

**Roll No: 45**

**Exam Seat No:**

**VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF  
TECHNOLOGY**

Hashu Advani Memorial Complex, Collector's Colony, R. C.  
Marg, Chembur, Mumbai – 400074. Contact No. 02261532532



Since 1962

**CERTIFICATE**

Certified that Mr./Miss Pushkar Prasad Sane  
of FYMCA has satisfactorily completed a course of the  
necessary experiments in AIML - Lab under my  
supervision in the Institute of Technology in the academic year 2023-24.

Principal

Head of Department  
(Dr. Shiv Kumar Goel)

Faculty Incharge

(Dr. Meenakshi Garg)

External Examiner



**V.E.S. Institute of Technology, Collector Colony,  
Chembur, Mumbai  
Department of M.C.A**

**INDEX**

Sr. No	Contents	Date Of Preparation	Date Of Submission	Marks	Sign
1	Implementation of Logic programming using PROLOG- Basic of Prolog DFS for the water jug solve 8- Puzzle Problem. Family Tree (objects and relations)	13/1/24	20/1/24		
2	Introduction to Python Programming: Learn the different libraries - NumPy, Pandas, SciPy, Matplotlib, Scikit Learn	20/1/24	27/1/24		
3	Implementation of Linear Regression, Logistic Regression	27/1/24	03/2/24		
4	Implementation of KNN classification	03/2/24	9/2/24		
5	Implementation of Decision Tree Using an Algorithm (ID3,C4.5,Gini) And Naïve Bayes Classifier	9/2/24	16/2/24		
6	Implementation and analysis of clustering algorithms like K-Means, K-medoid	16/2/24	23/2/24		
7	Implementation of Classifying data using Support Vector Machines (SVMs).	23/2/24	01/3/24		
8	Implementation of Classifying data using Non-Linear Kernels in Support Vector Machines (SVMs).	01/3/24	08/3/24		
9	Implementation of Bagging Algorithm: Decision Tree, Random Forest.	08/3/24	15/3/24		

10	Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.	15/3/24	22/3/24		
11	Implementation of dimensionality reduction techniques: Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis.	22/3/24	29/3/24		
12	Deployment of Machine Learning Models	29/3/24	05/4/24		

AIM: To implement the basics of python

Theory:

Python

Python is a popular and versatile programming language that can be used for various applications such as web development, data analysis, machine learning, and more. In this blog post, we will explain some of the basic concepts and features of Python, such as comparison operators, logical operators, for loops, while loops, range function, lambda expression, map and filter functions.

## Data types

Python has several built-in data types that are used to define variables. These data types include:

- Numeric data types: int, float, complex
  - Sequence data types: list, tuple, range •
- Text data type: str
- Mapping data type: dict
  - Set data type: set, frozenset
  - Boolean data type: bool

Each of these data types has its own properties and methods. For example, you can use the `type()` function to check the data type of a variable.

1. Numeric data types: These are used to represent numbers. The `int` data type is used to represent integers (whole numbers), the `float` data type is used to represent floating-point numbers (numbers with decimal points), and the `complex` data type is used to represent complex numbers (numbers with a real and imaginary part).
2. Sequence data types: These are used to represent sequences of values. The `list` data type is used to represent lists of values, the `tuple` data type is used to represent immutable (unchangeable) lists of values, and the `range` data type is used to represent sequences of numbers.
3. Text data type: This is used to represent strings of characters. The `str` data type is used to represent strings.

4. Mapping data type: This is used to represent mappings between keys and values. The dict data type is used to represent dictionaries.
5. Set data type: This is used to represent sets of unique values. The set data type is used to represent mutable (changeable) sets of values, and the frozenset data type is used to represent immutable (unchangeable) sets of values.
6. Boolean data type: This is used to represent boolean values (True or False)

## Sequence Data types:

- Lists: Lists are used to store a collection of items. They are ordered and mutable (changeable). You can add or remove items from a list using various methods. For example, you can use the `append()` method to add an item to the end of a list, or the `remove()` method to remove an item from a list.

Here's an example of how to create a list in Python:

```
my_list = [1, 2, 3, 4, 5]
```

- Sets: Sets are used to store a collection of unique items. They are unordered and mutable (changeable). You can add or remove items from a set using various methods. For example, you can use the `add()` method to add an item to a set, or the `remove()` method to remove an item from a set.

Here's an example of how to create a set in Python:

```
my_set = {1, 2, 3, 4, 5}
```

- Dictionaries: Dictionaries are used to store key-value pairs. They are unordered and mutable (changeable). You can add or remove items from a dictionary using various methods. For example, you can use the `update()` method to add or update an item in a dictionary, or the `pop()` method to remove an item from a dictionary.

Here's an example of how to create a dictionary in Python:

```
my_dict = {"name": "John", "age": 30}
```

## Operators

Comparison operators are used to compare two values and return a boolean value (True or False) based on the result of the comparison. For example, we can use the `==` operator to check if two values are equal, the `!=` operator to check if they are not equal, the `<` operator to check if one value is less than another, the `>` operator to check if one value is greater than another, and so on. Here are some examples of comparison operators in Python:

### Comparison operators

```
x = 10
y = 20
print(x == y) # False
print(x != y) # True
```

```
print(x < y) # True
print(x > y) # False
print(x <= y) # True
print(x >= y) # False
```

Logical operators are used to combine two or more boolean values and return a boolean value based on the logic of the operation. For example, we can use the and operator to check if both values are True, the or operator to check if at least one value is True, and the not operator to negate a value. Here are some examples of logical operators in Python:

## Logical operators

```
a = True
b = False
print(a and b) # False
print(a or b) # True
print(not a) # False
print(not b) # True
```

For loops are used to iterate over a sequence of items (such as a list, a tuple, a string, etc.) and execute a block of code for each item. For example, we can use a for loop to print each element of a list:

## Loops

### For loop

```
fruits = ["apple", "banana", "orange"]
for fruit in fruits:
    print(fruit)
```

Output:

apple banana orange

While loops are used to execute a block of code repeatedly as long as a condition is True. For example, we can use a while loop to print numbers from 1 to 10:

### While loop

```
n = 1
while n <= 10:
    print(n)
    n = n + 1
```

Output:

1 2 3 4 5 6 7 8 9 10

The range function is used to generate a sequence of numbers within a specified range. For example, we can use the range function to create a list of numbers from 0 to 9:

### Range function

```
numbers = list(range(10))
print(numbers)
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The range function can also take two or three arguments to specify the start, stop, and step values of the sequence. For example, we can use the range function to create a list of even numbers from 2 to 20:

Range function with arguments

```
even_numbers = list(range(2, 21, 2))
print(even_numbers)
```

Output:

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

A lambda expression is a way of creating an anonymous function (a function without a name) in Python. It can take any number of arguments but can only have one expression. For example, we can use a lambda expression to create a function that adds two numbers:

## Lambda expression

```
add = lambda x,y: x + y
print(add(3,5))
```

Output:

```
8
```

A map function is used to apply a function to each element of an iterable (such as a list) and return an iterator object that contains the results. For example, we can use a map function to square each element of a list:

## Map and Filter

Map function

```
numbers = [1,2,3]
squared_numbers = map(lambda x: x**2,numbers)
print(list(squared_numbers))
```

Output:

```
[1 ,4 ,9]
```

A filter function is used to filter out elements of an iterable that do not satisfy a condition and return an iterator object that contains the remaining elements. For example, we can use a filter function to remove odd numbers from a list:

## Filter function

```
numbers = [1, 2, 3, 4, 5]
```

To define a function in Python, you use the `def` keyword followed by the function name and parentheses. Inside the parentheses, you can optionally specify one or more parameters that the function can take as input. After the parentheses, you add a colon and then indent the body of the function. The body contains the statements that define what the function does. For example:

```
def greet(name):
    print("Hello, " + name + "!")
```

This defines a function called `greet` that takes one parameter called `name` and prints a greeting message. To call a function, you use the function name followed by parentheses and pass the arguments that match the parameters. For example:

```
greet("Alice")
greet("Bob")
```

This calls the `greet` function twice with different arguments and prints:

```
Hello, Alice!
Hello, Bob!
```

You can also return a value from a function using the `return` keyword. This allows you to assign the result of a function to a variable or use it in another expression. For example:

```
def add(x, y):
    return x + y
result = add(3, 4)
print(result)
```

This defines a function called `add` that takes two parameters called `x` and `y` and returns their sum. It then calls the `add` function with 3 and 4 as arguments and assigns the result to a variable called `result`. It then prints:

7

Functions are very useful and powerful tools in Python that can help you write better code. I hope this blog post helped you understand what functions are and how to use them.

Map and filter are built-in functions in Python that allow you to apply a function to each element of an iterable object, such as a list or a tuple, and return a new iterable object with the modified elements.

The map function takes two arguments: a function and an iterable. The function can be any callable object that takes one argument and returns a value. The iterable can be any object that supports iteration, such as a list, a tuple, a string, a dictionary, or a generator.

The map function returns a map object, which is also an iterable, that contains the results of applying the function to each element of the iterable.

For example, suppose you have a list of numbers and you want to square each number. You can use the map function to do this:

```
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(lambda x: x**2, numbers) # Lambda is an
# anonymous function that takes one argument x and returns x**2
print(list(squared_numbers)) # convert the map object to a list to
# print it
```

Output: [1, 4, 9, 16, 25]

The filter function takes two arguments: a function and an iterable. The function can be any callable object that takes one argument and returns a boolean value (True or False). The iterable can be any object that supports iteration, such as a list, a tuple, a string, a dictionary, or a generator. The filter function returns a filter object, which is also an iterable, that contains the elements of the iterable for which the function returns True.

For example, suppose you have a list of names and you want to filter out the names that start with 'A'. You can use the filter function to do this:

```
names = ['Alice', 'Bob', 'Charlie', 'David', 'Eve']
filtered_names = filter(lambda x: not x.startswith('A'), names) #
# Lambda is an anonymous function that takes one argument x and returns
# True if x does not start with 'A'
print(list(filtered_names)) # convert the filter object to a list to
# print it
```

Output: ['Bob', 'Charlie', 'David', 'Eve']

In summary, map and filter are useful functions in Python that allow you to transform and filter iterables using custom functions. They return iterable objects that can be converted to other data types or used in loops or comprehensions.

## Data Types

### Numbers

In [1]: 15 + 12

Out[1]: 27

In [2]: 1\*15

Out[2]: 15

In [3]: 1/2

```
Out[3]: 0.5
```

```
In [4]: 2**3
```

```
Out[4]: 8
```

```
In [5]: 15%2
```

```
Out[5]: 1
```

```
In [6]: 5%2
```

```
Out[6]: 1
```

```
In [7]: (21+3)*(58+5)
```

```
Out[7]: 1512
```

```
In [8]: name_of_var = 2
```

```
In [11]: x=21  
y=39
```

```
In [12]: z=x+y
```

```
In [13]: z
```

```
Out[13]: 60
```

## Strings

```
In [14]: 'single quotes'
```

```
Out[14]: 'single quotes'
```

```
In [15]: "doube quotes"
```

```
Out[15]: 'doube quotes'
```

```
In [16]: " wrap lot's of other quotes"
```

```
Out[16]: " wrap lot's of other quotes"
```

## Printing

```
In [17]: x="hello"
```

```
In [18]: x
```

```
Out[18]: 'hello'
```

```
In [19]: print(x)
```

```
hello
```

```
In [20]: RollNO=15  
        name='Sidharth'
```

```
In [22]: print('My number is {one}, and my name is : {two}'.format(one=RollNO,two=name))  
My number is 15, and my name is : Sidharth
```

```
In [23]: print('My number is : {}, and my name is : {}'.format(RollNO,name))  
My number is : 15, and my name is : Sidharth
```

## Lists

```
In [24]: [1,2,3]
```

```
Out[24]: [1, 2, 3]
```

```
In [25]: ['hi',1,[1,2]]
```

```
Out[25]: ['hi', 1, [1, 2]]
```

```
In [26]: my_list=['a','b','c']
```

```
In [27]: my_list.append('d')
```

```
In [28]: my_list
```

```
Out[28]: ['a', 'b', 'c', 'd']
```

```
In [29]: my_list[0]
```

```
Out[29]: 'a'
```

```
In [30]: my_list[1]
```

```
Out[30]: 'b'
```

```
In [31]: my_list[1:]
```

```
Out[31]: ['b', 'c', 'd']
```

```
In [32]: my_list[:1]
```

```
Out[32]: ['a']
```

```
In [33]: my_list[0]='NEW'
```

```
In [34]: nest = [1,2,3,[4,5,['target']]]
```

```
In [35]: nest[3]
```

```
Out[35]: [4, 5, ['target']]
```

```
In [36]: nest[3][2]
```

```
Out[36]: ['target']
```

```
In [37]: nest[3][2][0]
```

```
Out[37]: 'target'
```

## Dictionaries

```
In [38]: d = {'key1': 'item1', 'key2': 'item2'}
```

```
In [39]: d
```

```
Out[39]: {'key1': 'item1', 'key2': 'item2'}
```

```
In [40]: d['key1']
```

```
Out[40]: 'item1'
```

## Booleans

```
In [41]: True
```

```
Out[41]: True
```

```
In [42]: False
```

```
Out[42]: False
```

## Tuples

```
In [43]: t=(1,2,3)
```

```
In [44]: t[0]
```

```
Out[44]: 1
```

```
In [45]: t[0] = 'NEW'
```

```
-----  
TypeError  
Cell In[45], line 1  
----> 1 t[0] = 'NEW'
```

```
Traceback (most recent call  
last)
```

```
TypeError: 'tuple' object does not support item assignment
```

## Sets

```
In [46]: {1,2,3}
```

```
Out[46]: {1, 2, 3}
```

In [47]: `{1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2}`

```
Out[47]: {1, 2}
```

## Comparison Operators

```
In [48]: 1>2
```

```
Out[48]: False
```

```
In [49]: 1<2
```

```
Out[49]: True
```

```
In [50]: 1>=2
```

```
Out[50]: False
```

```
In [51]: 1<=4
```

```
Out[51]: True
```

```
In [52]: 1==1
```

```
Out[52]: True
```

## Logic Operators

```
In [53]: 'HI'=='BYE'
```

```
Out[53]: False
```

```
In [54]: (1>2)and(2<3)
```

```
Out[54]: False
```

```
In [57]: (15 > 2) or (2 < 3)
```

```
Out[57]: True
```

```
In [58]: (51 == 2) or (24 == 3) or (4 == 4)
```

```
Out[58]: True
```

If else statements

```
In [60]:
```

```
if 1 < 2:  
    print('SiddyBoy')
```

SiddyBoy

```
In [61]: if 1 < 2:  
    print('SiddyBoy')
```

SiddyBoy

```
In [62]: if 1 < 2:  
    print('first')  
else:  
    print('last')
```

first

```
In [63]: if 1 > 2:  
    print('first')  
else:  
    print('last')
```

last

```
In [64]: if 1 == 2:  
    print('first')  
elif 3 == 3:  
    print('middle')  
else:  
    print('Last')
```

middle

## for loops

```
In [65]: seq = [1,2,3,4,5]
```

```
In [66]: for item in seq:  
    print(item)
```

1  
2  
3  
4  
5

```
In [67]: for item in seq:  
    print('Yep')
```

Ye  
p  
Ye  
p  
Ye  
p  
Ye  
p  
Ye  
p

```
In [68]: for jelly in seq:  
    print(jelly+jelly)
```

2  
4  
6  
8  
10

## while loops

```
In [69]: i = 1  
while i < 5:
```

```
    print('i is: {}'.format(i))
    i = i+1
```

```
i is: 1
i is: 2
i is: 3
i is: 4
```

## range()

```
In [70]: range(5)
```

```
Out[70]: range(0, 5)
```

```
In [71]: list(range(5))
```

```
Out[71]: [0, 1, 2, 3, 4]
```

## list comprehension

```
In [72]: x = [1,2,3,4]
```

```
In [73]: out = []
for item in x:
    out.append(item**2)
print(out)
```

```
[1, 4, 9, 16]
```

```
In [74]: [item**2 for item in x]
```

```
Out[74]: [1, 4, 9, 16]
```

## functions

```
In [75]: def my_func(param1='default'):
    """
    Docstring goes here.
    """
    print(param1)
```

```
In [76]: my_func
```

```
Out[76]: <function _main_.my_func(param1='default')> In
```

```
[77]:
```

```
my_func()
```

```
default
```

```
In [78]: my_func('new param')
```

```
new param
```

```
In [79]: my_func(param1='new param')
```

```
new param
```

```
In [80]: def square(x):
           return x**2

In [81]: out = square(2)

In [82]: print(out)

4

In [83]: def times2(var):
           return var*2

In [ ]: times2(2)
```

Out[ ]: 4

## lambda expressions

```
lambda var: var*2
```

```
Out[84]: <function _main_.<lambda>(var)>
```

```
In [85]: seq = [1,2,3,4,5]
```

## map and filter

```
In [86]: map(times2,seq)
```

```
Out[86]: <map at 0x7fce47764be0>
```

```
In [87]: list(map(times2,seq))
```

```
Out[87]: [2, 4, 6, 8, 10]
```

```
In [88]: list(map(lambda var: var*2,seq))
```

```
Out[88]: [2, 4, 6, 8, 10]
```

```
In [89]: filter(lambda item: item%2 == 0,seq)
```

```
Out[89]: <filter at 0x7fce47767a90>
```

```
In [90]: list(filter(lambda item: item%2 == 0,seq))
```

```
Out[90]: [2, 4]
```

## methods

```
In [91]: st = 'hello my name is Ritika Dobhal'
```

```
In [92]: st.lower()
```

```
Out[92]: 'hello my name is Pushkar Sane'
```

```
In [93]: st.upper()
```

```
Out[93]: 'HELLO MY NAME IS Pushkar Sane' In
```

```
[94]: st.split()
```

```
Out[94]: ['hello', 'my', 'name', 'is', 'Sidharth', 'Krishnan']
```

```
In [95]: tweet = 'Go Sports! #Sports'
```

```
In [96]: tweet.split('#')
```

```
Out[96]: ['Go Sports!', 'Sports']
```

```
In [97]: tweet.split('#')[1]
```

```
Out[97]: 'Sports'
```

```
In [98]: d
```

```
Out[98]: {'key1': 'item1', 'key2': 'item2'}
```

```
In [101... d.keys()
```

```
Out[101]: dict_keys(['key1', 'key2'])
```

```
In [102... d.items()
```

```
Out[102]: dict_items([('key1', 'item1'), ('key2', 'item2')])
```

```
In [103... lst = [1,2,3]
```

```
In [104... lst.pop()
```

```
Out[104]: 3
```

```
In [105... lst
```

```
Out[105]: [1, 2]
```

```
In [106...
```

```
'x' in [1,2,3]
```

```
Out[106]: False
```

```
In [107... 'x' in ['x','y','z']
```

```
Out[107]: True
```

---

```
*** Python Data Types Exercise
```

## Exercises

\*\* What is 8 to the power of 5?\*\*

In [108... 8\*\*5

Out[108]: 32768

\*\* Split this string:\*\*

s = "Hi there Tushar!"

into a list.

In [109...]

```
s = "Hello My Name Is Ritika Dobhal" s.split()
```

Out[109]:

```
['Hello', 'My',  
 'Name', 'Is',  
 'Sidharth',  
 'Krishnan']
```

\*\* Given the variables:\*\*

```
p  
l  
a  
n  
e  
t  
= "  
M  
a  
r  
s  
"  
d  
i  
a  
m  
e  
t  
e  
r  
= 1  
5  
7  
4  
2
```

In [110...]

\*\* Use .format() to print the following string: \*\*

The diameter of  
Mars is 15742  
kilometers.

\*\* Given this nested list, use indexing to grab the word "hii" \*\*

```
planet = "Mars"
diameter = 15742
print(f'The diameter of {planet} is {diameter} kilometers.')
```

The diameter of Mars is 15742 kilometers.

In       `lst = [1,2,[3,4],[5,[100,200,['hii']],23,11],10,7]`

[111...     `d = {'k1':[1,2,3,['tricky':['OK','woman','deception',{'Rule':[1,2,3,'hii']}]]}]`

In       `d['k1'][3]['tricky'][3]['Rule'][3]`

[112...]

In

[113...]

Out[113]: 'hii'

\*\* What is the main difference between a tuple and a list? \*\*

The main difference between a tuple and a list is that a tuple is immutable while list is mutable.

\*\* Create a function that grabs the email website domain from a string in the form: \*\*

tushar@ves.ac.in

So for example, passing "tushar@ves.ac.in" would return: ves.ac.in

```
In [115... def getDomain(email:str):return email.split("@")[1]
getDomain("2022.Sidharth.Krishnan@ves.ac.in")
```

```
Out[115]: 'ves.ac.in'
```

\*\* Create a basic function that returns True if the word 'cat' is contained in the input string. Do account for capitalization. \*\*

```
In [116... def findcat(string:str):
    return string.find('cat')!=-1
```

```
In [117... findcat('Is there a cat here?')
```

```
Out[117]: True
```

\*\* Create a function that counts the number of times the word "cat" occurs in a string. Again ignore edge cases. \*\*

```
In [119... def countDog(string:str):
    count=0
    for x in string.split():
        if len(x)>=3 and x[:3]=='cat':
            count+=1
    return count
```

```
In [120... countDog('This cat runs faster than the other cats dude!')
```

```
Out[120]: 2
```

\*\* Use lambda expressions and the filter() function to filter out words from a list that don't start with the letter 's'. For example:\*\*

```
seq =
['soup', 'dog', 'salad', 'cat', 'great']
```

should be filtered down to:

```
['soup', 'salad']
```

```
In [121... seq = ['soup', 'dog', 'salad', 'cat', 'great']
```

```
In [122... list(filter(lambda item: item in ['soup', 'salad'], seq))
```

```
Out[122]: ['soup', 'salad']
```

## Final Problem

You are driving a little too fast, and a police officer stops you. Write a function to return one of 3 possible results: "No ticket", "Small ticket", or "Big Ticket". If your speed is 60 or less, the result is "No Ticket". If speed is between 61 and 80

inclusive, the result is "Small Ticket". If speed is 81 or more, the result is "Big Ticket". Unless it is your birthday (encoded as a boolean value in the parameters of the function) -- on your birthday, your speed can be 5 higher in all cases.

```
In [123... def caught_speeding(speed, is_birthday):
    if (is_birthday):
        speed-=5
    if (speed<=60):
        return "No Ticket"
    if (speed<=80):
        return "Small Ticket"
    return "Big Ticket"
```

```
In [124... caught_speeding(81,True)
```

```
Out[124]: 'Small Ticket'
```

```
In [125... caught_speeding(81,False)
```

```
Out[125]: 'Big Ticket'
```

## Questions

1. Write a program to find a number is odd or even

```
In [126... 
```

```
def oddOrEven(n:int):
    return "Even" if n%2==0 else "Odd"
print(oddOrEven(6))
print(oddOrEven(7))
```

E  
v  
e  
n  
o  
d  
d

2. to find if a number is prime or not

```
In [127... def isPrime(n:int):
    comp = "Composite"
    if n%2==0: return comp
    for i in range(3,n//2,2):
        if n%i==0: return comp
    return "Prime"
```

```
In [128... print(isPrime(97))
print(isPrime(27))
```

Prime  
Composite

3. triangle/pyramid pattern

```
In [129... def triangle(n:int=5):
    for i in range(1,n+1):
        print("*"*i)
triangle(9)
```

```
*  
**  
***  
****  
*****  
*****  
*****  
*****
```

In [130...]

```
def pyramid(n:int=5):  
    for i in range(1,n+1):  
        print(" "* (n-i)+"* " *i)  
pyramid(5)
```

```
*  
* *  
* * *  
* * * *  
* * * * *
```

In [131...]

```
def opposite_triangle(n:int=5):  
    for i in range(1,n+1):  
        print(" "* (n-i)+"*" *i)  
opposite_triangle(5)
```

```
*  
**  
***  
****  
*****
```

#### 4. factorial

In [132...]

```
def fact(num:int):return 1 if num<=1 else num*fact(num-1)  
fact(5)
```

Out[132]:

120

#### 5. largest of n

In [133...]

```
def largest(seq):return max(seq)  
largest([9,78,45,23])
```

Out[133]:

78

#### 6. sum of digits

In [134...]

```
def add_all(*num):  
    sum=0  
    for number in num:sum+=number  
    return sum  
add_all(1,2,3,4,5)
```

Out[134]:

15

## 7. Calculator

```
In [135...]class Calculator:
    def add(self,*nums):
        sum=0
        for number in nums:
            sum+=number
        return sum
    def subtract(self,num_a,num_b):return num_a-num_b
    def multiply(self,num_a,num_b):return num_a*num_b
    def divide(self,num_a,num_b):return num_a/num_b
    def mod(self,num_a,num_b):return num_a%num_b
calc = Calculator()
a,b=5,6
print("Sum is ",calc.add(a,b))
print("Difference is ",calc.subtract(a,b))
print("Product is ",calc.multiply(a,b))
print("Quotient is ",calc.divide(a,b))
print("Remainder is ",calc.mod(a,b))
```

```
Sum is 11
Difference is -1
Product is 30
Quotient is 0.8333333333333334
Remainder is 5
```

## 8. gross salary

```
In [138...]def gross_salary(basic=0,da=0,hra=0):return basic+da+hra

print("Salary: ",gross_salary(int(input("Basic: ")),int(input("Dearness Allowanc
```

```
Salary: 360
```

## 9. fibonacci

```
In [139...]def fibonacci(n):return n if n<=1 else (fibonacci(n-1)+fibonacci(n-2))
print(*list(map(fibonacci,range(int(input("Enter the number of fibonaci numbers
```

```
0, 1, 1, 2, 3
```

## 10. first n prime numbers

```
In [140...]def findPrime(N):
    primes = [2]
    num = 3
    while len(primes) < N:
        is_prime = True
        for prime in primes:
            if num % prime == 0:
                is_prime = False
                break
        if is_prime:
            primes.append(num)
        num += 2
```

```
    return primes
print(*findPrime(int(input("How many primes do you want: "))))
```

```
2 3 5 7 11 13 17 19 23 29
```

Conclusion: I have successfully understood and implemented basics of python.

<b>Name of Student: Pushkar Prasad Sane</b>			
<b>Roll Number: 45</b>	<b>Lab Assignment Number: 1</b>		
<b>Title of Lab Assignment:</b>			
<b>Implementation of logic programming using PROLOG-DFS for the water jug problem.</b>			
<b>DOP: 13/1/24</b>		<b>DOS: 20/1/24</b>	
<b>CO:</b> <b>CO1</b>	<b>PO:</b> <b>PO1, PO2, PO3, PSO1, PSO2</b>	<b>Faculty Signature:</b>	<b>Marks:</b>

## PRACTICAL - 1

### Aim:

Implementation of logic programming using PROLOG-DFS for the water jug problem.

### Theory:

#### Prolog:

Prolog is a logic programming language. It has important role in artificial intelligence. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Core heart of prolog lies at the logic being applied. Formulation or Computation is carried out by running a query over these relations.

### Water Jug Problem:

There are two jugs of volume A litre and B litre. Neither has any measuring mark on it. There is a pump that can be used to fill the jugs with water. How can you get exactly x litre of water into the A litre jug. Assuming that we have unlimited supply of water. Let's assume we have A=4 litre and B= 3 litre jugs. And we want exactly 2 Litre water into jug A (i.e 4 litre jug) how we will do this.

### Solution:

The state space for this problem can be described as the set of ordered pairs of integers (x,y)

Where,

x represents the quantity of water in the 4-gallon jug x=

0,1,2,3,4 y represents the quantity of water in 3-gallon jug

y=0,1,2,3 Start State: (0,0)

Goal State: (2,0)

Generate production rules for the water jug problem

We basically perform three operations to achieve the goal.

- Fill water jug.
- Empty water jug
- and Transfer water jug

#### Initialization:

- Start State: (0,0)
- Apply Rule 2:
- Fill 3-gallon jug
- Now the state is (x,3)

#### Iteration 1:

- Current State: (x,3)
- Apply Rule 7:
- Pour all water from 3-gallon jug into 4-gallon jug
- Now the state is (3,0)

#### Iteration 2:

- Current State : (3,0)

- Apply Rule 2:
- Fill 3-gallon jug
- Now the state is (3,3)

**Iteration 3:**

- Current State:(3,3)
- Apply Rule 5:
- Pour water from 3-gallon jug into 4-gallon jug until 4-gallon jug is full
- Now the state is (4,2)

**Iteration 4:**

- Current State : (4,2)
- Apply Rule 3:
- Empty 4-gallon jug
- Now state is (0,2)

**Iteration 5:**

- Current State : (0,2)
- Apply Rule 9:
- Pour 2 gallon water from 3 gallon jug into 4 gallon jug
- Now the state is (2,0)-- **Goal Achieved.**

**Execution:**

Code:

```

member(X,[X|_]).  

member(X,[Y|Z]):-member(X,  

Z).  
  

move(X,Y,_):-X=:=2,Y=:=0,write('done'),!.  

move(X,Y,Z):-X<4,\+member((4,Y),Z),write("fill 4 jug"),nl,move(4,Y,[(4,Y)|Z]).  

move(X,Y,Z):-Y<3,\+member((X,3),Z),write("fill 3 jug"),nl,move(X,3,[(X,3)|Z]).  

move(X,Y,Z):-X>0,\+member((0,Y),Z),write("pour 4 jug"),nl,move(0,Y,[(0,Y)|Z]).  

move(X,Y,Z):-Y>0,\+member((X,0),Z),write("pour 3 jug"),nl,move(X,0,[(X,0)|Z]).  

move(X,Y,Z):-P is X+Y,P>=4,Y>0,K is 4-X,M is Y-K,\+member((4,M),Z),write("pour from 3jug to  

4jug"),nl,move(4,M,[(4,M)|Z]).  

move(X,Y,Z):-P is X+Y,P>=3,X>0,K is 3-Y,M is X-K,\+member((M,3),Z),write("pour from 4jug to  

3jug"),nl,move(M,3,[(M,3)|Z]).  

move(X,Y,Z):-K is X+Y,K<4,Y>0,\+member((K,0),Z),write("pour from 3jug to 4jug"),nl,move(K,0,[(K,0)|Z]).  

move(X,Y,Z):-K is X+Y,K<3,X>0,\+member((0,K),Z),write("pour from 4jug to 3jug"),nl,move(0,K,[(0,K)|Z]).
```

Output :

```
move(0,0,[(0,0)]).  
Singleton variables: [Y]  
fill 4 jug  
fill 3 jug  
pour 4 jug  
pour 3 jug  
fill 4 jug  
pour from 4jug to 3jug  
pour 3 jug  
pour from 4jug to 3jug  
fill 4 jug  
pour from 4jug to 3jug  
pour 3 jug  
done  
true  
Next 10 100 1,000 Stop  
?- move(0,0,[ (0,0) ]).
```

## Conclusion:

I have successfully implemented prolog statements for the waterjug problem.

## 0.1 Practical-2.1

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

[1]:

NumPy - Indexing & Slicing Basic slicing is an

[2]: `Sidharth_list = [1,2,3]` Sidharth\_list

giving start, stop, and step parameters to the built-in slice function. This slice object is passed to the array to extract a part of array.

```
import numpy as np
```

[2]: [1, 2, 3]

[3]: `np.array(Sidharth_list)`

[3] : array([1, 2, 3])

[4] : `Sidharth_matrix = [[1,2,3],[4,5,6],[7,8,9]]`  
Sidharth\_matrix

[4] : [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

[5] : `np.array(Sidharth_matrix)`

[5] : array([[1, 2, 3],  
[4, 5, 6],  
[7, 8, 9]])

### 0.1.1 Built-in Methods

#### 0.1.2 arange

The arange([start,] stop[, step,][, dtype]) : Returns an array with evenly spaced elements as per the interval. The interval mentioned is half-opened i.e. [Start, Stop)

[6] : `np.arange(0,10)`

[6]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[7] : `np.arange(0,11,2)`

[7]: array([ 0, 2, 4, 6, 8, 10])

#### 0.1.3 Zeros & Ones

[8] : `np.zeros(3)`

[8]: array([0., 0., 0.])

[9]: `#5x5 zero matrix  
np.zeros((5,5))`

[9] : array([[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.]])

[10] : `np.ones(3)`

[10] : array([1., 1., 1.])

[11] : `np.ones((3,3))`

[11] : array([[1., 1., 1.],  
[1., 1., 1.],  
[1., 1., 1.]])

#### 0.1.4 Linspace

Return Evenly spaced numbers over a specified range

[12]:

`# 3 elements between 0 to 10  
np.linspace(0,10,3)`

[12] : array([ 0., 5., 10.])

```
[13] : np.linspace(0,10,50)
```

```
[13] : array([ 0.          ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
   1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
   2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
   3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
   4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
   5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
   6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
   7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
   8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
   9.18367347,  9.3877551 ,  9.59183673,  9.79591837,  10.        ])
```

## 0.1.5 Eye

### Create an Identity Matrix

```
[14] :
```

```
#4x4 Identity Matrix  
np.eye(4)
```

```
[14] : array([[1., 0., 0., 0.],
   [0., 1., 0., 0.],
   [0., 0., 1., 0.],
   [0., 0., 0., 1.]])
```

## 0.1.6 Random

Numpy has lots of ways to create random number arrays

## 0.2 Rand

Create an array of the given shape and populate it with random samples from uniform # 2 random numbers

```
[15]:
```

```
np.random.rand(2)
```

```
[15] : array([0.7489362 , 0.94822426])
```

```
[16] : np.random.rand(5,5)
```

```
[16]: array([[0.54400778, 0.40068716, 0.38635204, 0.35936806, 0.23916308],
   [0.32892716, 0.44880486, 0.18161207, 0.90373679, 0.96193848],
   [0.43943677, 0.24012986, 0.20639319, 0.37164465, 0.66593935],
   [0.77889016, 0.44666776, 0.69550924, 0.70738623, 0.07473673],
   [0.79260275, 0.31395118, 0.4638373 , 0.28036785, 0.89473988]])
```

### 0.2.1 Randn

[17]: `np.random.randn(2)`

[17] : array([-3.27383214, -0.16980301])

[18] : `np.random.randn(5,5)`

[18]: array([[ 0.76219635, 0.68182311, -1.10040674, 0.19793157, -0.41483687],  
[-0.08897539, -0.06879698, 1.24863616, 0.67120482, -0.38905689],  
[-1.23421977, 0.05228506, 0.58507694, 0.13232243, -1.44544138],  
[-1.38597096, -0.92113811, -1.68108643, -1.14943345, 1.57333774],  
[-0.35787929, -1.53896243, 0.78326792, -2.84415079, -0.23842257]])

### 0.2.2 randint

**Return random integers from low (inclusive) to high** `np.random.randint(1,100)`

[19] :

[19]: 28

[20] : `#generate 10 random numbers  
np.random.randint(1,100,10)`

[20] : array([34, 46, 91, 55, 61, 6, 93, 79, 65, 2])

### 0.2.3 Array Attributes & Methods

[21] : `Sidharth_arr1 = np.arange(25)  
= np.random.randint(0,50,10)`

[22] : `Sidharth_arr1`

[22] : array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,  
17, 18, 19, 20, 21, 22, 23, 24])

[23] : `ranarr`

[23] : array([ 2, 35, 48, 21, 26, 41, 19, 37, 9, 27])

### 0.2.4 Reshape

**Returns an array containing the same data with a different shape** `Sidharth_arr1.reshape(5,5)`

[24] :

[24]: array([[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

```
[10, 11, 12, 13, 14],  
[15, 16, 17, 18, 19],  
[20, 21, 22, 23, 24]])
```

### 0.2.5 max, min, argmax, argmin

```
[25]: ranarr
```

```
[25] : array([ 2, 35, 48, 21, 26, 41, 19, 37,  
              9, 27])
```

```
[26] : ranarr.max() #returns maximum from array
```

```
[26]: 48
```

```
[27] : ranarr.min() #returns minimum from array
```

```
[27]: 2
```

```
[28] : ranarr.argmax() #returns index of max element
```

```
[28]: 2
```

```
[29] : ranarr.argmin() #returns index of min element.
```

```
[29]: 0
```

### 0.2.6 shape

**shape is an attribute that arrays have**  
**dimens** Sidharth\_arr1.shape

```
[30] :
```

```
[30]: (25,)
```

```
[31] : Sidharth_arr1.reshape(1,25)
```

```
[31] : array([[ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,  
              16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

```
[32] : Sidharth_arr1.reshape(1,25).shape
```

```
[32]: (1, 25)
```

```
[33] : Sidharth_arr1.reshape(25,1)
```

```
[33] : array([[ 0],  
              [ 1],  
              [ 2],
```

```
[ 3],  
[ 4],  
[ 5],  
[ 6],  
[ 7],  
[ 8],  
[ 9],  
[10],  
[11],  
[12],  
[13],  
[14],  
[15],  
[16],  
[17],  
[18],  
[19],  
[20],  
[21],  
[22],  
[23],  
[24] ])
```

[34] : Sidharth\_arr1.reshape(25,1).shape

[34]: (25, 1)

## o.2.7 dtype

### Find data type of the object in an array

[35] :

Sidharth\_arr1.dtype

[35] : dtype('int64')

## o.3 Numpy Operations

### o.3.1 1. Arithemetic

[36] :

arr2 = np.arange(0,10)  
arr2

[36]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[37]: arr2 + arr2

[37]: array([ 0, 2, 4, 6, 8, 10, 12, 14, 16, 18])

```
[38]: arr2 * arr2
```

```
[38]: array([ 0, 1, 4, 9, 16, 25, 36, 49, 64, 81])
```

```
[39]: arr2 - arr2
```

```
[39]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
[40]: arr2 / arr2
```

```
<ipython-input-40-a5d8023d8ed7>:1: RuntimeWarning: invalid value encountered in true_divide  
arr2 / arr2
```

```
[40]: array([nan, 1., 1., 1., 1., 1., 1., 1., 1.])
```

```
[41]: 1/ arr2
```

```
<ipython-input-41-39a5363a91ec>:1: RuntimeWarning: divide by zero encountered in  
true_divide  
1/ arr2
```

```
[41]: array([ inf, 1. , 0.5 , 0.33333333, 0.25 ,  
0.2 , 0.16666667, 0.14285714, 0.125 , 0.11111111])
```

```
[42]: arr2 ** 3
```

```
[42]: array([ 0, 1, 8, 27, 64, 125, 216, 343, 512, 729])
```

### o.3.2 Universal Array Functions

```
[43]: # Square roots  
np.sqrt(arr2)
```

```
[43]: array([0. , 1. , 1.41421356, 1.73205081, 2.  
2.23606798, 2.44948974, 2.64575131, 2.82842712, 3. ,  
])
```

```
[44]: # Calculating Exponential (e^x)  
np.exp(arr2)
```

```
[44]: array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00,  
2.00855369e+01, 5.45981500e+01, 1.48413159e+02,  
4.03428793e+02, 1.09663316e+03,  
2.98095799e+03, 8.10308393e+03])
```

```
[45] : np.max(arr2)
```

```
[45]: 9
```

```
[46] : np.min(arr2)
```

[46]: 0

```
[47] : np.sin(arr2)
```

```
[47] : array([ 0.          ,  0.84147098, 0.90929743, 0.14112001, -0.7568025 ,
 -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825, 0.41211849])
```

```
[48] : #create sample array
Sidharth_arr = np.arange(0,5) # 5 is exclusive
Sidharth_arr
```

```
[48] : array([0, 1, 2, 3, 4])
```

## 1 Bracket Indexing & Selection

```
[49] :
```

```
#Get a value at an index
Sidharth_arr[3]
```

[49]: 3

```
[50] : #Get values in a range
Sidharth_arr[1:3] #1 - Inclusive & 3 Exclusive
```

```
[50] : array([1, 2])
```

```
[51] : #Get values in a range
Sidharth_arr[0:4]
```

```
[51] : array([0, 1, 2, 3])
```

### 1.1 Broadcasting

```
[52] :
```

```
#Setting a value with index range
Sidharth_arr[0:2] = 500 Sidharth_arr
```

```
[52] : array([500, 500,      2,      3,      4])
```

```
[53] : #rest array
Sidharth_arr = np.arange(0,15)
Sidharth_arr
```

```
[53] : array([ 0,      1,      2,      3,      4,      5,      6,      7,      8,      9, 10, 11, 12, 13, 14])
```

```
[54]: #slices  
slice_of_arr = Sidharth_arr[0:5]  
slice_of_arr
```

[54] : array([0, 1, 2, 3, 4])

```
[55]: #change of slice  
# replacing every element with 55 in array  
slice_of_arr[:] = 55  
slice_of_arr
```

[55] : array([55, 55, 55, 55, 55])

```
[56]: Sidharth_arr
```

[56]: array([55, 55, 55, 55, 55, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])

```
[57]: #copy array  
arr_copy = Sidharth_arr.copy() arr_copy
```

[57]: array([55, 55, 55, 55, 55, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])

### 1.1.1 Indexing a 2D array

```
[58]: arr_2d = np.array(([2,4,6],[8,10,12],[14,16,18]))  
arr_2d
```

[58]: array([[ 2, 4, 6],  
 [ 8, 10, 12],  
 [14, 16, 18]])

```
[59]: #indexing row  
arr_2d[1]
```

[59] : array([ 8, 10, 12])

```
[60] : #Format is arr_2d[row][col] or arr_2d[row,col]  
arr_2d[1][1]
```

[60]: 10

```
[61] : #Get individual element value  
arr_2d[1,0]
```

[61]: 8

```
[62] : #2d array slicing  
#shape (2,2) from top right corner  
arr_2d[:1,2:]
```

```
[62] : array([[6]])
```

```
[63] : #shape bottom row  
arr_2d[2]
```

```
[63] : array([14, 16, 18])
```

```
[64] : #shape bottom row  
arr_2d[1,:]
```

```
[64] : array([ 8, 10, 12])
```

## 1.2 Fancy Indexing

```
[65] : #Set up matrix  
arr2d = np.zeros((10,10))  
arr2d
```

```
[65]: array([[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.],  
           [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.],  
           [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.],  
           [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
```

```
[66] : #length of array Sidharth_arr_length  
= arr2d.shape Sidharth_arr_length
```

```
[66]: (10, 10)
```

```
[67] : Sidharth_arr_length = arr2d.shape[1]  
Sidharth_arr_length
```

```
[67]: 10
```

```
[68] : #Set up array  
for i in range(Sidharth_arr_length): arr2d[i]=i
```

## arr2d

```
[68]: array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
   [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
   [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
   [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
   [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
   [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
   [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
   [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
   [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],
   [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])
```

### 1.2.1 Fancy Indexing allows the following

```
[69] : arr2d[[2,4,6,8]]
```

```
[69] : array([[2., 2., 2., 2., 2., 2., 2., 2.],
   [4., 4., 4., 4., 4., 4., 4., 4.],
   [6., 6., 6., 6., 6., 6., 6., 6.],
   [8., 8., 8., 8., 8., 8., 8., 8.]])
```

```
[70]: arr2d[[6,4,2,7]]
```

```
[70] : array([[6., 6., 6., 6., 6., 6., 6., 6.],
   [4., 4., 4., 4., 4., 4., 4., 4.],
   [2., 2., 2., 2., 2., 2., 2., 2.],
   [7., 7., 7., 7., 7., 7., 7., 7.]])
```

### 1.2.2 Selection

```
[71]: arr = np.arange(1,11)
arr
```

```
[71] : array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
[72] : arr > 6
```

```
[72]: array([False, False, False, False, False,
   True])
```

```
[73]: Sidharth_bool_arr = arr > 6
Sidharth_bool_arr
```

```
[73]: array([False, False, False, False, False,
   True])
```

[74]:

```
[74]: array([ 7, 8, 9, 10])
```

[75]: arr[arr>2]

```
[75]: array([ 3, 4, 5, 6, 7, 8, 9, 10])
```

[76]: x=2  
arr[arr > x]

```
[76]: array([ 3, 4, 5, 6, 7, 8, 9, 10])
```

# This is formatted as code

## 2 NumPy Exercises

Now that we've learned about NumPy let's test your knowledge. We'll start off with a few simple tasks, and then you'll be asked some more complicated questions.

### Import NumPy as np

[77] :

```
import numpy as np
```

### Create an array of 10 zeros

[78] :

```
np.zeros(10,dtype=float)
```

```
[78]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

### Create an array of 10 ones

[79] : np.ones(10,dtype=float)

```
[79]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

### Create an array of 10 fives

[80] : arr = np.ones(10) \* 5  
arr

```
[80]: array([5., 5., 5., 5., 5., 5., 5., 5., 5., 5.])
```

**Create an array of the integers from 10 to 50**[81] : `np.arange(10,51)`

```
[81] : array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
   27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
   44, 45, 46, 47, 48, 49, 50])
```

**Create an array of all the even integers from 10 to 50**[82] : `np.arange(10,51,2)`

```
[82] : array([10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42,
   44, 46, 48, 50])
```

**Create a 3x3 matrix with values ranging from 0 to 8**[83] : `np.arange(0, 9).reshape(3,3)`

```
[83] : array([[0, 1, 2],
   [3, 4, 5],
   [6, 7, 8]])
```

**Create a 3x3 identity matrix**[84] : `np.eye(3)`

```
[84] : array([[1., 0., 0.],
   [0., 1., 0.],
   [0., 0., 1.]])
```

**Use NumPy to generate a random number between 0 and 1**[85] : `np.random.randint(0,2)`

```
[85]: 0
```

**Use NumPy to generate an array of 25 random numbers sampled from a standard normal distribution**[86] : `np.random.normal(0,1,25)`

```
[86]: array([-0.9792705 , -0.68324493, -0.95723843,  1.4721858 , -0.9361654 ,
   -0.31290161, -0.72619955,  0.54870017,  1.18955423, -0.04214437,
   -1.52966088, -1.40202744, -0.76799015, -0.77391678,  0.78609116,
   2.05305123,  0.88620352, -2.85189376, -0.69334934,  1.62491863,
   -0.31168407, -1.00185711, -1.04296744, -0.55189568,  0.29083458])
```

**Create the following matrix:**

```
[87] : np.linspace(0.01, 1, num=100).reshape(10, 10)
```

```
[87]: array([[0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 ],
 [0.11, 0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19, 0.2 ],
 [0.21, 0.22, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 0.3 ],
 [0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 0.38, 0.39, 0.4 ],
 [0.41, 0.42, 0.43, 0.44, 0.45, 0.46, 0.47, 0.48, 0.49, 0.5 ],
 [0.51, 0.52, 0.53, 0.54, 0.55, 0.56, 0.57, 0.58, 0.59, 0.6 ],
 [0.61, 0.62, 0.63, 0.64, 0.65, 0.66, 0.67, 0.68, 0.69, 0.7 ],
 [0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 0.77, 0.78, 0.79, 0.8 ],
 [0.81, 0.82, 0.83, 0.84, 0.85, 0.86, 0.87, 0.88, 0.89, 0.9 ],
 [0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1. ]])
```

### Create an array of 20 linearly spaced points between 0 and 1:

```
[88] : np.linspace(0, 1, 20)
```

```
[88]: array([0. , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
 0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
 0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
 0.78947368, 0.84210526, 0.89473684, 0.94736842, 1. ])
```

## 2.1 Numpy Indexing and Selection

Now you will be given a few matrices, and be asked to replicate the resulting matrix outputs.

```
[89] :
```

```
mat = np.arange(1,26).reshape(5,5)
mat
```

```
[89]: array([[ 1,  2,  3,  4,  5],
 [ 6,  7,  8,  9, 10],
 [11, 12, 13, 14, 15],
 [16, 17, 18, 19, 20],
 [21, 22, 23, 24, 25]])
```

```
[90] : # WRITE CODE HERE THAT REPRODUCES THE OUTPUT OF THE CELL BELOW
mat[2:5, 1:]
```

```
[90]: array([[12, 13, 14, 15],
 [17, 18, 19, 20],
 [22, 23, 24, 25]])
```

```
[91] : # WRITE CODE HERE THAT REPRODUCES THE OUTPUT OF THE CELL BELOW
mat[3,4]
```

```
[91]: 20
```

[92] : *# WRITE CODE HERE THAT REPRODUCES THE OUTPUT OF THE CELL BELOW*  
mat[0:3,1:2]

[92]: array([[ 2],  
[ 7],  
[12]])

[93] : *# WRITE CODE HERE THAT REPRODUCES THE OUTPUT OF THE CELL BELOW*  
mat[4,:]

[93]: array([21, 22, 23, 24, 25])

[94] : *# WRITE CODE HERE THAT REPRODUCES THE OUTPUT OF THE CELL BELOW*  
mat[3:5,:]

[94] : array([[16, 17, 18, 19, 20],  
[21, 22, 23, 24, 25]])

### 2.1.1 Now do the following

**Get the sum of all the values in mat**  
np.sum(mat)

[95] :

[95]: 325

**Get the standard deviation of the values in mat**

[96] : np.std(mat)

[96]: 7.211102550927978

**Get the sum of all the columns in mat**

[97] : np.sum(mat, axis=0)

[97]: array([55, 60, 65, 70, 75])

### 2.1.2 Conclusions:

**I have Successfully implemented and understood various Numpy Operations in this practical Successfully.**

<b>Name of Student:</b> Pushkar Prasad Sane			
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> 2	
<b>Title of Lab Assignment:</b>			
<b>Implementation of python libraries-Pandas</b>			
<b>DOP:</b> 20/1/24		<b>DOS:</b> 2 7 / 1 / 2 4	
CO: CO2	PO: PO3, PO5, PO6, PO7, PO11, PO12	Faculty signature:	

## Aim: Implementation of Python Libraries - Pandas

### Lesson 4.1 - Series in Pandas

In pandas, a Series is a one-dimensional labeled array-like object that can hold data of any type (integer, float, string, etc.). It is similar to a column in a spreadsheet or a SQL table.

A Series consists of two main components: the data itself and an index that labels each element of the data. The index can be used to select, filter, and manipulate the data in a variety of ways.

```
import pandas as Sidharth15
```

#### Creating a Series

```
a = [2, 4, 6]
Krishnanvar = Sidharth15.Series(a)
Krishnanvar

0    2
1    4
2    6
dtype: int64
```

#### Labels

##### accesssing values through index

```
Krishnanvar[1] # accesssing values through index
```

```
4
```

##### Create Labels - We can create our own labels instead of predefined index

```
a = [56, 78, 90]
Krishnanvar = Sidharth15.Series(a, index= ["a", "b", "c"])
Krishnanvar

a    56
b    78
c    90
dtype: int64
```

##### Accessing values through labels

```
Krishnanvar[
```

```
"b"] 78
```

##### Converting a list, numpy array or dictionary to a Series.

#### Numpy to Series

### Converting numpy array to Series

```
import numpy as np
Krishnanarr = np.array([11, 15, 25])
Sidharth15.Series(Krishnanarr)

0    10
1    20
2    30
dtype: int64
```

### Adding our own labels to Series

```
labels = ["r", "c", "b"]
Sidharth15.Series(Krishnanarr, labels
)

a    10
b    20
c    30
dtype: int64
```

### Converting Dictionary to Series

```
d = {'a':11, 'b':15, 'c':25}
Sidharth15.Series(d)

a    10
b    20
c    30
dtype: int64
```

### Adding our own labels to our Series

```
#creating 2 Series
ser1 = Sidharth15.Series([1,2,3,4],
index =
['Chembur', 'Mulund', 'Bhandup', 'Badlapur'
]) ser1

USA      1
Germany  2
USSR     3
Japan    4
dtype: int64

ser2 = Sidharth15.Series([1,2,5,4],
index =
['Airoli', 'Charni', 'Thane', 'Badlapur'])
ser2

USA      1
Germany  2
Italy    5
Japan    4
dtype: int64

ser1['Badlapur']
```

*Operations are then also done based on their index*

### Addition

```
ser1 + ser2
```

```
Germany    4.0
Italy      NaN
Japan     8.0
USA       2.0
USSR      NaN
dtype: float64
```

### End of Lesson 4.1

## Lesson 4.2 - Data frames in Pandas

### DataFrames

In pandas, a DataFrame is a two-dimensional labeled data structure with columns of potentially different types. It is similar to a spreadsheet or a SQL table, and it is one of the most commonly used data structures for data manipulation and analysis in Python.

A DataFrame can be created in several ways, but the most common way is by passing a dictionary of lists, where each key represents the column name, and each value represents the column data

```
#creating our own dataframe from dictionary
data = {
    "Age": [8, 9, 10],
    "height": [120, 123, 124]
}
#data gets entered column wise
#load data into a dataframe
object:
df = Sidharth15.DataFrame(data)
print(df)

   Age  height
0     8      120
1     9      123
2    10      124
```

### Locate Row

Pandas use the loc attribute to return one or more specified rows.

### Named Index

With the index argument, you can name your own indexes.

```

df = Sidharth15.DataFrame(data, index = ["Age1", "Age2", "Age3"]) df

    Age   height
Age1     8      120
Age2     9      123
Age3    10      124

import numpy as np

```

This code creates a Pandas DataFrame object with 5 rows and 4 columns, and assigns it to the variable df. The DataFrame contains random values generated from the standard normal distribution using the randn function from the NumPy library.

The index parameter specifies the row labels as 'A', 'B', 'C', 'D', and 'E', using the split() method to create a list of these labels from a string. The columns parameter specifies the column labels as 'W', 'X', 'Y', and 'Z', also using the split() method to create a list of these labels from a string.

```

from numpy.random import randn
np.random.seed(101)

df = Sidharth15.DataFrame(randn(5,4), index = 'A B C D
E'.split(),columns = 'W X Y Z'.split())
df

    W           X           Y           Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C  0.740122  0.528813 -0.589001
-2.018168
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509

```

## Selection & Indexing

### *Extract "W" column*

```

df['W']

A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64

```

### *Extract more than one column*

```

df[['W', 'Z']]

    W           Z
A  2.706850  0.503826
B  0.651118  0.605965
C  -0.589001
-2.018168

```

```
D 0.188695 0.955057
E 0.190794 0.683509
```

### SQL Syntax (Not Recommended)

```
df.W
```

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

### DataFrame columns are just Series

```
type(df['W'])
```

```
pandas.core.series.Series
```

```
s
```

### Creating a new column

#### Add 2 columns W & Y

```
df['new'] = df['W'] + df['Y']
df
```

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	-2.018168	0.740122	0.528813	-0.589001	-1.489355
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

### Removing Columns

#### axis = 1 (columns)

```
df.drop('new', axis=1)
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	0.740122	0.528813	-0.589001	
	-2.018168			
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

```
#Not inplace unless specified
```

```
df
```

	W	X	Y	Z	new
A	2.706850	0.628133	0.907969	0.503826	3.614819
B	0.651118	-0.319318	-0.848077	0.605965	-0.196959
C	0.740122	0.528813	-0.589001	-1.489355	
	-2.018168				
D	0.188695	-0.758872	-0.933237	0.955057	-0.744542
E	0.190794	1.978757	2.605967	0.683509	2.796762

*To make Permanent Changes use inplace parameter*

```
df.drop('new', axis=1,  
inplace=True) df
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	0.740122	0.528813	-0.589001	
	-2.018168			
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

### Can also drop this way

*axis=0 (row)*

```
df.drop('E', axis=0) #delete row
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057

### Selecting Rows

*select based on label*

```
df.loc['A']
```

W	2.706850
X	0.628133
Y	0.907969
Z	0.503826

Name: A, dtype: float64

### Select based on position

```
df.iloc[2]
```

W	-2.018168
X	0.740122
Y	0.528813
Z	-0.589001

Name: C, dtype: float64

### Selecting subset of rows & columns

```
df.loc['B', 'Y']
```

-0.8480769834036315

```
df.loc[['A', 'B'], ['W', 'Y']]
```

	W	Y
A	2.706850	0.907969
B	0.651118	-0.848077

## Conditional Selection

```
df
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	0.740122	0.528813	-0.589001	
	-2.018168			
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Returns True or False for columns having values greater than zero.

```
df>0
```

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

Returns actual values in the dataframe having values greater than zero.

```
df[df>0]
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

Select 'W' column but only having values greater than zero.

```
df[df['W']>0]
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

The first part of the code df[df['W']>0] selects all rows in the DataFrame df where the value in the 'W' column is greater than 0.

The second part of the code ['Y'] selects only the values in the 'Y' column of the filtered rows.

```
df[df['W']>0]['Y']
```

```
A    0.907969
B   -0.848077
D   -0.933237
E    2.605967
Name: Y, dtype: float64
```

## 1} Reset

## 2} Split

df

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	0.740122	0.528813	-0.589001	
	-2.018168			
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

*Reset to default 0,1...n index*

df.reset\_index()

	index	W	X	Y	Z
0	A	2.706850	0.628133	0.907969	0.503826
1	B	0.651118	-0.319318	-0.848077	0.605965
2	C	-2.018168	0.740122	0.528813	-0.589001
3	D	0.188695	-0.758872	-0.933237	0.955057
4	E	0.190794	1.978757	2.605967	0.683509

*Adding new columns States using split()*

method. newwind = 'CA NY WY OR CO'.split() df['States'] = newwind

df

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	CA
B	0.651118	-0.319318	-0.848077	0.605965	NY
C	0.740122	0.528813	-0.589001		WY
	-2.018168				
D	0.188695	-0.758872	-0.933237	0.955057	OR
E	0.190794	1.978757	2.605967	0.683509	CO

*changing index of dataframe to states*

df.set\_index('States')

States	W	X	Y	Z
CA	2.706850	0.628133	0.907969	0.503826
NY	0.651118	-0.319318	-0.848077	0.605965
WY	-2.018168	0.740122	0.528813	-0.589001
OR	0.188695	-0.758872	-0.933237	0.955057
CO	0.190794	1.978757	2.605967	0.683509

df

	W	X	Y	Z	States
A	2.706850	0.628133	0.907969	0.503826	C
					A
B	0.651118	-0.319318	-0.848077	0.605965	N
					Y
C	-2.018168	0.740122	0.528813	-0.589001	W
					Y

```
D  0.188695 -0.758872 -0.933237  0.955057      OR
E  0.190794  1.978757  2.605967  0.683509      CO
```

### Making the changes permanent

```
df.set_index('States', inplace =
True) df
```

```
          W         X         Y         Z
States
CA      2.706850  0.628133  0.907969  0.503826
NY      0.651118 -0.319318 -0.848077  0.605965
WY     -2.018168  0.740122  0.528813 -0.589001
OR      0.188695 -0.758872 -0.933237  0.955057
CO      0.190794  1.978757  2.605967  0.683509
```

## End of Lesson 4.2

### Handling Missing data - Lesson 4.3

```
import numpy as npSidharth15
```

#### *Creating a dataframe with missing values*

```
df = Sidharth15.DataFrame({'A':[1,2,np.nan],'B':[5,np.nan,8],'C':
[1,2,3]})
df
```

```
       A      B      C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  8.0  3
```

#### *checking for null values in the dataframe*

```
df.isnull()
```

```
       A      B      C
0  False  False  False
1  False   True  False
2   True  False  False
```

#### *checking for notnull() values in the dataframe.*

```
df.notnull()
```

```
       A      B      C
0   True   True  True
1   True  False  True
2  False   True  True
```

#### *dropna() - Remove all rows with NA values from the DataFrame*

```
df.dropna()
```

```
       A      B      C
0  1.0  5.0  1
```

### *Making the changes permanent*

```
df.dropna(axis=1, inplace=True) df

      C
0  1
1  2
2  3

df1 = Sidharth15.DataFrame({ 'A': [1, 2, np.nan], 'B': [5, np.nan, np.nan], 'C': [1, 2, 3] }) df1

      A      B  C
0  1.0  5.0  1
1  2.0  NaN  2
2  NaN  NaN  3
```

### **Keep only the rows having 2 or more valid data**

```
df1.dropna(axis=0, thresh=2)
```

```
      A      B  C
0  1.0  5.0  1
1  2.0  NaN  2
```

### **fillna() - Replace null values with the specified value**

```
df1.fillna(value=-99)
```

```
      A      B  C
0  1.0  5.0  1
1  2.0 -99.0  2
2 -99.0 -99.0  3
```

### *Fill only for a particular column*

```
df1['A'].fillna(value=df['A'].mean())
```

*The fillna() method is used to fill in missing values, and the argument method='pad' specifies that we want to use the previous valid value to fill in missing values.*

```
df1.fillna(method='pad')
```

```
      A      B  C
0  1.0  5.0  1
1  2.0  5.0  2
2  2.0  5.0  3
```

```
df2 = Sidharth15.DataFrame({ 'A': [np.nan, 2, 3], 'B': [5, np.nan, 7], 'C': [1, 2, 3] })
df2
```

```
      A      B  C
0  NaN  5.0  1
```

```
1 2.0 NaN 2
2 3.0 7.0 3
```

*The `fillna()` method is used to fill in missing values, and the argument `method='bfill'` specifies that we want to use the next valid value to fill in missing values.*

```
df2.fillna(method='bfill')
```

```
      A      B  C
0  2.0  5.0  1
1  2.0  7.0  2
2  3.0  7.0  3
```

*Replace nan values with -99*

```
df2.replace(to_replace = np.nan, value=-99)
```

```
      A      B  C
0 -99.0   5.0  1
1   2.0 -99.0  2
2   3.0   7.0  3
```

```
df2.replace(to_replace = np.nan, value=-99,
inplace=True) df2
```

```
      A      B  C
0 -99.0   5.0  1
1   2.0 -99.0  2
2   3.0   7.0  3
```

*Interpolation: Interpolation in pandas is a technique for filling in missing values in a DataFrame or Series by estimating the values that should be there based on the values that are present.*

```
df3 = Sidharth15.DataFrame({'A':[1,2,npSidharth15.nan], 'B':
[5,npSidharth15.nan,8], 'C':[1,npSidharth15.nan,3], 'D':
[4,5,npSidharth15.nan]})
```

```
df3
```

```
      A      B      C      D
0  1.0  5.0  1.0  4.0
1  2.0  NaN  NaN  5.0
2  NaN  8.0  3.0  NaN
```

*#to interpolate the missing values with mean*

```
df3.interpolate(method='linear', limit_direction='forward')
```

```
      A      B      C      D
0  1.0  5.0  1.0  4.0
1  2.0  6.5  2.0  5.0
2  2.0  8.0  3.0  5.0
```

## End of Lesson 4.3

## GroupBy with Pandas | Merge with Pandas | Joins with Pandas | Concatenation (Lesson 4.4)

In Pandas, the groupby() function is used to group a DataFrame by one or more columns, and apply a function to each group separately. The groupby() function returns a DataFrameGroupBy object, which can be used to perform various aggregation functions on the groups, such as sum(), mean(), min(), max(), etc.

```
import pandas as Sidharth15

data = {
    'Company': ['Parle', 'Parle', 'Godrej', 'Godrej', 'Lakme', 'Lakme'],
    'Person': ['Rakesh', 'Suresh', 'Mahesh', 'Kalpesh', 'Yogesh', 'Hitesh'],
    'Sales': [200, 120, 340, 124, 243, 350]
}

df = Sidharth15.DataFrame(data) df

   Company    Person  Sales
0    Parle     Rakesh    200
1    Parle     Suresh    120
2   Godrej     Mahesh    340
3   Godrej     Kalpesh    124
4   Lakme     Yogesh    243
5   Lakme     Hitesh    350

by_comp = df.groupby('Company')

<pandas.core.groupby.generic.DataFrameGroupBy object at
0x7f0086175be0>

first() - Displays the First Occurrence of each group.
by_comp.first()

          Person  Sales
Company
Godrej     Mahesh    340
Lakme     Yogesh    243
Parle     Rakesh    200
```

## Merging, Joining, Concatenating

### Create DataFrame with the name of the students

```
df1 = Sidharth15.DataFrame({
    'A': ['Krishnan', 'Rakesh', 'Rahul', 'Rohit'],
    'B': ['Akshay', 'Dev', 'Nitish', 'Prathan'],
    'C': ['Prathamesh', 'Sidharth', 'Yash', 'Finny'],
    'D': ['Omkar', 'Kapil', 'krishna', 'Vira
t'], index=[0, 1, 2, 3]
)
df1
```

```

          A           B           C           D
0   Sunny     Akshay  Prathamesh      Omkar
1   Rakesh        Dev  Sidharth       Kapil
2   Rahul     Nitish        Yash krishna
3   Rohit    Prathan       Finny      Virat

df2 = Sidharth15.DataFrame({
    'A': ['Sheetal', 'Laksmi', 'Rupa', 'Malvika'],
    'B': ['Pooja', 'Archana', 'Seeta', 'Siya'],
    'C': ['Krisha', 'Avisha', 'Mohini', 'Sakshi'],
    'D': ['Diya', 'Shivani', 'Riya', 'Priya']
}, index=[4, 5, 6, 7]
)
df2

          A           B           C           D
4  Sheetal     Pooja   Krisha      Diya
5  Laksmi    Archana  Avisha  Shivani
6    Rupa      Seeta   Mohini      Riya
7  Malvika      Siya   Sakshi      Priya

df3 = Sidharth15.DataFrame({
    'A': ['Raj', 'Rutik', 'Pranita', 'lia'],
    'B': ['Pia', 'Tia', 'Narendra', 'Gauri'],
    'C': ['Ganesh', 'Shiva', 'Durga', 'Shakti'],
    'D': ['Kartik', 'Ajay', 'Ram', 'Shahrukh']
}, index=[8, 9, 10, 11]
)
df3

```

	A	B	C	D
8	Raj	Pia	Ganesh	Kartik
9	Rutik	Tia	Shiva	Ajay
10	Pranita	Narendra	Durga	Ram
11	lia	Gauri	Shakti	Shahrukh

**Concatenation: Concatenation basically glues together DataFrames. Keep in mind that dimensions should match axis you are concatenating on.**

Concatenation in Pandas is the process of combining two or more data structures (such as Series or DataFrames) into a single one. It is done using the concat() function, which allows you to concatenate along rows or columns, and also specify how to handle missing data. By default, concatenation is done based on index values, but you can also specify the axis to concatenate along and set new indices if needed.

```
Sidharth15.concat([df1, df2, df3])

          A           B           C           D
0   Sunny     Akshay  Prathamesh      Omkar
1   Rakesh        Dev  Sidharth       Kapil
2   Rahul     Nitish        Yash krishna
3   Rohit    Prathan       Finny      Virat
4  Sheetal     Pooja       Krisha      Diya
```

```

5      Laksmi   Archana      Avisha   Shivani
6        Rupa    Seeta       Mohini   Riya
7     Malvika   Siya       Sakshi   Priya
8       Raj     Pia       Ganesh   Kartik
9     Rutik    Tia       Shiva    Ajay
10    Pranita Narendra     Durga   Ram
11      lia     Gauri     Shakti Shahrukh

```

### Concat column wise use axis=1

```
Sidharth15.concat([df1,df2,df3],axis=1)
```

	A	B	C	D	A	B	C
D \							
0	Sunny	Akshay	Prathamesh	Omkar	NaN	NaN	
Nan	Nan						
1	Rakesh	Dev	Sidharth	Kapil	NaN	NaN	
Nan	Nan						
2	Rahul	Nitish		Yash krishna	NaN	NaN	
Nan	Nan						
3	Rohit	Prathan		Finny	Virat	NaN	
Nan	Nan						
4	NaN		NaN		NaN	Sheetal	Pooja
Krishna	Diya						
5	NaN		NaN		NaN	Laksmi	Archana
Avisha	Shivani						
6	NaN		NaN		NaN	Rupa	Seeta
Mohini	Riya						
7	NaN		NaN		NaN	Malvika	Siya
Sakshi	Priya						
8	NaN		NaN		NaN	NaN	NaN
Nan	Nan						
9	NaN		NaN		NaN	NaN	NaN
Nan	Nan						
10	NaN		NaN		NaN	NaN	NaN
Nan	Nan						
11	NaN		NaN		NaN	NaN	NaN
Nan	Nan						

	A	B	C	D
0	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN
8	Raj	Pia Ganesh		Kartik

```

9      Rutik      Tia    Shiva      Ajay
10     Pranita   Narendra   Durga      Ram
11       lia      Gauri  Shakti  Shahrukh

left = Sidharth15.DataFrame({
    'key': ['K0', 'K1', 'K2', 'K3'],
    'A': ['Akshay', 'Dev', 'Nitish', 'Prathan'],
    'B': ['Prathamesh', 'Sidharth', 'Yash', 'Finny']})

right = Sidharth15.DataFrame({
    'key': ['K0', 'K1', 'K2', 'K3'],
    'C': ['Pooja', 'Archana', 'Sakshi', 'Nirma'],
    'D': ['Hema', 'Rekha', 'Jaya', 'Sushma']})

left

  key      A          B
0 K0    Akshay  Prathamesh
1 K1      Dev      Sidharth
2 K2    Nitish        Yash
3 K3    Prathan      Finny

right

  key      C          D
0 K0    Pooja      Hema
1 K1  Archana      Rekha
2 K2    Sakshi      Jaya
3 K3    Nirma      Sushma

```

## Merge

*The Merge function allows you to merge DataFrames together using a similar logic as merging together.*

Merge in Pandas is a function used to combine two or more DataFrames based on one or more common columns (keys). It allows you to perform different types of joins, such as inner, outer, left, and right, and specify how to handle missing data and duplicates. Merge can also be performed on indices instead of columns by using the `left_index` and `right_index` parameters.

```
Sidharth15.merge(left,right,how='inner',on='key')

  key      A          B          C          D
0 K0    Akshay  Prathamesh  Pooja      Hema
1 K1      Dev      Sidharth  Archana      Rekha
2 K2    Nitish        Yash  Sakshi      Jaya
3 K3    Prathan      Finny  Nirma      Sushma

left = Sidharth15.DataFrame({
    'key1': ['K0', 'K0', 'K1', 'K2'],
    'key2': ['K0', 'K1', 'K0', 'K1'],
```

```

'A':[ 'Akshay', 'Dev', 'Nitish', 'Prathan'],
'B':[ 'Prathamesh', 'Sidharth', 'Yash', 'Finny'] }))

right = Sidharth15.DataFrame({ 'key1': ['K0', 'K1', 'K1', 'K2'],
                               'key2': ['K0', 'K0', 'K0', 'K0'],
                               'C': ['Pooja', 'Archana', 'Sakshi', 'Nirma'],
                               'D': ['Hema', 'Rekha', 'Jaya', 'Sushma'] })

```

left

	key1	key2	A	B
0	K0	K0	Akshay	Prathamesh
1	K0	K1	Dev	Sidharth
2	K1	K0	Nitish	Yash
3	K2	K1	Prathan	Finny

right

	key1	key2	C	D
0	K0	K0	Pooja	Hema
1	K1	K0	Archana	Rekha
2	K1	K0	Sakshi	Jaya
3	K2	K0	Nirma	Sushma

The resulting DataFrame will have all the columns from both the left and right DataFrames, with rows matched based on the values in the 'key1' and 'key2' columns. Any rows that do not have matching keys in both DataFrames will be dropped from the result.

```
Sidharth15.merge(left,right, on=['key1','key2'])
```

	key1	key2	A	B	C	D
0	K0	K0	Akshay	Prathamesh	Pooja	Hema
1	K1	K0	Nitish	Yash	Archana	Rekha
2	K1	K0	Nitish	Yash	Sakshi	Jaya

### *Outer join - Joining matching & non-matching rows from both the table*

```
Sidharth15.merge(left,right,how='outer' ,on=['key1','key2'])
```

	key1	key2	A	B	C	D
0	K0	K0	Akshay	Prathamesh	Pooja	Hema
1	K0	K1	Dev	Sidharth	NaN	NaN
2	K1	K0	Nitish	Yash	Archana	Rekha
3	K1	K0	Nitish	Yash	Sakshi	Jaya
4	K2	K1	Prathan	Finny	NaN	NaN
5	K2	K0	NaN	NaN	Nirma	Sushma

### *Right join - Matching & Non-Matching rows of Right table and matching rows of lef table.*

```
Sidharth15.merge(left,right,how='right' ,on=['key1','key2'])
```

```

key1  key2      A          B          C          D
0     K0    K0  Akshay  Prathamesh  Pooja   Hema
1     K1    K0  Nitish        Yash  Archana  Rekha
2     K1    K0  Nitish        Yash  Sakshi   Jaya
3     K2    K0      NaN        NaN  Nirma   Sushma

```

### Lek Join - Matching & Non-Matching rows of Lek table and matching rows of Right table.

```
Sidharth15.merge(left,right,how='left' ,on=['key1','key2'])
```

```

key1  key2      A          B          C          D
0     K0    K0  Akshay  Prathamesh  Pooja   Hema
1     K0    K1      Dev    Sidharth      NaN   NaN
2     K1    K0  Nitish        Yash  Archana  Rekha
3     K1    K0  Nitish        Yash  Sakshi   Jaya
4     K2    K1  Prathan      Finny      NaN   NaN

```

### Joining

Joining in Pandas is a method used to combine two or more DataFrames based on their indices, rather than their columns. It is similar to merge, but the main difference is that join only considers the indices, while merge considers both indices and columns. Joining is done using the join() method, which allows you to perform inner, outer, left, and right joins, and also specify how to handle missing data.

```

left = Sidharth15.DataFrame({
    'A': ['Krishnan', 'Rakesh', 'Rahul'],
    'B': ['Akshay', 'Dev', 'Nitish']},
    index=['K0', 'K1', 'K2']
)

right = Sidharth15.DataFrame({
    'C': ['Krisha', 'Avisha', 'Mohini'],
    'D': ['Diya', 'Shivani', 'Riya']},
    index=['K0', 'K2', 'K3'])

```

```
left
```

	A	B
K0	Sunny	Akshay
K1	Rakesh	Dev
K2	Rahul	Nitish

```
right
```

	C	D
K0	Krishna	Diya
K2	Avisha	Shivani
K3	Mohini	Riya

### Display All rows of lek table and Only Matching rows of right table.

```
left.join(right)
```

	A	B	C	D
K0	Sunny	Akshay	Krishna	Diya
K1	Rakesh	Dev	NaN	NaN
K2	Rahul	Nitish	Avisha	Shivani

*Display all the rows of both Lek & Right Table.*

```
left.join(right, how='outer')
```

	A	B	C	D
K0	Sunny	Akshay	Krishna	Diya
K1	Rakesh	Dev	NaN	NaN
K2	Rahul	Nitish	Avisha	Shivani
K3	NaN	NaN	Mohini	Riya

## Lesson 4.5 - Exploratory Data

### Analysis What is EDA?

EDA (Exploratory Data Analysis) is an approach used in data science to analyze and summarize the main characteristics of a dataset. It is a preliminary step in data analysis that helps data analysts to understand the structure, features, and patterns of the data before applying any statistical or machine learning models to it.

EDA involves various techniques and methods to explore and visualize data. The primary goal of EDA is to get insights and find patterns in the data that can be used to inform decisions and drive further analyses. Some common techniques used in EDA include:

**Descriptive statistics:** EDA often starts with descriptive statistics, such as measures of central tendency, variability, and distribution. These statistics provide a summary of the key characteristics of the data.

**Data visualization:** EDA often involves creating visual representations of the data, such as histograms, scatterplots, and box plots. These visualizations can reveal patterns, outliers, and relationships in the data that might not be apparent from the descriptive statistics alone.

**Data cleaning:** EDA often includes data cleaning, which involves identifying and handling missing or invalid data, dealing with outliers and anomalies, and transforming variables as needed.

**Feature engineering:** EDA can also involve feature engineering, which is the process of creating new variables or features that can better capture the information in the data or make it more suitable for the modeling process.

### Types Of EDA.

There are several types of EDA (Exploratory Data Analysis) that can be used to analyze and summarize a dataset, each with its own strengths and weaknesses. Some common types of EDA include:

**Univariate Analysis:** This type of EDA involves analyzing a single variable at a time. It focuses on summarizing the distribution, central tendency, variability, and other properties of the variable using measures such as histograms, frequency tables, and summary statistics.

**Bivariate Analysis:** This type of EDA involves analyzing the relationship between two variables. It is useful for identifying patterns, correlations, and other relationships between variables that can be used to inform modeling or further analysis. Common techniques used in bivariate analysis include scatterplots, correlation matrices, and contingency tables.

**Multivariate Analysis:** This type of EDA involves analyzing more than two variables at a time. It is useful for identifying complex relationships and patterns between multiple variables, and can help to identify potential interactions or confounding variables that may be important to consider in further analysis. Common techniques used in multivariate analysis include principal component analysis (PCA), cluster analysis, and factor analysis.

**Time Series Analysis:** This type of EDA involves analyzing data that changes over time, such as stock prices or weather patterns. Time series analysis involves techniques such as trend analysis, seasonal decomposition, and autoregression to identify patterns, trends, and relationships in the data over time.

### Functions used in EDA

**describe():** This function is used to generate summary statistics of a DataFrame, including count, mean, standard deviation, minimum, and maximum values, and percentiles.

**info():** This function provides information about the DataFrame, such as the number of rows and columns, the data types of each column, and whether there are any missing values.

**isnull():** This function is used to identify missing values in a DataFrame. It returns a Boolean value for each cell, indicating whether the value is missing (True) or not (False).

**value\_counts():** This function is used to count the number of occurrences of each unique value in a column. It is often used to identify the frequency distribution of categorical variables.

**dropna():** This function is used to remove rows or columns that contain missing values from a DataFrame.

**fillna():** This function is used to fill in missing values in a DataFrame. It can be used to replace missing values with a specific value or to interpolate missing values based on the surrounding values.

**head():** Display First five rows of

DataFrame

**tail():** Display last five rows of

DataFrame.

`unique()`: is a pandas function that returns an array of unique values in a pandas object, such as a DataFrame or Series. It can be used to identify the unique values in a column and to count the number of unique values in a pandas object.

`nunique()`: is a pandas function that returns the number of unique values in a pandas object, such as a DataFrame or Series. It can be used to count the number of unique values in a column and to calculate the number of unique values for each column in a DataFrame.

`apply()`: is a pandas function that can be used to apply a function to each element or row of a pandas object, such as a DataFrame or Series. It is a powerful tool that can be used for data cleaning, data transformation, and feature engineering. By applying a function to each element or row, you can efficiently manipulate and process large datasets.

`sort_values()`: is a pandas function that can be used to sort a DataFrame or Series by one or more columns. By default, the function sorts the data in ascending order, but you can specify descending order by setting the "ascending" parameter to False

### Data Skewness with Example.

Data skewness refers to the asymmetry in the distribution of a dataset. Skewness measures the deviation of the distribution from the normal distribution. A dataset can be positively skewed, negatively skewed, or symmetric.

For example, consider a dataset that represents the salaries of employees in a company. If the distribution of salaries is positively skewed, it means that the majority of employees have salaries below the mean, and there are a few employees with high salaries that pull the mean to the right. In contrast, if the distribution is negatively skewed, it means that the majority of employees have salaries above the mean, and there are a few employees with low salaries that pull the mean to the left.

A symmetric distribution, on the other hand, has equal numbers of observations on both sides of the mean. Skewness can have a significant impact on statistical analyses, such as regression analysis, hypothesis testing, and confidence interval estimation, as it can affect the validity of assumptions about the data. Therefore, it is important to examine the skewness of a dataset before conducting any statistical analysis.

```
import pandas as Sidharth15
Sidharth15 = Sidharth15.DataFrame({'col1':[1,2,4,5,6],'col2':
[444,555,666,444,777],'col3':['abc','def','aghi','xyz','ghj']})
Sidharth15.head()

   col1  col2  col3
0     1    444    abc
1     2    555    def
2     4    666   aghi
3     5    444    xyz
4     6    777    ghj
```

### Info on unique values

```
Sidharth15.info()
```

```

<class
'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 3 columns):
 #   Column Non-Null Count Dtype  
----  -- 
 0   col1    5 non-null      int64  
 1   col2    5 non-null      int64  
 2   col3    5 non-null      object  
dtypes: int64(2), object(1) memory usage: 248.0+ bytes

Sidharth15['col2'].unique()
array([444, 555,
       666, 777])

Sidharth15['col2'].nunique() 4

Sidharth15['col2'].value_counts()
444    2
555    1
666    1
777    1
Name: col2, dtype: int64

print(Sidharth15.head(2))
   col1  col2  col3
0     1    444  abc
1     2    555  def

Applying Functions
Sidharth15['col3'].apply(len) #get string len of 3rd column
0    3
1    3
2    4
3    3
4    3
Name: col3, dtype: int64

Total of col1
Sidharth15['col1'].sum() 18

Median of col1
Sidharth15['col1'].median() 4.0

```

```
#Mean of col1
Sidharth15['col1'].me
an() 3.6
```

### *Permanently removing a column*

```
del Sidharth15['col1']

Sidharth15
```

```
    col1  col2  col3
0      1    444    abc
1      2    555    def
2      4    666  aghi
3      5    444    xyz
4      6    777    ghj
```

### *Get Columns & Index names*

```
Sidharth15.columns
Index(['col1', 'col2', 'col3'], dtype='object')

Sidharth15.index
RangeIndex(start=0, stop=5, step=1)
```

### *Sorting & Ordering a DataFrame*

```
Sidharth15
```

```
    col1  col2  col3
0      1    444    abc
1      2    555    def
2      4    666  aghi
3      5    444    xyz
4      6    777    ghj
```

```
Sidharth15.sort_values(by='col3') #inplace=False by default
    col1  col2  col3
0      1    444    abc
2      4    666  aghi
1      2    555    def
4      6    777    ghj
3      5    444    xyz
```

```
Sidharth15
```

```
    col1  col2  col3
0      1    444    abc
1      2    555    def
2      4    666  aghi
3      5    444    xyz
4      6    777    ghj
```

### Data Skewness

```
Sidharth15.describe() # describe only numeric columns
```

	col1	col2
count	5.000000	5.000000
mean	3.600000	577.200000
std	2.073644	144.726293
min	1.000000	444.000000
25%	2.000000	444.000000
50%	4.000000	555.000000
75%	5.000000	666.000000
max	6.000000	777.000000

```
Sidharth15.describe().transpose()
```

	count	std	min	25%	50%	75%	max
	mean						
col1	5.0	3.6	2.073644	1.0	2.0	4.0	5.0
col2	5.0	577.2	144.726293	444.0	444.0	555.0	666.0
							777.0

```
Sidharth15
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	4	666	aghi
3	5	444	xyz
4	6	777	ghj

```
#Get Minimum Values from each column
```

```
Sidharth15.min()
```

col1	1
col2	444
col3	
abc	dtype:
object	

```
#Get Maximum Values from each column
```

```
Sidharth15.max()
```

col1	6
col2	777
col3	
xyz	dtype:
object	

```
# count number of elements in each column
```

```
Sidharth15.count()
```

col1	5
col2	5
col3	5
	dtype: int64

## Pandas Exercise -1

We will be using the [SF Salaries Dataset](#) from Kaggle!

\*\* Import pandas as pd.\*\*

```
import pandas as pd
import io
```

\*\* Read Salaries.csv as a dataframe called sal.\*\*

```
from google.colab import files
uploaded = files.upload()

<IPython.core.display.HTML
object> Saving Salaries.csv to

Salaries.csv

Krishnan = pd.read_csv(io.BytesIO(uploaded['Salaries.csv']))

<ipython-input-54-eb59c4012eab>:1: DtypeWarning: Columns
(3,4,5,6,12) have mixed types. Specify dtype option on import or
set low_memory=False.
sunny = pd.read_csv(io.BytesIO(uploaded['Salaries.csv']))
```

\*\* Check the head of the DataFrame. \*\*

```
Krishnan.head()
```

```
   Id
EmployeeName
JobTitle \
0    1    NATHANIEL FORD GENERAL MANAGER-METROPOLITAN
TRANSIT AUTHORITY
1    2    GARY JIMENEZ                      CAPTAIN III
(POLICE DEPARTMENT)
2    3    ALBERT PARDINI                     CAPTAIN III
(POLICE DEPARTMENT)
3    4    CHRISTOPHER CHONG                 WIRE ROPE CABLE
MAINTENANCE MECHANIC
4    5    PATRICK GARDNER      DEPUTY CHIEF OF
DEPARTMENT, (FIRE DEPARTMENT)
```

	BasePay	OvertimePay	OtherPay	Benefits	
TotalPay	TotalPay	Benefits	\		
0	167411.18	0.0	400184.25	NaN	567595.43
	567595.43				
1	155966.02	245131.88	137811.38	NaN	538909.28
	538909.28				
2	212739.13	106088.18	16452.6	NaN	335279.91
	335279.91				
3	77916.0	56120.71	198306.9	NaN	332343.61
	332343.61				

```
4    134401.6      9737.0 182234.59      NaN 326373.19
326373.19
```

```
   Year  Notes      Agency Status
0 2011    NaN  San Francisco    NaN
1 2011    NaN  San Francisco    NaN
2 2011    NaN  San Francisco    NaN
3 2011    NaN  San Francisco    NaN
4 2011    NaN  San Francisco    NaN
```

\*\* Use the .info() method to find out how many entries there are.\*\*

```
Krishnan.info()
```

```
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 148654 entries, 0 to
148653 Data columns (total 13
columns):
 #   Column            Non-Null Count   Dtype  
--- 
 0   Id                148654 non-null    int64  
 1   EmployeeName       148654 non-null    object 
 2   JobTitle           148654 non-null    object 
 3   BasePay            148049 non-null    object 
 4   OvertimePay        148654 non-null    object 
 5   OtherPay           148654 non-null    object 
 6   Benefits           112495 non-null    object 
 7   TotalPay           148654 non-null    float64
 8   TotalPayBenefits  148654 non-null    float64
 9   Year               148654 non-null    int64  
 10  Notes              0 non-null        float64
 11  Agency              148654 non-null    object 
 12  Status              38119 non-null    object 
object dtypes: float64(3), int64(2),
object(8)
memory usage: 14.7+ MB
```

```
# replace 'Not Provided' with NaN in the DataFrame
import numpy as np
Krishnan.replace('Not Provided', np.nan,
inplace=True) Krishnan.replace('Not provided',
np.nan, inplace=True) Krishnan.head()
```

```
   Id      EmployeeName
JobTitle \
0   1      NATHANIEL FORD GENERAL MANAGER-METROPOLITAN
TRANSIT AUTHORITY
1   2      GARY JIMENEZ          CAPTAIN III
(POLICE DEPARTMENT)
2   3      ALBERT PARDINI        CAPTAIN III
(POLICE DEPARTMENT)
3   4      CHRISTOPHER CHONG    WIRE ROPE CABLE MAINTENANCE
```

MECHANIC

4 5 PATRICK GARDNER DEPUTY CHIEF OF  
DEPARTMENT, (FIRE DEPARTMENT)

	BasePay	OvertimePay	OtherPay	Benefits	
TotalPay	TotalPay	Benefits	\		
0	167411.18	0.0	400184.25	NaN	567595.43
	567595.43				
1	155966.02	245131.88	137811.38	NaN	538909.28
	538909.28				
2	212739.13	106088.18	16452.6	NaN	335279.91
	335279.91				
3	77916.0	56120.71	198306.9	NaN	332343.61
	332343.61				
4	134401.6	9737.0	182234.59	NaN	326373.19
	326373.19				

Year	Notes	Agency	Status
0	2011	NaN	San Francisco
1	2011	NaN	San Francisco
2	2011	NaN	San Francisco
3	2011	NaN	San Francisco
4	2011	NaN	San Francisco

Krishnan.tail()

	Id	EmployeeName	JobTitle
BasePay	OvertimePay	\	
148649	148650	Roy I Tillery	Custodian
0.00			0.00
148650	148651	NaN	NaN
NaN			NaN
148651	148652	NaN	NaN
NaN			NaN
148652	148653	NaN	NaN
NaN			NaN
148653	148654	Joe Lopez Counselor, Log Cabin Ranch	0.00
0.00			

	OtherPa	Benefits	TotalPay	TotalPay	Benefits	Year	Notes	\
	y							
148649	0.00	0.00	0.00		0.00	2014	NaN	
148650	NaN	NaN	0.00		0.00	2014	NaN	
148651	NaN	NaN	0.00		0.00	2014	NaN	
148652	NaN	NaN	0.00		0.00	2014	NaN	
148653	-618.13	0.00	-618.13		-618.13	2014	NaN	

	Agency
Status	148649 San
Francisco	PT 148650 San
Francisco	NaN 148651 San
Francisco	NaN

```
148652 San Francisco  
NaN 148653 San Francisco
```

```
PT
```

```
#Count number of NAN in DataFrame  
Krishnan.isna().sum()
```

```
Id          0  
EmployeeName 6  
JobTitle     4  
BasePay      609  
OvertimePay   4  
OtherPay      4  
Benefits     36163  
TotalPay      0  
TotalPayBenefits 0  
Year          0  
Notes         148654  
Agency        0  
Status        110535  
dtype: int64
```

```
Krishnan.hea
```

```
d()  
)
```

```
Id  
EmployeeName  
JobTitle \  
0    1      NATHANIEL FORD GENERAL MANAGER-METROPOLITAN  
TRANSIT AUTHORITY  
1    2      GARY JIMENEZ                      CAPTAIN III  
(POLICE DEPARTMENT)  
2    3      ALBERT PARDINI                     CAPTAIN III  
(POLICE DEPARTMENT)  
3    4      CHRISTOPHER CHONG                 WIRE ROPE CABLE  
MAINTENANCE MECHANIC  
4    5      PATRICK GARDNER      DEPUTY CHIEF OF  
DEPARTMENT, (FIRE DEPARTMENT)
```

```
BasePay OvertimePay OtherPay Benefits  
TotalPay TotalPayBenefits \  
0 167411.18      0.0  400184.25      NaN  567595.43  
567595.43  
1 155966.02      245131.88 137811.38      NaN  538909.28  
538909.28  
2 212739.13      106088.18 16452.6       NaN  335279.91  
335279.91  
3 77916.0        56120.71  198306.9      NaN  332343.61  
332343.61  
4 134401.6       9737.0   182234.59      NaN  326373.19  
326373.19  
0 2011 San Francisco      NaN
```

Year	Agency	Status
------	--------	--------

0	2011 San Francisco	NaN
---	--------------------	-----

```

1 2011 San Francisco     NaN
2 2011 San Francisco     NaN
3 2011 San Francisco     NaN
4 2011 San Francisco     NaN

Krishnan.info()

<class
'pandas.core.frame.DataFrame'>
RangeIndex: 148654 entries, 0 to
148653 Data columns (total 12
columns):
 #   Column           Non-Null Count   Dtype  
--- 
 0   Id               148654 non-null    int64  
 1   EmployeeName     148648 non-null    object  
 2   JobTitle         148650 non-null    object  
 3   BasePay          148045 non-null    object  
 4   OvertimePay      148650 non-null    object  
 5   OtherPay         148650 non-null    object  
 6   Benefits          112491 non-null    object  
 7   TotalPay         148654 non-null    float64
 8   TotalPayBenefits 148654 non-null    float64
 9   Year              148654 non-null    int64  
 10  Agency            148654 non-null    object  
 11  Status             38119 non-null   object  
object dtypes: float64(2), int64(2),
object(8)
memory usage: 13.6+ MB

Krishnan.head()

   Id
EmployeeName
JobTitle \
0   1      NATHANIEL FORD GENERAL MANAGER-METROPOLITAN
TRANSIT AUTHORITY
1   2      GARY JIMENEZ                      CAPTAIN III
(POLICE DEPARTMENT)
2   3      ALBERT PARDINI                     CAPTAIN III
(POLICE DEPARTMENT)
3   4      CHRISTOPHER CHONG                 WIRE ROPE CABLE
MAINTENANCE MECHANIC
4   5      PATRICK GARDNER      DEPUTY CHIEF OF
DEPARTMENT, (FIRE DEPARTMENT)

   BasePay  OvertimePay  OtherPay  Benefits
TotalPay  TotalPayBenefits \
0  167411.18        0.0  400184.25      NaN  567595.43
567595.43
1  155966.02        245131.88  137811.38      NaN  538909.28
538909.28
2  212739.13        106088.18   16452.6       NaN  335279.91
335279.91
3   77916.0          56120.71   198306.9      NaN  332343.61

```

```
332343.61
4    134401.6      9737.0 182234.59          NaN 326373.19
326373.19
```

```
Year      Agency Status
0  2011  San Francisco    NaN
1  2011  San Francisco    NaN
2  2011  San Francisco    NaN
3  2011  San Francisco    NaN
4  2011  San Francisco    NaN
```

What is the average BasePay ?

```
Krishnan['BasePay'] = Krishnan['BasePay'].astype(float)
Krishnan['BasePay'].mean()
```

```
66325.4488404877
```

\*\* What is the highest amount of OvertimePay in the dataset ? \*\*

```
Krishnan['OvertimePay'] = Krishnan['OvertimePay'].astype(float)
Krishnan['OvertimePay'].max()
```

```
245131.88
```

\*\* What is the job title of GARY JIMENEZ ? Note: Use all caps, otherwise you may get an answer that doesn't match up (there is also a lowercase Joseph Driscoll). \*\*

```
Krishnan[Krishnan['EmployeeName']=='JOSEPH DRISCOLL']['JobTitle']

24    CAPTAIN, FIRE SUPPRESSION
Name: JobTitle, dtype: object
```

\*\* How much does GARY JIMENEZ make (including benefits)? \*\*

```
Krishnan[Krishnan['EmployeeName']=='JOSEPH
DRISCOLL']['TotalPayBenefits']
```

```
24    270324.91
Name: TotalPayBenefits, dtype: float64
```

\*\* What is the name of highest paid person (including benefits)? \*\*

```
ind = Krishnan['TotalPayBenefits'].idxmax()
Krishnan.loc[ind]['EmployeeName']

{"type": "string"}
```

\*\* What is the name of lowest paid person (including benefits)? Do you notice something strange about how much he or she is paid?\*\*

```
ind =
Krishnan['TotalPayBenefits'].idxmin()
Krishnan.iloc[ind]
```

```
Id                               148654
EmployeeName                    Joe Lopez
JobTitle                        Counselor, Log Cabin Ranch
BasePay                          0.0
OvertimePay                     0.0
OtherPay                         -618.13
Benefits                         0.00
TotalPay                         -618.13
TotalPayBenefits                 -618.13
Year                            2014
Agency                           San Francisco
Status                           PT
Name: 148653, dtype: object
```

\*\* What was the average (mean) BasePay of all employees per year? (2011-2014) ? \*\*

```
Krishnan.groupby('Year').mean()['BasePay']
```

```
<ipython-input-77-782e1b60e15a>:1: FutureWarning: The default
value of numeric_only in DataFrameGroupBy.mean is deprecated. In a
future version, numeric_only will default to False. Either specify
numeric_only or select only columns which should be valid for the
function.
```

```
2011      63595.956517
2012      65436.406857
2013      69630.030216
2014      66564.421924
Name: BasePay, dtype:
float64
sunny.groupby('Year').mean()['BaseP
ay']  Year
```

\*\* How many unique job titles are there? \*\*

```
Krishnan['JobTitle'].nun
iqu e() 2158
```

\*\* What are the top 5 most common jobs? \*\*

```
Krishnan['JobTitle'].value_counts().head()

Transit Operator                7036
Special Nurse                   4389
Registered Nurse                3736
Public Svc Works                2518
Aide-Public
Police Officer 3                 2421
Name: JobTitle,      int64
dtype:
```

\*\* How many Job Titles were represented by only one person in 2013? (e.g. Job Titles with only one occurrence in 2013?) \*\*

```
(Krishnan[Krishnan['Year']==2013]['JobTitle'].value_counts()==1).sum()  
()
```

202

```
Krishnan.info()

<class
'pandas.core.frame.DataFrame'>
RangeIndex: 148654 entries, 0 to
148653 Data columns (total 12
columns):
 #   Column            Non-Null Count   Dtype  
--- 
 0   Id                148654 non-null    int64  
 1   EmployeeName      148648 non-null    object  
 2   JobTitle          148650 non-null    object  
 3   BasePay           148045 non-null    float64 
 4   OvertimePay       148650 non-null    float64 
 5   OtherPay          148650 non-null    object  
 6   Benefits          112491 non-null    object  
 7   TotalPay          148654 non-null    float64 
 8   TotalPayBenefits  148654 non-null    float64 
 9   Year              148654 non-null    int64  
 10  Agency             148654 non-null    object  
 11  Status             38119 non-null   object  
object dtypes: float64(4), int64(2),
object(6)
memory usage: 13.6+ MB
```

\*\* How many people have the word Chief in their job title? (This is pretty tricky) \*\*

```
Krishnan['JobTitle'] = Krishnan['JobTitle'].astype(str)
Krishnan['JobTitle'].apply(lambda str:('chief' in
str.lower())).sum()
```

627

**Conclusions: I have successfully implemented & understood Pandas Python libraries in this practical.**

Thank you

Matplotlib is a data visualization library in Python that is used to create static, animated, and interactive visualizations in Python. It is a powerful tool for creating high-quality 2D graphics and plots, and it can be used to generate a wide range of visualizations, including line plots, scatter plots, bar plots, histograms, and many more.

Matplotlib is widely used in the scientific community for visualizing data, and it is also used in data science and machine learning applications for exploring and analyzing data. The library is built on top of NumPy, another popular Python library for scientific computing, and it provides a simple and intuitive interface for creating complex visualizations with just a few lines of code.

### Functions used in Matplotlib

`plt.plot()`: This function is used to create line plots in Matplotlib. It takes two or more arrays of data as input and plots them as lines on a graph. The function also provides a wide range of customization options, including line styles, colors, markers, and more.

`plt.scatter()`: This function is used to create scatter plots in Matplotlib. It takes two arrays of data as input and plots them as individual points on a graph. The function also provides customization options for point size, color, marker style, and more.

`plt.bar()`: This function is used to create bar plots in Matplotlib. It takes one or more arrays of data as input and plots them as bars on a graph. The function also provides customization options for bar width, color, orientation, and more.

`plt.hist()`: This function is used to create histograms in Matplotlib. It takes one array of data as input and plots the distribution of the data as a series of bins on a graph. The function also provides customization options for bin size, color, and more.

`plt.pie()`: This function is used to create pie charts in Matplotlib. It takes one array of data as input and plots the data as slices of a pie on a graph. The function also provides customization options for slice colors, labels, and more.

`plt.imshow()`: This function is used to create image plots in Matplotlib. It takes an array of data as input and plots the data as an image on a graph. The function also provides customization options for color maps, interpolation, and more.

`plt.subplot()`: This function is used to create subplots in Matplotlib. It allows you to create multiple plots on the same figure and arrange them in a grid-like layout. The function also provides customization options for subplot spacing, titles, and more.

`plt.legend()`: This function is used to add a legend to a plot in Matplotlib. It allows you to label the different elements in a plot, such as lines or bars, and provide a key for interpreting the data. The function also provides customization options for legend location, font size, and more.

`plt.xlabel()` and `plt.ylabel()`: These functions are used to set the x and y-axis labels of a plot in Matplotlib. They allow you to provide a clear and concise description of the data being plotted and help readers understand the meaning of the axes.

`plt.title()`: This function is used to set the title of a plot in Matplotlib. It allows you to provide a brief summary of the data being plotted and can help readers quickly understand the main message of the plot.

`plt.xlim()` and `plt.ylim()`: These functions are used to set the limits of the x and y-axes of a plot in Matplotlib. They allow you to control the range of data being displayed on the plot and can be used to zoom in or out on specific parts of the data.

`plt.grid()`: This function is used to add a grid to a plot in Matplotlib. It can help readers better understand the data being plotted by providing a reference for the values on the x and y-axes.

`plt.subplots_adjust()`: This function is used to adjust the spacing between subplots in Matplotlib. It allows you to fine-tune the layout of your plots and ensure that they are clear and easy to read.

`plt.savefig()`: This function is used to save a plot as an image file in Matplotlib. It allows you to share your results with others and include your plots in reports, presentations, and other documents.

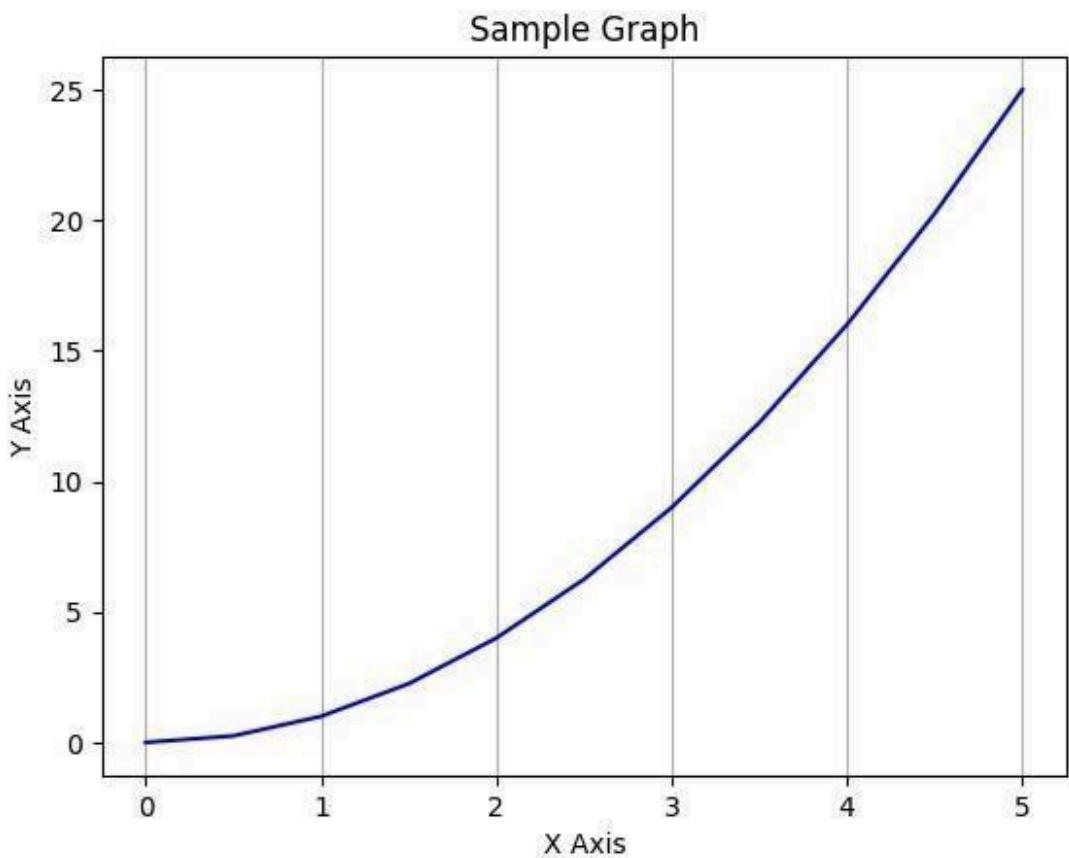
```
import matplotlib.pyplot as plt  
%matplotlib inline
```

### Example

```
import numpy as np  
x =  
np.linspace(0, 5, 11)  
y = x**2  
  
x  
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5, 5. ])  
  
y  
array([ 0. , 0.25, 0.5, 0.75, 1. , 1.25, 1.5, 1.75, 2. , 2.25, 2.5])
```

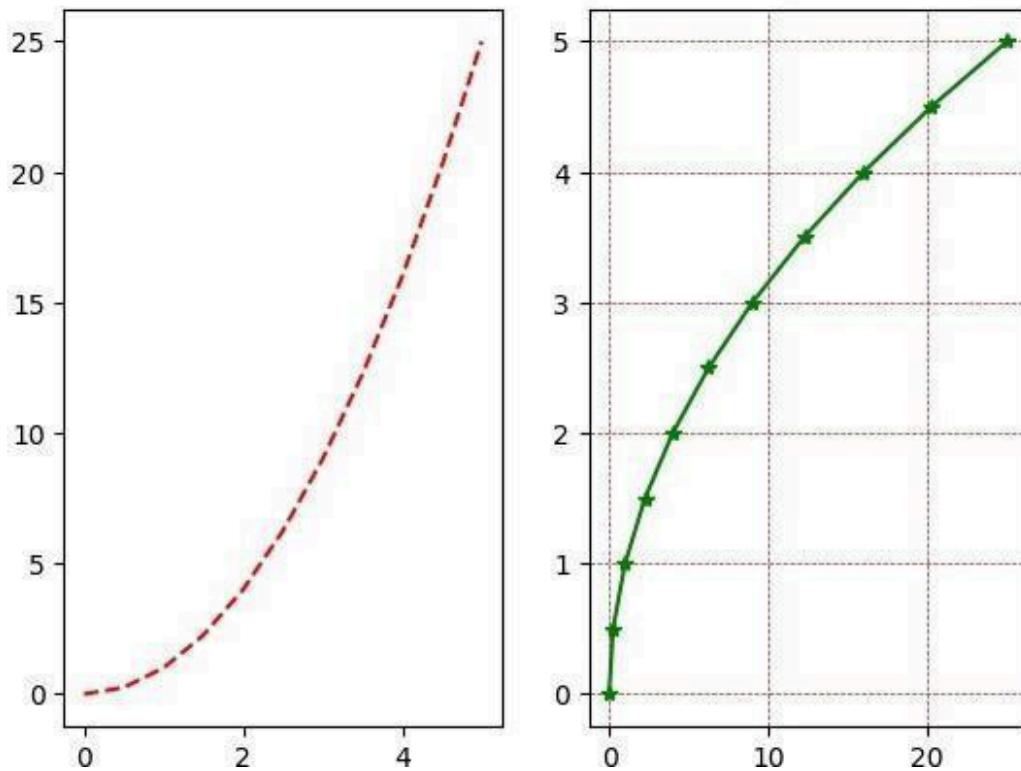
### Basic Matplotlib command

```
plt.plot(x,y,'b') #b =  
blue plt.xlabel('X  
Axis') plt.ylabel('Y  
Axis') plt.title("Sample  
Graph") plt.grid(axis =  
'x') plt.show()
```



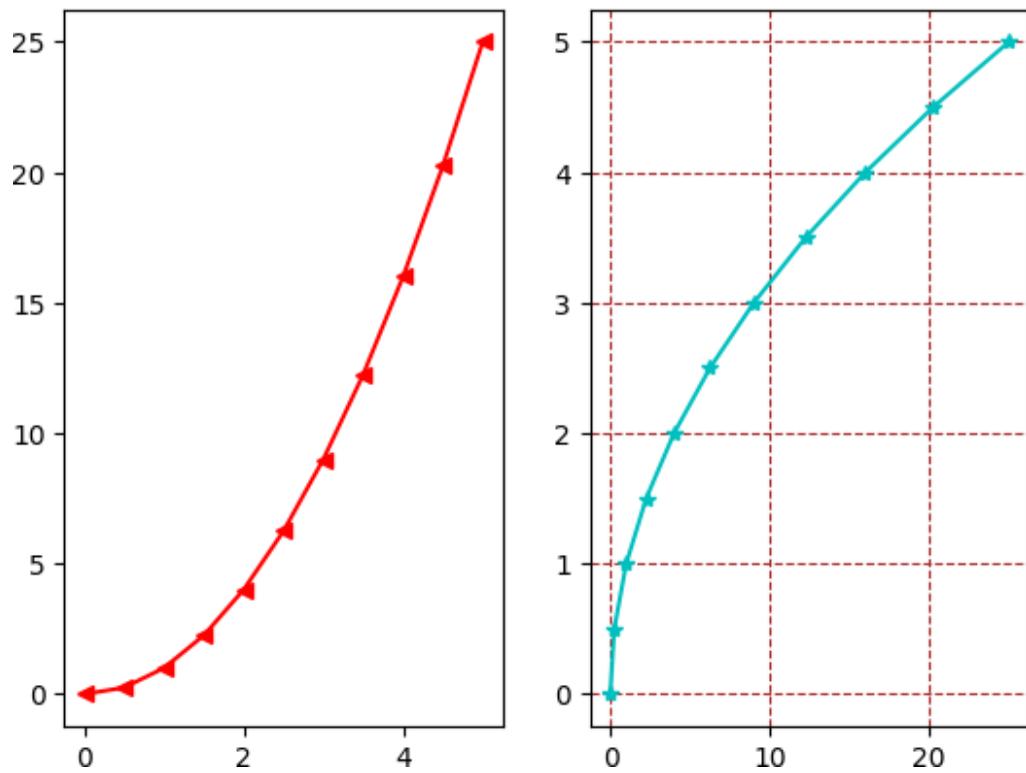
```
plt.subplot(1,2
,1)
plt.plot(x,y,'r
--')
plt.subplot(1,2
,2)
plt.grid(color='brown',linestyle = '--', linewidth =
0.5) plt.plot(y,x,'g*-')

[<matplotlib.lines.Line2D at 0x7f8e082dee00>]
```



```
plt.subplot(1,2
,1)
plt.plot(x,y, 'r
<--')
plt.subplot(1,2
,2)
plt.grid(color='brown',linestyle = '--', linewidth =
0.8) plt.plot(y,x, 'c*--')

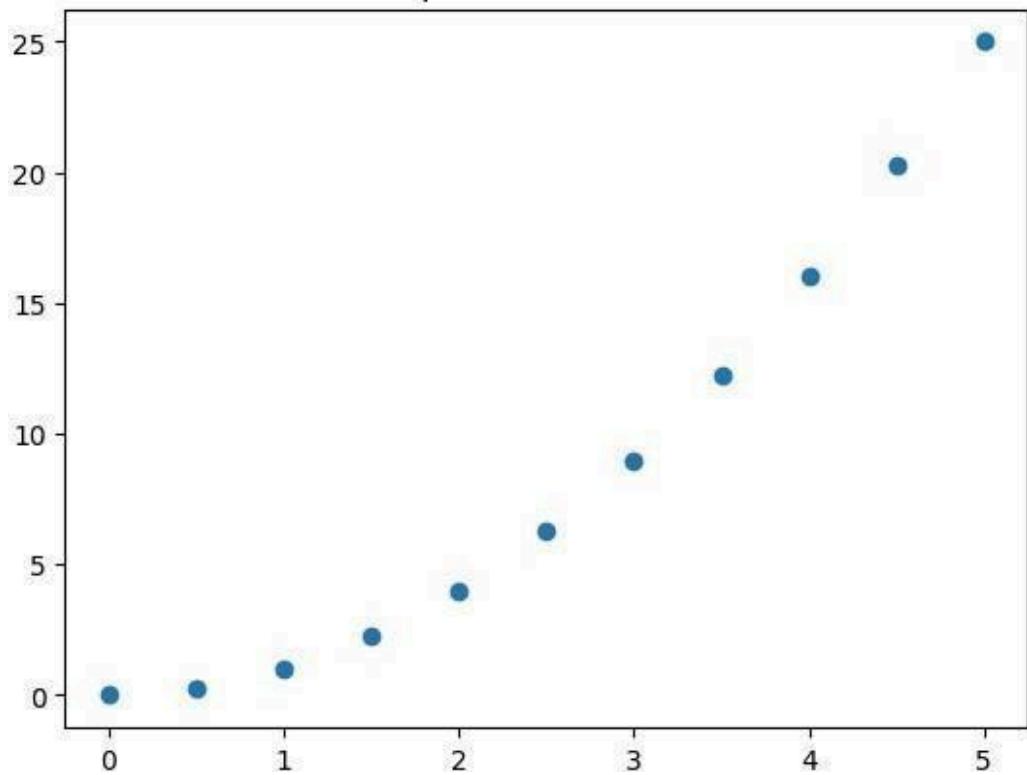
[<matplotlib.lines.Line2D at 0x7f8e08369090>]
```



```
plt.scatter(x,y)
plt.title("Sample Scattered Points")

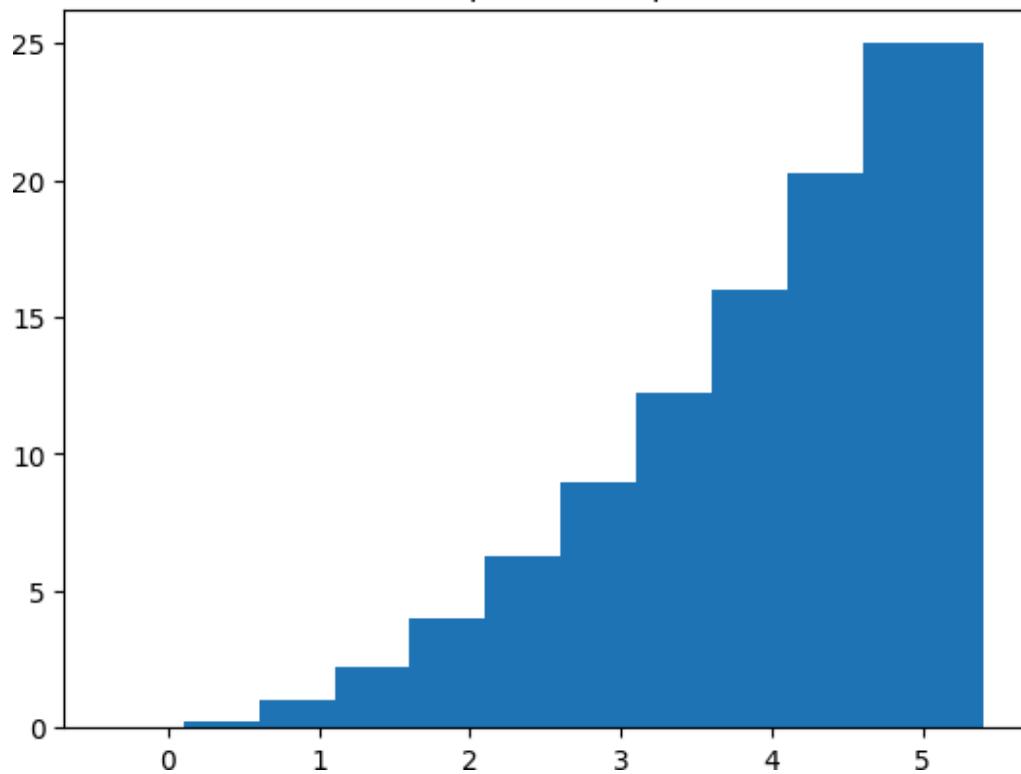
Text(0.5, 1.0, 'Sample Scattered
Points')
```

Sample Scattered Points

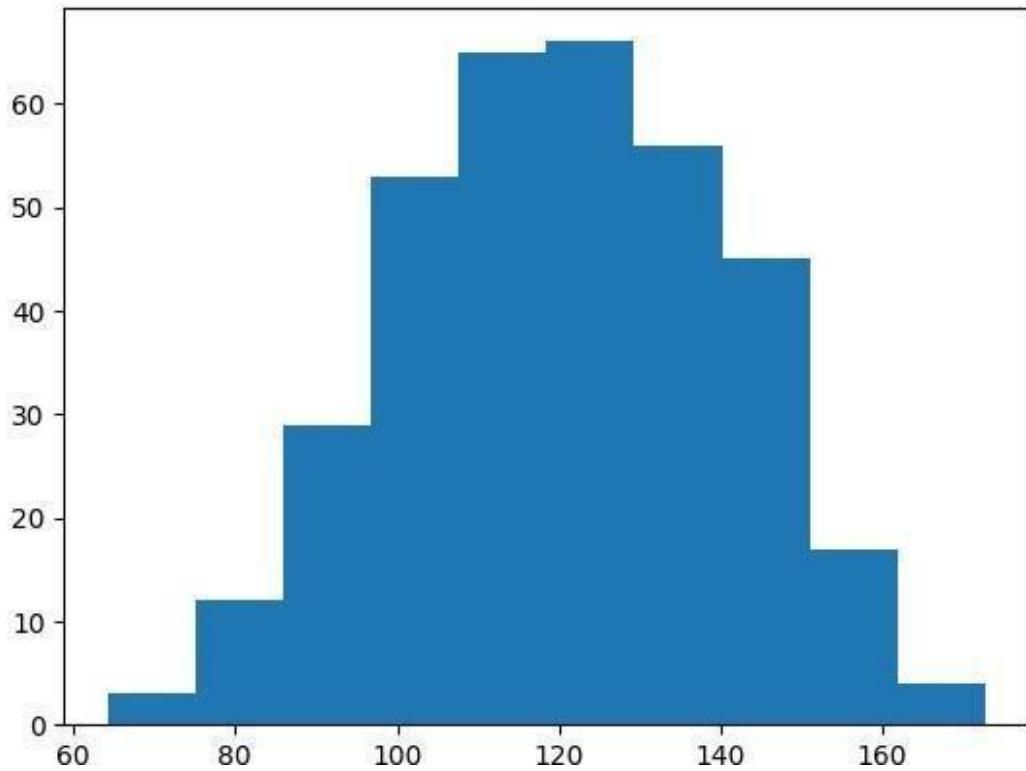


```
plt.bar(x,y)
plt.title("Sample Bar
Graph")
Text(0.5, 1.0, 'Sample Bar Graph')
```

### Sample Bar Graph



```
x = np.random.normal(120, 20, 350)
#np.random.normal(loc,scalea and size) loc sets the mean
distribution, #scale sets the standard deviation and size gives the
shape of numpy array
plt.hist(x)
plt.show()
```



`x`

```
array([116.68478926 101.68729593, 136.16753733, 162.9505946 ,  
       , 146.47845311, 106.54719812, 151.10626071, 124.66703975,  
       92.35268543, 81.94702501, 136.54797522, 139.67818061,  
       101.42465669, 92.84725894, 139.06278236, 134.37195897,  
       108.45287014, 118.8986514 , 117.6601463 , 130.09284006,  
       104.60412305, 119.74252476, 124.48952049, 92.64026653,  
       144.59570341, 158.03149925, 109.57502671, 129.73009742,  
       122.61097981, 150.03860933, 114.47049512, 140.1927212 ,  
       129.74985353, 111.80688556, 119.28482899, 76.79504484,  
       96.21333098, 111.47846799, 110.62480073, 123.41967591,  
       128.49071047, 94.67744057, 90.17957025, 149.16427725,  
       112.85603385, 140.71215783, 116.62607792, 96.25119731,  
       137.53545646, 110.5192244 , 105.12050185, 142.18261909,  
       116.0405155 , 137.27083313, 117.66621892, 111.4152536 ,  
       137.18080904, 114.38139098, 112.37455263, 164.50147552,  
       120.18535804, 125.52254762, 136.78463258, 132.24466098,  
       112.78782822, 126.61546787, 100.31969573, 132.24070154,  
       131.4803055 , 93.74754185, 111.84541187, 108.74210162,  
       98.57118639, 114.91729885, 103.88558604, 122.70672863,  
       106.39915036, 80.04322205, 127.77279739, 104.85654224,  
       141.35638209, 143.23261573, 120.62981922, 117.12179149,  
       123.8765076 , 142.16915942, 154.42721664, 101.40150616,  
       131.95346498, 105.50595273, 145.85685216, 116.39147948,  
       138.57896854, 103.89807931, 132.71877536, 142.94935681,
```

134.61020451, 69.22661557, 115.74933173, 90.74136006,  
152.68050206, 108.67024183, 128.64456603, 110.45511043,  
118.49688711, 135.76284491, 94.81306496, 142.8430495 ,  
140.42125427, 141.76670813, 118.25011408, 134.61022312,  
135.86167993, 70.34645287, 157.94798885, 99.64910684,  
139.30466092, 94.92155041, 86.23506058, 117.16930978,  
100.78186493, 75.87320867, 131.13438367, 98.75486314,  
94.16432016, 102.68157369, 129.80136775, 144.41264231,  
95.35435552, 95.7546287 , 101.53582764, 139.46908754,  
119.52450123, 103.30558151, 91.98428945, 127.55456209,  
121.333799 , 127.19332928, 103.84334038, 156.22799678,  
140.98405368, 113.30030954, 130.77166613, 84.24972685,  
127.92143133, 119.85854507, 84.07057701, 148.60657096,  
146.67796193, 94.90645898, 99.8205616 , 119.52736451,  
102.45689335, 147.38045394, 87.43905123, 108.12762545,  
145.0800277 , 126.18457465, 135.55551117, 105.83109352,  
99.5417883 , 161.20325117, 127.57316815, 138.9690165 ,  
95.44460373, 154.28580168, 118.72413573, 105.7436593 ,  
122.2115552 , 147.72451174, 90.0565301 , 127.57765766,  
121.83314219, 140.25243514, 107.87289136, 114.61684797,  
112.00423623, 131.82280701, 117.76372457, 116.17122228,  
141.36439283, 130.9103412 , 106.7551253 , 130.99089049,  
134.32370698, 102.86757309, 121.02453196, 101.60510701,  
88.09383513, 116.54893478, 116.14835748, 113.75497653,  
110.64594052, 149.45573553, 101.6429037 , 120.66025493,  
125.27568805, 125.5582373 , 128.1090949 , 92.92302525,  
117.55126293, 140.39073384, 125.93494974, 124.26013174,  
105.95290235, 109.81187245, 144.25335923, 116.79796389,  
121.72254763, 82.72888794, 99.85483483, 108.14291307,  
97.1812001 , 81.74839655, 132.61242682, 119.70619407,  
102.13018678, 142.31674243, 121.15326498, 113.80979353,  
154.05742223, 160.78757811, 85.99132201, 143.37782615,  
159.19207703, 107.20349419, 146.683582 , 99.31354666,  
142.48070755, 95.92174538, 98.11638299, 132.27271866,  
118.97223532, 154.95814484, 105.99992576, 105.59384403,  
135.78023944, 83.45024521, 151.72612019, 143.00728986,  
118.86452285, 121.09326613, 136.65105163, 135.77676657,  
109.06136184, 100.6121431 , 127.33886047, 96.47694999,  
154.3156676 , 120.57594448, 128.5446566 , 106.80808735,  
149.74537006, 121.6094723 , 130.50996692, 142.44336114,  
100.06733704, 97.15829317, 134.16109064, 111.45829591,  
114.01577883, 153.0907597 , 132.00514222, 143.72673698,  
99.4123735 , 125.83874401, 115.87310114, 141.01129208,  
111.89437306, 136.82631453, 93.94752165, 129.30458992,  
82.84789998, 111.38222729, 113.039687 , 172.64721903,  
147.50948837, 121.14724732, 145.53545438, 84.10502955,  
121.74359787, 120.34125536, 131.31250682, 103.27997564,  
103.03864076, 141.42744802, 112.15992928, 137.00252423,  
121.79956805, 134.02325263, 143.05833535, 129.98089987,  
109.56624299, 126.03765595, 116.4667076 , 154.56990822,

```

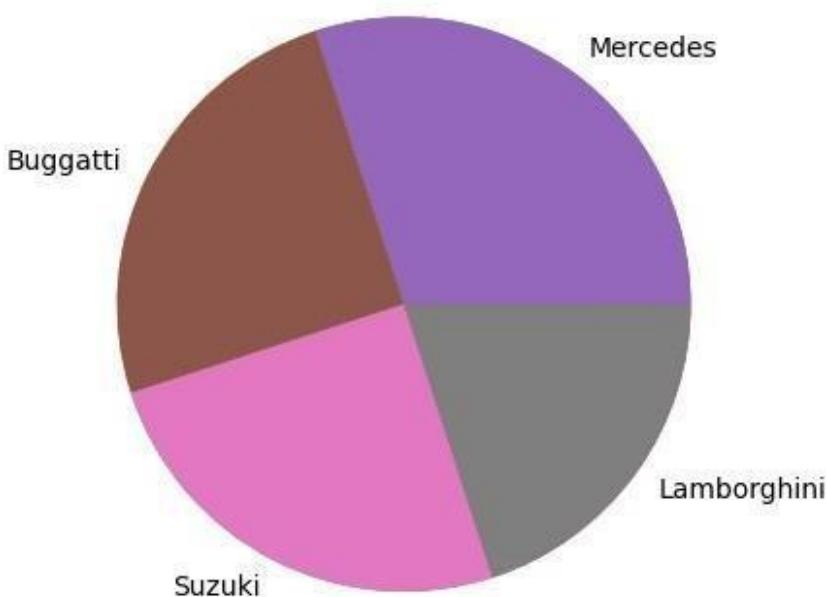
137.69568375, 105.67604288, 148.04878915 115.95841458,
104.35705987, 129.28532275, 116.62808253 129.67367501,
138.96298834, 120.14946007, 130.6427492 107.87990623,
115.97665118, 109.49352575, 116.47611854 130.34007772,
116.16506551, 115.0608593 134.9586472 109.94148561,
64.28186071, 137.18709704, 124.01843867 150.31496571,
128.16398536, 119.93294171, 120.83214252 121.62080028,
104.0753519 , 90.7567672 , 124.8122574 105.74233454,
81.93265876, 119.09141329, 120.97605664 113.5598729 ,
122.63970935, 100.66431341, 111.63384283 126.61039476,
137.39443387, 106.93697082, 93.38973566 155.60114549,
172.52018445, 111.11256243, 121.28274954 93.06057411,
142.1993444 , 105.01014508, 142.72866379 145.98782259,
97.68752369, 133.69123859]
)

```

```

y =
np.array([30,25,25,20])
plt.pie(y)
mylabels = ['Mercedes','Buggatti','Suzuki','Lamborghini']
plt.pie(y,labels = mylabels)
plt.show()

```



```
x =  
np.linspace(0, 5, 11)  
y= x**2  
#Create Figure (empty canvas)  
fig = plt.figure()  
#Addd set of axes to figure
```

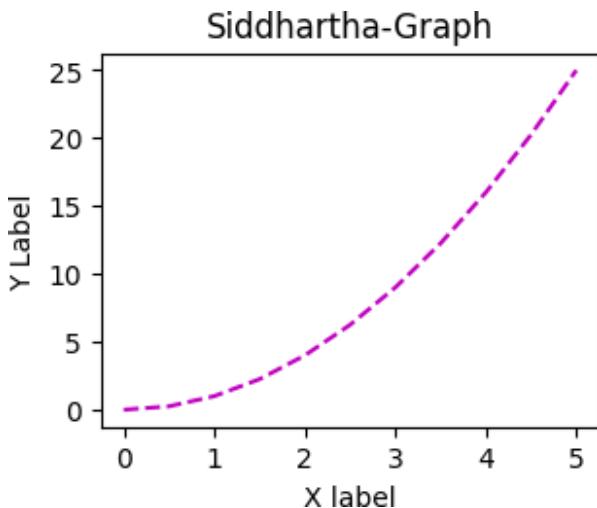
```

axes = fig.add_axes([0.2,0.2,0.4,0.4]) #left, bottom, width,
height(range 0 to 1)

#Plot on that set of axes
axes.plot(x,y,'m--')
axes.set_xlabel('X label')
axes.set_ylabel('Y Label')
axes.set_title('Sidharth-Gra
ph')

Text(0.5, 1.0, 'Sidharth-Graph')

```



## Inset Graphs

```

#Create blank canvas
fig = plt.figure()

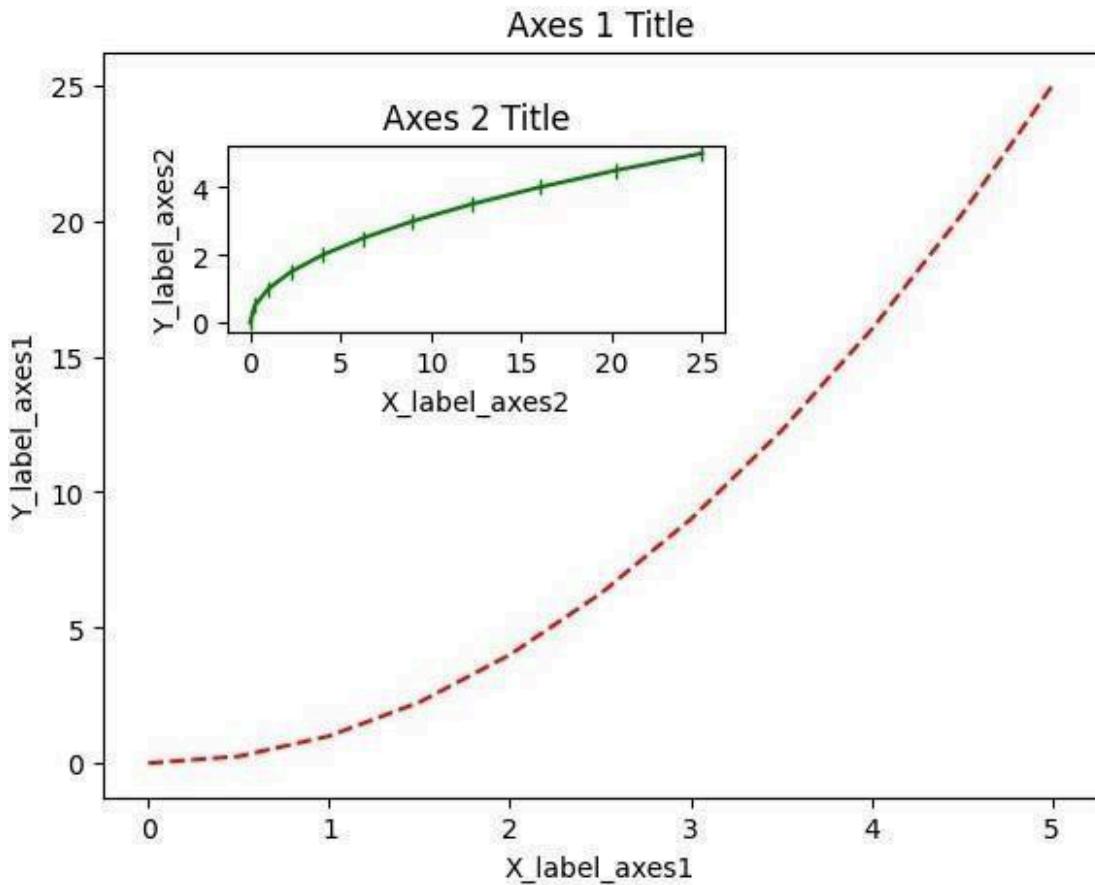
axes1 = fig.add_axes([0.1,0.1,0.8,0.8]) #main axes
axes2 = fig.add_axes([0.2,0.6,0.4,0.2]) #inset axes

#Larger Figure Axes 1
axes1.plot(x,y,'r--')
axes1.set_xlabel('X_label_axe
s1')
axes1.set_ylabel('Y_label_axe
s1') axes1.set_title('Axes 1
Title')

#Insert Figure Axes 2
axes2.plot(y,x,'g|-')
axes2.set_xlabel('X_label_axe
s2')
axes2.set_ylabel('Y_label_axe
s2') axes2.set_title('Axes 2
Title')

Text(0.5, 1.0, 'Axes 2 Title')

```

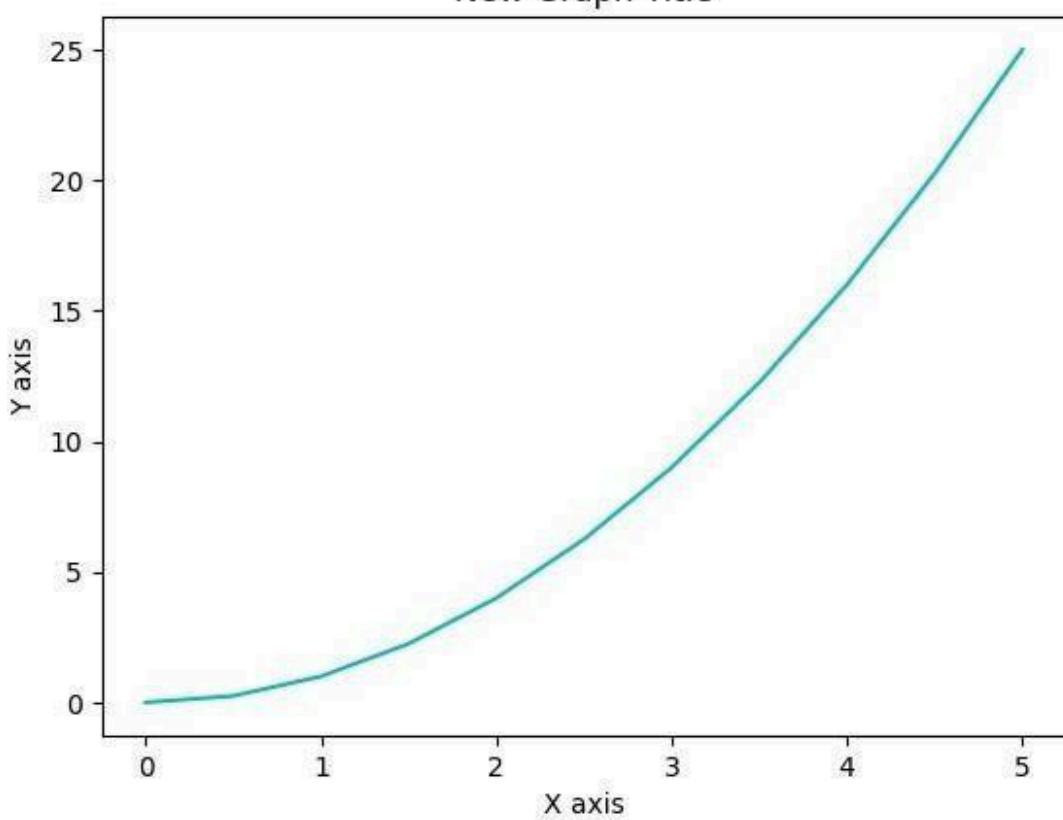


### Subplots

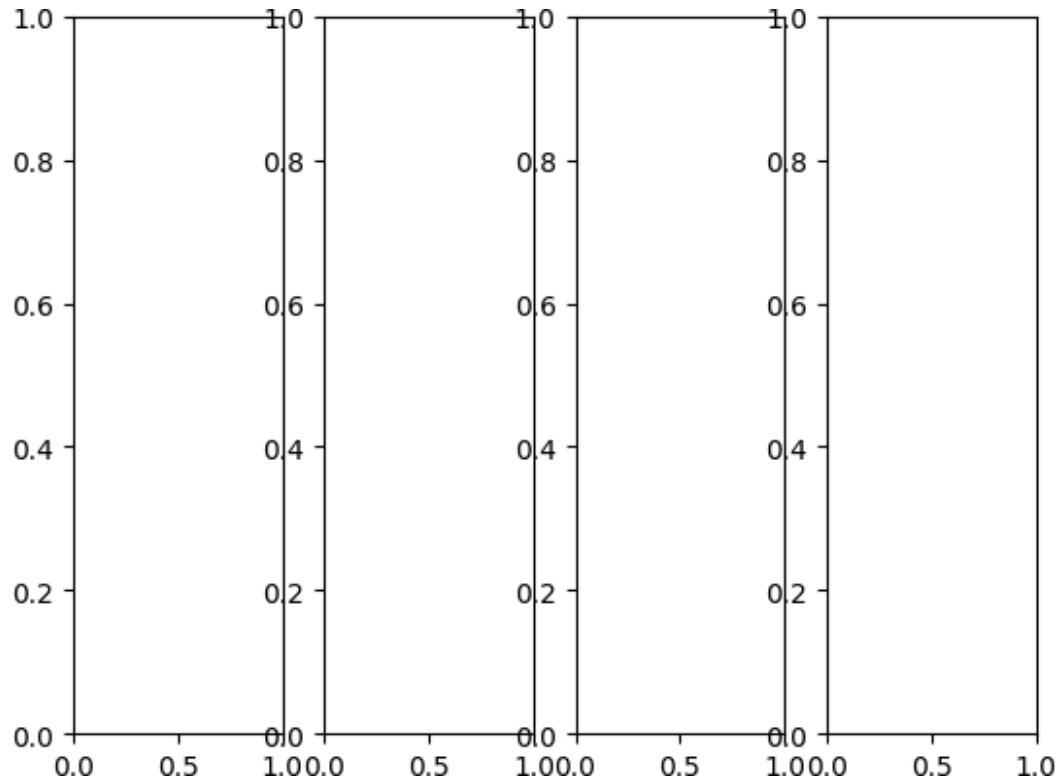
```
fig,axes = plt.subplots()
axes.plot(x,y,'c')
axes.set_xlabel('X axis')
axes.set_ylabel('Y axis')
axes.set_title('New Graph
Title')

Text(0.5, 1.0, 'New Graph Title')
```

New Graph Title



```
#Empty canvas of 1 by 2 subplots
fig,axes = plt.subplots(nrows=1,ncols=4)
```

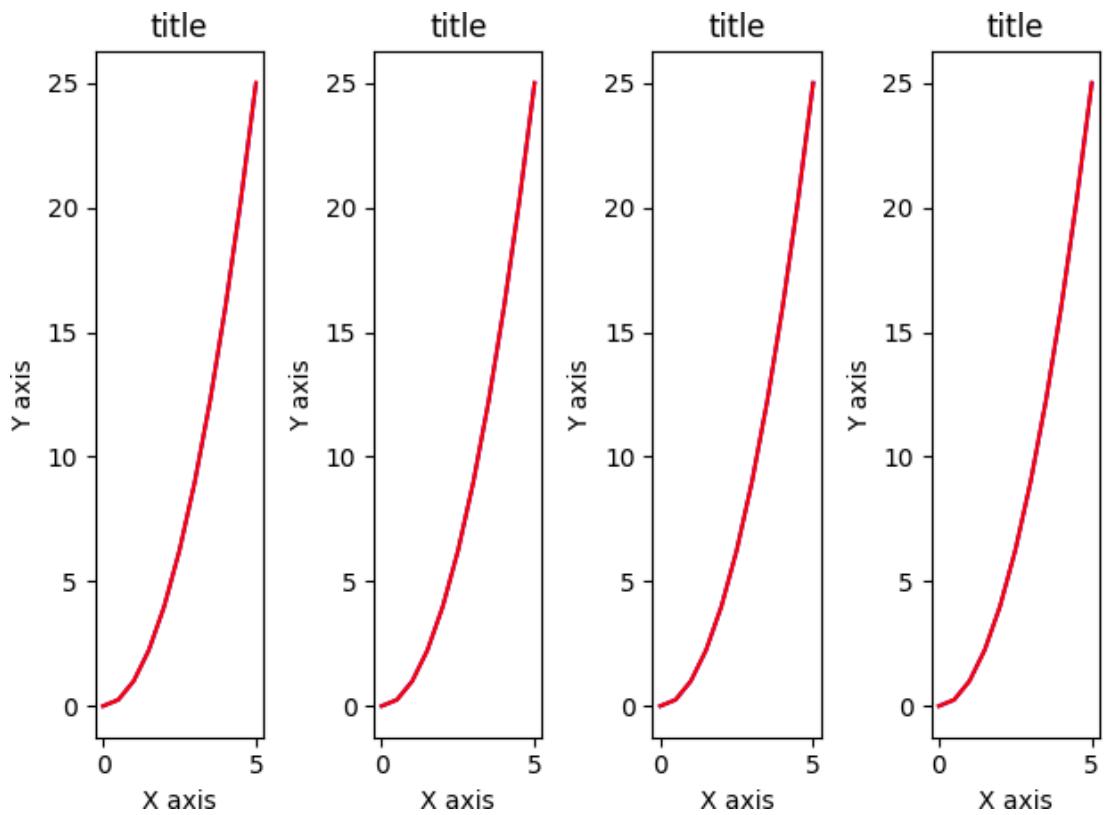


#Axes is an array of axes to plot on  
axes

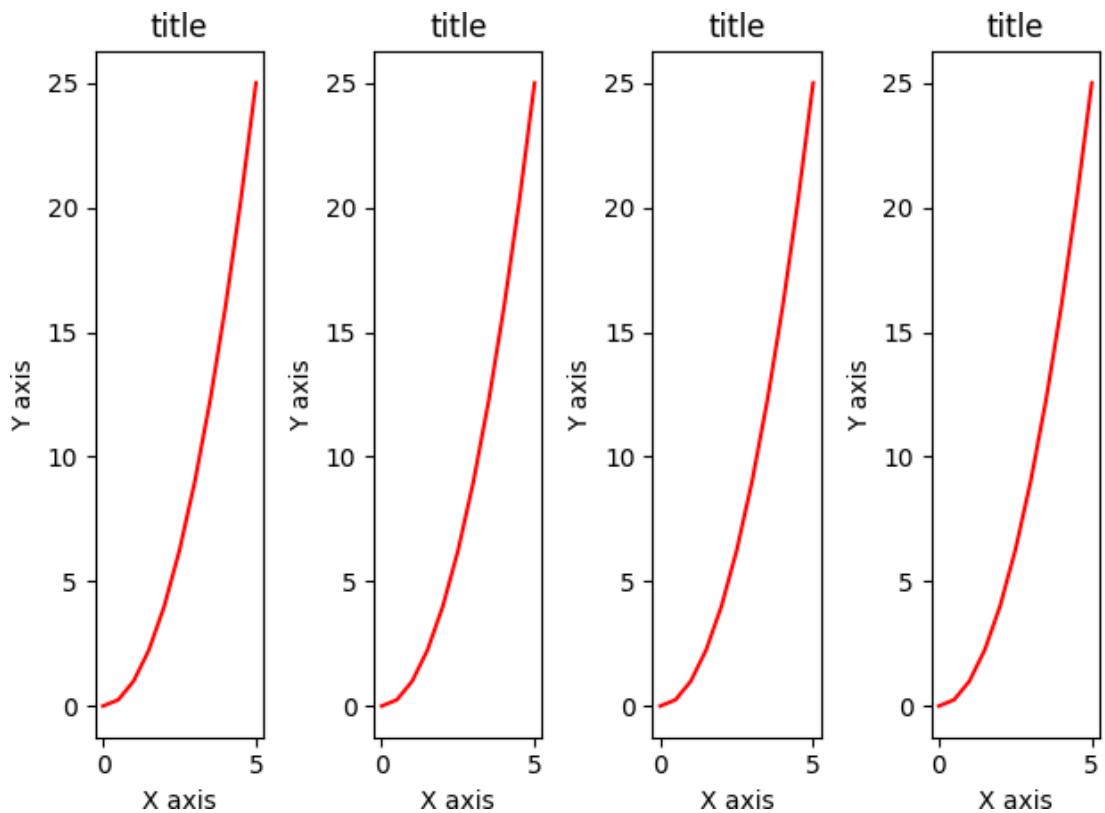
```
array([<Axes: >, <Axes: >, <Axes: >, <Axes: >], dtype=object)
```

We can iterate through this array

```
for ax in axes:  
    ax.plot(x,y,'r')  
    ax.set_xlabel('X  
axis')  
    ax.set_ylabel('Y  
axis')  
    ax.set_title('title  
)  
fig
```



```
fig,axes = plt.subplots(nrows=1,ncols=4)
for ax in axes:
    ax.plot(x,y,'r')
    ax.set_xlabel('X
axis')
    ax.set_ylabel('Y
axis')
    ax.set_title('title
')
fig.tight_layout()
```

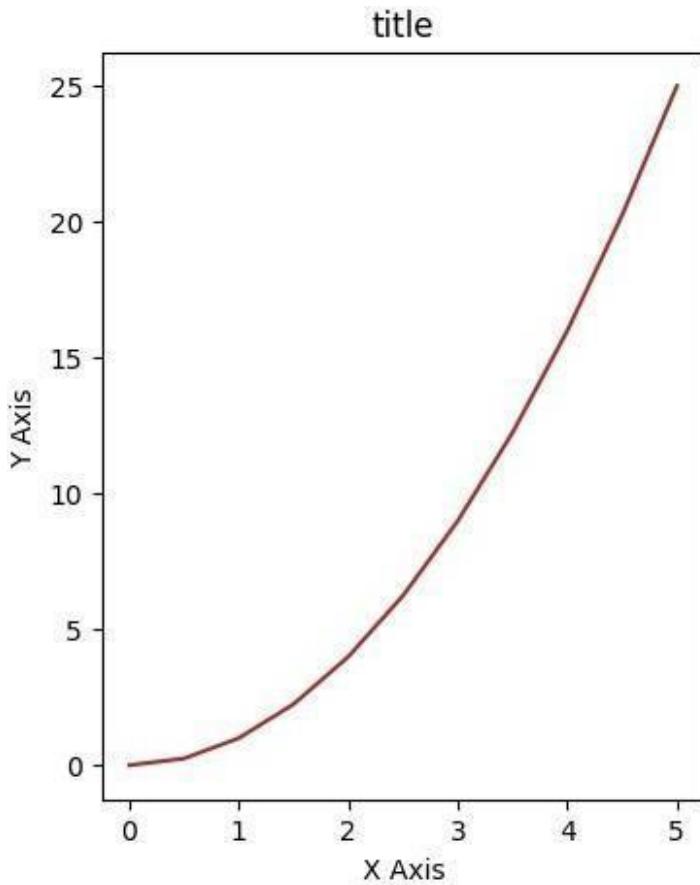


```
x =
np.linspace(0, 5, 11)
y = x ** 2

fig = plt.figure(figsize=(4, 4), dpi=50)
<Figure size 200x200 with 0 Axes>

fig, axes =
plt.subplots(figsize=(4, 5), dpi=100)
axes.plot(x,y,'brown')
axes.set_xlabel('X
Axis')
axes.set_ylabel('Y
Axis')
axes.set_title('title'
)

Text(0.5, 1.0, 'title')
```



```
import os
import matplotlib.pyplot as plt
script_dir= os.path.dirname('graph')
results_dir= os.path.join(script_dir, 'Results/')
sample_file_name = "sample"
if not os.path.isdir(results_dir):
    os.makedirs(results_dir)

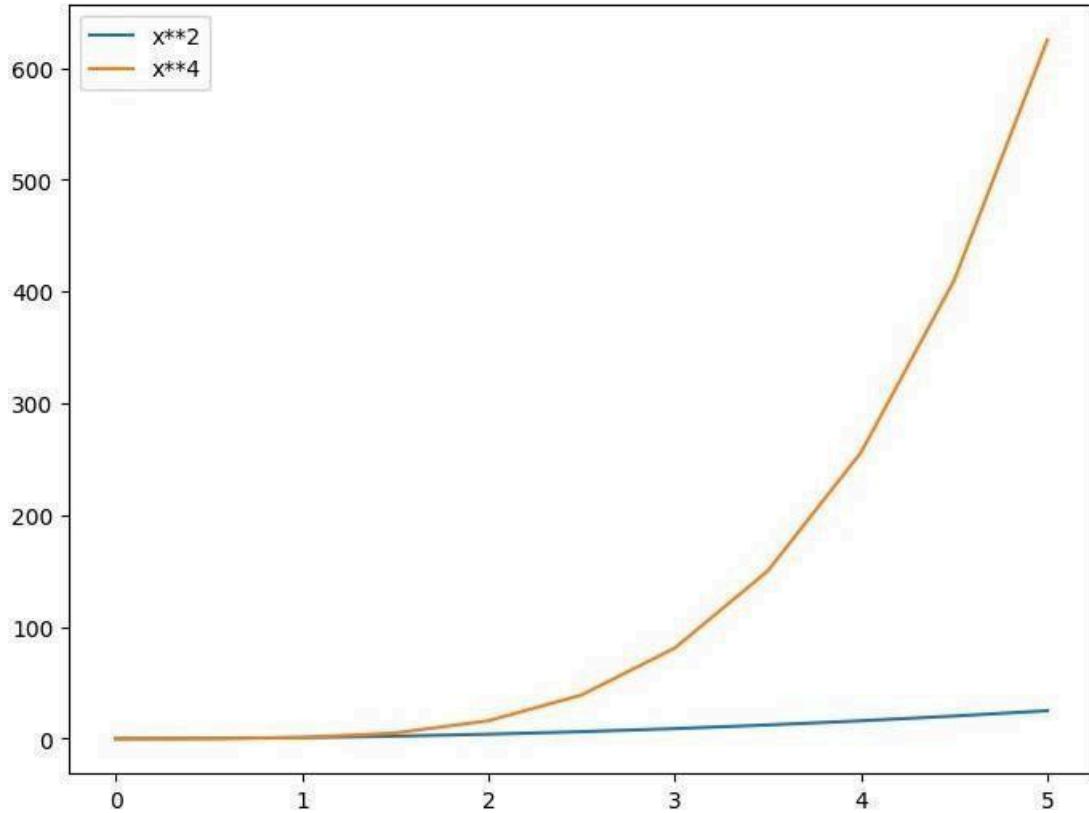
fig.savefig("graph2.png", dpi=200)
```

## Legends

You can use the `label="label text"` keyword argument when plots or other objects are added to the figure, and then using the `leg` method without arguments to add the legend to the figure:

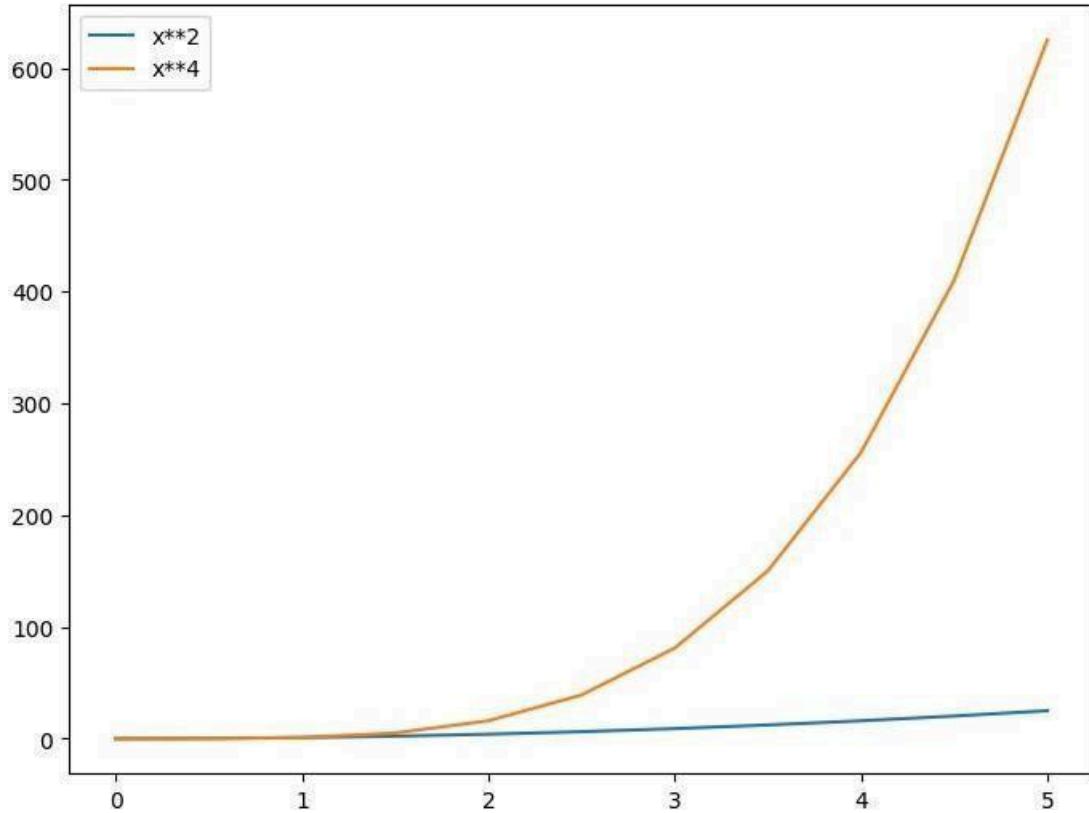
```
fig = plt.figure()
ax = fig.add_axes
([0,0,1,1]) ax.plot(x,
x**2, label="x**2")
ax.plot(x, x**4,
label="x**4") ax.leg()
```

<matplotlib.legend.Legend at 0x7f8de5aefd30>



The legend function takes an optional keyword argument loc that can be used to specify where in the figure the legend is to be drawn. The allowed values of loc are numerical codes for the various places the legend can be drawn.

```
# Lots of options....
ax.legend(loc=1) # upper right
corner ax.legend(loc=2) # upper
left corner ax.legend(loc=3) #
lower left corner ax.legend
(loc=4) # lower right corner # ..
many more options are available
# Most common to choose
ax.legend(loc=0) # let matplotlib decide the optimal location
fig
```

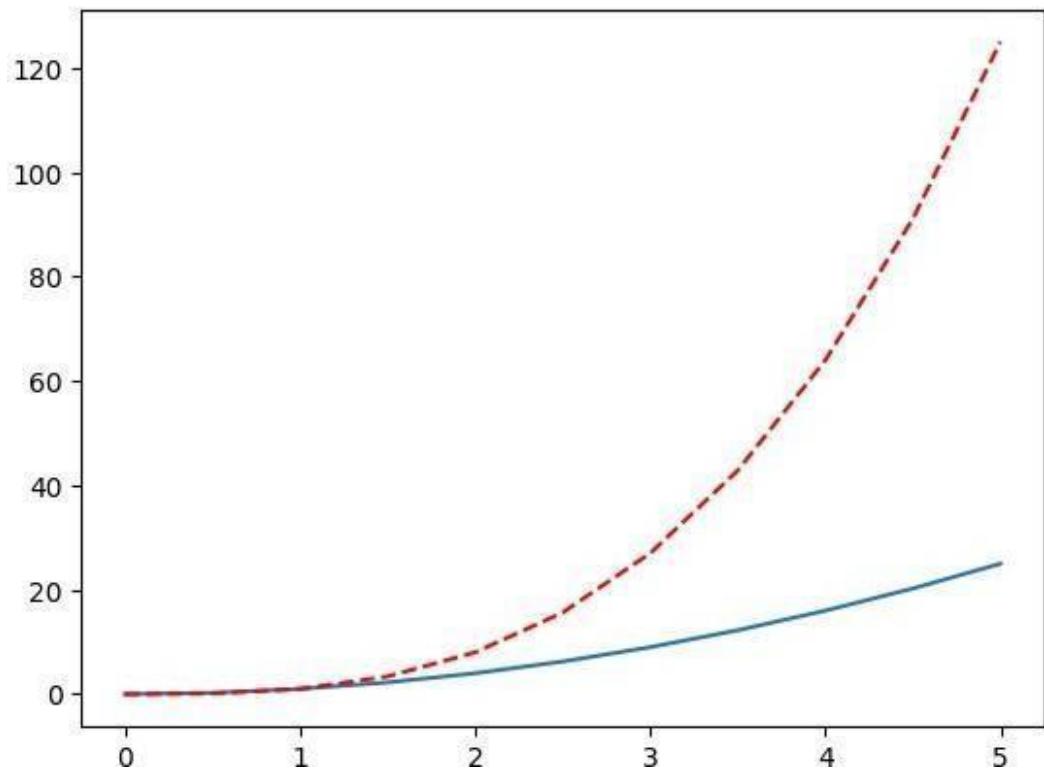


### Setting colors, linewidths, linetypes

Matplotlib gives you a lot of options for customizing colors, linewidths, and linetypes.

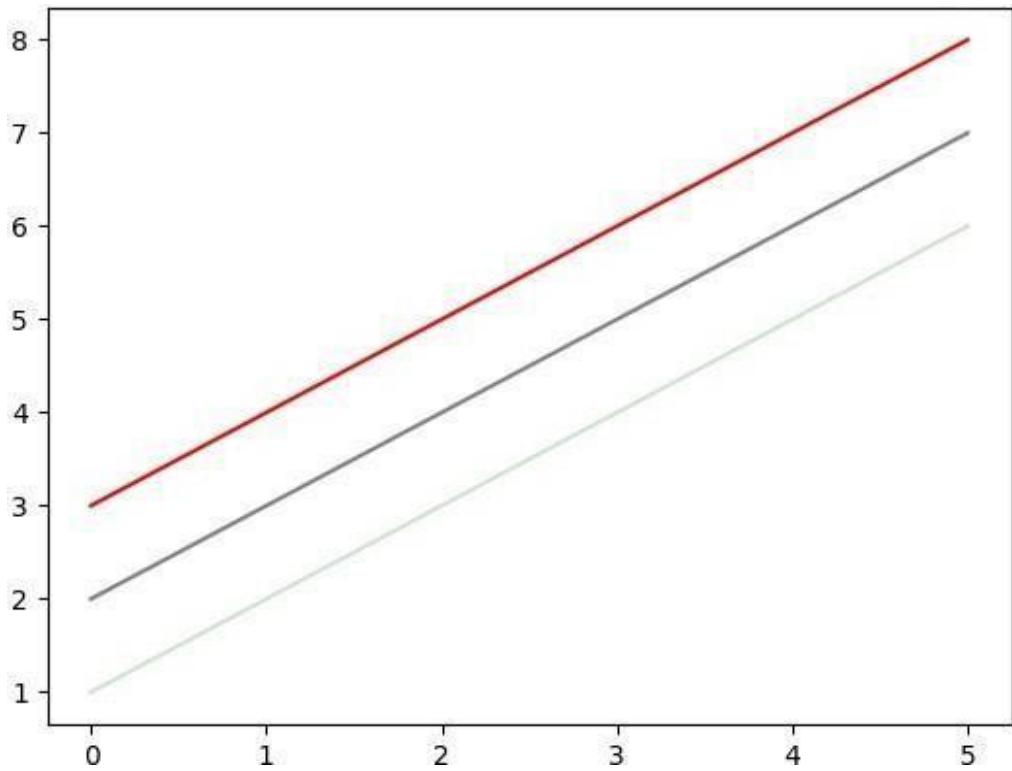
With matplotlib, we can define the colors of lines and other graphical elements in a number of ways. for example, 'b.-' means a b line with dots:

```
# MATLAB style line color and style
fig, ax = plt.subplots()
ax.plot(x, x**2, '-') # blue line with dots
ax.plot(x, x**3, 'r--') # green dashed line
[<matplotlib.lines.Line2D at 0x7f8de5acddb0>]
```



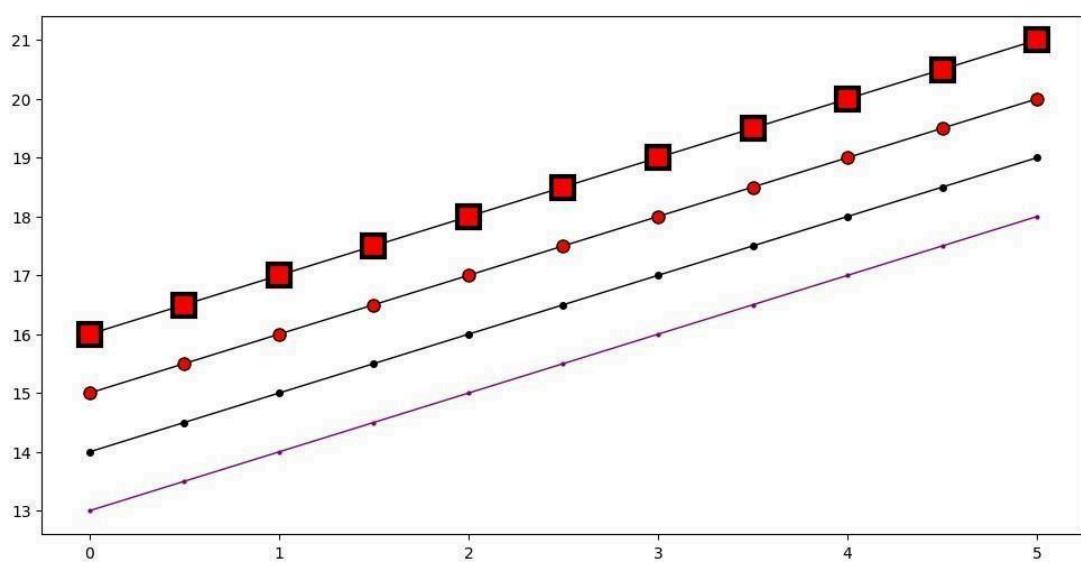
```
fig, ax = plt.subplots ()
ax.plot(x, x+1, color="green", alpha=0.2) # half-transparent
ax.plot(x, x+2, color="grey")
ax.plot(x, x+3, color="red")

[<matplotlib.lines.Line2D at 0x7f8de53521a0>]
```



```

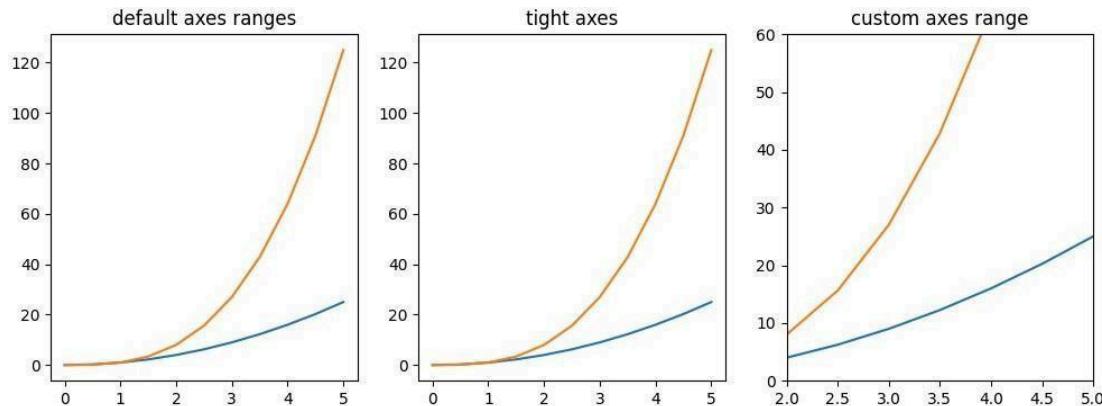
fig, ax = plt.subplots (figsize=(12, 6))
ax.plot(x, x+13, color="purple", lw=1, ls='--',
marker='o', markersize=2)
ax.plot(x, x+14, color="k", lw=1, ls='--', marker='o',
markersize=4) ax.plot(x, x+15, color="k", lw=1, ls='--',
marker='o', markersize=8, markerfacecolor="red")
ax.plot(x, x+16, color="k", lw=1, ls='--', marker='s',
markersize=15, markerfacecolor="red", markeredgewidth=3,
markeredgecolor="black");
    
```



## Plot range

We can configure the ranges of the axes using the `set ylim` and `set xlim` methods in the `axis` object, or `axis ('tight')` for automatically getting "tightly fitted" axes ranges:

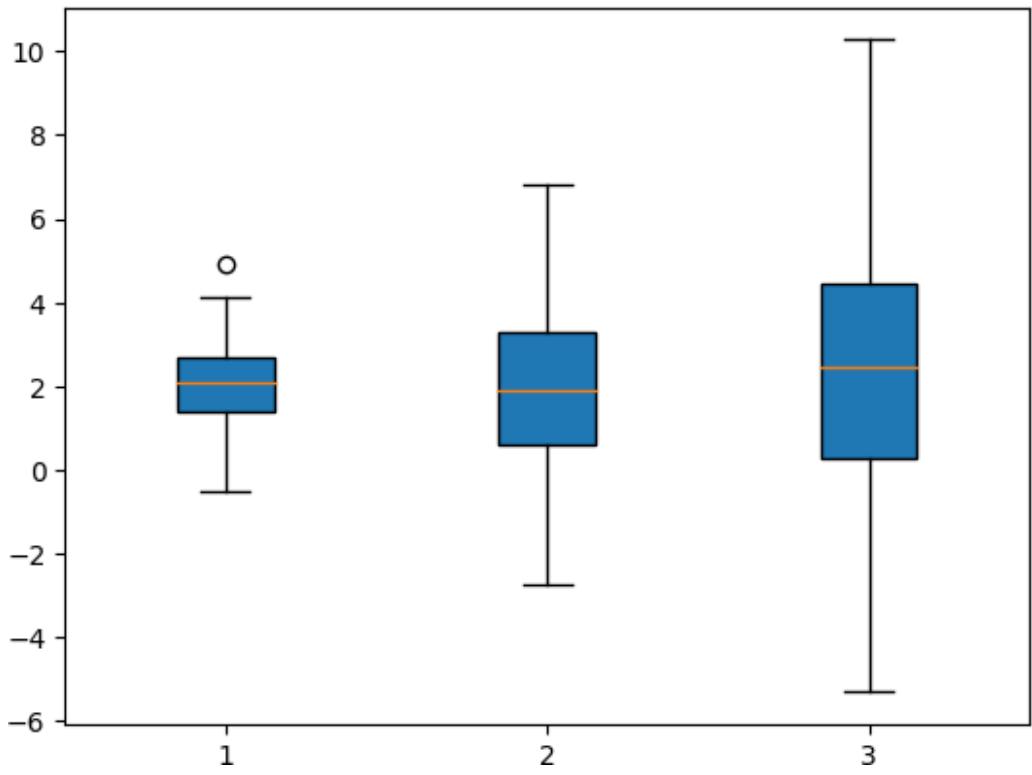
```
fig, axes = plt.subplots (1, 3, figsize=(12, 4))
axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")
axes[1].plot(x, x**2, x, x**3)
axes[1].axis ('tight')
axes[1].set_title("tight
axes") axes[2].plot(x, x**2,
x, x**3) axes[2].set_ylim
([0, 60])
axes[2].set_xlim ([2, 5])
axes[2].set_title("custom axes range");
```



## BoxPlot

It is used to identify outliers in datasets.

```
data = [np.random.normal (2, std, 100) for std in range(1, 4)]
# rectangular box plot Loading...
plt.boxplot (data, vert=True, patch_artist=True);
```



```
data
```

```
[array([ 3.54547107,  4.90202531,  1.72618052,  4.10562617,
2.89636303,
       2.33596092,  1.41899852,  2.07065061,  0.67021734,
2.06720598,
       2.67708759,  0.9138778 ,  2.63979198,  2.40620801,
1.41815297,
       2.58848217,  2.64656626,  3.66889209,  1.98487532,
1.83050364,
       3.41230138,  3.07636861,  0.59129823,  2.04145884,
2.54740008,
       3.93037043,  1.24618444,  1.36004138,  2.14925809,
3.41287424,
       3.4437647 ,  3.2337155 ,  1.34491963,  1.86650404,
1.88881206,
       1.6661322 ,  2.23854467,  0.44764475,  2.1706033 ,
3.24882574,
       1.03365724,  1.37394981,  0.85640735,  1.2347266 ,
2.39217215,
       2.17395076,  3.14165856,  1.29865876, -0.49252615,
0.40731494,
       2.0644662 ,  1.39657956,  2.05390461,  3.41321671,
1.86237015,
       0.34140262,  1.4367122 ,  2.09236867,  3.19310853,  2.280040
,
```

	3.32171913,	1.67019046,	1.91508909,	2.38964721,
3.60963797,		2.36392476,	1.08589546,	1.24237693,
				1.58759606, 1.061682
				2
,				
2.34975165,	1.9093815 ,	1.07467846,	2.70854254,	1.59330376,
1.94580598,	0.88374366,	2.19904951,	2.15231618,	3.68689143,
2.48508371,	2.7573376 ,	1.69758926,	1.38589695,	3.76180734,
2.83585155,	1.05461343,	3.0702265 ,	1.65048579,	0.3958311 ,
2.04401313,	1.88579762,	3.30984743,	3.74524296,	2.39987739,
0.87658935]),	2.23903857,	1.57228193,	2.40095432,	0.66177984,
array([ 1.90520977,	1.41282937,	-1.0991555 ,	3.38631887,	
1.21613728,	2.7302254 ,	0.38365692,	4.0756801 ,	3.06912884,
3.55838852,	2.93652666,	3.29562525,	3.4334434 ,	3.70529885,
5.05755943,	2.48280649,	6.03769081,	4.60594573,	1.75499026, -
0.45885499,	3.91420083,	-0.77947809,	4.03256121,	-0.12802203,
3.06329839,	0.77938552,	3.75498896,	1.28699989,	0.26545226,
2.55443067,	1.20205113,	0.85708884,	2.45263719,	0.61994023, -2.740216
				6
,				
1.19822469,	4.4018532 ,	0.58125719,	3.88825702,	0.71500987,
0.64092016,	4.74417517,	3.102524 ,	0.95519784,	2.14641178,
0.43553576,	1.91908883,	0.56765002,	-0.78617679,	2.50688772,
0.89738569,	-1.32215163,	4.14990758,	2.55124682,	0.12863474,
1.60994455,	1.52211146,	-1.97067795,	0.12965394,	1.70877 ,
0.06661896,	1.19897536,	-0.64059083,	2.18716502,	1.23160609,
2.03784721,	3.03259612,	1.9070944 ,	-2.54722843,	2.94879199,
1.68872711,	0.53497095,	4.45594839,	4.96349221,	-0.22081172,
1.36919204,	3.27626548,	0.42530851,	-1.31669682,	5.25937042, -
4.59752722,	3.35832931,	1.14854814,	1.15499537,	3.51084504,

1.56002228, 0.2139386 , 2.11015931, -0.7680856 , -  
1.99954364,  
3.25614747, 2.45541407, 5.49058485, 2.65764985,  
2.50120509,  
3.09271919, 3.31532858, 2.6857884 , 6.83130629,  
2.49954576]),  
array([ 1.54805074, 3.76647163, 6.53913158, 0.10141338,  
6.81826435,  
2.18617905, 5.15811704, 3.10826004, 4.74720476,  
3.00522911,  
2.68885277, -2.48928208, 0.24852702, 3.71924979,  
2.92219993,  
4.09436414, -0.05596572, 5.1529535 , 3.79492453, -  
0.06376706,  
4.41632428, 8.97304947, 9.23928038, 2.15383419, 2.1233357  
,

-0.97855506, 1.02192742, 3.84855616, -1.39679981, -  
2.36465143,  
-5.31037613, 8.02627247, 4.74459613, 2.5659882 , -  
0.48768884,  
-0.48258814, 4.51507774, 2.05578623, 1.25931314,  
0.83885011,  
-0.37579525, 1.83490774, -0.26017795, 5.06211815,  
2.18252395,  
-1.94461936, -4.11745833, 6.85763945, 0.55745329, 6.6407942  
,

6.24024073, 1.33670758, 6.71663946, -0.11884962, -  
3.58349119,  
1.01365851, 2.73543155, 0.08053506, 0.61050282,  
2.10945443,  
2.77173643, 1.31243359, -1.31934403, 3.17034059,  
0.56251155,  
10.27003954, 5.14694211, 0.16269928, 0.62457208,  
4.67583413,  
3.48382781, -2.31045246, 0.62665299, 0.70973567,  
6.65971885,  
2.42988593, 6.53869243, -1.4124189 , 0.44633382,  
7.11881703,  
2.46162708, 3.60701045, -0.68800733, 3.77293721, -  
0.08856646,  
0.2744635 , -2.57304838, 0.3794692 , 4.47957006,  
4.73844403,  
4.31919273, 5.27594847, 9.30250666, 3.48292549,  
3.85721022,  
1.55364188, 2.7861778 , 3.43034563, 2.48518241,  
3.29704751]))

---

## Matplotlib Exercises

\*\*\* NOTE: ALL THE COMMANDS FOR PLOTTING A FIGURE SHOULD ALL GO IN THE SAME CELL. SEPARATING THEM OUT INTO MULTIPLE CELLS MAY CAUSE NOTHING TO SHOW UP. \*\*\*

### Exercise 0

1. Take a dataset of your choice and plot graphs using matplotlib(convert numeric data to nominal where ever required)

Campus Recruitment Dataset: \* This data set consists of Placement data of students in a XYZ campus. It includes secondary and higher secondary school percentage and specialization. It also includes degree specialization, type and Work experience and salary offers to the placed students. Link:

<https://www.kaggle.com/datasets/benroshan/factors-affecting-campus-placement>

```
import pandas as pd
import io
import matplotlib.pyplot as plt
%matplotlib inline

from google.colab import files
uploaded = files.upload()

<IPython.core.display.HTML object>

Saving 20_Centuries_Virat_Kohli.csv to 20_Centuries_Virat_Kohli.csv

data =
pd.read_csv(io.BytesIO(uploaded['20_Centuries_Virat_Kohli.csv
'])) data.head()

   Score      Against Batting Order Inn. Strike Rate \
0     116      Australia             6      2       120.0
1     103    New Zealand             5      2       111.0
2     103        England             5      2       124.0
3     107      Australia             5      2       133.0
4     119  South Africa             4      1       141.2

                                         Venue
0                               Adelaide Oval
1                         M. Chinnaswamy Stadium
2  Vidarbha Cricket Association Stadium
3                M. A. Chidambaram Stadium
4                  Wanderers Stadium
```

How much dependency between MBA percentage and Salary with specialisation

```

import matplotlib.pyplot as plt

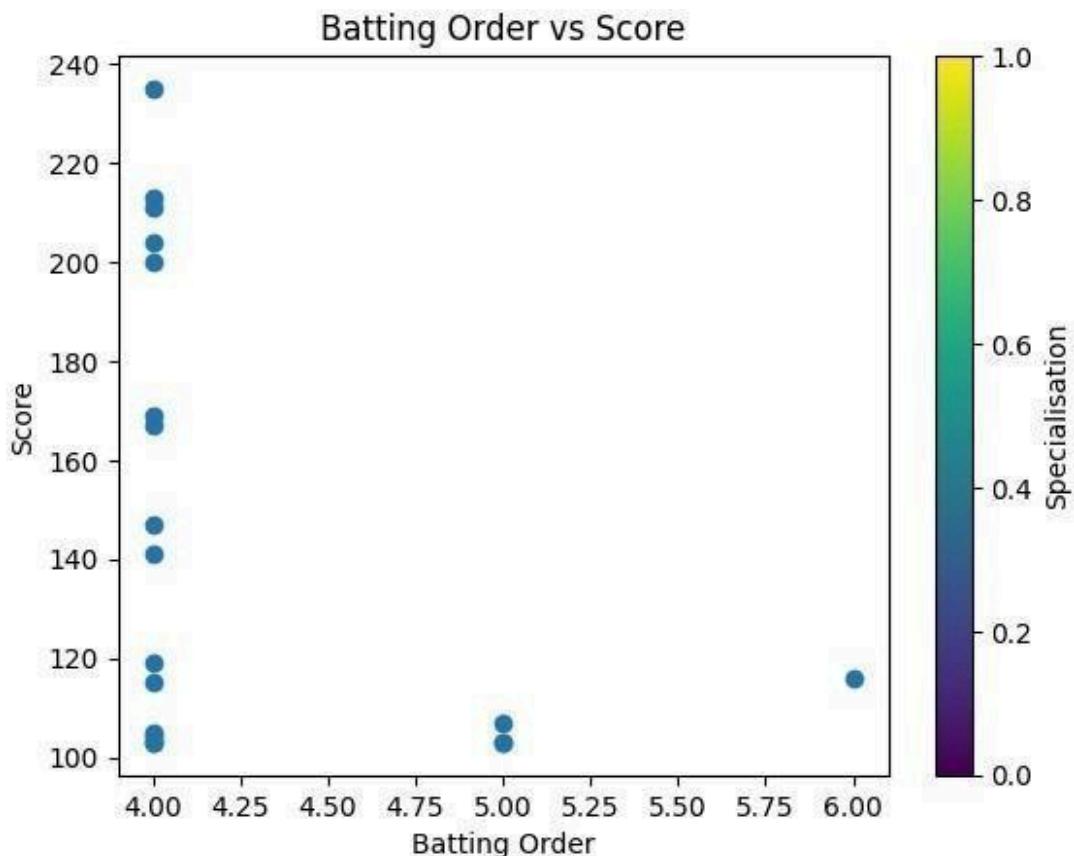
# Create scatter plot
plt.scatter(data['Batting Order'], data['Score'])

# Set title and axis labels
plt.title('Batting Order vs Score')
plt.xlabel('Batting Order')
plt.ylabel('Score')

# Add colorbar
colorbar = plt.colorbar()
colorbar.set_label('Specialisation')

# Display the plot
plt.show()

```



```

fig, ax = plt.subplots()

colors = ['#F18940', '#40C6F1', '#732DCE', '#DA2F67'] # specify
colors for the

bars # create the

bar plot

```

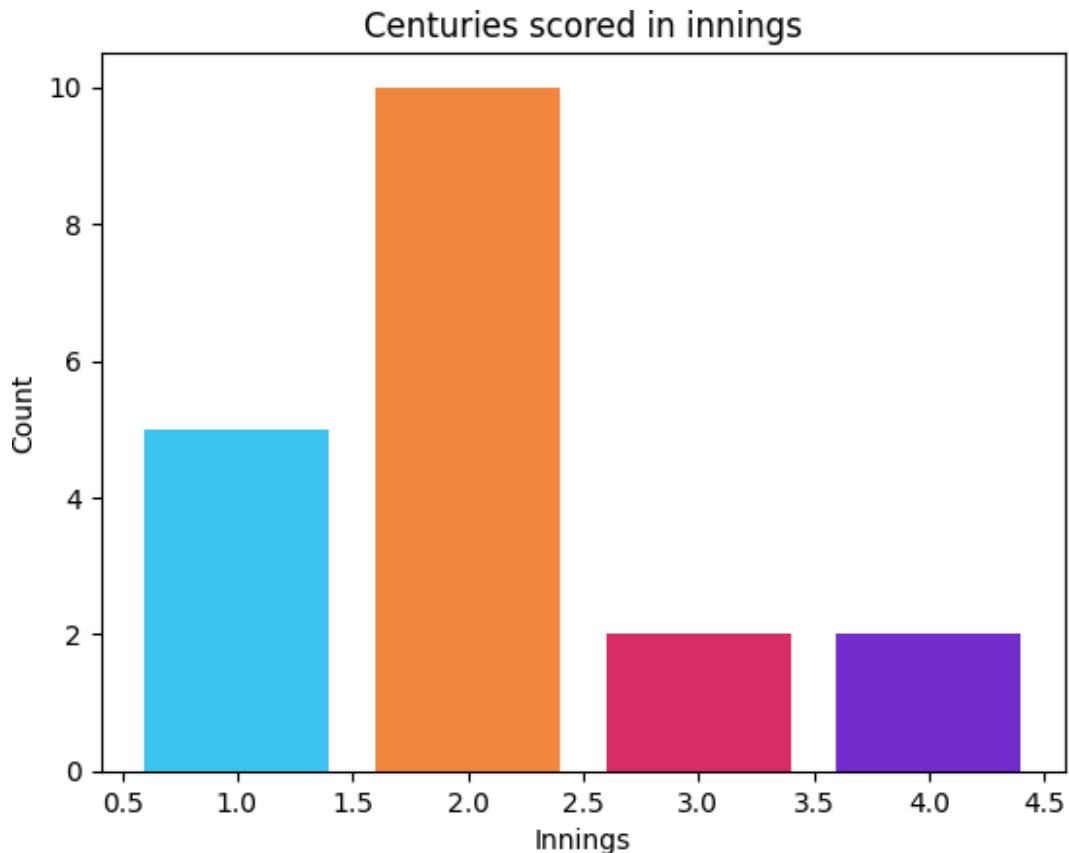
```

ax.bar(data['Inn.'].unique(),
       data['Inn.'].value_counts(), color=colors)

# set the title and axis labels
ax.set_title('Centuries scored in
innings') ax.set_xlabel('Innings')
ax.set_ylabel('Count')

plt.show()

```

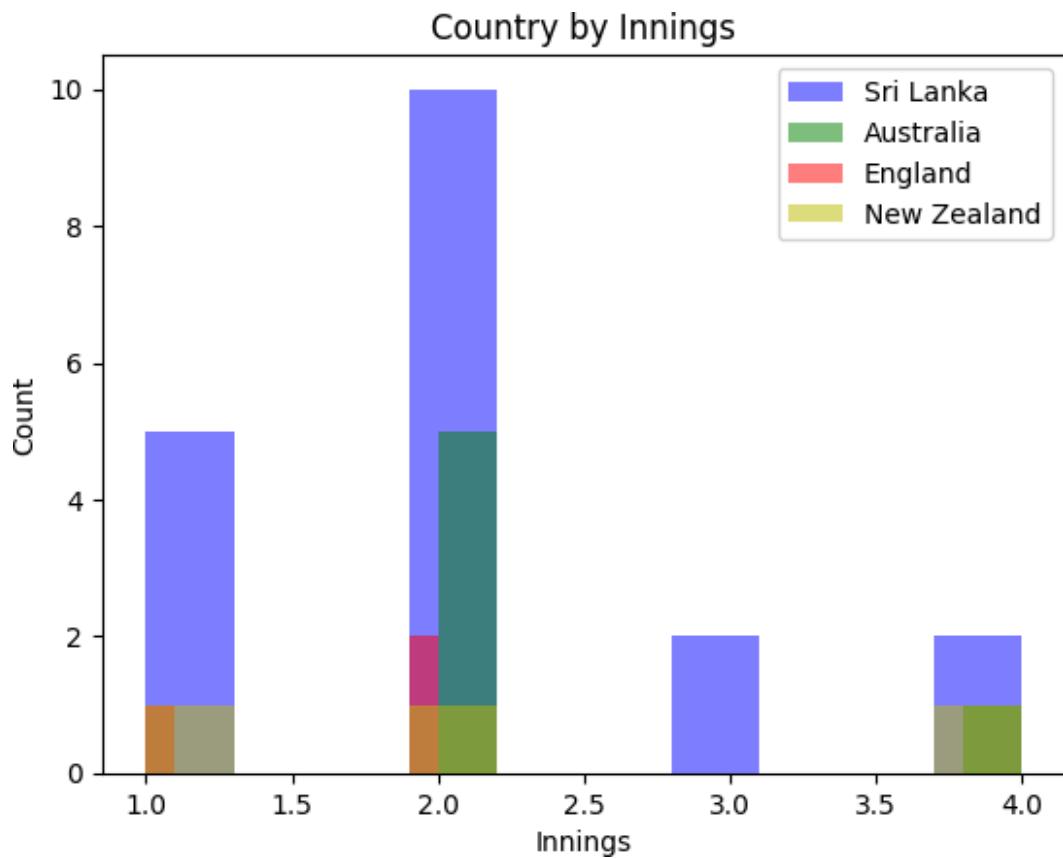


There are more males than females in this dataset. This can cause an imbalance.

```

plt.hist(data['Inn.'], color='b', alpha=0.5, label='Sri Lanka')
plt.hist(data[data['Against'] == 'Australia']['Inn.'], color='g',
alpha=0.5, label='Australia')
plt.hist(data[data['Against'] == 'England']['Inn.'], color='r',
alpha=0.5, label='England')
plt.hist(data[data['Against'] == 'New Zealand']['Inn.'], color='y',
alpha=0.5, label='New Zealand')
plt.xlabel('Innings')
plt.ylabel('Count')
plt.title('Country by Innings')
plt.legend()
plt.show()

```



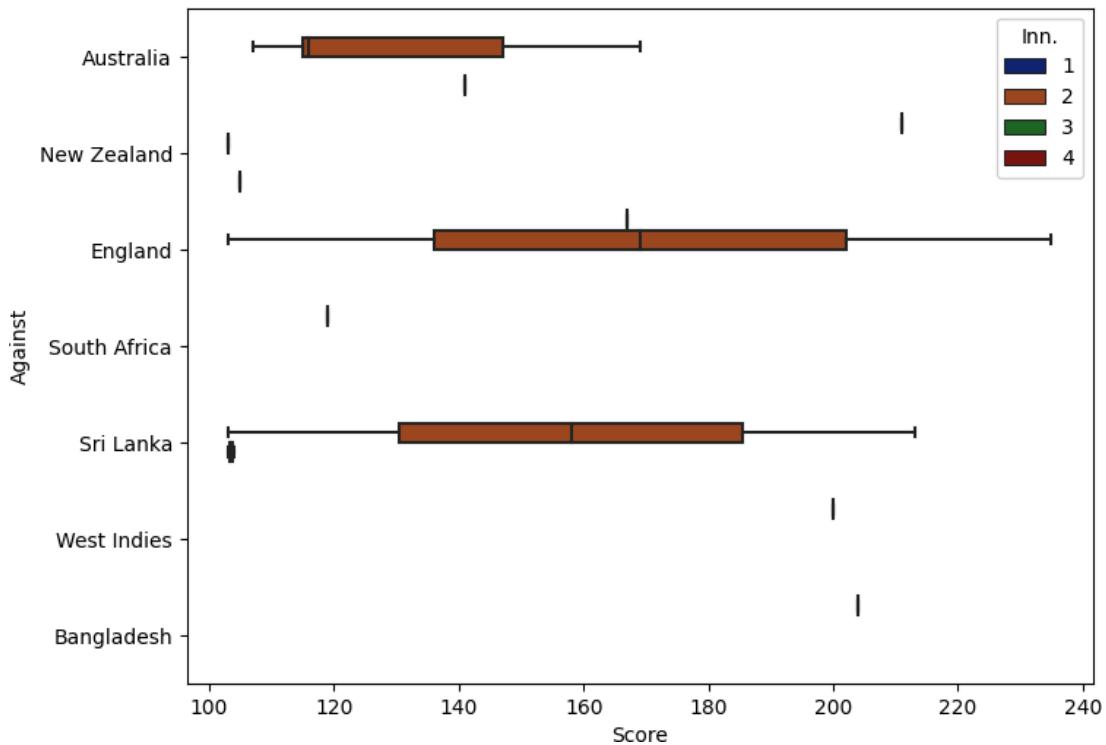
Females have more population in Central-based secondary school compared to Other secondary schools. The same trend is for male. Overall, the male population in secondary schools are more than females, irrespective of the type of school.

```
import seaborn as sns
fig, ax = plt.subplots(figsize=(8, 6))

sns.boxplot(x="Score", y="Against", hue="Inn.", palette="dark",
            data=data, ax=ax)

ax.set_xlabel("Score")
ax.set_ylabel("Against")

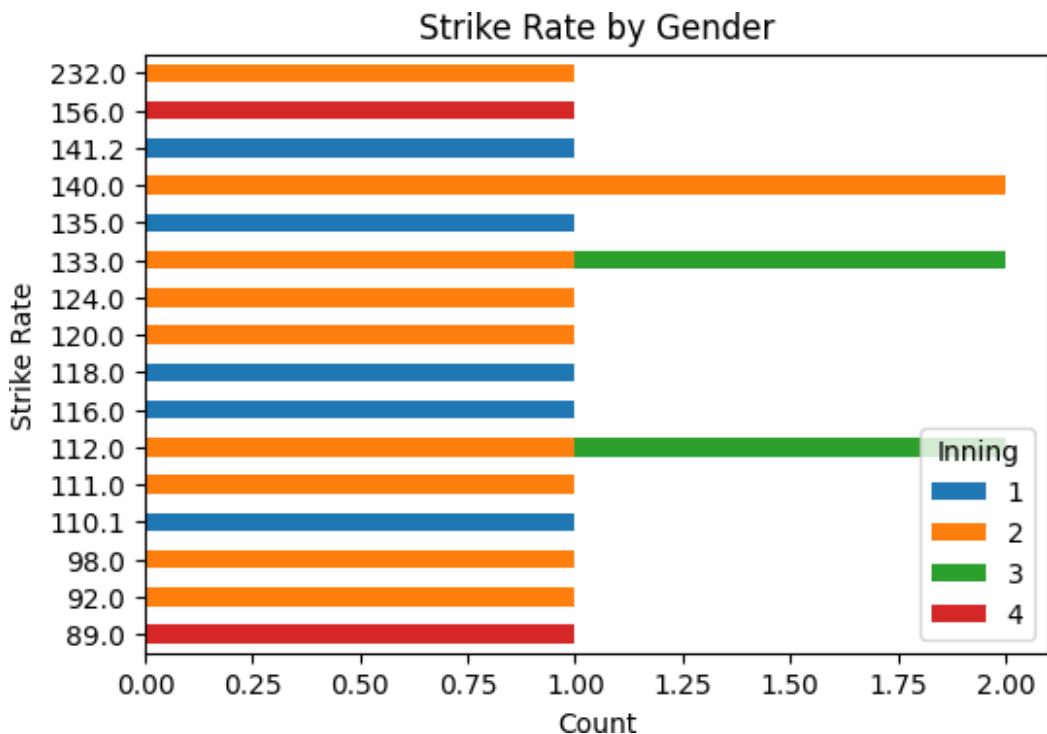
plt.show()
```



The boxplot clearly shows presence of outliers. High school percentage of male is better in Others compared to females. However, in Central, females score more than the female. If we compare the school types, both male and female students perform better in Central.

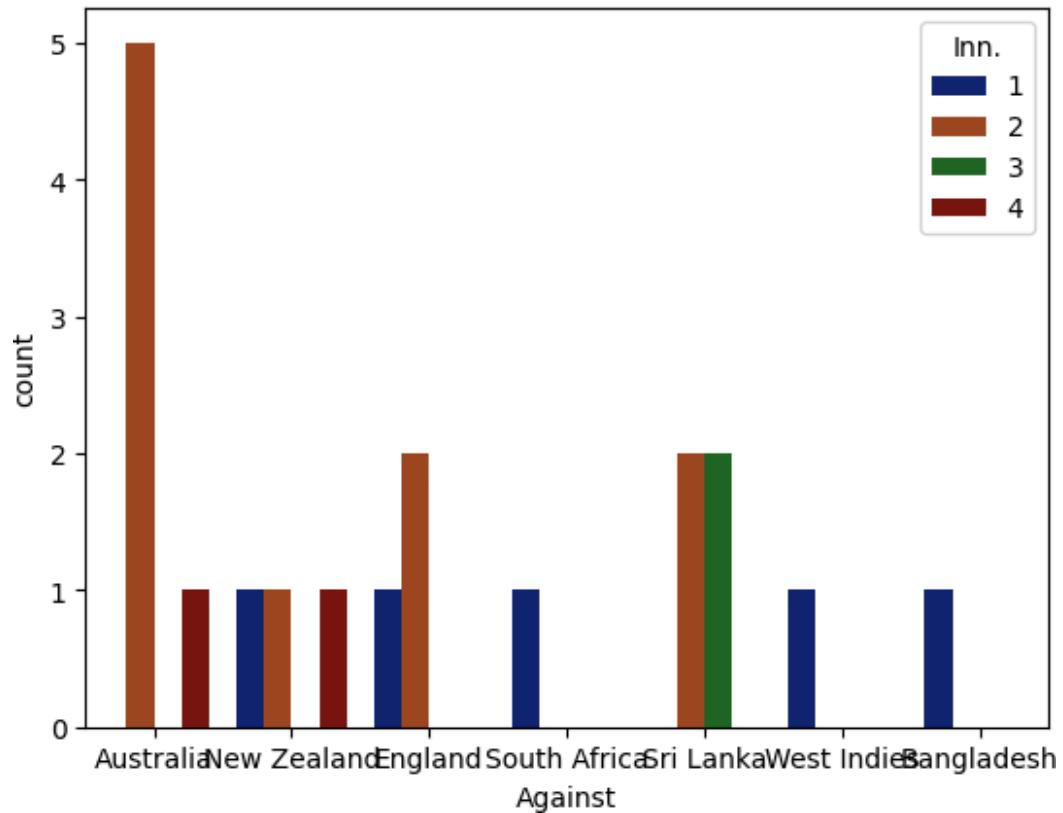
There is more population of female students in degree-level education compared to male students. More students tend to choose SciSTech compared to the CommSMgmt stream. The population of female in individual streams of the degrees is higher.

```
fig, ax = plt.subplots(figsize=(6, 4))
data.groupby(['Strike Rate', 'Inn.']).['Score'].count().unstack().plot(kind='barh', stacked=True, ax=ax)
ax.set_xlabel('Count')
ax.set_ylabel('Strike Rate')
ax.set_title('Strike Rate by Gender')
ax.legend(title='Inning')
plt.show()
```



Overall, most of the students don't tend to have work experience. More population of male have no experience compared to female.

```
sns.countplot(x="Against", hue="Inn.", palette="dark", data=data);
```



More male students are placed compared to female students. The no. of female students placed if half the amount of the no. of male students placed. In Not Placed, this proportion of difference between male and female Not Placed students is not as high as in the Placed scenario.

```

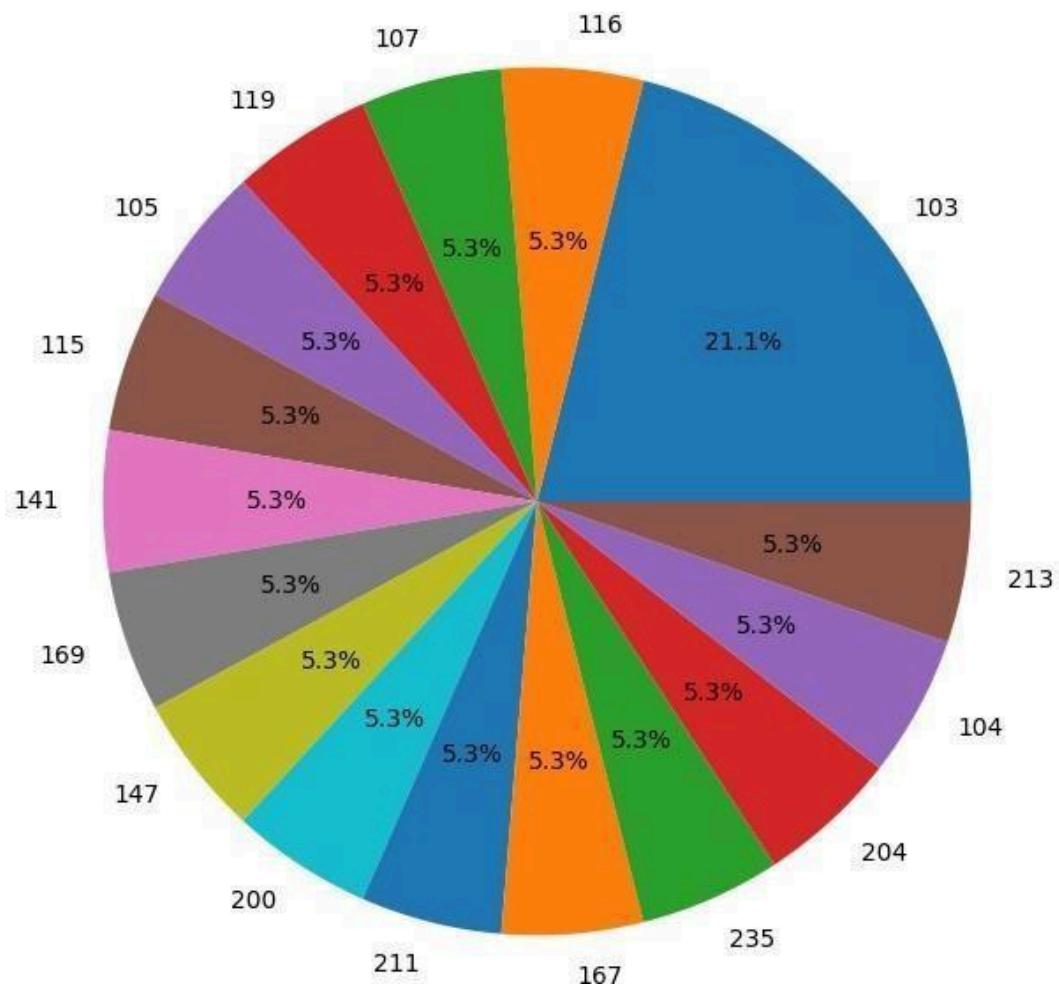
labels =
data['Score'].value_counts().index sizes
= data['Score'].value_counts().values

plt.figure(figsize = (8,8))
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title("Distribution of Samples by 'Score'",color =
'black',fontsize = 15)

Text(0.5, 1.0, "Distribution of Samples by 'Score'")

```

## Distribution of Samples by 'Score'



Follow the instructions to recreate the plots using this data:

### Data

```
import numpy as np
x =
np.arange(0,150)
y = x*2
z = x**3
```

\*\* Import matplotlib.pyplot as plt and set %matplotlib inline if you are using the jupyter notebook. What command do you use if you aren't using the jupyter notebook?\*\*

```
import matplotlib.pyplot as plt
%matplotlib inline
```

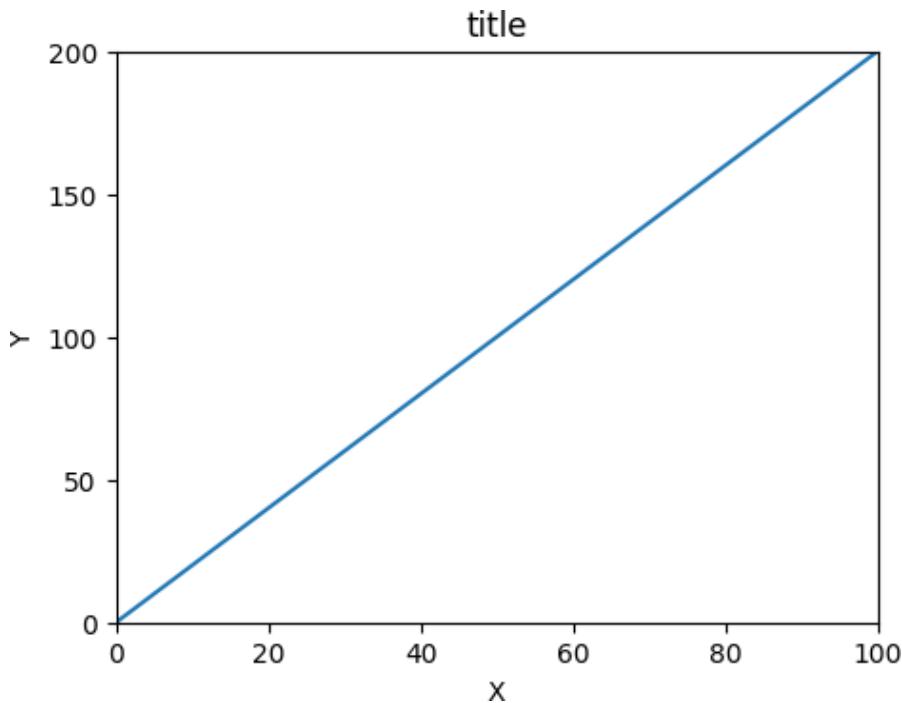
If you are not using Jupyter Notebook and want to display matplotlib plots, you can use the plt.show() command after creating the plot. This command will display the plot in a new window.

## Exercise 1

\*\* Follow along with these steps: \*\*

- \*\* Create a figure object called fig using plt.figure() \*\*
- \*\* Use add\_axes to add an axis to the figure canvas at [0,0,2,2]. Call this new axis ax.  
    \*\*
- \*\* Plot (x,y) on that axes and set the labels and titles to match the plot below:\*\*

```
fig = plt.figure(figsize=(2,  
1.5)) ax =  
fig.add_axes([0,0,2,2])  
ax.plot(x, y)  
ax.set_xlabel('X')  
ax.set_ylabel('Y')  
ax.set_title('title')  
ax.set_xlim([0, 100])  
ax.set_ylim([0, 200])  
ax.set_yticks([0, 50, 100, 150, 200])  
plt.show()
```

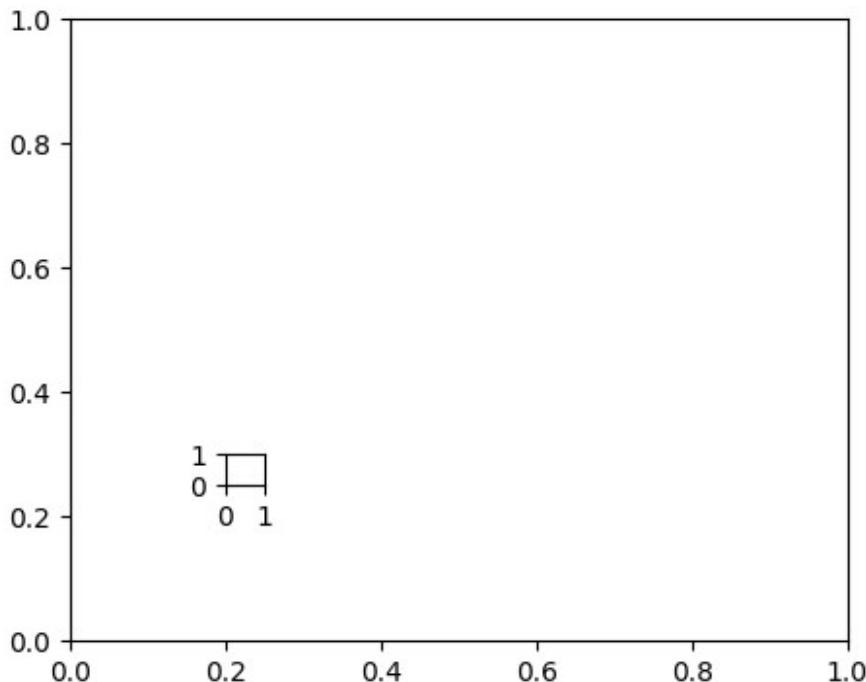


## Exercise 2

\*\* Create a figure object and put two axes on it, ax1 and ax2. Located at [0,0,2,2] and [0.4,0.5,1,1] respectively.\*\*

```
import matplotlib.pyplot as plt
```

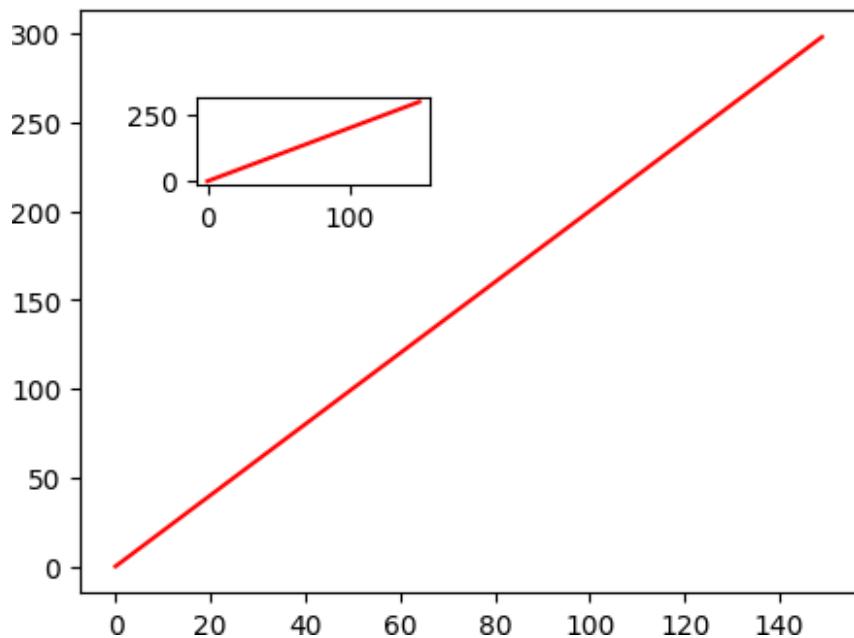
```
fig =  
plt.figure(figsize=(2,1.6)) ax1  
= fig.add_axes([0,0,2,2])  
ax2 = fig.add_axes([0.4,0.5,0.1,0.1])  
plt.show()
```



\*\* Now plot (x,y) on both axes. And call your figure object to show it.\*\*

```
# Create blank canvas  
fig = plt.figure(figsize=(2, 1.5))  
  
# Main Plot  
ax1 = fig.add_axes([0, 0, 2, 2]) #[left, bottom, width, height]  
ax1.plot(x, y, color='red')  
  
# Inset Plot  
ax2 = fig.add_axes([0.3, 1.4, 0.6, 0.3]) #[left, bottom,  
width, height]  
ax2.plot(x, y, color='red')
```

```
[<matplotlib.lines.Line2D at 0x7f8dd1bfff2b0>]
```



### Exercise 3

\*\* Create the plot below by adding two axes to a figure object at [0,0,1,1] and [0.2,0.5,.4,.4]\*\*

```
fig = plt.figure(figsize=(4.1,
3)) ax1 = fig.add_axes([0, 0,
1, 1])
ax1.set_xticks([0.0, 0.2, 0.4, 0.6, 0.8,1.0])
ax2 = fig.add_axes([0.2, 0.5, .4, .4])
ax2.set_xticks([0.0, 0.2, 0.4, 0.6, 0.8,1.0])
ax2.set_yticks([0.0, 0.2, 0.4, 0.6, 0.8,1.0])
plt.show()
```

```
<Figure size 410x300 with 0 Axes>
```

\*\* Now use x,y, and z arrays to recreate the plot below. Notice the xlims and y limits on the inserted plot:\*\*

```
import matplotlib.pyplot as plt
import numpy as np

# Create figure and axis objects
fig, ax = plt.subplots()
ax2 = ax.inset_axes([0.2, 0.6, 0.3, 0.3])
```

```

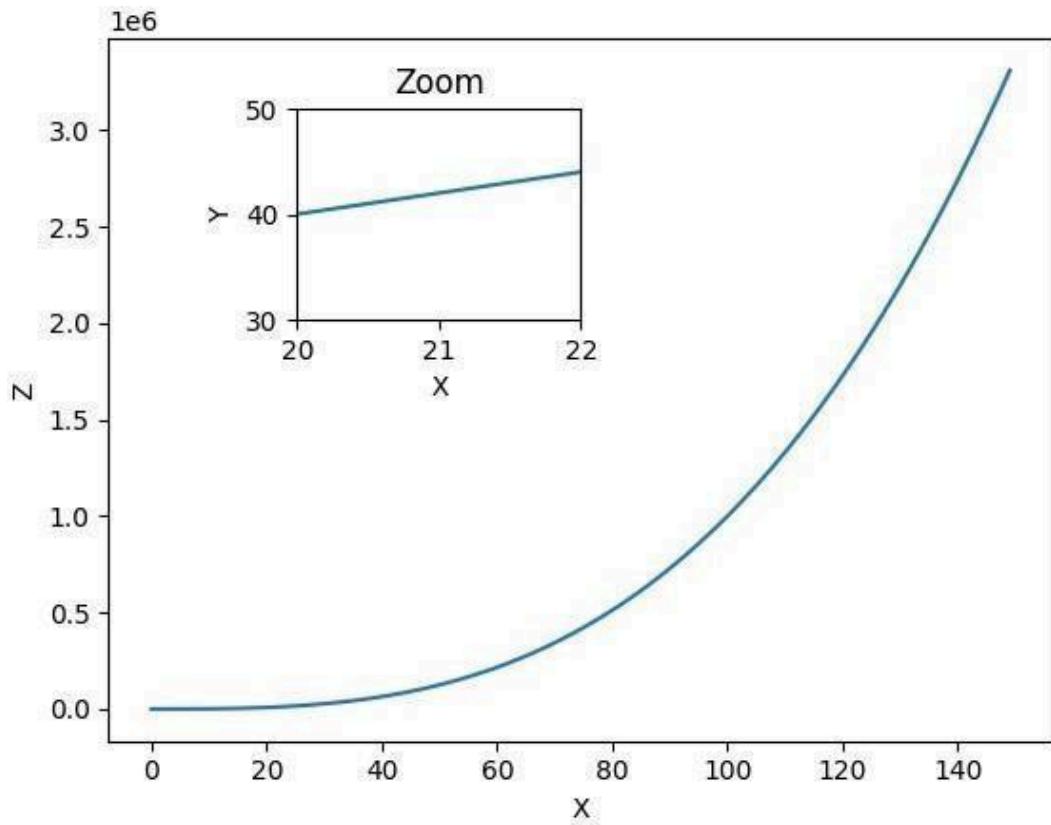
# Define data
x = np.arange(0,150)
y = x**2
z = x**3

# Plot data on main axis
ax.plot(x, z)
ax.set_xlabel('X')
ax.set_ylabel('Z')

# Plot data on inset
axis ax2.plot(x, y)
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_title('Zoom')
ax2.set_xlim(20,22)
ax2.set_ylim(30,50)

# Show the plot
plt.show()

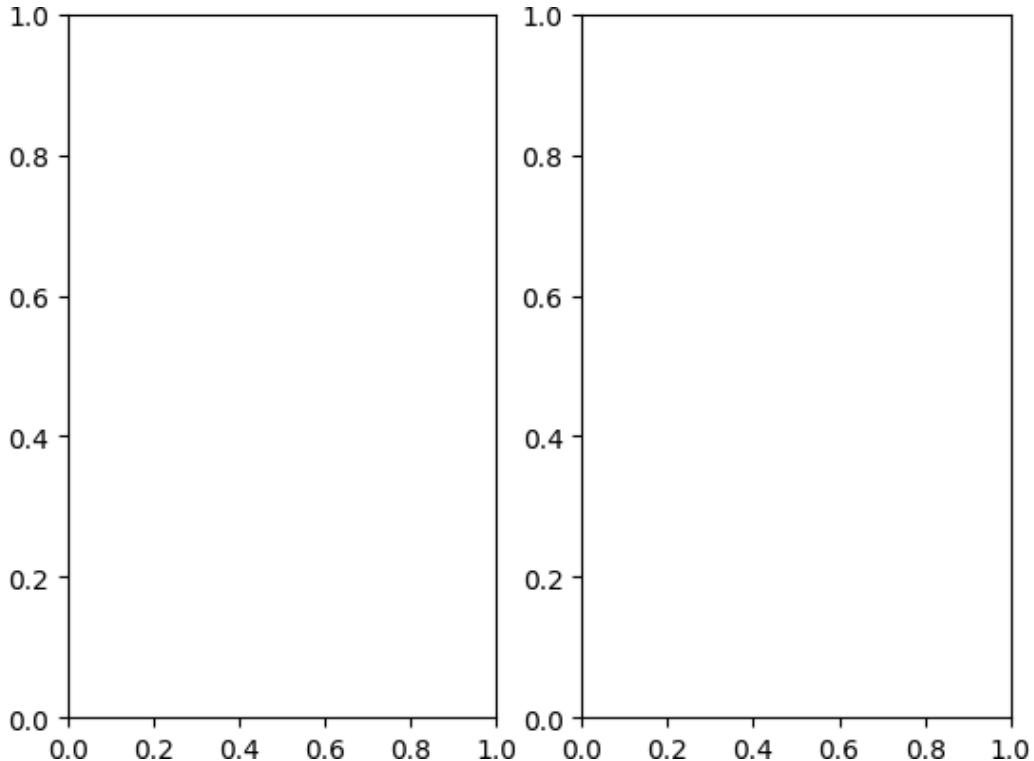
```

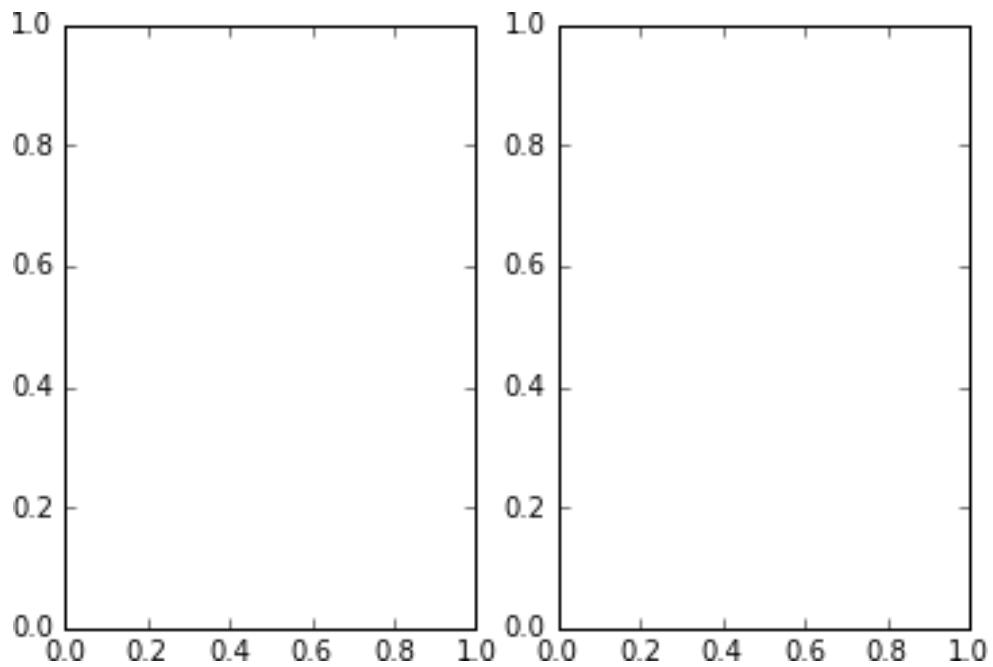


## Exercise 4

\*\* Use plt.subplots(nrows=1, ncols=2) to create the plot below.\*\*

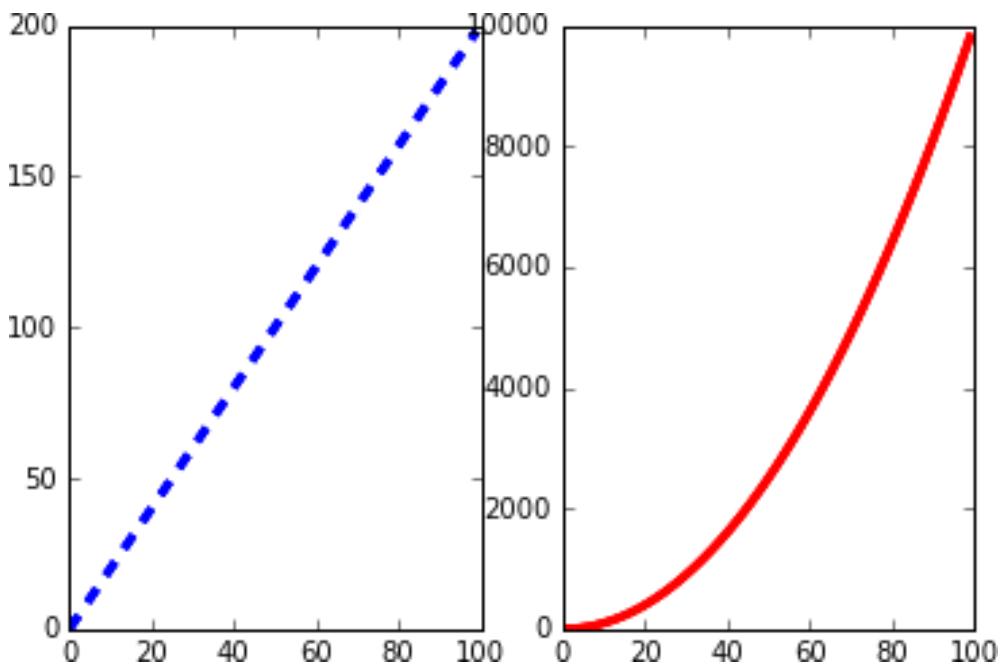
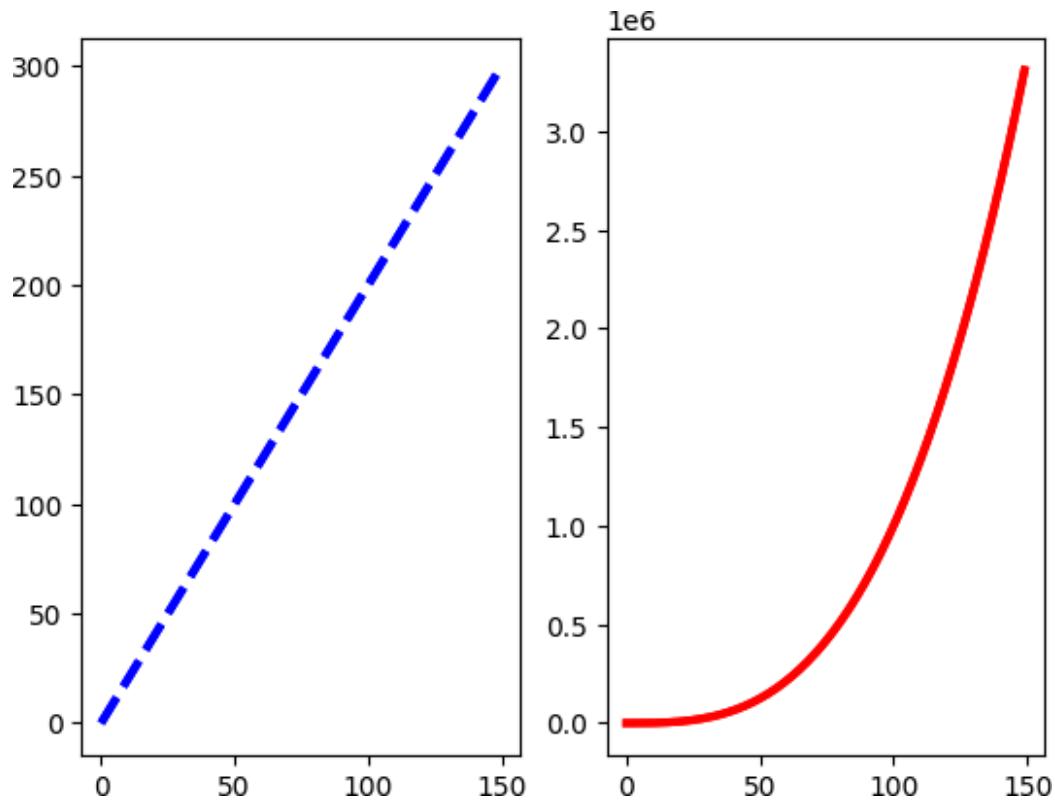
```
fig, axes = plt.subplots(nrows=1, ncols=2)
```





\*\* Now plot (x,y) and (x,z) on the axes. Play around with the linewidth and style\*\*

```
axes[0].plot(x,y,color="blue",lw=3,ls='--')
axes[1].plot(x,z,color="red",lw=3,ls=
'-' ) fig
```



\*\* See if you can resize the plot by adding the figsize() argument in plt.subplots() are copying and pasting your previous code.\*\*

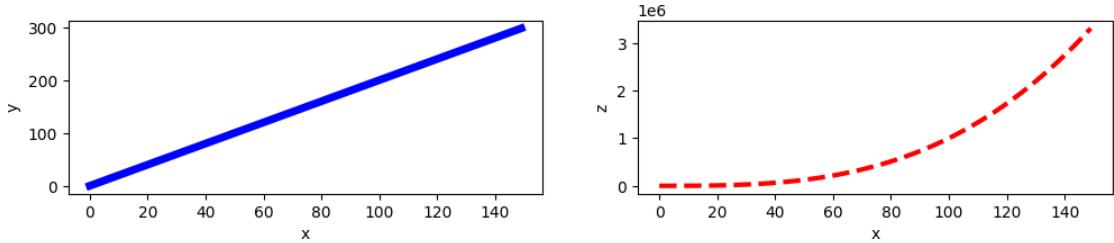
```

fig,axes =
plt.subplots(nrows=1,ncols=2,figsize=(12,2))
axes[0].plot(x,y,color="blue",lw=5)
axes[0].set_xlabel('x')
axes[0].set_ylabel('y')

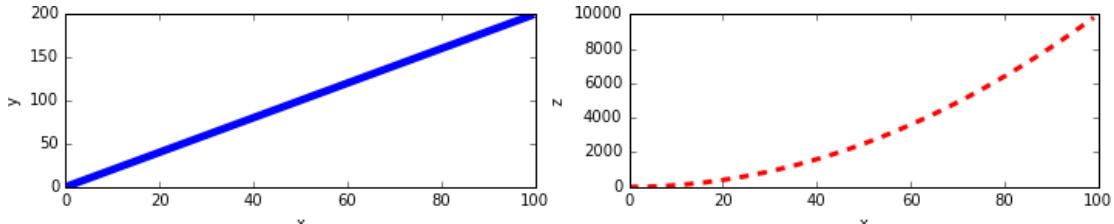
axes[1].plot(x,z,color="red",lw=3,ls='--')
axes[1].set_xlabel('x')
axes[1].set_ylabel('z')

Text(0, 0.5, 'z')

```



<matplotlib.text.Text at 0x1141b4ba8>



*Conclusions: We have successfully implemented matplotlib library in this practical*

## Aim: Implementation of Python Libraries - Seaborn

Seaborn is a popular Python data visualization library built on top of Matplotlib. It provides a high-level interface for creating visually appealing and informative statistical graphics.

Seaborn is designed to work seamlessly with pandas data structures and is particularly well-suited for exploratory data analysis.

Seaborn helps you explore and understand your data. Its plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots. Its dataset-oriented, declarative API lets you focus on what the different elements of your plots mean, rather than on the details of how to draw them.

Some key features and advantages of Seaborn include:

1. High-level interface: Seaborn provides a simple and intuitive API that allows users to create complex plots with minimal code. It abstracts away many of the low-level details, making it easier to create professional-looking visualizations.
2. Statistical visualization: Seaborn offers specialized functions for visualizing statistical relationships and distributions. It provides tools for visualizing categorical variables, examining relationships between variables, and displaying statistical summaries.
3. Integration with pandas: Seaborn seamlessly integrates with pandas, a popular data manipulation library in Python. It can directly plot data stored in pandas data structures, making it convenient for data analysis workflows.
4. Aesthetic customization: Seaborn allows users to easily customize the visual aesthetics of their plots. It provides options to control color palettes, plot styles, and other design elements, enabling users to create visually appealing and cohesive visualizations.
5. Support for complex plots: Seaborn supports a variety of plot types, including scatter plots, line plots, bar plots, box plots, violin plots, heatmaps, and more. It also provides functions for creating multi-panel plots and visualizing complex relationships.
6. Statistical inference: Seaborn incorporates statistical techniques to enhance visualizations. It provides features for visualizing uncertainty, conducting hypothesis tests, and estimating confidence intervals, among others.

Overall, Seaborn is a powerful tool for data visualization in Python, enabling users to quickly explore and understand their data through visually compelling graphics. Whether you're a data scientist, analyst, or researcher, Seaborn can greatly facilitate your data visualization tasks and help communicate insights effectively.

Seaborn is a comprehensive data visualization library in Python that provides various functions and attributes for creating visually appealing and informative statistical graphics. Here's an overview of some commonly used functions and attributes in the Seaborn module:

## 1. Functions:

### a. Data loading:

- `load_dataset()`: Loads built-in datasets from Seaborn.

### b. Plot types:

- `scatterplot()`: Creates a scatter plot to show the relationship between two continuous variables.
- `lineplot()`: Creates a line plot to visualize trends or evolution over time.
- `barplot()`: Creates a bar plot to display the average value of a variable in different categories.
- `countplot()`: Creates a bar plot to visualize the count of observations in each category of a categorical variable.
- `boxplot()`: Creates a box plot to summarize the distribution of a variable across different categories.
- `violinplot()`: Creates a violin plot to display the distribution and density of a variable.
- `distplot()`: Creates a histogram and kernel density estimate plot to visualize the distribution of a continuous variable.
- `heatmap()`: Creates a color-encoded matrix plot (heatmap) to show the correlation between variables.
- `pairplot()`: Creates a matrix of scatter plots to explore pairwise relationships between variables.
- `lmplot()`: Creates a linear regression plot to examine the relationship between two variables with a regression line.
- `jointplot()`: Creates a combination of scatter plot and histograms to visualize the joint distribution of two variables.

### c. Style and aesthetics:

- `set()` or `set_style()`: Sets the aesthetic style of plots.
- `set_palette()`: Sets the color palette for plots.
- `color_palette()`: Generates a color palette with specified colors or color codes.
- `despine()`: Removes the spines (borders) from plots.

### d. Statistical analysis:

- `regplot()`: Creates a scatter plot with a regression line and confidence interval.

- `lmplot()`: Provides options for fitting and visualizing linear regression models.
- `pairplot()`: Provides options for calculating and visualizing pairwise correlations between variables.

e. Other utility functions:

- `catplot()`: Creates a categorical plot that can be customized for different plot types.
- `displot()`: Creates a flexible and customizable distribution plot.
- `rugplot()`: Adds rug plots (small vertical lines) to an axis to show the distribution of a variable.

2. Attributes:

a. Color-related attributes:

- `color_palette()`: Specifies or retrieves the current color palette.
- `set_palette()`: Sets the color palette for the current Seaborn session.
- `cubebeelix_palette()`: Generates a colormap using a cubehelix system.

b. Style-related attributes:

- `set()` or `set_style()`: Sets the overall aesthetic style of plots.
- `axes_style()`: Retrieves or sets the visual style of the axes.
- `plotting_context()`: Retrieves or sets the context for plot elements. Certainly! Here are a few more commonly used functions and attributes in the Seaborn module:

3. Functions (continued):

f. Axis customization:

- `set_title()`: Sets the title of a plot.
- `set_xlabel()`: Sets the x-axis label of a plot.
- `set_ylabel()`: Sets the y-axis label of a plot.
- `set_xticks()`: Sets the tick locations on the x-axis.
- `set_yticks()`: Sets the tick locations on the y-axis.

g. Color palettes and color-related functions:

- `cubebeelix_palette()`: Generates a colormap using a cubehelix system.
- `light_palette()`: Generates a colormap with light colors.
- `dark_palette()`: Generates a colormap with dark colors.
- `color_palette()`: Generates a custom color palette.

h. Grids and subplots:

- `FacetGrid()`: Provides a multi-plot grid for plotting conditional relationships.
- `PairGrid()`: Provides a grid of subplots for plotting pairwise relationships.

- `JointGrid()`: Provides a grid of subplots for visualizing joint distributions.
4. Attributes (continued):
- Style-related attributes (continued):
    - `set_context()`: Sets the scaling factor for plot elements to control the size and scale of plots.
    - `set_color_codes()`: Sets the mapping between semantic categories and color codes.
    - `axes_style()`: Retrieves or sets the visual style of the axes.
    - `plotting_context()`: Retrieves or sets the context for plot elements.
  - Dataset-related attributes:
    - `get_dataset_names()`: Retrieves the list of available built-in datasets in Seaborn.
  - Miscellaneous attributes:
    - `utils.timeseries()` (deprecated): Converts various forms of time series data into a consistent format.

Seaborn provides a rich set of functions and attributes that cater to different aspects of data visualization and statistical analysis. These tools make it easier to create visually compelling and meaningful plots for exploratory data analysis, statistical modeling, and data communication purposes.

## Distribution Plots

Let's discuss some plots that allow us to visualize the distribution of a data set. These plots are:

- `distplot`
  - `jointplot`
  - `pairplot`
  - `rugplot`
  - `kdeplot`
- 

## Imports

```
import seaborn as sns
%matplotlib inline
```

## Data

Seaborn comes with built-in data sets!

```
tips = sns.load_dataset('tips') #loading tips dataset
```

```
tips.head() #displaying top 5 data
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female		No	Sun Dinner	2
1	10.34	1.66		Male	No	Sun Dinner	3
2	21.01	3.50		Male	No	Sun Dinner	3
3	23.68	3.31		Male	No	Sun Dinner	2
4	24.59	3.61	Female		No	Sun Dinner	4

## distplot

The distplot shows the distribution of a univariate set of observations.

```
sns.distplot(tips['total_bill'])  
# Safe to ignore warnings
```

<ipython-input-4-df46eda71b78>:1: UserWarning:

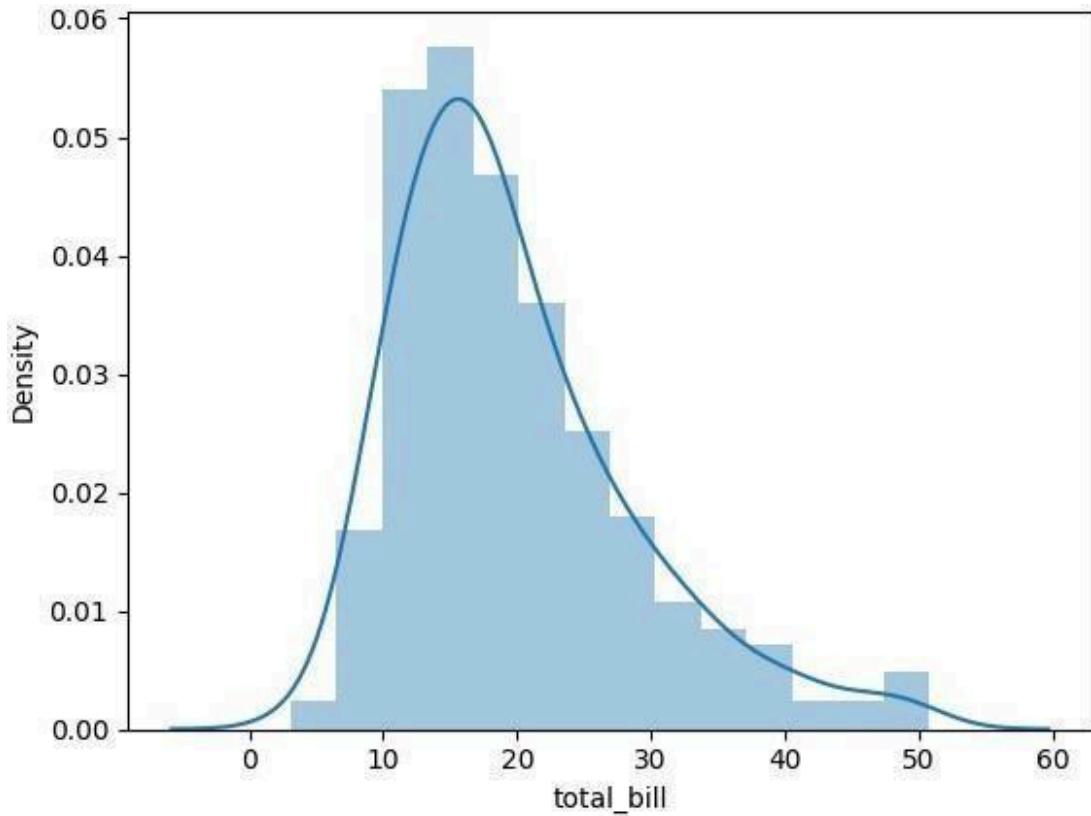
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
sns.distplot(tips['total_bill'])
```

<Axes: xlabel='total\_bill', ylabel='Density'>

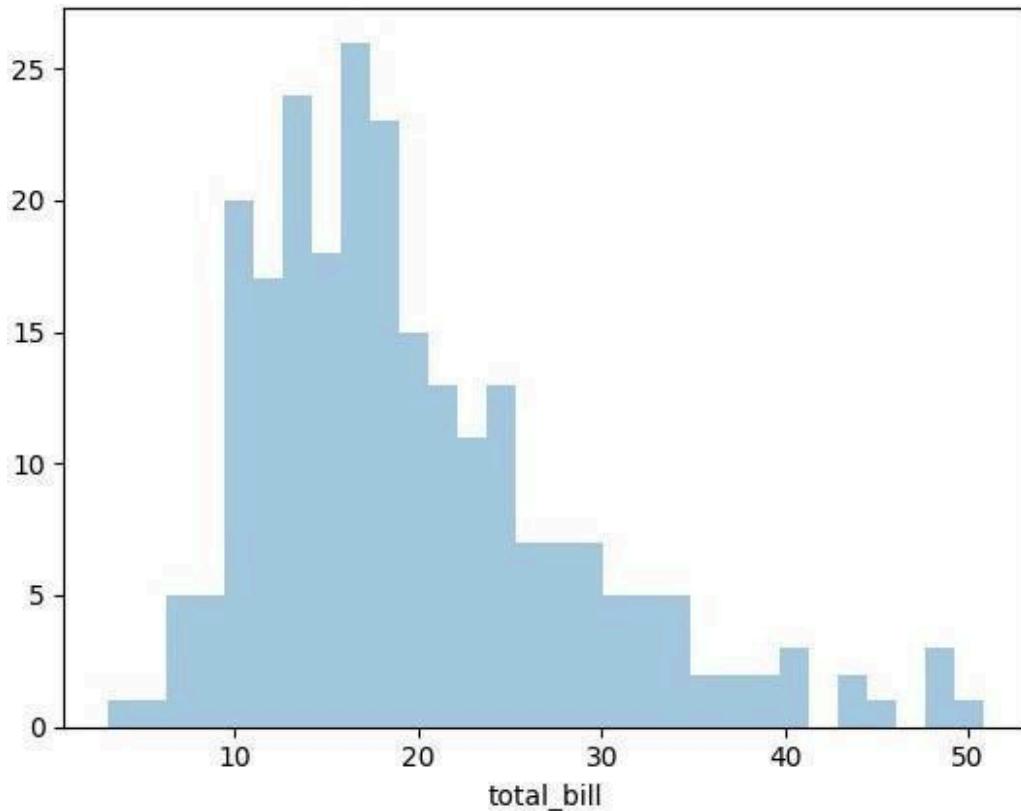


To remove the kde layer and just have the histogram use:

```
sns.distplot(tips['total_bill'], kde=False, bins=30)  
<ipython-input-5-007f6fa19ab1>:1: UserWarning:  
`distplot` is a deprecated function and will be removed in seaborn  
v0.14.0.  
Please adapt your code to use either `displot` (a  
figure-level function with  
similar flexibility) or `histplot` (an axes-level function for  
histograms).
```

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
sns.distplot(tips['total_bill'], kde=False, bins=30)  
<Axes: xlabel='total_bill'>
```

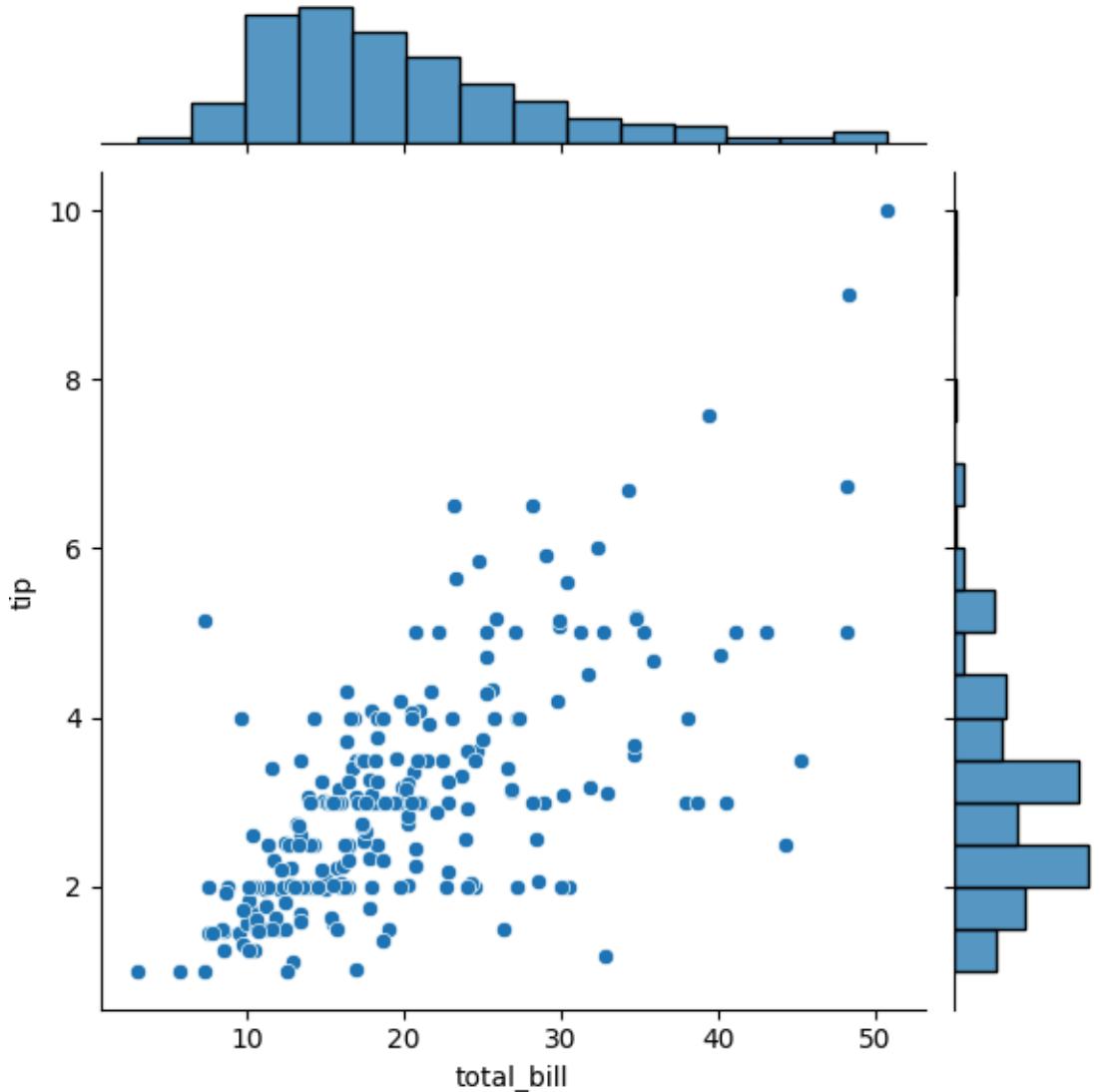


## jointplot

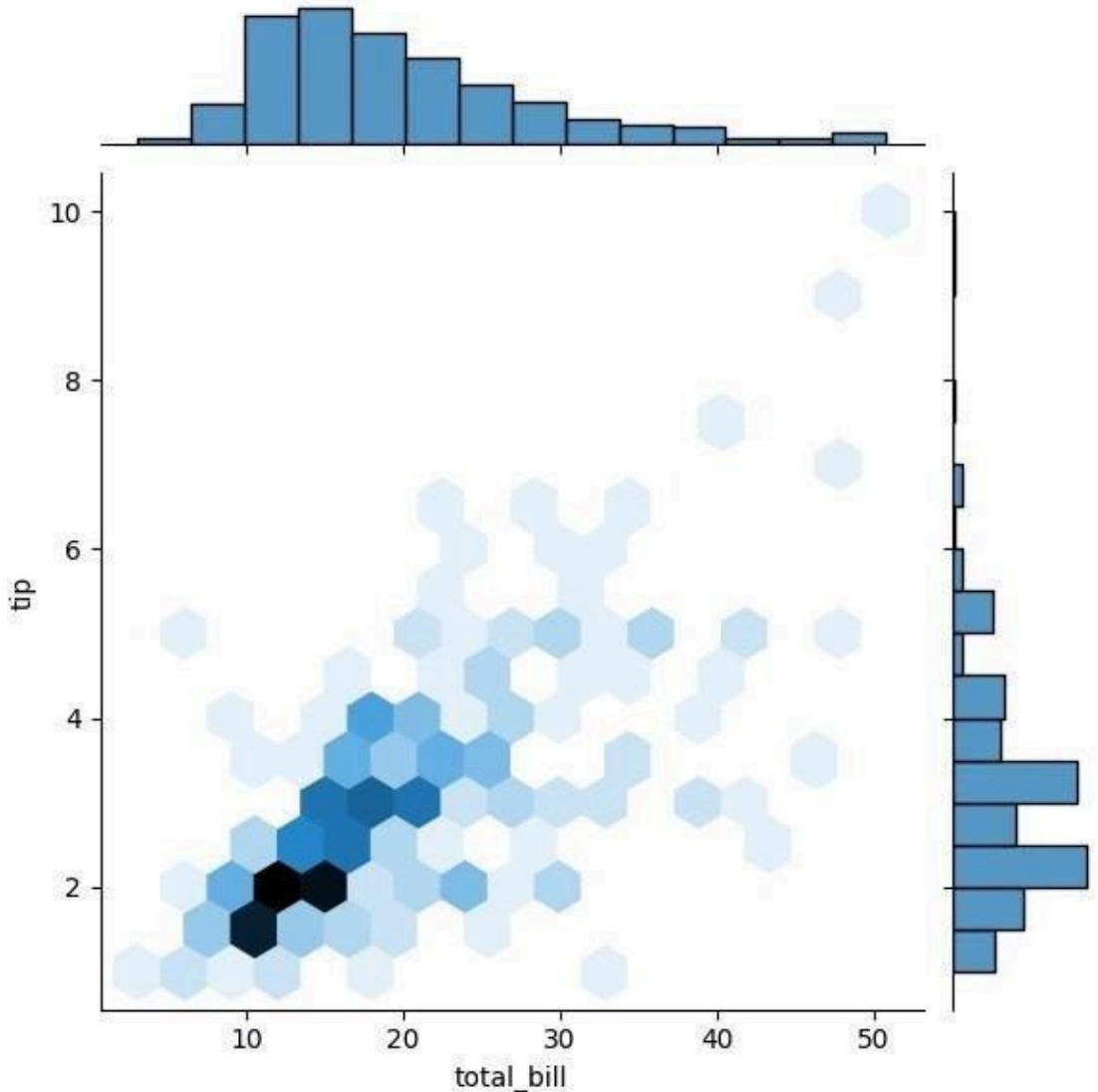
jointplot() allows you to basically match up two distplots for bivariate data. With your choice of what **kind** parameter to compare with:

- “scatter”
- “reg”
- “resid”
- “kde”
- “hex”

```
sns.jointplot(x='total_bill', y='tip', data=tips, kind='scatter')  
<seaborn.axisgrid.JointGrid at 0x7f3c680aa440>
```

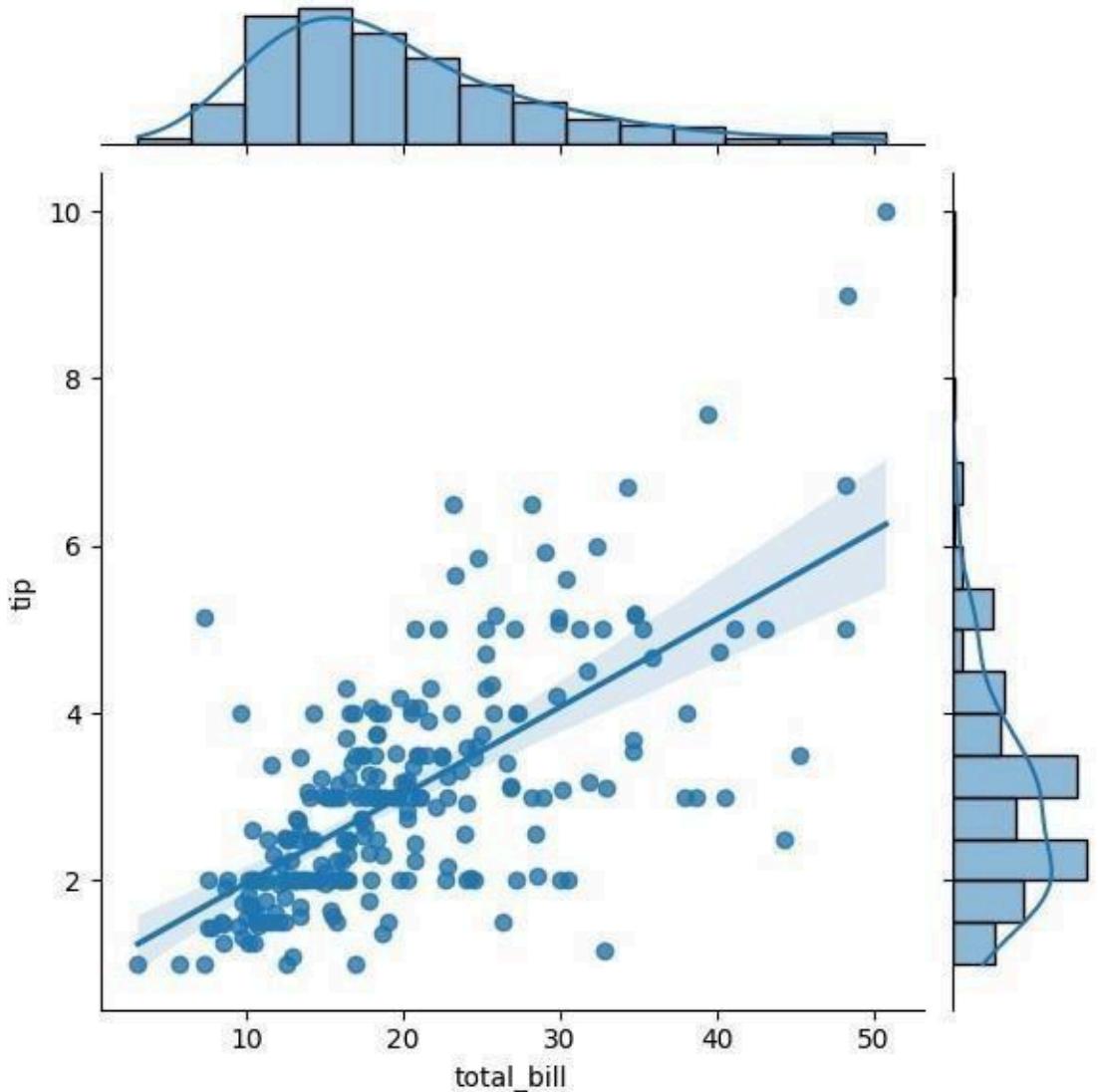


```
sns.jointplot(x='total_bill', y='tip', data=tips, kind='hex')  
<seaborn.axisgrid.JointGrid at 0x7f3c680a9f00>
```



```
sns.jointplot(x='total_bill', y='tip', data=tips, kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7f3c2bc67a90>
```

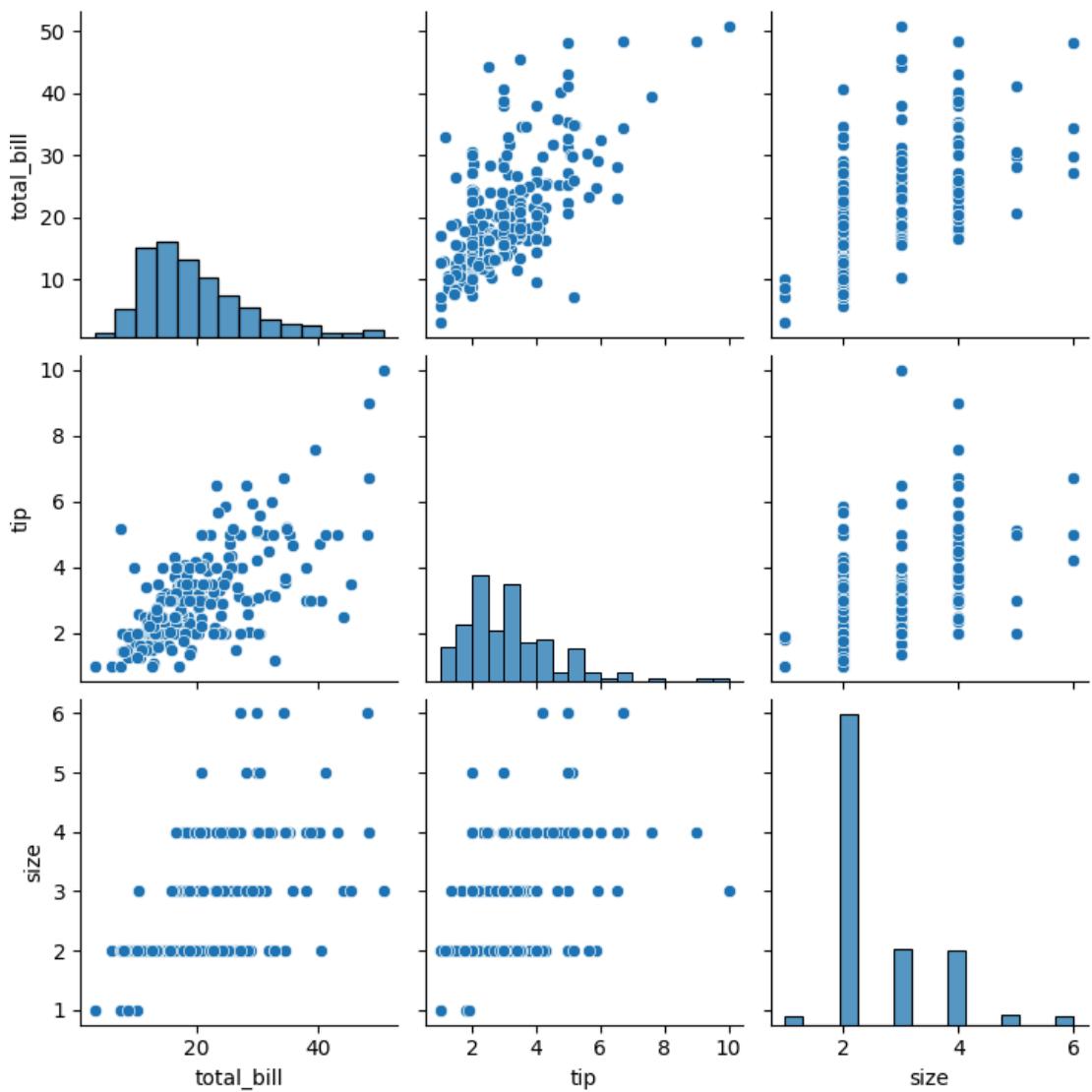


## pairplot

pairplot will plot pairwise relationships across an entire dataframe (for the numerical columns) and supports a color hue argument (for categorical columns).

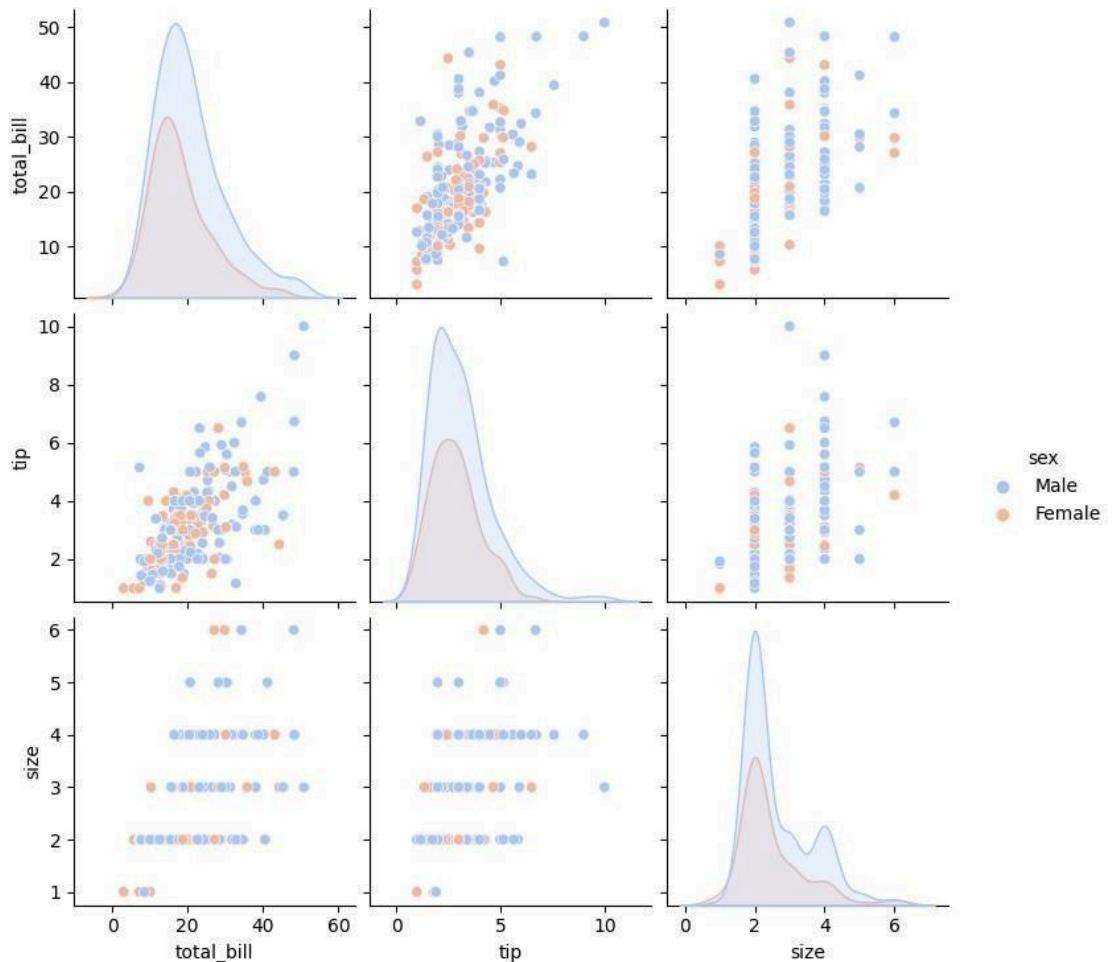
```
sns.pairplot(tips)
```

```
<seaborn.axisgrid.PairGrid at 0x7f3c2b2625c0>
```



```
sns.pairplot(tips,hue='sex',palette='coolwarm')
```

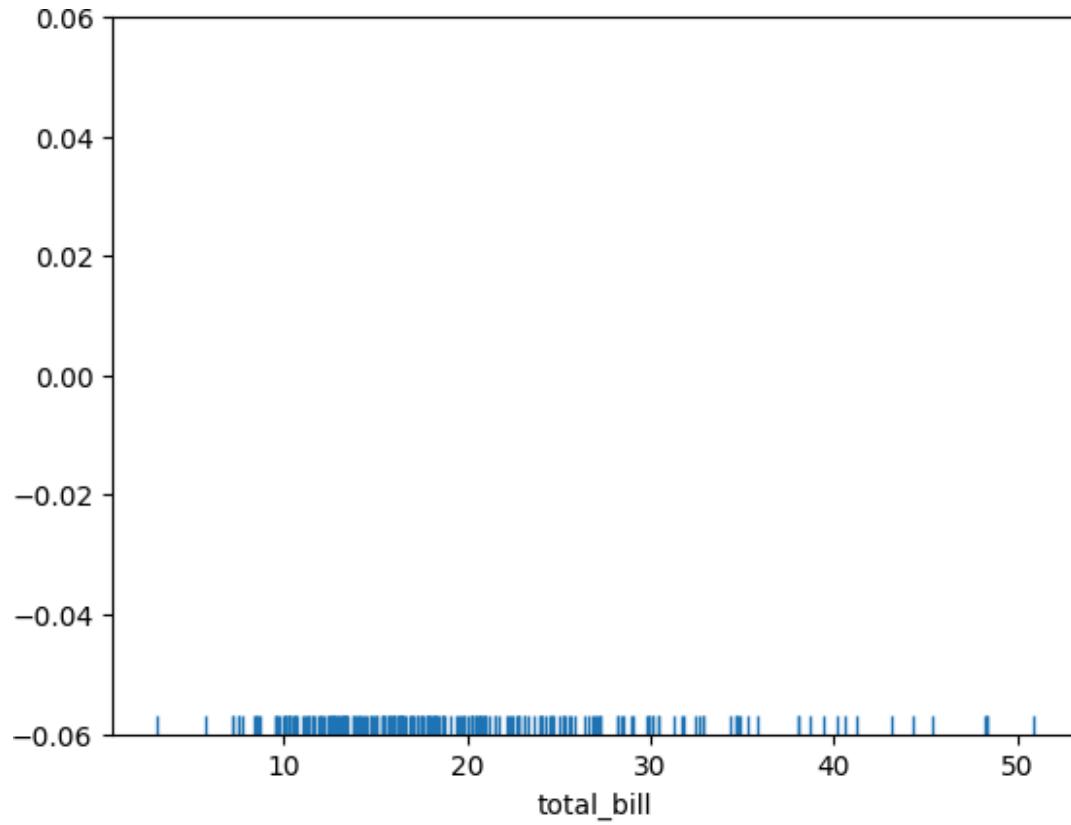
```
<seaborn.axisgrid.PairGrid at 0x7f3c2b2614e0>
```



## rugplot

rugplots are actually a very simple concept, they just draw a dash mark for every point on a univariate distribution. They are the building block of a KDE plot:

```
sns.rugplot(tips['total_bill'])
<Axes: xlabel='total_bill'>
```



## kdeplot

kdeplots are [Kernel Density Estimation plots](#). These KDE plots replace every single observation with a Gaussian (Normal) distribution centered around that value. For example:

```
# Don't worry about understanding this
# code! # It's just for the diagram below
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

#Create dataset
dataset = np.random.randn(25)

# Create another rugplot
sns.rugplot(dataset);

# Set up the x-axis for the plot
x_min = dataset.min() -
2 x_max = dataset.max()
+ 2

# 100 equally spaced points from x_min to x_max
x_axis = np.linspace(x_min,x_max,100)
```

```
# Set up the bandwidth, for info on this:
url =
'http://en.wikipedia.org/wiki/Kernel_density_estimation#Practical_estimation_of_the_bandwidth'

bandwidth = ((4*dataset.std()**5)/(3*len(dataset)))**.2

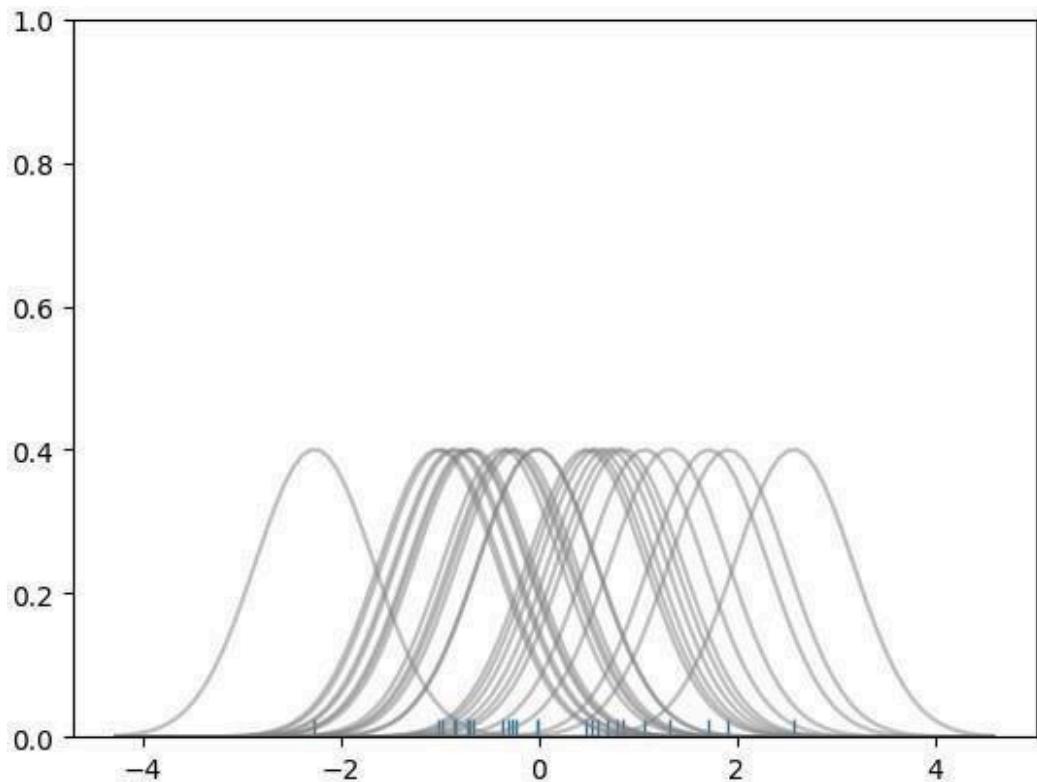
# Create an empty kernel list
kernel_list = []

# Plot each basis function
for data_point in dataset:

    # Create a kernel for each point and append to list
    kernel =
        stats.norm(data_point,bandwidth).pdf(x_axis)
    kernel_list.append(kernel)

    #Scale for plotting
    kernel = kernel /
    kernel.max() kernel =
    kernel * .4
    plt.plot(x_axis,kernel,color = 'grey',alpha=0.5)

plt.ylim(0,1)
(0.0, 1.0)
```



```
# To get the kde plot we can sum these basis
functions. # Plot the sum of the basis function
sum_of_kde = np.sum(kernel_list, axis=0)

# Plot figure
fig = plt.plot(x_axis, sum_of_kde, color='indianred')

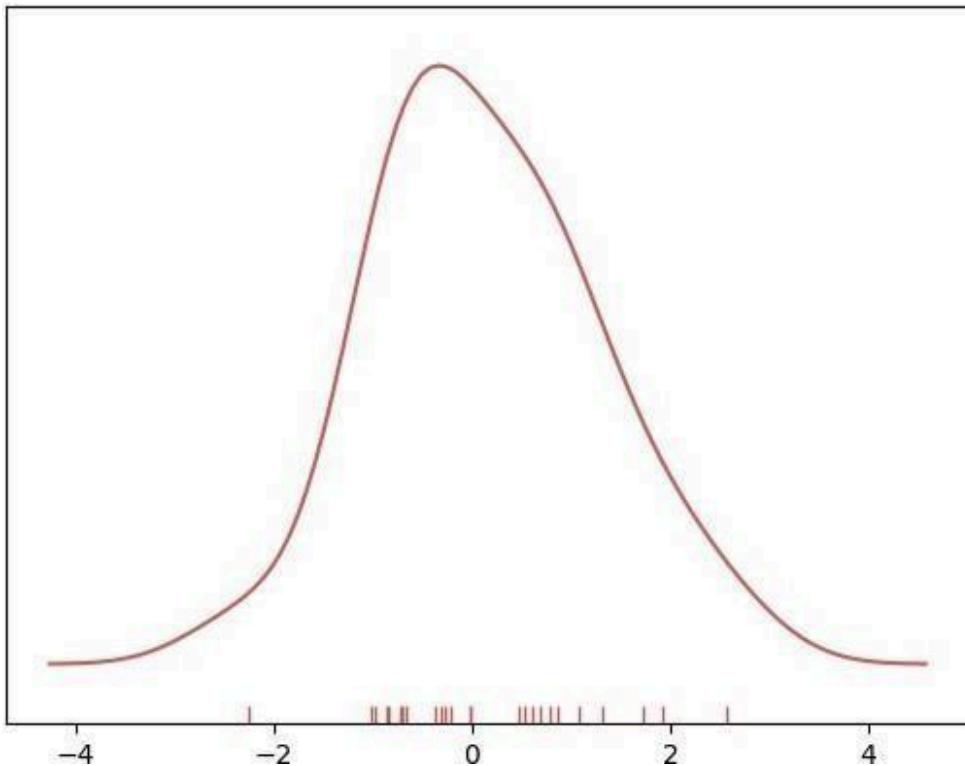
# Add the initial rugplot
sns.rugplot(dataset, c = 'indianred')

# Get rid of y-tick marks
plt.yticks([])

# Set title
plt.suptitle("Sum of the Basis Functions")

Text(0.5, 0.98, 'Sum of the Basis Functions')
```

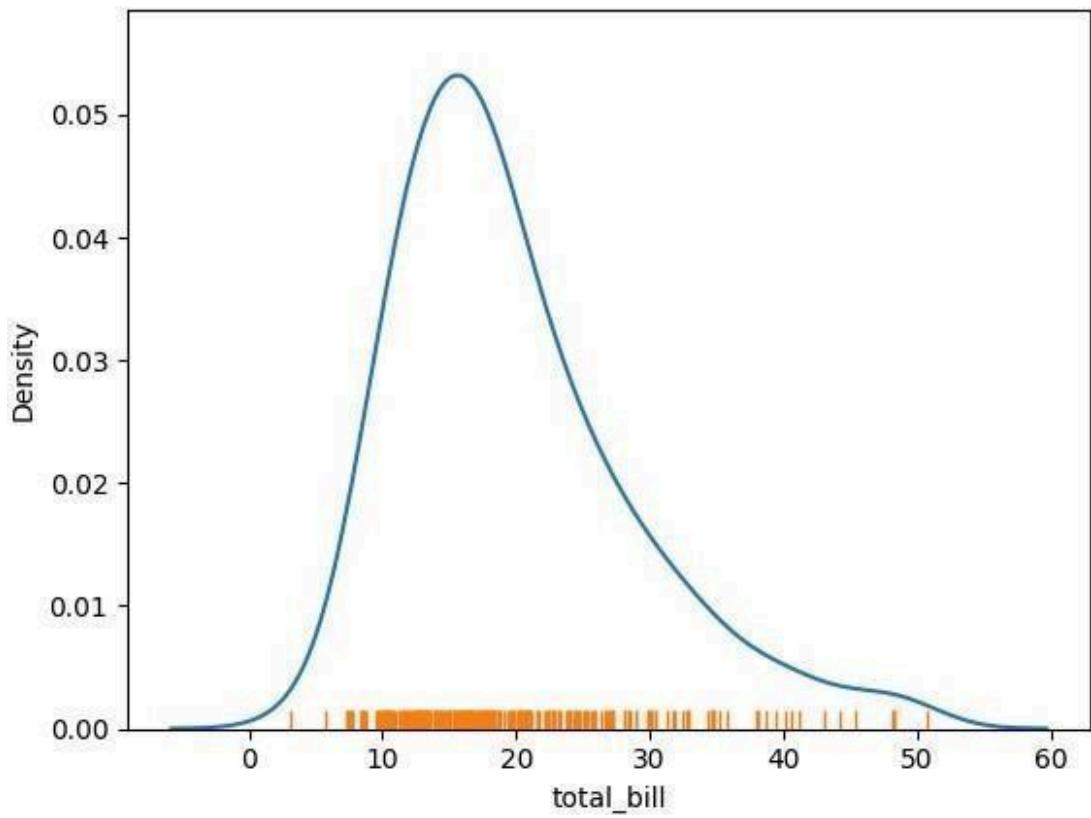
## Sum of the Basis Functions



So with our tips dataset:

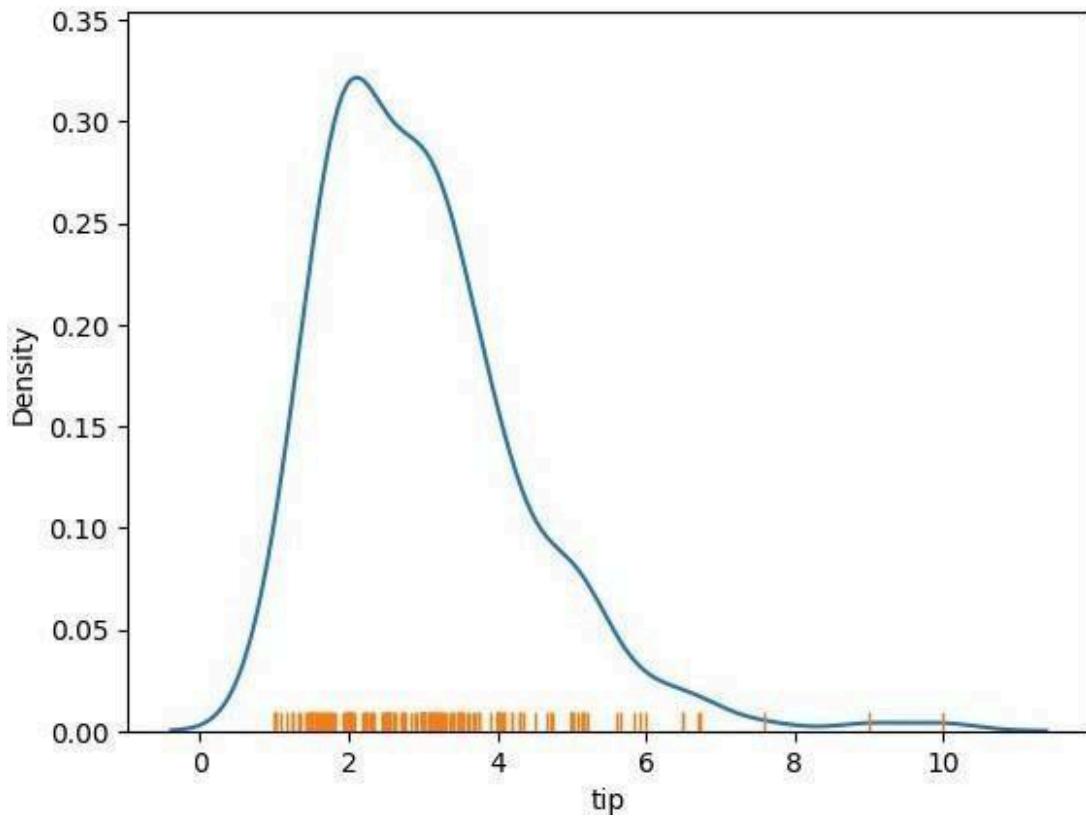
```
sns.kdeplot(tips['total_bill'])
sns.rugplot(tips['total_bill'])

<Axes: xlabel='total_bill', ylabel='Density'>
```



```
sns.kdeplot(tips['tip'])
sns.rugplot(tips['tip'])

<Axes: xlabel='tip', ylabel='Density'>
```



**Great Job!**

---

## Categorical Data Plots

Now let's discuss using seaborn to plot categorical data! There are a few main plot types for this:

- factorplot
- boxplot
- violinplot
- stripplot
- swarmplot
- barplot
- countplot

Let's go through examples of each!

```

import seaborn as sns
%matplotlib inline

tips = sns.load_dataset('tips')
tips.head()

   total_bill  tip    sex smoker day time size
0      16.99  1.01  Female     No Sun Dinner    2
1      10.34  1.66    Male     No Sun Dinner    3
2      21.01  3.50    Male     No Sun Dinner    3
3      23.68  3.31    Male     No Sun Dinner    2
4      24.59  3.61  Female     No Sun Dinner    4

```

## barplot and countplot

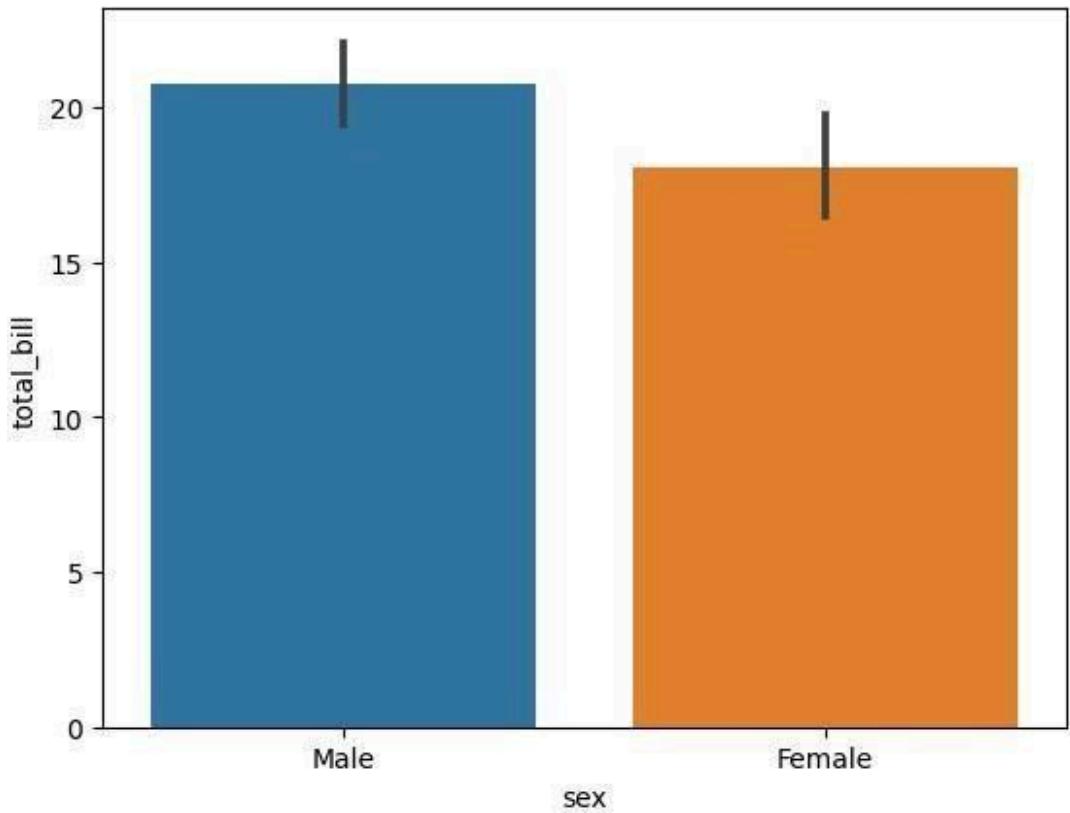
These very similar plots allow you to get aggregate data off a categorical feature in your data. **barplot** is a general plot that allows you to aggregate the categorical data based off some function, by default the mean:

```

sns.barplot(x='sex', y='total_bill', data=tips)

<Axes: xlabel='sex', ylabel='total_bill'>

```



```

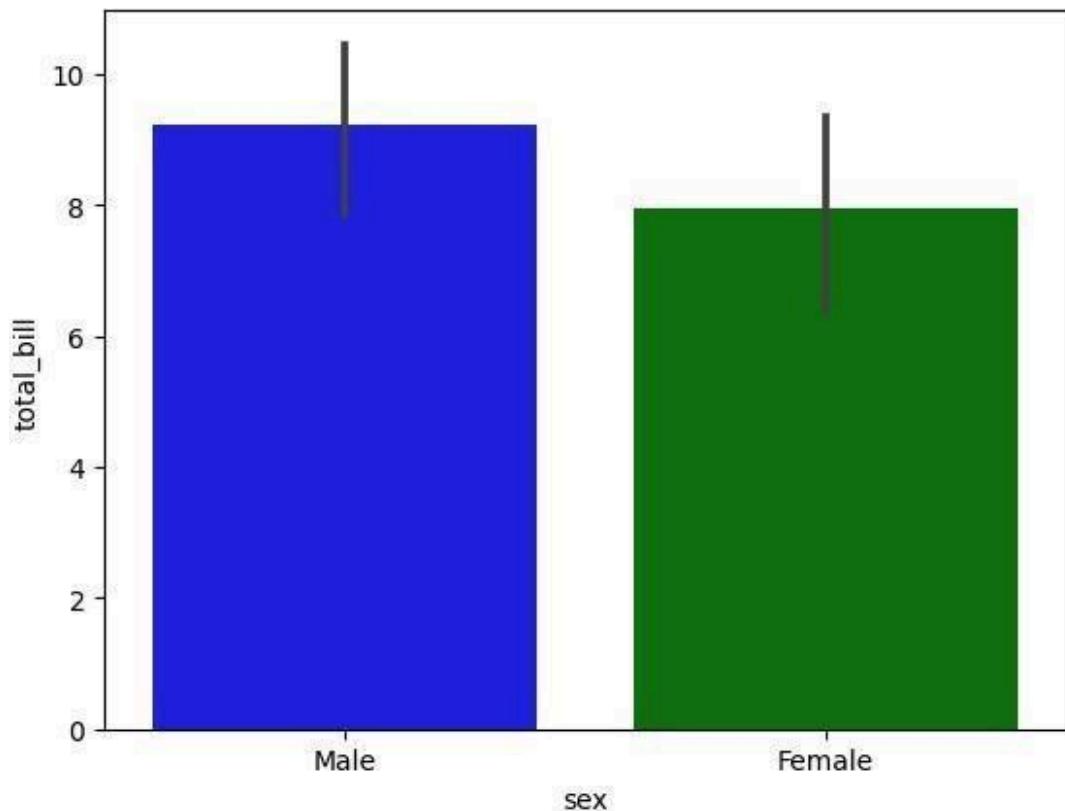
import numpy as np

```

You can change the estimator object to your own function, that converts a vector to a scalar:

```
sns.set_palette(['blue', 'green'])
sns.barplot(x='sex',y='total_bill',data=tips,estimator=np.std)

<Axes: xlabel='sex', ylabel='total_bill'>
```

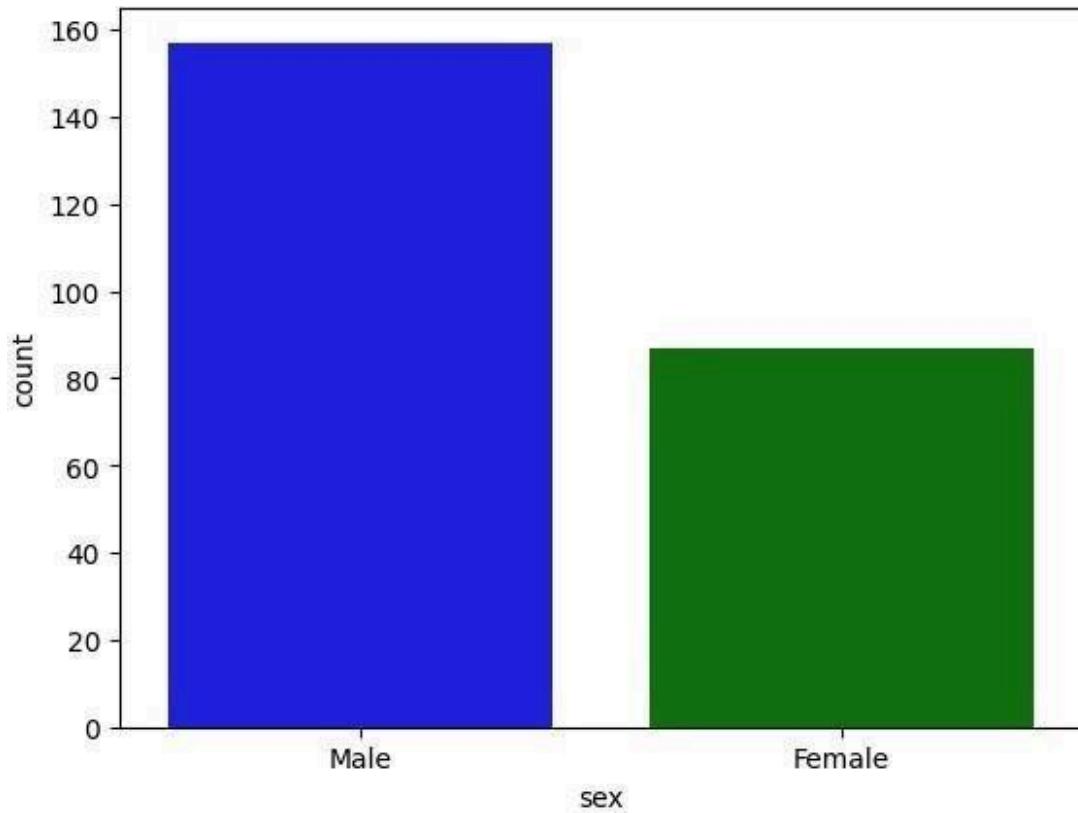


### countplot

This is essentially the same as barplot except the estimator is explicitly counting the number of occurrences. Which is why we only pass the x value:

```
sns.set_palette(['blue',
'green'])
sns.countplot(x='sex',data=tip
s)

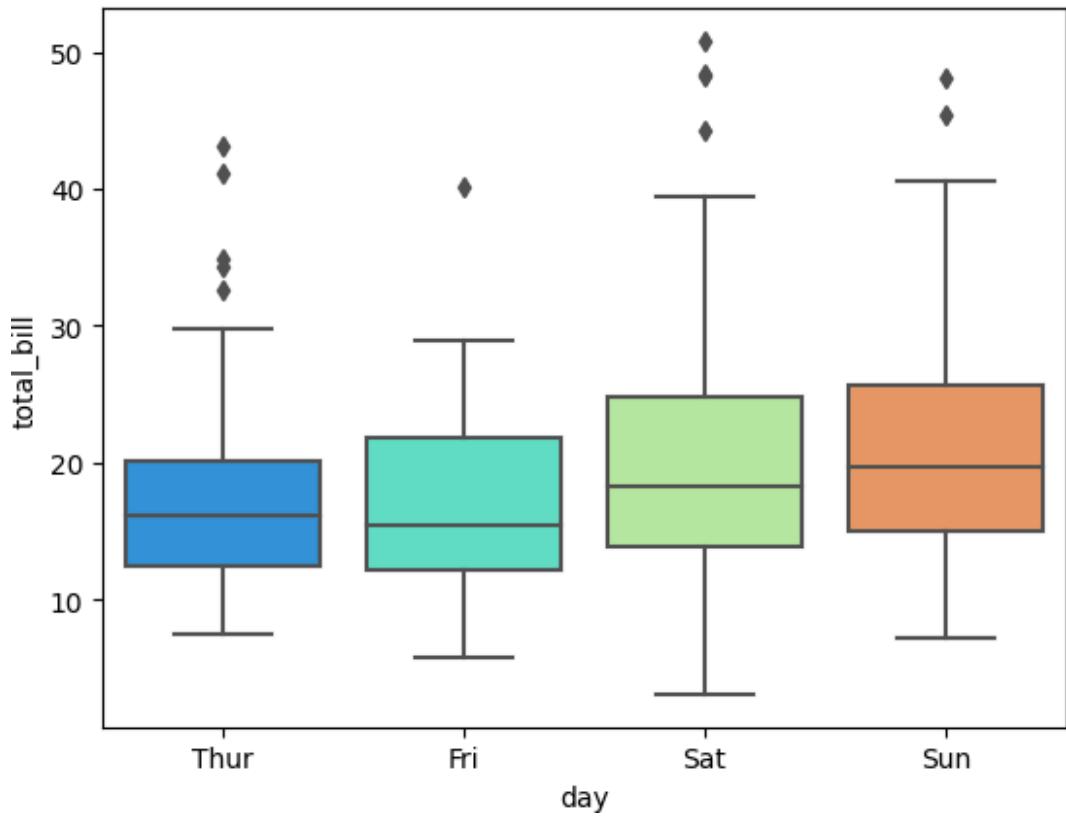
<Axes: xlabel='sex', ylabel='count'>
```



## boxplot and violinplot

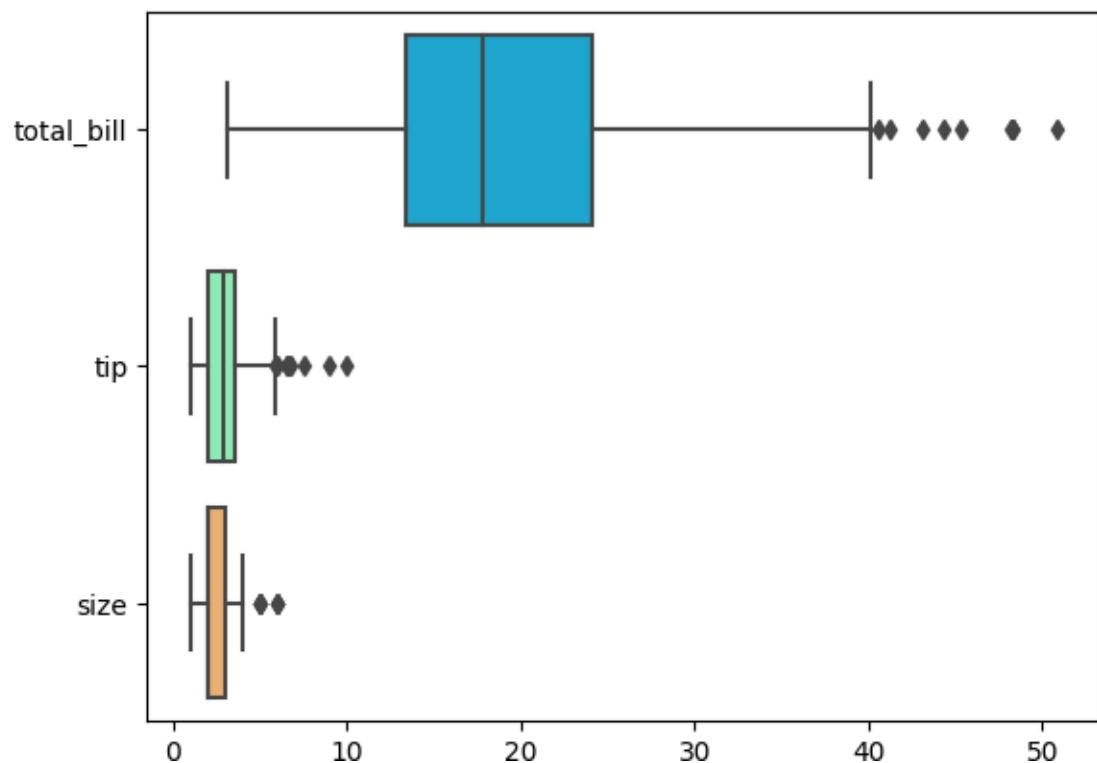
boxplots and violinplots are used to show the distribution of categorical data. A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

```
sns.boxplot(x="day", y="total_bill", data=tips, palette='rainbow')  
<Axes: xlabel='day', ylabel='total_bill'>
```



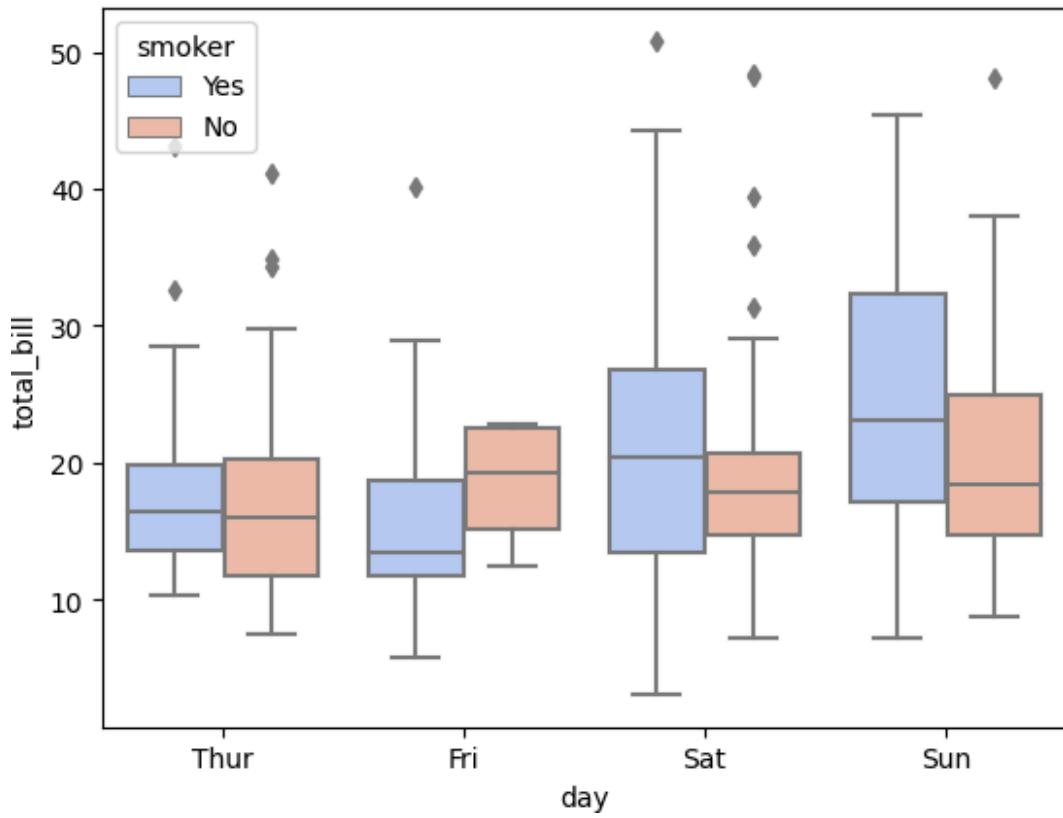
```
# Can do entire dataframe with orient='h'  
sns.boxplot(data=tips,palette='rainbow',orient='h')
```

```
<Axes: >
```



```
sns.boxplot(x="day", y="total_bill", hue="smoker", data=tips,  
palette="coolwarm")
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```

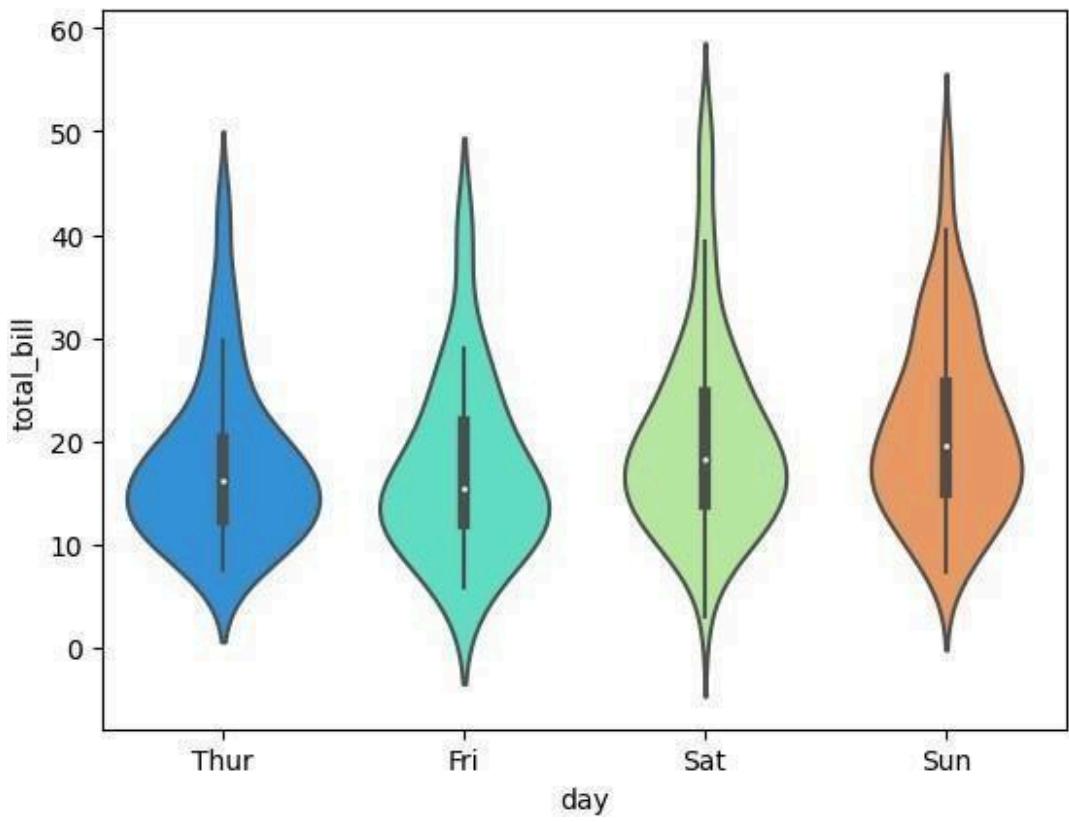


## violinplot

A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a box plot, in which all of the plot components correspond to actual datapoints, the violin plot features a kernel density estimation of the underlying distribution.

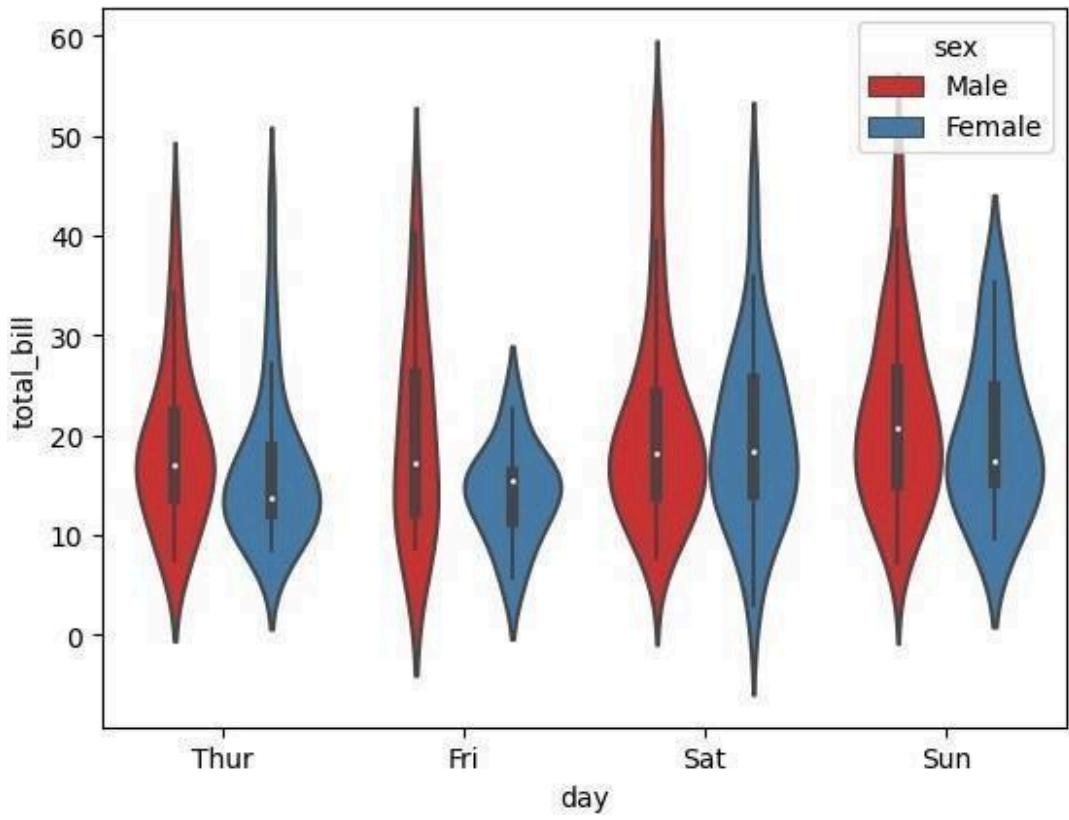
```
sns.violinplot(x="day", y="total_bill", data=tips, palette='rainbow')

<Axes: xlabel='day', ylabel='total_bill'>
```



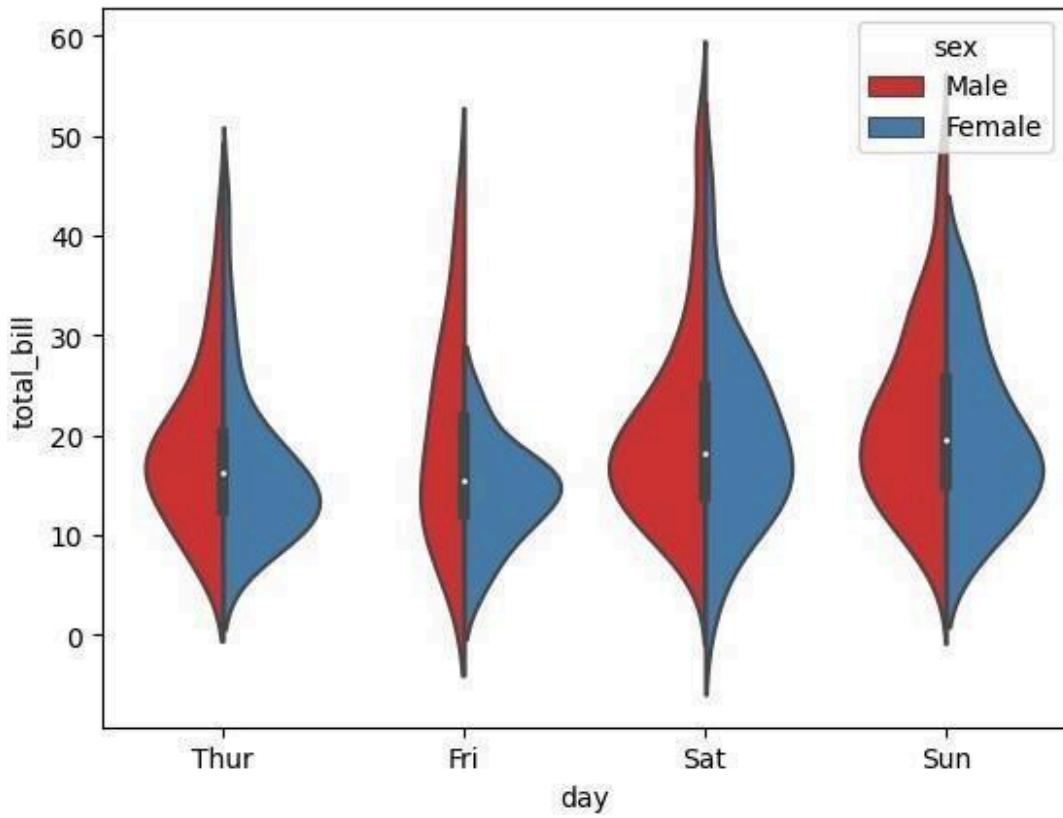
```
sns.violinplot(x="day", y="total_bill",
data=tips,hue='sex',palette='Set1')
```

```
<Axes: xlabel='day', ylabel='total_bill'>
```



```
sns.violinplot(x="day", y="total_bill",
data=tips,hue='sex',split=True,palette='Set1')

<Axes: xlabel='day', ylabel='total_bill'>
```

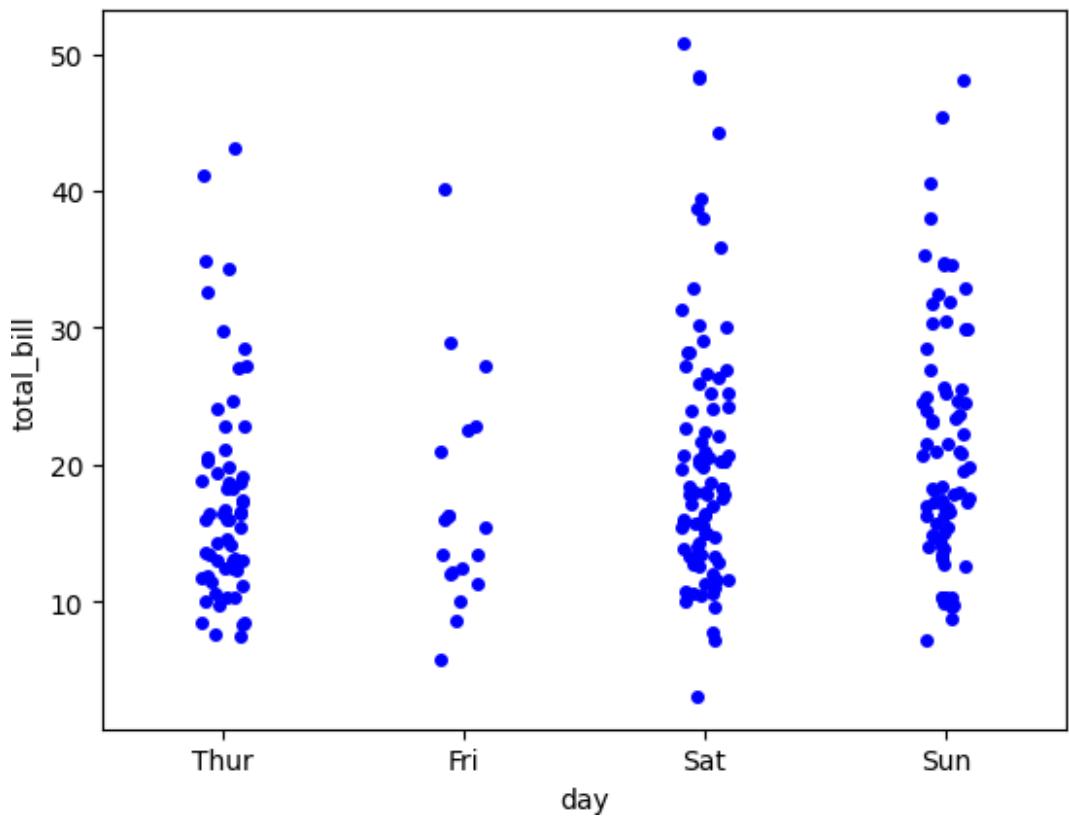


## stripplot and swarmplot

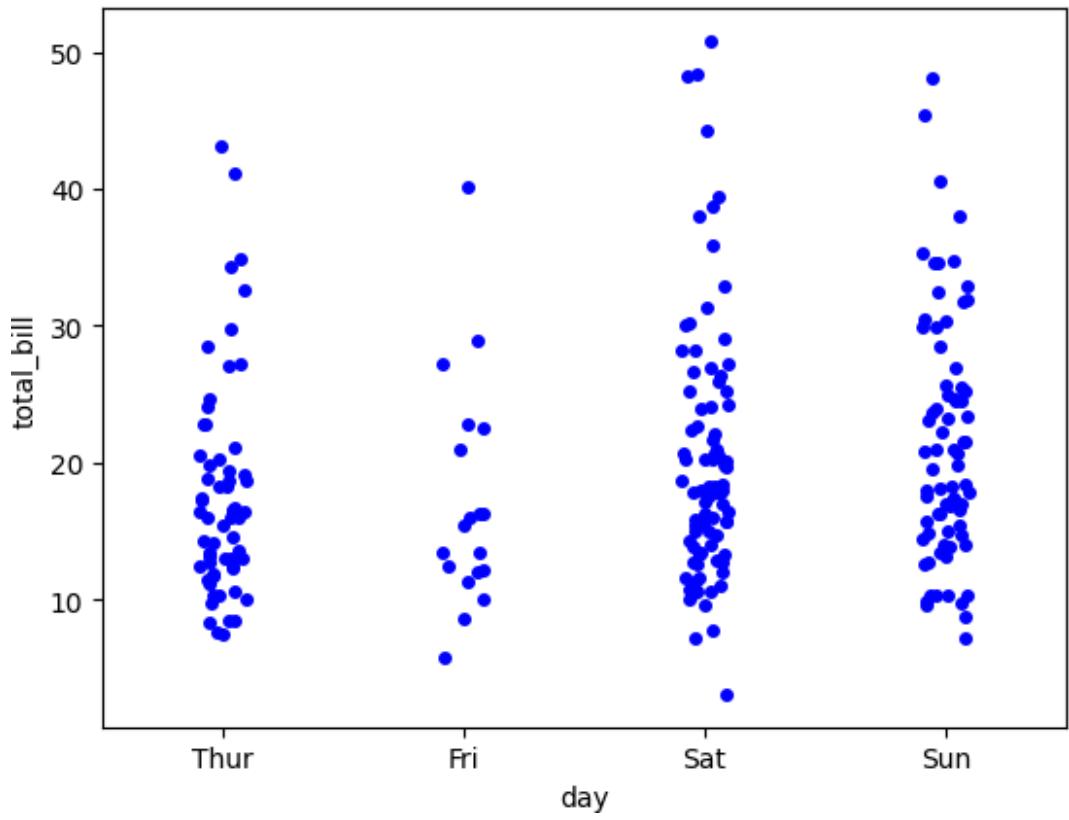
The stripplot will draw a scatterplot where one variable is categorical. A strip plot can be drawn on its own, but it is also a good complement to a box or violin plot in cases where you want to show all observations along with some representation of the underlying distribution.

The swarmplot is similar to stripplot(), but the points are adjusted (only along the categorical axis) so that they don't overlap. This gives a better representation of the distribution of values, although it does not scale as well to large numbers of observations (both in terms of the ability to show all the points and in terms of the computation needed to arrange them).

```
sns.stripplot(x="day", y="total_bill", data=tips)  
<Axes: xlabel='day', ylabel='total_bill'>
```

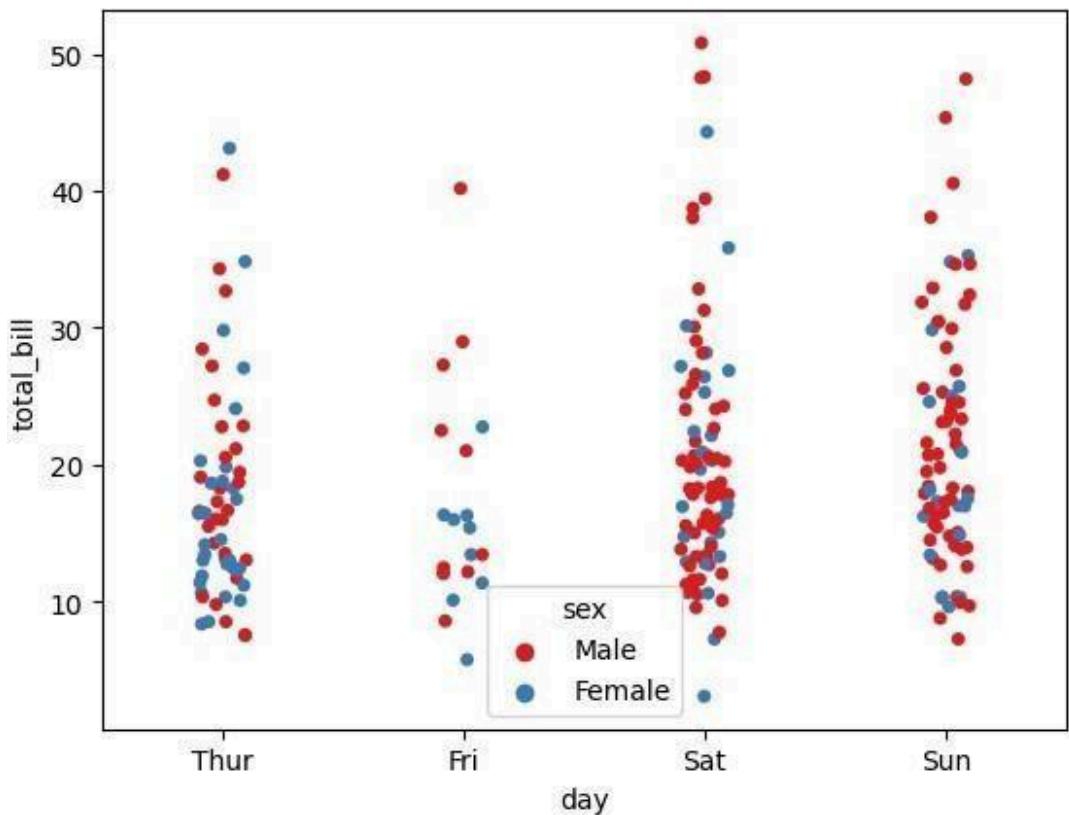


```
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True)  
<Axes: xlabel='day', ylabel='total_bill'>
```

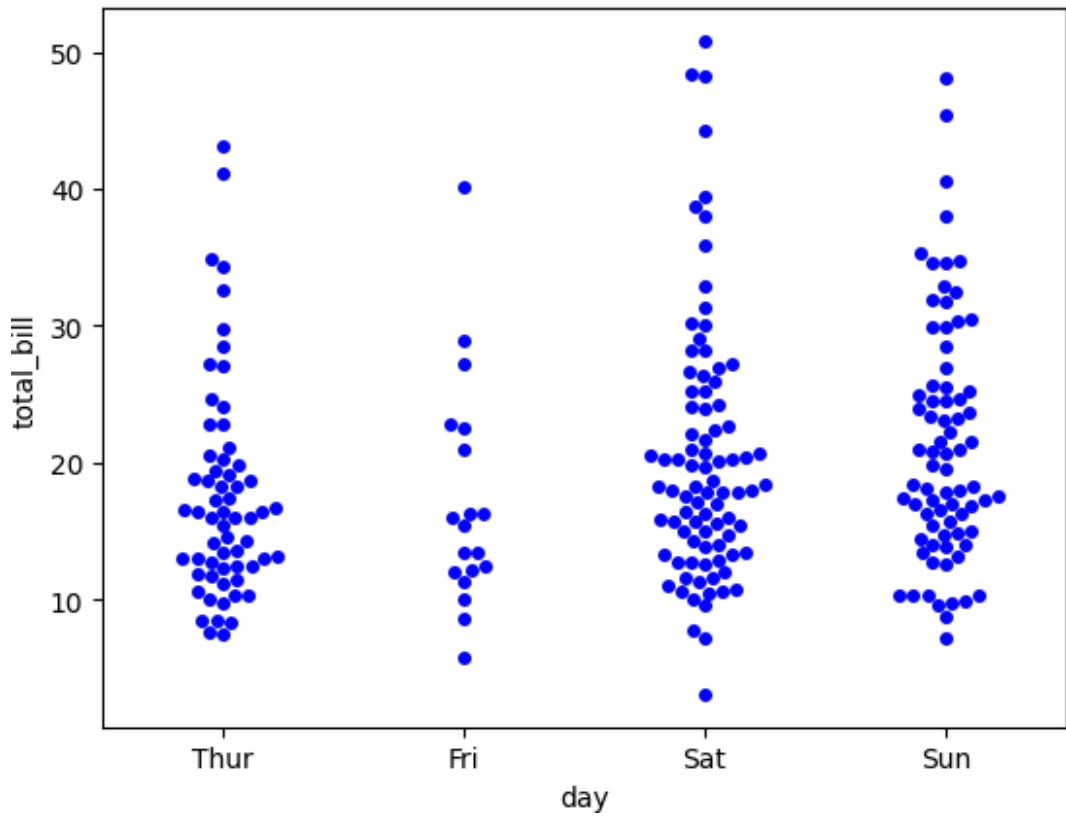


```
sns.stripplot(x="day", y="total_bill",
data=tips, jitter=True, hue='sex', palette='Set1')

<Axes: xlabel='day', ylabel='total_bill'>
```



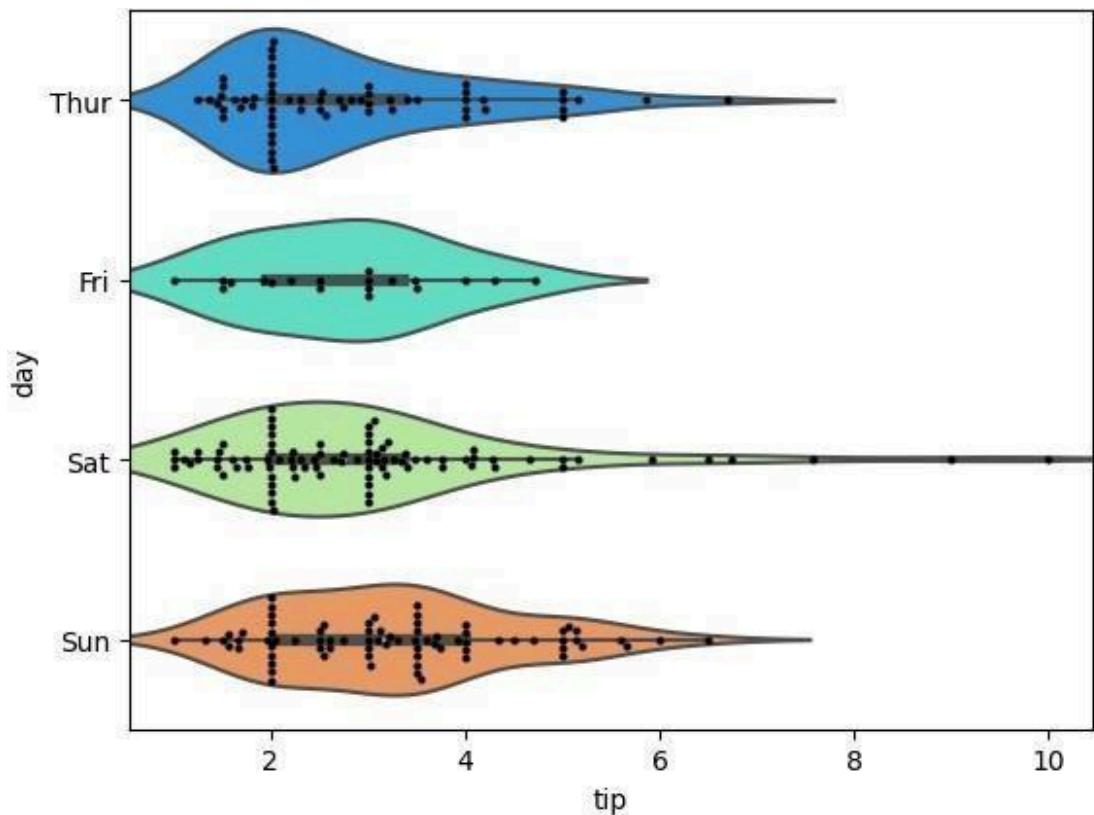
```
sns.swarmplot(x="day", y="total_bill", data=tips)  
<Axes: xlabel='day', ylabel='total_bill'>
```



### Combining Categorical Plots

```
sns.violinplot(x="tip", y="day", data=tips, palette='rainbow')
sns.swarmplot(x="tip", y="day", data=tips, color='black', size=3)

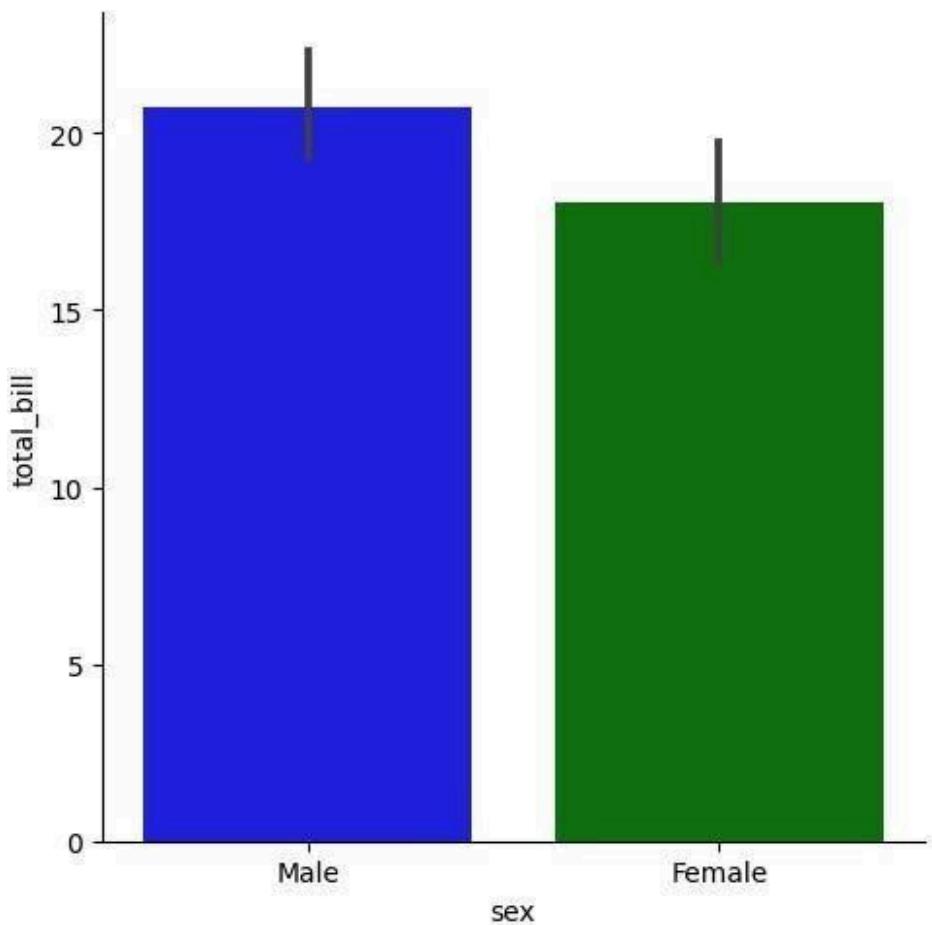
<Axes: xlabel='tip', ylabel='day'>
```



## factorplot

factorplot is the most general form of a categorical plot. It can take in a **kind** parameter to adjust the plot type:

```
sns.catplot(x='sex',y='total_bill',data=tips,kind='bar')  
<seaborn.axisgrid.FacetGrid at 0x7f5fc474bd60>
```



**Great Job!**

---

## Matrix Plots

Matrix plots allow you to plot data as color-encoded matrices and can also be used to indicate clusters within the data (later in the machine learning section we will learn how to formally cluster data).

Let's begin by exploring seaborn's heatmap and clutermapper:

```
import seaborn as sns
%matplotlib inline

flights =
sns.load_dataset('flights') tips =
sns.load_dataset('tips')
```

```

tips.head()

   total_bill  tip    sex smoker day     time size
0      16.99  1.01 Female     No Sun Dinner     2
1      10.34  1.66   Male     No Sun Dinner     3
2      21.01  3.50   Male     No Sun Dinner     3
3      23.68  3.31   Male     No Sun Dinner     2
4      24.59  3.61 Female     No Sun Dinner     4

flights.head()

   year month passengers
0 1949    Jan         112
1 1949    Feb         118
2 1949    Mar         132
3 1949    Apr         129
4 1949    May         121

```

## Heatmap

In order for a heatmap to work properly, your data should already be in a matrix form, the `sns.heatmap` function basically just colors it in for you. For example:

```

tips.head()

   total_bill  tip    sex smoker day     time size
0      16.99  1.01 Female     No Sun Dinner     2
1      10.34  1.66   Male     No Sun Dinner     3
2      21.01  3.50   Male     No Sun Dinner     3
3      23.68  3.31   Male     No Sun Dinner     2
4      24.59  3.61 Female     No Sun Dinner     4

# Matrix form for correlation data
tips.corr()

<ipython-input-14-494e8f871a3b>:2: FutureWarning: The default
value of numeric_only in DataFrame.corr is deprecated. In a future
version, it will default to False. Select only valid columns or
specify the value of numeric_only to silence this warning.
tips.corr()

   total_bill      tip      size
total_bill    1.000000  0.675734  0.598315
tip          0.675734  1.000000  0.489299
size         0.598315  0.489299  1.000000

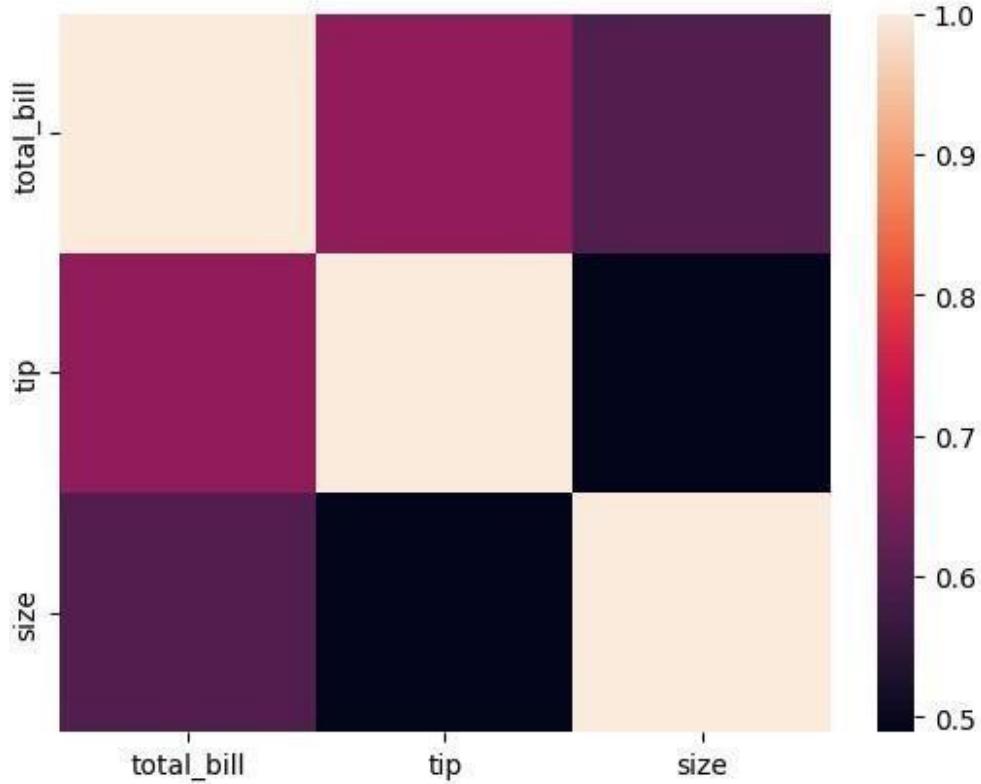
sns.heatmap(tips.corr())

```

<ipython-input-15-7ebab8d3a18f>:1: FutureWarning: The default value  
of numeric\_only in DataFrame.corr is deprecated. In a future  
version, it will default to False. Select only valid columns or  
specify the value

```
of numeric_only to silence this warning.  
sns.heatmap(tips.corr())
```

```
<Axes: >
```

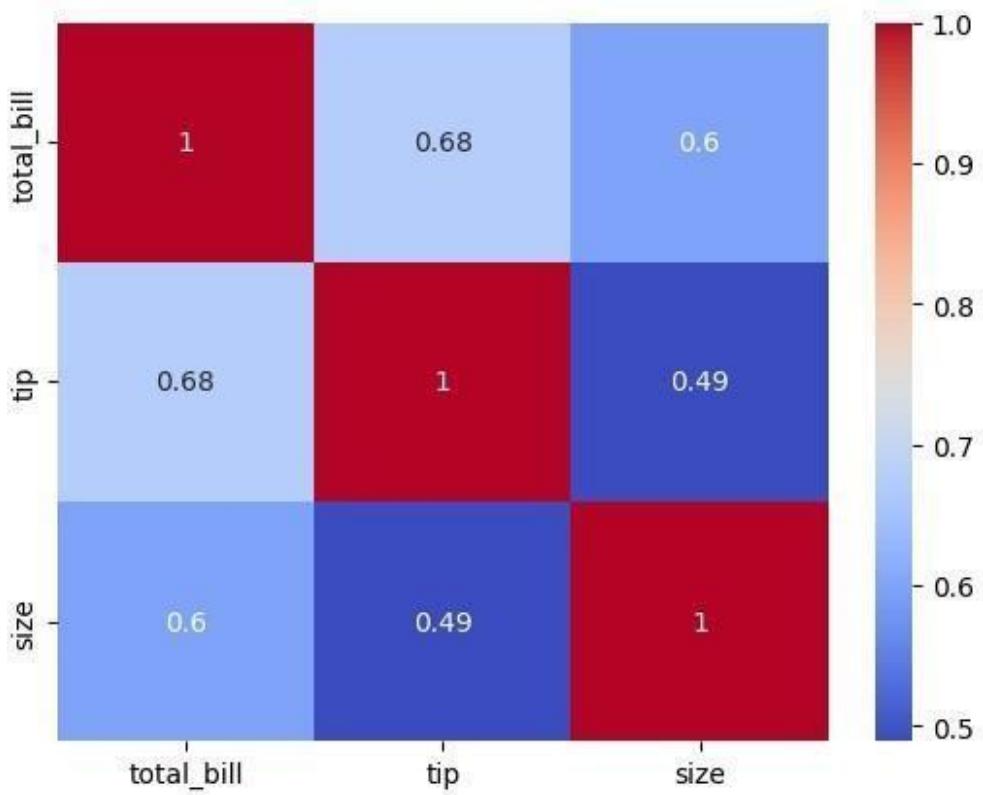


```
sns.heatmap(tips.corr(), cmap='coolwarm', annot=True)
```

```
<ipython-input-16-fdc7b9ef83f5>:1: FutureWarning: The default  
value of numeric_only in DataFrame.corr is deprecated. In a future  
version, it will default to False. Select only valid columns or  
specify the value of numeric_only to silence this warning.
```

```
sns.heatmap(tips.corr(), cmap='coolwarm', annot=True)
```

```
<Axes: >
```



Or for the flights data:

```
flights.pivot_table(values='passengers', index='month', columns='year')
)

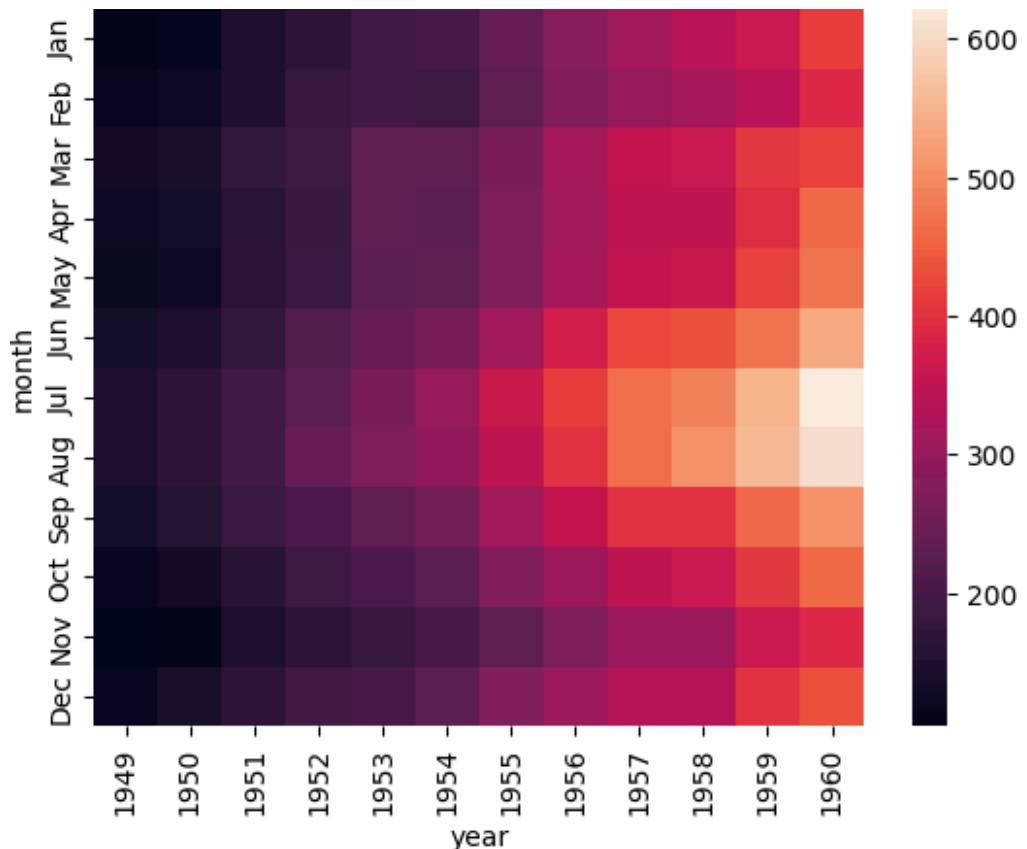
year    1949   1950   1951   1952   1953   1954   1955   1956   1957   1958
1959   1960
month

Jan      112   115   145   171   196   204   242   284   315   340
360     417
Feb      118   126   150   180   196   188   233   277   301   318
342     391
Mar      132   141   178   193   236   235   267   317   356   362
406     419
Apr      129   135   163   181   235   227   269   313   348   348
396     461
May      121   125   172   183   229   234   270   318   355   363
420     472
Jun      135   149   178   218   243   264   315   374   422   435
472     535
Jul      148   170   199   230   264   302   364   413   465   491
548     622
Aug      148   170   199   242   272   293   347   405   467   505
559     606
Sep      136   158   184   209   237   259   312   355   404   404
463     508
```

Oct	119	133	162	191	211	229	274	306	347	359
407	461									
Nov	104	114	146	172	180	203	237	271	305	310
362	390									
Dec	118	140	166	194	201	229	278	306	336	337
405	432									

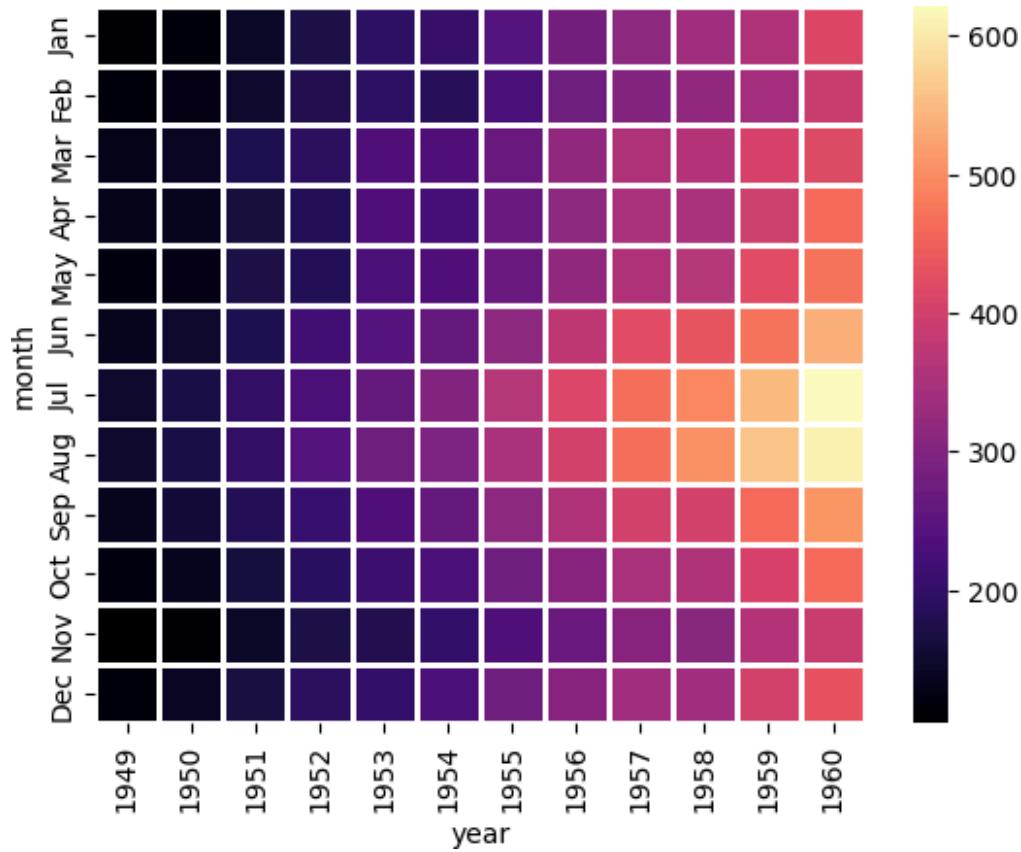
```
pvflights =
flights.pivot_table(values='passengers', index='month', columns='year')
sns.heatmap(pvflights)
```

<Axes: xlabel='year', ylabel='month'>



```
sns.heatmap(pvflights, cmap='magma', linewidths=1)
```

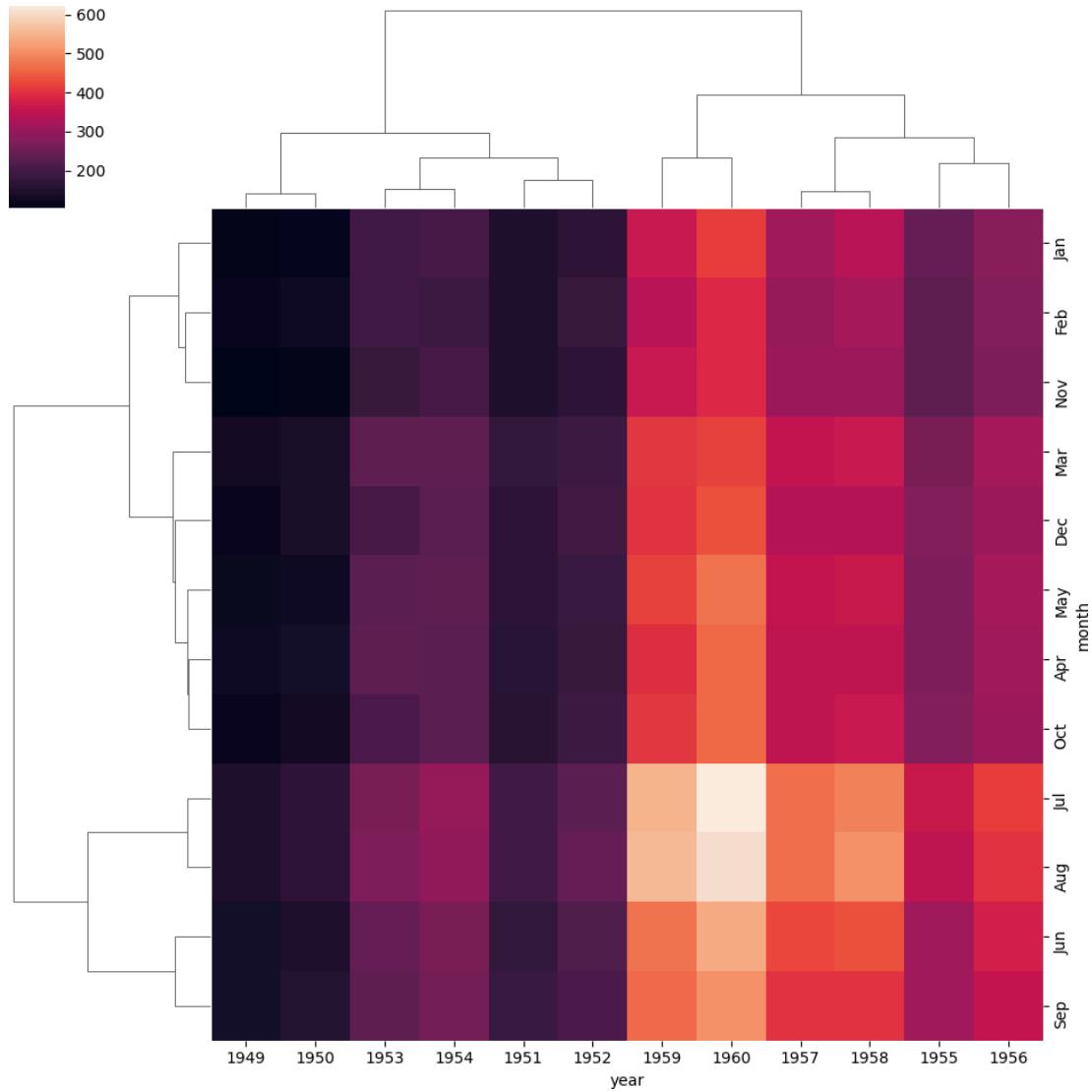
<Axes: xlabel='year', ylabel='month'>



## clustermap

The clustermap uses hierachal clustering to produce a clustered version of the heatmap. For example:

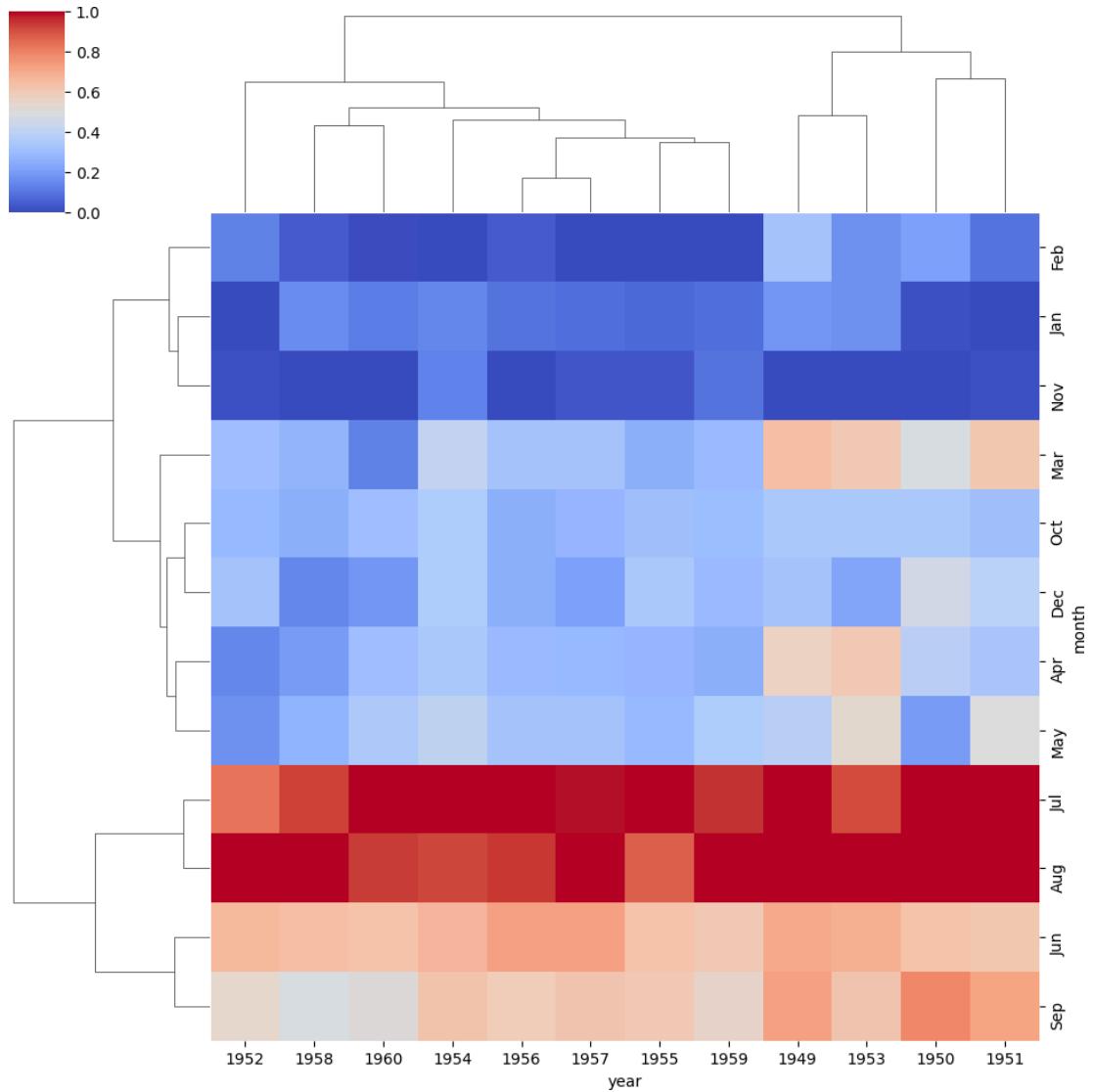
```
sns.clustermap(pvflights)  
<seaborn.matrix.ClusterGrid at 0x7f1d9b84e140>
```



Notice now how the years and months are no longer in order, instead they are grouped by similarity in value (passenger count). That means we can begin to infer things from this plot, such as August and July being similar (makes sense, since they are both summer travel months)

```
# More options to get the information a little clearer
# like normalization
sns.clustermap(pvflights, cmap='coolwarm', standard_scale=1)

<seaborn.matrix.ClusterGrid at 0x7f1d9b84d4b0>
```



**Great Job!**

---

## Grids

Grids are general types of plots that allow you to map plot types to rows and columns of a grid, this helps you create similar plots separated by features.

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
iris = sns.load_dataset('iris')

iris.head()

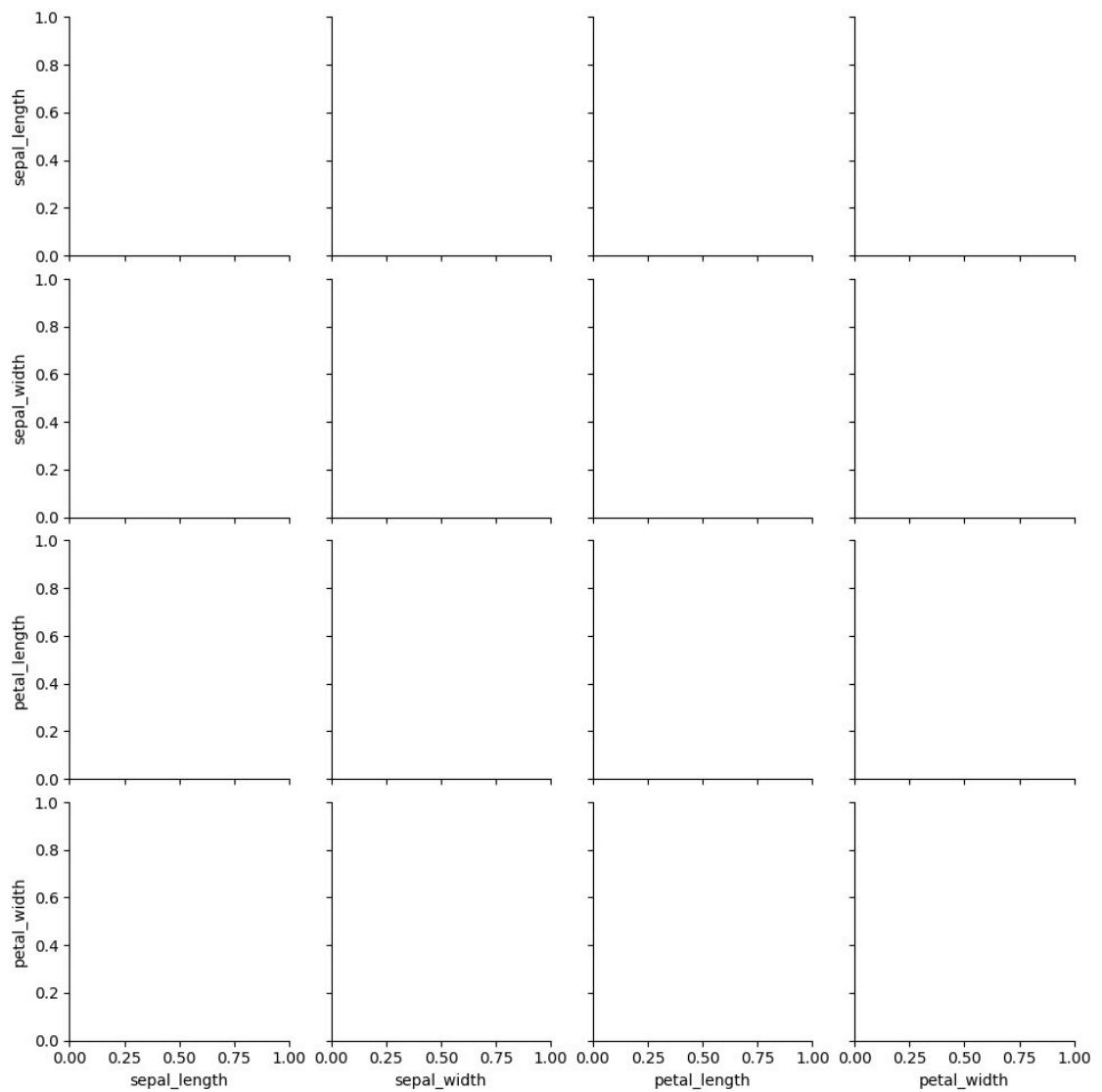
  sepal_length  sepal_width  petal_length  petal_width  species
0          5.1         3.5          1.4         0.2   setosa
1          4.9         3.0          1.4         0.2   setosa
2          4.7         3.2          1.3         0.2   setosa
3          4.6         3.1          1.5         0.2   setosa
4          5.0         3.6          1.4         0.2   setosa
```

## PairGrid

Pairgrid is a subplot grid for plotting pairwise relationships in a dataset.

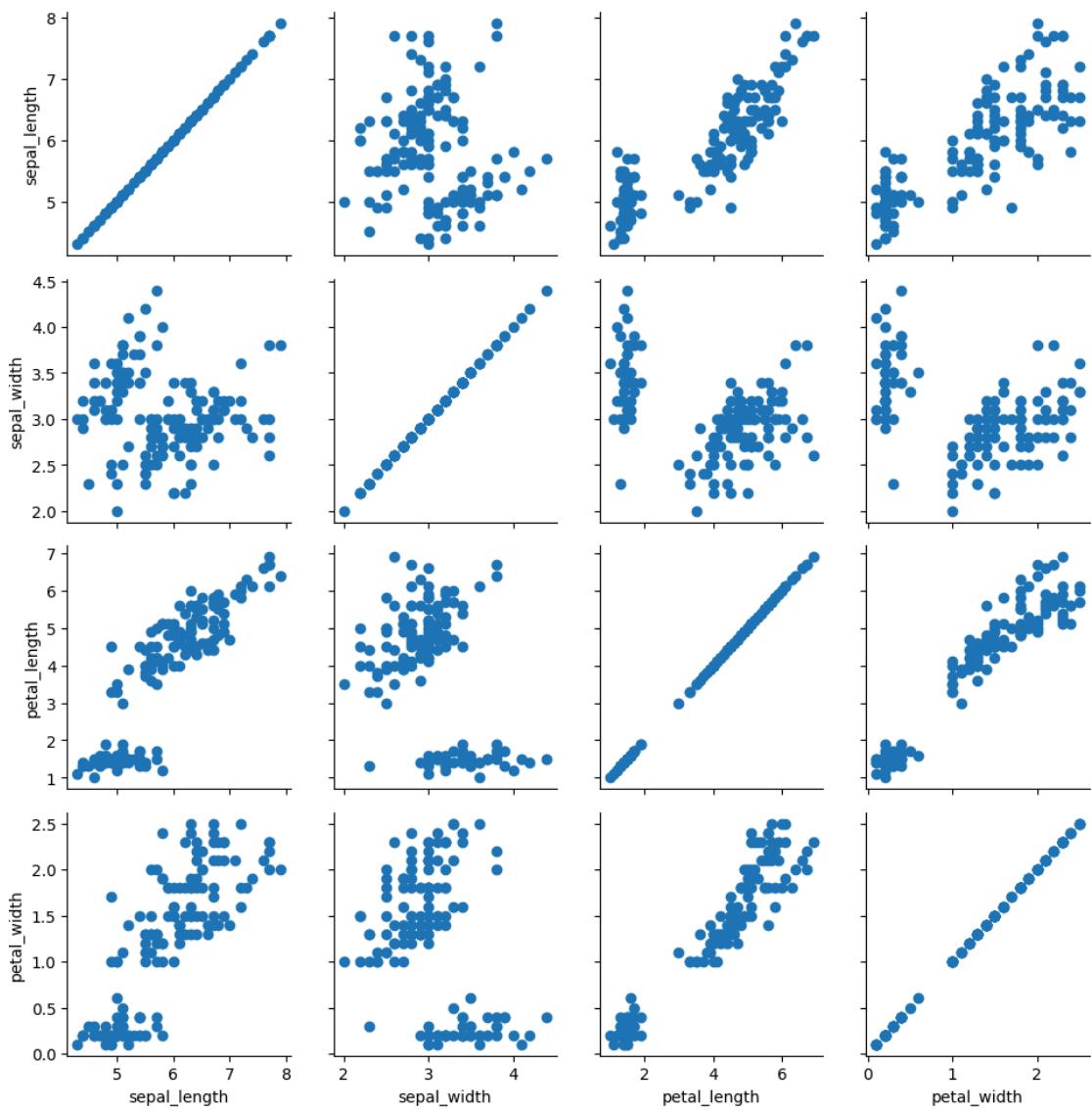
```
# Just the Grid
sns.PairGrid(iris)

<seaborn.axisgrid.PairGrid at 0x7f686ed59c00>
```



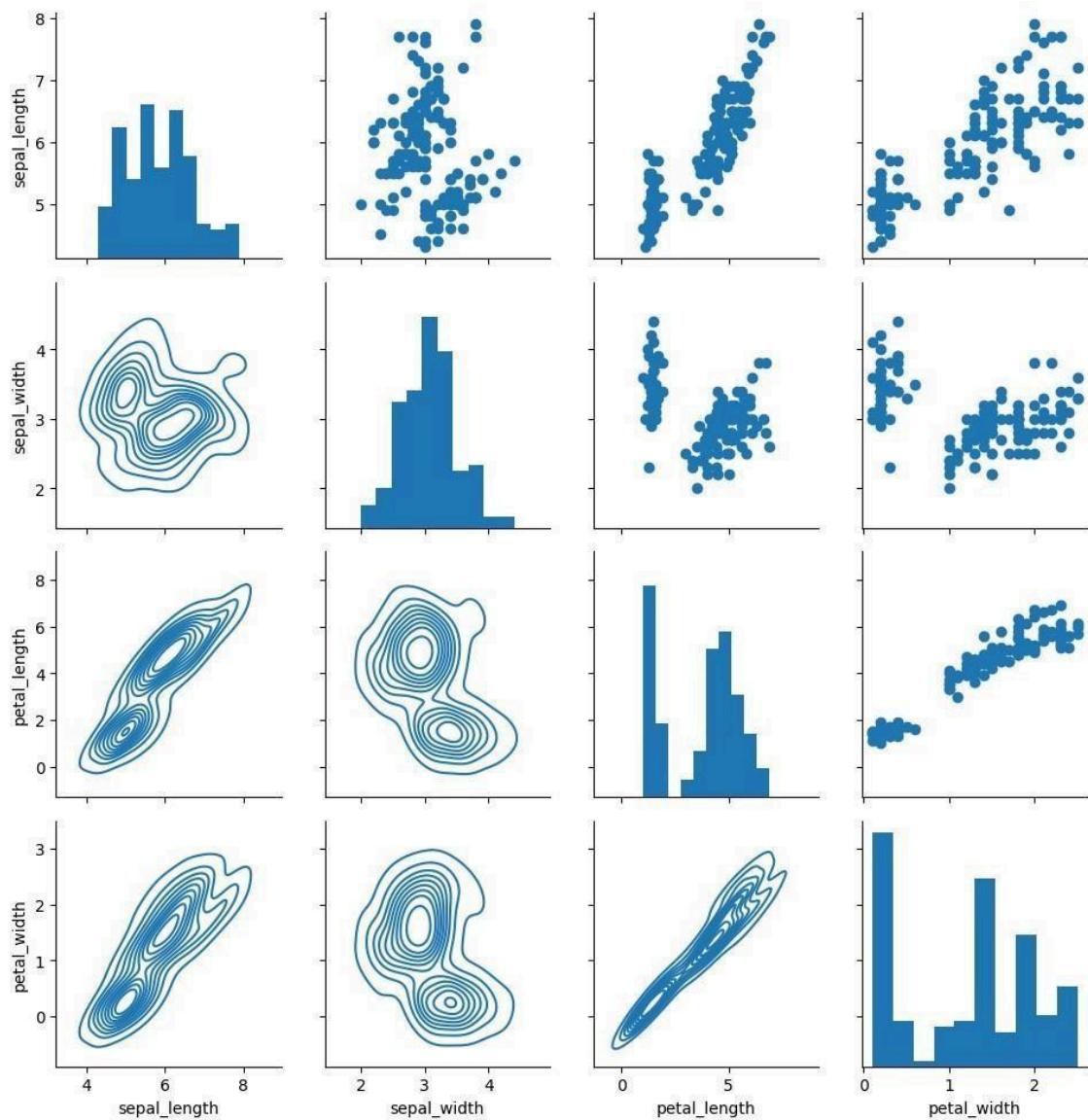
```
# Then you map to the
grid g =
sns.PairGrid(iris)
g.map(plt.scatter)

<seaborn.axisgrid.PairGrid at 0x7f1962c11090>
```



```
# Map to upper, lower, and
# diagonal g =
sns.PairGrid(iris)
g.map_diag(plt.hist)
g.map_upper(plt.scatter)
g.map_lower(sns.kdeplot)

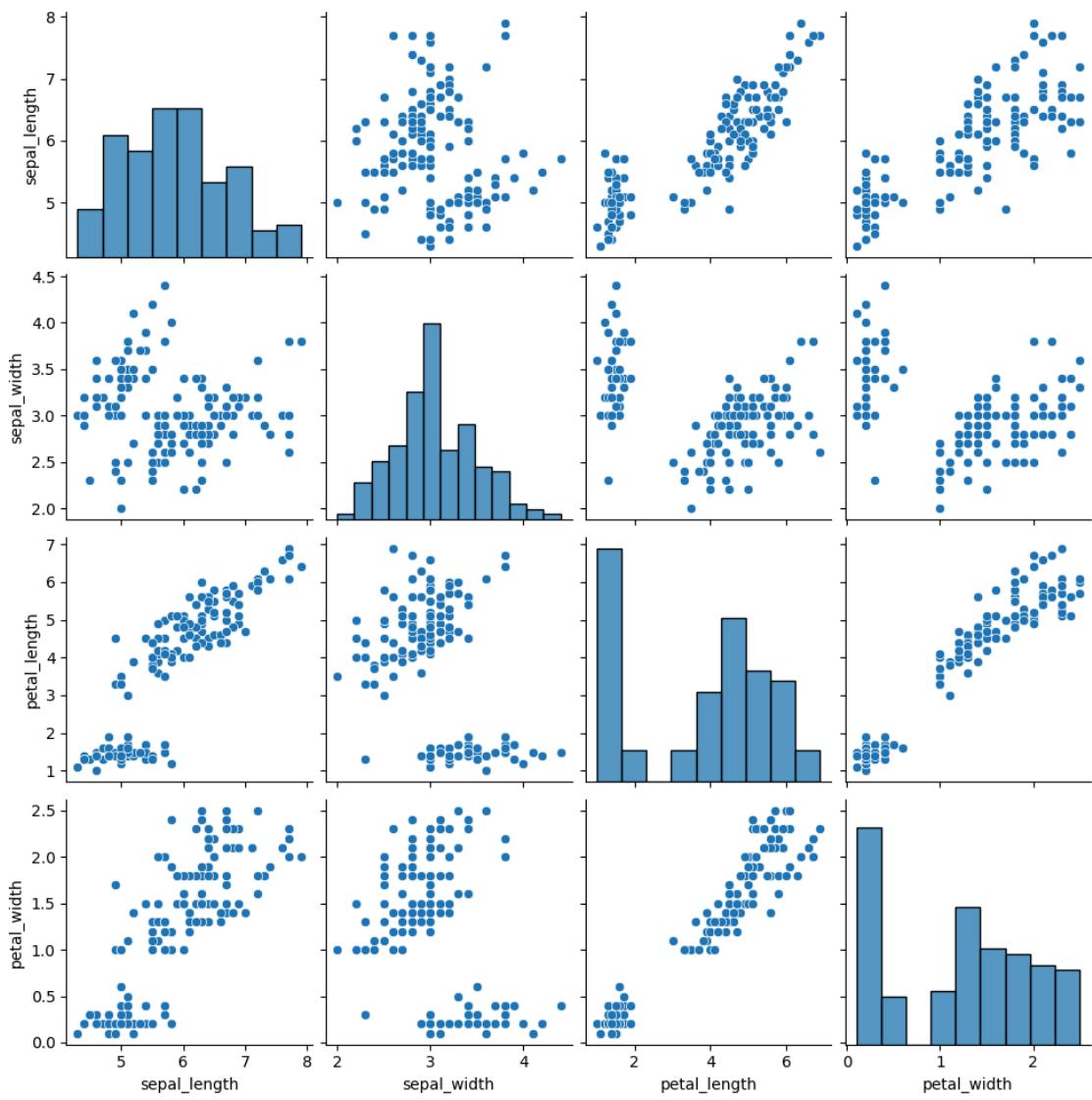
<seaborn.axisgrid.PairGrid at 0x7f19624fbb80>
```



## pairplot

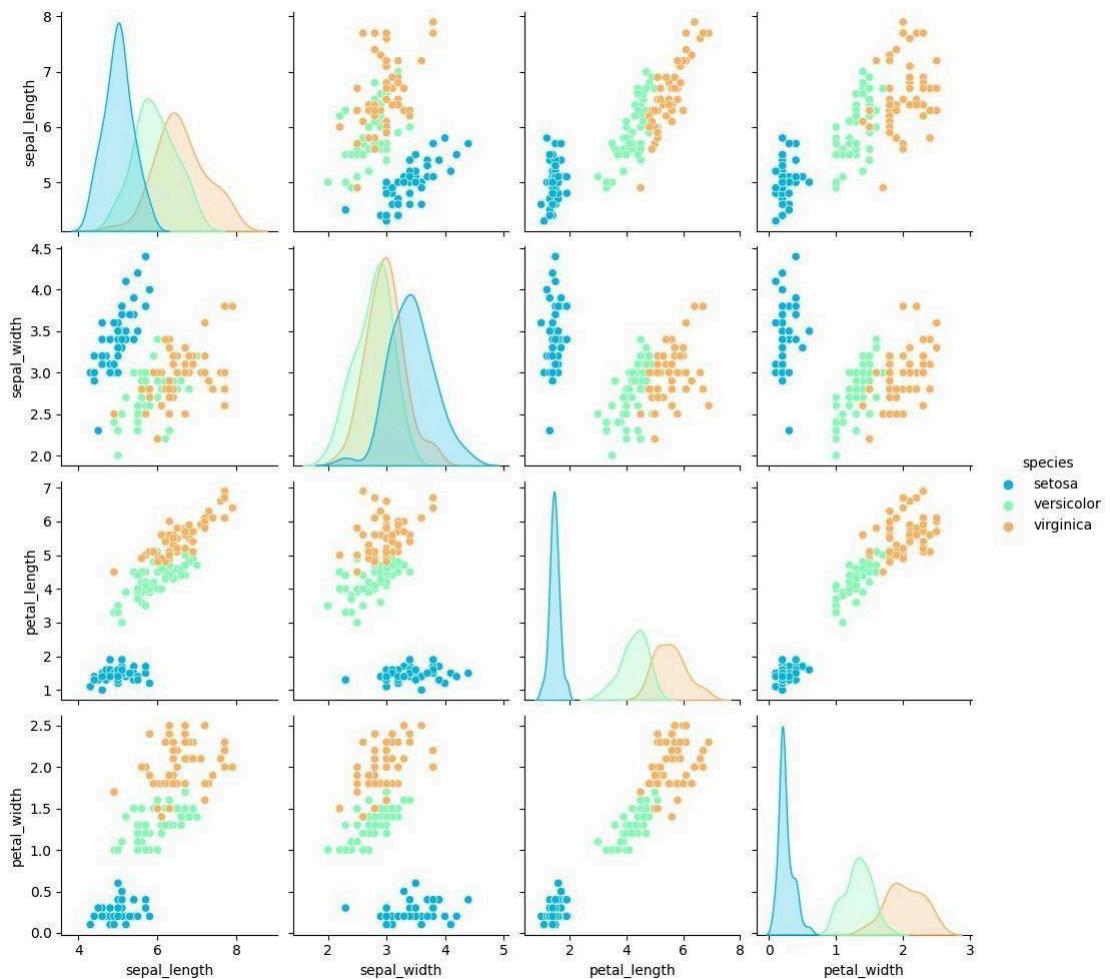
pairplot is a simpler version of PairGrid (you'll use quite often)

```
sns.pairplot(iris)  
<seaborn.axisgrid.PairGrid at 0x7f19624fbbe0>
```



```
sns.pairplot(iris, hue='species', palette='rainbow')
```

```
<seaborn.axisgrid.PairGrid at 0x7f1997df6dd0>
```



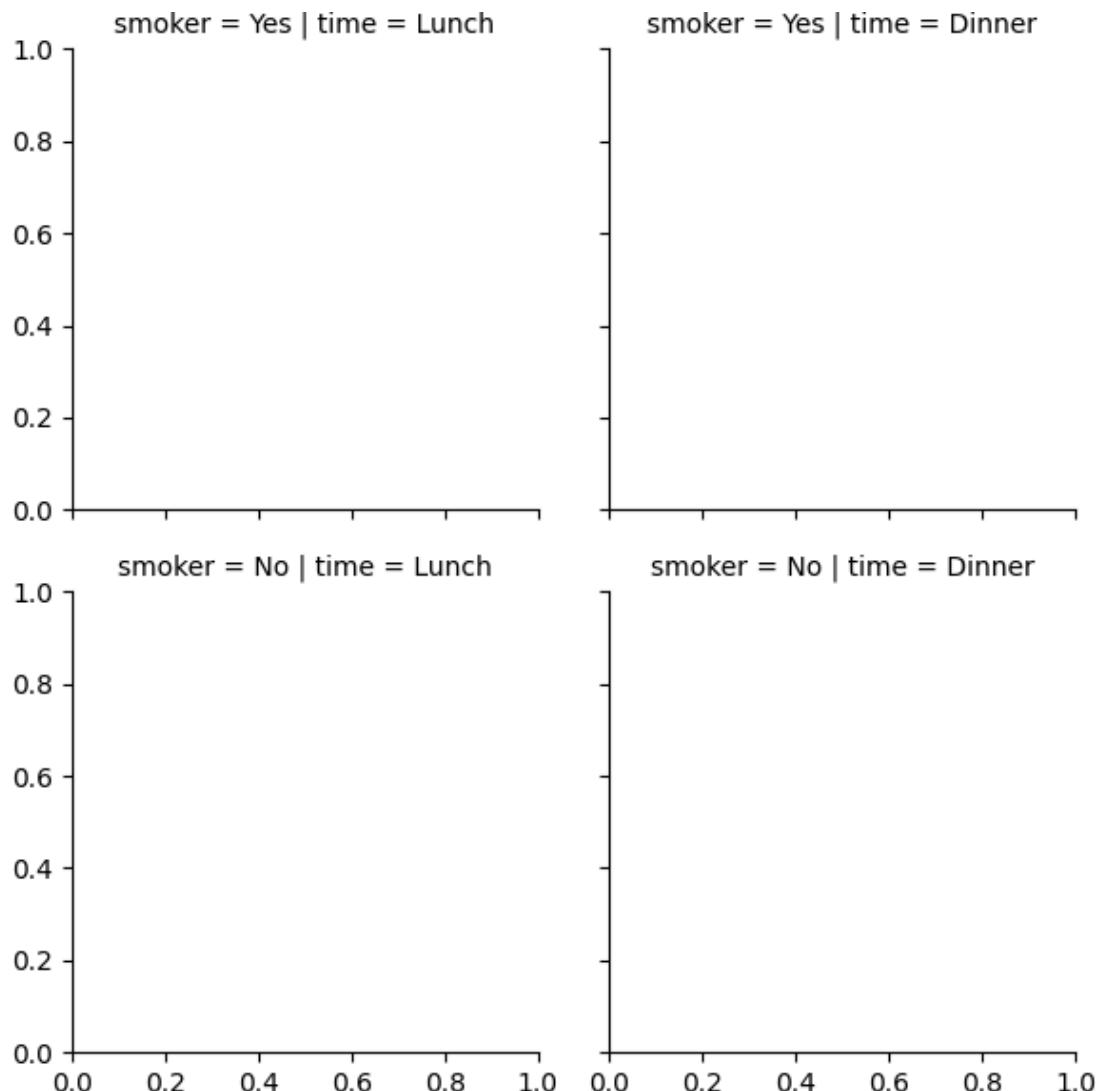
## Facet Grid

FacetGrid is the general way to create grids of plots based off of a feature:

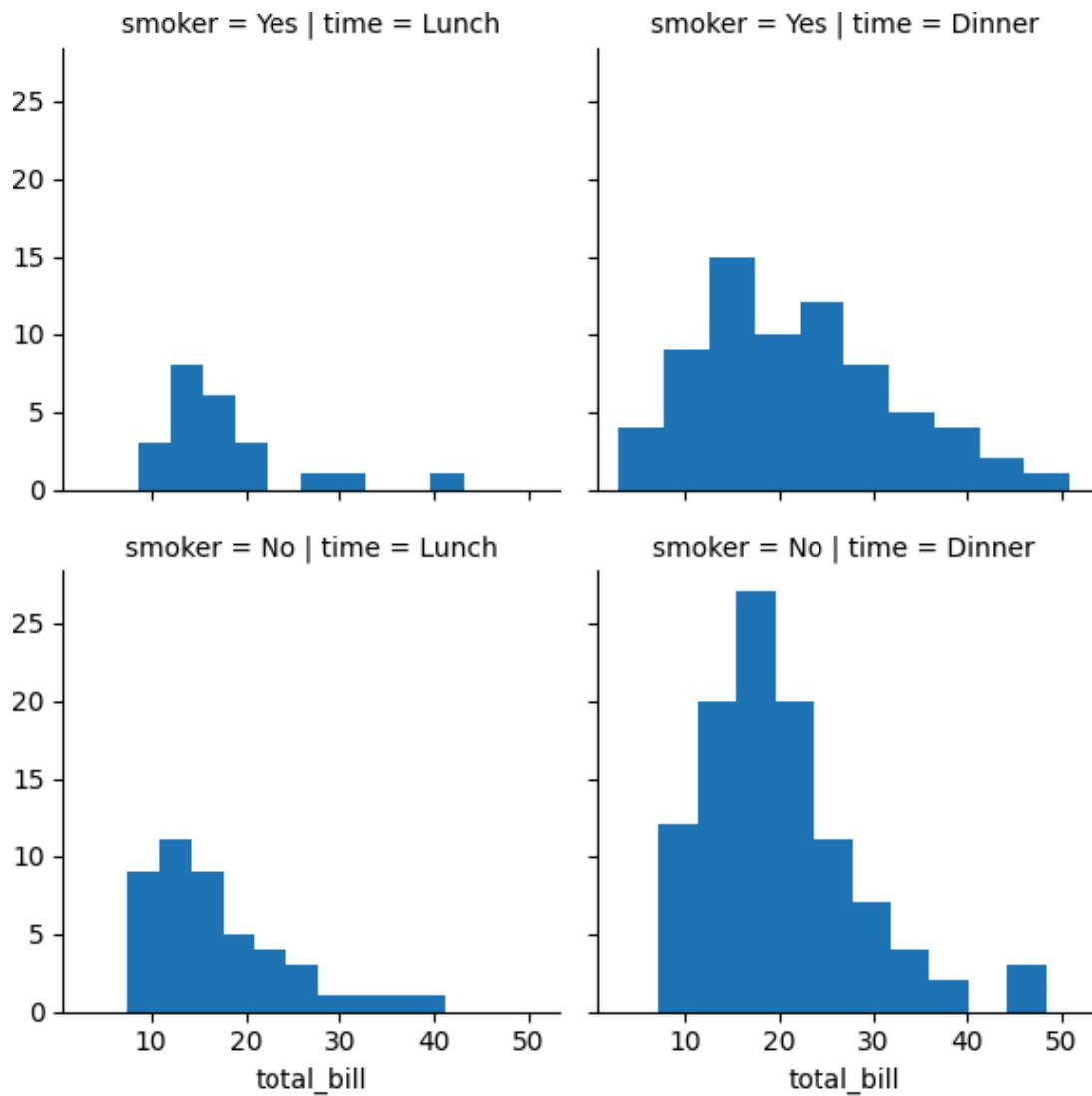
```
tips = sns.load_dataset('tips')
tips.head()

  total_bill  tip      sex smoker day    time size
0     16.99  1.01  Female     No Sun Dinner    2
1     10.34  1.66    Male     No Sun Dinner    3
2     21.01  3.50    Male     No Sun Dinner    3
3     23.68  3.31    Male     No Sun Dinner    2
4     24.59  3.61  Female     No Sun Dinner    4

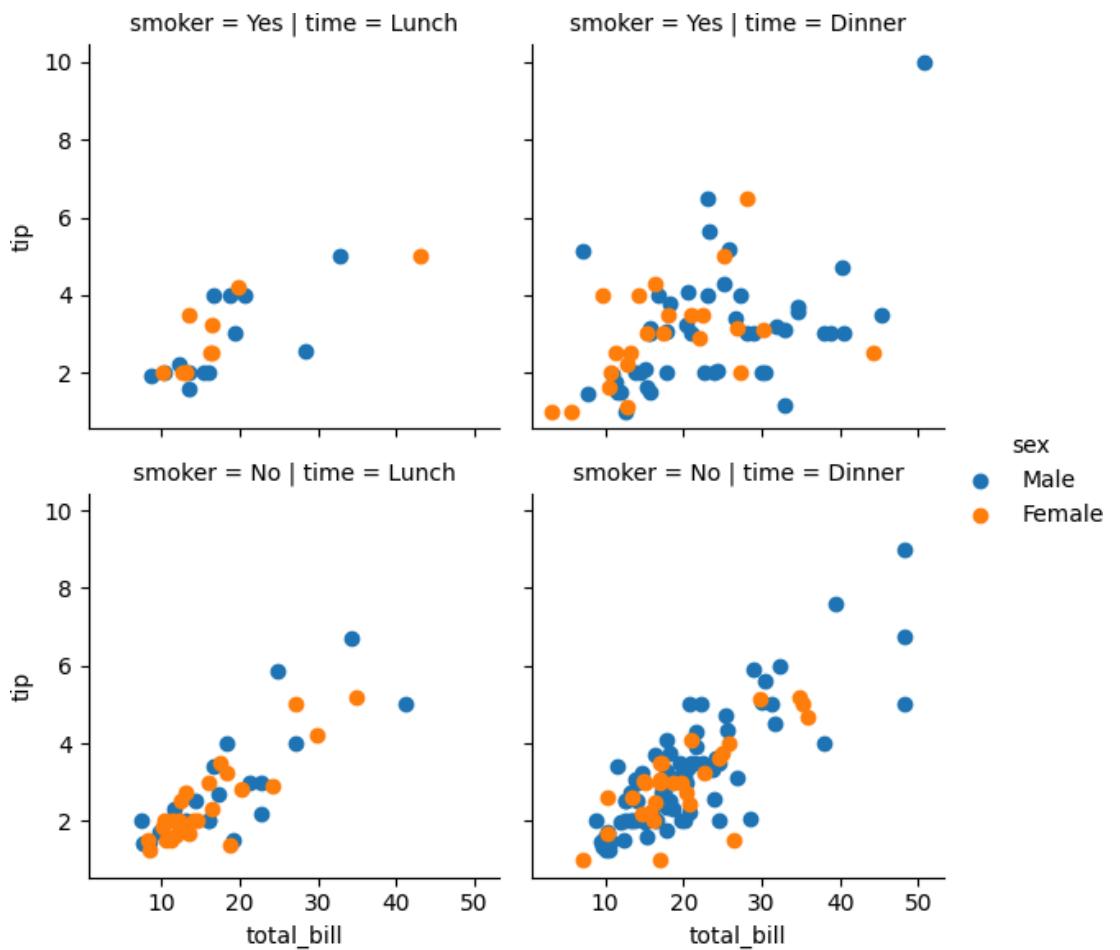
# Just the Grid
g = sns.FacetGrid(tips, col="time", row="smoker")
```



```
g = sns.FacetGrid(tips, col="time",
row="smoker") g = g.map(plt.hist,
"total_bill")
```



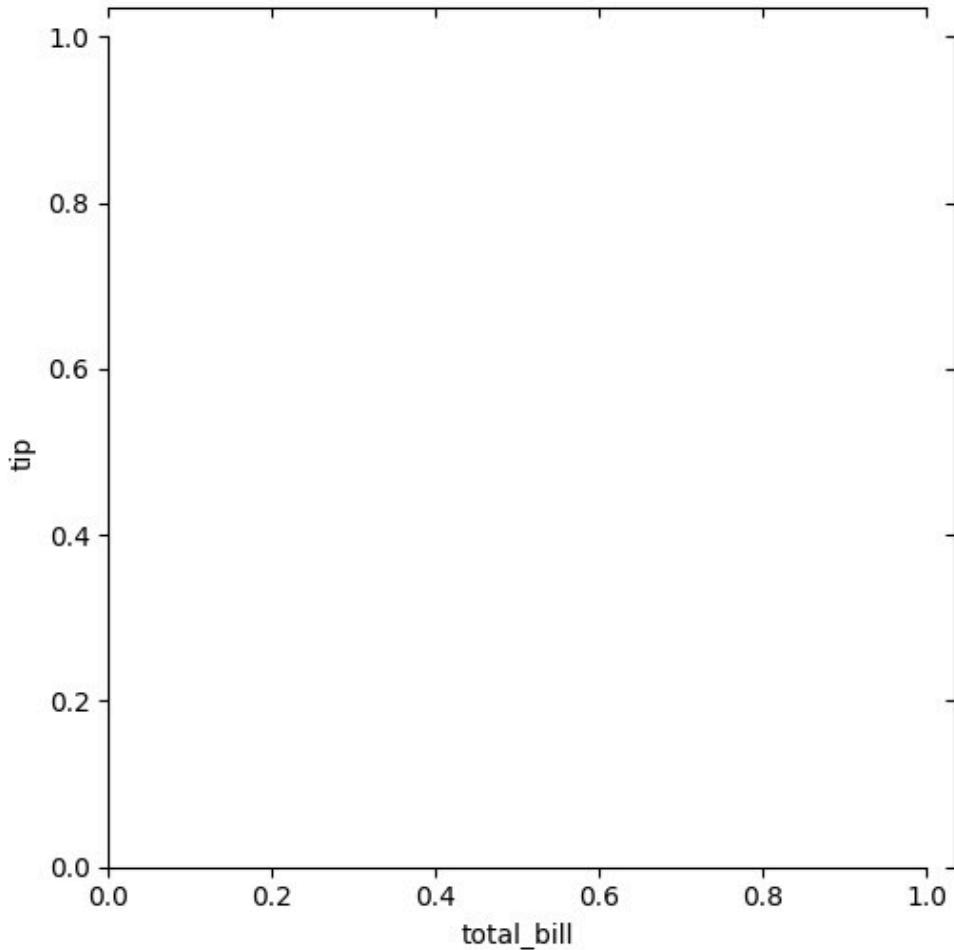
```
g = sns.FacetGrid(tips, col="time", row="smoker", hue='sex')
# Notice how the arguments come after plt.scatter call
g = g.map(plt.scatter, "total_bill", "tip").add_legend()
```



## JointGrid

JointGrid is the general version for jointplot() type grids, for a quick example:

```
g = sns.JointGrid(x="total_bill", y="tip", data=tips)
```



```
g = sns.JointGrid(x="total_bill", y="tip",
data=tips) g = g.plot(sns.regplot, sns.distplot)

/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:1880:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

Please adapt your code to use either `displot` (a
figure-level function with
similar flexibility) or `histplot` (an axes-level function for
histograms).
```

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

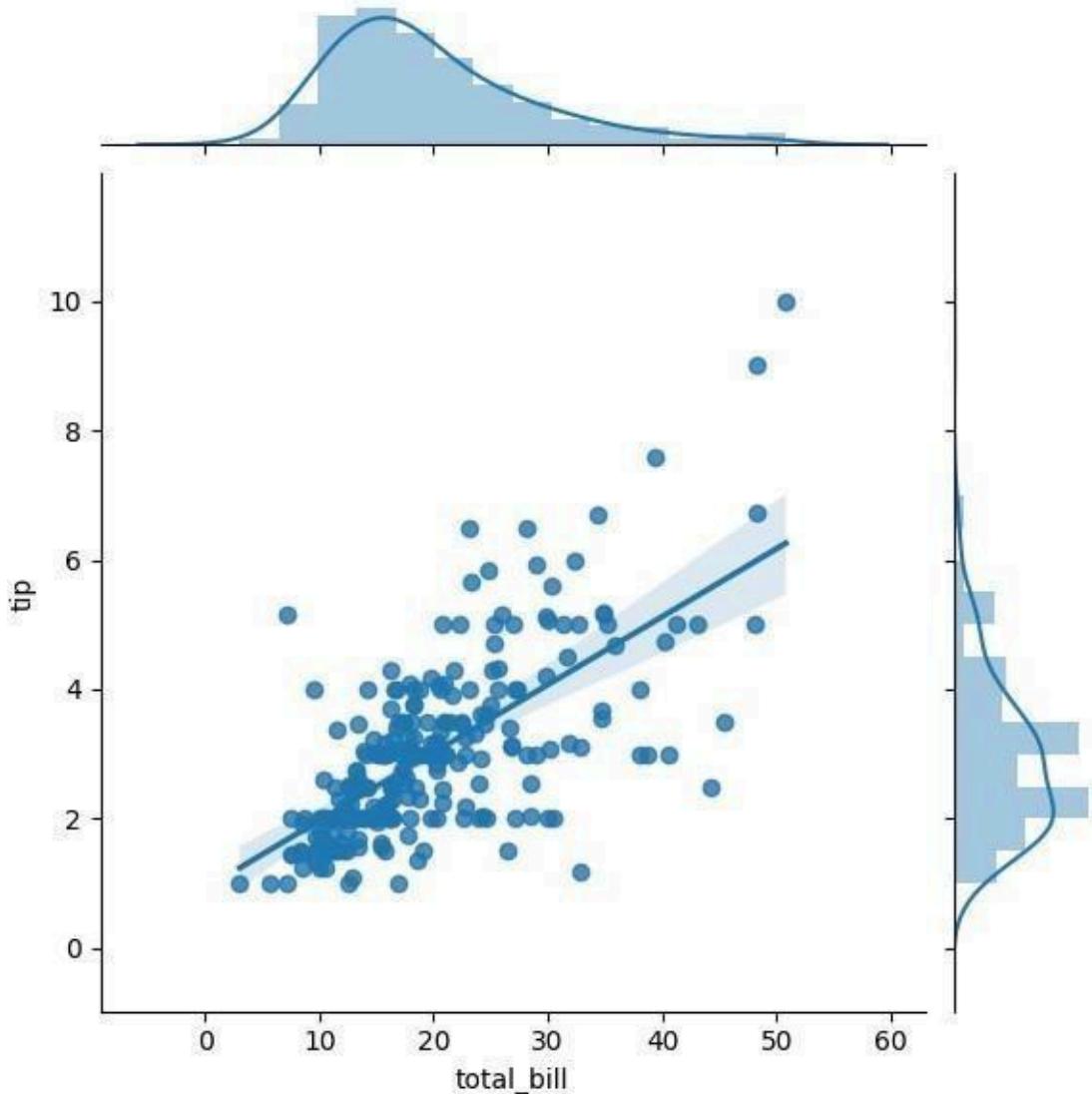
```
func(self.x, **orient_kw_x, **kwargs)
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:1886:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.
```

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
func(self.y, **orient_kw_y, **kwargs)
```



Reference the documentation as necessary for grid types, but most of the time you'll just use the easier plots discussed earlier.

**Great Job!**

---

## Regression Plots

Seaborn has many built-in capabilities for regression plots, however we won't really discuss regression until the machine learning section of the course, so we will only cover the `lmplot()` function for now.

**lmplot** allows you to display linear models, but it also conveniently allows you to split up those plots based off of features, as well as coloring the hue based off of features.

Let's explore how this works:

```
import seaborn as sns
%matplotlib inline

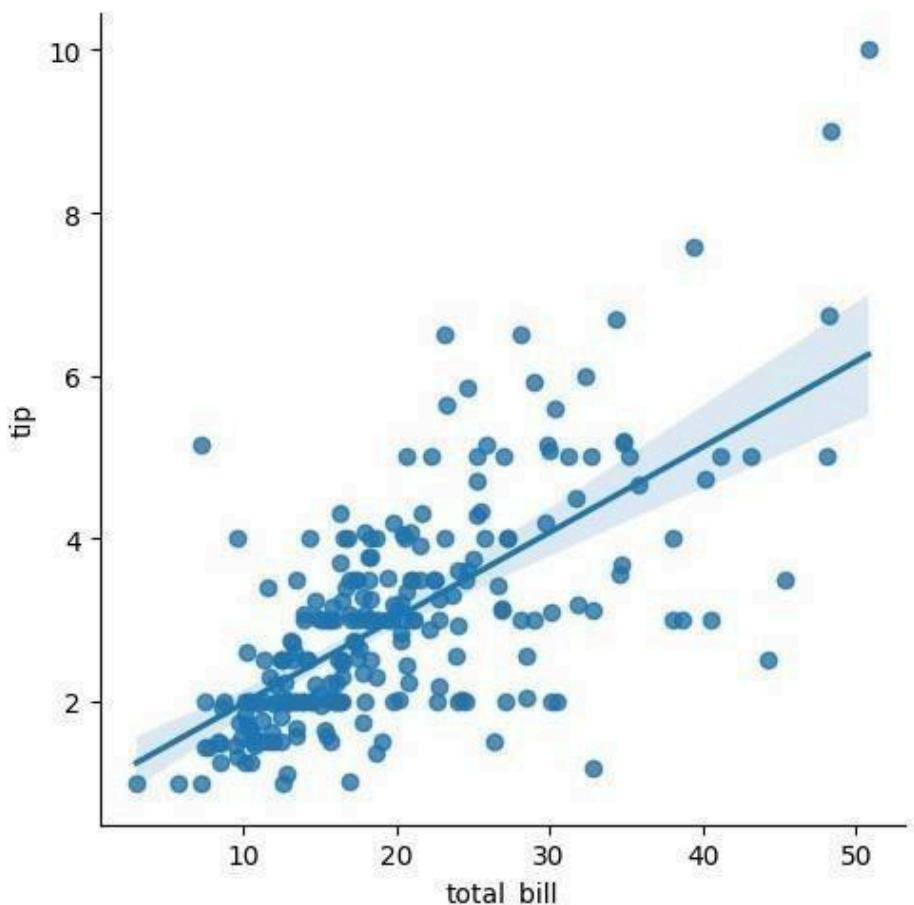
tips = sns.load_dataset('tips')
tips.head()

  total_bill  tip    sex smoker day   time size
0      16.99  1.01  Female     No  Sun Dinner     2
1      10.34  1.66    Male     No  Sun Dinner     3
2      21.01  3.50    Male     No  Sun Dinner     3
3      23.68  3.31    Male     No  Sun Dinner     2
4      24.59  3.61  Female     No  Sun Dinner     4
```

### lmplot()

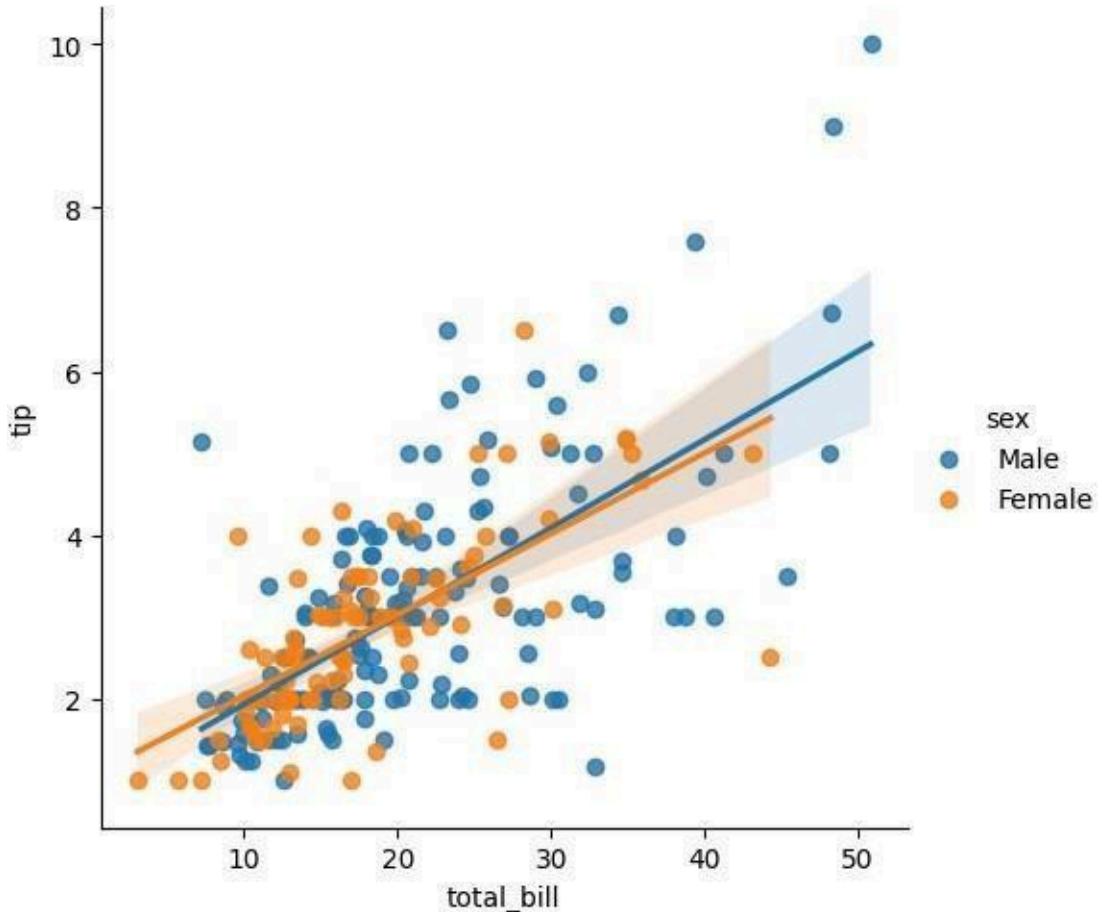
```
sns.lmplot(x='total_bill', y='tip', data=tips)

<seaborn.axisgrid.FacetGrid at 0x7fbba404e8f0>
```



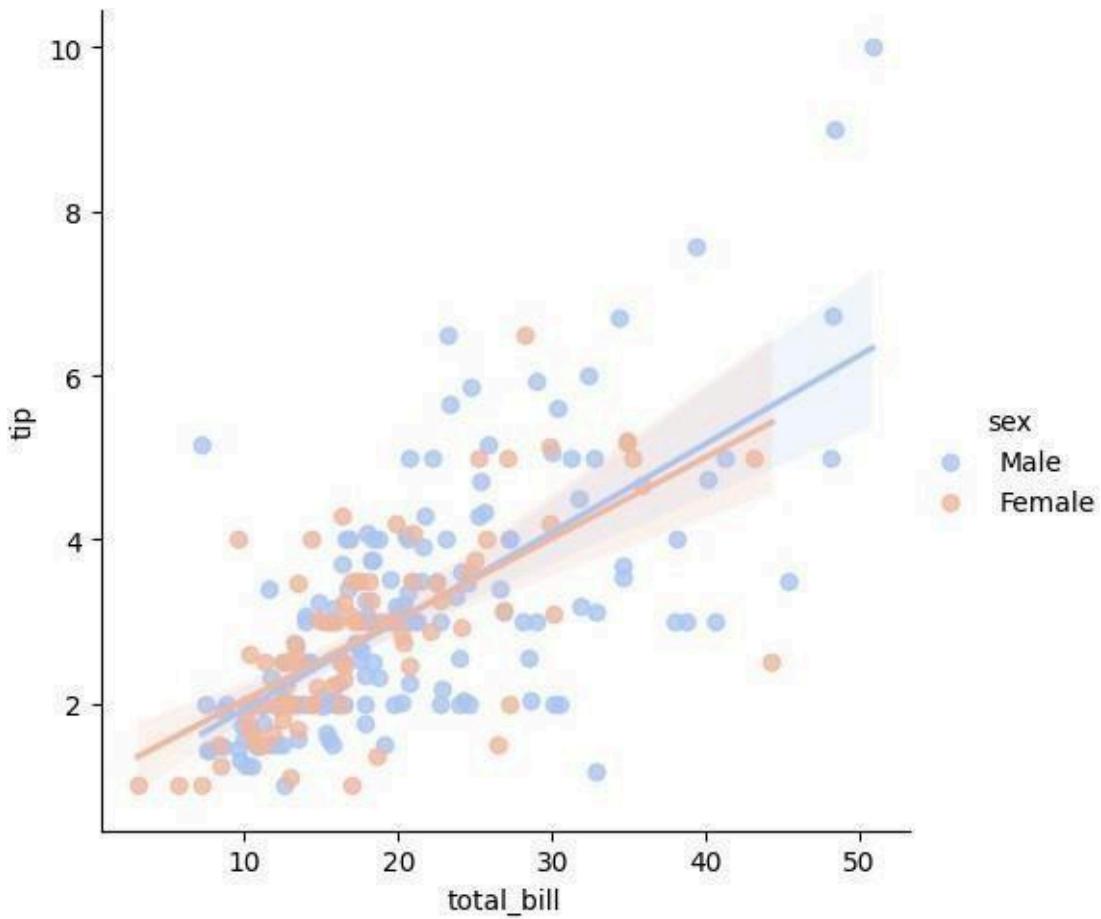
```
sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fbb6d943250>
```



```
sns.lmplot(x='total_bill',y='tip',data=tips,hue='sex',palette='coolwarm')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fbb6b730ca0>
```

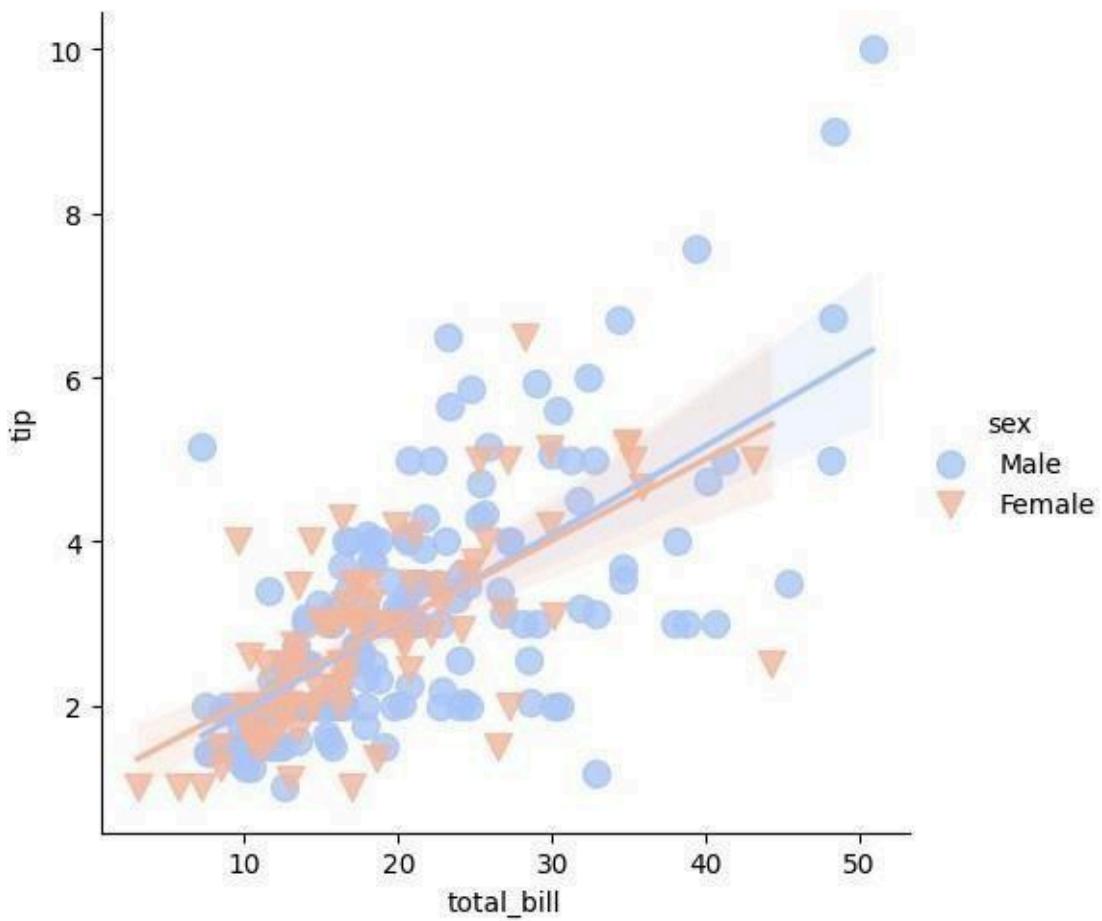


## Working with Markers

lmplot kwargs get passed through to **regplot** which is a more general form of lmplot(). regplot has a scatter\_kws parameter that gets passed to plt.scatter. So you want to set the s parameter in that dictionary, which corresponds (a bit confusingly) to the squared markersize. In other words you end up passing a dictionary with the base matplotlib arguments, in this case, s for size of a scatter plot. In general, you probably won't remember this off the top of your head, but instead reference the documentation.

```
# http://matplotlib.org/api/markers_api.html
sns.lmplot(x='total_bill', y='tip', data=tips, hue='sex', palette='coolwarm',
            markers=['o', 'v'], scatter_kws={'s': 100})
```

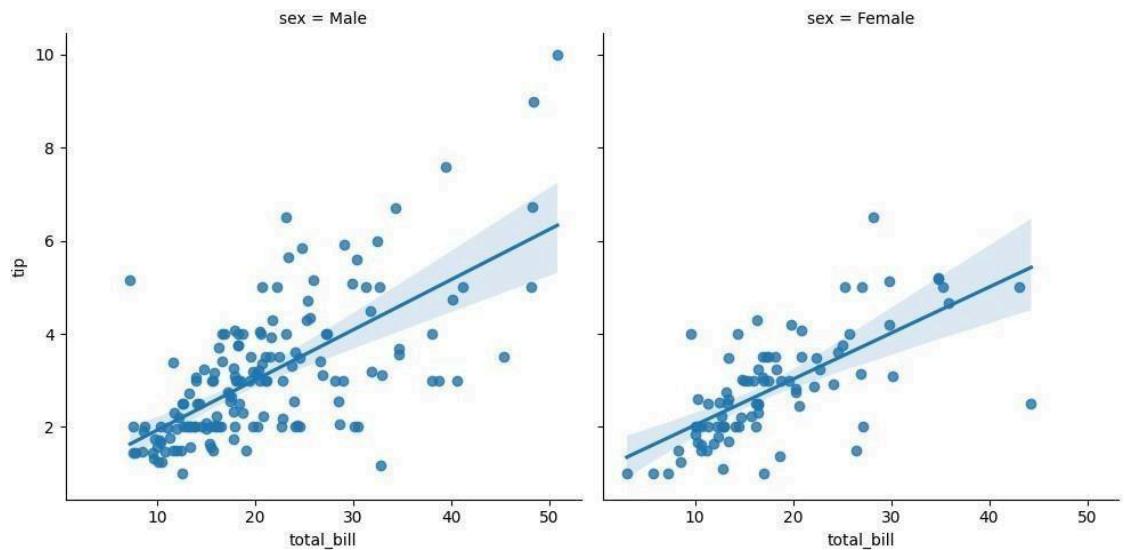
<seaborn.axisgrid.FacetGrid at 0x7fbb6b6793f0>



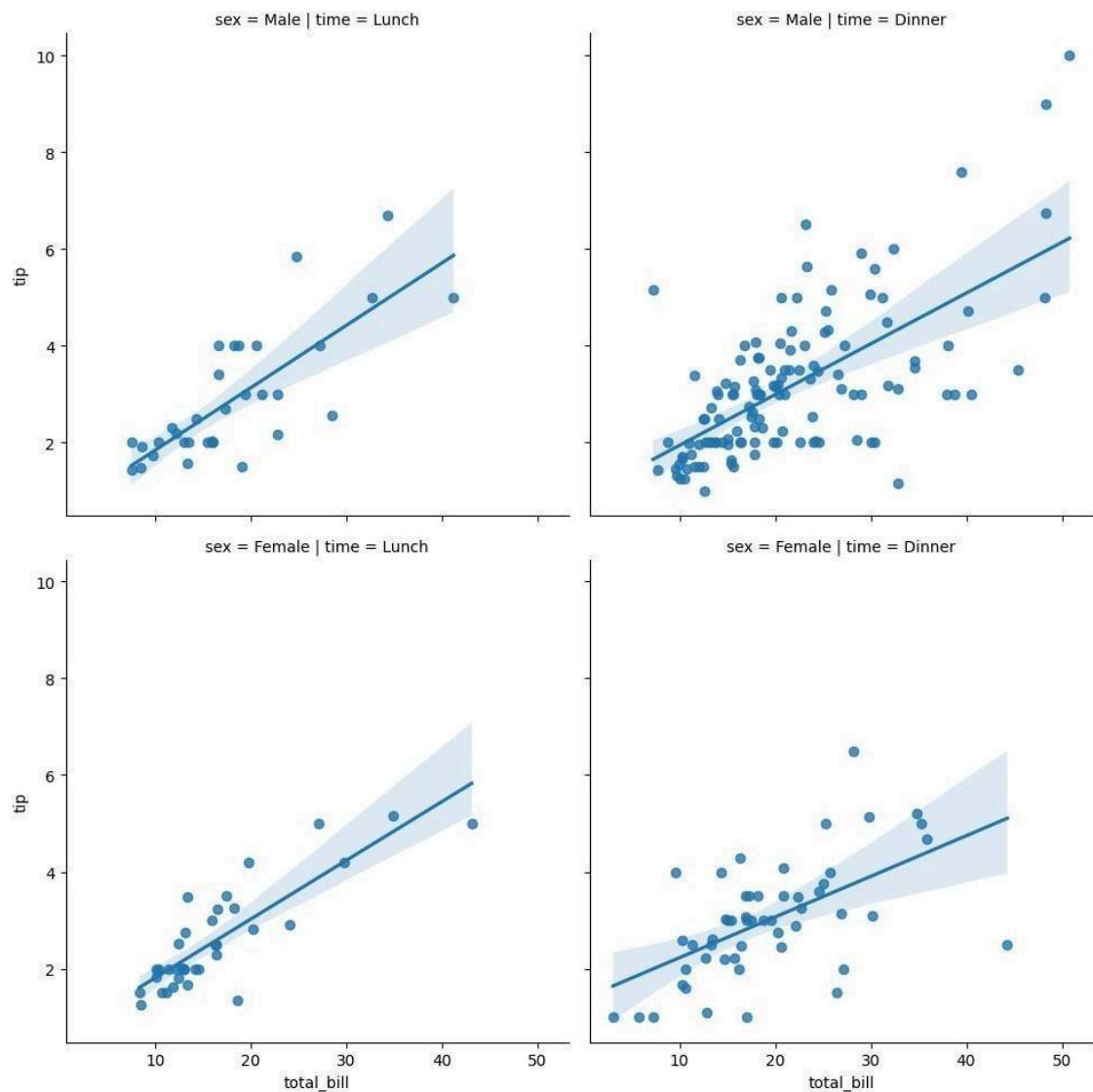
## Using a Grid

We can add more variable separation through columns and rows with the use of a grid. Just indicate this with the col or row arguments:

```
sns.lmplot(x='total_bill', y='tip', data=tips, col='sex')  
<seaborn.axisgrid.FacetGrid at 0x7fbb6b50ec80>
```

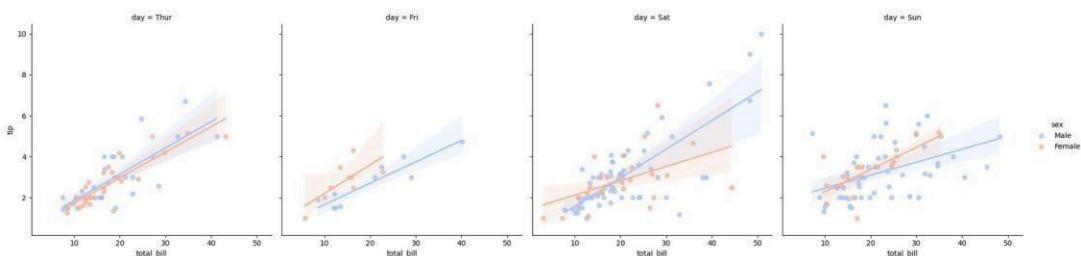


```
sns.lmplot(x="total_bill", y="tip", row="sex", col="time", data=tips)  
<seaborn.axisgrid.FacetGrid at 0x7fbb69a16650>
```



```
sns.lmplot(x='total_bill', y='tip', data=tips, col='day', hue='sex', palette='coolwarm')
```

```
<seaborn.axisgrid.FacetGrid at 0x7fb6924c5e0>
```

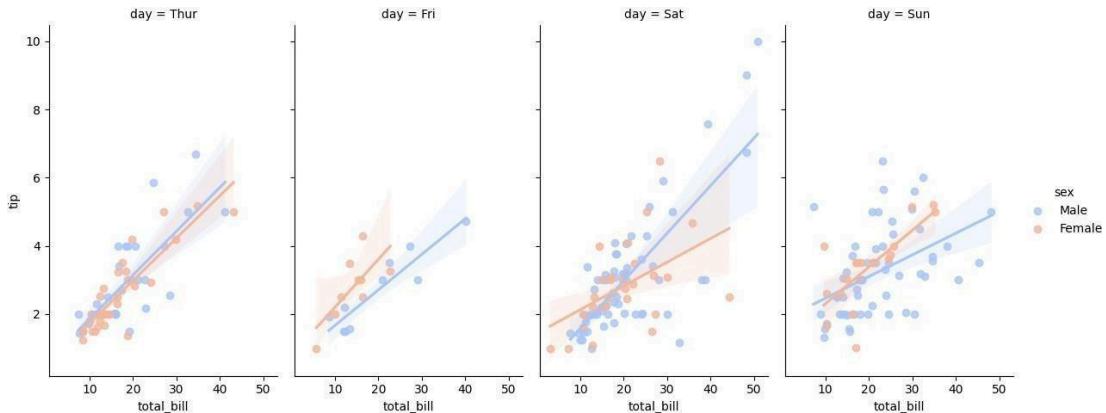


## Aspect and Size

Seaborn figures can have their size and aspect ratio adjusted with the **size** and **aspect** parameters:

```
sns.lmplot(x='total_bill', y='tip', data=tips, col='day', hue='sex', palette='coolwarm', aspect=0.6)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fbb68e65690>
```



You're probably wondering how to change the font size or control the aesthetics even more, check out the Style and Color Lecture and Notebook for more info on that!

## Great Job!

---

## Style and Color

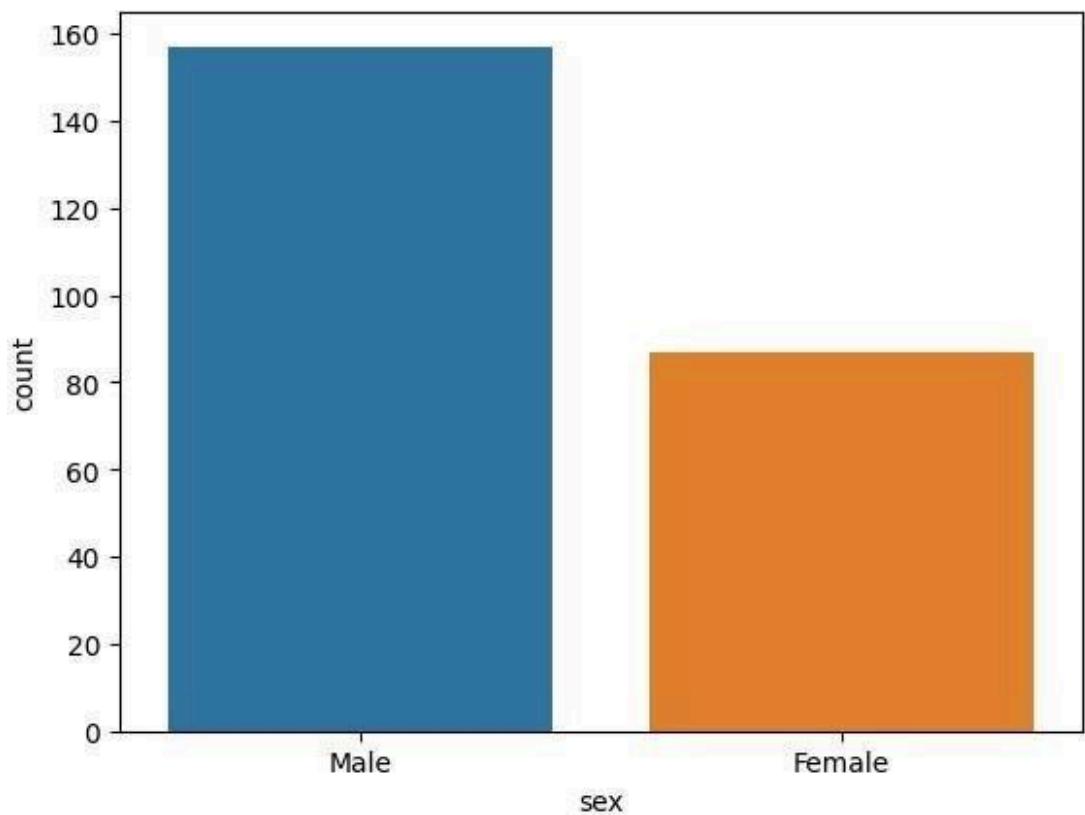
We've shown a few times how to control figure aesthetics in seaborn, but let's now go over it formally:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
tips = sns.load_dataset('tips')
```

### Styles

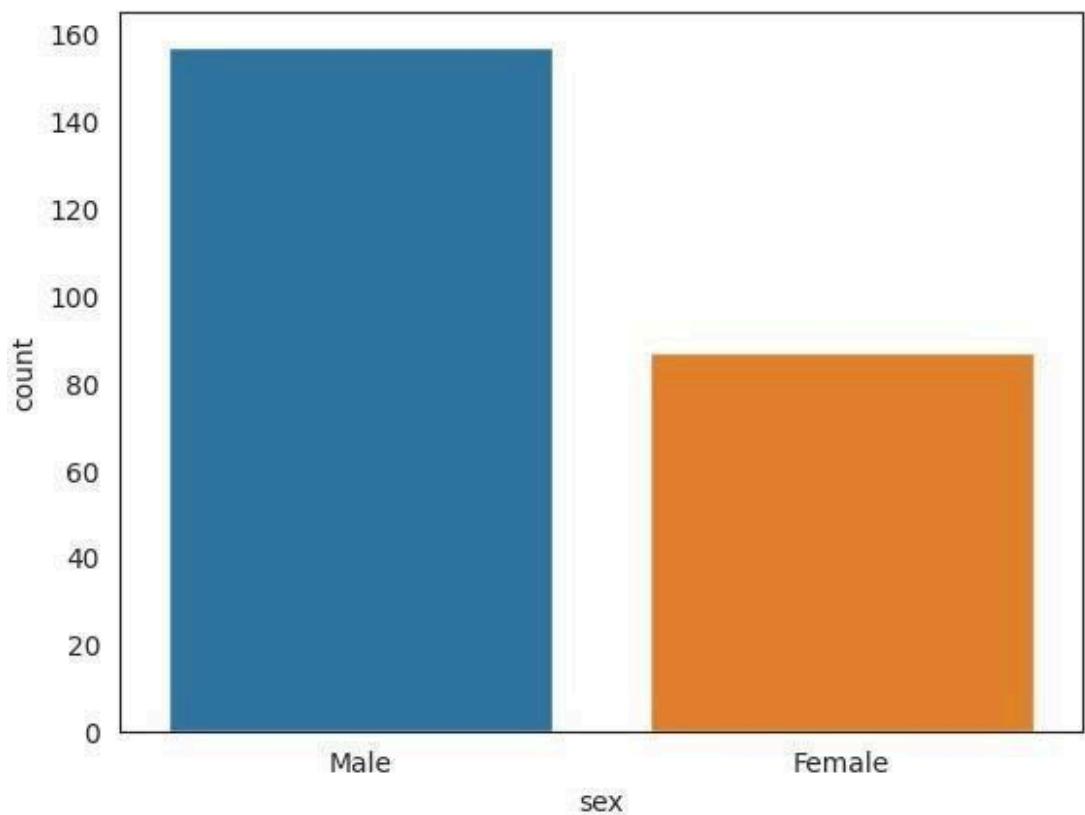
You can set particular styles:

```
sns.countplot(x='sex', data=tips)
<Axes: xlabel='sex', ylabel='count'>
```



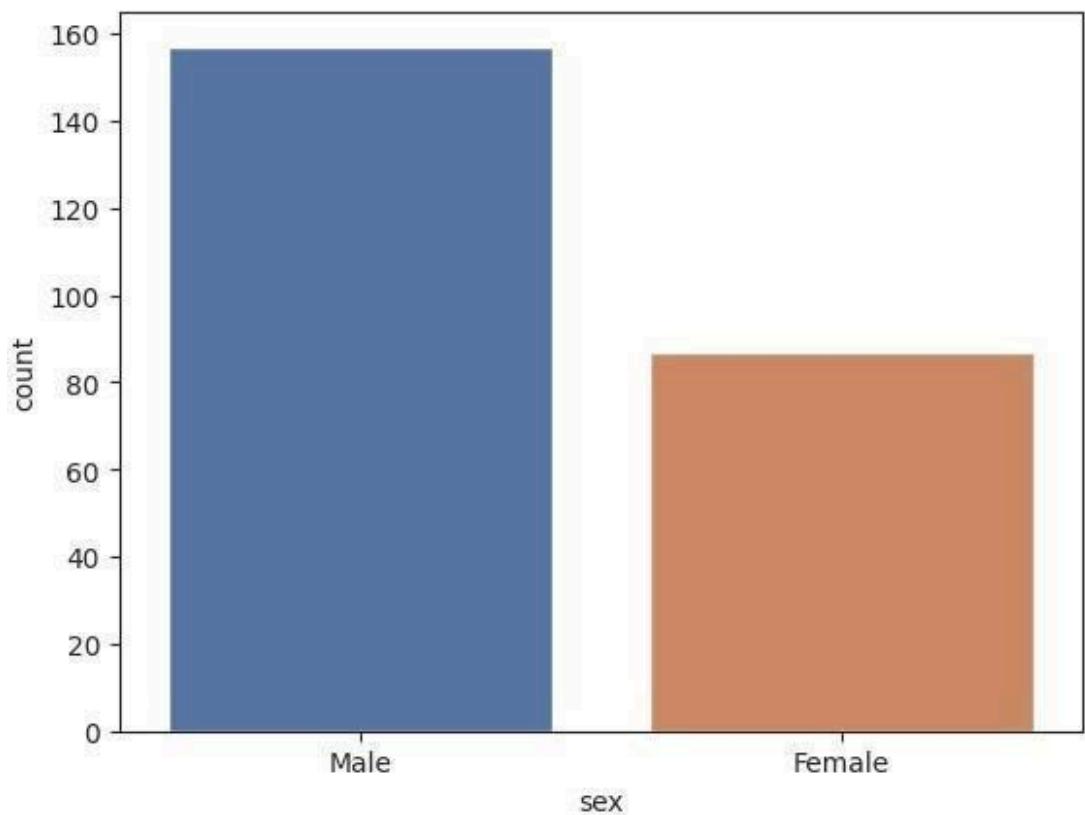
```
sns.set_style('white')
sns.countplot(x='sex', data=tips)

<Axes: xlabel='sex', ylabel='count'>
```



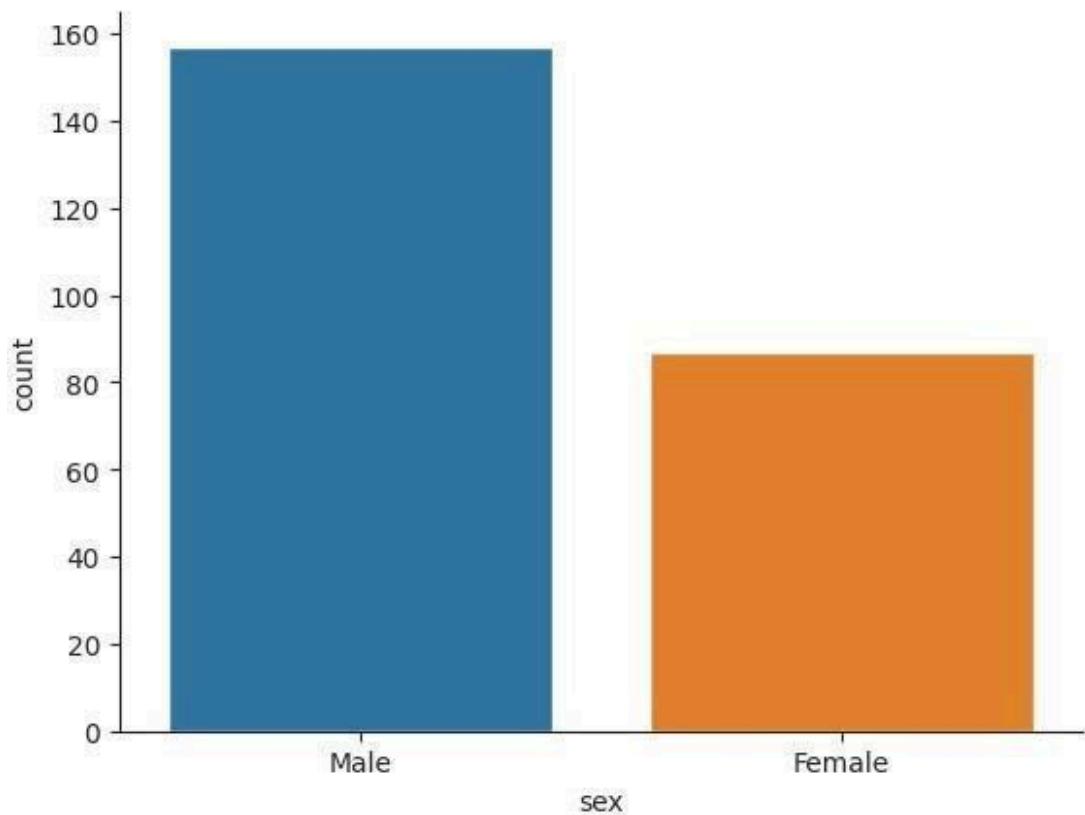
```
sns.set_style('ticks')
sns.countplot(x='sex', data=tips, palette='deep')

<Axes: xlabel='sex', ylabel='count'>
```

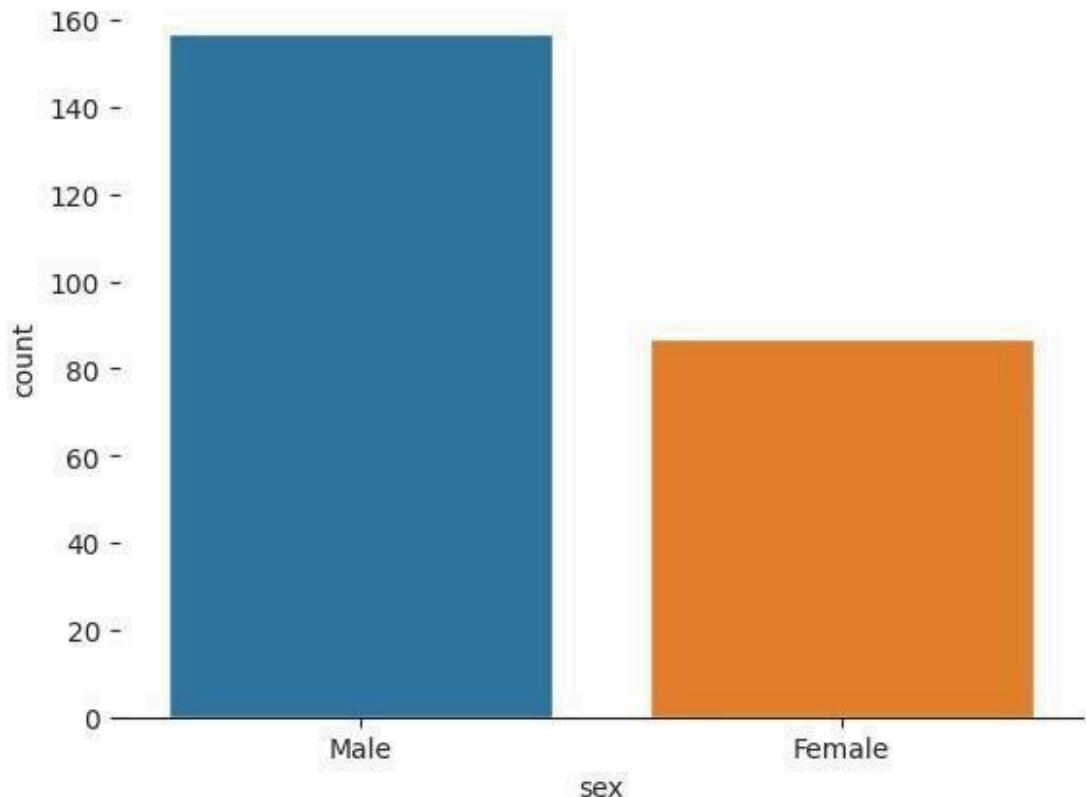


### Spine Removal

```
sns.countplot(x='sex', data=tips  
) sns.despine()
```



```
sns.countplot(x='sex', data=tips)
sns.despine(left=True)
```

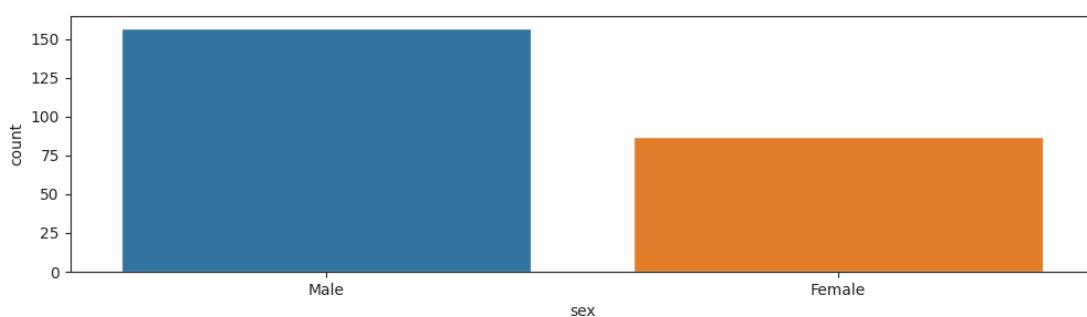


## Size and Aspect

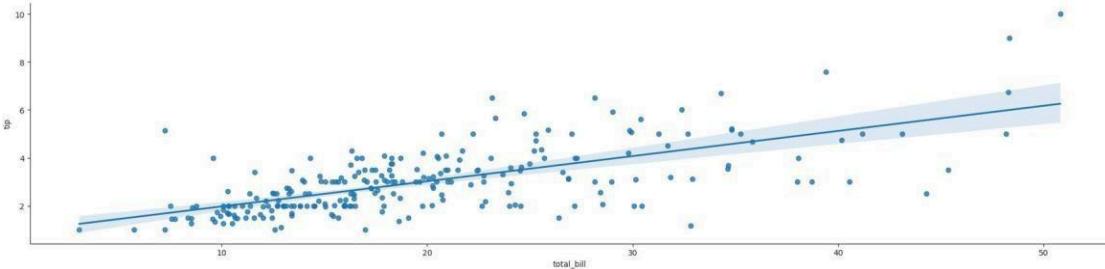
You can use matplotlib's `plt.figure(figsize=(width,height))` to change the size of most seaborn plots.

You can control the size and aspect ratio of most seaborn grid plots by passing in parameters: size, and aspect. For example:

```
# Non Grid Plot
plt.figure(figsize=(12,3))
sns.countplot(x='sex', data=tips
)
<Axes: xlabel='sex', ylabel='count'>
```



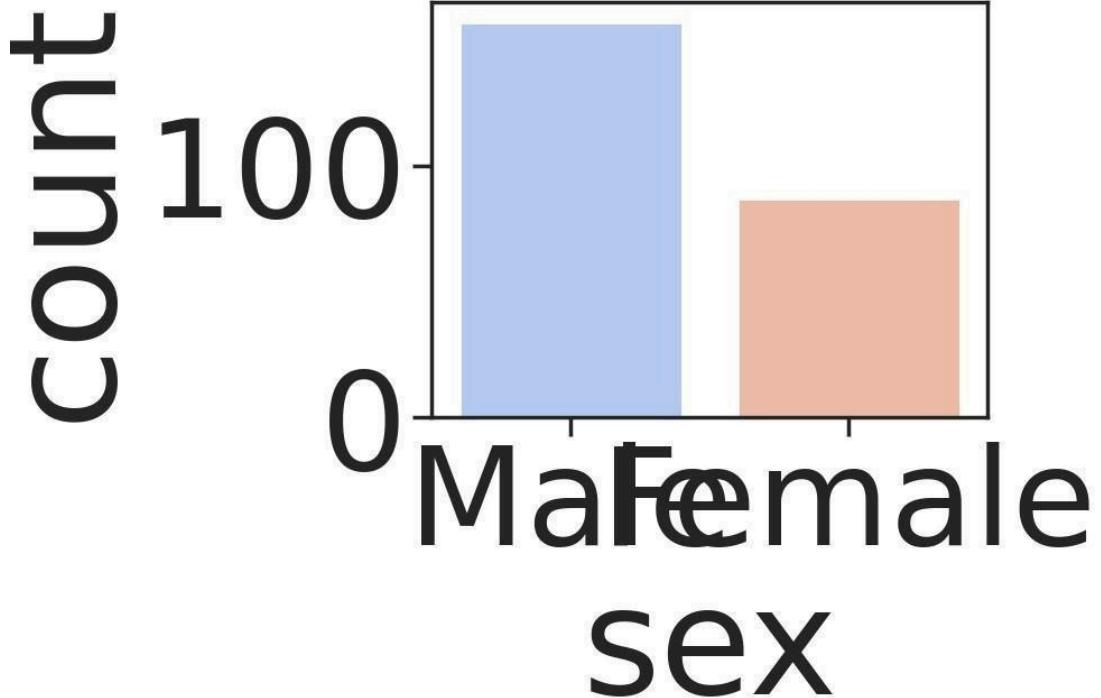
```
# Grid Type Plot
sns.lmplot(x='total_bill', y='tip', aspect=4, data=tips)
<seaborn.axisgrid.FacetGrid at 0x7f9a7a517a30>
```



## Scale and Context

The `set_context()` allows you to override default parameters:

```
sns.set_context('poster', font_scale=4)
sns.countplot(x='sex', data=tips, palette='coolwarm')
<Axes: xlabel='sex', ylabel='count'>
```



Check out the documentation page for more info on these topics:  
<https://stanford.edu/~mwaskom/software/seaborn/tutorial/aesthetics.html>

## Great Job!

---

## Seaborn Exercises

Time to practice your new seaborn skills! Try to recreate the plots below (don't worry about color schemes, just the plot itself).

### The Data

We will be working with a famous titanic data set for these exercises. Later on in the Machine Learning section of the course, we will revisit this data, and use it to predict survival rates of passengers. For now, we'll just focus on the visualization of the data with seaborn:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

sns.set_style('whitegrid')

titanic = sns.load_dataset('titanic')
titanic.head()

      survived
class \
0          0
First
1          1
Second
2          1
Third
3          1
First
4          0
Third

      who adult_male deck embark_town alive alone
0     man        True   Southampton    no   False
```

1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

## Exercises

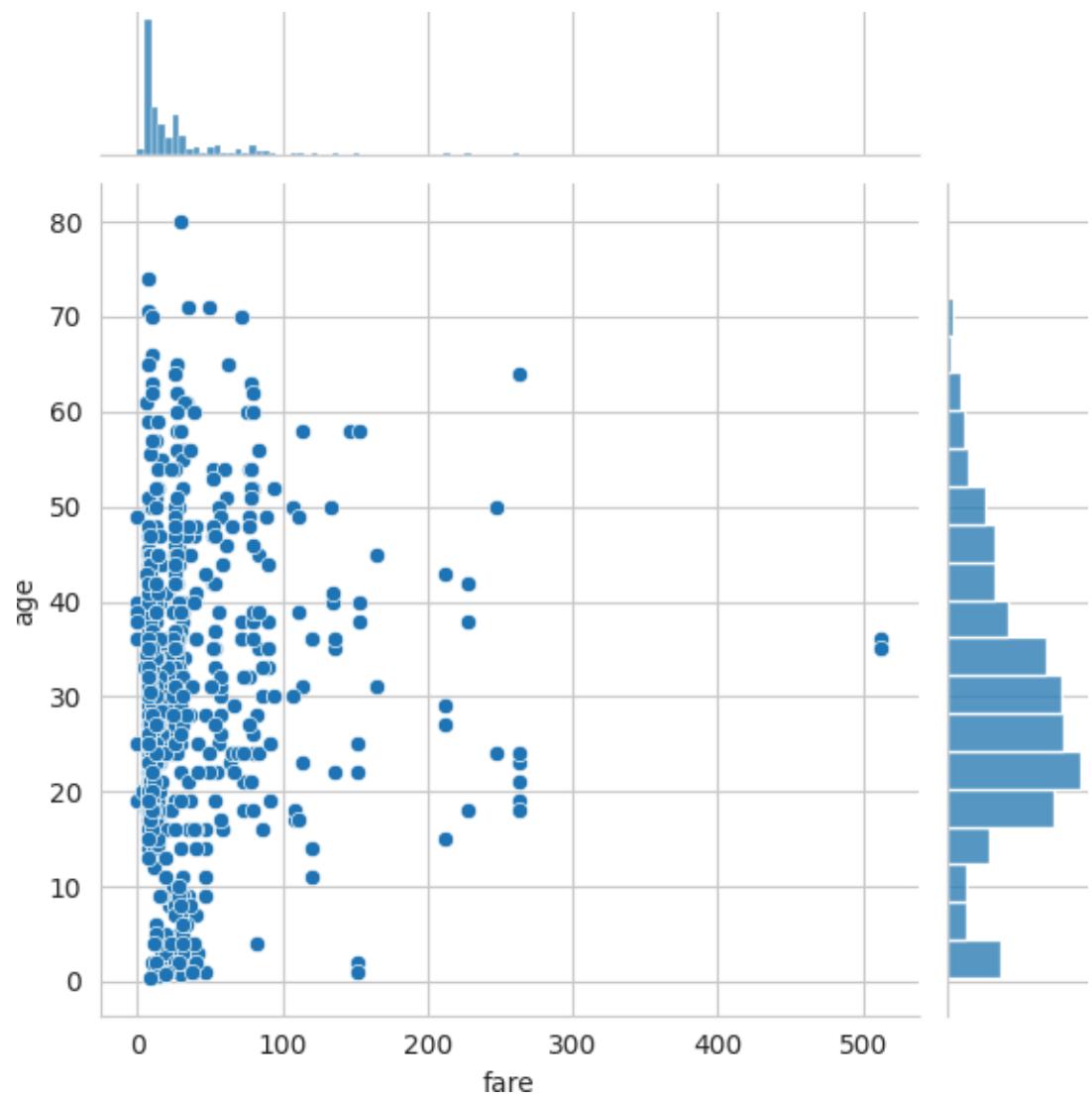
\*\* Recreate the plots below using the titanic dataframe. There are very few hints since most of the plots can be done with just one or two lines of code and a hint would basically give away the solution. Keep careful attention to the x and y labels for hints.\*\*

*\*\* Note! In order to not lose the plot image, make sure you don't code in the cell that is directly above the plot, there is an extra cell above that one which won't overwrite that plot!*

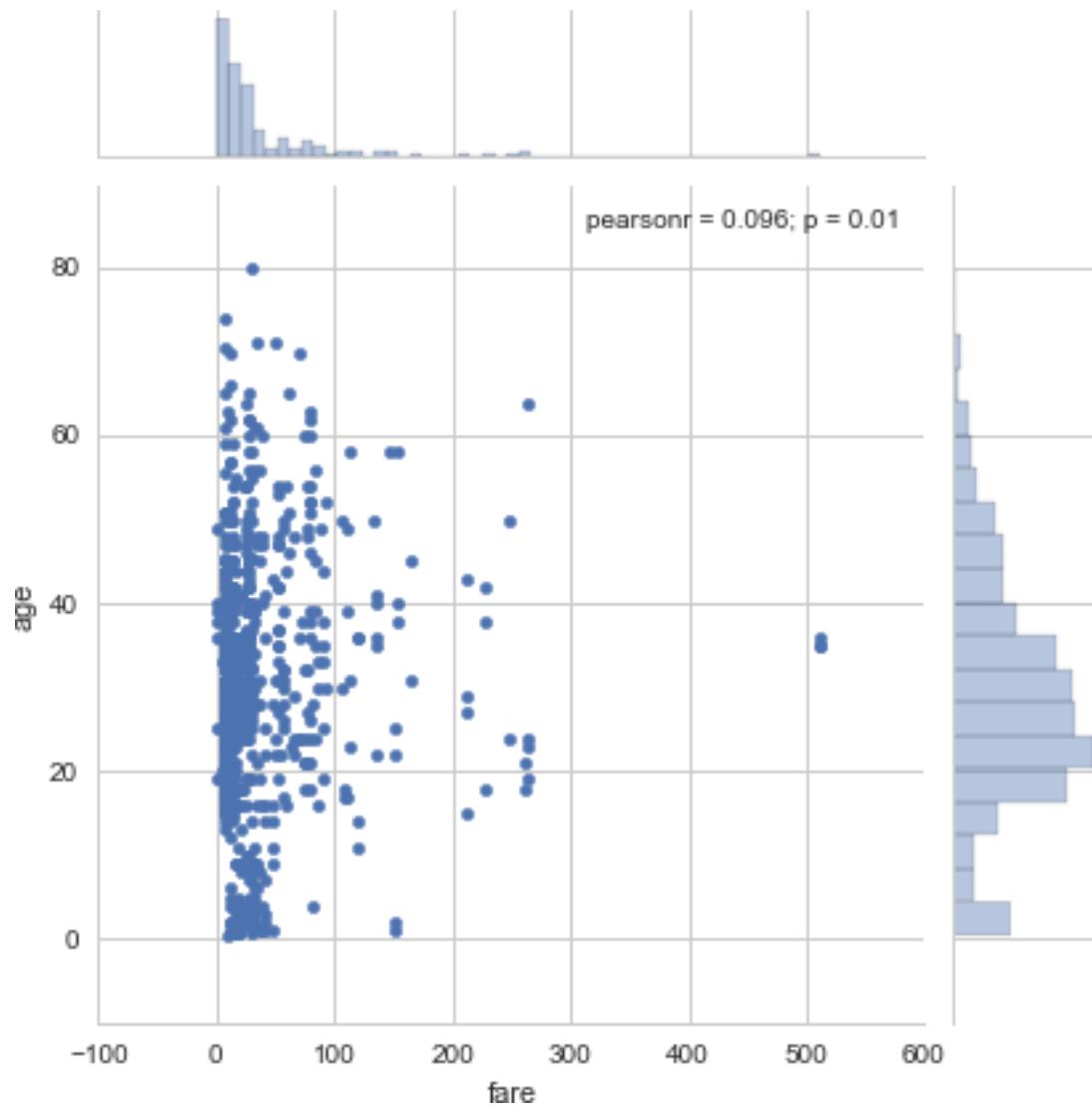
\*\*

```
# CODE HERE
# REPLICATE EXERCISE PLOT IMAGE BELOW
# BE CAREFUL NOT TO OVERWRITE CELL BELOW
# THAT WOULD REMOVE THE EXERCISE PLOT IMAGE!

sns.jointplot(x='fare',y='age',data=titanic)
<seaborn.axisgrid.JointGrid at 0x7f4a1b3d5930>
```



<seaborn.axisgrid.JointGrid at 0x11d0389e8>



```
# CODE HERE
# REPLICATE EXERCISE PLOT IMAGE BELOW
# BE CAREFUL NOT TO OVERWRITE CELL BELOW
# THAT WOULD REMOVE THE EXERCISE PLOT IMAGE!

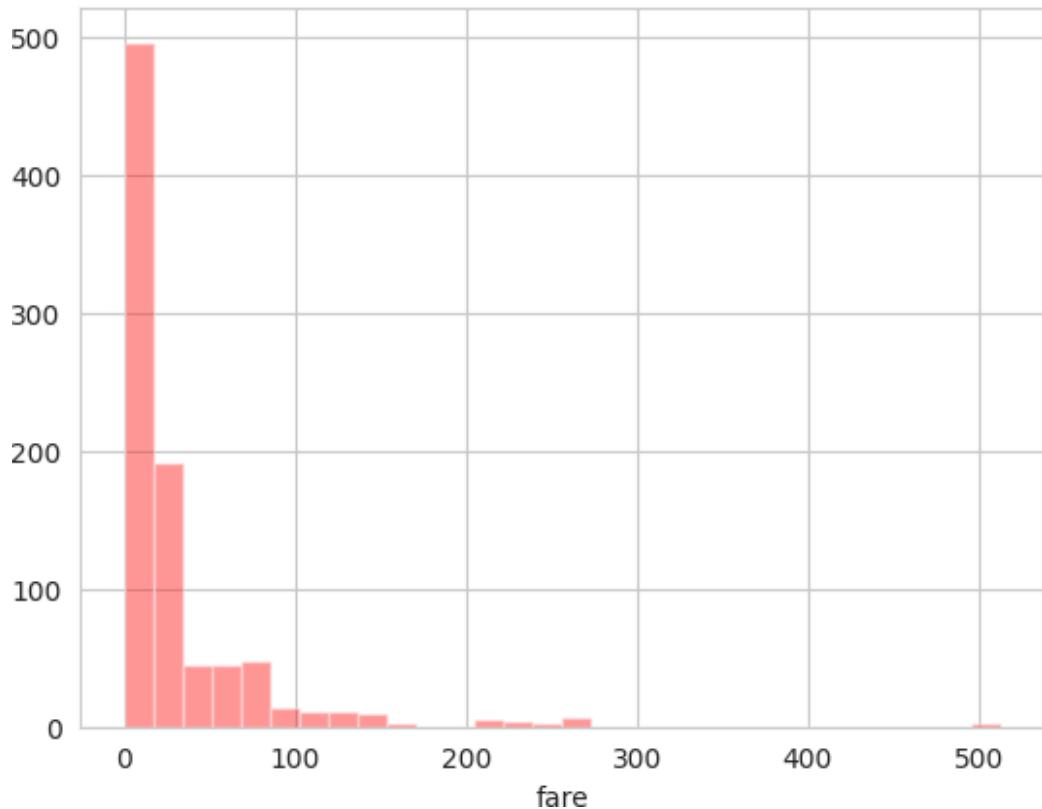
sns.distplot(titanic['fare'], bins=30, kde=False, color='red')

<ipython-input-6-6dee6f31d857>:1: UserWarning:
`distplot` is a deprecated function and will be removed in seaborn
v0.14.0.

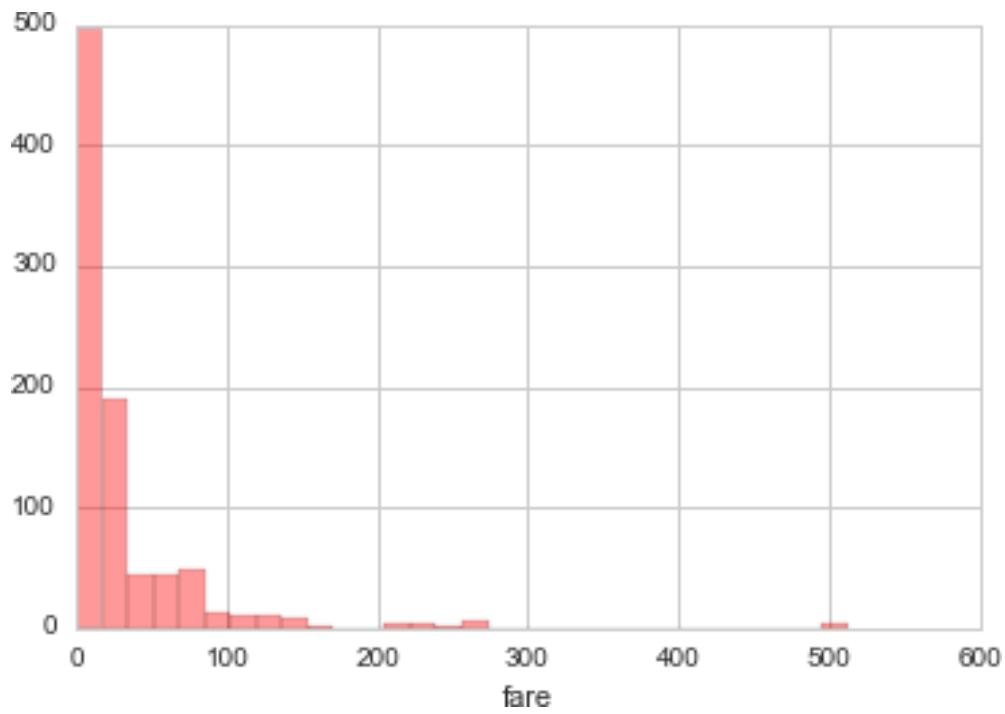
Please adapt your code to use either `displot` (a
figure-level function with
similar flexibility) or `histplot` (an axes-level function for
histograms).
```

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

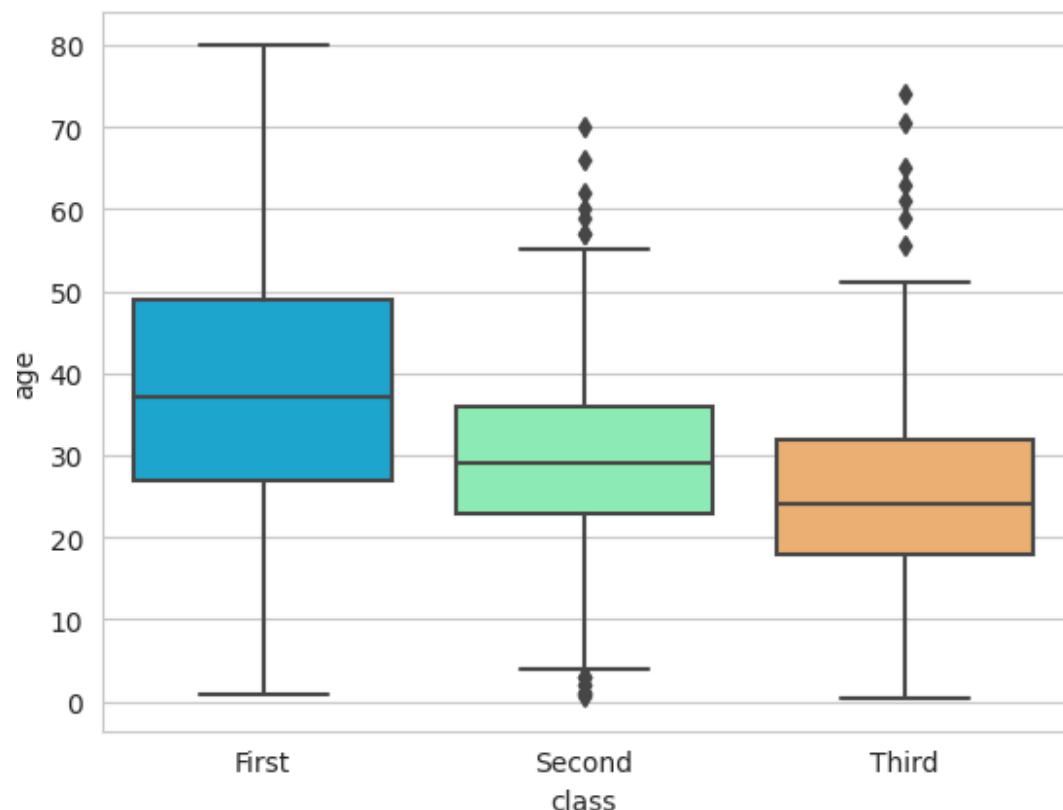
```
sns.distplot(titanic['fare'], bins=30, kde=False, color='red')  
<Axes: xlabel='fare'>
```



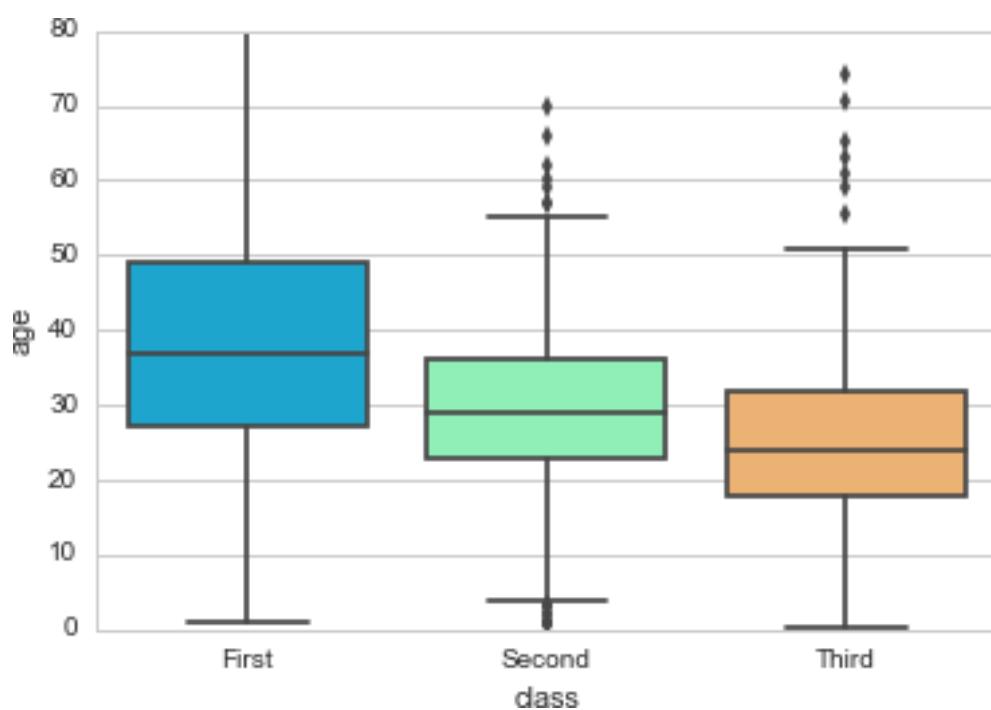
```
<matplotlib.axes._subplots.AxesSubplot at 0x11fc5ca90>
```



```
# CODE HERE
# REPLICATE EXERCISE PLOT IMAGE BELOW
# BE CAREFUL NOT TO OVERWRITE CELL BELOW
# THAT WOULD REMOVE THE EXERCISE PLOT IMAGE!
sns.boxplot(x='class',y='age',data=titanic,palette='rainbow')
<Axes: xlabel='class', ylabel='age'>
```



<matplotlib.axes.\_subplots.AxesSubplot at 0x11f23da90>



```

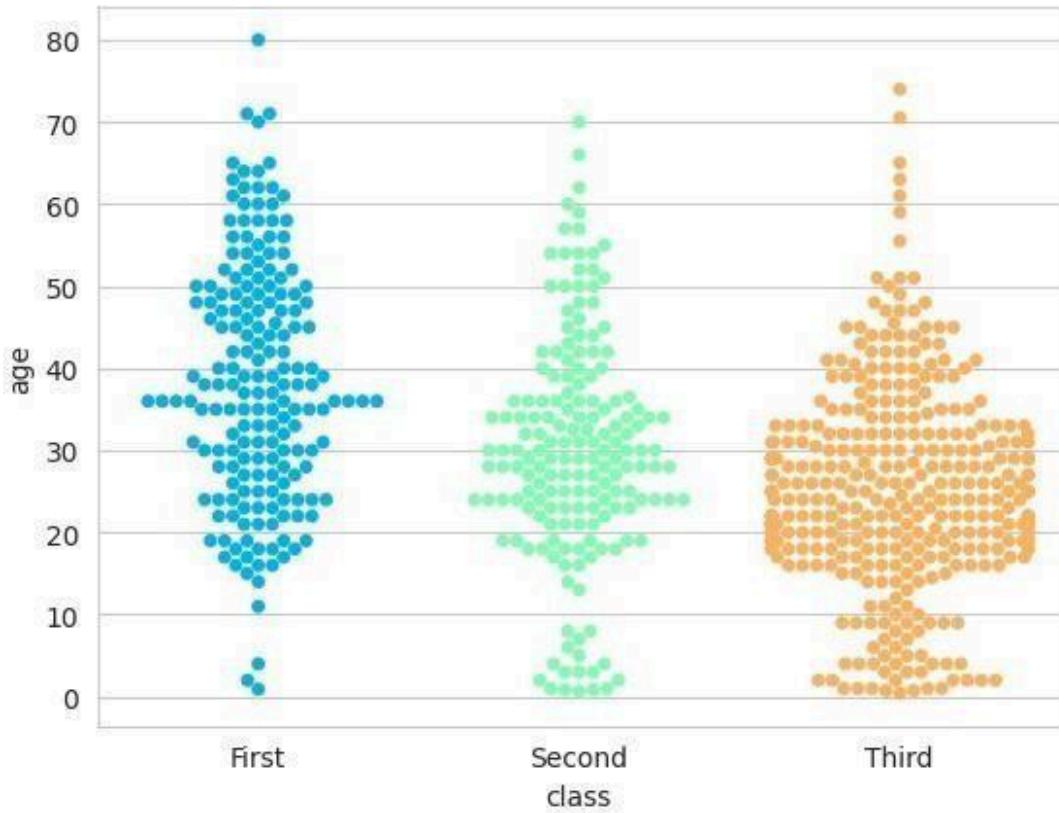
# CODE HERE
# REPLICATE EXERCISE PLOT IMAGE BELOW
# BE CAREFUL NOT TO OVERWRITE CELL BELOW
# THAT WOULD REMOVE THE EXERCISE PLOT IMAGE!
sns.swarmplot(x='class',y='age',data=titanic,palette='rainbow')

<ipython-input-8-f38fc6a5c10f>:5: FutureWarning: Passing `palette` without assigning `hue` is deprecated.
    sns.swarmplot(x='class',y='age',data=titanic,palette='rainbow')

<Axes: xlabel='class', ylabel='age'>

/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544: UserWarning: 15.2% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
    warnings.warn(msg, UserWarning)

```



```

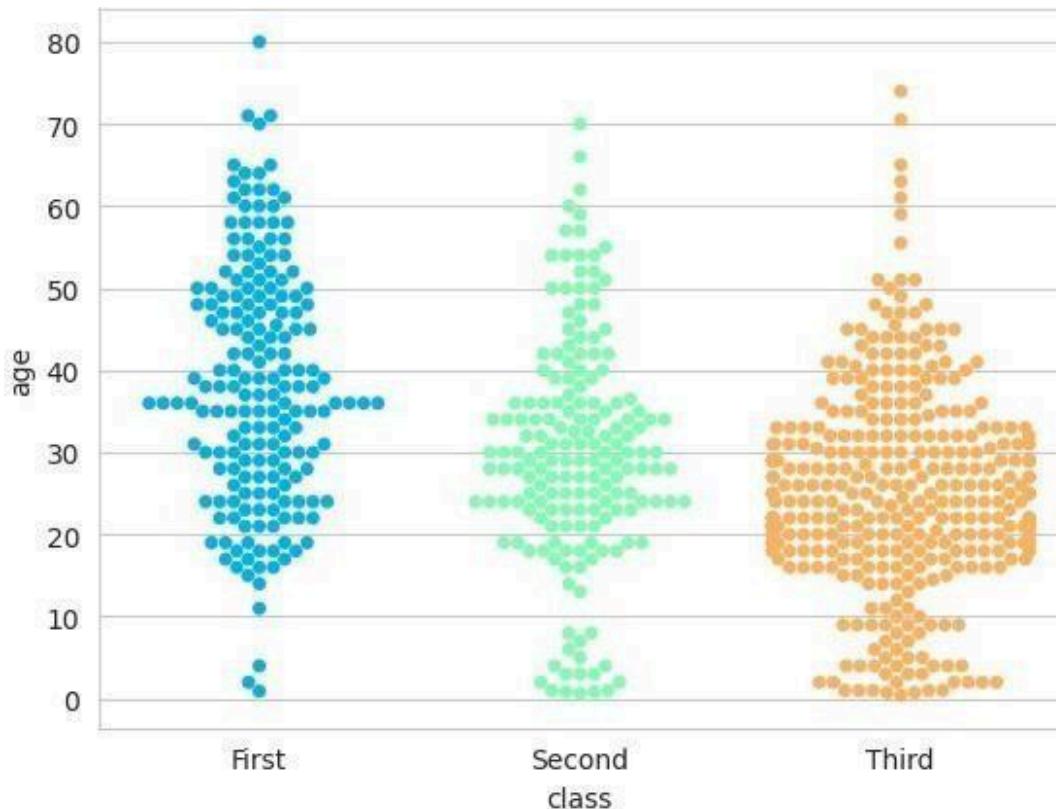
sns.swarmplot(x='class',y='age',data=titanic,palette='rainbow')

<ipython-input-8-3331a63207a1>:1: FutureWarning: Passing `palette` without assigning `hue` is deprecated.
    sns.swarmplot(x='class',y='age',data=titanic,palette='rainbow')

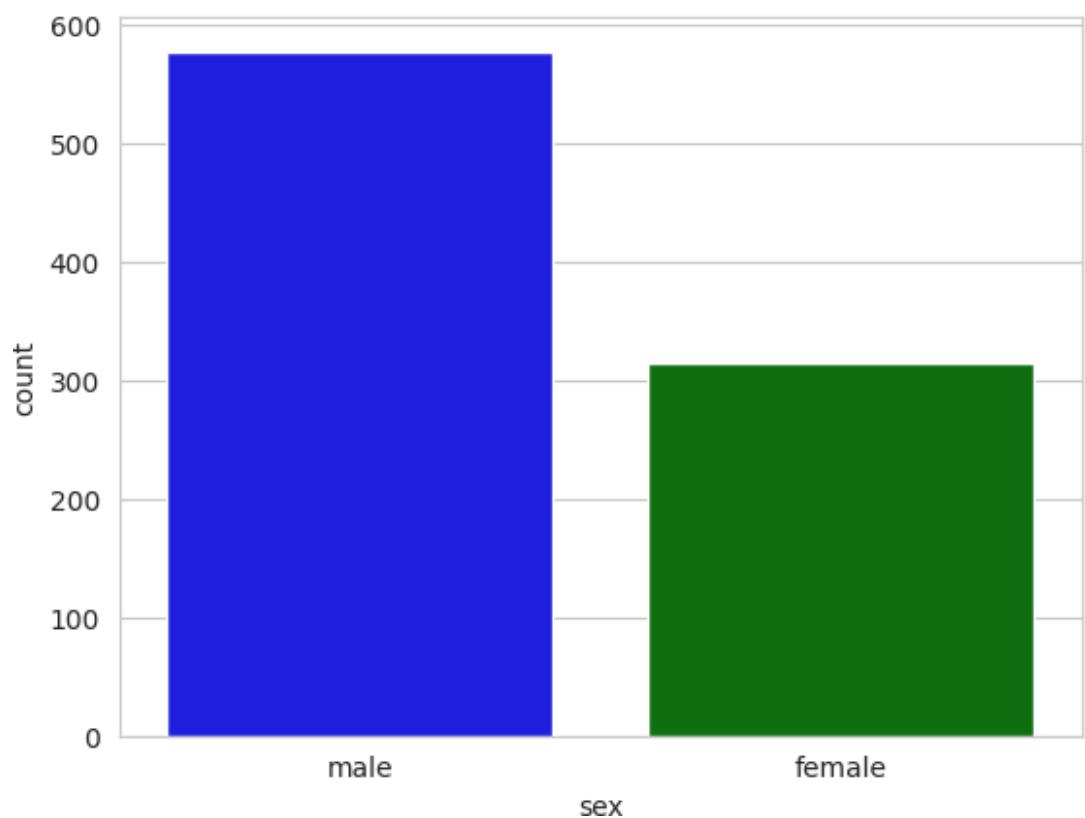
<Axes: xlabel='class', ylabel='age'>

```

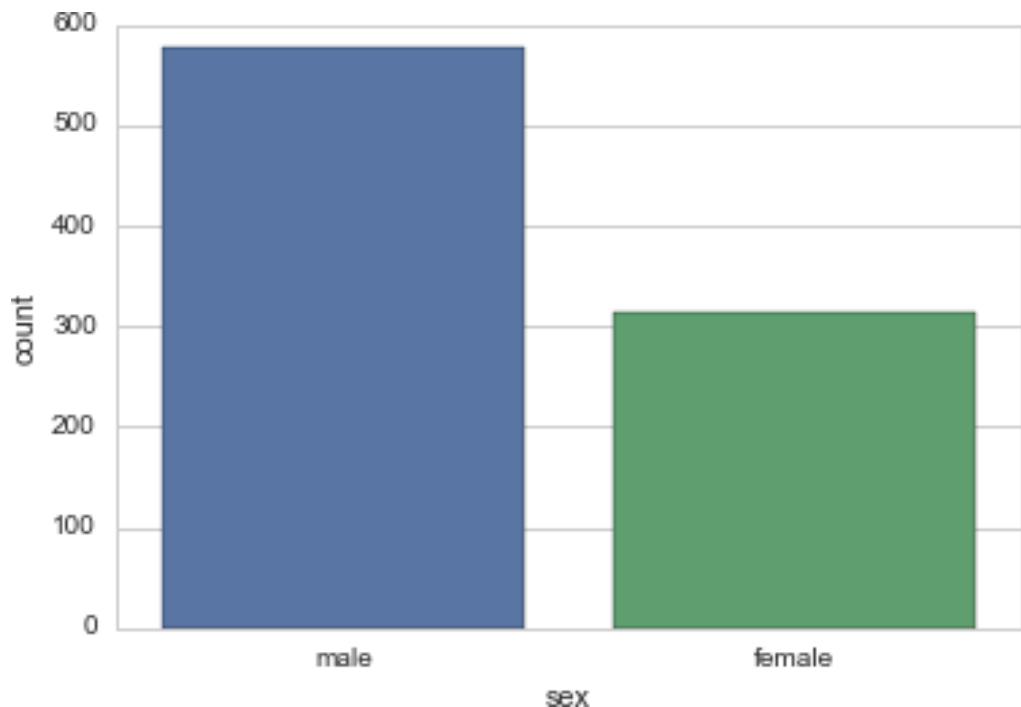
```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3544:  
UserWarning: 15.2% of the points cannot be placed; you may want to  
decrease the size of the markers or use stripplot.  
    warnings.warn(msg, UserWarning)
```



```
# CODE HERE  
# REPLICATE EXERCISE PLOT IMAGE BELOW  
# BE CAREFUL NOT TO OVERWRITE CELL BELOW  
# THAT WOULD REMOVE THE EXERCISE PLOT IMAGE!  
sns.set_palette(['blue', 'green'])  
sns.countplot(x='sex', data=titanic)  
  
<Axes: xlabel='sex', ylabel='count'>
```

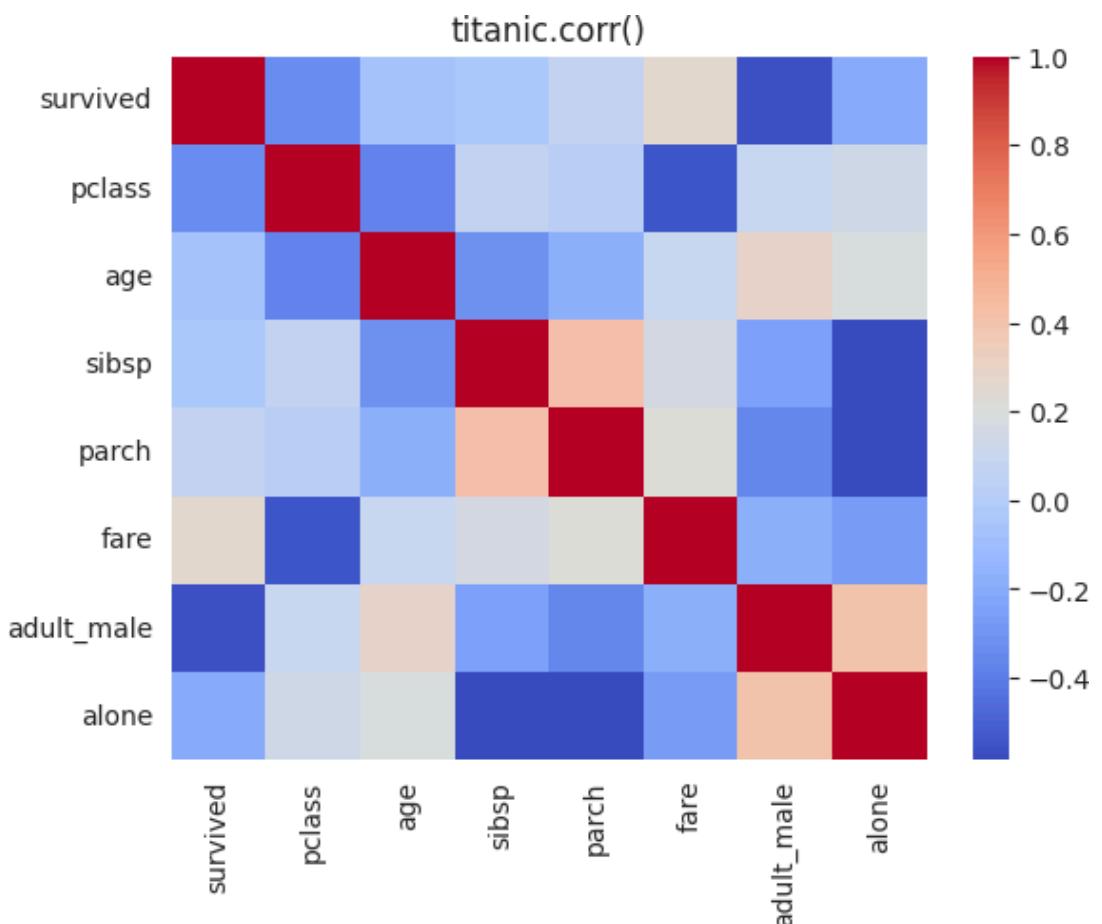


```
<matplotlib.axes._subplots.AxesSubplot at 0x11f207ef0>
```

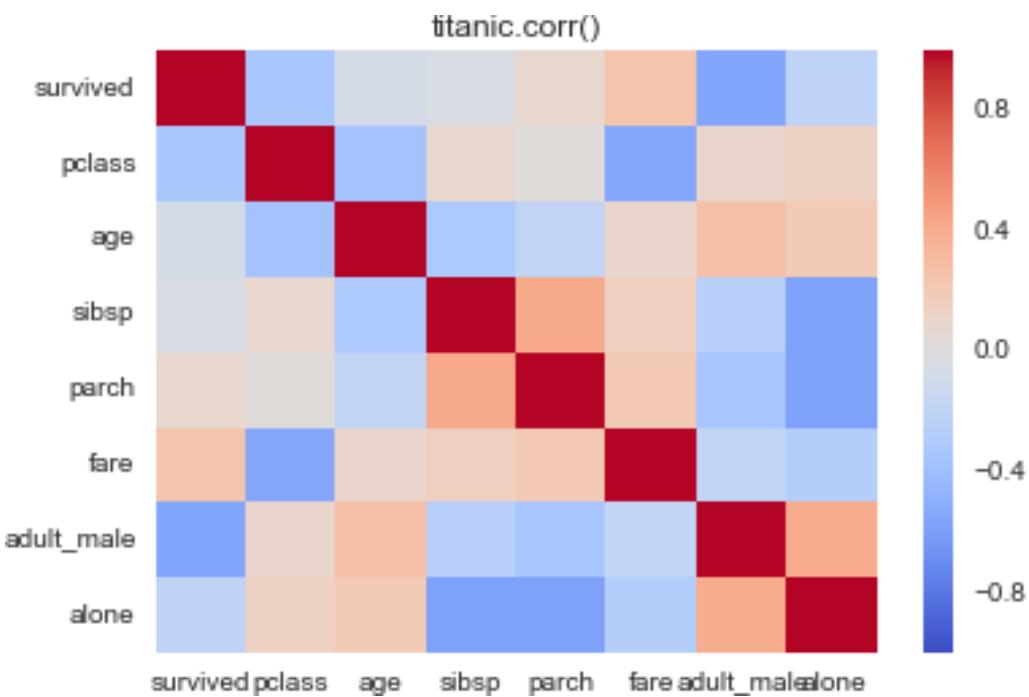


```
# CODE HERE
# REPLICATE EXERCISE PLOT IMAGE BELOW
# BE CAREFUL NOT TO OVERWRITE CELL BELOW
# THAT WOULD REMOVE THE EXERCISE PLOT IMAGE!
sns.heatmap(titanic.corr(),cmap='coolwarm')
plt.title('titanic.corr()')

<ipython-input-17-7f646f80ca08>:5: FutureWarning: The default
value of numeric_only in DataFrame.corr is deprecated. In a future
version, it will default to False. Select only valid columns or
specify the value of numeric_only to silence this warning.
    sns.heatmap(titanic.corr(),cmap='coolw
arm') Text(0.5, 1.0, 'titanic.corr()')
```

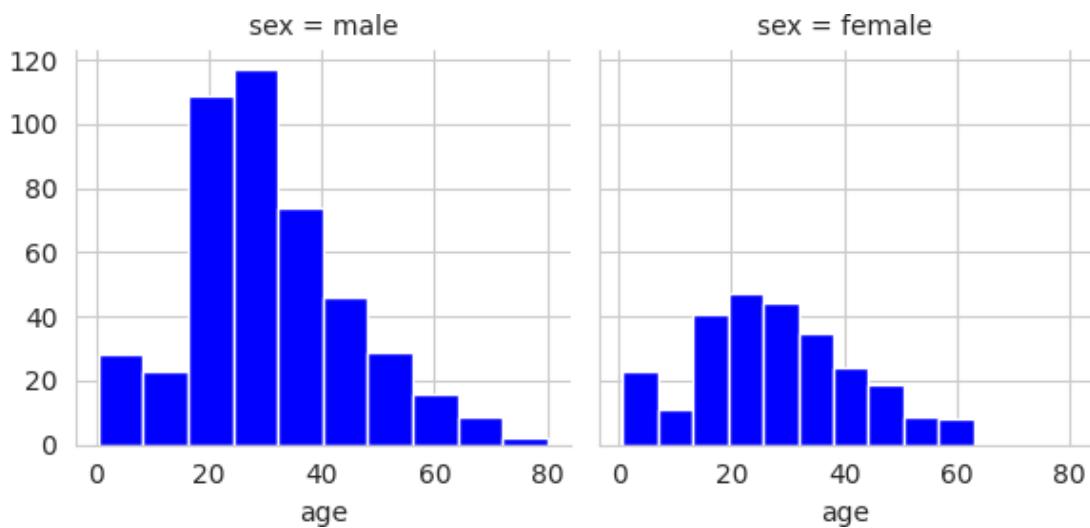


<matplotlib.text.Text at 0x11d72da58>

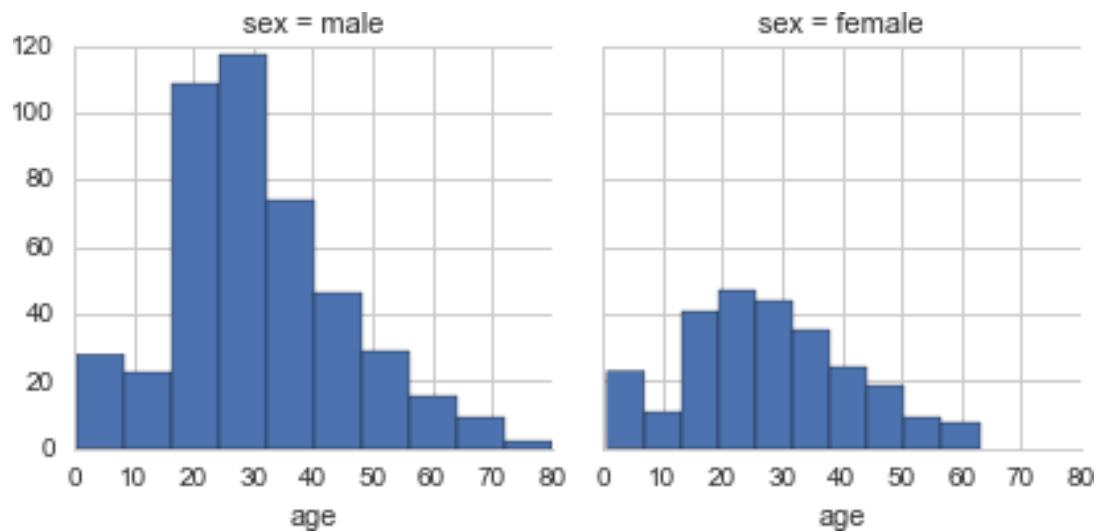


```
# CODE HERE
# REPLICATE EXERCISE PLOT IMAGE BELOW
# BE CAREFUL NOT TO OVERWRITE CELL BELOW
# THAT WOULD REMOVE THE EXERCISE PLOT IMAGE!
g = sns.FacetGrid(data=titanic,col='sex')
g.map(plt.hist,'age')

<seaborn.axisgrid.FacetGrid at 0x7f686bc7a740>
```



```
<seaborn.axisgrid.FacetGrid at 0x11d81c240>
```



**Great Job!**

That is it for now! We'll see a lot more of seaborn practice problems in the machine learning section!

Conclusions: We have successfully implemented Python library - Seaborn in this practical.

## Aim: Implementation of Python Libraries - SciKit

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Here are some of the key features and functions provided by SciKit-learn:

1. Data preprocessing:
  - `preprocessing`: This module includes functions for scaling, standardizing, encoding, and imputing missing values in the data.
2. Supervised learning algorithms:
  - `linear_model`: This module provides implementations of linear regression, logistic regression, and other linear models.
  - `svm`: It includes support vector machine (SVM) algorithms for classification and regression.
  - `neighbors`: This module offers nearest neighbors algorithms, such as K-Nearest Neighbors (KNN).
  - `ensemble`: It contains ensemble methods like random forests and gradient boosting.
3. Unsupervised learning algorithms:
  - `cluster`: This module provides various clustering algorithms, including K-means clustering and hierarchical clustering.
  - `decomposition`: It includes functions for matrix factorization, such as Principal Component Analysis (PCA) and Non-negative Matrix Factorization (NMF).
4. Model selection and evaluation:
  - `model_selection`: This module provides functions for model selection, cross-validation, and hyperparameter tuning.
  - `metrics`: It includes a wide range of evaluation metrics for classification, regression, and clustering tasks.
5. Dimensionality reduction:
  - `decomposition`: This module offers techniques for reducing the dimensionality of datasets, including PCA and Singular Value Decomposition (SVD).
6. Model persistence and utility functions:
  - `model_selection`: This module provides functions for saving and loading models, as well as generating train-test splits.
  - `utils`: It includes various utility functions for data manipulation and model evaluation.

These are just a few examples of the modules and functions available in SciKit-learn. The library provides extensive documentation with detailed explanations and examples for each function, making it easier for users to understand and utilize its capabilities.

```
from sklearn.linear_model import LinearRegression as  
  
Krishnan model =  
Krishnan() model  
  
LinearRegression()  
  
import numpy as np  
from sklearn.model_selection import  
train_test_split x,y =  
np.arange(20).reshape((10,2)), range(10)  
x  
  
array([[ 1],  
       [ 0],  
       [ 2,  3],  
       [ 4,  5],  
       [ 6,  7],  
       [ 8,  9],  
       [10, 11],  
       [12, 13],  
       [14, 15],  
       [16, 17],  
       [18, 19]])  
  
list(y)  
  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The code snippet you provided is using the `train_test_split` function from the `model_selection` module in SciKit-learn. This function is commonly used to split a dataset into training and testing subsets for machine learning tasks.

Here's a breakdown of what each component does:

`x` and `y`: These are the input features and target variable, respectively, that you want to split into training and testing subsets. Typically, `x` represents the input data, while `y` represents the corresponding labels or targets.

`test_size=0.33`: This parameter specifies the proportion of the dataset that should be allocated for testing. In this case, 33% (or 0.33) of the data will be used for testing, and the remaining 67% will be used for training. You can adjust this value to change the size of the testing set.

`random_state=30`: This parameter sets the random seed for reproducibility. By specifying a particular value, in this case, 30, you ensure that the random splitting of the data will be the same every time you run the code. This is useful for obtaining consistent results and comparing different models or experiments.

X\_train, X\_test, y\_train, y\_test: These are the output variables that will hold the resulting training and testing subsets. X\_train will contain the training input features, X\_test will contain the testing input features, y\_train will contain the training target variable, and y\_test will contain the testing target variable.

```
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.33, random_state=30)

X_train

array([[12, 13],
       [ 8,  9],
       [14, 15],
       [16, 17],
       [18, 19],
       [10, 11]])

y_train

[6, 4, 7, 8, 9, 5]

X_test

array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])

y_test

[0, 1, 2, 3]

train_test_split(y,shuffle=False)
e) [[0, 1, 2, 3, 4, 5, 6], [7,
8, 9]]
```

The model.fit(X\_train, y\_train) method is used to train a machine learning model using the training data.

Here's a brief explanation of what this function does:

X\_train: This parameter represents the input features of the training set. It contains the data that the model will use to learn patterns and make predictions.

y\_train: This parameter represents the target variable or labels of the training set. It contains the corresponding correct outcomes for the input features in X\_train.

model.fit(): This method fits the model to the training data by adjusting its internal parameters or coefficients based on the input features (X\_train) and the target variable (y\_train). The model learns the patterns and relationships in the training data during this process.

```
model.fit(X_train,y_train)
```

```
LinearRegression()  
predictions = model.predict(X_test)  
predictions  
array([3.44169138e-15, 1.0000000e+00, 2.0000000e+00,  
3.0000000e+00])
```

Conclusions: I have successfully implemented and understood SciKit library in this practical successfully.

<b>Name of Student:</b> Pushkar Prasad Sane			
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> 3	
<b>Title of Lab Assignment:</b> <b>Implementation of Linear Regression &amp; Logistic Regression</b>			
<b>DOP:</b> 27/1/24		<b>DOS:</b> 03/2/24	
CO:  CO2, CO4	PO:  PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO8, PO9, PO11, PO12, PSO1, PSO12	<b>Faculty signature:</b>	

**Aim:** Implementation of Linear Regression S Logistic Regression

**Linear regression** is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression. The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable. For example, in finance, linear regression might be used to understand the relationship between a company's stock price and its earnings or to predict the future value of a currency based on its past performance.

**Assumption for Linear Regression Model** Linear regression is a powerful tool for understanding and predicting the behavior of a variable, however, it needs to meet a few conditions in order to be accurate and dependable solutions.

**Linearity:** The independent and dependent variables have a linear relationship with one another. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion.

**Independence:** The observations in the dataset are independent of each other. This means that the value of the dependent variable for one observation does not depend on the value of the dependent variable for another observation.

**Homoscedasticity:** Across all levels of the independent variable(s), the variance of the errors is constant. This indicates that the amount of the independent variable(s) has no impact on the variance of the errors.

**Normality:** The errors in the model are normally distributed.

**No multicollinearity:** There is no high correlation between the independent variables. This indicates that there is little or no correlation between the independent variables.

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

df_regression =
pd.read_csv("/content/KingKohli.csv") print("Data
imported successfully") df_regression.head(10)
```

Data imported successfully

	Score	Out/Not	Out	Against	Batting	Order	Inn.	Strike	rate \
0	116.0	NaN		Australia			6	NaN	NaN
1	103.0	Out	New Zealand				5	2.0	111.00
2	103.0	NaN		NaN			5	2.0	123.20
3	107.0	NaN	Australia				5	2.0	NaN
4	119.0	Out		NaN			4	1.0	NaN
5	105.0	Not	Out	New Zealand			4	4.0	NaN
6	115.0	Out		Australia			4	2.0	NaN
7	141.0	Out		Australia			4	4.0	141.23
8	NaN	Out		Australia			4	2.0	NaN
9	147.0	Out		Australia			4	2.0	123.00

Date \	Venue	Column1	H/A
0 2012	Adelaide Oval	Adelaide	NaN 24-01-
1 NaN	M. Chinnaswamy Stadium	Bangalore	Home
2 Vidarbha 2012	Cricket Association Stadium	NaN	Home 13-12-
3 2013		Chennai	Home 22-02-
4 NaN	Wanderers Stadium	Johannesburg	Away
5 2014	Basin Reserve	Wellington	Away 14-02-
6 2014	Adelaide Oval	NaN	Away 09-12-
7 2014	Adelaide Oval	Adelaide	Away 09-12-
8 2014		Melbourne	Away 26-12-
9 06-01- 2015	Sydney Cricket Ground	Sydney	Away

	Result	Format	Man of the Match	Captain
0	Lost	NaN	NaN	No
1	Won	Test	NaN	No
2	NaN	Test	No	NaN
3	Won	NaN	No	No
4	Drawn	Test	No	NaN
5	Drawn	NaN	No	No
6	NaN	Test	No	Yes
7	Lost	Test	NaN	NaN
8	Drawn	NaN	No	No
9	NaN	Test	No	

Yes df\_regression.shape

```
(19, 14)
```

```
df_regression.info()
```

```
<class  
'pandas.core.frame.DataFrame'>  
RangeIndex: 19 entries, 0 to 18  
Data columns (total 14 columns):  
 #   Column           Non-Null Count Dtype   
---  
 0   Score            1 non-null      float64  
     8  
 1   Out/Not Out      1 non-null      object  
     6  
 2   Against          1 non-null      object  
     6  
 3   Batting Order    1 non-null      int64  
     9  
 4   Inn.             1 non-null      float64  
     8  
 5   Strike rate      1 non-null      float64  
     1  
 6   Venue             1 non-null      object  
     5  
 7   Column1          1 non-null      object  
     6  
 8   H/A               1 non-null      object  
     8  
 9   Date              1 non-null      object  
     5  
 1   Result            1 non-null      object  
 0  
 1   Format            1 non-null      object  
 1  
 1   Man of the Match 1 non-null      object  
 2  
 1   Captain           1 non-null      object  
 3  
dtypes: float64(3), int64(1), object(10)  
memory usage: 2.2+ KB
```

```
df_regression.columns
```

```
Index(['Score', 'Out/Not Out', 'Against', 'Batting Order', 'Inn.',  
       'Strike rate', 'Venue', 'Column1', 'H/A', 'Date', 'Result',  
       'Format',  
       'Man of the Match', 'Captain'],  
      dtype='object')
```

```
df_regression.describe()
```

	Score	Batting Order	Inn.	Strike rate
count	18.000000	19.000000	18.000000	11.000000
mean	144.222222	4.263158	2.055556	143.675455
std	47.450972	0.561951	0.937595	37.594088
min	103.000000	4.000000	1.000000	111.000000
25%	104.250000	4.000000	1.250000	122.000000
50%	117.500000	4.000000	2.000000	128.000000
75%	191.750000	4.000000	2.000000	148.500000
max	235.000000	6.000000	4.000000	233.000000

The Key take aways are: Max Score: 95, Min score 17, avg score = 51.48, On an avg stuent study for 5 hours

```
#count na values under entire dataframe  
df_regression.isna().sum()
```

```
Score           1
Out/Not Out     3
Against         3
Batting Order   0
Inn.            1
Strike rate     8
Venue            4
Column1          3
H/A              1
Date             4
Result           4
Format           6
Man of the Match 5
Captain          5
dtype: int64
```

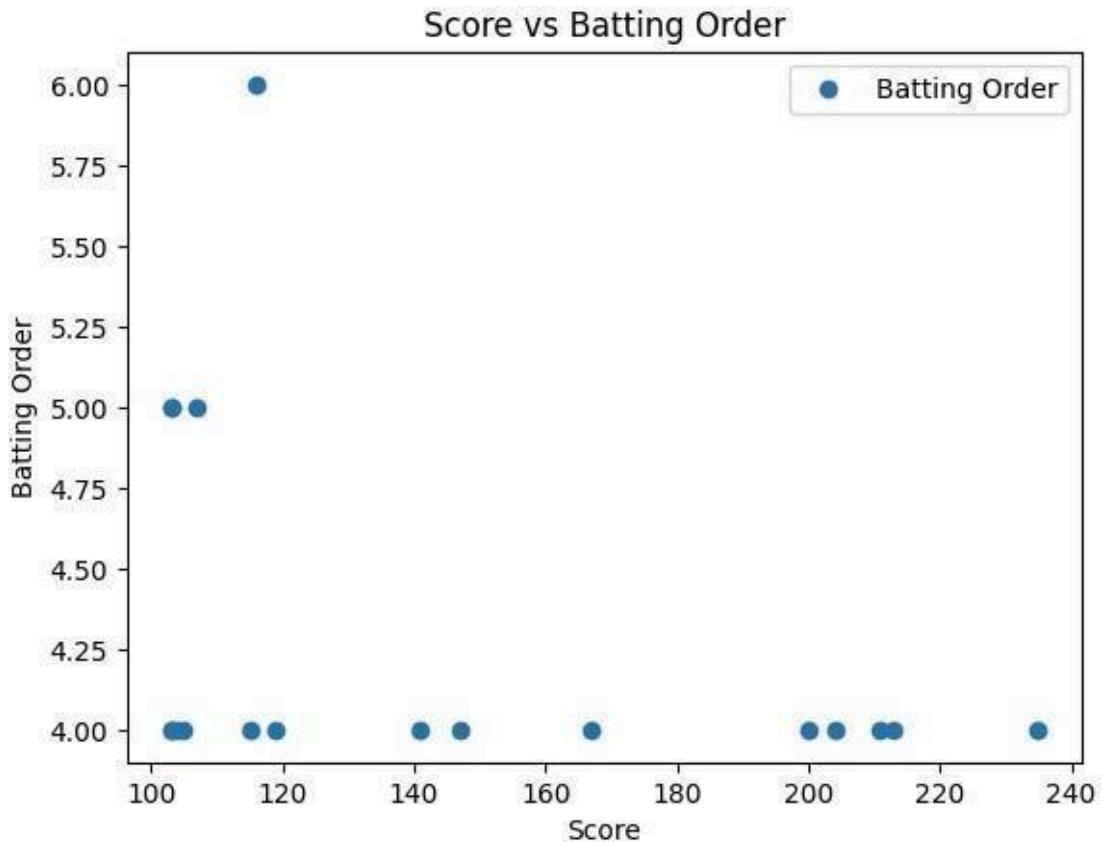
```
#Finding correlation of Dependent & Independent Variables
df_regression.corr()
```

```
<ipython-input-9-d68cf9f442d3>:2: FutureWarning: The default
value of numeric_only in DataFrame.corr is deprecated. In a
future version, it will default to False. Select only valid
columns or specify the value of numeric_only to silence this
warning.
```

```
df_regression.corr()

      Score  Batting Order      Inn.  Strike rate
Score    1.000000       -0.380008 -0.465166   0.271065
Batting Order -0.380008        1.000000 -0.027267  -0.349503
Inn.      -0.465166       -0.027267  1.000000  -0.529315
Strike rate  0.271065       -0.349503 -0.529315   1.000000
```

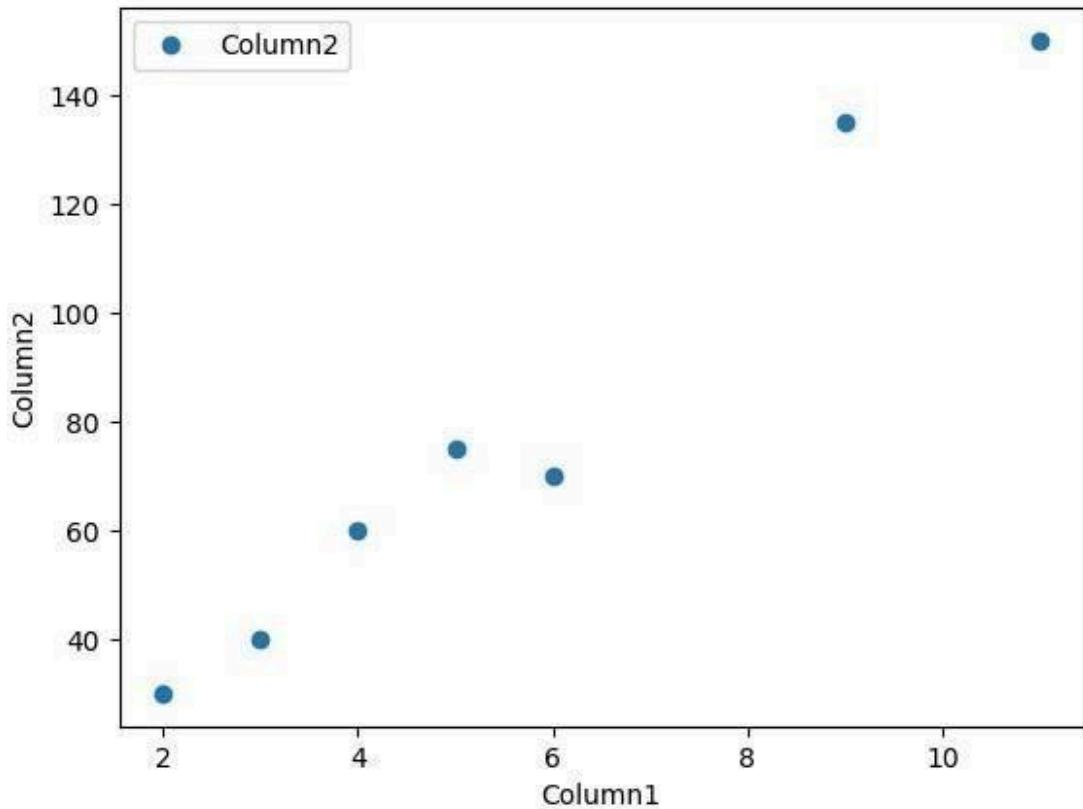
```
#plotting the data to check if relationship is linear
df_regression.plot(x="Score", y="Batting Order", style='o')
plt.title("Score vs Batting Order")
plt.xlabel("Score")
plt.ylabel("Batting
Order") plt.show()
```



Since it is not showing linear relation in my selected dataset. There i am doing it by assuming different x and y values.

```
import pandas as pd
x=[2,4,6,11,9,3,5]
y=[30,60,70,150,135,40,75]
df=pd.DataFrame({'Column1':x,'Column2':y})
df.plot(x="Column1", y="Column2",
style='o') plt.title("Column1 vs
Column2") plt.xlabel("Column1")
plt.ylabel("Column2")
plt.show()
```

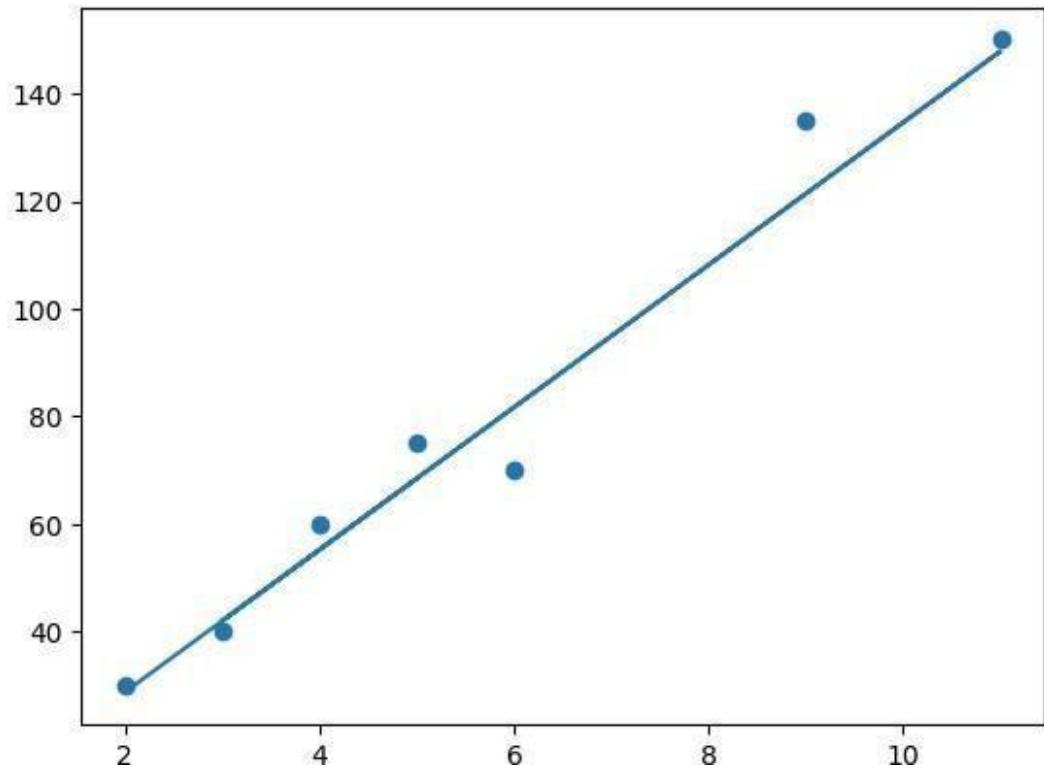
Column1 vs Column2



```
x_regression =  
df.iloc[:, :-1].values  
y_regression =  
df.iloc[:, 1].values #Splitting  
the data  
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(x_regression,  
y_regression, test_size=0.2)  
from sklearn.linear_model import LinearRegression  
regressor = LinearRegression()  
#Fitting the data  
regressor.fit(X_train, y_train)  
print("Training Complete")
```

Training Complete

```
#Plotting the regression line y=mx+c  
line = regressor.coef_*x_regression + regressor.intercept_  
#plotting fot the test data  
plt.scatter(x_regression,  
y_regression)  
plt.plot(x_regression, line)  
plt.show()
```



```

print(X_test) #testing data in hours
y_pred = regressor.predict(X_test) #predicting the scores
[[2]
 [9]]

#Comparing actual vs predicted scores
df = pd.DataFrame({'Actual':y_test,
 'Predicted':y_pred}) df

      Actual    Predicted
0        30    28.659794
1       135   121.391753

regressor.score(X_train, y_train) #Score of our trained model
0.969967985431903

#Calculate Error in Model
from sklearn import metrics
print("Mean Absolute Error: ",
metrics.mean_absolute_error(y_test,y_pre
d)) Mean Absolute Error:
7.474226804123706

R^2(coefficient of determination). Best Possible score is 1.0 and it can be negative

```

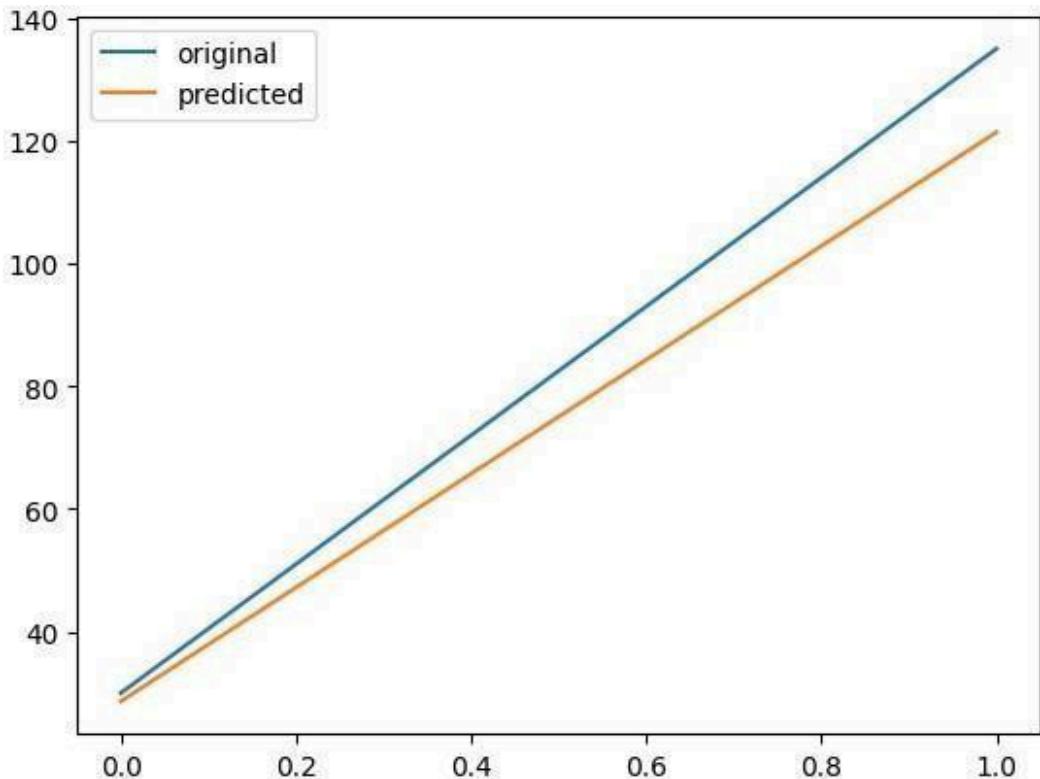
```

print('r2
      score',metrics.r2_score(y_test,y_pred)) r2
score 0.9660806257529759

x_axis =
range(len(y_test))
x_axis
range(0, 2)

#Plotting the values to visualize how well our model works
plt.plot(x_axis,y_test, label='original')
plt.plot(x_axis,y_pred,label='predicted')
plt.legend()
plt.show()

```



## Logistic Regression

It is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression. it's referred to as regression because it takes the output of the linear regression function as input and uses a sigmoid function to estimate the probability for the given class. The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

Terminologies involved in Logistic Regression: Here are some common terms involved in logistic regression:

**Independent variables:** The input characteristics or predictor factors applied to the dependent variable's predictions.

**Dependent variable:** The target variable in a logistic regression model, which we are trying to predict.

**Logistic function:** The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.

**Odds:** It is the ratio of something occurring to something not occurring. it is different from probability as probability is the ratio of something occurring to everything that could possibly occur.

**Log-odds:** The log-odds, also known as the logit function, is the natural logarithm of the odds. In logistic regression, the log odds of the dependent variable are modeled as a linear combination of the independent variables and the intercept.

**Coefficient:** The logistic regression model's estimated parameters, show how the independent and dependent variables relate to one another.

**Intercept:** A constant term in the logistic regression model, which represents the log odds when all independent variables are equal to zero.

**Maximum likelihood estimation:** The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model.

```
import pandas as pd
#reading the dataset using
pandas
df=pd.read_csv('MyData.csv')
print(df)
```

	ID	Name	Gender	Salary	Cost of Item	Purchased
0	101	Adam	M	21000	15000	0
1	102	Steve	F	28000	16678	1
2	103	Josh	M	26000	23444	0
3	104	Joe	F	25000	16999	1
4	105	Johny	M	19000	21000	0
5	106	Marnus	M	31000	84000	0
6	107	Maxwell	M	37000	69000	0
7	108	Kane	M	15000	12333	0
8	109	Chrish	M	18000	15555	0
9	110	David	M	24000	11870	1

This dataset consist information of different user with their id,name,gender and salary. Also one column named as cost of item is there which tells the price of different items

available. here salary and cost of item are independent variable and using these two variable we are going to predict whether user will purchase that item or not. so here we can say that Purchased column is independent variable.

### **Creating inputs and output(target) and splitting the inputs and targets into training and testing set**

```
from sklearn.model_selection import train_test_split
X=df[['Salary','Cost of Item']].values
Y=df[['Purchased']].values
x_train, x_test, y_train,
y_test=train_test_split(X,Y,test_size=0.25,random_state=0)
```

### **Performing feature scaling to get accurate result of the prediction**

```
from sklearn.preprocessing import StandardScaler
st_x=StandardScaler()
x_train=st_x.fit_transform(x_train)
x_test=st_x.fit_transform(x_test)
```

### **Fitting the training sets of x\_train and y\_train into logistic regression model**

```
from sklearn.linear_model import LogisticRegression
lm=LogisticRegression(random_state=0)
lm.fit(x_train,y_train)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/
validation.py:1143: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y
to (n_samples, ), for example using ravel().
    y = column_or_1d(y,
warn=True)

LogisticRegression(random_state
```

### **Predicting the logistic regression**

#### **model**

```
y_pred=lm.predict(x_test)
print(y_pred)

[0 1 0]
```

### **Printing the accuracy and confusion Matrix**

**Confusion Matrix** is a matrix that summarizes the performance of a machine learning model on a set of test data. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance. The matrix displays the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) produced by the model on the test data.

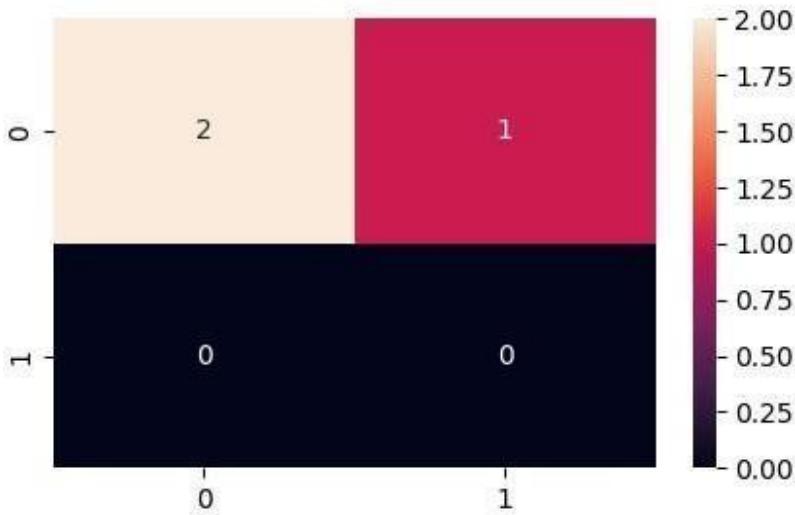
For binary classification, the matrix will be of a 2X2 table, For multi-class classification, the matrix shape will be equal to the number of classes i.e for n classes it will be nXn.

```
from sklearn.metrics import confusion_matrix as cm, accuracy_score
print(accuracy_score(y_test,y_pred))
df_cm=cm(y_test,y_pred)
print(df_cm)

0.6666666666666666
[[2 1]
 [0 0]]
```

```
import seaborn as sn
import matplotlib.pyplot as plt
plt.figure(figsize=(5,3))
sn.heatmap(df_cm, annot=True)
```

<Axes: >



**Conclusion:** I have successfully implemented and understood Linear Regression in this practical successfully.

<b>Name of Student:</b> Pushkar Prasad Sane			
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> 4	
<b>Title of Lab Assignment:</b> Implementation of KNN			
<b>DOP:</b> 03/2/24		<b>DOS:</b> 9 / 2 / 2 4	
CO:  CO4	PO:  POf1, PO2,PO3, PO4, PO5, PO6, PO7, PO8, PO9, PO11, PO12, PSO1, PSO2	<b>Faculty signature:</b>	

**KNN** K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suited category by using K- NN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

**Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

## Importing libraries

```
import numpy as Sidharth_np  
import matplotlib.pyplot as Sidharth_plt  
import pandas as Sidharth_pd
```

## Importing dataset

```
ds = Sidharth_pd.read_csv("myData.csv")  
ds.head(10)
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1

```

8 15600575    Male   25          33000      0
9 15727311    Female  35          65000      0

```

## Extracting Independent and dependent Variables

```

x =
ds.iloc[:, [2, 3]].values
y = ds.iloc[:, 4].values

```

## Splitting dataset into training and test set

```

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x, y, test_size=0.25, random_state=0)

```

## Feature scaling

```

from sklearn.preprocessing import StandardScaler
st_x=StandardScaler()
x_train=st_x.fit_transform(x_train)
x_test=st_x.transform(x_test)

```

## Fitting K-NN classifier to the training set

```

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=5,
metric='minkowski', p=2)
classifier.fit(x_train, y_train)

KNeighborsClassifier()

```

## Predicting the test set result

```

y_pred =
classifier.predict(x_test)
y_pred

array([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, 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, 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,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

```

## Creating confusion matrix

```

from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test, y_pred)
cm

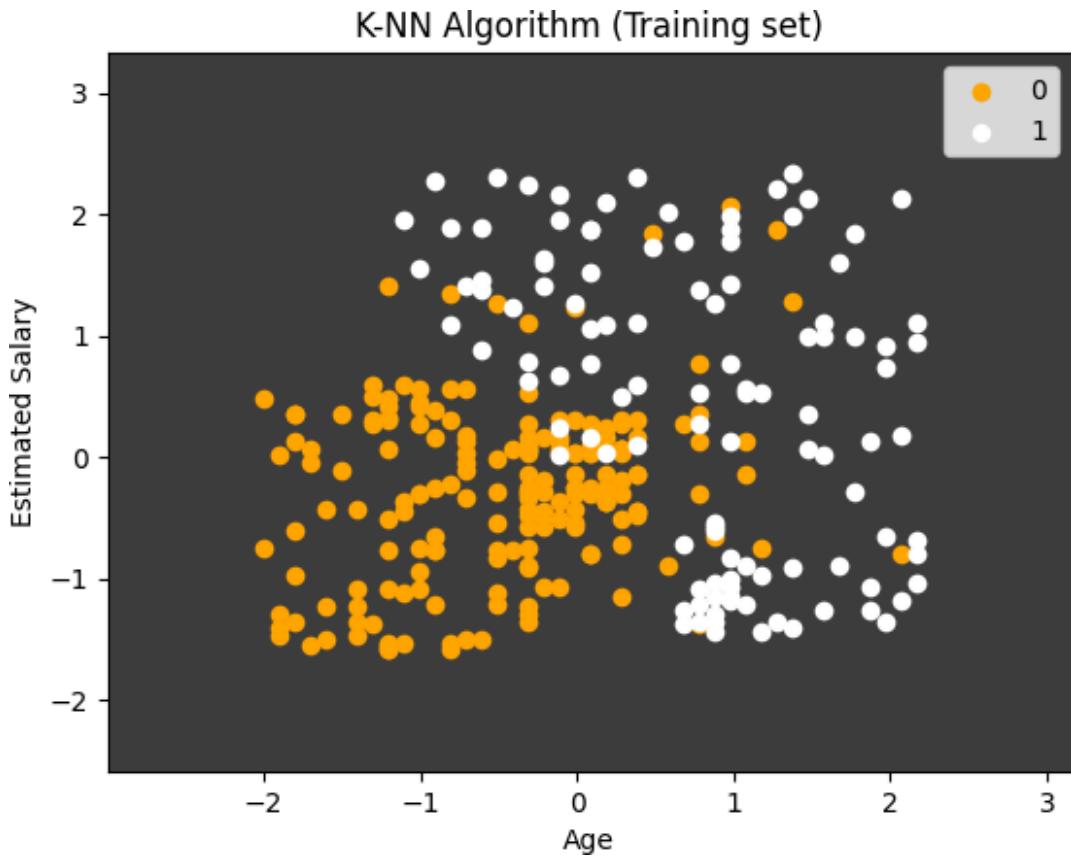
```

```
array([[68, 0],
       [32, 0]])
```

## Visualising the trianing set result

```
from matplotlib.colors import
ListedColormap x_set, y_set = x_train,
y_train
x1, x2 =Sidharth_np.meshgrid(Sidharth_np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
Sidharth_np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
Sidharth_plt.contourf(x1, x2,
classifier.predict(Sidharth_np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1. shape), alpha = 0.75, cmap =
ListedColormap(('black','black' )))
Sidharth_plt.xlim(x1.min(), x1.max())
Sidharth_plt.ylim(x2.min(), x2.max())
for i, j in enumerate(Sidharth_np.unique(y_set)):
    Sidharth_plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c
= ListedColormap(('orange', 'white'))(i), label
= j) Sidharth_plt.title('K-NN Algorithm
(Training set)') Sidharth_plt.xlabel('Age')
Sidharth_plt.ylabel('Estimated Salary')
Sidharth_plt.legend()
Sidharth_plt.show()

<ipython-input-16-38f851b02f78>:9: UserWarning: *c* argument
looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length
matches with *x* & *y*. Please use the *color* keyword-argument
or provide a 2D array with a single row if you intend to specify
the same RGB or RGBA value for all points.
    Sidharth_plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c
= ListedColormap(('orange', 'white'))(i), label = j)
```



### Visualising the test result

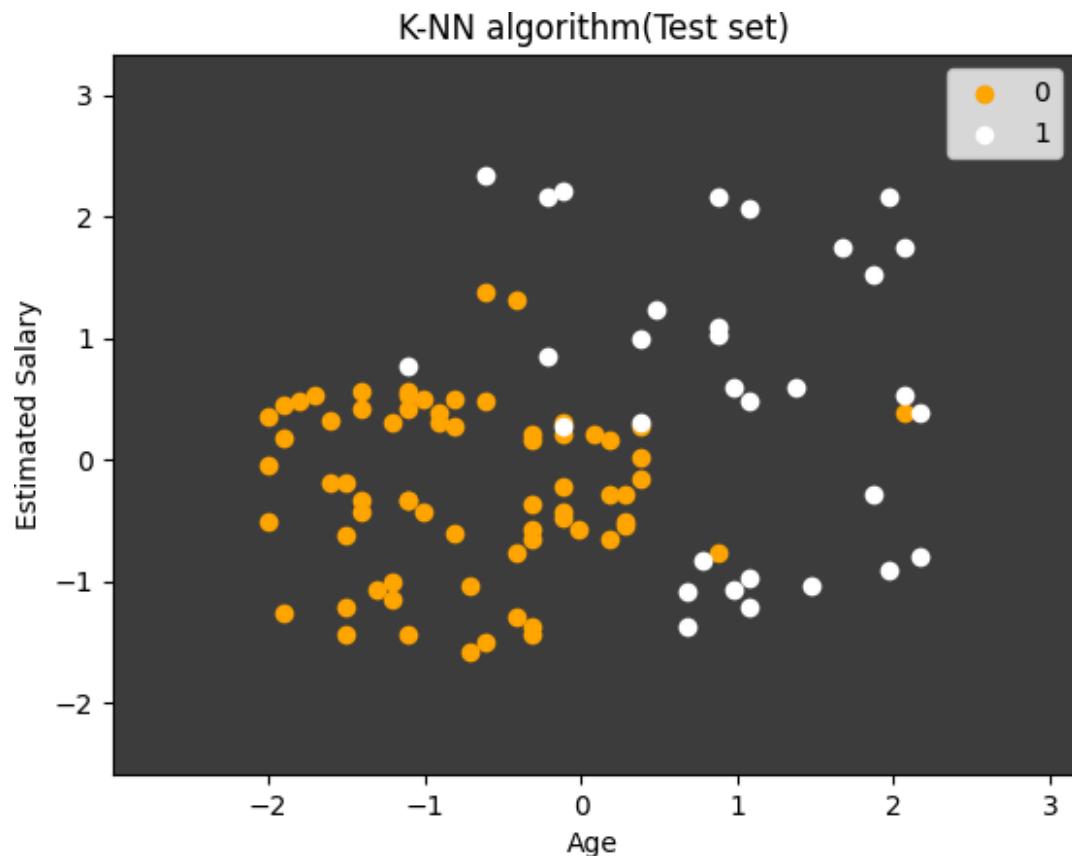
```

from matplotlib.colors import
ListedColormap x_set, y_set = x_test,
y_test
x1, x2 = Sidharth_np.meshgrid(Sidharth_np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step=0.01),
Sidharth_np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
Sidharth_plt.contourf(x1, x2,
classifier.predict(Sidharth_np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1. shape), alpha = 0.75, cmap =
ListedColormap(('black', 'black')))
Sidharth_plt.xlim(x1.min(), x1.max())
Sidharth_plt.ylim(x2.min(), x2.max())
for i, j in enumerate(Sidharth_np.unique(y_set)):
    Sidharth_plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c
= ListedColormap(('orange','white'))(i), label
= j) Sidharth_plt.title('K-NN algorithm(Test
set') Sidharth_plt.xlabel('Age')
Sidharth_plt.ylabel('Estimated Salary')
Sidharth_plt.legend()
Sidharth_plt.show()

```

```
<ipython-input-14-0e320db7891f>:8: UserWarning: *c* argument  
looks like a single numeric RGB or RGBA sequence, which should be  
avoided as value-mapping will have precedence in case its length  
matches with *x* & *y*. Please use the *color* keyword-argument  
or provide a 2D array with a single row if you intend to specify  
the same RGB or RGBA value for all points.
```

```
Sidharth_plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],  
c = ListedColormap(('orange','white'))(i), label = j)
```



I have successfully implemented KNN on myData set.

<b>Name of Student:</b> Pushkar Prasad Sane					
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> 5			
<b>Title of Lab Assignment:</b>					
Implementation of Decision Tree Using ID3,C4.5,Gini And Naïve Bayes Classifier					
<b>DOP:</b> 9/2/24		<b>DOS:</b> 1 6 / 2 / 2 4			
CO:  CO2, CO4	PO:  PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO8, PO9, PO11, PSO1, PSO2	<b>Faculty signature:</b>			

### Aim: Implementation of Decision Tree Using ID3,C4.5,Gini And Naïve Bayes Classifier

Decision Tree:

In simple words, a decision tree is a structure that contains nodes (rectangular boxes) and edges(arrows) and is built from a dataset (table of columns representing features/attributes and rows corresponding to records). Each node is either used to make a decision (known as decision node) or represent an outcome (known as leaf node).

ID3:

ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step. Invented by Ross Quinlan, ID3 uses a top-down greedy approach to build a decision tree. In simple words, the top-down approach means that we start building the tree from the top and the greedy approach means that at each iteration we select the best feature at the present moment to create a node. Most generally ID3 is only used for classification problems with nominal features only.

C4.5:

C4.5 is a decision tree algorithm that was developed by Ross Quinlan. It is an extension of the ID3 (Iterative Dichotomiser 3) algorithm and is commonly used for classification tasks. C4.5 incorporates several enhancements over ID3, including the ability to handle continuous and discrete attribute types, handling missing attribute values, and pruning to reduce overfitting.

Gini:

In the context of a decision tree, the Gini index (also known as Gini impurity or Gini coefficient) is a measure of impurity or the degree of uniformity in a set of data. It is commonly used as a criterion to evaluate the quality of a split at each node in a decision tree algorithm.

The Gini index is calculated based on the probability of misclassifying a randomly chosen element from the dataset if it were randomly labeled according to the class distribution in the node. In other words, it measures the probability of two randomly chosen elements being incorrectly labeled if they are randomly assigned a class label according to the distribution in the node.

Naive Bayes: Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other. Bayes' Theorem finds the probability of an event occurring given the probability of

another event that has already occurred. Bayes' theorem is stated mathematically as the

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

following equation:

where A and B are events and  $P(B) \neq 0$ . • Basically, we are trying to find the probability of event A, given the event B is true. Event B is also termed as evidence. •  $P(A)$  is the priori of A (the prior probability, i.e., Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B). •  $P(A|B)$  is a posteriori probability of B, i.e., probability of event after evidence is seen.

```
import numpy as np #linear algebra
import pandas as pd #data processing
from sklearn.preprocessing import LabelEncoder #if data isstring
then use it to preprocess the data
from sklearn.tree import DecisionTreeClassifier

Sidharth_df = pd.read_csv('/content/glass.csv') Sidharth_df

      RI      Na      Mg      Al      Si      K      Ca      Ba      Fe  Type
0    1.52101  13.64  4.49  1.10  71.78  0.06  8.75  0.00  0.0       1
1    1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.00  0.0       1
2    1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.00  0.0       1
3    1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.00  0.0       1
4    1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.00  0.0       1
...
209   1.51623  14.14  0.00  2.88  72.61  0.08  9.18  1.06  0.0       7
210   1.51685  14.92  0.00  1.99  73.06  0.00  8.40  1.59  0.0       7
211   1.52065  14.36  0.00  2.02  73.42  0.00  8.44  1.64  0.0       7
212   1.51651  14.38  0.00  1.94  73.61  0.00  8.48  1.57  0.0       7
213   1.51711  14.23  0.00  2.08  73.36  0.00  8.62  1.67  0.0       7

[214 rows x 10 columns]

#Extracting Features
X = Sidharth_df.iloc[:, :-1] y
= Sidharth_df.iloc[:, 9]

#making Model
Sidharth_dt = DecisionTreeClassifier(criterion = "entropy")

Sidharth_dt = DecisionTreeClassifier(criterion='entropy')

#Splitting the dataset into training, testing and predicting values:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
= 0.3, random_state = 0)
```

```

Sidharth_dt.fit(X_train, y_train)
y_pred =
Sidharth_dt.predict(X_test)
y_pred

1, 1, 6, 2, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 7, 3, 2, 3, 2, 5,
7, 1, 1, 7, 1, 1, 2, 1, 3, 2, 2, 1, 1, 3, 1, 7, 2, 6, 2,
1, 1, 2, 1, 2, 1, 6, 7, 1, 1, 2, 1, 2, 1, 3, 1, 1, 1, 7, 1])

array([
7, 1,
      7,
1,
      2,

#Making Confusion Matrix
from sklearn.metrics import confusion_matrix
Sidharth_cm = confusion_matrix(y_test, y_pred)
print(Sidharth_cm)

[[16  4  1   0   0  0]
 [10 13  1   0   1  1]
 [ 2  2  3   0   0  0]
 [ 0  1  0   1   0  0]
 [ 0  0  0   0   2  0]
 [ 0  0  0   0   0  7]]]

#finding accuracy from the confusion matrix.
a = Sidharth_cm.shape
corrPred = 0
falsePred = 0
for row in range(a[0]):
    for c in range(a[1]):
        if row == c:
            corrPred += Sidharth_cm[row,c]
        else:
            falsePred += Sidharth_cm[row,c]
print('Correct predictions:', corrPred)
print('False predictions', falsePred)
print ('\n\nAccuracy of the ID3 is: ', corrPred/(Sidharth_cm.sum()))

Correct predictions: 42
False predictions 23

```

Accuracy of the ID3 is: 0.6461538461538462

#### B] C4.5:

```

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text

```

Reading the dataset:

```
Sidharth_iris = load_iris()
```

Splitting dataset into training, testing and predicting values:

```
decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
decision_tree = decision_tree.fit(Sidharth_iris.data,
Sidharth_iris.target)
```

Predicting values:

```
r = export_text(decision_tree,
feature_names=Sidharth_iris['feature_names'])
print(r)

|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) > 0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |   |--- petal width (cm) > 1.75
|   |   |--- class: 2
```

### C. Naive Bayes

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.datasets import load_iris
Sidharth_iris = load_iris()

store the feature matrix (X) and response vector (y)
X = Sidharth_iris.data y
= Sidharth_iris.target

Sidharth_dataset = pd.read_csv('/content/golf.csv')

Sidharth_dataset.head()

    Outlook Temp Humidity Windy Play Golf
0      Rainy   Hot     High  False      No
1      Rainy   Hot     High   True      No
2  Overcast   Hot     High  False     Yes
3     Sunny  Mild     High  False     Yes
4     Sunny  Cool    Normal  False     Yes
```

Splitting the data:

```
# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.4, random_state=1)
```

Preparing the Data:

```

from sklearn.preprocessing import
StandardScaler sc = StandardScaler()
X_train =
sc.fit_transform(X_train)
X_test = sc.transform(X_test)

```

Fitting Naïve Bayes Classification:

```

# training the model on training set
from sklearn.naive_bayes import
GaussianNB gnb = GaussianNB()
gnb.fit(X_train,
y_train) GaussianNB()

```

Predicting Data Set:

```

# making predictions on the testing
set y_pred = gnb.predict(X_test)
print(y_pred)

[0 1 1 0 2 2 2 0 0 2 1 0 2 1 1 0 1 1 0 0 1 1 2 0 2 1 0 0 1 2 1 2 1 2
2
0 1
0 1 2 2 0 1 2 1 2 0 0 0 1 0 0 2 2 2 2 1 2 1]

```

Compare The data:

```

# comparing actual response values (y_test) with predicted response
values (y_pred)
y_compare = np.vstack((y_test,y_pred)).T
y_compare[:5,:]

array([[0, 0],
       [1, 1],
       [1, 1],
       [0, 0],
       [2, 2]])

```

10.Making Confusion matrix:

```

# Making the Confusion Matrix
from sklearn.metrics import
confusion_matrix cm =
confusion_matrix(y_test, y_pred)
print(cm)

[[19  0  0]
 [ 0 19  2]
 [ 0  1 19]]

#finding accuracy from the confusion matrix.
a = cm.shape
corrPred = 0
falsePred = 0
for row in range(a[0]):

```

```
for c in range(a[1]):  
    if row == c:  
        corrPred += cm[row, c]  
    else:  
        falsePred += cm[row, c]  
print('Correct predictions: ',  
corrPred) print('False predictions',  
falsePred)  
print ('\n\nAccuracy of the Naive Bayes Clasification is: ',  
corrPred/(cm.sum()))  
  
Correct predictions: 57  
False predictions 3
```

Accuracy of the Naive Bayes Clasification is: 0.95

#### CONCLUSION:

I have understood and implemented decision tree using ID3, C4.5 and Naïve Bayes Classification in this practical.

<b>Name of Student:</b> Pushkar Prasad Sane					
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> 6			
<b>Title of Lab Assignment:</b>					
<b>Implementation and analysis of clustering algorithms like K-Means, and K-medoid.</b>					
<b>DOP:</b> 16/2/24		<b>DOS:</b> 23/2/24			
<b>CO:</b>  CO2, CO4	<b>PO:</b>  PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO8, PO9, PO11, PO12, PSO1, PSO2	<b>Faculty Signature:</b>			

**Aim: Implementation and analysis of clustering algorithms like K-Means ,K-medoid.**

### **K-Means Clustering:**

K-means clustering is an unsupervised machine learning algorithm used for partitioning a dataset into K distinct clusters based on their similarity.

The algorithm begins by randomly initializing K cluster centroids in the feature space. Each centroid represents the mean value of the data points assigned to that cluster.

It assigns each data point to the nearest centroid based on the Euclidean distance between the data point and the centroids. The data points are then grouped together based on their assigned centroids.

After the initial assignment, the algorithm recalculates the centroids by taking the mean of all the data points within each cluster. This step aims to update the centroids' positions to better represent the data points within each cluster.

The algorithm iteratively repeats the assignment and centroid update steps until convergence is achieved. Convergence occurs when the centroids no longer move significantly or when a maximum number of iterations is reached.

K-means clustering aims to minimize the within-cluster sum of squares (WCSS) or the total squared distance between each data point and its assigned centroid. It seeks to create compact and well-separated clusters.

The final result of the K-means clustering algorithm is a set of K cluster centroids and a partitioning of the data points into K clusters. These clusters can be used for various purposes, such as grouping similar data points, identifying outliers, or initializing other machine learning algorithms. However, it is important to note that the algorithm's outcome can vary depending on the initial centroid placement and is sensitive to the choice of the number of clusters, K.

### **K-Medoids:**

K-medoid clustering, also known as Partitioning Around Medoids (PAM), is a variation of K-means clustering that uses representative points called medoids instead of centroids.

The algorithm starts by randomly selecting K data points from the dataset as initial medoids. These initial medoids can be any points in the feature space.

It assigns each data point to the nearest medoid based on a distance metric, typically the Euclidean distance. The data points are grouped together based on their assigned medoids.

After the initial assignment, the algorithm evaluates the total dissimilarity or cost of the clustering solution. The dissimilarity is usually defined as the sum of distances between each data point and its assigned medoid.

The algorithm then iteratively updates the medoids within each cluster to reduce the total dissimilarity. It considers swapping a medoid with a non-medoid data point and calculates the new total dissimilarity. The medoid swap is accepted if it leads to a decrease in the total dissimilarity.

The algorithm repeats the assignment and medoid update steps until convergence is achieved. Convergence occurs when no more medoid swaps result in a decrease in the total dissimilarity or when a maximum number of iterations is reached.

The final result of K-medoid clustering is a set of K medoids and a partitioning of the data points into K clusters. Unlike K-means, where the cluster centers can be any point in the feature space, K-medoid clusters are represented by actual data points. This makes K-medoid clustering more robust to outliers or noise in the dataset.

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

# Importing the dataset
Sidharth_dataset = pd.read_csv('/content/Mall_Customers.csv')
#dataset
Sidharth_dataset
```

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19		15
39	2	Male	21		15
81	3	Female	20		16
6	4	Female	23		16
77	5	Female	31		17
40	...	...	...	...	...
..	196	Female	35		120
79	197	Female	45		126
28	198	Male	32		126
74	199	Male	32		137
18	200	Male	30		137
83					

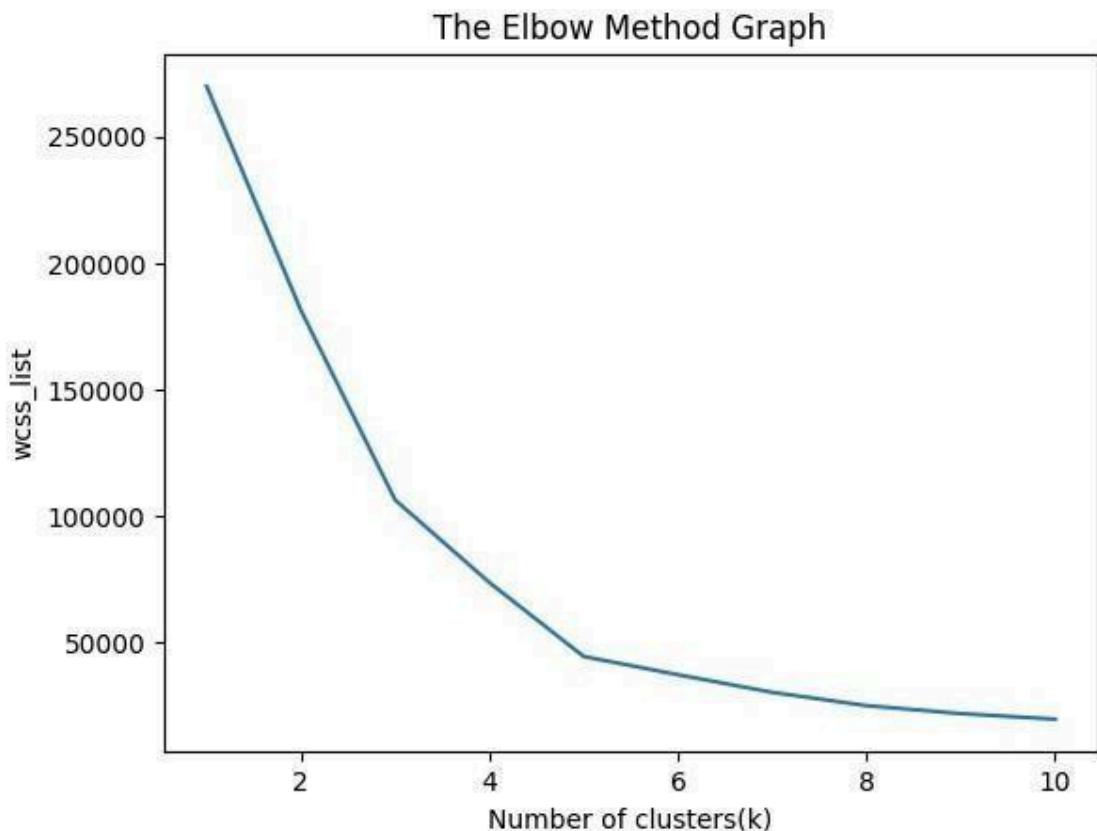
[200 rows x 5 columns]



```

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:8
70
: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
    warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:8
70
: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
    warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:8
70
: FutureWarning: The default value of `n_init` will change from 10
to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
the warning
    warnings.warn(

```



### TRAINING K-MEANS MODEL ON A DATASET:

```

kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
y_predict= kmeans.fit_predict(x)
#visualizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100,
c = 'blue', label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100,
c = 'green', label = 'Cluster 2') #for second

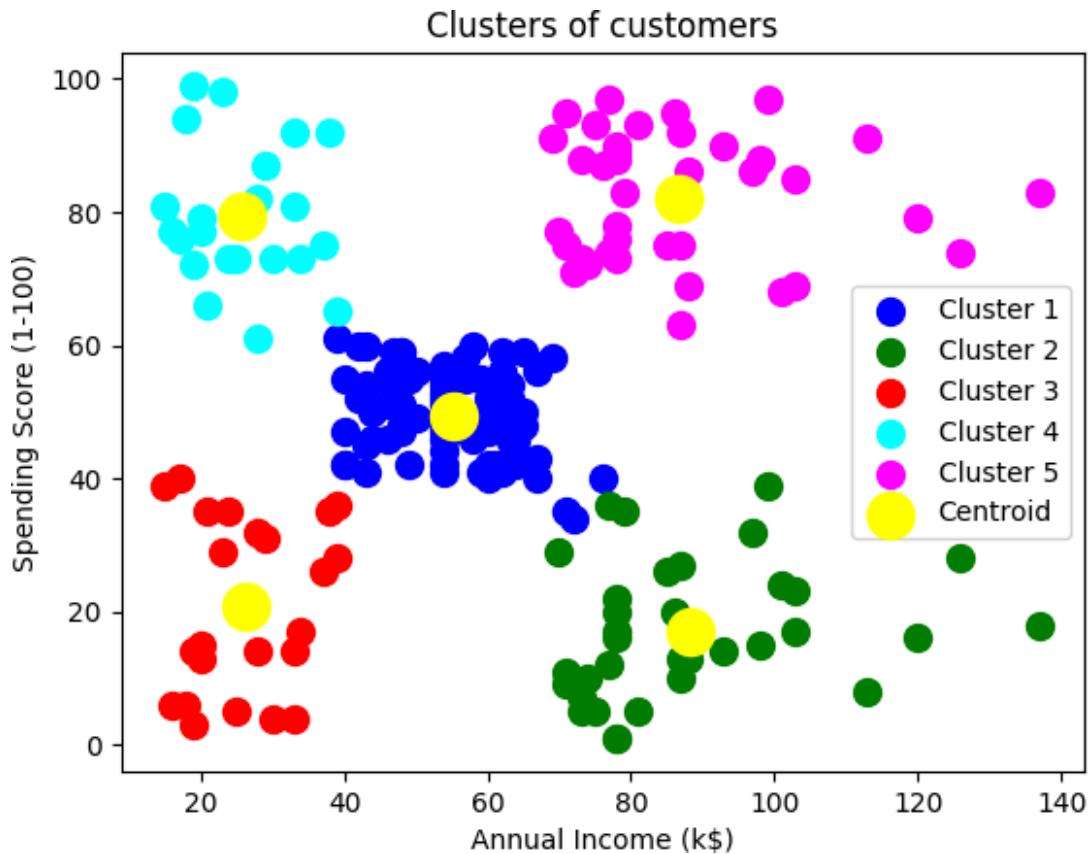
```

```

mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100,
c = 'red', label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100,
c = 'cyan', label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100,
c = 'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/
_kmeans.py:870: FutureWarning: The default value of `n_init` will
change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
warnings.warn(

```



```

import numpy as np #linear
algebra import matplotlib.pyplot
as plt import pandas as pd #data
processing from sklearn.cluster
import KMeans

```

```
from sklearn_extra.cluster import KMedoids
from sklearn import datasets

#loading Dataset
myiris = datasets.load_iris()
x = myiris.data #we get independent
variables y = myiris.target #we get target
variables myiris

{'data': array([[5.1, 3.5, 1.4, 0.2],
   [4.9, 3. , 1.4, 0.2],
   [4.7, 3.2, 1.3, 0.2],
   [4.6, 3.1, 1.5, 0.2],
   [5. , 3.6, 1.4, 0.2],
   [5.4, 3.9, 1.7, 0.4],
   [4.6, 3.4, 1.4, 0.3],
   [5. , 3.4, 1.5, 0.2],
   [4.4, 2.9, 1.4, 0.2],
   [4.9, 3.1, 1.5, 0.1],
   [5.4, 3.7, 1.5, 0.2],
   [4.8, 3.4, 1.6, 0.2],
   [4.8, 3. , 1.4, 0.1],
   [4.3, 3. , 1.1, 0.1],
   [5.8, 4. , 1.2, 0.2],
   [5.7, 4.4, 1.5, 0.4],
   [5.4, 3.9, 1.3, 0.4],
   [5.1, 3.5, 1.4, 0.3],
   [5.7, 3.8, 1.7, 0.3],
   [5.1, 3.8, 1.5, 0.3],
   [5.4, 3.4, 1.7, 0.2],
   [5.1, 3.7, 1.5, 0.4],
   [4.6, 3.6, 1. , 0.2],
   [5.1, 3.3, 1.7, 0.5],
   [4.8, 3.4, 1.9, 0.2],
   [5. , 3. , 1.6, 0.2],
   [5. , 3.4, 1.6, 0.4],
   [5.2, 3.5, 1.5, 0.2],
   [5.2, 3.4, 1.4, 0.2],
   [4.7, 3.2, 1.6, 0.2],
   [4.8, 3.1, 1.6, 0.2],
   [5.4, 3.4, 1.5, 0.4],
   [5.2, 4.1, 1.5, 0.1],
   [5.5, 4.2, 1.4, 0.2],
   [4.9, 3.1, 1.5, 0.2],
   [5. , 3.2, 1.2, 0.2],
   [5.5, 3.5, 1.3, 0.2],
   [4.9, 3.6, 1.4, 0.1],
   [4.4, 3. , 1.3, 0.2],
   [5.1, 3.4, 1.5, 0.2],
   [5. , 3.5, 1.3, 0.3],
   [4.5, 2.3, 1.3, 0.3],
```

[4.4, 3.2, 1.3, 0.2],  
[5., 3.5, 1.6, 0.6],  
[5.1, 3.8, 1.9, 0.4],  
[4.8, 3., 1.4, 0.3],  
[5.1, 3.8, 1.6, 0.2],  
[4.6, 3.2, 1.4, 0.2],  
[5.3, 3.7, 1.5, 0.2],  
[5., 3.3, 1.4, 0.2],  
[7., 3.2, 4.7, 1.4],  
[6.4, 3.2, 4.5, 1.5],  
[6.9, 3.1, 4.9, 1.5],  
[5.5, 2.3, 4., 1.3],  
[6.5, 2.8, 4.6, 1.5],  
[5.7, 2.8, 4.5, 1.3],  
[6.3, 3.3, 4.7, 1.6],  
[4.9, 2.4, 3.3, 1. ],  
[6.6, 2.9, 4.6, 1.3],  
[5.2, 2.7, 3.9, 1.4],  
[5., 2., 3.5, 1. ],  
[5.9, 3., 4.2, 1.5],  
[6., 2.2, 4., 1. ],  
[6.1, 2.9, 4.7, 1.4],  
[5.6, 2.9, 3.6, 1.3],  
[6.7, 3.1, 4.4, 1.4],  
[5.6, 3., 4.5, 1.5],  
[5.8, 2.7, 4.1, 1. ],  
[6.2, 2.2, 4.5, 1.5],  
[5.6, 2.5, 3.9, 1.1],  
[5.9, 3.2, 4.8, 1.8],  
[6.1, 2.8, 4., 1.3],  
[6.3, 2.5, 4.9, 1.5],  
[6.1, 2.8, 4.7, 1.2],  
[6.4, 2.9, 4.3, 1.3],  
[6.6, 3., 4.4, 1.4],  
[6.8, 2.8, 4.8, 1.4],  
[6.7, 3., 5., 1.7],  
[6., 2.9, 4.5, 1.5],  
[5.7, 2.6, 3.5, 1. ],  
[5.5, 2.4, 3.8, 1.1],  
[5.5, 2.4, 3.7, 1. ],  
[5.8, 2.7, 3.9, 1.2],  
[6., 2.7, 5.1, 1.6],  
[5.4, 3., 4.5, 1.5],  
[6., 3.4, 4.5, 1.6],  
[6.7, 3.1, 4.7, 1.5],  
[6.3, 2.3, 4.4, 1.3],  
[5.6, 3., 4.1, 1.3],  
[5.5, 2.5, 4., 1.3],  
[5.5, 2.6, 4.4, 1.2],  
[6.1, 3., 4.6, 1.4],

[5.8, 2.6, 4., 1.2],  
[5., 2.3, 3.3, 1. ],  
[5.6, 2.7, 4.2, 1.3],  
[5.7, 3., 4.2, 1.2],  
[5.7, 2.9, 4.2, 1.3],  
[6.2, 2.9, 4.3, 1.3],  
[5.1, 2.5, 3., 1.1],  
[5.7, 2.8, 4.1, 1.3],  
[6.3, 3.3, 6., 2.5],  
[5.8, 2.7, 5.1, 1.9],  
[7.1, 3., 5.9, 2.1],  
[6.3, 2.9, 5.6, 1.8],  
[6.5, 3., 5.8, 2.2],  
[7.6, 3., 6.6, 2.1],  
[4.9, 2.5, 4.5, 1.7],  
[7.3, 2.9, 6.3, 1.8],  
[6.7, 2.5, 5.8, 1.8],  
[7.2, 3.6, 6.1, 2.5],  
[6.5, 3.2, 5.1, 2. ],  
[6.4, 2.7, 5.3, 1.9],  
[6.8, 3., 5.5, 2.1],  
[5.7, 2.5, 5., 2. ],  
[5.8, 2.8, 5.1, 2.4],  
[6.4, 3.2, 5.3, 2.3],  
[6.5, 3., 5.5, 1.8],  
[7.7, 3.8, 6.7, 2.2],  
[7.7, 2.6, 6.9, 2.3],  
[6., 2.2, 5., 1.5],  
[6.9, 3.2, 5.7, 2.3],  
[5.6, 2.8, 4.9, 2. ],  
[7.7, 2.8, 6.7, 2. ],  
[6.3, 2.7, 4.9, 1.8],  
[6.7, 3.3, 5.7, 2.1],  
[7.2, 3.2, 6., 1.8],  
[6.2, 2.8, 4.8, 1.8],  
[6.1, 3., 4.9, 1.8],  
[6.4, 2.8, 5.6, 2.1],  
[7.2, 3., 5.8, 1.6],  
[7.4, 2.8, 6.1, 1.9],  
[7.9, 3.8, 6.4, 2. ],  
[6.4, 2.8, 5.6, 2.2],  
[6.3, 2.8, 5.1, 1.5],  
[6.1, 2.6, 5.6, 1.4],  
[7.7, 3., 6.1, 2.3],  
[6.3, 3.4, 5.6, 2.4],  
[6.4, 3.1, 5.5, 1.8],  
[6., 3., 4.8, 1.8],  
[6.9, 3.1, 5.4, 2.1],  
[6.7, 3.1, 5.6, 2.4],  
[6.9, 3.1, 5.1, 2.3],

[5.8, 2.7, 5.1, 1.9],  
[6.8, 3.2, 5.9, 2.3],



perhaps the best known database to be found in the\ npattern  
recognition literature. Fisher\'s paper is a classic in the field  
and\nis referenced frequently to this day. (See Duda & Hart, for  
example.) The\ndata set contains 3 classes of 50 instances each,

where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.\n\n.. topic::  
References\n\nFisher,  
R.A. "The use of multiple measurements in taxonomic problems"\nAnnual Eugenics, 7, Part II, 179-188 (1936); also in  
"Contributions to\nMathematical Statistics" (John Wiley, NY, 1950). -  
Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.\n - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n Structure and Classification Rule for Recognition in Partially Exposed\n Environments". IEEE Transactions on Pattern Analysis and Machine\n Intelligence, Vol. PAMI-2, No. 1, 67-71.\n - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions\n on Information Theory, May 1972, 431-433.\n - See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II\n conceptual clustering system finds 3 classes in the data.\n - Many, many more ...',  
'feature\_names': ['sepal length (cm)',  
'sepal width (cm)',  
'petal length  
(cm)', 'petal  
width (cm)'],  
'filename': 'iris.csv',  
'data\_module': 'sklearn.datasets.data'}

*#Scaling and Fitting KMedoids:*

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(x)
x_scaled = scaler.transform(x)
kMedoids = KMedoids(n_clusters = 3, random_state = 0)
kMedoids.fit(x_scaled)
y_kmed = kMedoids.fit_predict(x_scaled)
#to find out the cluster labels corresponding to different observations.
```

Silhouette Method to evaluate cluster:

```
from sklearn.metrics import silhouette_samples, silhouette_score
kMedoids = KMedoids(n_clusters = 3, random_state = 0)
kMedoids.fit(x_scaled)
y_kmed = kMedoids.fit_predict(x_scaled)
silhouette_avg = silhouette_score(x_scaled,
y_kmed)
print(silhouette_avg) # we get average value of all clusters
```

0.4590416105554613

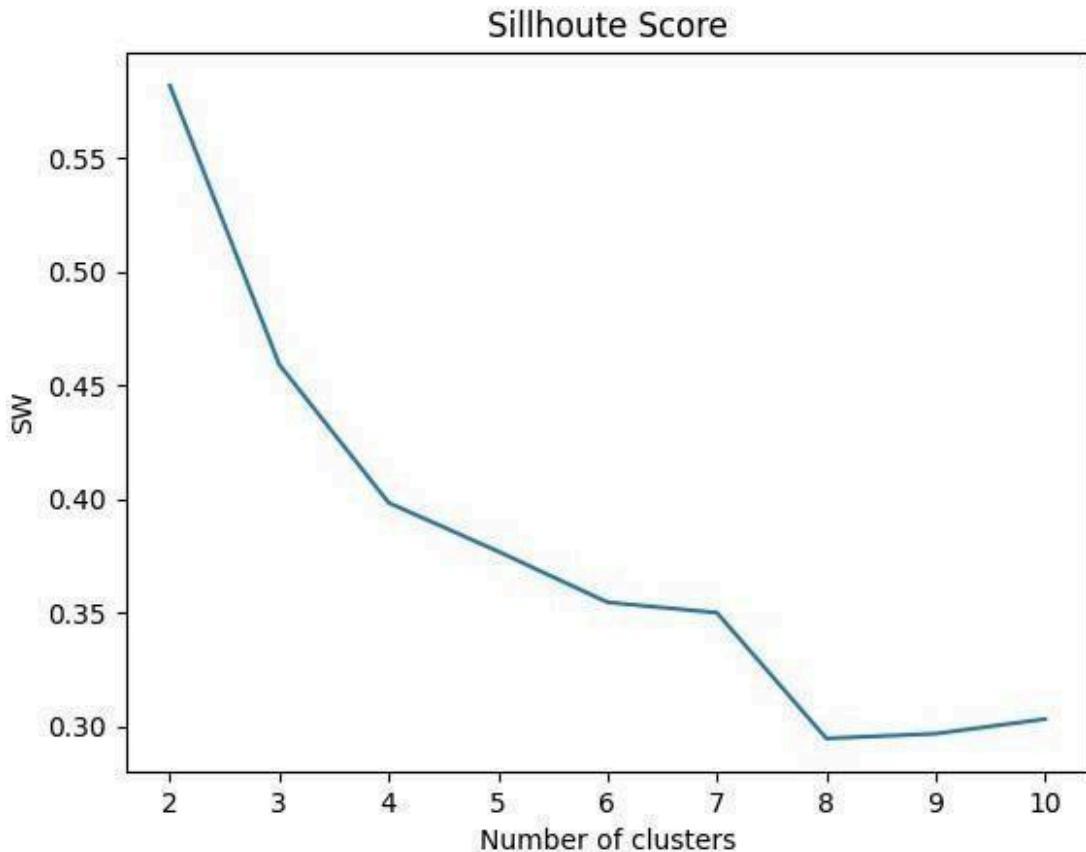
Silhouette Width to find number of cluster:

```
sw = []
for i in range(2, 11):
    kMedoids = KMedoids(n_clusters = i, random_state = 0)
    kMedoids.fit(x_scaled)
```

```

y_kmed = kMedoids.fit_predict(x_scaled)
silhouette_avg = silhouette_score(x_scaled,
y_kmed) sw.append(silhouette_avg)
#different silhouette is calculated for r different clusters
plt.plot(range(2, 11), sw)
plt.title('Silhouette Score')
plt.xlabel('Number of clusters')
plt.ylabel('SW') #within cluster sum of
squares plt.show()

```



Computing Purity:

```

from sklearn import metrics
def purity_score(y_true, y_pred):
    # compute contingency matrix (also called confusion matrix)
    contingency_matrix = metrics.cluster.contingency_matrix(y_true,
y_pred)
    # return purity
    return np.sum(np.amax(contingency_matrix, axis=0)) /
np.sum(contingency_matrix)

```

How extreme values effect K-Medoid compared to K-means:

```

kmeans = KMeans(n_clusters = 3, init = 'random', max_iter =
300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x_scaled)
purity_score(y,y_kmeans)

0.8333333333333334

```

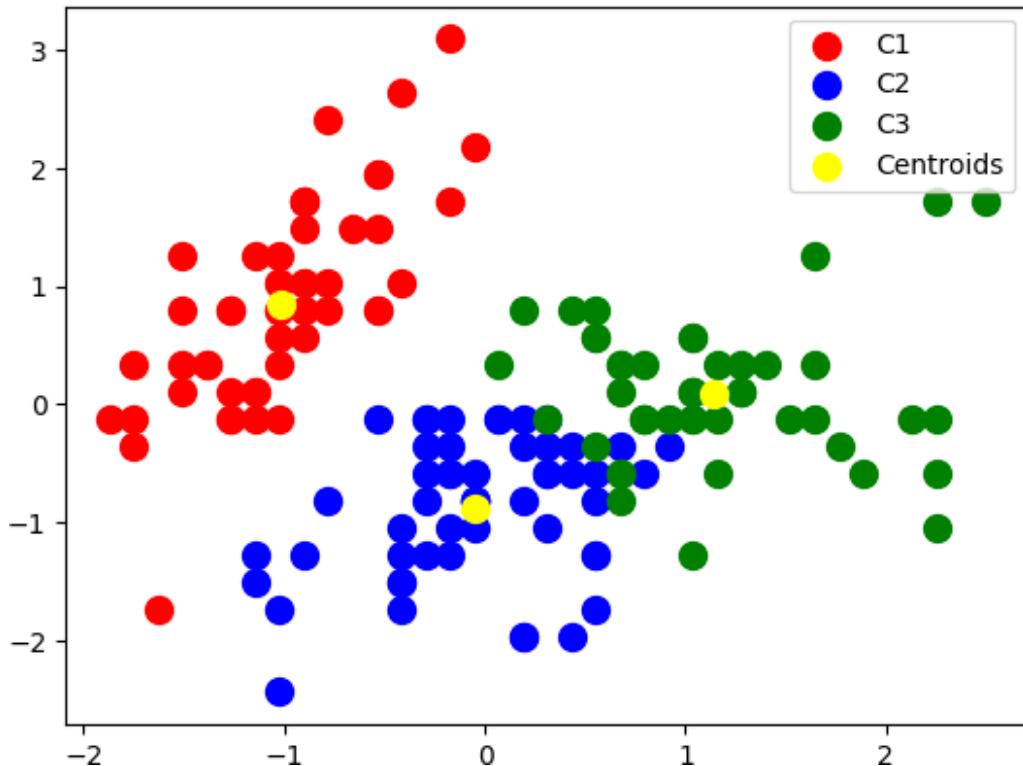
Plotting values:

```

plt.scatter(x_scaled[y_kmeans == 0, 0], x_scaled[y_kmeans == 0, 1], s
== 100, c = 'red', label =
'C1')
plt.scatter(x_scaled[y_kmeans == 1, 0], x_scaled[y_kmeans == 1, 1], s
== 100, c = 'blue', label =
'C2')
plt.scatter(x_scaled[y_kmeans == 2, 0], x_scaled[y_kmeans == 2, 1], s
== 100, c = 'green', label = 'C3')
plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:,1] , s = 100, c = 'yellow', label =
'Centroids')
plt.legend()

<matplotlib.legend.Legend at 0x7f2711d7a590>

```



Adding extreme values:

```

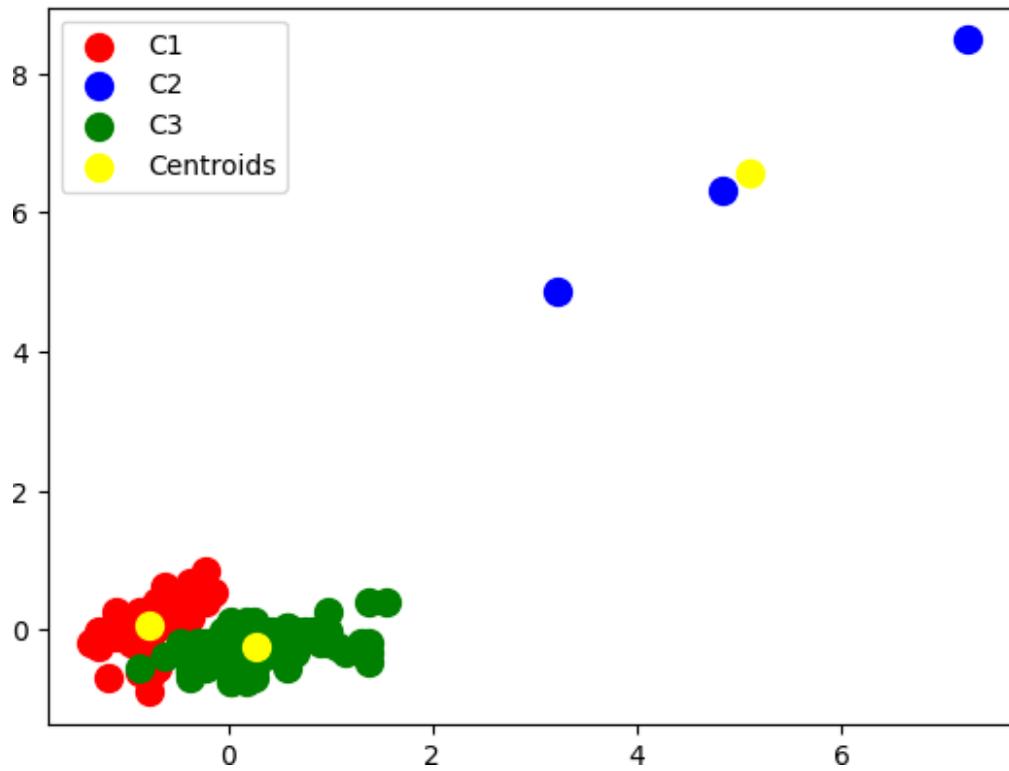
import numpy as np
m=np.append(x, [[10,10,10,10],[15,15,15,15],[12,12,12,12]],axis =0)
m.shape

```

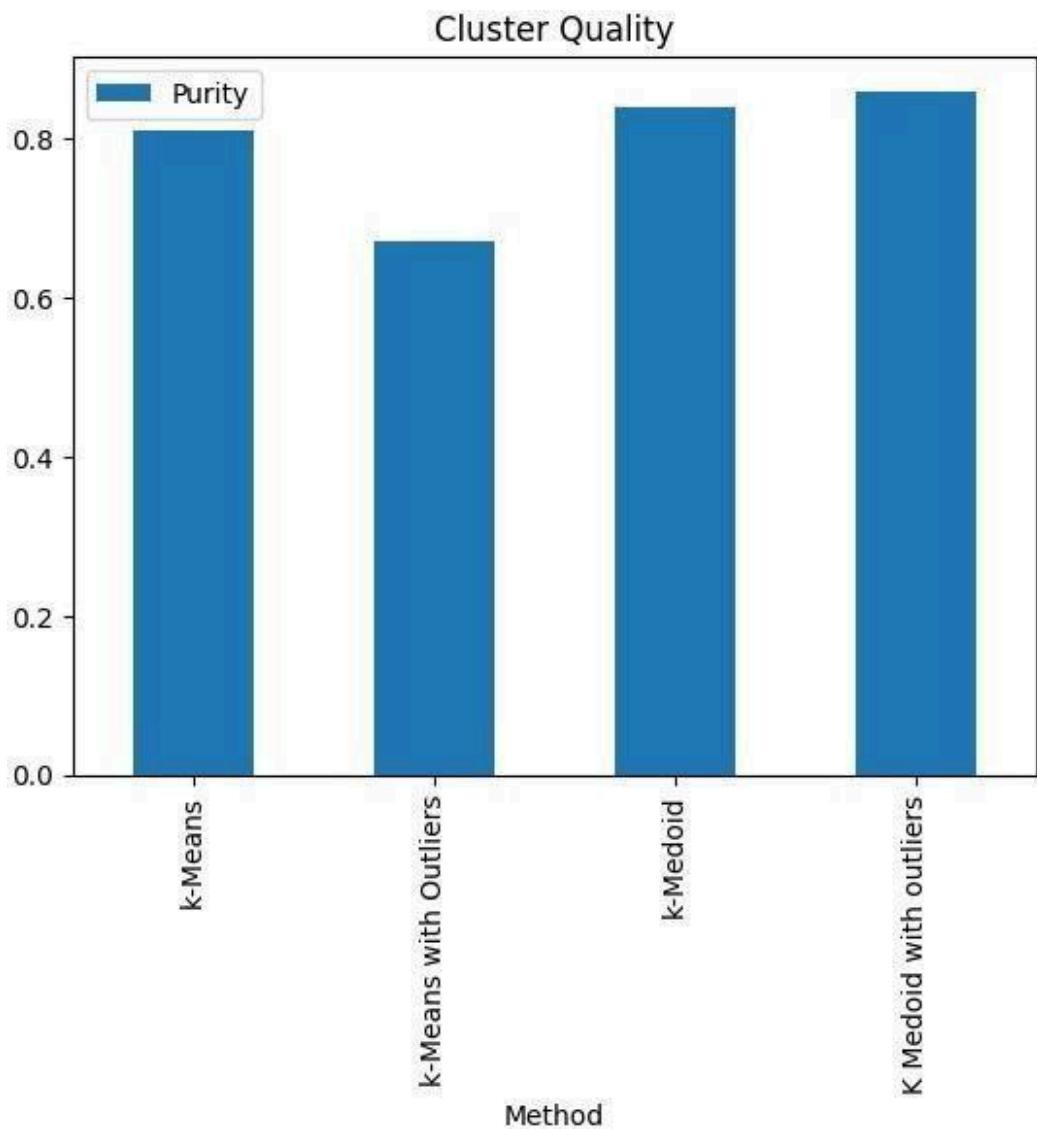
Plot:

```
plt.scatter(x_scaled[y_kmeans == 0, 0], x_scaled[y_kmeans == 0, 1],
           s = 100, c = 'red', label = 'C1')
plt.scatter(x_scaled[y_kmeans == 1, 0], x_scaled[y_kmeans == 1, 1],
           s = 100, c = 'blue', label = 'C2')
plt.scatter(x_scaled[y_kmeans == 2, 0], x_scaled[y_kmeans == 2, 1],
           s = 100, c = 'green', label = 'C3')
plt.scatter(kmeans.cluster_centers_[:, 0],
            kmeans.cluster_centers_[:, 1], s = 100, c = 'yellow', label =
            'Centroids')
#here 3 points formed cluster on their own.
plt.legend()

<matplotlib.legend.Legend at 0x7f2711d9eb30>
```



```
data = [['k-Means', 0.81], ['k-Means with Outliers', 0.67], ['k-Medoid', 0.84], ['K Medoid with outliers', 0.86]]  
df = pd.DataFrame(data, columns = ['Method', 'Purity'])  
df.plot.bar(x='Method', y='Purity', title='Cluster Quality')  
<Axes: title={'center': 'Cluster Quality'}, xlabel='Method'>
```



CONCLUSION:

- K-Medoids are also called PAM(Partition around Medoids)
- Mean is computed from dataset S median is chosen from dataset.
- Medoids are representative objects of a dataset or cluster with a dataset whose average dissimilarities to all the objects in the cluster is minimal.

<b>Name of Student:</b> Pushkar Prasad Sane					
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> 7			
<b>Title of Lab Assignment:</b>					
<b>Implementation of Classifying data using Linear Kernels in Support Vector Machines (SVMs).</b>					
<b>DOP:</b> 23/2/24		<b>DOS:</b> 01/3/24			
<b>CO:</b>  CO2, CO4	<b>PO:</b>  PO1, PO2, PO3, PO4, PO5, PO6, PO7, PO8, PO9, PO1, PO12, PSO1, PSO2	<b>Faculty signature:</b>			

Aim: Implementation of Classifying data using Linear Kernels in Support Vector Machines (SVMs).

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It is particularly effective in solving binary classification problems, but can be extended to handle multi-class classification as well.

SVM works by finding an optimal hyperplane in a high-dimensional feature space that separates the data points of different classes with the maximum margin. The hyperplane is chosen such that it maximizes the distance (margin) between the nearest data points of different classes, known as support vectors.

The algorithm uses a kernel function to map the input data into a higher-dimensional space, where it becomes easier to find a hyperplane that can linearly separate the classes. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

In addition to linear separation, SVM can also handle non-linear decision boundaries by using the kernel trick. The kernel trick allows SVM to implicitly map the data points into a higher-dimensional space without explicitly computing the transformation, saving computational resources.

During the training phase, SVM optimizes a cost function that aims to find the hyperplane with the maximum margin while minimizing the classification errors. This optimization problem is typically solved using convex optimization techniques.

Once the SVM model is trained, it can be used to predict the class labels of new, unseen data points by determining which side of the hyperplane they fall on. The decision boundary is based on a threshold value, and the predicted class is determined by comparing the signed distance of the data point to the hyperplane.

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

dataset = pd.read_csv('pulsar_data_train.csv')
dataset.head()

      Mean of the integrated profile \
0            121.156250
1            76.968750
2           130.585938
3           156.398438
4            84.804688

      Standard deviation of the integrated profile \
0             48.372971
```

```

1                         36.175557
2                         53.229534
3                         48.865942
4                         36.117659

    Excess kurtosis of the integrated profile \
0                         0.375485
1                         0.712898
2                         0.133408
3                         -0.215989
4                         0.825013

    Skewness of the integrated profile   Mean of the DM-SNR curve \
0                         -0.013165          3.168896
1                         3.388719          2.399666
2                         -0.297242          2.743311
3                         -0.171294          17.471572
4                         3.274125          2.790134

    Standard deviation of the DM-SNR curve \
0                         18.399367
1                         17.570997
2                         22.362553
3                         NaN
4                         20.618009

    Excess kurtosis of the DM-SNR curve   Skewness of the DM-SNR
curve
\
0                         7.449874
65.159298

1                         9.414652
102.722975

2                         8.508364
74.031324

3                         2.958066
7.197842

4                         8.405008
76.291128

    target_clas
    s
0                         0.0
1                         0.0
2                         0.0
3                         0.0
4                         0.0

dataset=dataset.dropna()
dataset.isnull().sum()

```

```
Mean of the integrated profile          0
Standard deviation of the integrated profile 0
Excess kurtosis of the integrated profile 0
Skewness of the integrated profile      0
Mean of the DM-SNR curve              0
Standard deviation of the DM-SNR curve 0
Excess kurtosis of the DM-SNR curve   0
Skewness of the DM-SNR curve         0
target_class                         0
dtype: int64

dataset.head()

()

    Mean of the integrated profile  \
0                  121.156250
1                  76.968750
2                 130.585938
4                 84.804688
7                 109.406250

    Standard deviation of the integrated profile \
0                   48.372971
1                   36.175557
2                   53.229534
4                   36.117659
7                   55.912521

    Excess kurtosis      of the integrated  \
0                      profile
                           0.375485
1                      0.712898
2                      0.133408
4                      0.825013
7                      0.565106

    Skewness of the integrated profile  Mean of the DM-SNR curve \
0                   -0.013165           3.168896
1                   3.388719           2.399666
2                   -0.297242          2.743311
4                   3.274125           2.790134
7                   0.056247          2.797659

    Standard deviation of the DM-SNR curve \
0                   18.399367
1                   17.570997
2                   22.362553
4                   20.618009
7                   19.496527

    Excess kurtosis of the DM-SNR curve  Skewness of the DM-SNR
curve
\
```

0	7.449874	65.159298
1	9.414652	102.722975
2	8.508364	74.031324
4	8.405008	76.291128
7	9.443282	97.374578

	target_class
0	0.0
1	0.0
2	0.0
4	0.0
7	0.0

```

x = dataset.iloc[:, [4,0]]
y = dataset.iloc[:, -1]

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)
x_train, x_test, y_train, y_test

```

	Mean of the DM-SNR curve	Mean of the integrated profile
12520	3.354515	115.375000
1391	3.549331	130.312500
9215	1.459866	126.789062
4638	3.006689	104.546875
11607	0.443980	118.968750
...	...	...
10669	1.689799	122.039062
12459	55.751672	92.078125
6544	3.829431	121.250000
4382	0.954013	119.929688
3661	2.868729	123.804688

[7418 rows x 2 columns],	Mean of the DM-SNR curve	Mean of the integrated profile
11372	3.692308	120.750000
3450	13.680602	126.273438
7024	33.632943	74.476562
1366	1.617893	139.500000
2444	2.770067	115.882812
...	...	...
2109	3.307692	109.750000
8389	2.653010	113.835938
3424	0.753344	127.781250

```

7933          20.774247          135.859375
5748          2.795151          90.320312

[1855 rows x 2 columns],
12520      0.0
1391       0.0
9215       0.0
4638       0.0
11607      0.0
...
10669      0.0
12459      0.0
6544       0.0
4382       0.0
3661       0.0
Name: target_class, Length: 7418, dtype: float64,
11372      0.0
3450       0.0
7024       1.0
1366       0.0
2444       0.0
...
2109       0.0
8389       0.0
3424       0.0
7933       0.0
5748       0.0
Name: target_class, Length: 1855, dtype: float64)

```

```

cols = x_train.columns
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

```

Fitting the SVM classifier to the training set:

```

from sklearn.svm import SVC
classifier = SVC(kernel="linear", random_state=0)
classifier.fit(x_train,y_train)

SVC(kernel='linear', random_state=0)

```

Predicting the test set result:

```

y_pred =
classifier.predict(x_test)
y_pred

array([0., 0., 0., ..., 0., 0., 0.])

```

Creating the confusion matrix:

```

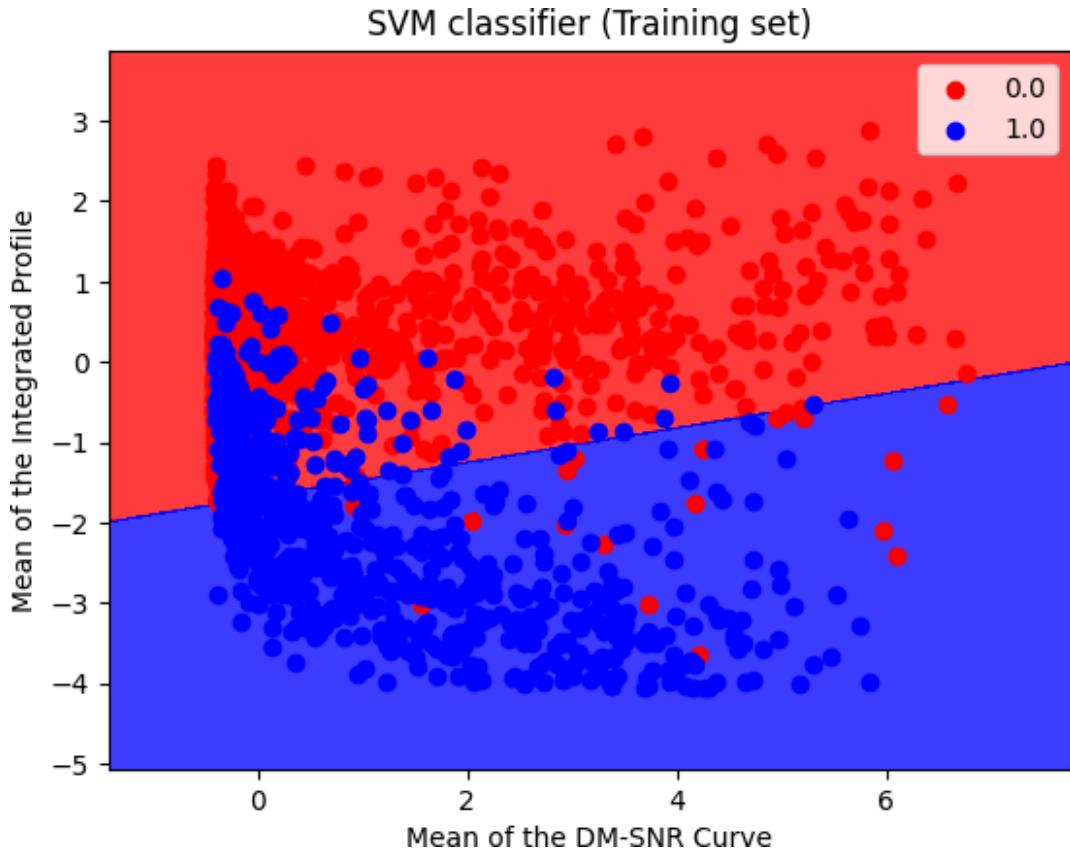
from sklearn.metrics import
confusion_matrix cm =
confusion_matrix(y_test,y_pred)
cm

array([[1676,     8],
       [ 63, 108]])

#train
from matplotlib.colors import
ListedColormap x_set, y_set = x_train,
y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1,
stop = x_set[:, 0].max() + 1, step =0.01), np.arange(start =
x_set[:, 1].min() - 1, stop = x_set[:, 1].max() +
1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(( 'red', 'blue')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
    ListedColormap(( 'red', 'blue'))(i), label =
j) plt.title('SVM classifier (Training set)')
plt.xlabel('Mean of the DM-SNR Curve')
plt.ylabel('Mean of the Integrated Profile')
plt.legend()
plt.show()

<ipython-input-11-aa5560be4a54>:11: UserWarning: *c* argument
looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length
matches with *x* & *y*. Please use the *color* keyword-argument
or provide a 2D array with a single row if you intend to specify
the same RGB or RGBA value for all points.
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
ListedColormap(( 'red', 'blue'))(i), label = j)

```

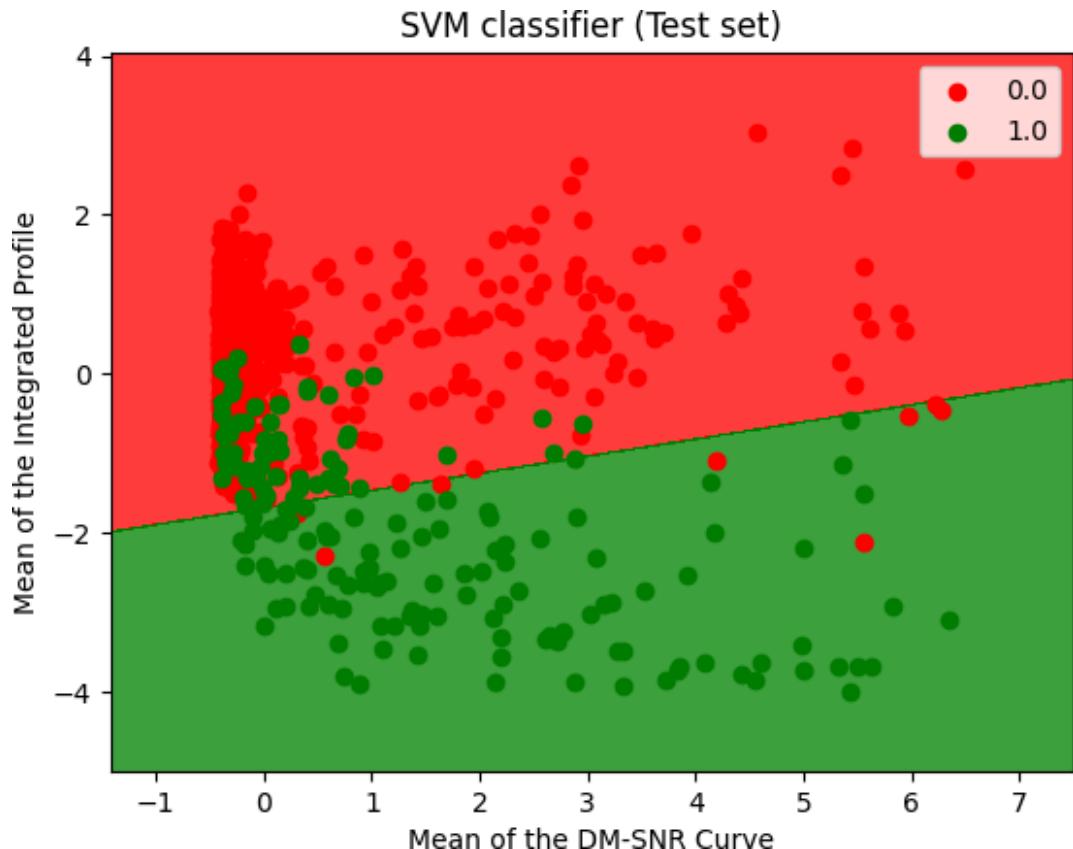


Visualizing the test set result:

```
#test
from matplotlib.colors import ListedColormap
x_set, y_set = x_test,
y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step = 0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
ListedColormap(('red', 'green'))(i), label =
j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Mean of the DM-SNR Curve')
plt.ylabel('Mean of the Integrated Profile')
plt.legend()
plt.show()
```

```
<ipython-input-12-4d685cf1a26f>:13: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

```
plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)
```



Conclusion:

I have learned the implementation of Classifying data using Linear Kernels i Support Vector Machines (SVMs).

<b>Name of Student:</b> Pushkar Prasad Sane					
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> fifi			
<b>Title of Lab Assignment:</b>					
<b>Implementation of Classifying data using Non-Linear Kernels in Support Vector Machines (SVMs).</b>					
<b>DOP:</b> 01/3/24		<b>DOS:</b> 0 8 / 3 / 2 4			
CO: CO2, CO4	PO: PO1, PO2, PO3, PO4 , PO5, PO6, PO7, PO 8, PO9, PO11, PO12 PSO1, PSO2	<b>Faculty signature:</b>			

Aim: Implementation of Classifying data using Non-Linear Kernels in Support Vector Machines (SVMs).

#### About Pulsar Star Dataset:

Pulsars are a rare type of Neutron star that produce radio emission detectable here on Earth. They are of considerable scientific interest as probes of space-time, the interstellar medium, and states of matter. Each candidate is described by 8 continuous variables, and a single class variable. The first four are simple statistics obtained from the integrated pulse profile (folded profile). This is an array of continuous variables that describe a longitude- resolved version of the signal that has been averaged in both time and frequency . The remaining four variables are similarly obtained from the DM-SNR curve. These are summarized below:

- Mean of the integrated profile.
- Standard deviation of the integrated profile.
- Excess kurtosis of the integrated profile.
- Skewness of the integrated profile.
- Mean of the DM-SNR curve.
- Standard deviation of the DM-SNR curve.
- Excess kurtosis of the DM-SNR curve.
- Skewness of the DM-SNR curve.
- Class

#### Support Vector Machine:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine.

#### Non - Linear SVM:

If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

- $z=x^2+y^2$

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_moons

dataset = pd.read_csv('/content/pulsar_data_train.csv')
dataset

Mean of the integrated profile \
0           121.156250
1            76.968750
```

2	130.585938
3	156.398438
4	84.804688
...	...
12523	124.312500
12524	115.617188
12525	116.031250
12526	135.664062
12527	120.726562

	Standard deviation of the integrated profile \
0	48.372971
1	36.175557
2	53.229534
3	48.865942
4	36.117659
...	..
1252	.
3	53.17905
3	3
12524	46.784600
12525	43.213846
12526	49.933749
12527	50.472256

	Excess kurtosis of the integrated profile \
0	0.375485
1	0.712898
2	0.133408
3	-0.215989
4	0.825013
...	..
1252	-0.012418
3	
12524	0.218177
12525	0.663456
12526	-0.089940
12527	0.346178

	Skewness of the integrated profile	Mean of the DM-SNR curve
\		
0	-0.013165	3.168896
1	3.388719	2.399666
2	-0.297242	2.743311
3	-0.171294	17.471572
4	3.274125	2.790134
...	..	..

12523	-0.556021	7.186455
12524	0.226757	6.140468
12525	0.433088	0.785117
12526	-0.226726	3.859532
12527	0.184797	0.769231

Standard deviation of the DM-SNR curve \

0	18.399367
1	17.570997
2	22.362553
3	NaN
4	20.618009
...	..
1252	.
3	29.30826
	6
12524	NaN
12525	11.628149
12526	21.501505
12527	11.792603

Excess kurtosis of the DM-SNR curve    Skewness of the DM-SNR  
curve \

0	7.449874
65.159298	
1	9.414652
102.722975	
2	8.508364
74.031324	
3	2.958066
7.197842	
4	8.405008
76.291128	
...	...
...	...
12523	4.531382
21.725143	
12524	5.732201
34.357283	
12525	17.055215
312.204325	
12526	7.398395
62.334018	
12527	17.662222
329.548016	

```
target_clas
s
0          0.0
1          0.0
2          0.0
3          0.0
4          0.0
...
12523      0.0
12524      0.0
12525      0.0
12526      0.0
12527      0.0

[12528 rows x 9 columns]

dataset=dataset.dropna()
dataset.isnull().sum()

Mean of the integrated profile          0
Standard deviation of the integrated profile 0
Excess kurtosis of the integrated profile   0
Skewness of the integrated profile         0
Mean of the DM-SNR curve                 0
Standard deviation of the DM-SNR curve    0
Excess kurtosis of the DM-SNR curve       0
Skewness of the DM-SNR curve             0
target_class                            0
dtype: int64

dataset.head

()

Mean of the integrated profile \
0           121.156250
1           76.968750
2          130.585938
4           84.804688
7           109.406250

Standard deviation of the integrated profile \
0            48.372971
1            36.175557
2            53.229534
4            36.117659
7            55.912521

Excess kurtosis of the integrated profile \
0            0.375485
1            0.712898
2            0.133408
4            0.825013
```

```

7                                0.565106

Skewness      of the integrated      Mean of the DM-SNR curve \
                profile
0              -0.013165                  3.168896
1              3.388719                  2.399666
2              -0.297242                  2.743311
4              3.274125                  2.790134
7              0.056247                  2.797659

Standard deviation of the DM-SNR curve \
0                      18.399367
1                      17.570997
2                      22.362553
4                      20.618009
7                      19.496527

Excess kurtosis of the DM-SNR curve   Skewness of the DM-SNR
curve
\
0                      7.449874
65.159298

1                      9.414652
102.722975

2                      8.508364
74.031324

4                      8.405008
76.291128

7                      9.443282
97.374578

target_clas
s
0          0.0
1          0.0
2          0.0
4          0.0
7          0.0

x = dataset.iloc[:,  

[2,4]] y =  

dataset.iloc[:, -1]

from sklearn.model_selection import train_test_split  

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size  

= 0.25, random_state = 0)  

x_train, x_test, y_train, y_test

(      Excess kurtosis of the integrated profile   Mean of the
DM- SNR curve
7731                               0.709628
1.566890
3091                               0.084360

```

2.567726

11452	0.257193
2.596990	
10631	-0.029067
1.220736	
5401	0.181712
2.511706	
...	...
...	...
10669	0.085125
1.689799	
12459	0.730128
55.751672	
6544	-0.183962
3.829431	
4382	0.277253
0.954013	
3661	0.018395
2.868729	

	[6954 rows x 2 columns],	Excess kurtosis of the integrated profile	Mean of the DM-
SNR			
curve		0.454148	
11372			
3.692308			
3450		0.031220	
13.680602			
7024		1.417642	
33.632943			
1366		-0.109619	
1.617893			
2444		0.126617	
2.770067			..
...			
...			.
2895		0.08977	

4

2.506689	
3945	0.095741
5.466555	
7197	1.065848
34.447324	
9204	0.361255
2.240803	
8307	0.403299
1.921405	

	[2319 rows x 2 columns],
7731	0.0
3091	0.0
11452	0.0

```
10631      0.0

5401      0.0
...
10669      0.0
12459      0.0
6544       0.0
4382       0.0
3661       0.0
Name: target_class,      Length: 6954, dtype:
:          0.0                  float64,
1137
2
3450      0.0
7024      1.0
1366      0.0
2444      0.0
...
2895      0.0
3945      0.0
7197      0.0
9204      0.0
8307      0.0
Name: target_class,      Length: 2319, dtype:
                           float64)
```

```
cols = x_train.columns
from sklearn.preprocessing import
StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Fitting the SVM classifier to the training set:

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf',
degree=8)
svclassifier.fit(x_train,y_train)
svclassifier
```

```
SVC(degree=8)
```

Predicting the test set result:

```
y_pred =
svclassifier.predict(x_test) y_pred
array([0., 0., 1., ..., 1., 0., 0.])
```

Creating the confusion matrix:

```
from sklearn.metrics import
confusion_matrix cm =
confusion_matrix(y_test,y_pred)
cm
```

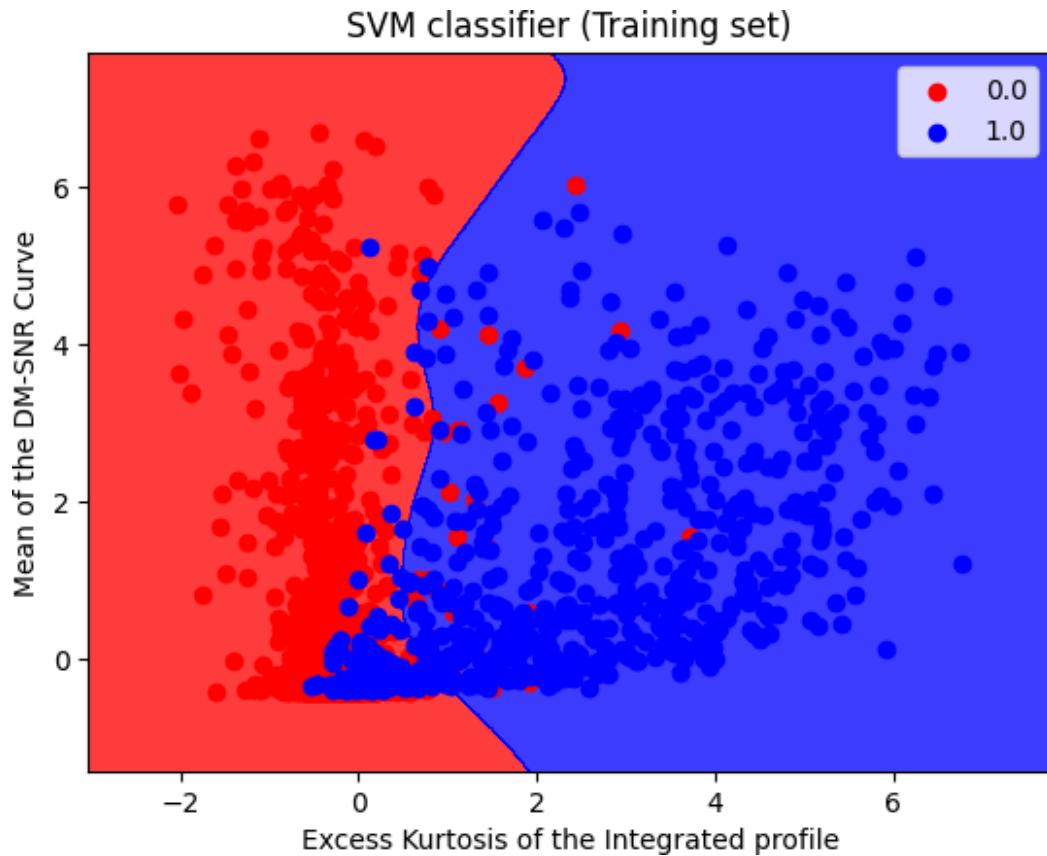
```
array([[2094,    14],  
       [ 49, 162]])
```

```

from matplotlib.colors import
ListedColormap x_set, y_set = x_train,
y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step = 0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, svclassifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'blue')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
    ListedColormap(('red', 'blue'))(i), label = j)
plt.title('SVM classifier (Training set)')
plt.xlabel('Excess Kurtosis of the Integrated
profile') plt.ylabel('Mean of the DM-SNR Curve')
plt.legend()
plt.show()

<ipython-input-11-1dfbd24c8ad3>:13: UserWarning: *c* argument
looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length
matches with *x* & *y*. Please use the *color* keyword-argument
or provide a 2D array with a single row if you intend to specify
the same RGB or RGBA value for all points.
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
ListedColormap(('red', 'blue'))(i), label = j)

```



Visualizing the test set result:

```

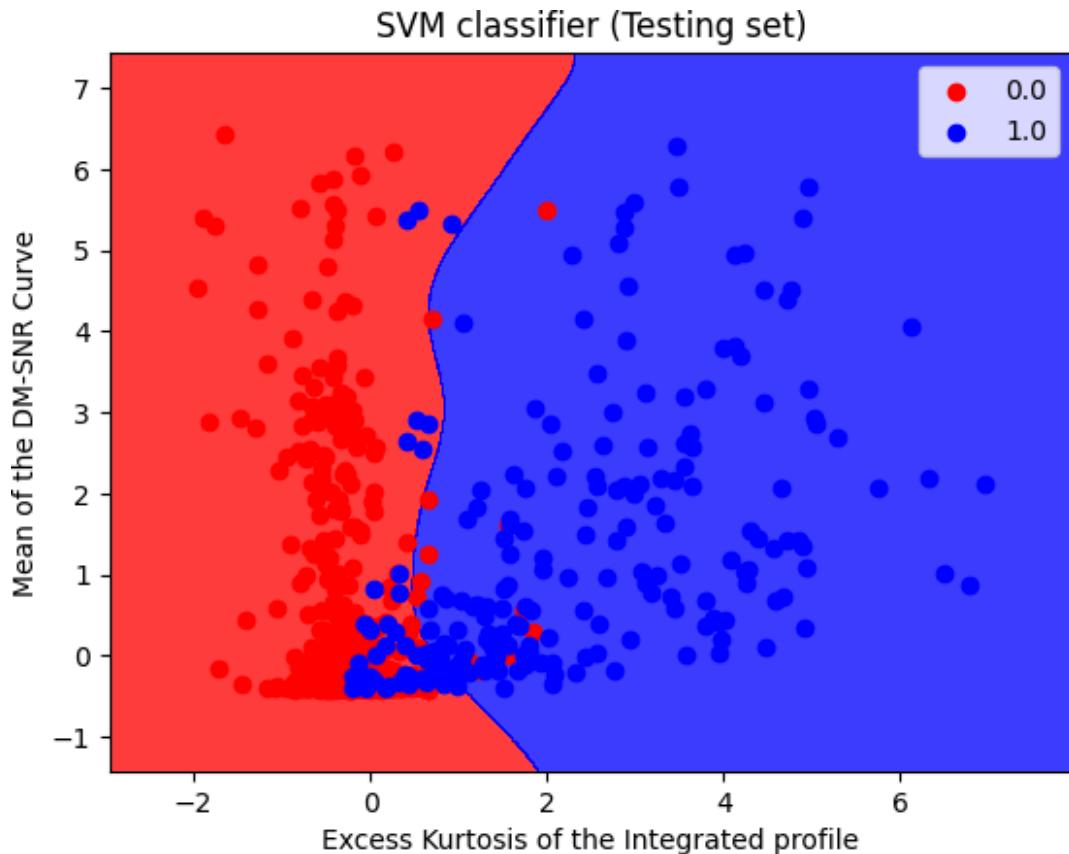
from matplotlib.colors import
ListedColormap x_set, y_set = x_test,
y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step = 0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, svclassifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'blue')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
    ListedColormap(('red', 'blue'))(i), label = j)

plt.title('SVM classifier (Testing set)')
plt.xlabel('Excess Kurtosis of the Integrated
profile')
plt.ylabel('Mean of the DM-SNR Curve')
plt.legend()
plt.show()

```

```
<ipython-input-12-a08ec9d95802>:13: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```

```
plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c = ListedColormap(('red', 'blue'))(i), label = j)
```



Conclusion: I have learned and understood the implementation of Classifying data using Non-Linear Kernels in Support Vector Machines (SVMs).

<b>Name of Student:</b> Pushkar Prasad Sane		
<b>Roll Number:</b> 45	<b>Lab Assignment Number:</b> 9	
<b>Title of Lab Assignment:</b>		
<b>Implementation of Classifying data using Non-Linear Kernels in Support Vector Machines (SVMs).</b>		
<b>DOP:</b> 08/3/24	<b>DOS:</b> 15/3/24	
<b>CO:</b> CO2, CO4	<b>PO:</b> POfi, PO2, PO3, PO4 , PO5, PO6, PO7, PO 8, PO9, POififi, POfi2 PSOfi, PSO2	<b>Faculty Signature:</b>

**Aim:** Implementation of Bagging Algorithm: Random Forest.

Random Forest is an ensemble learning algorithm that combines the predictions of multiple decision trees to make more accurate and robust predictions. It belongs to the family of bagging algorithms.

The algorithm creates an ensemble of decision trees, where each tree is trained on a random subset of the original dataset, known as bootstrap samples. These bootstrap samples are created by randomly selecting data points with replacement from the original dataset.

At each node of a decision tree, instead of considering all features, a random subset of features is considered for splitting. This process helps to introduce randomness and reduce the correlation among individual trees, making the forest more diverse and less prone to overfitting.

The decision trees in the random forest are grown until a stopping criterion is met, such as reaching a maximum depth or a minimum number of samples at a leaf node. This ensures that the trees do not become too complex or overfit to the training data.

During the prediction phase, each tree in the random forest independently predicts the class (for classification) or value (for regression) of a new data point. The final prediction is determined by aggregating the predictions of all the trees, usually by majority voting for classification or averaging for regression.

Random Forest algorithm offers several benefits, including high accuracy, robustness against overfitting, and the ability to handle large datasets with high-dimensional features. It is widely used in various domains, such as finance, healthcare, and image recognition, for tasks like classification, regression, and feature selection.

**Working of Random Forest:** Random Forest works in two-phase first is to create the random forest by combining N decision trees, and second is to make predictions for each tree created in the first phase.

- Step-1: Select random K data points from the training set.
- Step-2: Build the decision trees associated with the selected data points (Subsets).
- Step-3: Choose the number N for decision trees that you want to build.
- Step-4: Repeat Step 1 S 2
- Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```

import seaborn as sns
from sklearn.metrics import accuracy_score

dataset =
pd.read_csv('pulsar_data_train.csv')

dataset=dataset.dropna()
dataset.isnull().sum()

Mean of the integrated profile          0
Standard deviation of the integrated profile 0
Excess kurtosis of the integrated profile   0
Skewness of the integrated profile         0
Mean of the DM-SNR curve                 0
Standard deviation of the DM-SNR curve    0
Excess kurtosis of the DM-SNR curve       0
Skewness of the DM-SNR curve              0
target_class                           0
dtype: int64

dataset.head

()

Mean of the integrated profile \
0           121.156250
1           76.968750
2          130.585938
4           84.804688
7          109.406250

Standard deviation of the integrated profile \
0            48.372971
1            36.175557
2            53.229534
4            36.117659
7            55.912521

Excess kurtosis of the integrated profile \
0            0.375485
1            0.712898
2            0.133408
4            0.825013
7            0.565106

Skewness of the integrated profile  Mean of the DM-SNR curve \
0           -0.013165          3.168896
1            3.388719          2.399666
2           -0.297242          2.743311
4            3.274125          2.790134
7            0.056247          2.797659

Standard deviation of the DM-SNR curve \
0            18.399367

```

```

1          17.570997
2          22.362553
4          20.618009
7          19.496527

    Excess kurtosis of the DM-SNR curve   Skewness of the DM-SNR
    curve

\

0          7.449874
65.159298

1          9.414652
102.722975

2          8.508364
74.031324

4          8.405008
76.291128

7          9.443282
97.374578

target_clas
s
0          0.0
1          0.0
2          0.0
4          0.0
7          0.0

x = dataset.iloc[:,  

[2,4]] y =  

dataset.iloc[:, -1]

from sklearn.model_selection import train_test_split  

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size  

= 0.25, random_state = 0)  

x_train, x_test, y_train, y_test

(      Excess kurtosis of the integrated profile   Mean of the
DM- SNR curve
7731          0.709628
1.566890
3091          0.084360
2.567726
11452          0.257193
2.596990
10631          -0.029067
1.220736
5401          0.181712
2.511706
...
...
10669          0.085125
1.689799
12459          0.730128

```

55.751672	
6544	-0.183962
3.829431	
4382	0.277253
0.954013	
3661	0.018395
2.868729	
[6954 rows x 2 columns],	
SNR	Excess kurtosis of the integrated profile    Mean of the DM-
curve	0.454148
11372	
3.692308	
3450	0.031220
13.680602	
7024	1.417642
33.632943	
1366	-0.109619
1.617893	
2444	0.126617
2.770067	..
...	
...	.
2895	0.08977
4	
2.506689	
3945	0.095741
5.466555	
7197	1.065848
34.447324	
9204	0.361255
2.240803	
8307	0.403299
1.921405	
[2319 rows x 2 columns],	
7731	0.0
3091	0.0
11452	0.0
10631	0.0
5401	0.0
...	
10669	0.0
12459	0.0
6544	0.0
4382	0.0
3661	0.0
Name: target_class, Length: 6954, dtype:	
	float64,
11372	0.0
3450	0.0
7024	1.0

```
1366      0.0
2444      0.0
...
2895      0.0
3945      0.0
7197      0.0
9204      0.0
8307      0.0
Name: target_class, Length: 2319, dtype: float64)
```

```
cols = x_train.columns
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

Fitting the SVM classifier to the training set:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=5)
classifier.fit(x_train,y_train)

RandomForestClassifier(n_estimators=5)
```

Predicting the test set result:

```
y_pred =
classifier.predict(x_test)
y_pred

array([0., 0., 0., ..., 0., 0., 0.])
```

Creating the confusion matrix:

```
from sklearn.metrics import confusion_matrix
cm =
confusion_matrix(y_test,y_pred)
cm

array([[2085,   23],
       [ 42, 169]])
```

Visualizing the training set result:

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train,
y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step = 0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
```

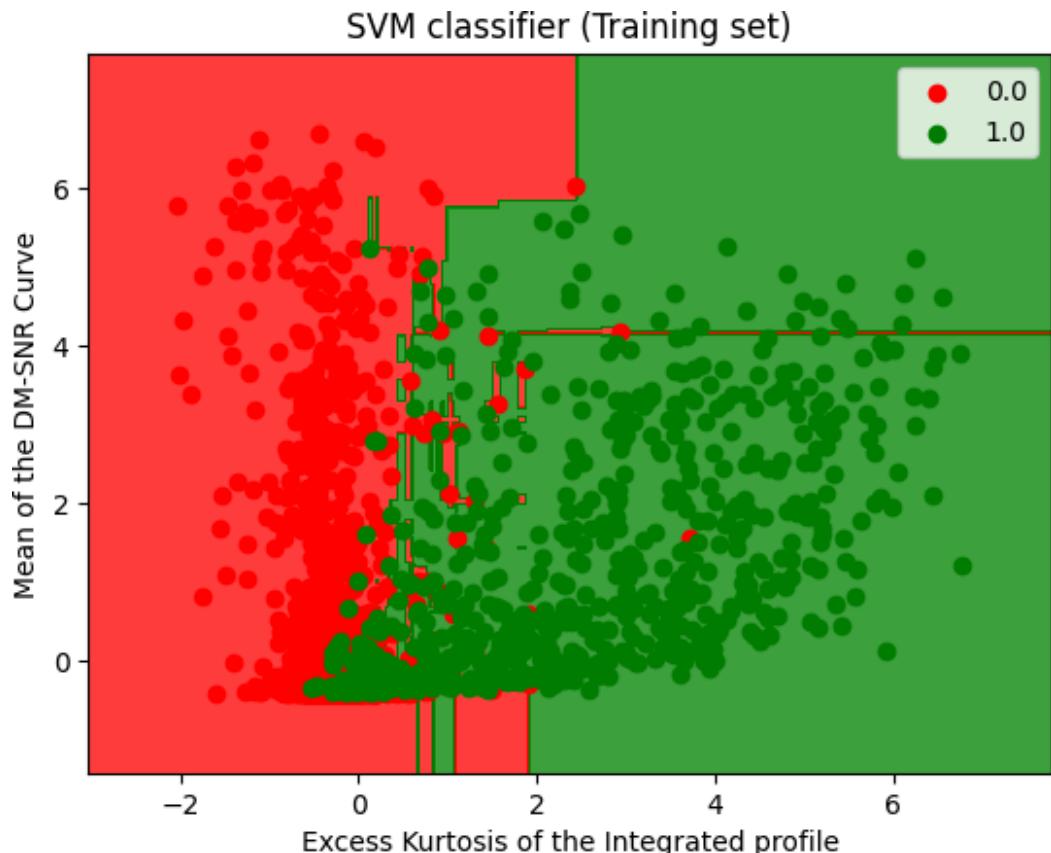
```

for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
    ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM classifier (Training set)')
plt.xlabel('Excess Kurtosis of the Integrated profile')
plt.ylabel('Mean of the DM-SNR Curve')
plt.legend()
plt.show()

<ipython-input-11-7b6a7b8c2129>:13: UserWarning: *c* argument
looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length
matches with *x* & *y*. Please use the *color* keyword-argument
or provide a 2D array with a single row if you intend to specify
the same RGB or RGBA value for all points.
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1], c =
    ListedColormap(('red', 'green'))(i), label = j)

```



```

#Visualizing the test set result:
from matplotlib.colors import
ListedColormap x_set, y_set = x_test,
y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step =0.01),

```

```

np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('SVM classifier (Testing set)')
plt.xlabel('Excess Kurtosis of the Integrated profile')
plt.ylabel('Mean of the DM-SNR Curve')
plt.legend()
plt.show()

<ipython-input-13-d0470b10fd12>:14: UserWarning: *c* argument
looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length
matches with *x* & *y*. Please use the *color* keyword-argument
or provide a 2D array with a single row if you intend to specify
the same RGB or RGBA value for all points.
plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],

```

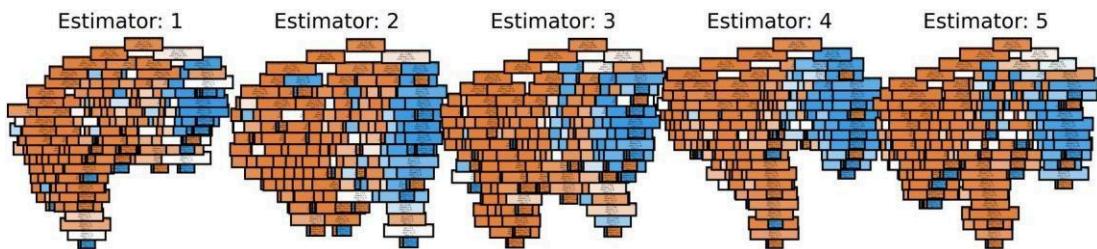


```
accuracy_score(y_test,y_pred)
```

```
0.9719706770159552
```

Plotting the estimator trees:

```
from sklearn import tree #plot the RandomForestClassifier's first 5 trees
fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (10,2), dpi=900)
for index in range(0, 5):
    tree.plot_tree(classifier.estimators_[index],feature_names=x.columns, class_names= 'status',filled = True,ax = axes[index])
    axes[index].set_title('Estimator: ' + str(index+1), fontsize = 11)
fig.savefig('Random Forest 5 Trees.png')
```



Conclusion: From this practical I have learned the implementation of Bagging Algorithm: Random Forest.

<b>Name of Student:</b> Pushkar Prasad Sane		
<b>Roll Number:</b> 45	<b>Lab Assignment Number:</b> fi0	
<b>Title of Lab Assignment:</b>		
<b>Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.</b>		
<b>DOP:</b> 15/3/24	<b>DOS:</b> 22/3/24	
<b>CO:</b> CO2, CO4	<b>PO:</b> POfi, PO2, PO3, PO4 , PO5, PO6, PO7, PO 8, PO9, POfifi, POfi2 PSOfi, PSO2	<b>Faculty Signature:</b>

**Aim: Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.**

### **About Iris Dataset:**

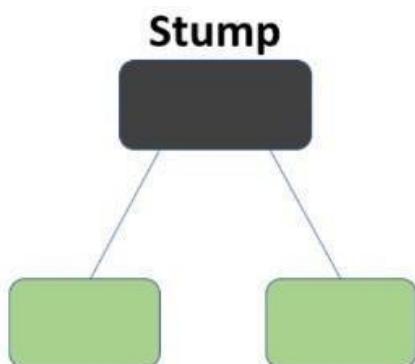
The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, can also be found on the UCI Machine Learning Repository. It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other. The columns in this dataset are:

- Id
- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species

### **AdaBoost:**

AdaBoost also called Adaptive Boosting is a technique in Machine Learning used as an Ensemble Method. The most common algorithm used with AdaBoost is decision trees with one level that means with Decision trees with only 1 split. These trees are also called Decision Stumps. What this algorithm does is that it builds a model and gives equal weights to all the data points. It then assigns higher weights to points that are wrongly classified.

Now all the points which have higher weights are given more importance in the next model. It will keep training models until and unless a lower error is received.

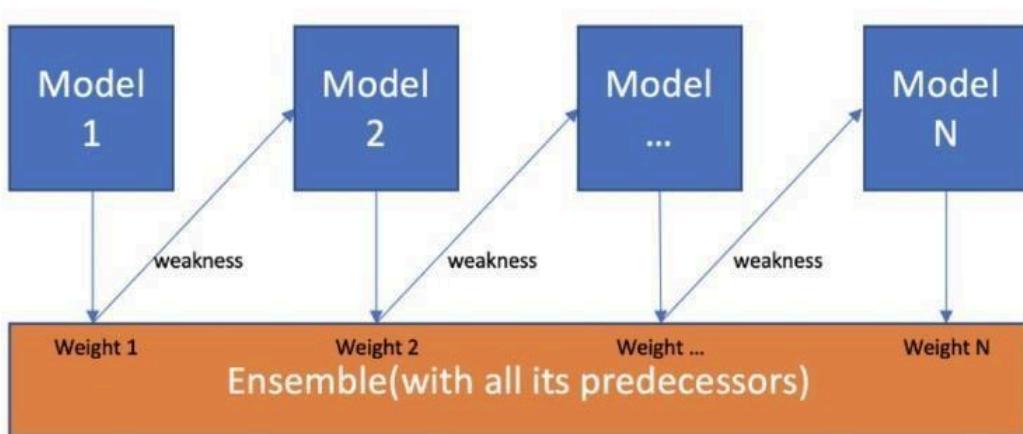


### **Working of AdaBoost:**

- Initially, Adaboost selects a training subset randomly.

- It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.
- It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.
- Also, It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.
- This process iterates until the complete training data fits without any error or until reached to the specified maximum number of estimators.
- To classify, perform a "vote" across all of the learning algorithms you built.

Model 1,2,..., N are individual models (e.g. decision tree)



```

import numpy as nm
import pandas as pd
from sklearn.datasets import load_iris

iris =
load_iris() x =
iris.data[:, :4]
y = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test,
y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)

```

Fitting the AdaBoost classifier to the training set:

```

from sklearn.ensemble import AdaBoostClassifier
adaboost = AdaBoostClassifier(n_estimators=100,
base_estimator=DecisionTreeClassifier(),learning_ra
te=1) Sidharth_model =
adaboost.fit(x_train,y_train) Sidharth_model

/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:16
6: FutureWarning: `base_estimator` was renamed to `estimator` in
version

```

```

1.2 and will be removed in 1.4.
    warnings.warn(
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(),
learning_rate=1,
                n_estimators=100)

y_pred = Sidharth_model.predict(x_test)
from sklearn.metrics import
accuracy_score print(y_test.tolist())
print(y_pred)
print("The accuracy of the model on validation set is",
accuracy_score(y_test,y_pred))

[0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
1,
0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 2, 2, 1]
[0 1 1 0 2 1 2 0 0 2 1 0 2 1 1 0 1 1 2 0 2 1 0 0 1 2 1 2 1 2
2
0 1
0 1 2 2 0 1 2 1]
The accuracy of the model on validation set is 0.9555555555555555

# Define the class names
class_names = ['setosa', 'versicolor', 'virginica']

# Convert numerical labels to class names in y_test
y_test_labels = [class_names[label] for label in y_test]

# Convert numerical labels to class names in y_pred
y_pred_labels = [class_names[label] for label in y_pred]

# Print y_test and y_pred with class names
print(y_test_labels)
print(y_pred_labels)

# Calculate accuracy using class names
accuracy = accuracy_score(y_test_labels, y_pred_labels)
print("The accuracy of the model on the validation set is:",
accuracy)

['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',
'versicolor', 'virginica', 'setosa', 'setosa', 'virginica',
'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
isetosa', 'versicolor', 'versicolor', 'setosa', 'setosa',
'versicolor', 'versicolor', 'versicolor', 'setosa', 'virginica',
'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica',
'versicolor', 'virginica', 'versicolor', 'virginica',
'verginica', 'setosa', 'versicolor', 'setosa', 'versicolor',
'verginica', 'virginica', 'setosa', 'virginica', 'virginica',
'versicolor'] ['setosa', 'versicolor', 'versicolor', 'setosa',
'verginica', 'versicolor', 'virginica', 'setosa', 'setosa',
'verginica', 'versicolor', 'setosa', 'virginica', 'versicolor',
'versicolor', 'setosa', 'versicolor', 'versicolor', 'setosa',
'setosa',

```

```

'versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica',
'versicolor', 'virginica', 'versicolor', 'virginica', 'virginica',
'setosa', 'versicolor', 'setosa', 'versicolor', 'virginica',
'virginica', 'setosa', 'versicolor', 'virginica', 'versicolor']
The accuracy of the model on the validation set is:
0.9555555555555556

```

## Gradient Descent

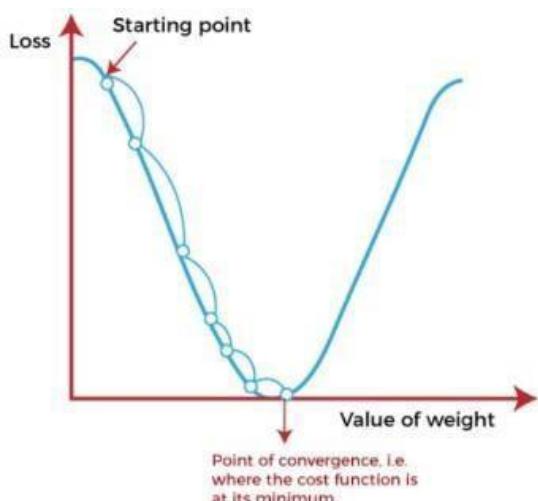
Gradient descent (GD) is an iterative first-order optimization algorithm used to find a local minimum/maximun of a given function. This method is commonly used in machine learning

(ML) and deep learning(DL) to minimize a cost/loss function (e.g. in a linear regression).

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

- If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the local minimum of that function.
- Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the local maximum of that function.

**Working of Gradient Descent:** The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias). The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called a point of convergence. The main objective of gradient descent is to minimize the cost function or the error between expected and actual. To minimize the cost function, two data points are required:



### Stochastic Gradient Descent:

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration. Or in other words, it processes a training epoch for each example within a database and updates each training example's parameters one at a time.

```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import
GradientBoostingClassifier from
sklearn.linear_model import LogisticRegression from
sklearn.neighbors import KNeighborsClassifier from
sklearn.svm import SVC
from xgboost import XGBClassifier
from numpy import loadtxt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = pd.read_csv('diabetes.csv', delimiter=',')
data.head()

   Pregnancies    Glucose    BloodPressure    SkinThickness    Insulin
s BMI \
0          6        148                 72                  35          0    33.6
1          1         85                 66                  29          0    26.6
2          8        183                 64                  0          0    23.3
3          1         89                 66                  23         94    28.1
4          0        137                 40                  35        168    43.1

   DiabetesPedigreeFunction    Age    Outcome
0            0.627      50          1
1            0.351      31          0
2            0.672      32          1
3            0.167      21          0
4            2.288      33          1

x = data.iloc[:,0:8].values
y = data.iloc[:,8].values

x_train, x_test, y_train,
y_test=train_test_split(x,y,test_size=0.2,random_state=1)
```

Running various models -

```

models = []
models.append(('Logistic Regression', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))

models.append(('SVM', SVC()))
models.append(('XGB', XGBClassifier(eta=0.01, gamma=10)))

import time
results = []
name = []
scoring = 'accuracy'

for name,model in models:
    start_time = time.time()
    model.fit(x_train,y_train)
    y_pred =
    model.predict(x_test)
    predictions = [round(value) for value in y_pred]
    accuracy = accuracy_score(y_test, predictions)
    print("Accuracy: %.2f%%" % (accuracy * 100.0),
    name) print("--- %s seconds ---" % (time.time() -
    start_time))

Accuracy: 77.92% Logistic Regression
--- 0.044225454330444336 seconds ---
Accuracy: 73.38% KNN
--- 0.01197052001953125 seconds ---
Accuracy: 78.57% SVM
--- 0.018146038055419922 seconds ---
Accuracy: 78.57% XGB
--- 0.14017462730407715 seconds ---

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/
_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the
data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver
options:

https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(

```

### Voting Ensemble:

A voting ensemble (or a “majority voting ensemble”) is an ensemble machine learning model that combines the predictions from multiple other models. It is a technique that may be used to improve model performance, ideally achieving better performance than any single model used in the ensemble. A voting ensemble works by combining the predictions from multiple models. It can be used for classification or regression. In the case of

regression, this involves calculating the average of the predictions from the models. In the case of classification, the predictions for each label are summed and the label with the majority vote is predicted.

- Regression Voting Ensemble: Predictions are the average of contributing models.
- Classification Voting Ensemble: Predictions are the majority vote of contributing models.

There are two approaches to the majority vote prediction for classification; they are hard voting and soft voting.

**Hard voting** involves summing the predictions for each class label and predicting the class label with the most votes. Soft voting involves summing the predicted probabilities (or probability-like scores) for each class label and predicting the class label with the largest probability.

- Hard Voting: Predict the class with the largest sum of votes from models
- Soft Voting: Predict the class with the largest summed probability from models.

```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris =
load_iris() x =
iris.data[:, :4]
y = iris.target

x_train, x_test, y_train,
y_test=train_test_split(x,y,test_size=0.20,random_state=42)

estimator = []
estimator.append(('Logistic Regression',
LogisticRegression(solver = 'lbfgs', multi_class =
'multinomial', max_iter = 200)))
estimator.append(('SVC',
SVC(gamma = 'auto', probability = True)))
estimator.append(('Decision Tree Classifier',
DecisionTreeClassifier()))

vote_hard = VotingClassifier(estimators=estimator, voting='hard')
vote_hard.fit(x_train, y_train)
y_pred = vote_hard.predict(x_test)
score = accuracy_score(y_test,
y_pred) print("Hard Voting Score
%d" % score)

Hard Voting Score 1

vote_soft = VotingClassifier(estimators=estimator, voting='soft')
vote_soft.fit(x_train, y_train)
```

```
y_pred = vote_soft.predict(x_test)
score = accuracy_score(y_test,
y_pred) print("Soft Voting Score
%d" % score)
```

```
Soft Voting Score 1
```

Conclusion: I have successfully understood & implemented Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.

<b>Name of Student:</b> Pushkar Prasad Sane		
<b>Roll Number:</b> 45		<b>Lab Assignment Number:</b> fifi
<b>Title of Lab Assignment:</b>		
<b>Principal Component Analysis Aim:</b> Implementation of dimensionality reduction techniques: Features Extraction and Selection, Normalization, Transformation, and Principal Components Analysis.		
<b>DOP:</b> 22/3/24		<b>DOS:</b> 2 9 / 3 / 2 4
<b>CO:</b> CO3	<b>PO:</b> PO2,PO3, PO4 ,PO5,PO6,PO7, PSOfi, PSO2	<b>Faculty Signature:</b>

Practical 11: Principal Component Analysis Aim: Implementation of dimensionality reduction techniques: Features Extraction and Selection, Normalization, Transformation, Principal Components Analysis.

Theory: Principal Component Analysis (PCA) Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. PC is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover, PCA is an unsupervised statistical technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of fit.best

Purpose Feature Extraction and Selection:

PCA can be used to extract the most important features from a dataset by identifying the principal components. These components are linear combinations of the original features that capture the maximum variance in the data. To perform feature extraction with PCA, you start by normalizing the dataset (explained in the next point). Then, you calculate the covariance matrix of the normalized data. Next, you compute the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors represent the principal components, and the corresponding eigenvalues indicate the amount of variance explained by each component. You can then select a subset of the principal components based on their corresponding eigenvalues or by setting a threshold for the amount of variance you want to retain. This allows you to reduce the dimensionality of the dataset while retaining the most important information. Normalization:

PCA requires that the variables in the dataset are on the same scale to avoid biasing the analysis towards variables with larger ranges. Normalization can be achieved by subtracting the mean of each feature and dividing by the standard deviation (standardization) or by scaling the features to a specific range, such as [0, 1] or [-1, 1]. Standardizing the variables is a common choice as it ensures that all variables have zero mean and unit variance, making them comparable. 3 . Transformation:

After normalizing the dataset, you can perform the actual PCA transformation. The normalized dataset is projected onto the new feature space defined by the principal components. Each data point is transformed into a set of values corresponding to the principal components, representing a lower-dimensional representation of the original data. The transformed data can be used for visualization, clustering, classification, or further analysis. It's important to note that PCA assumes linearity and may not be suitable for datasets with nonlinear relationships. In such cases, nonlinear dimensionality reduction techniques like t-SNE or autoencoders may be more appropriate.

Overall, PCA is a powerful tool for dimensionality reduction, feature extraction, and selection, and it can help to simplify complex datasets while retaining information.

fit.

Implementation Analysis

```

import numpy as np
import pandas as pd
import pprint
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
%matplotlib inline
%precision 3
np.set_printoptions(precision=3)
) import pylab as pl

iris = load_iris()

iris_df = pd.DataFrame(iris.data,columns=[iris.feature_names])
iris_df.head()

    sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)
0                 5.1              3.5             1.4
0.2
1                 4.9              3.0             1.4
0.2
2                 4.7              3.2             1.3
0.2
3                 4.6              3.1             1.5
0.2
4                 5.0              3.6             1.4
0.2

X =
iris.data
X.shape

(150, 4)

```

## Scaling

```

from sklearn.preprocessing import
StandardScaler X_std =
StandardScaler().fit_transform(X)
print (X_std[0:5])
print ("The shape of Feature Matrix is
-",X_std.shape) from sklearn.preprocessing import
StandardScaler X_std =
StandardScaler().fit_transform(X)
print (X_std[0:5])
print ("The shape of Feature Matrix is -",X_std.shape)

[[ -0.901  1.019 -1.34 -1.315]
 [ -1.143 -0.132 -1.34 -1.315]
 [ -1.385  0.328 -1.397 -1.315]
 [ -1.507  0.098 -1.283 -1.315]
 [ -1.022  1.249 -1.34 -1.315]]
The shape of Feature Matrix is -
(150, 4) [[ -0.901  1.019 -1.34 -1.315]
 [ -1.143 -0.132 -1.34 -1.315]

```

```

[-1.385 0.328 -1.397 -1.315]
[-1.507 0.098 -1.283 -1.315]
[-1.022 1.249 -1.34 -1.315]]
The shape of Feature Matrix is - (150, 4)

```

## Covariance

```

X_covariance_matrix = np.cov(X_std.T)
X_covariance_matrix

array([[ -0.118,   0.878,   0.823],
       1.007,
      [-0.118,   1.007,  -0.431,  -0.369],
      [ 0.878,  -0.431,   1.007,   0.969],
      [ 0.823,  -0.369,   0.969,   1.007]])

```

## Eigen Vectors

```

eig_vals, eig_vecs =
np.linalg.eig(X_covariance_matrix)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)

Eigenvectors
[[ 0.521 -0.377 -0.72   0.261]
 [-0.269 -0.923  0.244 -0.124]
 [ 0.58  -0.024  0.142 -0.801]
 [ 0.565 -0.067  0.634  0.524]]

Eigenvalues
[2.938 0.92 0.148 0.021]

# Make a list of (eigenvalue, eigenvector) tuples
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for
i in range(len(eig_vals))]
# Sort the (eigenvalue, eigenvector) tuples from high to low
eig_pairs.sort(key=lambda x: x[0], reverse=True)
# Visually confirm that the list is correctly sorted by decreasing
eigenvalues
print('Eigenvalues in descending order:')
for i in eig_pairs:
    print(i[0])

Eigenvalues in descending order:
2.938085050199995
0.9201649041624864
0.1477418210449475
0.020853862176462696

tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
print ("Variance captured by each component is
\n",var_exp) print(40 * '-')

```

```

print ("Cumulative variance captured as we travel each component \
n",cum_var_exp)

Variance captured by each component is
[72.96244541329989, 22.850761786701753, 3.668921889282865,
0.5178709107154905]
-----
Cumulative variance captured as we travel each
component [ 72.962 95.813 99.482 100.      ]

print ("All Eigen Values along with Eigen Vectors")
pprint.pprint(eig_pairs)
print(40 * '-')
matrix_w = np.hstack((eig_pairs[0][1].reshape(4,1),
eig_pairs[1][1].reshape(4,1)))

print ('Matrix W:\n', matrix_w)

All Eigen Values along with Eigen Vectors
[(2.938085050199995, array([ 0.521, -0.269,  0.58 ,
0.565])),
(0.9201649041624864, array([-0.377, -0.923, -0.024, -0.067])),
(0.1477418210449475, array([-0.72 ,  0.244,  0.142,  0.634])),
(0.020853862176462696, array([ 0.261, -0.124, -0.801,  0.524]))]
-----
Matrix W:
[[ 0.521 -0.377]
 [-0.269 -0.923]
 [ 0.58 -0.024]
 [ 0.565 -0.067]]

Y = X_std.dot(matrix_w)
print (Y[0:5])

[[-2.265 -0.48 ]
 [-2.081  0.674]
 [-2.364  0.342]
 [-2.299  0.597]
 [-2.39   -0.647]]

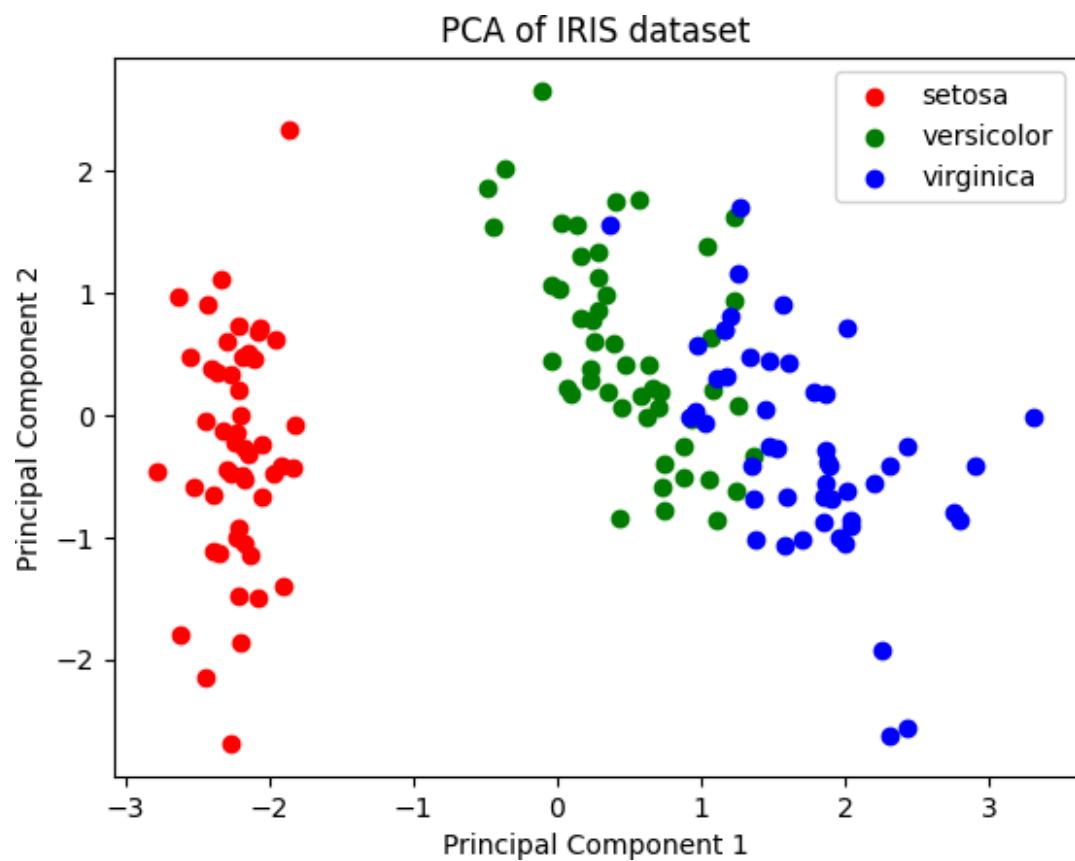
```

## Plotting PCA

```

pl.figure()
target_names =
iris.target_names y =
iris.target
for c, i, target_name in zip("rgb", [0, 1, 2], target_names):
    pl.scatter(Y[y==i,0], Y[y==i,1], c=c, label=target_name)
pl.xlabel('Principal Component 1')
pl.ylabel('Principal Component 2')
pl.legend()
pl.title('PCA of IRIS dataset')
pl.show()

```



**Conclusion:** I have successfully implemented various dimensionality reduction techniques and learnt how to extract features, normalize data, and do principal component analysis on the data.

<b>Name of Student:</b> Pushkar Prasad Sane		
<b>Roll Number:</b> 45	<b>Lab Assignment Number:</b> fi2	
<b>Title of Lab Assignment:</b>		
<b>Implementation of Boosting Algorithms: AdaBoost, Stochastic Gradient Boosting, Voting Ensemble.</b>		
<b>DOP:</b> 29/3/24	<b>DOS:</b> 05/4/24	
<b>CO:</b> CO2, CO4	<b>PO:</b> POfi, PO2, PO3, PO4 , PO5, PO6, PO7, PO 8, PO9, POfifi, POfi2 PSOfi, PSO2	<b>Faculty Signature:</b>

## PRACTICAL-12

**Aim:** Deployment of Machine Learning Models

### Theory:

#### ML-Model-Flask-Deployment:

We hope to deploy a ML model based on the Insurance dataset.

This project has four major parts :

- model.py - This contains code a wrapper function for train.py. Function get\_model() returns a model if already present and if not, then it builds a new model and returns it. When more than one models are present, it picks the most recent model.
- train.py – This contains code for our Machine Learning model to predict insurance charges based on training data in given file.
- api.py - This contains Flask APIs that receives client details through GUI, computes the predicted value based on our model and returns it.
- dev.py – This file runs the project in a dev environment
- wsgi.py – This file will allow us to deploy it on a machine using wsgi interface with deployment tools like gunicorn.
- templates - This folder contains the HTML template to allow users to enter their details and displays the predicted insurance charges.

Running the project:

The dev server can simply be started by executing the following while in project root directory.

>python dev.py

or

>py dev.py

Then we need to navigate to port 5000 of localhost or 127.0.0.1 with http.

When the machine learning models are not present, they will be generated automatically.

Additionally the project can be deployed using a wsgi server while in a production environment.

### Execution:

requirements.txt:

```
scikit-learn==1.2.2
Flask==2.2.3 pandas==2.0.1
```

api.py:

```
from model import get_model
from flask import Flask, request, jsonify, render_template, send_from_directory app =
Flask(__name__)

@app.route('/') def
index():
```

```

return render_template('./index.html')

@app.route('/images/<path:path>') def
send_images(path):
    return send_from_directory('images', path)
@app.route('/predict', methods=['POST'])
@app.route('/api/predict/insurance', methods=['POST']) def
api():
    if request.method == 'POST': data
        = request.get_json() print (data)
    model = get_model() if
        not model:
            return jsonify({'error': 'no model found'}), 500
    elif data.get('age') is None or data.get('sex') is None or data.get(
        'bmi') is None or data.get('children') is None or data.get('smoker') is None: return
        jsonify({'error': 'missing data'}), 400
    else:
        prediction = model.predict([[[
            data['age'],
            •   if data['sex']=="male" else 1,
            data['bmi'],
            data['children'],
            •   if data['smoker']=="yes" else 0
        ]])
        print (prediction[0])
        return jsonify({'charges': prediction[0]}), 200

if __name__      == '_main__':
    app.run(debug=True)

```

model.py:

```

import pickle
import glob
from train import train

def get_model():
    list_of_files = glob.glob("models/insurance-model_*.pkl")
    list_of_files.sort(reverse=True)
    try:
        model = pickle.load(open(list_of_files[0], "rb")) except
IndexError:
        print("No model found, Generating one now...") train()
        return get_model() return
    model

```

train.py:

```

import pickle

```

```

from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pandas as pd
def train() -> None:
    original_df = pd.read_csv("./insurance.csv")
    smoker = {"yes": 1, "no": 0}
    original_df["smoker"] = original_df["smoker"].map(smoker)
    sex = {"male": 0, "female": 1}
    original_df["sex"] = original_df["sex"].map(sex)

    x = original_df.iloc[:, 0:5].values
    y = original_df.iloc[:, 6].values
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
    regressor = LinearRegression()

    regressor.fit(x_train, y_train)
    y_pred = regressor.predict(x_test)
    predictions = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
    print(predictions)
    print(regressor.score(x_test, y_test))
    pickle.dump(
        regressor,
        open(f'models/insurance-model_{datetime.now().strftime("%Y-%m-%d_%H-%M-%S")}.pkl', "wb"),
    )

if __name__ == "__main__":
    train()

```

wsgi.py:

```

from api import app

if __name__ == "__main__":
    app.run()

```

dev.py:

```

from api import app

if __name__ == '__main__':
    app.run(debug=True, host='localhost', port=5000)

```

templates/index.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">

```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Beautiful Website</title>
<link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">
<style>
.bg-image {
  background-image: url('/images/desktop-1920x1080.jpeg');
  background-repeat: no-repeat;
  background-size: cover;
}
</style>
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script type="text/javascript">
function predict() {
  const age = document.getElementById('age').value;
  const sex = document.getElementById('sex').value;
  const bmi = document.getElementById('bmi').value;
  const children = document.getElementById('children').value;
  const smoker = document.getElementById('smoker').value;
  const region = document.getElementById('region').value;
  axios.post('http://localhost:5000/api/predict/insurance', {
    age:parseInt(age),
    bmi:parseFloat(bmi)
    , smoker,
    sex,
    region
    ,
    children:parseInt(children)
  })
  .then(function (response) {
    document.getElementById('prediction-div').hidden = false;
    document.getElementById('Prediction').innerHTML = response.data.charges;
  })
  .catch(function (error) {
    console.log(error);
  });
}
</script>
</head>
<body class="bg-image">
<div class="min-h-screen flex items-center justify-center">
<div class="bg-white p-8 rounded shadow-lg">
<h1 class="text-3xl font-bold mb-4">Beautiful Website</h1>
<form>
<div class="mb-4">
<label class="block text-gray-700 text-sm font-bold mb-2" for="age"> Age
</label>
<input class="appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="age" type="text" placeholder="Enter Age">
</div>
<div class="mb-4">
```

```
<label class="block text-gray-700 text-sm font-bold mb-2" for="sex"> Sex
</label>
<select class="appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="sex">
    <option value="male">Male</option>
    <option value="female">Female</option>
</select>
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="bmi"> BMI
    </label>
    <input class="appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="bmi" type="text" placeholder="Enter BMI">
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="children"> Children
    </label>
    <input class="appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="children" type="text" placeholder="Enter Nol. of Children">
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="smoker"> Are you a smoker?
    </label>
    <select class="appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="smoker">
        <option value="yes">Yes</option>
        <option value="no">No</option>
    </select>
</div>
<div class="mb-4">
    <label class="block text-gray-700 text-sm font-bold mb-2" for="region"> Region
    </label>
    <select class="appearance-none border rounded w-full py-2 px-3 text-gray-700 leading-tight focus:outline-none focus:shadow-outline" id="region">
        <option value="northeast">Northeast</option>
        <option value="southwest">Southwest</option>
        <option value="northwest">Northwest</option>
        <option value="southeast">Southeast</option>
    </select>
</div>
<button class="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded focus:outline-none focus:shadow-outline" type="button" onclick="predict()">
    Submit
</button>
</form>
```

```
<div class="mb-4" id="prediction-div" hidden>
  <label class="block text-gray-700 text-sm font-bold mb-2" for="region"> Prediction
  </label>
  <p id="Prediction" class="text-gray-700 text-lg"></p>
</div>
</div>
</div>
</div>
</body>
</html>
```

Output :

The image shows two side-by-side screenshots of a web application. Both screenshots have a teal header bar with the text 'localhost:5000'. The left screenshot shows an input form with fields for Age (22), Sex (Male), BMI (26), Children (0), Are you a smoker? (No), and Region (Southwest). A blue 'Submit' button is at the bottom. The right screenshot shows the same form filled with the same values. Below the form, there is a 'Prediction' section containing the output '2097.384359034224'.

```
PS C:\Users\paxyi\Documents\GitHub\AIML-Practicals\19-AIML-Practical-12-insurance> py dev.py
* Serving Flask app "api"
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://localhost:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 562-471-338
127.0.0.1 - - [06/Jun/2023 18:59:07] "GET /images/desktop-1920x1080.jpeg HTTP/1.1" 304 -
{'age': 22, 'bmi': 26, 'smoker': 'no', 'sex': 'male', 'region': 'southwest', 'children': 0}
2097.384359034224
127.0.0.1 - - [06/Jun/2023 18:59:13] "POST /api/predict/insurance HTTP/1.1" 200 -
```

## Conclusion:

I have successfully deployed a machine learning model on Flask API in python.