memory usage: 30.6+ KB
```

4) WRITE A PROGRAM FOR SETTING CUSTOM INDEX?

PROGARM:

```
#creating a subset using head
df_head = df.head()
#Setting name as custom index
df_head.set_index('name', inplace = True)
df_head
```

OUTPUT:

name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	usa
buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	usa
plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	usa
amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	usa
ford torino	17.0	8	302.0	140.0	3449	10.5	70	usa

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

5) WRITE PROGRAMS USING ILOC?

iloc- Access a group of rows and columns by integer index.

PROGRAM 1:

```
df.iloc[2,1]
```

OUTPUT: **8**

PROGRAM 2:

```
[ ]  df.iloc[1:5, 4:6]
```

OUTPUT:

	displacement	horsepower	weight	acceleration
1	350.0	165.0	3693	11.5
2	318.0	150.0	3436	11.0
3	304.0	150.0	3433	12.0
4	302.0	140.0	3449	10.5

6) WRITE A PROGRAM USING LOC?

loc- Access a group of rows and columns by custom index.

PROGRAM:

```
[ ]  #Subsetting from the full dataset
     df.loc[0:5, ['cylinders', 'horsepower', 'name']]
```

OUTPUT:

	cylinders	horsepower	name
0	8	130.0	chevrolet chevelle malibu
1	8	165.0	buick skylark 320
2	8	150.0	plymouth satellite
3	8	150.0	amc rebel sst
4	8	140.0	ford torino
5	8	198.0	ford galaxie 500

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

OPERATIONS AND PANDAS:

7) WRITE A PROGRAM TO RETRIEVE DETAILS OF CARS BUILT IN YEAR 72?

PROGRAM:

```
df.loc[df['model_year'] == 72 ].head()
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
57	24.0	4	113.0	95.0	2278	15.5	72	japan	toyota corona hardtop
58	25.0	4	97.5	80.0	2126	17.0	72	usa	dodge colt hardtop
59	23.0	4	97.0	54.0	2254	23.5	72	europe	volkswagen type 3
60	20.0	4	140.0	90.0	2408	19.5	72	usa	chevrolet vega
61	21.0	4	122.0	86.0	2226	16.5	72	usa	ford pinto runabout

8) WRITE PROGRAM TO RETRIEVE DETAILS OF CARS BUILT IN JAPAN AND 6 CYLINDERS?

PROGRAM:

```
df.loc[(df['origin'] == 'japan') & (df['cylinders'] == 6)]
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
123	20.0	6	156.0	122.0	2807	13.5	73	japan	toyota mark ii
210	19.0	6	156.0	108.0	2930	15.5	76	japan	toyota mark ii
241	22.0	6	146.0	97.0	2815	14.5	77	japan	datsun 810
333	32.7	6	168.0	132.0	2910	11.4	80	japan	datsun 280-zx
361	25.4	6	168.0	116.0	2900	12.6	81	japan	toyota cressida
362	24.2	6	146.0	120.0	2930	13.8	81	japan	datsun 810 maxima

9)

Problem Statement:

XYZ Custom Cars want to categorize cars in different categories as follows:

Category	Description	Features coming in play
Fuel efficient	Cars designed with low power and high fuel efficiency	High MPG, Low Horsepower, Low weight
Muscle Cars	Intermediate sized cars designed for high performance	High displacement, High horsepower, Moderate weight
SUV	Big sized cars designed for high performance, long distance trips and family comfort	High horsepower, High weight
Racecar	Cars specifically designed for race tracks	Low weight, High acceleration

Solution:

Their experienced engineers and mechanics have come up with the following parameters for these categories-

Category	Description	Features involved
Fuel efficient	Cars designed with low power and high fuel efficiency	MPG > 29, Horsepower < 93.5, Weight < 2500
Muscle Cars	Intermediate sized cars designed for high performance	Displacement >262, Horsepower > 126, Weight in range[2800, 3600]
SUV	Big sized cars designed for high performance, long distance trips and family comfort	Horsepower > 140 , Weight > 4500
Racecar	Cars specifically designed for race tracks	Weight <2223, acceleration > 17

10) WRITE PROGRAMS TO FIND OUT CARS BELONGING TO THESE CATEGORIES BASED ON GIVEN PARAMETERS?

PROGRAM 1:

```
# Fuel efficient
# MPG > 29, Horsepower < 93.5,
# Weight < 2500
df.loc[(df['mpg'] > 29) & (df['horsepower'] < 93.5) & (df['weight'] < 2500)]
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
51	30.0	4	79.0	70.0	2074	19.5	71	europe	peugeot 304
52	30.0	4	88.0	76.0	2065	14.5	71	europe	fiat 124b
53	31.0	4	71.0	65.0	1773	19.0	71	japan	toyota corolla 1200
54	35.0	4	72.0	69.0	1613	18.0	71	japan	datsun 1200
129	31.0	4	79.0	67.0	1950	19.0	74	japan	datsun b210
...	...	...	...	...	...	...	...	...	...
384	32.0	4	91.0	67.0	1965	15.7	82	japan	honda civic (auto)
385	38.0	4	91.0	67.0	1995	16.2	82	japan	datsun 310 gx
391	36.0	4	135.0	84.0	2370	13.0	82	usa	dodge charger 2.2
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage

81 rows × 9 columns

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

PROGRAM 2:

```
[ ]  # Muscle cars
     # Displacement >262, Horsepower > 126, Weight in range[2800, 3600]
     df.loc[(df['displacement'] > 262) & (df['horsepower'] > 126) & (df['weight'] >=2800) & (df['weight'] <= 3600)]
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino
10	15.0	8	383.0	170.0	3563	10.0	70	usa	dodge challenger se
13	14.0	8	455.0	225.0	3086	10.0	70	usa	buick estate wagon (sw)
121	15.0	8	318.0	150.0	3399	11.0	73	usa	dodge dart custom
166	13.0	8	302.0	129.0	3169	12.0	75	usa	ford mustang ii
251	20.2	8	302.0	139.0	3570	12.8	78	usa	mercury monarch ghia
262	19.2	8	305.0	145.0	3425	13.2	78	usa	chevrolet monte carlo landau
264	18.1	8	302.0	139.0	3205	11.2	78	usa	ford futura

PROGRAM 3:

```
# SUV
# Horsepower > 140 , Weight > 4500
df.loc[(df['horsepower'] > 140) & (df['weight'] >=4500)]
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
25	10.0	8	360.0	215.0	4615	14.0	70	usa	ford f250
28	9.0	8	304.0	193.0	4732	18.5	70	usa	hi 1200d
42	12.0	8	383.0	180.0	4955	11.5	71	usa	dodge monaco (sw)
43	13.0	8	400.0	170.0	4746	12.0	71	usa	ford country squire (sw)
44	13.0	8	400.0	175.0	5140	12.0	71	usa	pontiac safari (sw)
67	11.0	8	429.0	208.0	4633	11.0	72	usa	mercury marquis
68	13.0	8	350.0	155.0	4502	13.5	72	usa	buick lesabre custom
90	12.0	8	429.0	198.0	4952	11.5	73	usa	mercury marquis brougham
94	13.0	8	440.0	215.0	4735	11.0	73	usa	chrysler new yorker brougham
95	12.0	8	455.0	225.0	4951	11.0	73	usa	buick electra 225 custom
103	11.0	8	400.0	150.0	4997	14.0	73	usa	chevrolet impala
104	12.0	8	400.0	167.0	4906	12.5	73	usa	ford country

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

11) WRITE A PROGRAM TO STORE DATA ACCORDING TO NUMBER OF CYLINDERS?
PROGRAM:

```
df.sort_values(by = 'cylinders')
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name
243	21.5	3	80.0	110.0	2720	13.5	77	japan	mazda rx-4
334	23.7	3	70.0	100.0	2420	12.5	80	japan	mazda rx-7 gs
111	18.0	3	70.0	90.0	2124	13.5	73	japan	maxda rx3
71	19.0	3	70.0	97.0	2330	13.5	72	japan	mazda rx2 coupe
237	30.5	4	98.0	63.0	2051	17.0	77	usa	chevrolet chevette
...	...	...	...	...	...	...	...	...	...
86	14.0	8	304.0	150.0	3672	11.5	73	usa	amc matador
85	13.0	8	350.0	175.0	4100	13.0	73	usa	buick century 350
285	17.0	8	305.0	130.0	3840	15.4	79	usa	chevrolet caprice classic
92	13.0	8	351.0	158.0	4363	13.0	73	usa	ford ltd
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu

392 rows × 9 columns

12) WRITE A PROGRAM TO FIND MINIMUM AND MAXIMUM OF ALL NUMERICAL COLUMNS?
PROGRAM:

```
#Using list comprehension to get the numerical columns
list1 = [col for col in df.columns if df[col].dtype in ['float', 'int64']]
df[list1].agg(['min', 'max'])
```

OUTPUT:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
min	9.0	3	68.0	46.0	1613	8.0	70
max	46.6	8	455.0	230.0	5140	24.8	82

13) WRITE A PROGRAM TO FIND HOW MANY CARS BELONG TO EACH YEAR?
PROGRAM:

```
df.groupby(['model_year']).count()[['name']]
```

OUTPUT:

model_year	name
70	29
71	27
72	28
73	40
74	26
75	30
76	34

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

14) WRITE A PROGRAM TO UNDERSTAND ABOUT THE EFFECT OF MODEL YEAR AND NUMBER OF CYLINDERS ON HORSEPOWER?

PROGRAM:

```
#Creating a DataFrame grouped on cylinders and model_year and finding mean, min and max of horsepower
grouped_multiple = df.groupby(['cylinders', 'model_year']).agg({'horsepower': ['mean', 'min', 'max']})
#Naming columns in grouped DataFrame
grouped_multiple.columns = ['hp_mean', 'hp_min', 'hp_max']
#Resetting index
grouped_multiple = grouped_multiple.reset_index()
#Viewing head of resulting DataFrame
grouped_multiple.head()
```

OUTPUT:

	cylinders	model_year	hp_mean	hp_min	hp_max
0	3	72	97.000000	97.0	97.0
1	3	73	90.000000	90.0	90.0
2	3	77	110.000000	110.0	110.0
3	3	80	100.000000	100.0	100.0
4	4	70	87.714286	46.0	113.0

NOTE:

A Pivot Table is used to summarise, sort, reorganise, group, count, total or average data stored in a table. If we want to create spreadsheet-style pivot table as a data frame, pandas provides us with an option.

15) WRITE A PROGRAM TO FIND THE MEAN OF ALL THE NUMERICAL ATTRIBUTES OF CARS FOR EACH YEAR?

PROGRAM:

```
pivot1 = pd.pivot_table(df, index = 'model_year', aggfunc=np.mean)
pivot1
```

OUTPUT:

model_year	acceleration	cylinders	displacement	horsepower	mpg	weight
70	12.948276	6.758621	281.413793	147.827586	17.689655	3372.793103
71	15.000000	5.629630	213.888889	107.037037	21.111111	3030.592593
72	15.125000	5.821429	218.375000	120.178571	18.714286	3237.714286
73	14.312500	6.375000	256.875000	130.475000	17.100000	3419.025000
74	16.173077	5.230769	170.653846	94.230769	22.769231	2878.038462
75	16.050000	5.600000	205.533333	101.066667	20.266667	3176.800000
76	15.941176	5.647059	197.794118	101.117647	21.573529	3078.735294
77	15.435714	5.464286	191.392857	105.071429	23.375000	2997.357143
78	15.805556	5.361111	177.805556	99.694444	24.061111	2861.805556
79	15.813793	5.827586	206.689655	101.206897	25.093103	3055.344828
80	17.018519	4.148148	116.074074	77.481481	33.803704	2441.592593
81	16.325000	4.642857	136.571429	81.035714	30.185714	2530.178571
82	16.510000	4.200000	128.133333	81.466667	32.000000	2434.166667

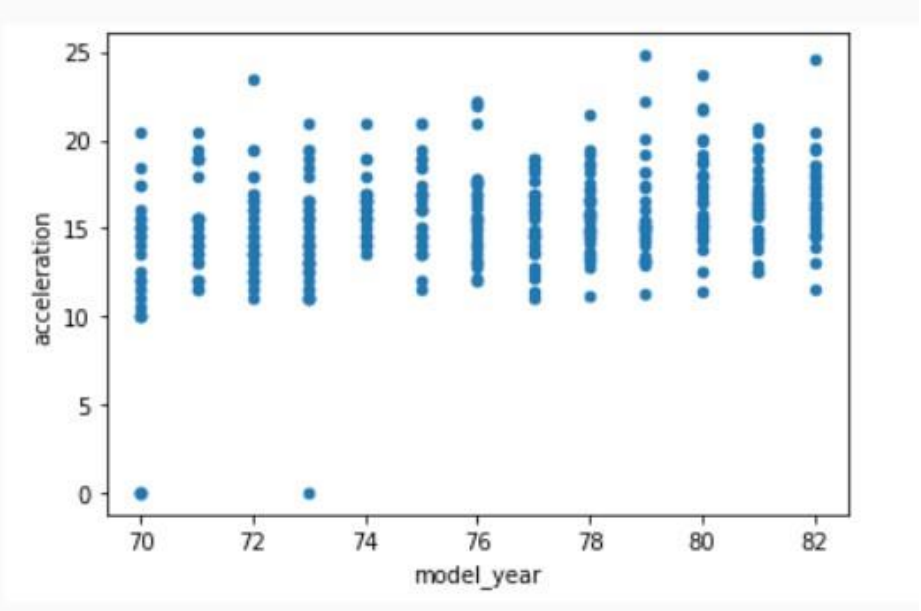
PANDAS PLOTS:

16) WRITE A PROGRAM TO DISPLAY SCATTER PLOT?

PROGRAM:

```
[ ]  df.plot(x = 'model_year', y = 'acceleration', marker = 'o', kind = 'scatter');
```

OUTPUT:

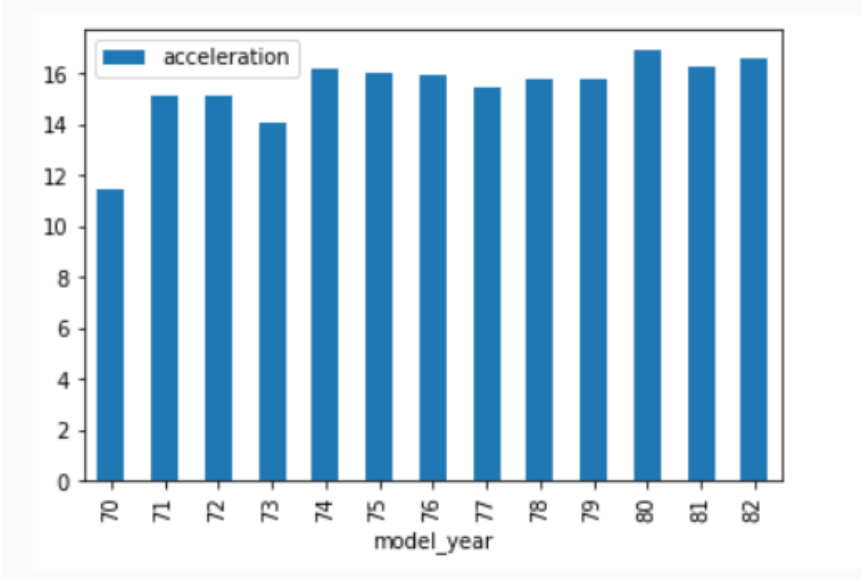


17) WRITE A PROGRAM TO DISPLAY BOX PLOT?

PROGRAM:

```
[ ]  df.groupby('model_year').mean()[['acceleration']].plot(kind = 'bar');
```

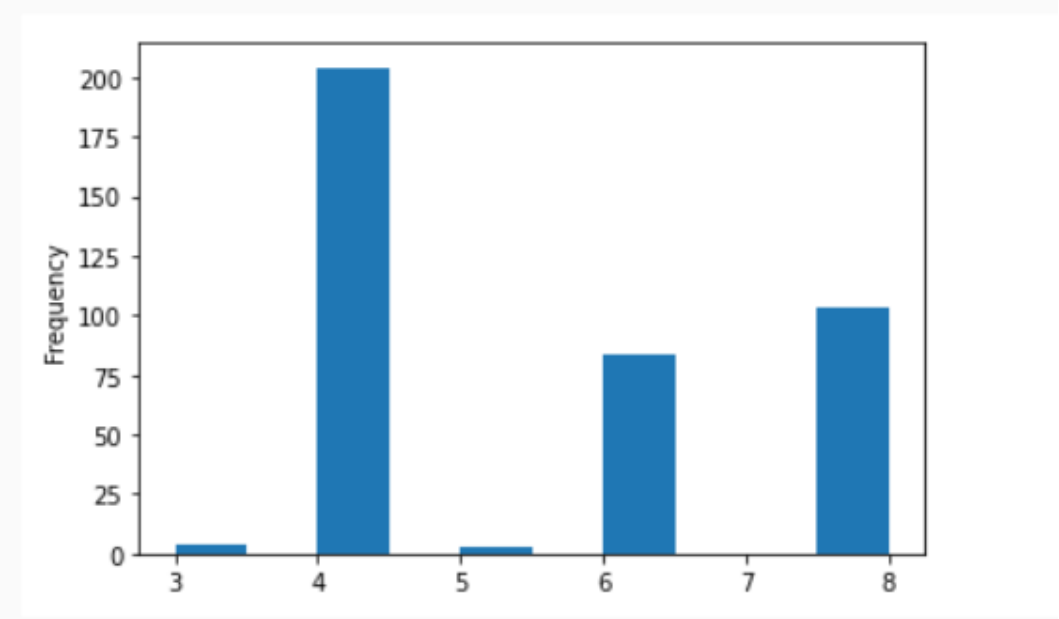
OUTPUT:



18) WRITE A PROGRAM TO DISPLAY HISTOGRAM?
PROGRAM:

```
[ ]  df['cylinders'].plot(kind = 'hist')
```

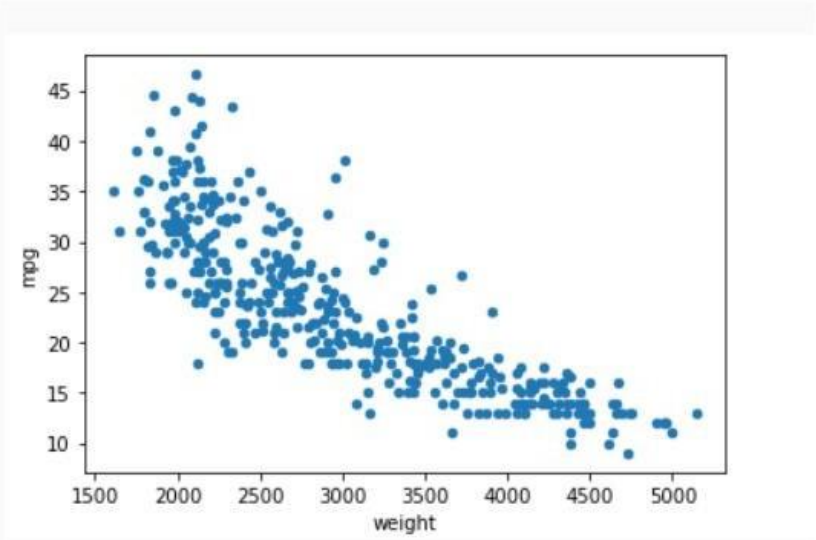
OUTPUT:



19) DISPLAY SCATTER PLOT RELATIONSHIP BETWEEN MPG AND WEIGHT?
PROGRAM:

```
[ ]  df.plot(x = 'weight', y = 'mpg', kind = 'scatter')
```

OUTPUT:



VNR VJIET

Name of the Laboratory:_____ **Name of the Experiment: _____**

Experiment No:_____Date: _____

WEEK 5

1. **Importing Data and Dataset**

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

```
In [3]: titanic_train=pd.read_csv("titanic.csv")
```

```
In [4]: titanic_train.shape
```

2. **Obtain number of rows and columns.**

```
In [4]: titanic_train.shape
```

Output:

```
Out[4]: (418, 12)
```

3. **Make column names same in test and train dataset so that missing values can be filled simultaneously.**

```
In [5]: titanic_train.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
 2   Pclass       418 non-null    int64
 3   Name         418 non-null    object
 4   Sex          418 non-null    object
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64
 7   Parch        418 non-null    int64
 8   Ticket       418 non-null    object
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object
 11  Embarked     418 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

4. Summarizing statistics of the 'titanic_train' dataset

```
In [6]: #numerical summary of continuous columns
        titanic_train.describe()
```

Output:

Out[6]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329200

---

5. Counting the no. of occurrences of 'Survived' Column using crosstab.

```
In [8]: pd.crosstab(index=titanic_train["Survived"], columns="count")
```

Output:

Out[8]:

col_0	count
Survived	
0	266
1	152

6. Counting the no. of occurrences of 'Pclass' Column using crosstab.

```
In [9]: pd.crosstab(index=titanic_train["Pclass"], columns="count")
```

Output:

Out[9]:

Pclass	col_0 count
1	107
2	93
3	218

7. Counting the no. of occurrences of 'Sex' Column using crosstab

```
In [10]: pd.crosstab(index=titanic_train["Sex"],  columns="count")
```

Output:

Out[10]:

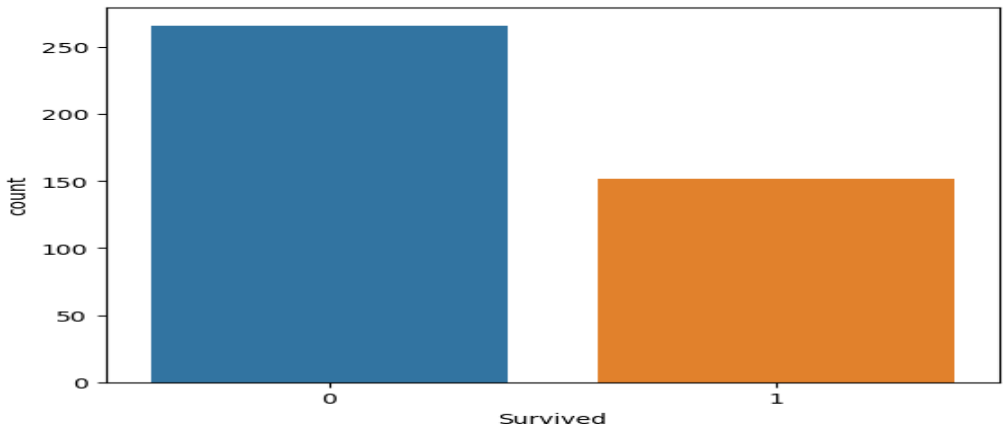
Sex	col_0 count
female	152
male	266

8. Plotting a count plot based on the 'Survived' column from the 'titanic_train' dataset.

```
In [12]: sns.countplot(x='Survived',data=titanic_train)
```

Output:

```
Out[12]: <Axes: xlabel='Survived', ylabel='count'>
```

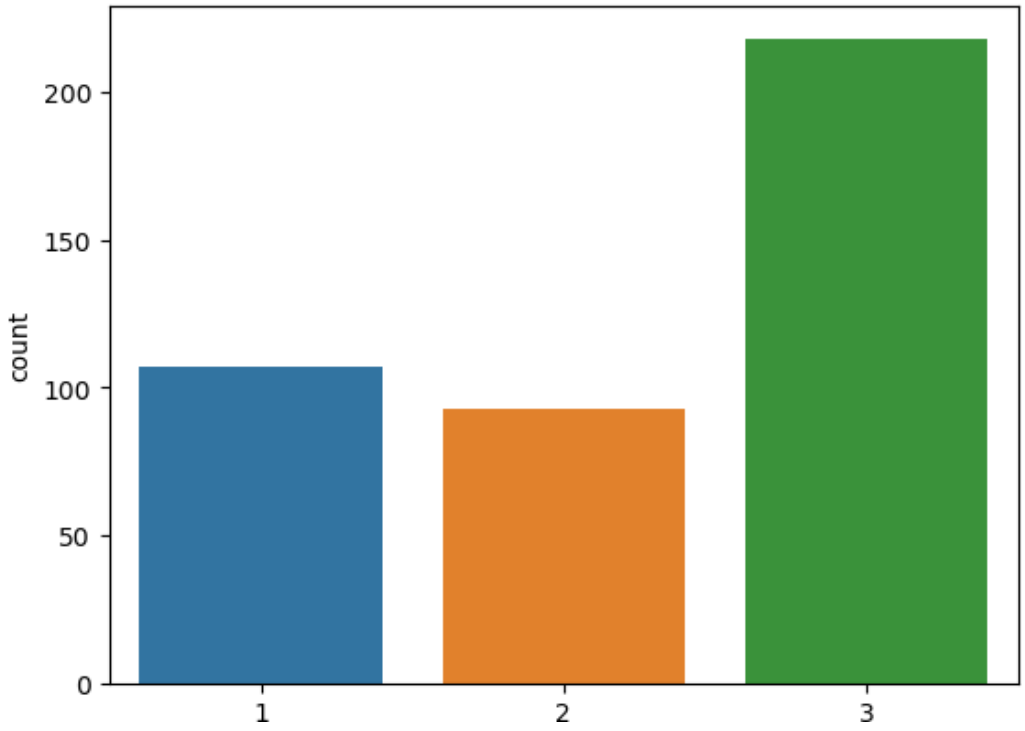


9. Plotting a countplot based on the 'Pclass' column from the 'titanic_train' dataset.

```
In [14]: sns.countplot(x='Pclass',data=titanic_train)
```

Output:

Out[14]: <Axes: xlabel='Pclass', ylabel='count'>



10. Summarizing statistics of the 'Fare' column in the 'titanic_train' dataset using describe.

```
In [16]: titanic_train['Fare'].describe()
```

Output:

```
Out[16]: count    417.000000
         mean      35.627188
         std       55.907576
         min        0.000000
         25%        7.895800
         50%       14.454200
         75%       31.500000
         max      512.329200
         Name: Fare, dtype: float64
```

11. Creating a boxplot based on the 'Fare' column from the 'titanic_train' dataset.

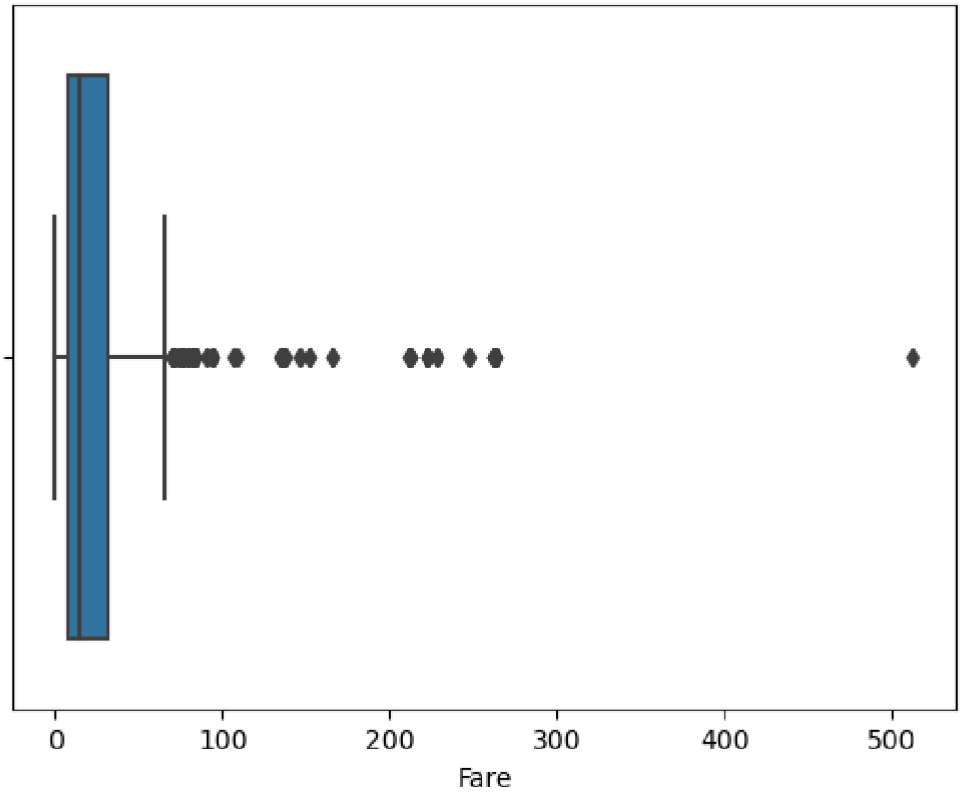
```
In [18]: sns.boxplot(x='Fare',data=titanic_train)
```

Name of the Laboratory: _____ **Name of the Experiment:** _____

Experiment No:_____**Date:** _____

Output:

```
Out[18]: <Axes: xlabel='Fare'>
```



12. Importing Data and creating sample data for plotting and creating empty canvas using Matplotlib, sets up axes, and plots two lines on the same set of axes.

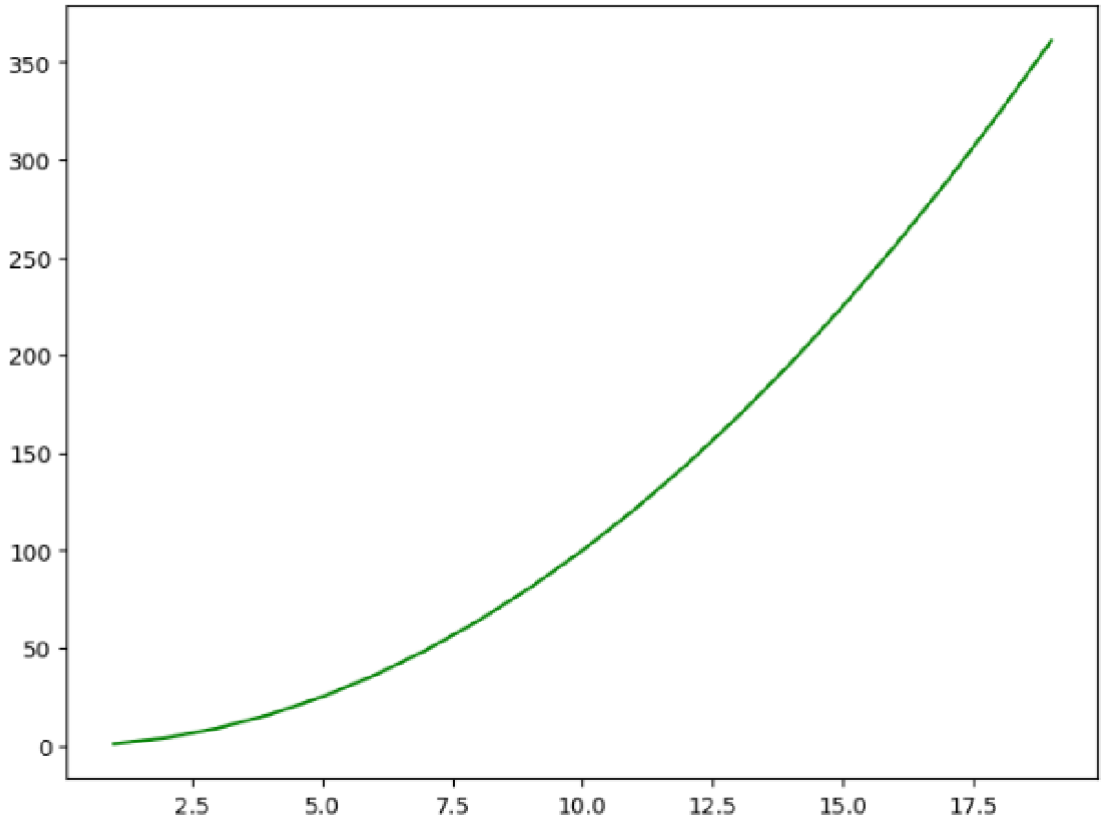
```
In [24]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
```

```
In [15]: #creating sample data for plotting
         x = np.arange(1,20)    #1 … 15
         y = x**2    #square of x values
         z = (y/2)+40    #arithmetic operation applied to y values
```

```
In [16]: #creating an empty canvas/figure
         fig = plt.figure()
         #setting axes
         ax = fig.add_axes([0,0,1,1])
         #plotting the lines
         ax.plot(x,y,color="g",label='line1' )
```

Output:

```
Out[16]: [<matplotlib.lines.Line2D at 0x241116a5750>]
```

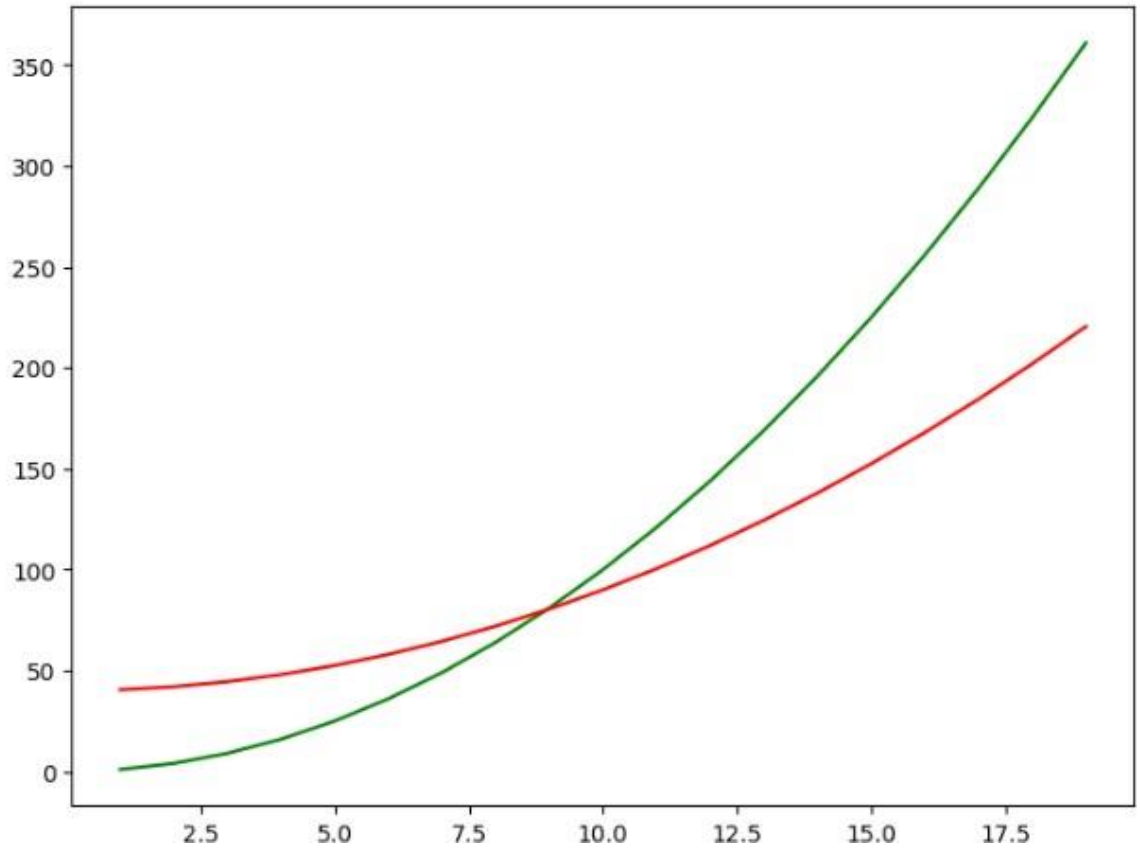


13. Creating Empty Canvas using Matplotlib, sets up axes, and plots two lines on the same set of axes with 2 lines.

```
In [17]: #creating an empty canvas/figure
         fig = plt.figure()
         #setting axes
         ax = fig.add_axes([0,0,1,1])
         #plotting the lines
         ax.plot(x,y,color="g",label='line1' )
         ax.plot(x,z,color="r",label='line2')
```

Output:

```
Out[17]: [<matplotlib.lines.Line2D at 0x241118ae310>]
```

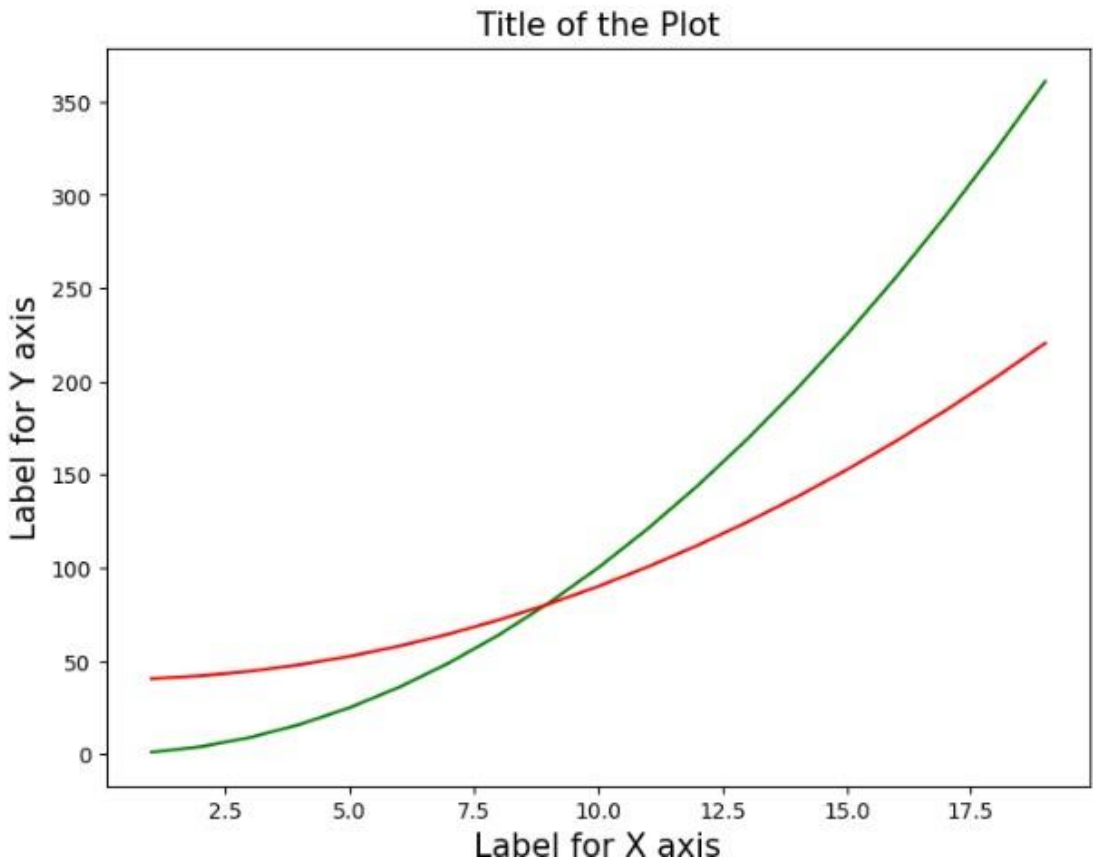


14. Adding a title to the plot and labeling the x-axis and y-axis

```
In [18]: #creating an empty canvas/figure
         fig = plt.figure()
         #setting axes
         ax = fig.add_axes([0,0,1,1])
         #plotting the lines
         ax.plot(x,y,color="g", label='line1' )
         ax.plot(x,z,color="r", label='line2')
         ax.set_title('Title of the Plot',fontsize=15)    #setting title
         ax.set_xlabel('Label for X axis',fontsize=15)    # setting X label
         ax.set_ylabel('Label for Y axis',fontsize=15)    # setting Y _label
```

Output:

Out[18]: Text(0, 0.5, 'Label for Y axis')

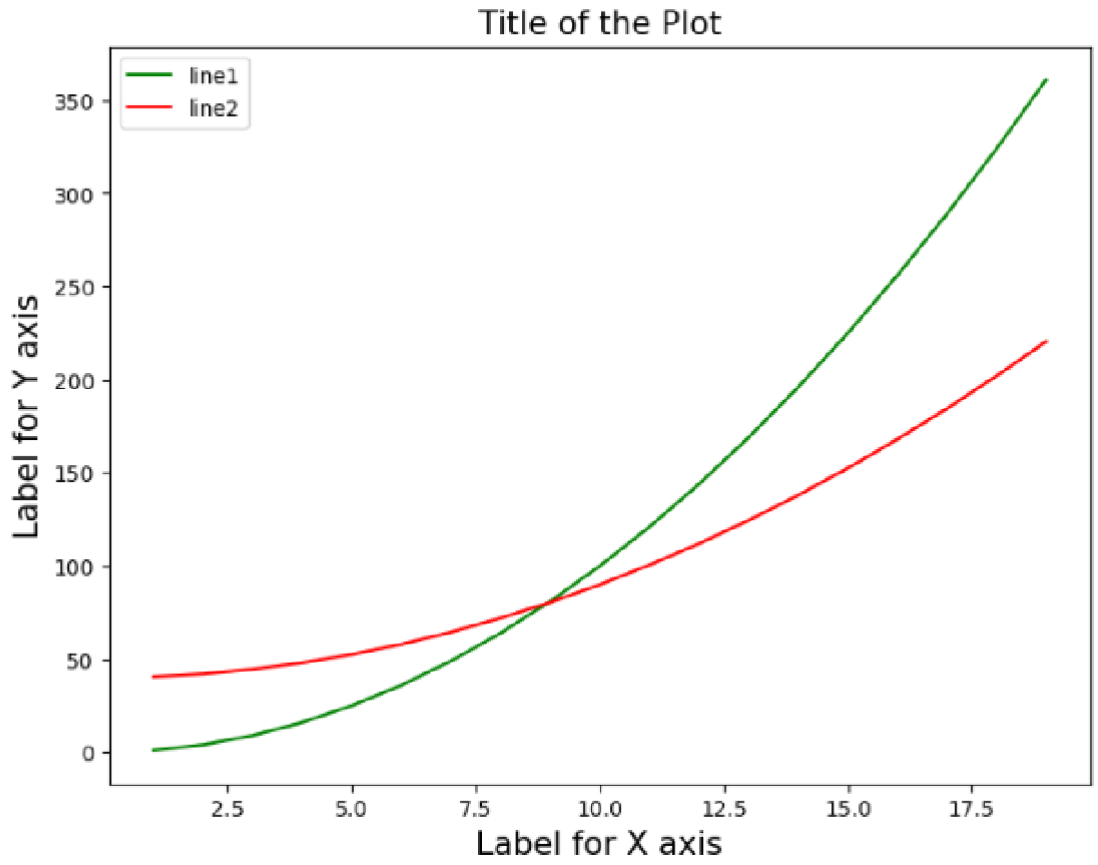


15. Adding a legend to the plot.

In [19]:
```
#creating an empty canvas/figure
fig = plt.figure()
#setting axes
ax = fig.add_axes([0,0,1,1])
#plotting the lines
ax.plot(x,y,color="g", label='line1' )
ax.plot(x,z,color="r", label='line2')
ax.set_title('Title of the Plot',fontsize=15)    #setting title
ax.set_xlabel('Label for X axis',fontsize=15)    # setting X label
ax.set_ylabel('Label for Y axis',fontsize=15)    # setting Y _label
ax.legend()    #adding legend
```

Output:

```
Out[19]: <matplotlib.legend.Legend at 0x2410f9e0750>
```

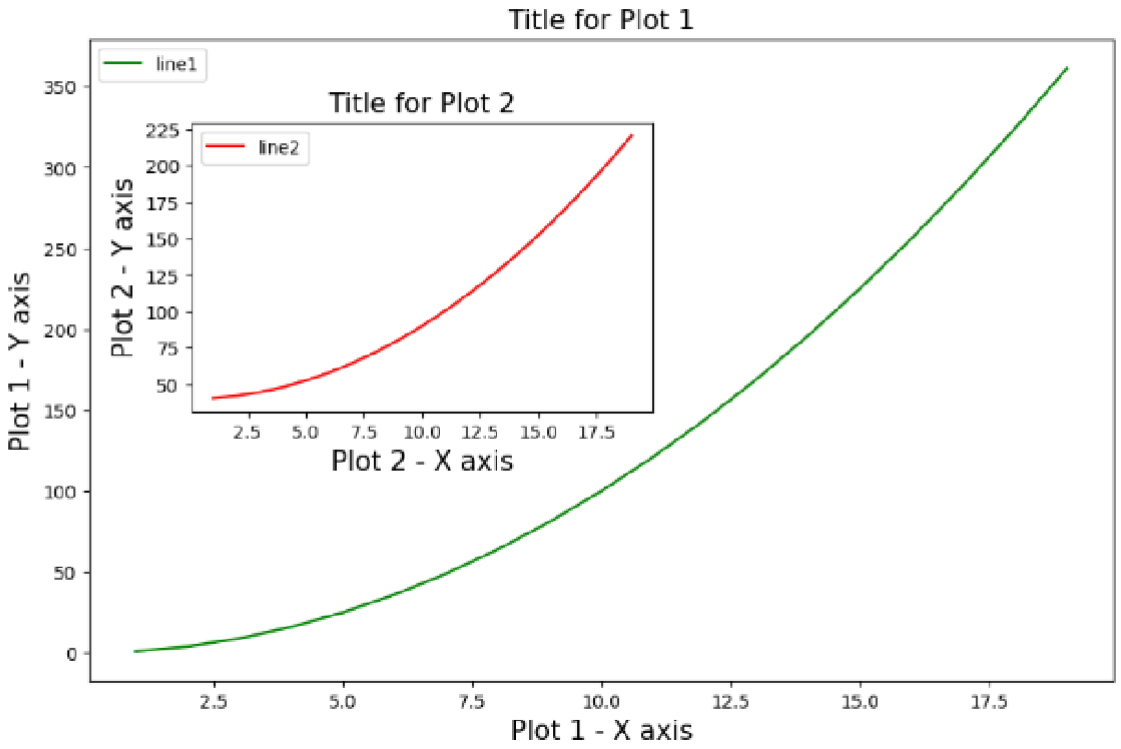


16. Plotting line1 and line2 in different axes.

```
In [20]: #creating sample data for plotting
x = np.arange(1,20)
y = x**2    #square of x values
z = (y/2)+40    #arithmetic operation applied to y values
#creating an empty canvas/figure
fig = plt.figure(figsize=(8,5))
#setting axes as [left, bottom, width, height]
ax1 = fig.add_axes([0,0,1,1])
ax2 = fig.add_axes([0.10,0.42,0.45,0.45])
#plotting the lines
ax1.plot(x,y,color="g", label='line1' )
ax1.set_title('Title for Plot 1',fontsize=15)    #setting title
ax1.set_xlabel('Plot 1 - X axis',fontsize=15)    # setting X label
ax1.set_ylabel('Plot 1 - Y axis',fontsize=15)    # setting Y _label
ax2.plot(x,z,color="r", label='line2')
ax2.set_title('Title for Plot 2',fontsize=15)    #setting title
ax2.set_xlabel('Plot 2 - X axis',fontsize=15)    # setting X label
ax2.set_ylabel('Plot 2 - Y axis',fontsize=15)    # setting Y _label
ax1.legend()    #adding legend
ax2.legend()
```

Output:

```
Out[20]: <matplotlib.legend.Legend at 0x241111f6690>
```

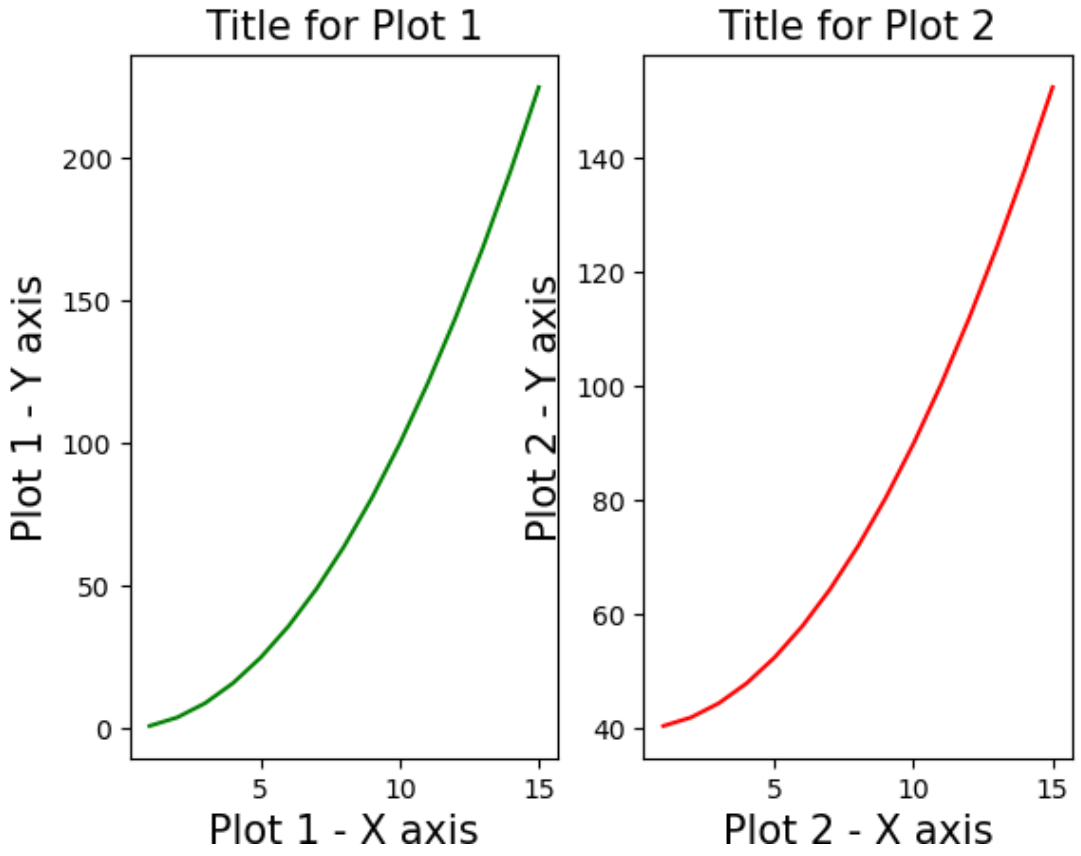


17. Subplotting that has 1 row and 2 columns avoiding, 'fig.tight_layout()'.

```
In [21]: #creating sample data for plotting
x = np.arange(1,16)    #1 … 15
y = x**2    #square of x values
z = (y/2)+40    #arithmetic operation applied to y values
#initializing two axes for the subplot
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(x,y,color="g", label='line1' )
ax1.set_title('Title for Plot 1',fontsize=15)    #setting title
ax1.set_xlabel('Plot 1 - X axis',fontsize=15)    # setting X label
ax1.set_ylabel('Plot 1 - Y axis',fontsize=15)    # setting Y _label
ax2.plot(x,z,color="r", label='line2')
ax2.set_title('Title for Plot 2',fontsize=15)    #setting title
ax2.set_xlabel('Plot 2 - X axis',fontsize=15)    # setting X label
ax2.set_ylabel('Plot 2 - Y axis',fontsize=15)    # setting Y _label
```

Output:

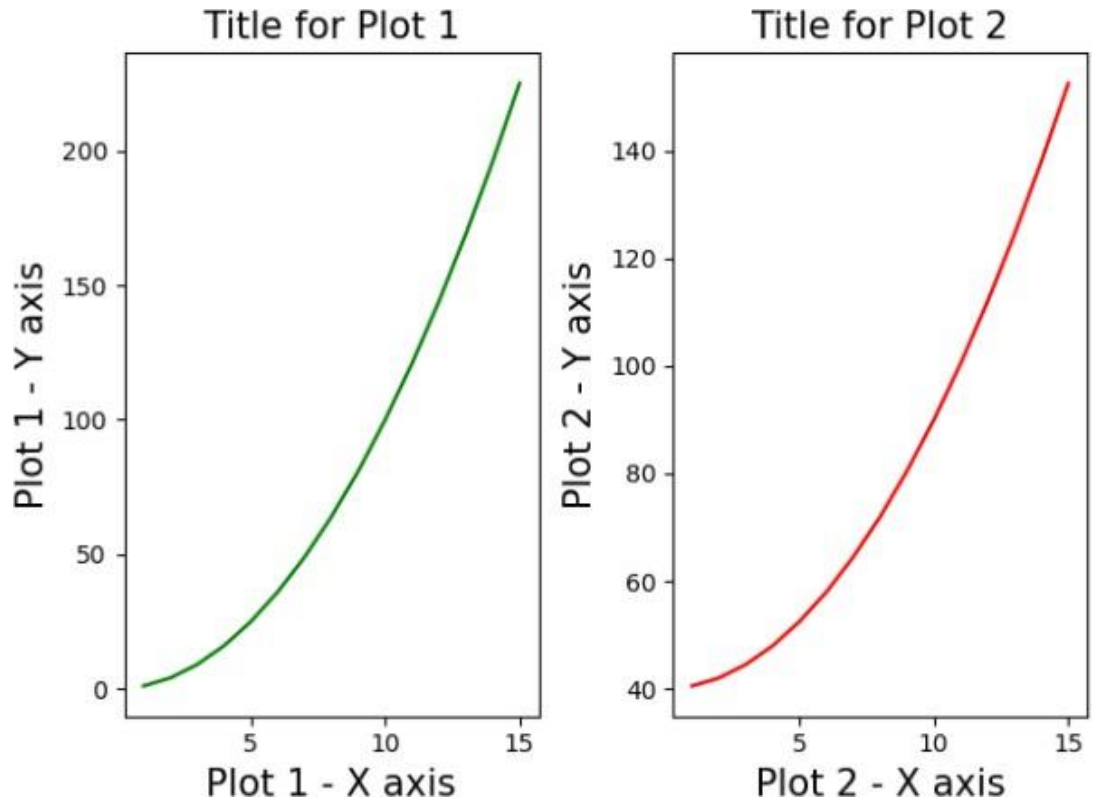
```
Out[21]: Text(0, 0.5, 'Plot 2 - Y axis')
```



18. Subplotting that has 1 row and 2 columns with 'fig.tight_layout()'

```
In [22]: #creating sample data for plotting
x = np.arange(1,16)    #1 … 15
y = x**2    #square of x values
z = (y/2)+40    #arithmetic operation applied to y values
#initializing two axes for the subplot
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.plot(x,y,color="g", label='line1' )
ax1.set_title('Title for Plot 1',fontsize=15)    #setting title
ax1.set_xlabel('Plot 1 - X axis',fontsize=15)    # setting X label
ax1.set_ylabel('Plot 1 - Y axis',fontsize=15)    # setting Y _label
ax2.plot(x,z,color="r", label='line2')
ax2.set_title('Title for Plot 2',fontsize=15)    #setting title
ax2.set_xlabel('Plot 2 - X axis',fontsize=15)    # setting X label
ax2.set_ylabel('Plot 2 - Y axis',fontsize=15)    # setting Y _label
fig.tight_layout()
```

Output:



19.T-o know the column names, missing values and their data type.

```
In [32]: #importing the data
         df = pd.read_csv('auto-mpg.csv')
         df.info()    #to know the column names, missing values and their data type
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64  
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object 
 4   weight        398 non-null    int64  
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64  
 7   origin        398 non-null    int64  
 8   car name      398 non-null    object 
dtypes: float64(3), int64(4), object(2)
memory usage: 28.1+ KB
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

WEEK-6

```
[ ]  %matplotlib inline
     import pandas as pd
     import matplotlib.pyplot as plt
```

```
movies = pd.read_csv("movies.csv")
movies.head()
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

```
movies
```

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 6 columns

1) Check the number of rows and columns.

Program-

```
[ ]  # Answer:
     movies.shape
```

Output-

```
(979, 6)
```

2) **Check the data type of each column.**

Program-

```
[ ]  # Answer:
     movies.info()
```

Output-

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 979 entries, 0 to 978
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   star_rating     979 non-null    float64
 1   title           979 non-null    object
 2   content_rating  976 non-null    object
 3   genre           979 non-null    object
 4   duration        979 non-null    int64
 5   actors_list     979 non-null    object
dtypes: float64(1), int64(1), object(4)
memory usage: 46.0+ KB
```

3) **Calculate the average movie duration.**

Program-

```
[ ]  # Answer:
     movies.duration.mean()
```

Output-

```
120.97957099080695
```

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

4) Sort the DataFrame by duration to find the shortest and longest movies

Program-

```
[ ]  # Answer:
     movies.sort_values(by = "duration").head(1)
```

```
[ ]  movies.sort_values(by = "duration").tail(1)
```

Output-

	star_rating	title	content_rating	genre	duration	actors_list
389	8.0	Freaks	UNRATED	Drama	64	[u'Wallace Ford', u'Leila Hyams', u'Olga Bacla...

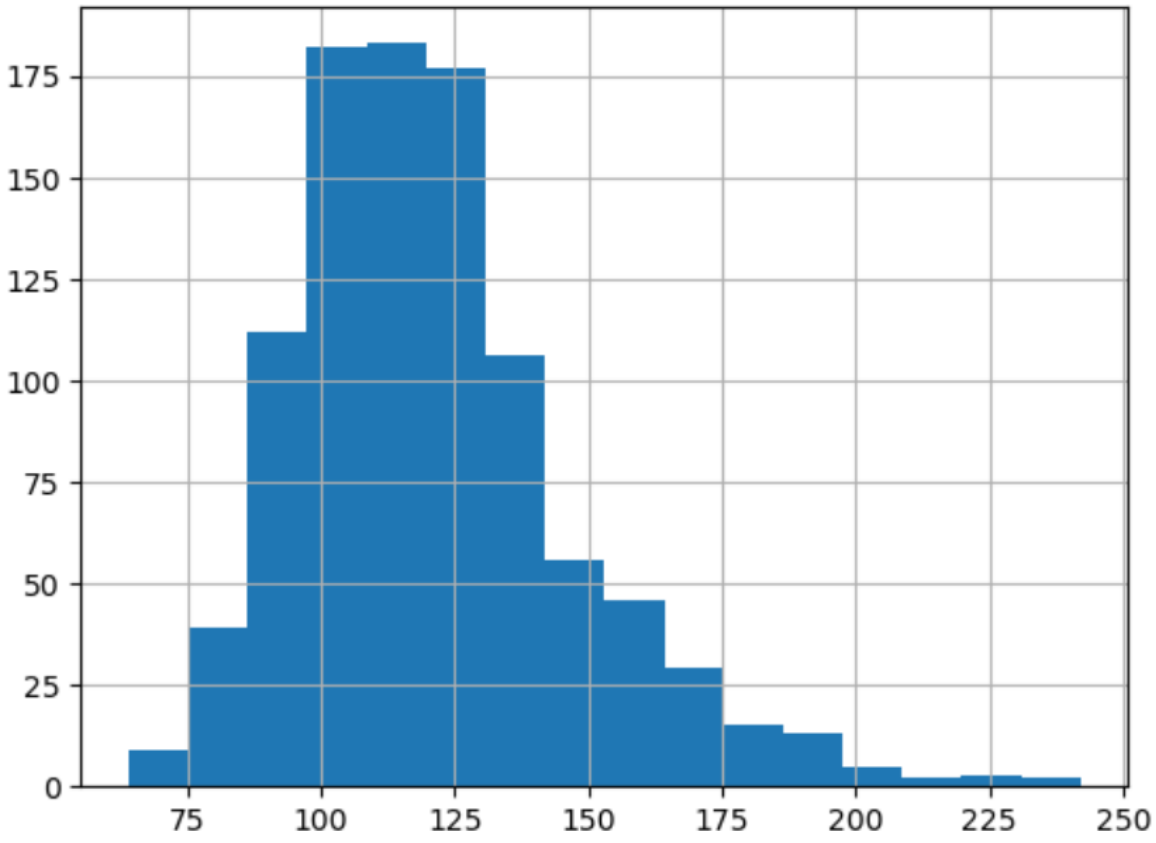
	star_rating	title	content_rating	genre	duration	actors_list
476	7.8	Hamlet	PG-13	Drama	242	[u'Kenneth Branagh', u'Julie Christie', u'Dere...

5) Create a histogram of duration, choosing an "appropriate" number of bins.

Program-

```
# Answer:
hist = movies['duration'].hist(bins=16)
```

Output-

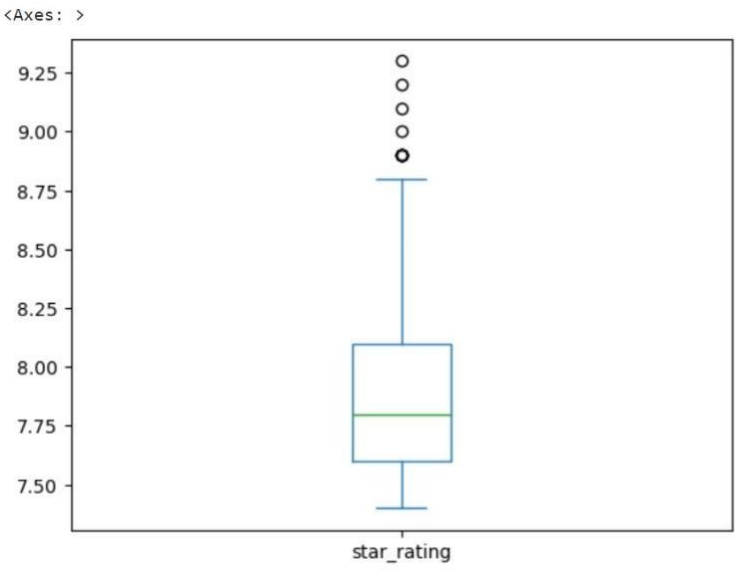


6) **Use a box plot to display that same data.**

Program-

```
# Answer:
movies.star_rating.plot(kind='box')
```

Output-



7) Count how many movies have each of the content ratings

Program-

```
[ ]  # Answer:
     movies.content_rating.value_counts()
```

Output-

```
R             460
PG-13         189
PG            123
NOT RATED      65
APPROVED       47
UNRATED        38
G              32
PASSED          7
NC-17           7
X               4
GP              3
TV-MA           1
Name: content_rating, dtype: int64
```

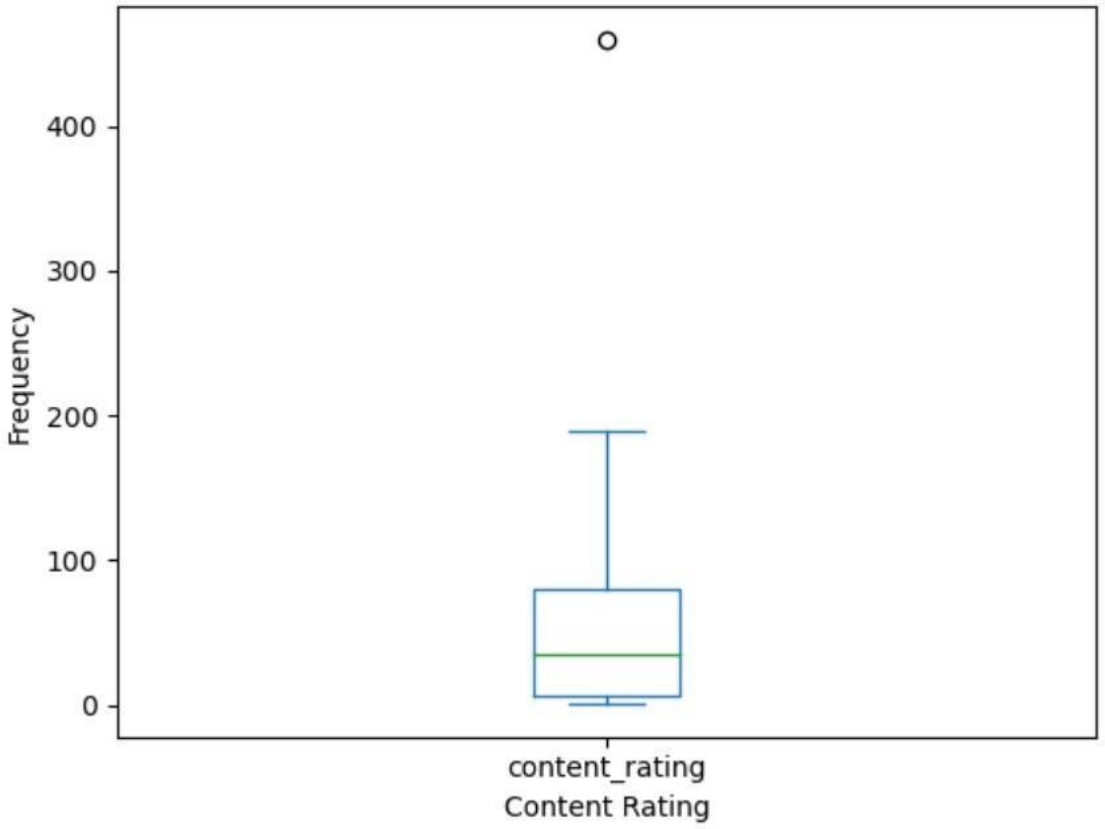
8) Use a visualization to display that same data, including a title and x and y labels.

Program-

```
# Answer:
movies.content_rating.value_counts().plot(kind='box')
plt.xlabel('Content Rating')
plt.ylabel('Frequency')
```

Output-

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



9) Convert the following content ratings to "UNRATED": NOT RATED, APPROVED, PASSED, GP.

Program-

```
# Answer:
movies.content_rating.replace(['NOT RATED', 'APPROVED', 'PASSED', 'GP'], 'UNRATED', inplace=True)
movies
```

Output-

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 6 columns

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

10) Convert the following content ratings to "NC-17": X, TV-MA.

Program-

```
# Answer:
movies.content_rating.replace(['X', 'TV-MA'], 'NC-17', inplace=True)
movies
```

Output-

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 6 columns

11) Count the number of missing values in each column.

Program-

```
# Answer:
movies.isnull().sum()
```

Output-

```
star_rating       0
title             0
content_rating    3
genre             0
duration          0
actors_list       0
dtype: int64
```

12) If there are missing values: examine them, then fill them in with "reasonable" values.

Program-

```
[ ]  # Answer:
     movies[movies.content_rating.isnull()]
     movies.content_rating.fillna('UNRATED', inplace=True)
     movies
```

Output-

[ ]

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
974	7.4	Tootsie	PG	Comedy	116	[u'Dustin Hoffman', u'Jessica Lange', u'Teri G...
975	7.4	Back to the Future Part III	PG	Adventure	118	[u'Michael J. Fox', u'Christopher Lloyd', u'Ma...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
977	7.4	Poltergeist	PG	Horror	114	[u'JoBeth Williams', u"Heather O'Rourke", u'Cr...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

979 rows × 6 columns

13) Calculate the average star rating for movies 2 hours or longer, and compare that with the average star rating for movies shorter than 2 hours.

Program-

```
# Answer:
print(" the average star rating for movies 2 hours or longer")
movies[movies.duration >= 120].star_rating.mean()
```

```
[ ]  #Answer
     print("the average star rating for movies shorter than 2 hours")
     movies[movies.duration < 120].star_rating.mean()
```

Output-

```
 the average star rating for movies 2 hours or longer
7.948898678414097

the average star rating for movies shorter than 2 hours
7.838666666666665
```

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

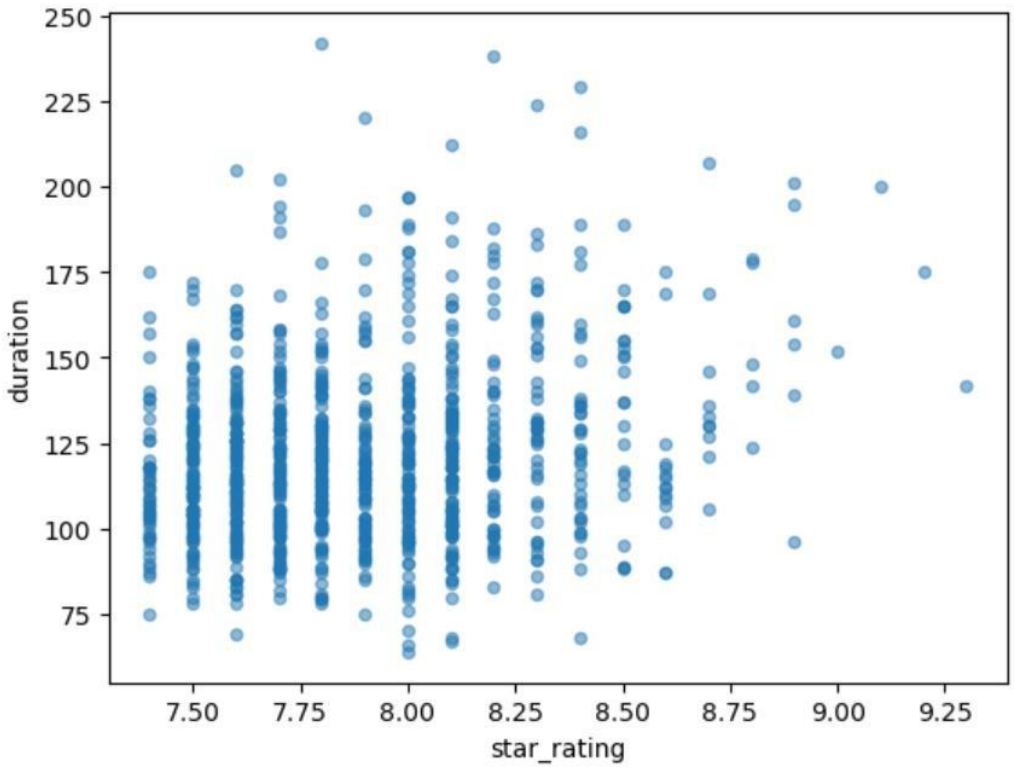
14) Use a visualization to detect whether there is a relationship between duration and star rating.

Program-

```
# Answer:
movies.plot(kind='scatter', x='star_rating', y='duration', alpha=0.5)
```

Output-

```
<Axes: xlabel='star_rating', ylabel='duration'>
```



15) Calculate the average duration for each genre.

Program-

```
# Answer:
movies.groupby('genre').duration.mean()
```

Output-

```
genre
Action        126.485294
Adventure     134.840000
Animation      96.596774
Biography     131.844156
Comedy        107.602564
Crime         122.298387
Drama         126.539568
Family        107.500000
Fantasy       112.000000
Film-Noir      97.333333
History        66.000000
Horror        102.517241
Mystery       115.625000
Sci-Fi        109.000000
Thriller      114.200000
Western       136.666667
Name: duration, dtype: float64
```

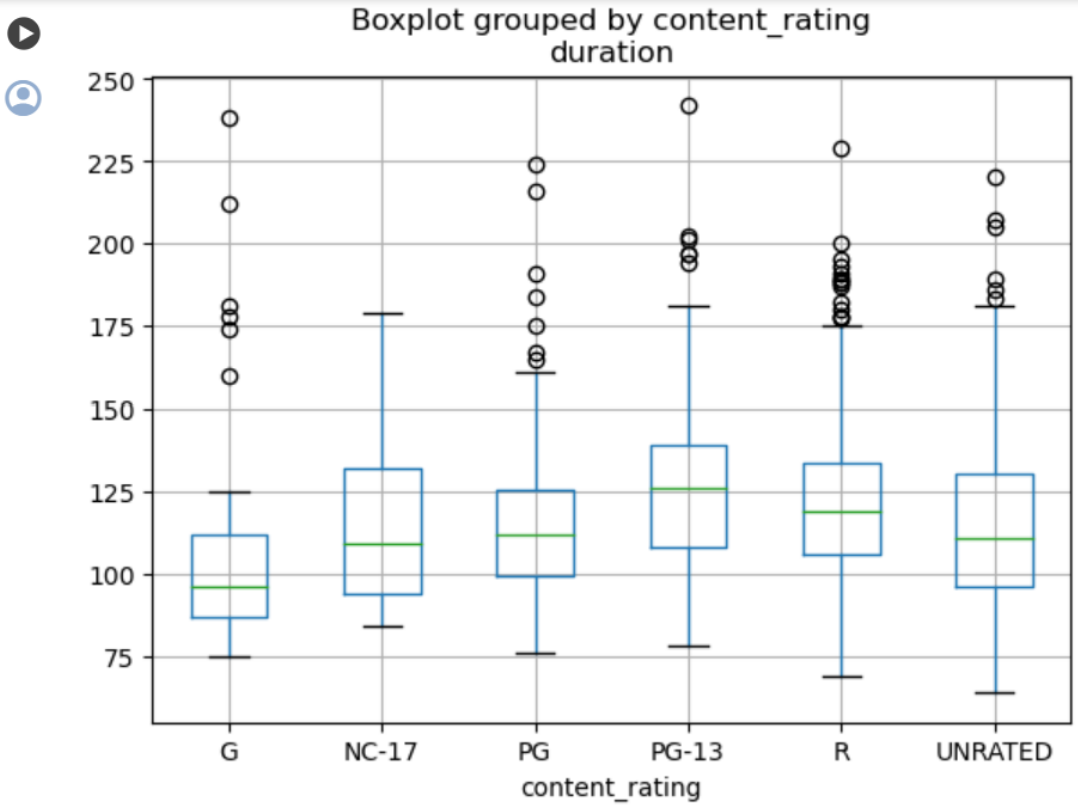
16) Visualize the relationship between content rating and duration.

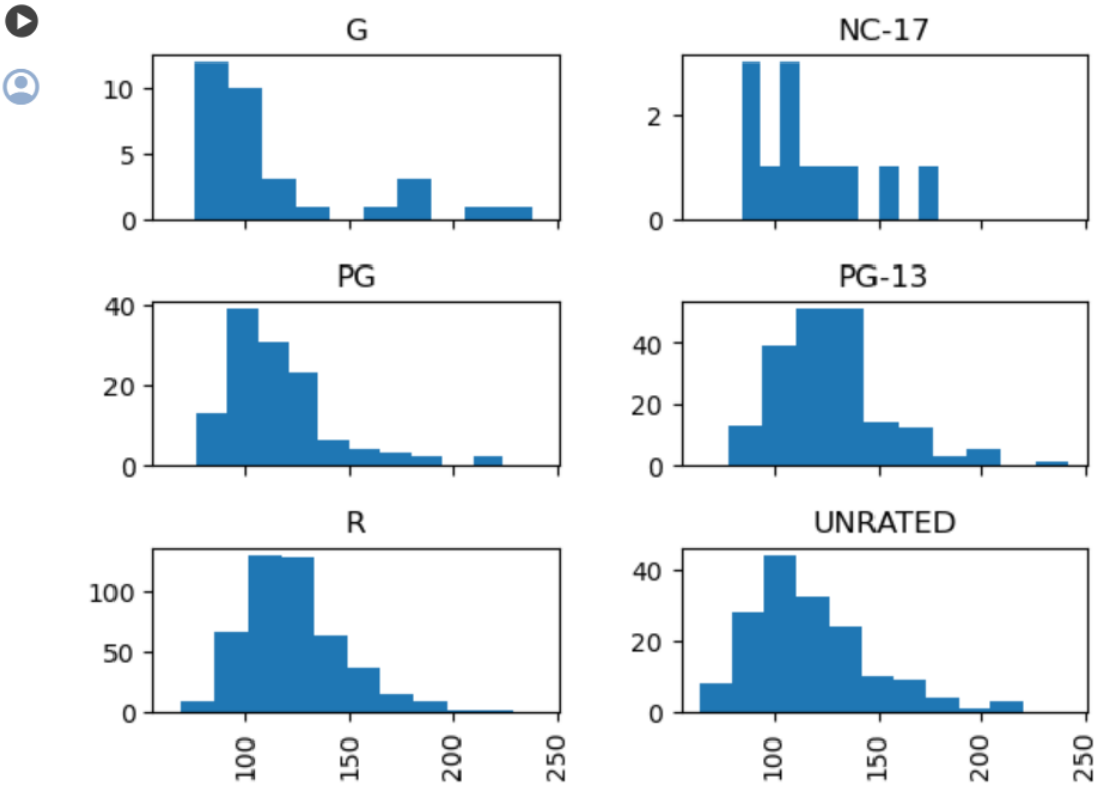
Program-

```
[ ]  # Answer:
     movies.boxplot(column='duration', by='content_rating')
     movies.duration.hist(by=movies.content_rating, sharex=True)
```

Output-

```
array([[<Axes: title={'center': 'G'}>, <Axes: title={'center': 'NC-17'}>],
       [<Axes: title={'center': 'PG'}>,
        <Axes: title={'center': 'PG-13'}>],
       [<Axes: title={'center': 'R'}>,
        <Axes: title={'center': 'UNRATED'}>]], dtype=object)
```





17) Determine the top rated movie (by star rating) for each genre.

Program-

```
# Answer:
#movies.sort_values('star_rating', ascending=False).groupby('genre').title.first()
movies.groupby('genre').title.first()
```

Output-

```
genre
Action                                      The Dark Knight
Adventure    The Lord of the Rings: The Return of the King
Animation                                     Spirited Away
Biography                                   Schindler's List
Comedy                                    Life Is Beautiful
Crime                               The Shawshank Redemption
Drama                                           12 Angry Men
Family                            E.T. the Extra-Terrestrial
Fantasy                            The City of Lost Children
Film-Noir                                     The Third Man
History                                 Battleship Potemkin
Horror                                                Psycho
Mystery                                         Rear Window
Sci-Fi                                          Blade Runner
Thriller                                  Shadow of a Doubt
Western                      The Good, the Bad and the Ugly
Name: title, dtype: object
```

18) Check if there are multiple movies with the same title, and if so, determine if they are actually duplicates.

Program-

```
# Answer:
repeated_titles = movies[movies.title.duplicated()].title
#repeated_titles
movies[movies.title.isin(repeated_titles)]
```

Output-

	star_rating	title	content_rating	genre	duration	actors_list
466	7.9	The Girl with the Dragon Tattoo	R	Crime	158	[u'Daniel Craig', u'Rooney Mara', u'Christophe...
482	7.8	The Girl with the Dragon Tattoo	R	Crime	152	[u'Michael Nyqvist', u'Noomi Rapace', u'Ewa Fr...
662	7.7	True Grit	PG-13	Adventure	110	[u'Jeff Bridges', u'Matt Damon', u'Hailee Stei...
678	7.7	Les Miserables	PG-13	Drama	158	[u'Hugh Jackman', u'Russell Crowe', u'Anne Hat...
703	7.6	Dracula	UNRATED	Horror	85	[u'Bela Lugosi', u'Helen Chandler', u'David Ma...
905	7.5	Dracula	R	Horror	128	[u'Gary Oldman', u'Winona Ryder', u'Anthony Ho...
924	7.5	Les Miserables	PG-13	Crime	134	[u'Liam Neeson', u'Geoffrey Rush', u'Uma Thurm...
936	7.4	True Grit	UNRATED	Adventure	128	[u'John Wayne', u'Kim Darby', u'Glen Campbell']

19) Calculate the average star rating for each genre, but only include genres with at least 10 movies

Program-

```
#Answer
movies.groupby('genre').star_rating.mean()[movies.genre.value_counts() >= 10]
```

Output-

```
genre
Action       7.884559
Adventure    7.933333
Animation    7.914516
Biography    7.862338
Comedy       7.822436
Crime        7.916935
Drama        7.902518
Horror       7.806897
Mystery      7.975000
Name: star_rating, dtype: float64
```

Name of the Laboratory:_____ **Name of the Experiment:** _____

Experiment No:_____ **Date:** _____

WEEK 7

1. **Importing Data**

```
[1]  import pandas as pd
     import numpy as np
     from collections import Counter
     import warnings
     warnings.filterwarnings("ignore")
```

```
[3]  train_input=pd.read_csv("Credit_Risk_Train_Data.csv")
     test_input=pd.read_csv("Credit_Risk_Validate_Data.csv")
```

```
     print (train_input.columns)
     print (test_input.columns)
```

Output:

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

---

2. **Make column names same in test and train dataset so that missing values can be filled simultaneously.**

```
[5]  #the last column has a different name in both,
     #lets make the names same. and then merge them togehter
     #so that we can fill the missing values simulteneously
     test_input.rename(columns={"outcome": "Loan_Status"},inplace=True)
```

```
[6]  data_all=pd.concat([train_input,test_input],axis=0)
     data_all.shape
```

Output:

```
(981, 13)
```

3. Reset Data index.

```
[8] data_all.reset_index(inplace=True,drop=True)
    # reset index else merging will have issues,
    print (data_all.tail())
```

Output:

```
     Loan_ID Gender Married Dependents     Education Self_Employed  \
976  LP002971   Male     Yes         3+  Not Graduate           Yes
977  LP002975   Male     Yes          0      Graduate            No
978  LP002980   Male      No          0      Graduate            No
979  LP002986   Male     Yes          0      Graduate            No
980  LP002989   Male      No          0      Graduate           Yes

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
976             4009             1777.0       113.0             360.0
977             4158              709.0       115.0             360.0
978             3250             1993.0       126.0             360.0
979             5000             2393.0       158.0             360.0
980             9200                0.0        98.0             180.0

     Credit_History Property_Area Loan_Status
976             1.0         Urban           Y
977             1.0         Urban           Y
978             NaN     Semiurban           Y
979             1.0         Rural           N
980             1.0         Rural           Y
```

4. Obtain missing values of all columns.

```
[9] data_all.isnull().sum()#gives the missing value of all columns
```

Output:

```
Loan_ID               0
Gender               24
Married               3
Dependents           25
Education             0
Self_Employed        55
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           27
Loan_Amount_Term     20
Credit_History       79
Property_Area         0
Loan_Status           0
dtype: int64
```

5. **Obtain number of rows and columns.**

```
[10] data_all.shape # read the decription of each column from the word document
```

Output:

```
(981, 13)
```

6. **What is the distribution of genders in the 'Gender' column of the 'data_all' dataset?**

```
[12] Counter(data_all['Gender'])
```

Output:

```
Counter({'Male': 775, 'Female': 182, nan: 24})
```

---

7. **Find out how many people didn't provide their gender?**

```
[14] #lets fill them by Male
     print (data_all[data_all['Gender'].isnull()].index.tolist())
     #these rows are null for gender
     #lets fill them with the Model of Gender  i.e Male
     gender_null=data_all[data_all['Gender'].isnull()].index.tolist()
```

Output:

```
[23, 126, 171, 188, 314, 334, 460, 467, 477, 507, 576, 588, 592, 636, 665, 720, 752, 823, 845, 859, 893, 910, 917, 932]
```

---

8. **Handle missing values in a dataset by particularly replacing them with values based on other features in the data.**

```
[15] gender_null_M = gender_null[:12]
     gender_null_F = gender_null[12:]
     data_all['Gender'].iloc[gender_null_M]="Male"
     data_all['Gender'].iloc[gender_null_F]="Female"
```

```
[16] data_all['Gender'].iloc[gender_null]="Male"
```

```
#check if filed
print (sum(data_all['Gender'].isnull()))#oky done
Counter(data_all['Gender'])
```

Output:

```
0
Counter({'Male': 799, 'Female': 182})
```

---

9. What is the distribution of 'Married' column in the dataset?

```
[18] #lets fill Married now
     print (Counter(data_all['Married']))#most are married
```

Output:

```
Counter({'Yes': 631, 'No': 347, nan: 3})
```

---

10. Find out the indexes where 'Married' field is nan.

```
[19] married_null=data_all[data_all['Married'].isnull()].index.tolist()
     married_null
```

Output:

```
[104, 228, 435]
```

---

11. Fill the nan values with 'yes'.

```
[20] data_all['Married'].iloc[married_null]="Yes"
```

```
[21] data_all.isnull().sum()
```

Output:

```
Loan_ID              0
Gender               0
Married              0
Dependents          25
Education            0
Self_Employed       55
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          27
Loan_Amount_Term    20
Credit_History      79
Property_Area        0
Loan_Status          0
dtype: int64
```

---

12. W hat is the distribution of 'Dependants' column in the dataset?

```
[22] Counter(data_all['Dependents'])
```

Output:

```
Counter({'0': 545, '1': 160, '2': 160, '3+': 91, nan: 25})
```

13. How many null values are there in the 'Dependents' column of the DataFrame 'data_all', grouped by the 'Married' status?

```
[23] # lets see the Dependents wrt Marriage
     pd.crosstab(data_all['Married'],data_all['Dependents'].isnull())
```

Output:

Married	Dependents False	Dependents True
No	338	9
Yes	618	16

---

14. How is the distribution of individuals across different numbers of dependents related to their marital status in the dataset 'data_all'?

```
[24] pd.crosstab(data_all['Dependents'],data_all['Married'])
```

Output:

Dependents	Married No	Married Yes
0	276	269
1	36	124
2	14	146
3+	12	79

---

15. What is the list of index values for individuals in the 'data_all' DataFrame who are unmarried ('Married' status is "No") and have missing values in the 'Dependents' column?

```
[26] # for the bachelors, lets fill the missing dependents as 0
     #lets find the index of all rows with Depednednts mssing and Married NO
     bachelor_nulldependent= data_all[(data_all['Married']=="No") &
                         (data_all['Dependents'].isnull())].index.tolist()
     print (bachelor_nulldependent)
```

Output:

```
[293, 332, 355, 597, 684, 752, 879, 916, 926]
```

---

16. What are the counts of different values in the 'Dependents' column after replacing missing values with '0' for individuals who are unmarried?

```
[27] data_all['Dependents'].iloc[bachelor_nulldependent]='0'
```

```
[28] Counter(data_all['Dependents'])
```

Output:

```
Counter({'0': 554, '1': 160, '2': 160, '3+': 91, nan: 16})
```

---

17. How many dependents are reported by males and females respectively, among the remaining 16 missing dependent values in the 'data_all' DataFrame?

```
#for the remaining 16 missing depemdents,
#let see how mnay dependents Male & Female have
pd.crosstab(data_all['Gender'],data_all['Dependents'])
```

Output:

Dependents	0	1	2	3+
Gender				
Female	127	32	13	9
Male	427	128	147	82

---

18. How many missing values are there in the 'Dependents' column of the 'data_all' DataFrame, categorized by gender?

```
pd.crosstab(data_all['Gender'],data_all['Dependents'].isnull())
```

Output:

Dependents	False	True
Gender		
Female	181	1
Male	784	15

---

19. Retrieve the gender information for individuals with missing values in the 'Dependents' column in the 'data_all' DataFrame.

```
# so feamle have less dependents
#lets see the gender of the 16 missing dependents
data_all['Gender'].iloc[data_all[data_all['Dependents'].isnull()].
                        index.tolist()]
```

Output:

```
102      Male
104      Male
120      Male
226      Male
228      Male
301      Male
335      Male
346      Male
435    Female
517      Male
571      Male
660      Male
725      Male
816      Male
861      Male
865      Male
Name: Gender, dtype: object
```

---

20. How does the number of dependents vary between males and females in the dataset 'data_all'?

```
[32] # all of them are Male
     # lets fill them with the mode of all dependent of male
     pd.crosstab(data_all['Gender'],data_all['Dependents'])
```

Output:

Dependents	0	1	2	3+
Gender				
Female	127	32	13	9
Male	427	128	147	82

---

21. How does the distribution of dependents differ between married males and females in the dataset 'data_all'?

```
help(pd.crosstab)
```

```
[34] pd.crosstab((data_all['Gender']=='Male')&
                 (data_all['Married']=='Yes') ,data_all['Dependents'])
```

Output:

Dependents	0	1	2	3+
row_0				
False	318	48	23	15
True	236	112	137	76

---

22. After assigning the value "1" to the missing entries in the 'Dependents' column, how many missing values remain in each column of the 'data_all' DataFrame?

```
[35] #let sfill the # dependent with 1
     data_all['Dependents'].iloc[data_all[data_all['Dependents'].isnull()].
                                 index.tolist()]="1"
```

```
[36] data_all.isnull().sum()
```

Output:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed        55
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           27
Loan_Amount_Term     20
Credit_History       79
Property_Area         0
Loan_Status           0
dtype: int64
```

23. Could you please provide the count of occurrences for each unique value in the 'Self_Employed' column of the 'data_all' DataFrame?

```
[37] Counter(data_all['Self_Employed'])
```

Output:

```
Counter({'No': 807, 'Yes': 119, nan: 55})
```

---

24. After replacing missing values in the 'Self_Employed' column with "No", could you share the count of missing values in each column of the 'data_all' DataFrame?

```
[38] self_emp_null=data_all[data_all['Self_Employed'].isnull()].index.tolist()

[39] #fill missing selfemployed with NO
     data_all['Self_Employed'].iloc[self_emp_null]="No"

[40] data_all.isnull().sum()
```

Output:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          27
Loan_Amount_Term    20
Credit_History      79
Property_Area        0
Loan_Status          0
dtype: int64
```

---

25. How does the presence of missing values in the 'LoanAmount' column relate to the presence of missing values in the 'Loan_Amount_Term' column in the 'data_all' DataFrame?

```
[41] # To check if any row with both LoanAmount and Loan_Amount_Term as NAN
     pd.crosstab(data_all['LoanAmount'].isnull(),
                 data_all['Loan_Amount_Term'].isnull())
```

Output:

Loan_Amount_Term	False	True
LoanAmount		
False	934	20
True	27	0

---

26. What is the distribution of missing values in the 'LoanAmount' column based on different values of the 'Loan_Amount_Term' column in the 'data_all' DataFrame?

```
[42] pd.crosstab(data_all['LoanAmount'].isnull(),data_all['Loan_Amount_Term'])
```

Output:

Loan_Amount_Term	6.0	12.0	36.0	60.0	84.0	120.0	180.0	240.0	300.0	350.0	360.0	480.0
LoanAmount												
False	1	2	3	3	7	4	64	7	20	1	800	22
True	0	0	0	0	0	0	2	1	0	0	23	1

27. Could you please provide the mean loan amount for each unique value of the 'Loan_Amount_Term' column in the 'data_all' DataFrame?

```
[43] data_all.groupby(data_all['Loan_Amount_Term'])['LoanAmount'].mean()
```

Output:

```
Loan_Amount_Term
6.0       95.000000
12.0     185.500000
36.0     117.666667
60.0     139.666667
84.0     121.142857
120.0     36.750000
180.0    131.125000
240.0    128.857143
300.0    166.250000
350.0    133.000000
360.0    144.420000
480.0    137.181818
Name: LoanAmount, dtype: float64
```

---

28. After assigning specific loan amounts for missing values based on the loan term, how many missing values remain in the 'LoanAmount' column of the 'data_all' DataFrame?

```
[44] #lets fill the missing values in LoanAmount
     #with the mean of the respective Loan_Term
     #we see that 180 & 240 has the almost same Loan amount 128-131
     #& 360 has high i.e 144
     #so lets fill only 360 by 144
     #and all remaining by 130
     data_all['LoanAmount'][(data_all['LoanAmount'].isnull())
                            & (data_all['Loan_Amount_Term']==360)]=144
     data_all['LoanAmount'][(data_all['LoanAmount'].isnull())
                            & (data_all['Loan_Amount_Term']==480)]=137
```

```
data_all['LoanAmount'][(data_all['LoanAmount'].isnull())]=130
[46] #lets fill Loan Amount Term
     (data_all['Loan_Amount_Term']).value_counts()
```

Output:

```
Loan_Amount_Term
360.0    823
180.0     66
480.0     23
300.0     20
240.0      8
84.0       7
120.0      4
60.0       3
36.0       3
12.0       2
350.0      1
6.0        1
Name: count, dtype: int64
```

---

29. After replacing missing values in the 'Loan_Amount_Term' column with a value of 360, could you share the count of missing values in each column of the 'data_all' DataFrame?

```
[47] #lets fill the Loan Tenure by the mode i,e 360
     data_all['Loan_Amount_Term'][data_all['Loan_Amount_Term'].isnull()]=360
```

```
[48] data_all.isnull().sum()
```

Output:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History      79
Property_Area        0
Loan_Status          0
dtype: int64
```

---

30. What are the counts of different values in the 'Credit_History' column of the 'data_all' DataFrame?

```
[49] data_all['Credit_History'].value_counts()
```

Output:

```
Credit_History
1.0    754
0.0    148
Name: count, dtype: int64
```

---

31. How is the distribution of credit history related to gender in the 'data_all' DataFrame?

```
[50] pd.crosstab(data_all['Gender'],data_all['Credit_History'])
     # Gender makes no difference
```

Output:

Credit_History	0.0	1.0
Gender		
Female	30	135
Male	118	619

---

32. How is the distribution of credit history related to self-employment status in the 'data_all' DataFrame?

```
[51] pd.crosstab(data_all['Self_Employed'],data_all['Credit_History'])
     # Self_Employed makes no difference
```

Output:

Credit_History	0.0	1.0
Self_Employed		
No	134	658
Yes	14	96

VNR VJIET

Name of the Laboratory: _____ **Name of the Experiment: _____**

Experiment No:_____Date: _____

33. How does the education level relate to credit history in the 'data_all' DataFrame?

```
pd.crosstab(data_all['Education'],data_all['Credit_History'])
# Education makes no difference
```

Output:

Education	Credit_History 0.0	1.0
Graduate	106	596
Not Graduate	42	158

---

34. How does the marital status relate to credit history in the 'data_all' DataFrame?

```
pd.crosstab(data_all['Married'],data_all['Credit_History'])
# married makes no difference
```

Output:

Married	Credit_History 0.0	1.0
No	56	263
Yes	92	491

---

35. After replacing missing values in the 'Credit_History' column with a value of 1, could you provide the count of missing values in each column of the 'data_all' DataFrame?

```
[54] data_all['Credit_History'][data_all['Credit_History'].isnull()]=1
```

```
data_all.isnull().sum()
```

Output:

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

---

36. Given the series x = ['a', 'b', 'c', 'a', 'a'], could you display its contents?

```
help(pd.get_dummies)
```

```
[57] x = pd.Series(list('abcaa'))
     print (x)
```

Output:

```
0    a
1    b
2    c
3    a
4    a
dtype: object
```

---

37. What does the one-hot encoding of the series x produce?

```
[58] pd.get_dummies(x)
```

Output:

	a	b	c
0	True	False	False
1	False	True	False
2	False	False	True
3	True	False	False
4	True	False	False

---

38. What does the one-hot encoding of the series x produce, with the parameter drop_first set to True?

```
[59] pd.get_dummies(x, drop_first=True)
```

Output:

	b	c
0	False	False
1	True	False
2	False	True
3	False	False
4	False	False

---

39. Could you display the first few rows of the 'data_all' DataFrame?

```
[60] data_all.head()
```

Output:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	144.0	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

---

40. After performing one-hot encoding on the 'data_all' DataFrame by dropping the 'Loan_ID' column and setting `drop_first` to True, could you show the first few rows of the resulting DataFrame 'data_all_new'?

```
[63] data_all_new=pd.get_dummies(data_all.drop(['Loan_ID'],axis=1),
                                 drop_first=True)
```

```
data_all_new.head()
```

Output:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Property_Area_Urban	L
0	5849	0.0	144.0	360.0	1.0	True	False	False	False	False	False	False	False	True	
1	4583	1508.0	128.0	360.0	1.0	True	True	True	False	False	False	False	False	False	
2	3000	0.0	66.0	360.0	1.0	True	True	False	False	False	False	True	False	True	
3	2583	2358.0	120.0	360.0	1.0	True	True	False	False	False	True	False	False	True	
4	6000	0.0	141.0	360.0	1.0	True	False	False	False	False	False	False	False	True	

41. Could you display the first few rows of the feature matrix X after removing the target variable 'Loan_Status_Y'?

```
[65] X=data_all_new.drop(['Loan_Status_Y'],axis=1)
     y=data_all_new['Loan_Status_Y']
```

```
[66] X.head()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiurban	Property_Area_Urban
0	5849	0.0	144.0	360.0	1.0	True	False	False	False	False	False	False	False	True
1	4583	1508.0	128.0	360.0	1.0	True	True	True	False	False	False	False	False	False
2	3000	0.0	66.0	360.0	1.0	True	True	False	False	False	False	True	False	True
3	2583	2358.0	120.0	360.0	1.0	True	True	False	False	False	True	False	False	True
4	6000	0.0	141.0	360.0	1.0	True	False	False	False	False	False	False	False	True

42. Could you display the first few values of the target variable y?

```
y.head()
```

Output:

```
0     True
1    False
2     True
3     True
4     True
Name: Loan_Status_Y, dtype: bool
```

Train Test Split

43. After splitting the dataset into training and testing sets, could you provide the shape of the training feature matrix X_train?

```
[68] from sklearn.model_selection import train_test_split
```

```
help(train_test_split)
```

```
[70] X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
[ ] scaler = StandardScaler()
```

Output:

```
(735, 14)
```

---

44. What is the shape of the testing feature matrix X_test?

```
X_test.shape
```

Output:

```
(246, 14)
```

Data Preprocessing :

45. Prior to scaling the data, could you confirm if the scaler object has been fitted to the training data?

```
[73] from sklearn.preprocessing import StandardScaler

[74] scaler = StandardScaler()

[75] # Fit only to the training data
     scaler.fit(X_train)
```

Output:

```
▾ StandardScaler
StandardScaler()
```

---

46. After scaling the training and testing feature matrices, could you display the first five rows of the scaled training data X_train?

```
[76] # Now apply the transformations to the data:
     X_train = scaler.transform(X_train)
     X_test = scaler.transform(X_test)
```

```
X_train[:5]
```

Output:

```
array([[ 0.16254279,  0.89814832,  2.37678203,  0.26198502,  0.42399915,
         0.46137976,  0.72013543,  2.10818511, -0.46137976, -0.30883657,
        -0.54433105, -0.3543646 , -0.73764331,  1.35971953],
       [-0.53723664,  0.03788176, -1.2524516 ,  0.26198502,  0.42399915,
         0.46137976,  0.72013543, -0.47434165,  2.16741192, -0.30883657,
         1.83711731, -0.3543646 ,  1.35566878, -0.73544579],
       [-0.19732836, -0.5596399 , -0.64121225,  0.26198502,  0.42399915,
         0.46137976,  0.72013543,  2.10818511, -0.46137976, -0.30883657,
        -0.54433105, -0.3543646 ,  1.35566878, -0.73544579],
       [-0.32609784,  0.19497988, -0.10637782,  0.26198502,  0.42399915,
```

Training the model using SVM :

47. Using a support vector classifier with a linear kernel and regularization parameter C=1.0, could you fit the model to the training data?

```
[78] from sklearn import svm
```

```
[79] clf = svm.SVC(kernel='linear', C = 1.0)
     clf.fit(X_train,y_train)
```

Output:

```
SVC(kernel='linear')
```

48. After making predictions on the testing data, could you print the confusion matrix?

```
[80] predictions = clf.predict(X_test)
```

```
[81] from sklearn.metrics import classification_report,confusion_matrix
```

```
[82] print(confusion_matrix(y_test,predictions))
```

Output:

```
[[ 33  35]
 [  3 175]]
```

49. Could you print the classification report for the predictions on the testing data?

```
print(classification_report(y_test,predictions))
```

Output:

	precision	recall	f1-score	support
False	0.92	0.49	0.63	68
True	0.83	0.98	0.90	178
accuracy			0.85	246
macro avg	0.88	0.73	0.77	246
weighted avg	0.86	0.85	0.83	246

Submission :

50. Could you display the column names of the 'Sub_Test_input' DataFrame?

```
[84] Sub_Test_input=pd.read_csv("Credit_Risk_Test_Data.csv")
```

```
[85] print(Sub_Test_input.columns)
```

Output:

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

---

51. What is the shape of the 'Sub_Test_input' DataFrame?

```
[86] Sub_Test_input.shape
```

Output:

```
(367, 12)
```

---

52. How many missing values are there in each column of the 'Sub_Test_input' DataFrame?

```
Sub_Test_input.isnull().sum()#gives the missing value of all columns
```

Output:

```
Loan_ID               0
Gender               11
Married               0
Dependents           10
Education             0
Self_Employed        23
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
Property_Area         0
dtype: int64
```

---

53. Could you provide the indices of rows where the 'Gender' column is missing in the 'Sub_Test_input' DataFrame?

```
[ ] gender_null=Sub_Test_input[Sub_Test_input['Gender'].isnull()].index.tolist()
    print(gender_null)
```

Output:

```
[22, 51, 106, 138, 209, 231, 245, 279, 296, 303, 318]
```

---

54. After replacing missing values in the 'Gender' column with "Male" in the 'Sub_Test_input' DataFrame, how many missing values remain in each column?

```
[89] Sub_Test_input['Gender'].iloc[gender_null]="Male"
```

```
Sub_Test_input.isnull().sum()#gives the missing value of all columns
```

Output:

```
Loan_ID              0
Gender               0
Married              0
Dependents          10
Education            0
Self_Employed       23
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           5
Loan_Amount_Term     6
Credit_History      29
Property_Area        0
dtype: int64
```

---

55. Could you provide the indices of rows where individuals are unmarried ('Married' is "No") and have missing values in the 'Dependents' column in the 'Sub_Test_input' DataFrame?

```
[91] bachelor_nulldependent=Sub_Test_input[(Sub_Test_input['Married']=="No") &
                                           (Sub_Test_input['Dependents'].isnull())].index.tolist()
     print (bachelor_nulldependent)
```

Output:

```
[70, 138, 265, 302, 312]
```

---

56. After filling missing values in the 'Dependents' column with "0" for unmarried individuals and "1" for the remaining missing values in the 'Sub_Test_input' DataFrame, how many missing values are left in each column?

```
Sub_Test_input['Dependents'].iloc[bachelor_nulldependent]='0'
```

```
[93] Sub_Test_input['Dependents'].iloc[Sub_Test_input[Sub_Test_input['Dependents'].isnull()].index.tolist()]="1"
```

```
[94] Sub_Test_input.isnull().sum()#gives the missing value of all columns
```

Output:

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed       23
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           5
Loan_Amount_Term     6
Credit_History      29
Property_Area        0
dtype: int64
```

---

57. Could you provide the indices of rows where the 'Self_Employed' column is missing in the 'Sub_Test_input' DataFrame?

```
self_emp_null=Sub_Test_input[Sub_Test_input['Self_Employed'].isnull()].index.tolist()
print(self_emp_null)
```

Output:

```
[8, 11, 13, 36, 72, 89, 142, 161, 168, 175, 192, 243, 255, 259, 276, 278, 285, 287, 294, 297, 301, 323, 326]
```

---

58. After filling missing values in the 'Self_Employed' column with "No" in the 'Sub_Test_input' DataFrame, how many missing values remain in each column?

```
[96] #fill missing selfemployed with NO
     Sub_Test_input['Self_Employed'].iloc[self_emp_null]="No"
```

```
Sub_Test_input.isnull().sum()#gives the missing value of all columns
```

Output:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      6
Credit_History       29
Property_Area         0
dtype: int64
```

---

59. Are there any rows in the 'Sub_Test_input' DataFrame where both 'LoanAmount' and 'Loan_Amount_Term' are missing?

```
[98] # To check if any row with both LoanAmount and Loan_Amount_Term as NAN
     pd.crosstab(Sub_Test_input['LoanAmount'].isnull(),Sub_Test_input['Loan_Amount_Term'].isnull())
```

Output:

Loan_Amount_Term	False	True
LoanAmount		
False	356	6
True	5	0

---

60. What is the distribution of missing values in the 'LoanAmount' column based on different values of the 'Loan_Amount_Term' column in the 'Sub_Test_input' DataFrame?

```
pd.crosstab(Sub_Test_input['LoanAmount'].isnull(),Sub_Test_input['Loan_Amount_Term'])
```

Output:

Loan_Amount_Term	6.0	12.0	36.0	60.0	84.0	120.0	180.0	240.0	300.0	350.0	360.0	480.0
LoanAmount												
False	1	1	1	1	3	1	22	4	7	1	307	7
True	0	0	0	0	0	0	0	0	0	0	4	1

---

61. After assigning specific loan amounts for missing values based on the loan term in the 'Sub_Test_input' DataFrame, how many missing values remain in each column?

```
[100] Sub_Test_input['LoanAmount'][(Sub_Test_input['LoanAmount'].isnull()) & (Sub_Test_input['Loan_Amount_Term']==360)]=144
      Sub_Test_input['LoanAmount'][(Sub_Test_input['LoanAmount'].isnull()) & (Sub_Test_input['Loan_Amount_Term']==480)]=137
```

```
[101] Sub_Test_input.isnull().sum()#gives the missing value of all columns
```

Output:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      6
Credit_History       29
Property_Area         0
dtype: int64
```

---

62. After filling missing values in the 'Loan_Amount_Term' column with the mode value of 360 in the 'Sub_Test_input' DataFrame, could you provide the count of missing values in each column?

```
[102] #lets fill the Loan Tenure by the mode i,e 360
      Sub_Test_input['Loan_Amount_Term'][Sub_Test_input['Loan_Amount_Term'].isnull()]=360
```

Output:

```
Loan_ID               0
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            0
Loan_Amount_Term      0
Credit_History       29
Property_Area         0
dtype: int64
```

---

63. After replacing missing values in the 'Credit_History' column with a value of 1 in the 'Sub_Test_input' DataFrame, could you provide the count of missing values in each column?

```
[104] Sub_Test_input['Credit_History'][Sub_Test_input['Credit_History'].isnull()]=1
```

```
Sub_Test_input.isnull().sum()#gives the missing value of all columns
```

Output:

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

---

64. After performing one-hot encoding on the 'Sub_Test_input' DataFrame by dropping the 'Loan_ID' column and setting `drop_first` to True, could you display the first few rows of the resulting DataFrame 'Sub_Test_New'?

```
Sub_Test_New=pd.get_dummies(Sub_Test_input.drop(['Loan_ID'],axis=1),drop_first=True)
```

```
[107] Sub_Test_New.head()
```

Output:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_Male	Married_Yes	Dependents_1	Dependents_2	Dependents_3+	Education_Not Graduate	Self_Employed_Yes	Property_Area_Semiu
0	5720	0	110.0	360.0	1.0	True	True	False	False	False	False	False	F
1	3076	1500	126.0	360.0	1.0	True	True	True	False	False	False	False	F
2	5000	1800	208.0	360.0	1.0	True	True	False	True	False	False	False	F
3	2340	2546	100.0	360.0	1.0	True	True	False	True	False	False	False	F
4	3276	0	78.0	360.0	1.0	True	False	False	False	False	True	False	F

65. After scaling the features of the 'Sub_Test_New' DataFrame using the previously fitted scaler, could you display the first five rows of the scaled testing data X_test?

```
[108] X_test = scaler.transform(Sub_Test_New)
```

```
X_test[:5]
```

Output:

```
array([[ 0.09725881, -0.5596399 , -0.42473165,  0.26198502,  0.42399915,
         0.46137976,  0.72013543, -0.47434165, -0.46137976, -0.30883657,
        -0.54433105, -0.3543646 , -0.73764331,  1.35971953],
       [-0.37825308, -0.04512641, -0.2209852 ,  0.26198502,  0.42399915,
         0.46137976,  0.72013543,  2.10818511, -0.46137976, -0.30883657,
        -0.54433105, -0.3543646 , -0.73764331,  1.35971953],
       [-0.03223005,  0.05777629,  0.82321535,  0.26198502,  0.42399915,
         0.46137976,  0.72013543, -0.47434165,  2.16741192, -0.30883657,
        -0.54433105, -0.3543646 , -0.73764331,  1.35971953],
       [-0.51061948,  0.313661  , -0.55207318,  0.26198502,  0.42399915,
         0.46137976,  0.72013543, -0.47434165,  2.16741192, -0.30883657,
        -0.54433105, -0.3543646 , -0.73764331,  1.35971953],
       [-0.34228395, -0.5596399 , -0.83222455,  0.26198502,  0.42399915,
         0.46137976, -1.38862769, -0.47434165, -0.46137976, -0.30883657,
         1.83711731, -0.3543646 , -0.73764331,  1.35971953]])
```

66. After making predictions on the testing data, could you display the first five predictions?

```
[110] predictions = clf.predict(X_test)
```

```
[111] predictions[:5]
```

Output:

```
array([ True,  True,  True,  True,  True])
```

67. Could you display the first few rows of the 'Final_df' DataFrame, which contains the original data along with the predicted loan status?

```
[112] Final_df = Sub_Test_input
```

```
Final_df.head()
```

Output:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	1.0	Urban
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	1.0	Urban
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	1.0	Urban
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	1.0	Urban
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	1.0	Urban

68. Could you display the first few rows of the 'Final_df' DataFrame with the predicted loan status added as a new column?

```
[114] Final_df['Loan_Status']=predictions
```

```
[115] Final_df.head()
```

Output:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	1.0	Urban	True
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	1.0	Urban	True
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	1.0	Urban	True
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	1.0	Urban	True
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	1.0	Urban	True

69. After replacing the numerical representation of loan status predictions with 'Y' for approved and 'N' for not approved in the 'Final_df' DataFrame, could you display the first few rows?

```
[116] Final_df = Final_df.replace({'Loan_Status': {0:'N',1:'Y'}})
```

```
[117] Final_df.head()
```

Output:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	1.0	Urban	True
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	1.0	Urban	True
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	1.0	Urban	True
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	1.0	Urban	True
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	1.0	Urban	True

70. How many loan statuses are predicted as approved ('Y') and not approved ('N') in the 'Final_df' DataFrame?

```
help(pd.DataFrame.replace)
```

```
[119] Counter(Final_df['Loan_Status'])
```

Output:

```
Counter({True: 308, False: 59})
```

---

71. Save the 'Final_df' DataFrame to a CSV file named "2_Credit_Risk_Submission.csv" without including the index?

```
Final_df.to_csv("2_Credit_Risk_Submission.csv",index = False)
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

WEEK-8
ORDINAL NUMBER ENCODING

1) Retrieve the current date and time?

```
[1]: import datetime
     today_date=datetime.datetime.today()
     today_date
```

Output:

```
[1]: datetime.datetime(2024, 3, 26, 15, 58, 7, 592912)
```

2) Get the date three days ago from the current date

```
[2]: today_date-datetime.timedelta(3)# It will subtract 3 from todays date
```

Output:

```
[2]: datetime.datetime(2024, 3, 23, 15, 58, 7, 592912)
```

3) Generate a list of dates representing the last 15 days, starting from today

```
[4]: [today_date-datetime.timedelta(x) for x in range(0,15)]#last fifteen days data
```

Output:

```
[4]: [datetime.datetime(2024, 3, 26, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 25, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 24, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 23, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 22, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 21, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 20, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 19, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 18, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 17, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 16, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 15, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 14, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 13, 15, 58, 7, 592912),
      datetime.datetime(2024, 3, 12, 15, 58, 7, 592912)]
```

4) Create a list of the past 15 days using list comprehension

```
[5]: #### List Comprehension
     days=[today_date-datetime.timedelta(x) for x in range(0,15)]
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

5) Create a Data Frame from a list of dates and assign a column name in pandas

```
[6]: import pandas as pd
     data=pd.DataFrame(days)
     data.columns=["Day"]

[7]: data.head()
```

Output:

[7]:

	Day
0	2024-03-26 15:58:07.592912
1	2024-03-25 15:58:07.592912
2	2024-03-24 15:58:07.592912
3	2024-03-23 15:58:07.592912
4	2024-03-22 15:58:07.592912

6) Extract the day of the month from a pandas Data Frame containing datetime objects?

```
[8]: data['Day'].dt.day
```

Output:

```
[8]: 0     26
     1     25
     2     24
     3     23
     4     22
     5     21
     6     20
     7     19
     8     18
     9     17
     10    16
     11    15
     12    14
     13    13
     14    12
     Name: Day, dtype: int32
```

7) Extract the name of the day (like Monday or Tuesday) from a pandas Data Frame with datetime objects

```
[9]: data['Day'].dt.day_name()
```

Output:

```
[9]: 0       Tuesday
     1        Monday
     2        Sunday
     3      Saturday
     4        Friday
     5      Thursday
     6     Wednesday
     7       Tuesday
     8        Monday
     9        Sunday
     10     Saturday
     11       Friday
     12     Thursday
     13    Wednesday
     14      Tuesday
     Name: Day, dtype: object
```

8) Add a new column to a panda Data Frame containing the name of the day for each date

```
[10]: data['weekday']=data['Day'].dt.day_name()
      data.head()
```

Output:

[10]:

	Day	weekday
0	2024-03-26 15:58:07.592912	Tuesday
1	2024-03-25 15:58:07.592912	Monday
2	2024-03-24 15:58:07.592912	Sunday
3	2024-03-23 15:58:07.592912	Saturday
4	2024-03-22 15:58:07.592912	Friday

9) Create a dictionary that maps the names of the days of the week to their corresponding numerical order

```
[11]: dictionary={'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturday':6,'Sunday':7

      }
```

```
[12]: dictionary
```

Output:

```
[12]: {'Monday': 1,
       'Tuesday': 2,
       'Wednesday': 3,
       'Thursday': 4,
       'Friday': 5,
       'Saturday': 6,
       'Sunday': 7}
```

10) Add a new column to a pandas Data Frame, mapping day names to their numerical values using a dictionary

```
[13]: data['weekday_ordinal']=data['weekday'].map(dictionary)
```

```
[14]: data
```

Output:

[14]:

	Day	weekday	weekday_ordinal
0	2024-03-26 15:58:07.592912	Tuesday	2
1	2024-03-25 15:58:07.592912	Monday	1
2	2024-03-24 15:58:07.592912	Sunday	7
3	2024-03-23 15:58:07.592912	Saturday	6
4	2024-03-22 15:58:07.592912	Friday	5
5	2024-03-21 15:58:07.592912	Thursday	4
6	2024-03-20 15:58:07.592912	Wednesday	3
7	2024-03-19 15:58:07.592912	Tuesday	2
8	2024-03-18 15:58:07.592912	Monday	1
9	2024-03-17 15:58:07.592912	Sunday	7
10	2024-03-16 15:58:07.592912	Saturday	6
11	2024-03-15 15:58:07.592912	Friday	5
12	2024-03-14 15:58:07.592912	Thursday	4
13	2024-03-13 15:58:07.592912	Wednesday	3

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

COUNT OR FREQUENCY ENCODING

1) Load a CSV file from a given URL into a pandas Data Frame and display the first few rows

```
[1]: import pandas as pd
     train_set = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data' , header = None,index_col=None)
     train_set.head()
```

Output:

[1]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	United-States	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	Cuba	<=50K

2) Identify the data types of all columns in a pandas Data Frame and select specific columns based on their index

```
[2]: columns=[1,3,5,6,7,8,9,13]
```

```
[3]: train_set.dtypes
```

Output:

```
[3]: 0      int64
     1     object
     2      int64
     3     object
     4      int64
     5     object
     6     object
     7     object
     8     object
     9     object
     10     int64
     11     int64
     12     int64
     13    object
     14    object
     dtype: object
```

3) Retrieve the names of columns with object data types from a pandas Data Frame.

```
[4]: col1 = list(train_set.select_dtypes(['object']).columns) # to select columns which are object type

[5]: col1
```

Output:

```
[5]: [1, 3, 5, 6, 7, 8, 9, 13, 14]
```

4) Select specific columns from a pandas Data Frame and then rename them with meaningful names

```
[6]: train_set=train_set[columns]

[7]: train_set.columns=['Employment','Degree','Status','Designation','family_job','Race','Sex','Country']

[8]: train_set.head()
```

Output:

[8]:

	Employment	Degree	Status	Designation	family_job	Race	Sex	Country
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba

5) Determine the number of unique values (labels) in each column

```
[9]: for feature in train_set.columns[:]:
         print(feature,":",len(train_set[feature].unique()),'labels')
```

Output:

```
Employment : 9 labels
Degree : 16 labels
Status : 7 labels
Designation : 15 labels
family_job : 6 labels
Race : 5 labels
Sex : 2 labels
Country : 42 labels
```

6) Count the occurrences of each unique value in a specific column

```
[10]: train_set['Country'].value_counts()
```

Output:

```
[11]: {' United-States': 29170,
       ' Mexico': 643,
       ' ?': 583,
       ' Philippines': 198,
       ' Germany': 137,
       ' Canada': 121,
       ' Puerto-Rico': 114,
       ' El-Salvador': 106,
       ' India': 100,
       ' Cuba': 95,
       ' England': 90,
       ' Jamaica': 81,
       ' South': 80,
       ' China': 75,
       ' Italy': 73,
       ' Dominican-Republic': 70,
       ' Vietnam': 67,
       ' Guatemala': 64,
       ' Japan': 62,
       ' Poland': 60,
       ' Columbia': 59,
       ' Taiwan': 51,
       ' Haiti': 44,
       ' Iran': 43,
       ' Portugal': 37,
       ' Nicaragua': 34,
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

7) Map the values in a pandas Data Frame column to their corresponding counts using a dictionary,

```
[12]: country_map=train_set['Country'].value_counts().to_dict()
```

```
[13]: train_set['Country']=train_set['Country'].map(country_map)
      train_set.head(20)
```

Output:

[13]:

	Employment	Degree	Status	Designation	family_job	Race	Sex	Country
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	29170
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	29170
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	29170
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	29170
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	95
5	Private	Masters	Married-civ-spouse	Exec-managerial	Wife	White	Female	29170
6	Private	9th	Married-spouse-absent	Other-service	Not-in-family	Black	Female	81
7	Self-emp-not-inc	HS-grad	Married-civ-spouse	Exec-managerial	Husband	White	Male	29170
8	Private	Masters	Never-married	Prof-specialty	Not-in-family	White	Female	29170
9	Private	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	29170
10	Private	Some-college	Married-civ-spouse	Exec-managerial	Husband	Black	Male	29170
11	State-gov	Bachelors	Married-civ-spouse	Prof-specialty	Husband	Asian-Pac-Islander	Male	100
12	Private	Bachelors	Never-married	Adm-clerical	Own-child	White	Female	29170
13	Private	Assoc-acdm	Never-married	Sales	Not-in-family	Black	Male	29170

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

Target Guided Ordinal Encoding

1) Read a CSV file into a pandas Data Frame while selecting specific columns, and then display the first few rows

```
[14]: import pandas as pd
      df=pd.read_csv('titanic_fe.csv', usecols=['Cabin','Survived'])
      df.head()
```

Output:

[14]:

	Survived	Cabin
0	0	NaN
1	1	C85
2	1	NaN
3	1	C123
4	0	NaN

2) Replace missing values in a specific column of a pandas Data Frame with a placeholder, and then transform that column to only keep the first character as a string

```
[16]: df['Cabin'].fillna('Missing',inplace=True)
      df['Cabin']=df['Cabin'].astype(str).str[0]
      df.head()
```

[16]:

	Survived	Cabin
0	0	M
1	1	C
2	1	M
3	1	C
4	0	M

[17]:
```
df.groupby(['Cabin'])['Survived'].mean()
```

Output:

[17]:
```
Cabin
A    0.466667
B    0.744681
C    0.593220
D    0.757576
E    0.750000
F    0.615385
G    0.500000
M    0.299854
T    0.000000
Name: Survived, dtype: float64
```

4) get the index of a sorted list based on the mean of a column for each group

[18]:
```
df.groupby(['Cabin'])['Survived'].mean().sort_values().index
```

```
[18]: Index(['T', 'M', 'A', 'G', 'C', 'F', 'B', 'E', 'D'], dtype='object', name='Cabin')
```

5) Extract the sorted index of a grouped Data Frame based on the mean values

```
[19]: ordinal_labels=df.groupby(['Cabin'])['Survived'].mean().sort_values().index
      ordinal_labels
```

Output:

```
[19]: Index(['T', 'M', 'A', 'G', 'C', 'F', 'B', 'E', 'D'], dtype='object', name='Cabin')
```

FEATURE SELECTION

I-UNIVARIATE SELECTION

1) Load a dataset from a CSV file into a pandas Data Frame and separate the features (independent variables) from the target variable (dependent variable)

```
[1]: import pandas as pd
     import numpy as np
     from sklearn.feature_selection import SelectKBest
     from sklearn.feature_selection import chi2
     data = pd.read_csv("train.csv")#mobile dataset
     X = data.iloc[:,0:20]  #independent columns
     y = data.iloc[:,-1]    #target column i.e price range
```

2) Select the top 10 features from a dataset using the Chi-squared test in sklearn

```
[2]: #apply SelectKBest class to extract top 10 best features
     bestfeatures = SelectKBest(score_func=chi2, k=10)
     fit = bestfeatures.fit(X,y)
```

3) Create separate pandas Data Frames for the scores of features and the corresponding feature names

```
[3]: dfscores = pd.DataFrame(fit.scores_)
     dfcolumns = pd.DataFrame(X.columns)
```

4) Concatenate two pandas Data Frames to create a new Data Frame that contains feature names and their corresponding scores

```
[4]: #concat two dataframes for better visualization
     featureScores = pd.concat([dfcolumns,dfscores],axis=1)
     featureScores.columns = ['Specs','Score']  #naming the dataframe columns
```

```
[5]: featureScores
```

Output:

[5]:

	Specs	Score
0	battery_power	14129.866576
1	blue	0.723232
2	clock_speed	0.648366
3	dual_sim	0.631011
4	fc	10.135166
5	four_g	1.521572
6	int_memory	89.839124
7	m_dep	0.745820
8	mobile_wt	95.972863
9	n_cores	9.097556
10	pc	9.186054
11	px_height	17363.569536
12	px_width	9810.586750
13	ram	931267.519053

5) Find and print the top 10 features with the highest scores containing feature scores

```
[6]: print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

Output:

```
            Specs          Score
13            ram  931267.519053
11      px_height   17363.569536
0   battery_power   14129.866576
12       px_width    9810.586750
8       mobile_wt      95.972863
6       int_memory      89.839124
15           sc_w      16.480319
16      talk_time      13.236400
4              fc      10.135166
14           sc_h       9.614878
```

II- FEATURE IMPORTANCE

6) Train a tree-based classifier on a set of features and a target variable

```
[7]: from sklearn.ensemble import ExtraTreesClassifier
     import matplotlib.pyplot as plt
     model = ExtraTreesClassifier()
     model.fit(X,y)
```

Output:

```
[7]: ExtraTreesClassifier()
```

7) Get the feature importances from a trained tree-based classifier

```
[8]: print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers
```

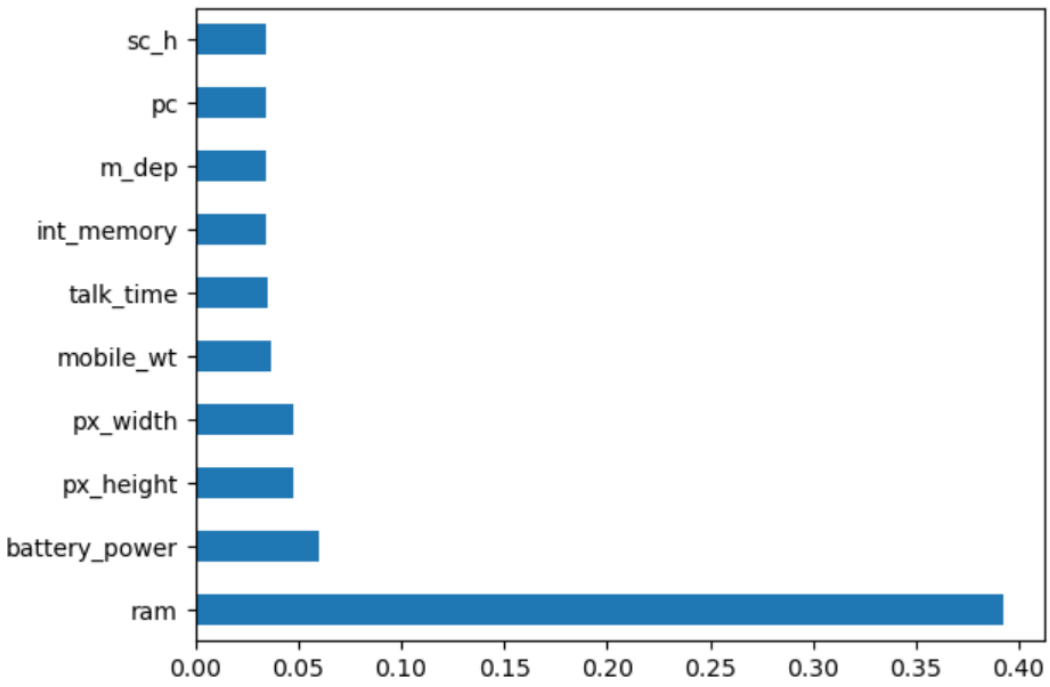
Output:

```
[0.06028759 0.01922831 0.03317357 0.01905613 0.03319318 0.01816559
 0.03462521 0.03439677 0.03638947 0.03307831 0.03434327 0.04796383
 0.04763134 0.39261372 0.03430622 0.03377027 0.03503229 0.01436724
 0.01783262 0.02054506]
```

8) Create a horizontal bar plot to visualize the 10 most important features

```
[9]: #plot graph of feature importances for better visualization
     feat_importances = pd.Series(model.feature_importances_, index=X.columns)
     feat_importances.nlargest(10).plot(kind='barh')
     plt.show()
```

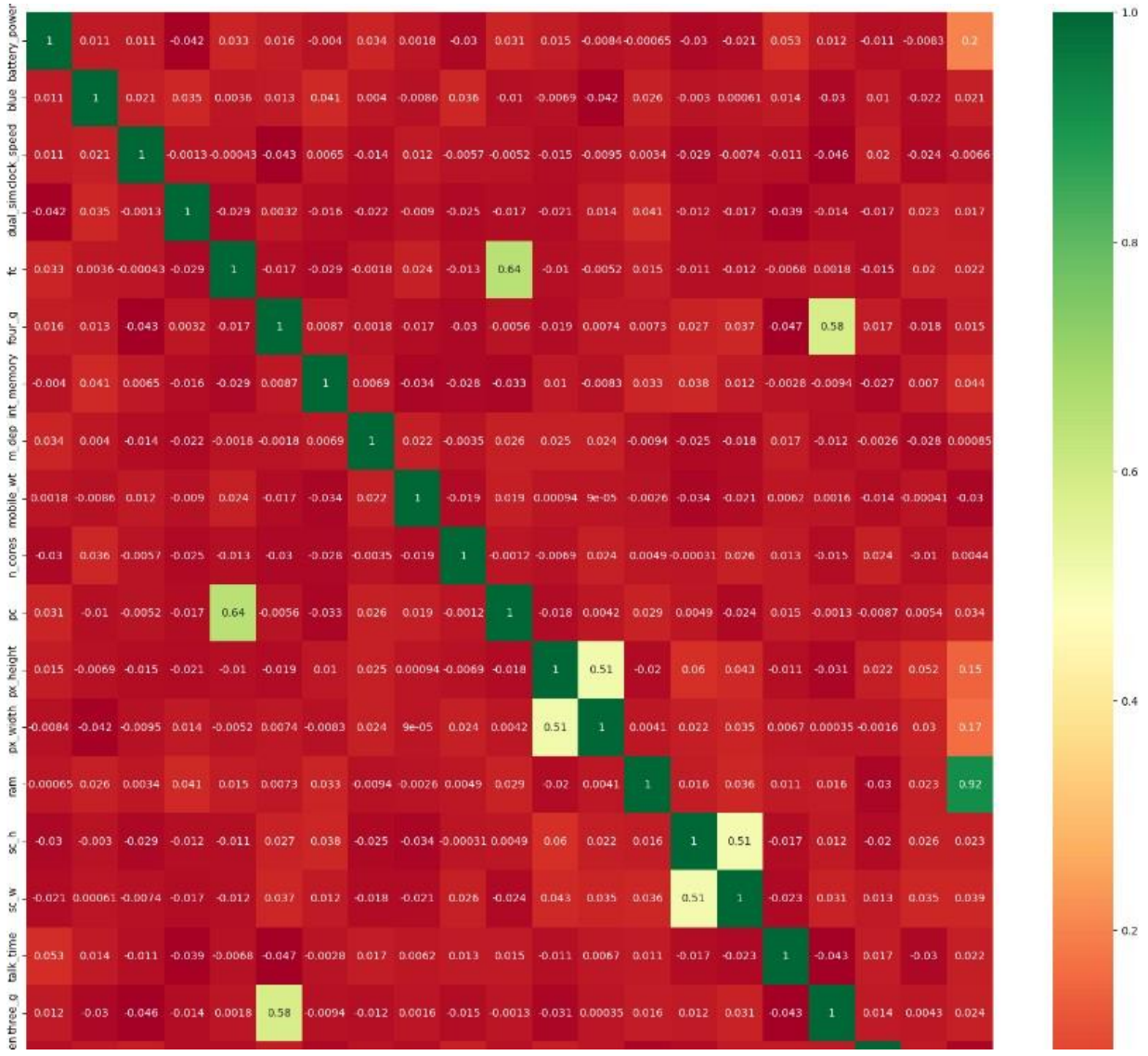
Output:



9) Create a heatmap of feature correlations in a dataset

```
[10]: import seaborn as sns
      #get correlations of each features in dataset
      corrmat = data.corr()
      top_corr_features = corrmat.index
      plt.figure(figsize=(20,20))
      #plot heat map
      g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Output:



VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

NORMALIZATION AND STANDARDIZATION

1) Read a CSV file into a pandas Data Frame while selecting specific columns, and display the first few rows

```
[4]: df=pd.read_csv('titanic_fe.csv',usecols=['Cabin','Survived'])
     df.head()
```

Output:

[4]:

	Survived	Cabin
0	0	NaN
1	1	C85
2	1	NaN
3	1	C123
4	0	NaN

2) Replace NaN values in a specific Data Frame column with a placeholder, and then display the first few rows

```
[5]: ### Replacing
     df['Cabin'].fillna('Missing',inplace=True)
     df.head()
```

Output:

[5]:

	Survived	Cabin
0	0	Missing
1	1	C85
2	1	Missing
3	1	C123
4	0	Missing

3) Find the unique values in a specific column

```
[6]: df['Cabin'].unique()
```

Output:

```
[6]: array(['Missing', 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
            'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
            'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
            'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
            'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
            'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
            'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
            'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
            'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
            'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
            'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
            'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
            'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
            'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
            'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
            'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
            'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
            'C148'], dtype=object)
```

4) Transform a Data Frame column to keep only the first character as a string

```
[7]: df['Cabin']=df['Cabin'].astype(str).str[0]  # astype(str) will convert string into array of characters
     df.head()
```

Output:

[7]:

	Survived	Cabin
0	0	M
1	1	C
2	1	M
3	1	C
4	0	M

5) Obtain all unique values from Cabin column

```
[8]: df.Cabin.unique()
```

Output:

```
[8]: array(['M', 'C', 'E', 'G', 'D', 'A', 'B', 'F', 'T'], dtype=object)
```

6) Group a Data Frame by a specific column and calculate the mean of another column (e.g., 'Survived'), then create a new Data Frame with the results

```
[9]: prob_df=df.groupby(['Cabin'])['Survived'].mean()
```

```
[10]: prob_df=pd.DataFrame(prob_df)
      prob_df
```

Output:

[10]:

Cabin	Survived
A	0.466667
B	0.744681
C	0.593220
D	0.757576
E	0.750000
F	0.615385
G	0.500000
M	0.299854
T	0.000000

7) Add a new column to a Data Frame, calculated from an existing column, and display the first few rows

```
[11]: prob_df['Died']=1-prob_df['Survived']
```

```
[12]: prob_df.head()
```

8) Create a new column in a Data Frame that represents the ratio of two other columns, and then display the first few rows

```
[13]: prob_df['Probability_ratio']=prob_df['Survived']/prob_df['Died']
      prob_df.head()
```

Output:

[13]:

Cabin	Survived	Died	Probability_ratio
A	0.466667	0.533333	0.875000
B	0.744681	0.255319	2.916667
C	0.593220	0.406780	1.458333
D	0.757576	0.242424	3.125000
E	0.750000	0.250000	3.000000

9) Create a dictionary from a Data Frame column and map this dictionary to another column

```
[14]: probability_encoded=prob_df['Probability_ratio'].to_dict()
```

```
[15]: df['Cabin_encoded']=df['Cabin'].map(probability_encoded)
      df.head()
```

Output:

[15]:

	Survived	Cabin	Cabin_encoded
0	0	M	0.428274
1	1	C	1.458333
2	1	M	0.428274
3	1	C	1.458333
4	0	M	0.428274

10) Read a CSV file into a pandas Data Frame with specific columns, then display the first few rows

```
[17]: import pandas as pd
      df=pd.read_csv('titanic_fe.csv', usecols=['Pclass','Age','Fare','Survived'])
      df.head()
```

Output:

[17]:

	Survived	Pclass	Age	Fare
0	0	3	22.0	7.2500
1	1	1	38.0	71.2833
2	1	3	26.0	7.9250
3	1	1	35.0	53.1000
4	0	3	35.0	8.0500

11) Fill missing values in a Data Frame column with the median value

```
[18]: df['Age'].fillna(df.Age.median(),inplace=True)

[19]: df.isnull().sum()
```

Output:

```
[19]: Survived    0
      Pclass      0
      Age         0
      Fare        0
      dtype: int64
```

12) standardize a Data Frame using sk learn's StandardScaler

```
[21]: #standarisation: We use the Standardscaler from sklearn library
      from sklearn.preprocessing import StandardScaler

[22]: scaler=StandardScaler()
      ### fit vs fit_transform
      df_scaled=scaler.fit_transform(df)
```

13) Import matplotlib.pyplot and ensure that plots are displayed, then show the scaled Data Frame

```
[23]: #pd.Dataframe(df_scaled)

[24]: import matplotlib.pyplot as plt
      %matplotlib inline

[25]: df_scaled
```

Output:

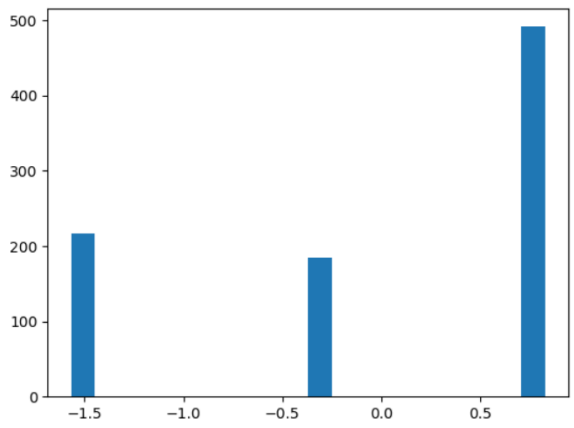
```
[25]: array([[-0.78927234,  0.82737724, -0.56573646, -0.50244517],
             [ 1.2669898 , -1.56610693,  0.66386103,  0.78684529],
             [ 1.2669898 ,  0.82737724, -0.25833709, -0.48885426],
             ...,
             [-0.78927234,  0.82737724, -0.1046374 , -0.17626324],
             [ 1.2669898 , -1.56610693, -0.25833709, -0.04438104],
             [-0.78927234,  0.82737724,  0.20276197, -0.49237783]])
```

14) create a histogram with 20 bins for the second column of a scaled Data Frame

```
[26]: plt.hist(df_scaled[:,1],bins=20)
```

Output:

```
[26]: (array([216.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 184.,
              0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 491.]),
       array([-1.56610693, -1.44643272, -1.32675851, -1.2070843 , -1.08741009,
              -0.96773588, -0.84806167, -0.72838747, -0.60871326, -0.48903905,
              -0.36936484, -0.24969063, -0.13001642, -0.01034222,  0.10933199,
               0.2290062 ,  0.34868041,  0.46835462,  0.58802883,  0.70770304,
               0.82737724]),
       <BarContainer object of 20 artists>)
```



Name of the Laboratory: **Name of the Experiment:**

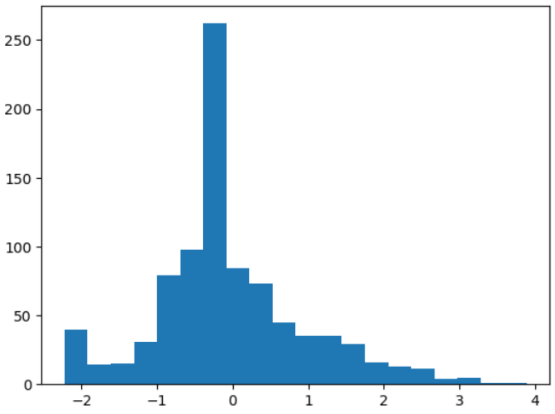
Experiment No: **Date:**

15) Create a histogram with 20 bins for the third column of a scaled Data Frame

```
[27]: plt.hist(df_scaled[:,2],bins=20)
```

Output:

```
[27]: (array([ 40.,  14.,  15.,  31.,  79.,  98., 262.,  84.,  73.,  45.,  35.,
         35.,  29.,  16.,  13.,  11.,   4.,   5.,   1.,   1.]),
 array([-2.22415608, -1.91837055, -1.61258503, -1.3067995 , -1.00101397,
        -0.69522845, -0.38944292, -0.08365739,  0.22212813,  0.52791366,
         0.83369919,  1.13948471,  1.44527024,  1.75105577,  2.05684129,
         2.36262682,  2.66841235,  2.97419787,  3.2799834 ,  3.58576892,
         3.89155445]),
 <BarContainer object of 20 artists>)
```

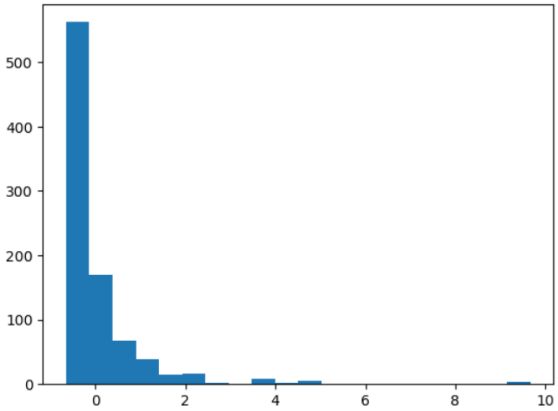


16) Create a histogram with 20 bins for the fourth column of a scaled Data Frame

```
[28]: plt.hist(df_scaled[:,3],bins=20)
```

Output:

```
[28]: (array([562., 170.,  67.,  39.,  15.,  16.,   2.,   0.,   9.,   2.,   6.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   3.]),
 array([-0.64842165, -0.13264224,  0.38313716,  0.89891657,  1.41469598,
         1.93047539,  2.4462548 ,  2.96203421,  3.47781362,  3.99359303,
         4.50937244,  5.02515184,  5.54093125,  6.05671066,  6.57249007,
         7.08826948,  7.60404889,  8.1198283 ,  8.63560771,  9.15138712,
         9.66716653]),
 <BarContainer object of 20 artists>)
```



Name of the Laboratory: **Name of the Experiment:**

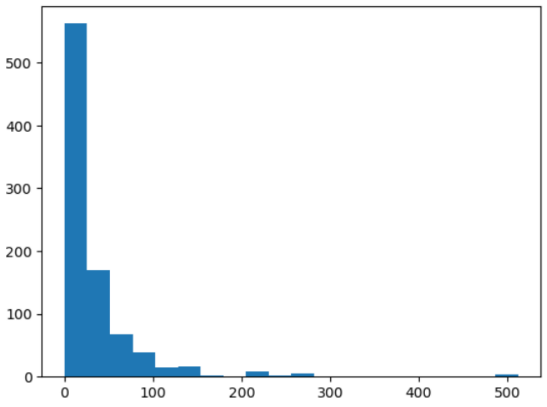
Experiment No: **Date:**

17) Create a histogram with 20 bins for the 'Fare' column in a Data Frame

```
[29]: plt.hist(df['Fare'],bins=20)
```

Output:

```
[29]: (array([562., 170.,  67.,  39.,  15.,  16.,   2.,   0.,   9.,   2.,   6.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   3.]),
       array([  0.    ,  25.61646,  51.23292,  76.84938, 102.46584, 128.0823 ,
       153.69876, 179.31522, 204.93168, 230.54814, 256.1646 , 281.78106,
       307.39752, 333.01398, 358.63044, 384.2469 , 409.86336, 435.47982,
       461.09628, 486.71274, 512.3292 ]),
       <BarContainer object of 20 artists>)
```



18) Apply MinMax scaling to a pandas Data Frame and create a new Data Frame with scaled values, while preserving the original column names

```
[30]: from sklearn.preprocessing import MinMaxScaler
      min_max=MinMaxScaler()
      df_minmax=pd.DataFrame(min_max.fit_transform(df),columns=df.columns)
      df_minmax.head()
```

Output:

[30]:

	Survived	Pclass	Age	Fare
0	0.0	1.0	0.271174	0.014151
1	1.0	0.0	0.472229	0.139136
2	1.0	1.0	0.321438	0.015469
3	1.0	0.0	0.434531	0.103644
4	0.0	1.0	0.434531	0.015713

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

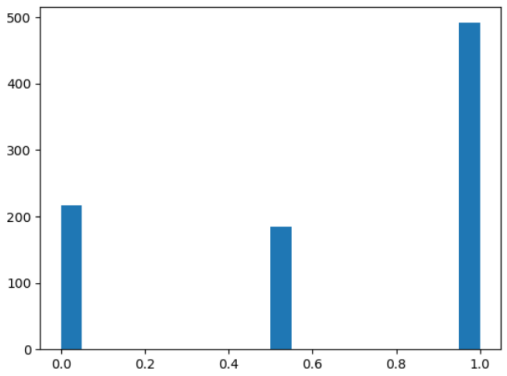
Experiment No: **Date:**

19) create a histogram with 20 bins for the 'Pclass' column in a DataFrame named df_minmax

```
[31]: plt.hist(df_minmax['Pclass'],bins=20)
```

Output:

```
[31]: (array([216.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 184.,
                0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 491.]),
       array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
              0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1.  ]),
       <BarContainer object of 20 artists>)
```

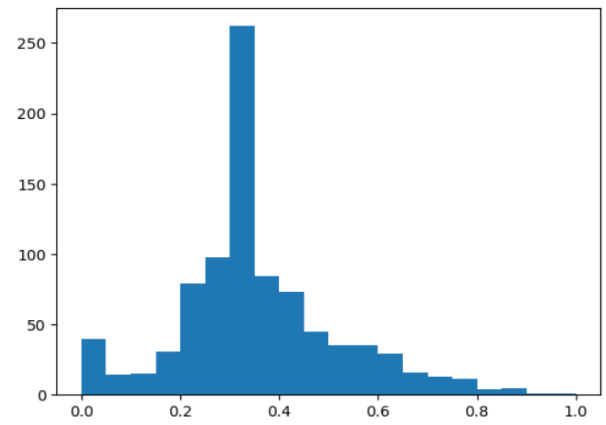


20) Create a histogram with 20 bins for the 'Age' column in a DataFrame named df_minmax

```
[32]: plt.hist(df_minmax['Age'],bins=20)
```

Output:

```
[32]: (array([ 40.,  14.,  15.,  31.,  79.,  98., 262.,  84.,  73.,  45.,  35.,
                35.,  29.,  16.,  13.,  11.,   4.,   5.,   1.,   1.]),
       array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
              0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1.  ]),
       <BarContainer object of 20 artists>)
```

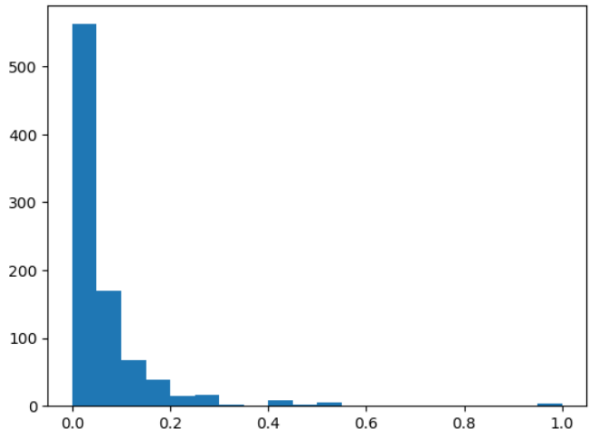


21) Create a histogram with 20 bins for the 'Fare' column in a DataFrame named df_minmax

```
[33]: plt.hist(df_minmax['Fare'],bins=20)
```

Output:

```
[33]: (array([562., 170.,  67.,  39.,  15.,  16.,   2.,   0.,   9.,   2.,   6.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   3.]),
       array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
              0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1.  ]),
       <BarContainer object of 20 artists>)
```



22)Apply RobustScaler to a DataFrame and convert it into a new DataFrame, maintaining the original column names

```
[34]: from sklearn.preprocessing import RobustScaler
      scaler=RobustScaler()
      df_robust_scaler=pd.DataFrame(scaler.fit_transform(df),columns=df.columns)
      df_robust_scaler.head()
```

Output:

[34]:

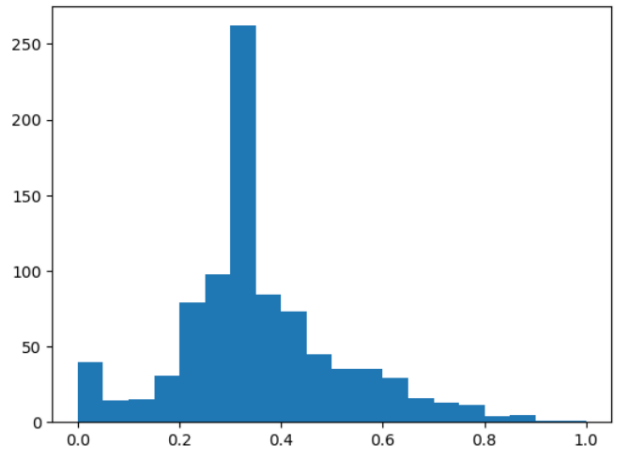
	Survived	Pclass	Age	Fare
0	0.0	0.0	-0.461538	-0.312011
1	1.0	-2.0	0.769231	2.461242
2	1.0	0.0	-0.153846	-0.282777
3	1.0	-2.0	0.538462	1.673732
4	0.0	0.0	0.538462	-0.277363

23) Create histograms with 20 bins for both the 'Age' column

```
[32]: plt.hist(df_minmax['Age'],bins=20)
```

Output:

```
[32]: (array([ 40.,  14.,  15.,  31.,  79.,  98., 262.,  84.,  73.,  45.,  35.,
         35.,  29.,  16.,  13.,  11.,   4.,   5.,   1.,   1.]),
 array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
        0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1.  ]),
 <BarContainer object of 20 artists>)
```

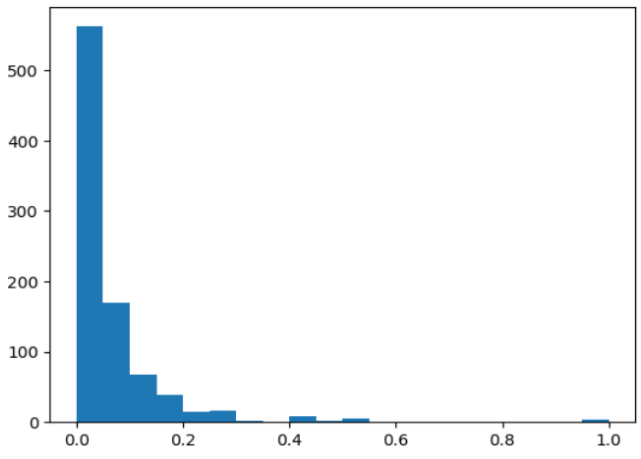


24) Create histograms with 20 bins for both the 'Fare' column

```
[33]: plt.hist(df_minmax['Fare'],bins=20)
```

Output:

```
[33]: (array([562., 170.,  67.,  39.,  15.,  16.,   2.,   0.,   9.,   2.,   6.,
          0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   3.]),
 array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25, 0.3 , 0.35, 0.4 , 0.45, 0.5 ,
        0.55, 0.6 , 0.65, 0.7 , 0.75, 0.8 , 0.85, 0.9 , 0.95, 1.  ]),
 <BarContainer object of 20 artists>)
```



25)Load a CSV file into a pandas DataFrame, selecting specific columns, and then display the first few rows

```
[38]: df=pd.read_csv('titanic_fe.csv',usecols=['Age','Fare','Survived'])
      df.head()
```

Output:

[38]:

	Survived	Age	Fare
0	0	22.0	7.2500
1	1	38.0	71.2833
2	1	26.0	7.9250
3	1	35.0	53.1000
4	0	35.0	8.0500

26) Fill missing values in a 'Age' column with the median value, and then check for remaining NaNs

```
[39]: ### fillnan
      df['Age']=df['Age'].fillna(df['Age'].median())
```

```
[40]: df.isnull().sum()
```

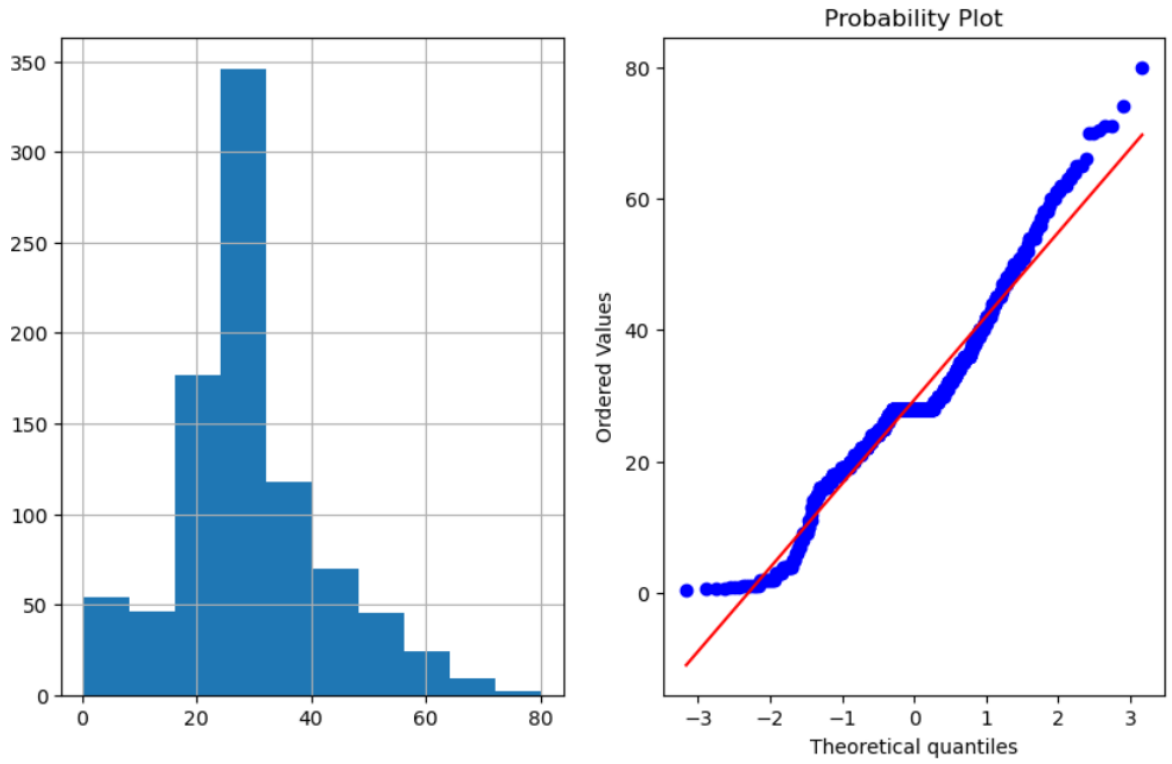
Output:

```
[40]: Survived    0
      Age         0
      Fare        0
      dtype: int64
```

27) Create a histogram and a Q-Q plot to check if a DataFrame column has a Gaussian or normal distribution

```
[41]: import scipy.stats as stat
      import pylab

[42]: #### If you want to check whether feature is guassian or normal distributed
      #### Q-Q plot
      def plot_data(df,feature):
          plt.figure(figsize=(10,6))
          plt.subplot(1,2,1)
          df[feature].hist()
          plt.subplot(1,2,2)
          stat.probplot(df[feature],dist='norm',plot=pylab)
          plt.show()

[43]: plot_data(df,'Age')
```

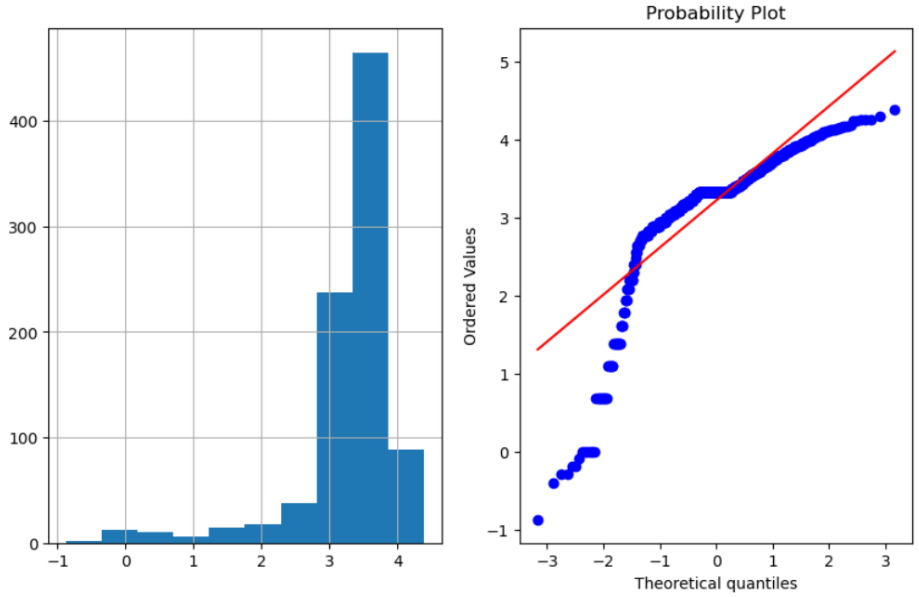
Output:



28) Perform a log transformation on a DataFrame column and visualize it using a histogram and Q-Q plot

```
[44]: import numpy as np
      df['Age_log']=np.log(df['Age'])
      plot_data(df,'Age_log')
```

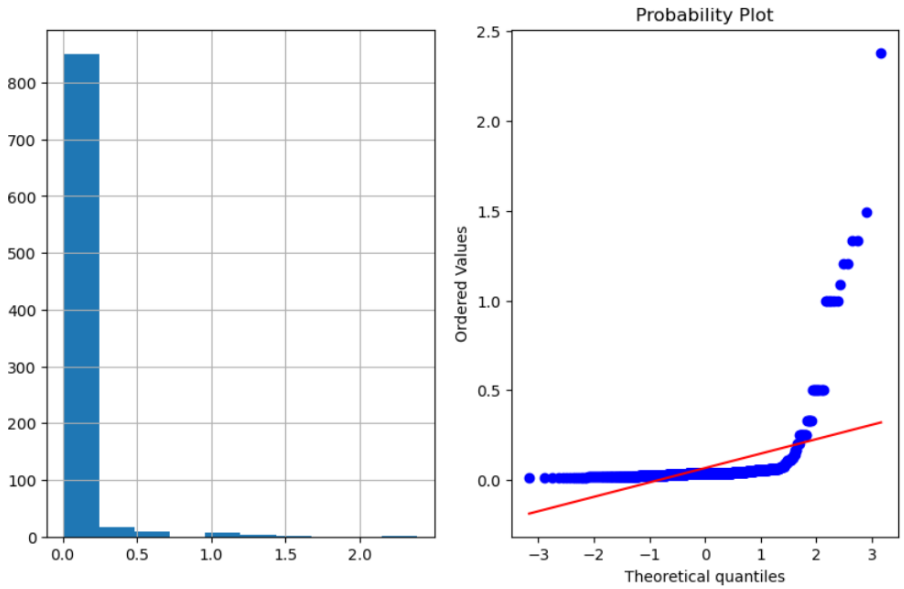
Output:



29)Apply a reciprocal transformation to a DataFrame column and visualize it using a histogram and Q-Q plot

```
[45]: df['Age_reciprocal']=1/df.Age
      plot_data(df,'Age_reciprocal')
```

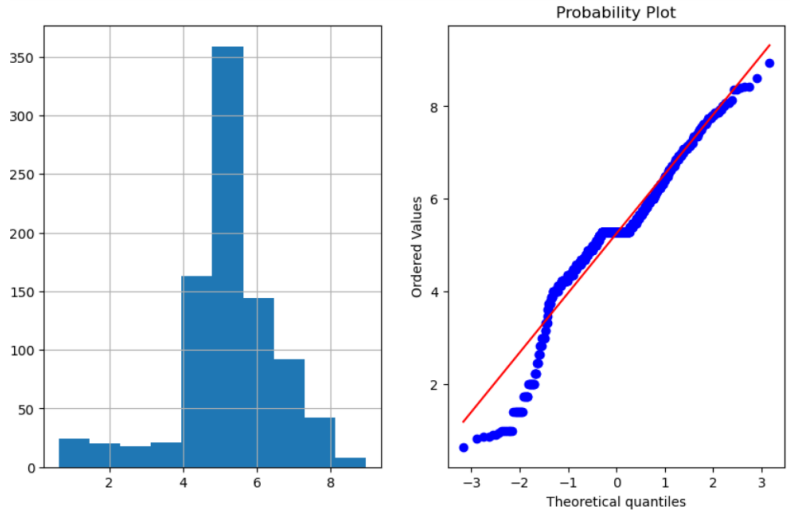
Output:



30)Apply a square root transformation to a DataFrame column and visualize it using a histogram and Q-Q plot

```
[46]: ##### Square Root Transformation
      df['Age_sqaure']=df.Age**(1/2)
      plot_data(df,'Age_sqaure')
```

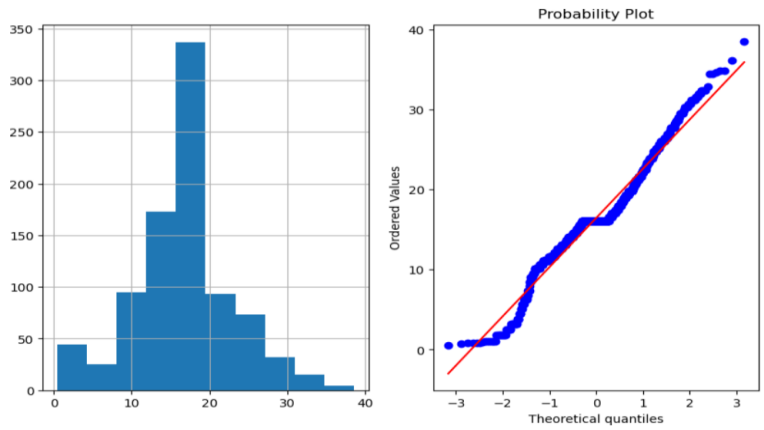
Output:



31)Apply an exponential transformation to a DataFrame column and visualize it using a histogram and Q-Q plot

```
[47]: #### Exponential Transdormation
      df['Age_exponential']=df.Age**(1/1.2)
      plot_data(df,'Age_exponential')
```

Output:



32) Apply a Box-Cox transformation to a DataFrame column and get the transformation parameters

```
[48]: df['Age_Boxcox'],parameters=stat.boxcox(df['Age'])

[49]: print(parameters)
```

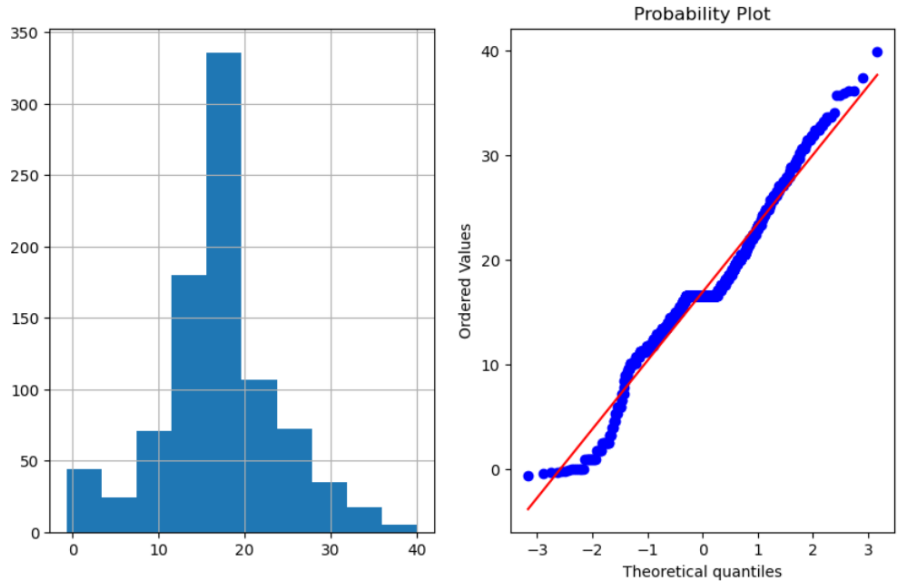
Output:

```
0.7964531473656952
```

33) visualize a DataFrame column that has undergone a Box-Cox transformation using a histogram and Q-Q plot

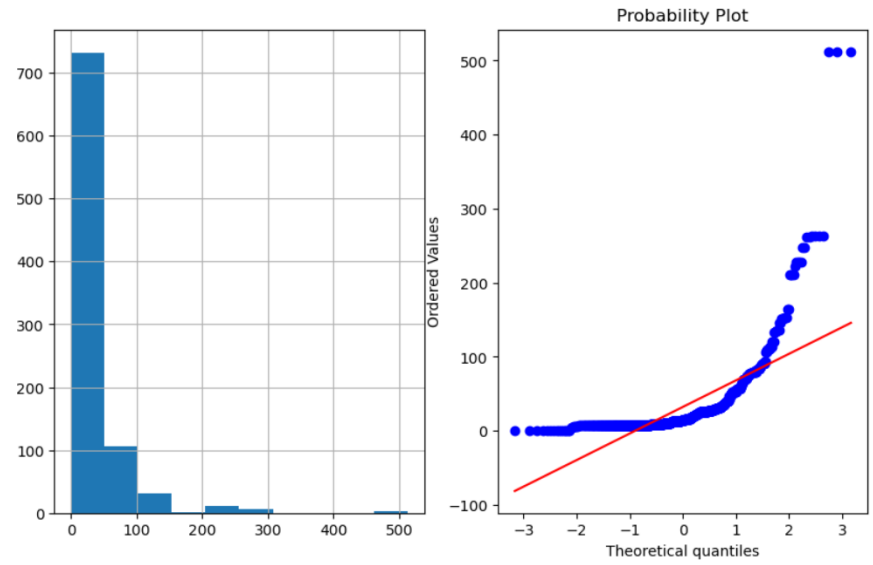
```
[50]: plot_data(df,'Age_Boxcox')
```

Output:



34) visualize the distribution of the 'Fare' column in a DataFrame using a histogram and Q-Q plot

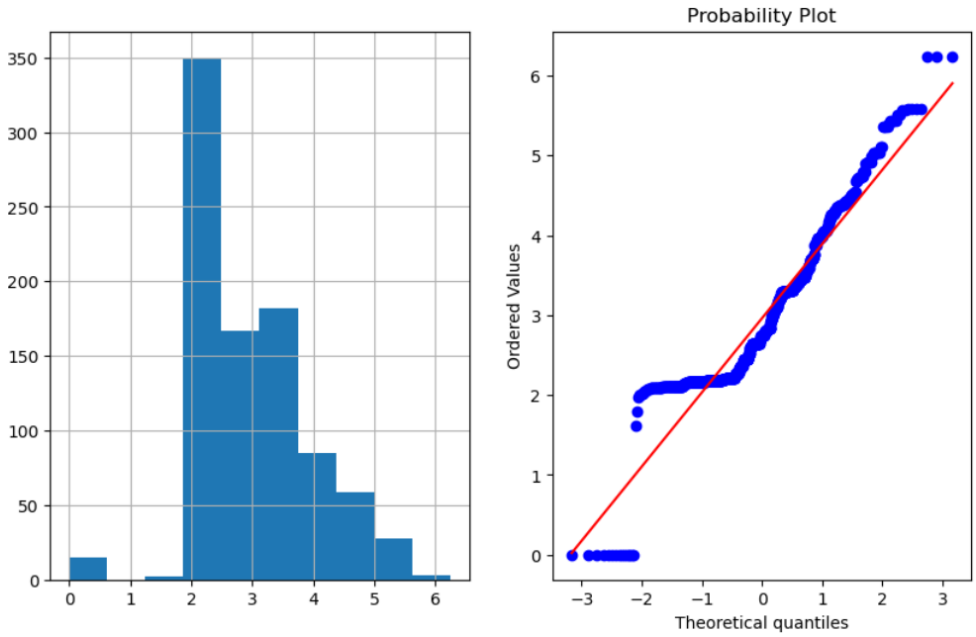
```
[51]: plot_data(df,'Fare')
```



35) Apply a log transformation to the 'Fare' column in a DataFrame,and visualize it

```
: #### Fare
df['Fare_log']=np.log1p(df['Fare'])
plot_data(df,'Fare_log')
```

Output:



VNR VJIET

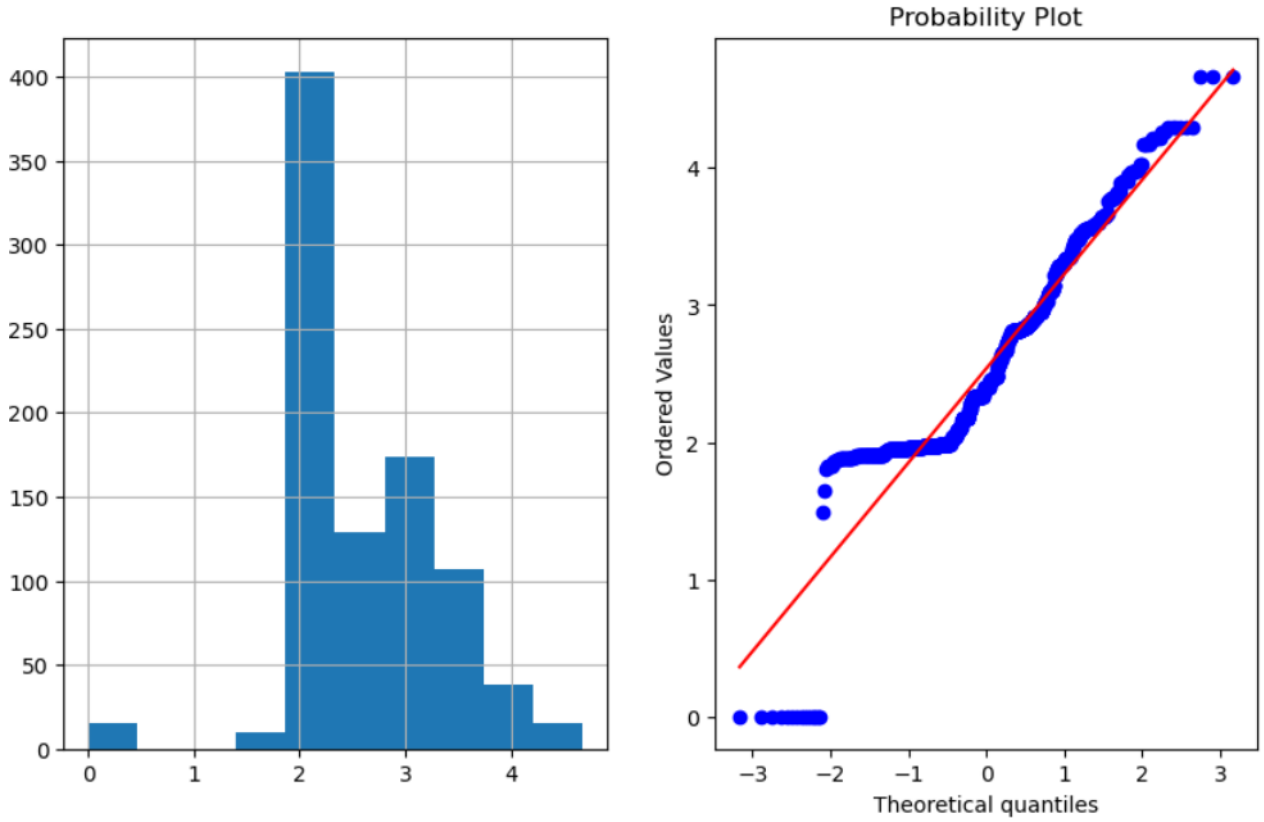
Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

36) Apply a Box-Cox transformation to the 'Fare' column in a DataFrame, and visualize it

```
[53]: df['Fare_Boxcox'],parameters=stat.boxcox(df['Fare']+1)
      plot_data(df,'Fare_Boxcox')
```

Output:



VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

WEEK-9

Decision Tree on diabetic data set using sklearn

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
```

In [3]:
```
df = pd.read_csv('Decision-Tree-Classification-Data.csv')
df
```

Out[3]:

	age	bp	diabetes
0	65	65	1
1	45	82	0
2	35	73	1
3	45	90	0
4	50	68	1
...	...	...	...
982	45	87	0
983	40	83	0
984	40	83	0
985	40	60	1
986	45	82	0

987 rows × 3 columns

In [4]:
```
df.shape
```

Out[4]: (987, 3)

In [6]:
```
df = df.dropna()
df.shape
```

Out[6]: (987, 3)

In [10]:
```
y.value_counts()
```

Out[10]:
```
diabetes
1    495
0    492
Name: count, dtype: int64
```

In [11]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [12]:
```
from sklearn import tree

model = tree.DecisionTreeClassifier(max_depth=5)
```

In [13]:
```
model
```

Out[13]:
```
DecisionTreeClassifier(max_depth=5)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [14]:
```
model.fit(X_train, y_train)
```

Out[14]:
```
DecisionTreeClassifier(max_depth=5)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [15]:
```
y_predict = model.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_predict)
```

Out[15]:
```
0.9149797570850202
```

In [16]:
```
from sklearn.metrics import confusion_matrix

pd.DataFrame(
    confusion_matrix(y_test, y_predict),
    columns=['Predicted Not diabetes ', 'Predicted diabetes'],
    index=['True Not diabetes', 'True diabetes']
)
```

Out[16]:

	Predicted Not diabetes	Predicted diabetes
True Not diabetes	113	5
True diabetes	16	113

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

Implementation of ID3

In [1]:
```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
from numpy import log2 as log
```

In [3]:
```
outlook = 'overcast,overcast,overcast,overcast,rainy,rainy,rainy,rainy,rainy,sunny,sunny,sunny,sunny,sunny'.split(',')
temp = 'hot,cool,mild,hot,mild,cool,cool,mild,mild,hot,hot,mild,cool,mild'.split(',')
humidity = 'high,normal,high,normal,high,normal,normal,normal,high,high,high,high,normal,normal'.split(',')
windy = 'FALSE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,FALSE,TRUE'.split(',')
play = 'yes,yes,yes,yes,yes,yes,no,yes,no,no,no,no,yes,yes'.split(',')
```

In [4]:
```
dataset ={'outlook':outlook,'temp':temp,'humidity':humidity,'windy':windy,'play':play}
df = pd.DataFrame(dataset,columns=['outlook','temp','humidity','windy','play'])
```

In [5]:
```
df
```

Out[5]:

	outlook	temp	humidity	windy	play
0	overcast	hot	high	FALSE	yes
1	overcast	cool	normal	TRUE	yes
2	overcast	mild	high	TRUE	yes
3	overcast	hot	normal	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	rainy	mild	normal	FALSE	yes
8	rainy	mild	high	TRUE	no
9	sunny	hot	high	FALSE	no
10	sunny	hot	high	TRUE	no
11	sunny	mild	high	FALSE	no
12	sunny	cool	normal	FALSE	yes
13	sunny	mild	normal	TRUE	yes

Calculate entropy of the whole dataset

In [6]:
```
entropy_node = 0  #Initialize Entropy
values = df.play.unique()  #Unique objects - 'Yes', 'No'
for value in values:
    fraction = df.play.value_counts()[value]/len(df.play)
    entropy_node += -fraction*np.log2(fraction)
```

In [7]:
```
entropy_node
```

Out[7]:
```
0.9402859586706311
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

In [8]:
```
def ent(df,attribute):
    target_variables = df.play.unique()  #This gives all 'Yes' and 'No'
    variables = df[attribute].unique()    #This gives different features in that attribute (like 'Sweet')


    entropy_attribute = 0
    for variable in variables:
        entropy_each_feature = 0
        for target_variable in target_variables:
            num = len(df[attribute][df[attribute]==variable][df.play ==target_variable]) #numerator
            den = len(df[attribute][df[attribute]==variable])  #denominator
            fraction = num/(den+eps)  #pi
            entropy_each_feature += -fraction*log(fraction+eps) #This calculates entropy for one feature like 'Sweet'
        fraction2 = den/len(df)
        entropy_attribute += -fraction2*entropy_each_feature   #Sums up all the entropy ETaste

    return(abs(entropy_attribute))
```

In [9]:
```
a_entropy = {k:ent(df,k) for k in df.keys()[:-1]}
a_entropy
```

Out[9]:
```
{'outlook': 0.6935361388961914,
 'temp': 0.9110633930116756,
 'humidity': 0.7884504573082889,
 'windy': 0.892158928262361}
```

In [10]:
```
def ig(e_dataset,e_attr):
    return(e_dataset-e_attr)
```

In [11]:
```
#entropy_node = entropy of dataset
#a_entropy[k] = entropy of k(th) attr
IG = {k:ig(entropy_node,a_entropy[k]) for k in a_entropy}
```

In [12]:
```
IG
```

Out[12]:
```
{'outlook': 0.24674981977443977,
 'temp': 0.029222565658955535,
 'humidity': 0.15183550136234225,
 'windy': 0.048127030408270155}
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

In [13]:

```
#import sys
#sys.setrecursionlimit(10**6)
def find_entropy(df):
    Class = df.keys()[-1]   #To make the code generic, changing target variable class name
    entropy = 0
    values = df[Class].unique()
    for value in values:
        fraction = df[Class].value_counts()[value]/len(df[Class])
        entropy += -fraction*np.log2(fraction)
    return entropy

def find_entropy_attribute(df,attribute):
    Class = df.keys()[-1]   #To make the code generic, changing target variable class name
    target_variables = df[Class].unique()  #This gives all 'Yes' and 'No'
    variables = df[attribute].unique()    #This gives different features in that attribute (like 'Hot','Cold' in Temperature)
    entropy2 = 0
    for variable in variables:
        entropy = 0
        for target_variable in target_variables:
            num = len(df[attribute][df[attribute]==variable][df[Class] ==target_variable])
            den = len(df[attribute][df[attribute]==variable])
            fraction = num/(den+eps)
            entropy += -fraction*log(fraction+eps)
        fraction2 = den/len(df)
        entropy2 += -fraction2*entropy
    return abs(entropy2)
```

```
def find_winner(df):
    Entropy_att = []
    IG = []
    for key in df.keys()[:-1]:
#         Entropy_att.append(find_entropy_attribute(df,key))
        IG.append(find_entropy(df)-find_entropy_attribute(df,key))
    return df.keys()[:-1][np.argmax(IG)]

def get_subtable(df, node,value):
    return df[df[node] == value].reset_index(drop=True)
```

```
def buildTree(df,tree=None):
    Class = df.keys()[-1]   #To make the code generic, changing target variable class name
    #Here we build our decision tree

    #Get attribute with maximum information gain
    node = find_winner(df)

    #Get distinct value of that attribute e.g Salary is node and Low,Med and High are values
    attValue = np.unique(df[node])

    #Create an empty dictionary to create tree
    if tree is None:
        tree={}
        tree[node] = {}

   #We make loop to construct a tree by calling this function recursively.
    #In this we check if the subset is pure and stops if it is pure.

    for value in attValue:

        subtable = get_subtable(df,node,value)
        clValue,counts = np.unique(subtable['play'],return_counts=True)

        if len(counts)==1:#Checking purity of subset
            tree[node][value] = clValue[0]
        else:
            tree[node][value] = buildTree(subtable) #Calling the function recursively

    return tree
```

In [14]:
```
t = buildTree(df)
```

In [15]:
```
import pprint
pprint.pprint(t)
```

```
{'outlook': {'overcast': 'yes',
             'rainy': {'windy': {'FALSE': 'yes', 'TRUE': 'no'}},
             'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}}}}
```

Explaination

In [16]:
```
df
```

Out[16]:

	outlook	temp	humidity	windy	play
0	overcast	hot	high	FALSE	yes
1	overcast	cool	normal	TRUE	yes
2	overcast	mild	high	TRUE	yes
3	overcast	hot	normal	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	rainy	mild	normal	FALSE	yes
8	rainy	mild	high	TRUE	no
9	sunny	hot	high	FALSE	no
10	sunny	hot	high	TRUE	no
11	sunny	mild	high	FALSE	no
12	sunny	cool	normal	FALSE	yes
13	sunny	mild	normal	TRUE	yes

In [17]:
```
df['outlook'] = df['outlook'].map({'overcast': 0, 'rainy': 1, 'sunny':2})
```

In [18]:
```
df['temp'] = df['temp'].map({'hot': 0, 'cool': 1, 'mild':2})
```

In [19]:
```
df['humidity'] = df['humidity'].map({'high': 0, 'normal': 1})
```

In [20]:
```
df['windy'] = df['windy'].map({'FALSE': 0, 'TRUE': 1})
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

In [21]:
```
df
```

Out[21]:

	outlook	temp	humidity	windy	play
0	0	0	0	0	yes
1	0	1	1	1	yes
2	0	2	0	1	yes
3	0	0	1	0	yes
4	1	2	0	0	yes
5	1	1	1	0	yes
6	1	1	1	1	no
7	1	2	1	0	yes
8	1	2	0	1	no
9	2	0	0	0	no
10	2	0	0	1	no
11	2	2	0	0	no
12	2	1	1	0	yes
13	2	2	1	1	yes

In [22]:
```
X = df.drop('play', axis=1)
y = df['play']
```

In [23]:
```
X
```

Out[23]:

	outlook	temp	humidity	windy
0	0	0	0	0
1	0	1	1	1
2	0	2	0	1
3	0	0	1	0
4	1	2	0	0
5	1	1	1	0
6	1	1	1	1
7	1	2	1	0
8	1	2	0	1
9	2	0	0	0
10	2	0	0	1
11	2	2	0	0
12	2	1	1	0
13	2	2	1	1

In [24]:
```
y
```

Out[24]:
```
0     yes
1     yes
2     yes
3     yes
4     yes
5     yes
6      no
7     yes
8      no
9      no
10     no
11     no
12    yes
13    yes
Name: play, dtype: object
```

In [25]:
```
y.value_counts()
```

Out[25]:
```
play
yes    9
no     5
Name: count, dtype: int64
```

In [26]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [27]:
```
from sklearn import tree
model = tree.DecisionTreeClassifier() #criterion='entropy', max_depth=2
```

In [28]:
```
model
```

Out[28]:
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [29]:
```
model.fit(X_train, y_train)
```

Out[29]:
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [30]:
```
y_predict = model.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_predict)
```

Out[30]:
```
0.5
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

K Nearest Neighbours with Python

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

Get the Data

Set index_col=0 to use the first column as the index.

In [7]:
```
df = pd.read_csv("Classified Data.csv",index_col=0)
```

In [8]:
```
df.head()
```

Out[8]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

Standardize the Variables

Because the KNN classifier predicts the class of a given test observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have a much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale.

In [9]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [10]:
```
scaler.fit(df.drop('TARGET CLASS',axis=1))
```

Out[10]:
```
StandardScaler()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [11]:
```
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
```

In [12]:
```
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

Out[12]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ
0	-0.123542	0.185907	-0.913431	0.319629	-1.033637	-2.308375	-0.798951	-1.482368	-0.949719	-0.643314
1	-1.084836	-0.430348	-1.025313	0.625388	-0.444847	-1.152706	-1.129797	-0.202240	-1.828051	0.636759
2	-0.788702	0.339318	0.301511	0.755873	2.031693	-0.870156	2.599818	0.285707	-0.682494	-0.377850
3	0.982841	1.060193	-0.621399	0.625299	0.452820	-0.267220	1.750208	1.066491	1.241325	-1.026987
4	1.139275	-0.640392	-0.709819	-0.057175	0.822886	-0.936773	0.596782	-1.472352	1.040772	0.276510

Pair Plot

In [13]:
```
#import seaborn as sns

#sns.pairplot(df,hue='TARGET CLASS')
```

Train Test Split

In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'],
```

Using KNN

In [15]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
```

Out[15]:
```
KNeighborsClassifier(n_neighbors=1)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]:
```
pred = knn.predict(X_test)
```

Predictions and Evaluations

In [17]:
```
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import cross_val_score
print(confusion_matrix(y_test,pred))
```

```
[[126  19]
 [ 12 143]]
```

In [18]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.87      0.89       145
           1       0.88      0.92      0.90       155

    accuracy                           0.90       300
   macro avg       0.90      0.90      0.90       300
weighted avg       0.90      0.90      0.90       300
```

VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

PCA

In [1]:
```
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt
```

Generate data

In [2]:
```
X = np.random.normal(0, 1, (100, 4))
X[:,2] = 3 * X[:,0] - 2 * X[:,1] + np.random.normal(0, 0.1, 100)
X[:,3] = 1.5 * X[:,0] - 0.5 * X[:,1] + np.random.normal(0, 0.1, 100)
```

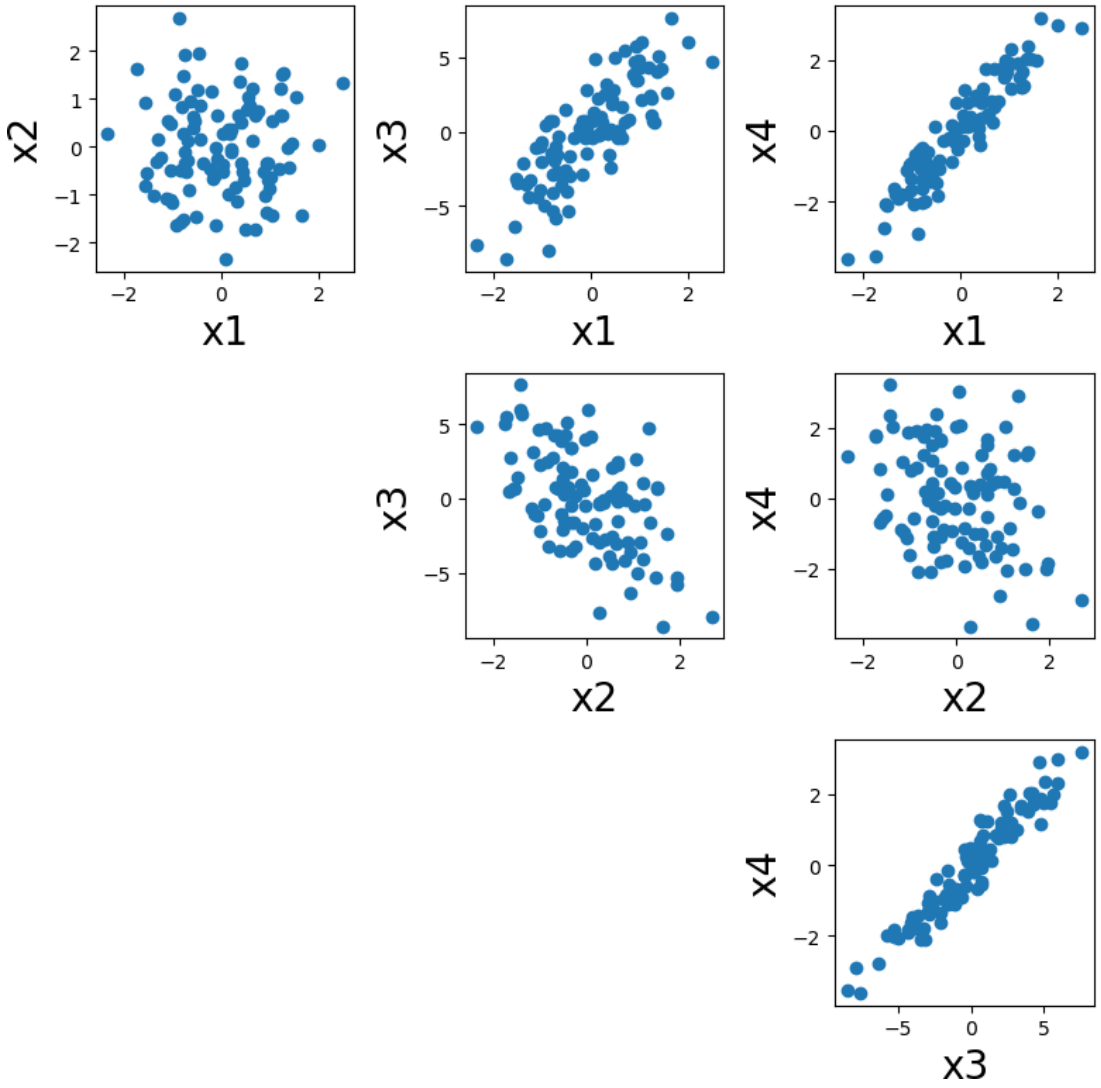
In [3]:
```
X
```

Out[3]:
```
array([[-1.43429550e+00,  4.44852404e-01, -5.13211511e+00,
        -2.23715766e+00],
       [-7.64640980e-01,  7.66403718e-01, -3.69882909e+00,
        -1.50289588e+00],
       [-8.16104982e-01,  1.09808558e+00, -4.81459188e+00,
        -1.87903927e+00],
       [-1.28494741e+00,  9.94448459e-01, -5.74844722e+00,
        -2.48789519e+00],
       [-1.20350017e+00,  2.59296176e+00, -8.72023703e+00,
        -3.31890170e+00],
       [ 2.56750172e-01,  7.57568145e-01, -8.18535629e-01,
         4.62851033e-02],
       [-1.61911229e+00,  6.96354064e-02, -5.13201125e+00,
        -2.33864815e+00],
       [ 6.88734836e-02,  1.63750595e+00, -3.15312519e+00,
        -8.11365084e-01],
       [-1.28646676e-01,  2.57174857e-01, -8.46812483e-01,
        -2.47394921e-01],
       [ 8.83089170e-02,  4.19162093e-01, -5.81621710e-01,
```

each feature will have zero mean

In [4]:
```
X = X - np.mean(X, axis=0)
```

In [5]:
```
plt.figure(figsize=(10,10))
for i in range(4):
    for j in range(4):
        if j > i:
            plt.subplot(4,4,i*4+j+1)
            plt.scatter(X[:,i], X[:,j])
            plt.xlabel(f'x{i+1}', fontsize=20)
            plt.ylabel(f'x{j+1}', fontsize=20)
plt.tight_layout()
```



Observations:

x1 and x2 do not seem correlated

x1 seems very correlated with both x3 and x4

x2 seems somewhat correlated with both x3 and x4

x3 and x4 seem very correlated

Applying PCA

In [6]:
```
#initialize
pca = PCA(n_components=4)

#fit
pca.fit(X)
```

Out[6]:
```
PCA(n_components=4)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [7]:
```
#get principal components
principal_comps_builtin = pca.components_.T
```

In [8]:
```
#print each principal component
for i,component in enumerate(pca.components_):
    print(f'principal component {i}')
    print(component)
    print()
```

```
principal component 0
[-0.20653441  0.13619518 -0.89003662 -0.38292196]

principal component 1
[ 0.4859455   0.79690059 -0.13405399  0.33292034]

principal component 2
[-0.19035203 -0.30415914 -0.37089045  0.85655916]

principal component 3
[ 0.8276271  -0.50387255 -0.22870204 -0.09402777]
```

PCA by hand

Principal components are the eigenvectors of the covariance matrix

In [9]:
```
#compute covariance matrix
cov_matrix = sum([X[i].reshape(-1,1) @ X[i].reshape(1,-1) for i in range(100)]) / 100
```

In [10]:
```
cov_matrix
```

Out[10]:
```
array([[ 0.8693231 ,  0.02445809,  2.55279346,  1.30537329],
       [ 0.02445809,  0.96398177, -1.84675505, -0.45487929],
       [ 2.55279346, -1.84675505, 11.32861099,  4.81319944],
       [ 1.30537329, -0.45487929,  4.81319944,  2.22105667]])
```

In [11]:
```
#eigenvalues and eigenvectors of covariance matrix
eigvecs = np.linalg.eig(cov_matrix)
```

In [12]:
```
#sort order by magnitude of eigenvalue
ordering = np.argsort(eigvecs[0])[::-1]
```

In [13]:
```
#get eigenvectors
principal_comps_byhand = eigvecs[1][:,ordering]
```

In [14]:
```
#our by-hand eigenvectors
principal_comps_byhand
```

Out[14]:
```
array([[ 0.20653441,  0.4859455 , -0.19035203,  0.8276271 ],
       [-0.13619518,  0.79690059, -0.30415914, -0.50387255],
       [ 0.89003662, -0.13405399, -0.37089045, -0.22870204],
       [ 0.38292196,  0.33292034,  0.85655916, -0.09402777]])
```

In [15]:
```
#results from built-in call
principal_comps_builtin
```

Out[15]:
```
array([[-0.20653441,  0.4859455 , -0.19035203,  0.8276271 ],
       [ 0.13619518,  0.79690059, -0.30415914, -0.50387255],
       [-0.89003662, -0.13405399, -0.37089045, -0.22870204],
       [-0.38292196,  0.33292034,  0.85655916, -0.09402777]])
```

In [18]:
```
df1
```

Out[18]:

	X1	X2	X3
0	10	20	10
1	2	5	2
2	8	17	7
3	9	20	10
4	12	22	11

In [19]:
```
pca = decomposition.PCA(n_components=3)
```

In [20]:
```
pca
```

Out[20]:
```
PCA(n_components=3)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [21]:
```
#find eigen values and eigen vectors of covariance matrix of df
pca.fit(df1)
```

Out[21]:
```
PCA(n_components=3)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [22]:
```
#convert all the data points from standard basis to eigen vector basis
df1_pca = pca.transform(df1)
```

In [23]:
```
df1_pca
```

Out[23]:
```
array([[-4.17288843e+00, -1.98923940e-04, -2.58847587e-01],
       [ 1.46146195e+01, -1.71937350e-01, -2.51663442e-01],
       [ 3.51842744e-01,  1.00363798e-01,  9.72694089e-01],
       [-3.73883152e+00,  8.99599868e-01, -3.03082462e-01],
       [-7.05474229e+00, -8.27827393e-01, -1.59100599e-01]])
```

In [26]:
```
#understand how much variance captured by each principal component
print(pca.explained_variance_)
print(pca.explained_variance_ratio_)
print(pca.explained_variance_ratio_.cumsum())
```

```
[73.71803604  0.38355337  0.29841058]
[0.99083382 0.00515529 0.00401089]
[0.99083382 0.99598911 1.        ]
```

In [27]:
```
#show the principal components
#show eighen vectors of covariance matrix of df
pca.components_[0]
pca.components_[1]
pca.components_[2]
```

Out[27]:
```
array([ 0.04423488,  0.45128105, -0.89128486])
```

In [29]:
```
#specify number of required dimensions as n_components
pca = decomposition.PCA(n_components=2)
pca.fit(df1)
pca.explained_variance_
pca.components_[0]
pca.components_[1]
df1_pca = pca.transform(df1)
```

In [30]:
```
df1_pca
```

Out[30]:
```
array([[-4.17288843e+00, -1.98923940e-04],
       [ 1.46146195e+01, -1.71937350e-01],
       [ 3.51842744e-01,  1.00363798e-01],
       [-3.73883152e+00,  8.99599868e-01],
       [-7.05474229e+00, -8.27827393e-01]])
```

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

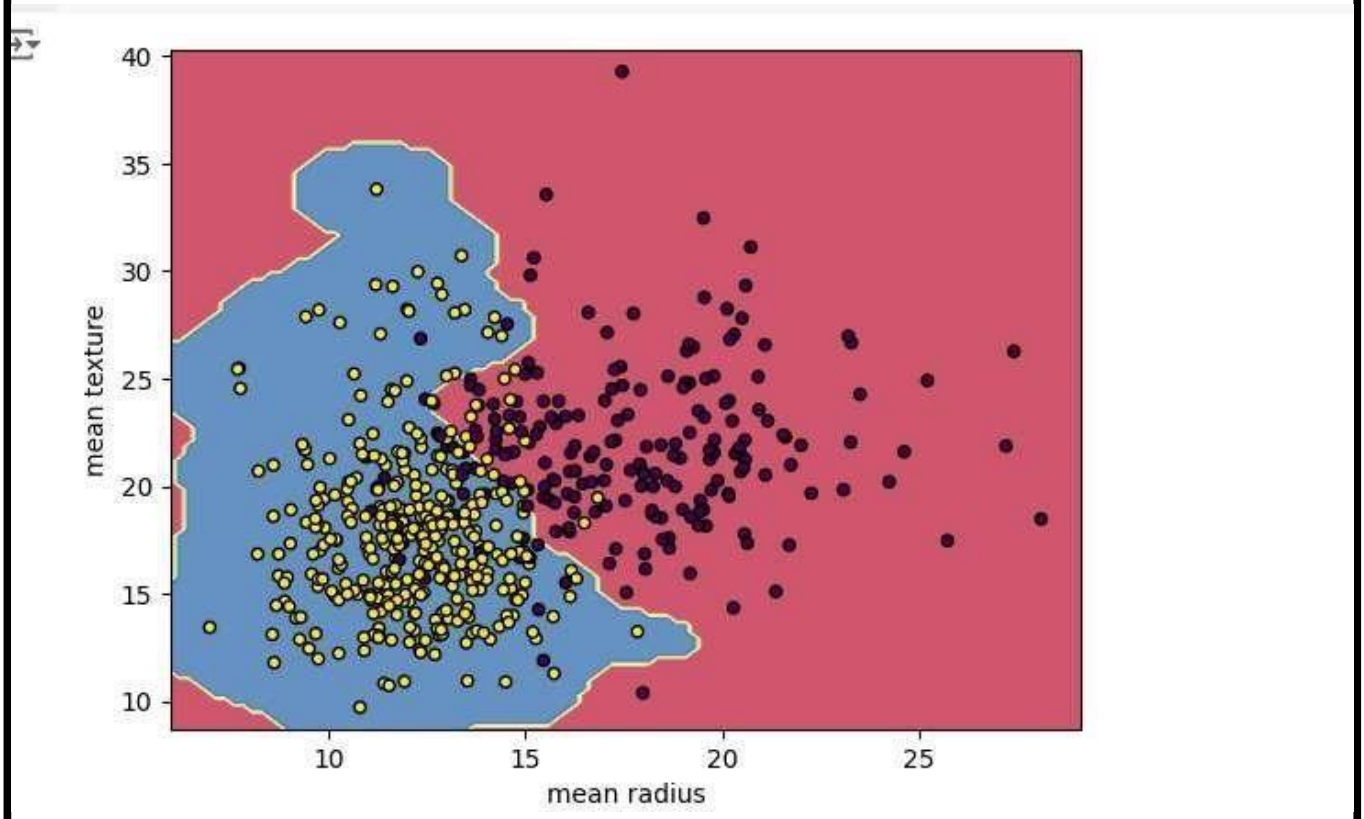
WEEK – 10

END-TO-END SVM ALGORITHM IMPLEMENTATION

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

#Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)

# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    cmap=plt.cm.Spectral,
    alpha=0.8,
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1],
  )

# Scatter plot
plt.scatter(X[:, 0], X[:, 1],
      c=y,
      s=20, edgecolors="k")
plt.show()
```

Experiment No: **Date:**

OUTPUT:



VNR VJIET

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**

END-TO-END MULTI LAYER PERCEPTRON ALGORITHM IMPLEMENTATION

```python
import numpy as np

class NeuralNetwork:
  def __init__(self, input_size, hidden_size, output_size):
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.output_size = output_size

    # Initialize weights
    self.weights_input_hidden = np.random.randn(self.input_size, self.hidden_size)
    self.weights_hidden_output = np.random.randn(self.hidden_size, self.output_size)

    # Initialize the biases
    self.bias_hidden = np.zeros((1, self.hidden_size))
    self.bias_output = np.zeros((1, self.output_size))

  def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))

  def sigmoid_derivative(self, x):
    return x * (1 - x)

  def feedforward(self, X):
    # Input to hidden
    self.hidden_activation = np.dot(X, self.weights_input_hidden) + self.bias_hidden
    self.hidden_output = self.sigmoid(self.hidden_activation)

    # Hidden to output
    self.output_activation = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
    self.predicted_output = self.sigmoid(self.output_activation)

    return self.predicted_output
```

Name of the Laboratory:

Name of the Experiment:

Experiment No: **Date:**

```
  def backward(self, X, y, learning_rate):
    # Compute the output layer error
    output_error = y - self.predicted_output
    output_delta = output_error * self.sigmoid_derivative(self.predicted_output)

    # Compute the hidden layer error
    hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
    hidden_delta = hidden_error * self.sigmoid_derivative(self.hidden_output)

    # Update weights and biases
    self.weights_hidden_output += np.dot(self.hidden_output.T, output_delta) * learning_rate
    self.bias_output += np.sum(output_delta, axis=0, keepdims=True) * learning_rate
    self.weights_input_hidden += np.dot(X.T, hidden_delta) * learning_rate
    self.bias_hidden += np.sum(hidden_delta, axis=0, keepdims=True) * learning_rate

  def train(self, X, y, epochs, learning_rate):
    for epoch in range(epochs):
      output = self.feedforward(X)
      self.backward(X, y, learning_rate)
      if epoch % 4000 == 0:
        loss = np.mean(np.square(y - output))
        print(f"Epoch {epoch}, Loss:{loss}")

X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])

nn = NeuralNetwork(input_size=2, hidden_size=4, output_size=1)
nn.train(X, y, epochs=10000, learning_rate=0.1)

# Test the trained model
output = nn.feedforward(X)
print("Predictions after training:")
print(output)
```

OUTPUT:

```
Epoch 0, Loss:0.2714727418134531
Epoch 4000, Loss:0.018749541970485697
Epoch 8000, Loss:0.0033743174774034665
Predictions after training:
[[0.04338492]
 [0.96045466]
 [0.94674104]
 [0.05355973]]
```

VNR VJIET

Name of the Laboratory:

Name of the Experiment:

Experiment No: **Date:**

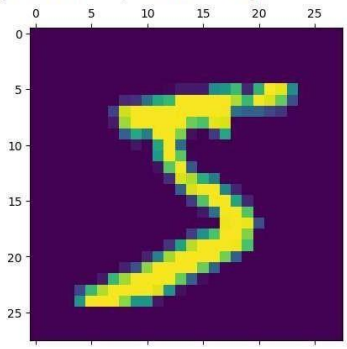
WEEK-11

END-TO-END PERCEPTRON ALGORITHM IMPLEMENTATION

```
[ ] import numpy as np
    import tensorflow as tf
    from tensorflow import keras
    import matplotlib.pyplot as plt
    %matplotlib inline
    (x_train, y_train),(x_test, y_test) = keras.datasets.mnist.load_data()
    len(x_train)
    len(x_test)
    x_train[0].shape
    plt.matshow(x_train[0])
    # Normalizing the dataset
    x_train = x_train/255
    x_test = x_test/255

    # Flatting the dataset in order
    # to compute for model building
    x_train_flatten = x_train.reshape(len(x_train), 28*28)
    x_test_flatten = x_test.reshape(len(x_test), 28*28)
    model = keras.Sequential([
      keras.layers.Dense(10, input_shape=(784,),
              activation='sigmoid')
    ])
    model.compile(
      optimizer='adam',
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'])

    model.fit(x_train_flatten, y_train, epochs=5)
    model.evaluate(x_test_flatten, y_test)
```

OUTPUT:

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4664 - accuracy: 0.8782
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3042 - accuracy: 0.9149
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2832 - accuracy: 0.9200
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2727 - accuracy: 0.9239
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2670 - accuracy: 0.9253
313/313 [==============================] - 1s 1ms/step - loss: 0.2671 - accuracy: 0.9264
[0.2671075165271759, 0.9264000058174133]
```



VNR VJIET

Name of the Laboratory:

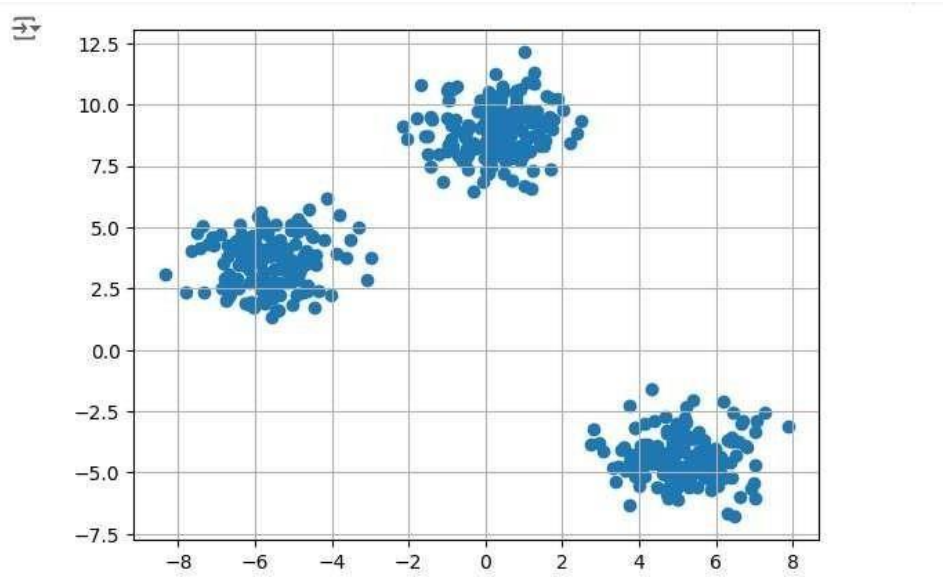
Name of the Experiment:

Experiment No: **Date:**

END-TO-END KMEANS ALGORITHM IMPLEMENTATION

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.datasets import make_blobs
     X,y = make_blobs(n_samples = 500,n_features = 2,centers = 3,random_state = 23)

     fig = plt.figure(0)
     plt.grid(True)
     plt.scatter(X[:,0],X[:,1])
     plt.show()
```

OUTPUT:



Random Forest

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import warnings
warnings.filterwarnings('ignore')
# Load the Titanic dataset
url = "https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv"
titanic_data = pd.read_csv(url)

# Drop rows with missing target values
titanic_data = titanic_data.dropna(subset=['Survived'])

# Select relevant features and target variable
X = titanic_data[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']]
y = titanic_data['Survived']

# Convert categorical variable 'Sex' to numerical using .loc
X.loc[:, 'Sex'] = X['Sex'].map({'female': 0, 'male': 1})

# Handle missing values in the 'Age' column using .loc
X.loc[:, 'Age'].fillna(X['Age'].median(), inplace=True)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)
```

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

# Print the results
print(f"Accuracy: {accuracy:.2f}")
print("\nClassification Report:\n", classification_rep)
```

OUTPUT:

```
Accuracy: 0.80

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.85      0.83       105
           1       0.77      0.73      0.75        74

    accuracy                           0.80       179
   macro avg       0.79      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179
```

END-TO-END HEIRARCHIAL ALGORITHM IMPLEMENTATION

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage
# Generate sample data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)
# Perform agglomerative clustering
agg_clustering = AgglomerativeClustering(n_clusters=4)
y_pred = agg_clustering.fit_predict(X)
# Plot the clusters
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='rainbow')
plt.title('Agglomerative Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
# Generate the linkage matrix
Z = linkage(X, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dendrogram(Z)
plt.title('Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```

Name of the Laboratory: **Name of the Experiment:**

Experiment No: **Date:**