Python with arslan

        - Normality test
        - Tests to be used
        - 2. Homogeneity test
        - Test to be used
        - Purpose
        - Two type of purpose
            - Comparison
            - Relationship
    - Data type
        - CATEGORICAL
        - CONTINUOUS
    - Statistical tests
        - 3 Families of statiscal tests
            - Chi-Squared
                - When and where to use?
            - t-Test/ANOVA
                - When and where to use?
            - Correlation
                - When and where to use?
    - Important Things
        - Assumptions about your data
        - If Assumptions are not met! then
            - Non-parametric test
        - ANOVA
            - Types of ANOVA
        - Some other test
            - Reliability tests
            - Inter-rater Reliability tests
        - Whole Process diagram
    - Statical hypotsis tests
        - Normality Tests
        - Correlation Tests
        - Stationary Tests
        - Parametric Statistical Hypothesis Tests
        - Nonparametric Statistical Hypothesis Tests
    - Normality Tests
        - Shapiro-Wilk Test
            - Assumptions
            - Interpretation
        - D'Agostino's K^2 Test
            - Assumptions
            - Interpretation
        - Anderson-Darling Test

Chapter 1 Python Fundamental

Return to the Top

Python Basics

Math Operators

From **Highest** to **Lowest** precedence:

Operators	Operation	Example
**	Exponent	`2 ** 3 = 8`
%	Modulus/Remainder	`22 % 8 = 6`
//	Integer division	`22 // 8 = 2`
/	Division	`22 / 8 = 2.75`
*	Multiplication	`3 * 3 = 9`
-	Subtraction	`5 - 2 = 3`

Operators	Operation	Example
+	Addition	`2 + 2 = 4`

Examples of expressions in the interactive shell:

```
2 + 3 * 6
20
```

```
(2 + 3) * 6
30
```

```
2 ** 8
256
```

```
23 // 7
3
```

```
23 % 7
2
```

```
(5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

Return to the Top

Data Types

Data Type	Examples
Integers	`-2, -1, 0, 1, 2, 3, 4, 5`
Floating-point numbers	`-1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25`
Strings	`'a', 'aa', 'aaa', 'Hello!', '11 cats'`

Return to the Top

String Concatenation and Replication

String concatenation:

```
'Arslan' 'babaji'
'Arslanbabaji'
```

Note: Avoid `+` operator for string concatenation. Prefer string formatting.

String Replication:

```
'Arslan' * 5
'ArslanArslanArslanArslanArslan'
```

Return to the Top

Variables

You can name a variable anything as long as it obeys the following rules:

1. It can be only one word.
2. It can use only letters, numbers, and the underscore (_) character.
3. It can't begin with a number.
4. Variable name starting with an underscore (_) are considered as "unuseful`.

Example:

```
 spam = 'Hello'
 spam
'Hello'
```

```
 _spam = 'Hello'
```

`_spam` should not be used again in the code.

Return to the Top

Comments

Inline comment:

```
# This is a comment
```

Multiline comment:

```
# This is a
# multiline comment
```

Code with a comment:

```
a = 1  # initialization
```

Please note the two spaces in front of the comment.

Function docstring:

```
def foo():
    """
    This is a function docstring
    You can also use:
    ''' Function Docstring '''
    """
```

Return to the Top

The print() Function

```
 print('Hello world!')
Hello world!
```

```
 a = 1
 print('Hello world!', a)
Hello world! 1
```

Return to the Top

The input() Function

Example Code:

```
print('What is your name?')   # ask for their name
myName = input()
print('It is good to meet you, {}'.format(myName))
What is your name?
Al
It is good to meet you, Al
```

Return to the Top

The len() Function

Evaluates to the integer value of the number of characters in a string:

```
len('hello')
5
```

Note: test of emptiness of strings, lists, dictionary, etc, should **not** use len, but prefer direct boolean evaluation.

```
a = [1, 2, 3]
if a:
 print("the list is not empty!")
```

Return to the Top

The str(), int(), and float() Functions

Integer to String or Float:

```
str(29)
'29'
```

```
print('I am {} years old.'.format(str(29)))
I am 29 years old.
```

```
str(-3.14)
'-3.14'
```

Float to Integer:

```
int(7.7)
7
```

```
int(7.7) + 1
8
```

Return to the Top

Flow Control

if Statements

```
if name == 'Arslan':
    print('Hi, Arslan.')
```

Return to the Top

else Statements

```
name = 'babaji'
if name == 'Arslan':
    print('Hi, Arslan.')
else:
    print('Hello, stranger.')
```

Return to the Top

elif Statements

```
name = 'babaji'
age = 5
if name == 'Arslan':
    print('Hi, Arslan.')
```

```
elif age < 12:
    print('You are not Arslan, kiddo.')
```

```
name = 'babaji'
age = 30
if name == 'Arslan':
    print('Hi, Arslan.')
elif age < 12:
    print('You are not Arslan, kiddo.')
else:
    print('You are neither Arslan nor a little kid.')
```

Return to the Top

while Loop Statements

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Return to the Top

break Statements

If the execution reaches a break statement, it immediately exits the while loop's clause:

```
while True:
    print('Please type your name.')
    name = input()
    if name == 'your name':
        break
print('Thank you!')
```

Return to the Top

continue Statements

When the program execution reaches a continue statement, the program execution immediately jumps back to the start of the loop.

```

```

```
while True:
    print('Who are you?')
    name = input()
    if name != 'Arslan':
        continue
    print('Hello, Arslan. What is the password? (It is a fish.)')
    password = input()
    if password == 'babaji':
        break
print('Access granted.')
```

Return to the Top

for Loops and the range() Function

```
print('My name is')
for i in range(5):
  print('Arslanmy Five Times ({})'.format(str(i)))
My name is
Arslan Five Times (0)
Arslan Five Times (1)
Arslan Five Times (2)
Arslan Five Times (3)
Arslan Five Times (4)
```

The *range()* function can also be called with three arguments. The first two arguments will be the start and stop values, and the third will be the step argument. The step is the amount that the variable is increased by after each iteration.

```
for i in range(0, 10, 2):
 print(i)
0
2
4
6
8
```

You can even use a negative number for the step argument to make the for loop count down instead of up.

```
for i in range(5, -1, -1):
  print(i)
5
```

```
4
3
2
1
0
```

For else statement

This allows to specify a statement to execute in case of the full loop has been executed. Only useful when a `break` condition can occur in the loop:

```
for i in [1, 2, 3, 4, 5]:
  if i == 3:
    break
else:
 print("only executed when no item of the list is equal to 3")
```

Return to the Top

Functions

```
def hello(name):
  print('Hello {}'.format(name))

hello('Arslan')
hello('babaji')
Hello Arslan
Hello babaji
```

Return to the Top

Return Values and return Statements

When creating a function using the def statement, you can specify what the return value should be with a return statement. A return statement consists of the following:

- The return keyword.
- The value or expression that the function should return.

```
import random
def getAnswer(answerNumber):
    if answerNumber == 1:
```

```
        return 'It is certain'
    elif answerNumber == 2:
        return 'It is decidedly so'
    elif answerNumber == 3:
        return 'Yes'
    elif answerNumber == 4:
        return 'Reply hazy try again'
    elif answerNumber == 5:
        return 'Ask again later'
    elif answerNumber == 6:
        return 'Concentrate and ask again'
    elif answerNumber == 7:
        return 'My reply is no'
    elif answerNumber == 8:
        return 'Outlook not so good'
    elif answerNumber == 9:
        return 'Very doubtful'

r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

Return to the Top

The None Value

```
spam = print('Hello!')
Hello!
```

```
spam is None
True
```

Note: never compare to `None` with the `==` operator. Always use `is`.

Return to the Top

Keyword Arguments and print()

```
print('Hello', end='')
print('World')
HelloWorld
```

```
print('cats', 'dogs', 'mice')
cats dogs mice
```

```
print('cats', 'dogs', 'mice', sep=',')
cats,dogs,mice
```

Return to the Top

Lists

```
spam = ['cat', 'bat', 'rat', 'elephant']

spam
['cat', 'bat', 'rat', 'elephant']
```

Return to the Top

Getting Individual Values in a List with Indexes

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[0]
'cat'
```

```
spam[1]
'bat'
```

```
spam[2]
'rat'
```

```
spam[3]
'elephant'
```

Return to the Top

Negative Indexes

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[-1]
'elephant'
```

```
spam[-3]
'bat'
```

```
'The {} is afraid of the {}.'.format(spam[-1], spam[-3])
'The elephant is afraid of the bat.'
```

Return to the Top

Getting Sublists with Slices

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[0:4]
['cat', 'bat', 'rat', 'elephant']
```

```
spam[1:3]
['bat', 'rat']
```

```
spam[0:-1]
['cat', 'bat', 'rat']
```

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[:2]
['cat', 'bat']
```

```
spam[1:]
['bat', 'rat', 'elephant']
```

Slicing the complete list will perform a copy:

```
spam2 = spam[:]
['cat', 'bat', 'rat', 'elephant']
spam.append('dog')
spam
['cat', 'bat', 'rat', 'elephant', 'dog']
spam2
['cat', 'bat', 'rat', 'elephant']
```

Return to the Top

Getting a List's Length with len()

```
spam = ['cat', 'dog', 'moose']
len(spam)
3
```

Changing Values in a List with Indexes

```
spam = ['cat', 'bat', 'rat', 'elephant']
spam[1] = 'aardvark'

spam
['cat', 'aardvark', 'rat', 'elephant']

spam[2] = spam[1]

spam
['cat', 'aardvark', 'aardvark', 'elephant']

spam[-1] = 12345

spam
['cat', 'aardvark', 'aardvark', 12345]
```

Return to the Top

List Concatenation and List Replication

```
[1, 2, 3] + ['A', 'B', 'C']
```

```
[1, 2, 3, 'A', 'B', 'C']

['X', 'Y', 'Z'] * 3

['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']

spam = [1, 2, 3]

spam = spam + ['A', 'B', 'C']

spam

[1, 2, 3, 'A', 'B', 'C']
```

Return to the Top

Removing Values from Lists with del Statements

```
spam = ['cat', 'bat', 'rat', 'elephant']
del spam[2]
spam
['cat', 'bat', 'elephant']
```

```
del spam[2]
spam
['cat', 'bat']
```

Return to the Top

Using for Loops with Lists

```
supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
for i, supply in enumerate(supplies):
  print('Index {} in supplies is: {}'.format(str(i), supply))
Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flame-throwers
Index 3 in supplies is: binders
```

Return to the Top

Looping Through Multiple Lists with zip()

```
>>> name = ['Pete', 'John', 'Elizabeth']
>>> age = [6, 23, 44]
>>> for n, a in zip(name, age):
>>>     print('{} is {} years old'.format(n, a))
Pete is 6 years old
John is 23 years old
Elizabeth is 44 years old
```

The in and not in Operators

```
'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
```

```
spam = ['hello', 'hi', 'howdy', 'heyas']
'cat' in spam
False
```

```
'howdy' not in spam
False
```

```
'cat' not in spam
True
```

Return to the Top

The Multiple Assignment Trick

The multiple assignment trick is a shortcut that lets you assign multiple variables with the values in a list in one line of code. So instead of doing this:

```
cat = ['fat', 'orange', 'loud']

size = cat[0]

color = cat[1]

disposition = cat[2]
```

You could type this line of code:

```
cat = ['fat', 'orange', 'loud']

size, color, disposition = cat
```

The multiple assignment trick can also be used to swap the values in two variables:

```
a, b = 'Alice', 'Bob'
a, b = b, a
print(a)
'Bob'
```

```
print(b)
'Alice'
```

Return to the Top

Augmented Assignment Operators

Operator	Equivalent
`spam += 1`	`spam = spam + 1`
`spam -= 1`	`spam = spam - 1`
`spam *= 1`	`spam = spam * 1`
`spam /= 1`	`spam = spam / 1`
`spam %= 1`	`spam = spam % 1`

Examples:

```
spam = 'Hello'
spam += ' world!'
spam
'Hello world!'

bacon = ['Zophie']
bacon *= 3
```

```
bacon
['Zophie', 'Zophie', 'Zophie']
```

Return to the Top

Finding a Value in a List with the index() Method

```
spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']

spam.index('Pooka')
1
```

Return to the Top

Adding Values to Lists with the append() and insert() Methods

append():

```
spam = ['cat', 'dog', 'bat']

spam.append('moose')

spam
['cat', 'dog', 'bat', 'moose']
```

insert():

```
spam = ['cat', 'dog', 'bat']

spam.insert(1, 'chicken')

spam
['cat', 'chicken', 'dog', 'bat']
```

Return to the Top

Removing Values from Lists with remove()

```
spam = ['cat', 'bat', 'rat', 'elephant']

spam.remove('bat')
```

```
spam
['cat', 'rat', 'elephant']
```

If the value appears multiple times in the list, only the first instance of the value will be removed.

Return to the Top

Removing Values from Lists with pop()

```
spam = ['cat', 'bat', 'rat', 'elephant']

spam.pop()
'elephant'

spam
['cat', 'bat', 'rat']

spam.pop(0)
'cat'

spam
['bat', 'rat']
```

Return to the Top

Sorting the Values in a List with the sort() Method

```
spam = [2, 5, 3.14, 1, -7]
spam.sort()
spam
[-7, 1, 2, 3.14, 5]
```

```
spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
spam.sort()
spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

You can also pass True for the reverse keyword argument to have sort() sort the values in reverse order:

```

```

```
spam.sort(reverse=True)
spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

If you need to sort the values in regular alphabetical order, pass str. lower for the key keyword argument in the sort() method call:

```
spam = ['a', 'z', 'A', 'Z']
spam.sort(key=str.lower)
spam
['a', 'A', 'z', 'Z']
```

You can use the built-in function `sorted` to return a new list:

```
spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
sorted(spam)
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

Return to the Top

Tuple Data Type

```
eggs = ('hello', 42, 0.5)
eggs[0]
'hello'
```

```
eggs[1:3]
(42, 0.5)
```

```
len(eggs)
3
```

The main way that tuples are different from lists is that tuples, like strings, are immutable.

Return to the Top

Converting Types with the list() and tuple() Functions

```
tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
```

```
list(('cat', 'dog', 5))
['cat', 'dog', 5]
```

```
list('hello')
['h', 'e', 'l', 'l', 'o']
```

Return to the Top

Dictionaries and Structuring Data

Example Dictionary:

```
myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
```

Return to the Top

The keys(), values(), and items() Methods

values():

```
spam = {'color': 'red', 'age': 42}
for v in spam.values():
  print(v)
red
42
```

keys():

```
for k in spam.keys():
  print(k)
color
age
```

items():

```
for i in spam.items():
  print(i)
('color', 'red')
('age', 42)
```

Using the keys(), values(), and items() methods, a for loop can iterate over the keys, values, or key-value pairs in a dictionary, respectively.

```
 spam = {'color': 'red', 'age': 42}

for k, v in spam.items():
  print('Key: {} Value: {}'.format(k, str(v)))
Key: age Value: 42
Key: color Value: red
```

Return to the Top

Checking Whether a Key or Value Exists in a Dictionary

```
spam = {'name': 'Zophie', 'age': 7}
```

```
'name' in spam.keys()
True
```

```
'Zophie' in spam.values()
True
```

```
# You can omit the call to keys() when checking for a key
'color' in spam
False
```

```
'color' not in spam
True
```

Return to the Top

The get() Method

Get has two parameters: key and default value if the key did not exist

```
picnic_items = {'apples': 5, 'cups': 2}

'I am bringing {} cups.'.format(str(picnic_items.get('cups', 0)))
'I am bringing 2 cups.'
```

```
'I am bringing {} eggs.'.format(str(picnic_items.get('eggs', 0)))
'I am bringing 0 eggs.'
```

Return to the Top

The setdefault() Method

Let's consider this code:

```
spam = {'name': 'Pooka', 'age': 5}

if 'color' not in spam:
    spam['color'] = 'black'
```

Using `setdefault` we could write the same code more succinctly:

```
spam = {'name': 'Pooka', 'age': 5}
spam.setdefault('color', 'black')
'black'
```

```
spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

```
spam.setdefault('color', 'white')
'black'
```

```
spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

Return to the Top

Merge two dictionaries

```
x = {'a': 1, 'b': 2}
y = {'b': 3, 'c': 4}
z = {**x, **y}
z
{'c': 4, 'a': 1, 'b': 3}
```

sets

> A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Initializing a set

There are two ways to create sets: using curly braces `{}` and the built-in function `set()`

```
s = {1, 2, 3}
s = set([1, 2, 3])
```

When creating an empty set, be sure to not use the curly braces `{}` or you will get an empty dictionary instead.

```
s = {}
type(s)
<class 'dict'>
```

sets: unordered collections of unique elements

A set automatically remove all the duplicate values.

```
s = {1, 2, 3, 2, 3, 4}
s
{1, 2, 3, 4}
```

And as an unordered data type, they can't be indexed.

```
s = {1, 2, 3}
s[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

set add() and update()

Using the `add()` method we can add a single element to the set.

```
s = {1, 2, 3}
s.add(4)
s
{1, 2, 3, 4}
```

And with `update()`, multiple ones .

```
s = {1, 2, 3}
s.update([2, 3, 4, 5, 6])
s
{1, 2, 3, 4, 5, 6}  # remember, sets automatically remove duplicates
```

set remove() and discard()

Both methods will remove an element from the set, but `remove()` will raise a `key error` if the value doesn't exist.

```
s = {1, 2, 3}
s.remove(3)
s
{1, 2}
```

```
s.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

`discard()` won't raise any errors.

```
s = {1, 2, 3}
s.discard(3)
s
{1, 2}
s.discard(3)
```

set union()

`union()` or `|` will create a new set that contains all the elements from the sets provided.

```
s1 = {1, 2, 3}
s2 = {3, 4, 5}
s1.union(s2)  # or 's1 | s2'
{1, 2, 3, 4, 5}
```

set intersection

`intersection` or `&` will return a set containing only the elements that are common to all of them.

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s3 = {3, 4, 5}
s1.intersection(s2, s3)  # or 's1 & s2 & s3'
{3}
```

set difference

`difference` or `-` will return only the elements that are unique to the first set (invoked set).

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s1.difference(s2)  # or 's1 - s2'
{1}
```

```
s2.difference(s1) # or 's2 - s1'
{4}
```

set symetric_difference

`symetric_difference` or `^` will return all the elements that are not common between them.

```
s1 = {1, 2, 3}
s2 = {2, 3, 4}
s1.symmetric_difference(s2)  # or 's1 ^ s2'
{1, 4}
```

Return to the Top

Comprehensions

List comprehension

```
a = [1, 3, 5, 7, 9, 11]

[i - 1 for i in a]
[0, 2, 4, 6, 8, 10]
```

Set comprehension

```
b = {"abc", "def"}
{s.upper() for s in b}
{"ABC", "DEF"}
```

Dict comprehension

```
c = {'name': 'Pooka', 'age': 5}
{k, v in c.items()}
```

Return to the Top

Chapter 2 Plots

In this chapter I have divided the plotting methods into 3 categories depending upon the importance and level.

Return to the Top

Introduction

Data Visualization: It is a way to express your data in a visual context so that patterns, correlations, trends between the data can be easily understood. Data Visualization helps in finding hidden insights by providing skin to your raw data (skeleton).

In this chapter, we will be using multiple datasets to show exactly how things work. The base dataset will be the iris dataset which we will import from sklearn. We will create the rest of the dataset according to need.

Let's import all the libraries which are required for doing

```
# Library Import
import math,os
import random
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as stat
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA
```

Reading the data (Main focused dataset)

```
# Reading the Dataset
iris = pd.read_csv('IRIS.csv')
iris_feat = iris.iloc[:,:-1]
iris_species = iris.iloc[:,-1]
```

Section - 1

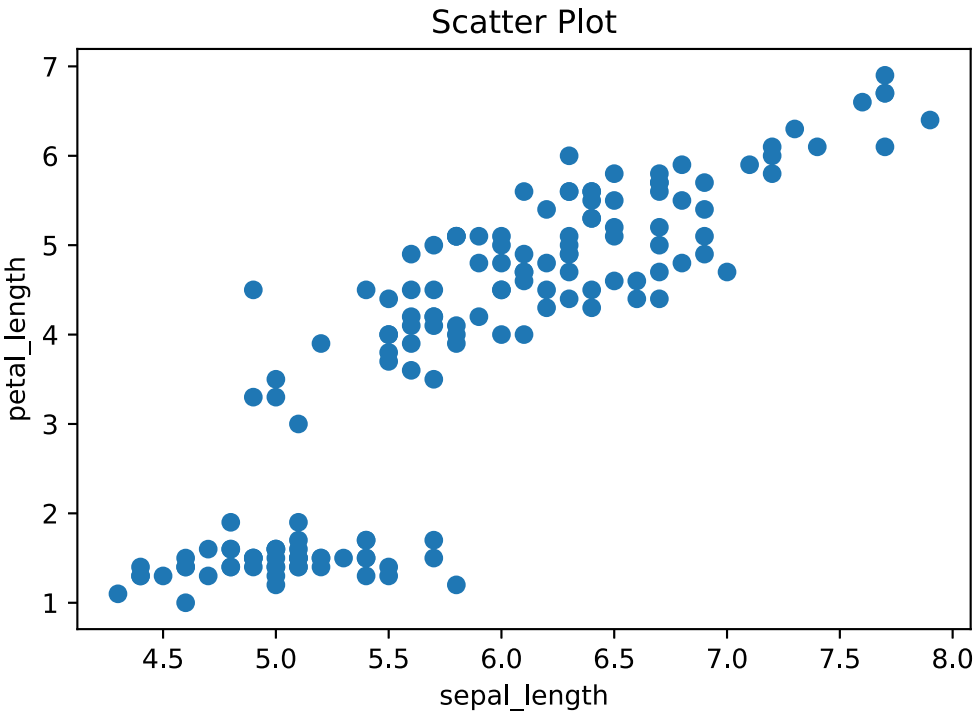
Return to the Top

Scatter Plot

These are the charts/plots that are used to observe and display relationships between variables using Cartesian Coordinates. The values (x: first variable , y: second variable) of the variables are represented by dots. Scatter plots are also known as scattergrams, scatter graphs, scatter charts , or scatter diagrams. It is best suited for situations where the dependent variable can have multiple values for the independent variable.

Scatter Plot with Matplotlib

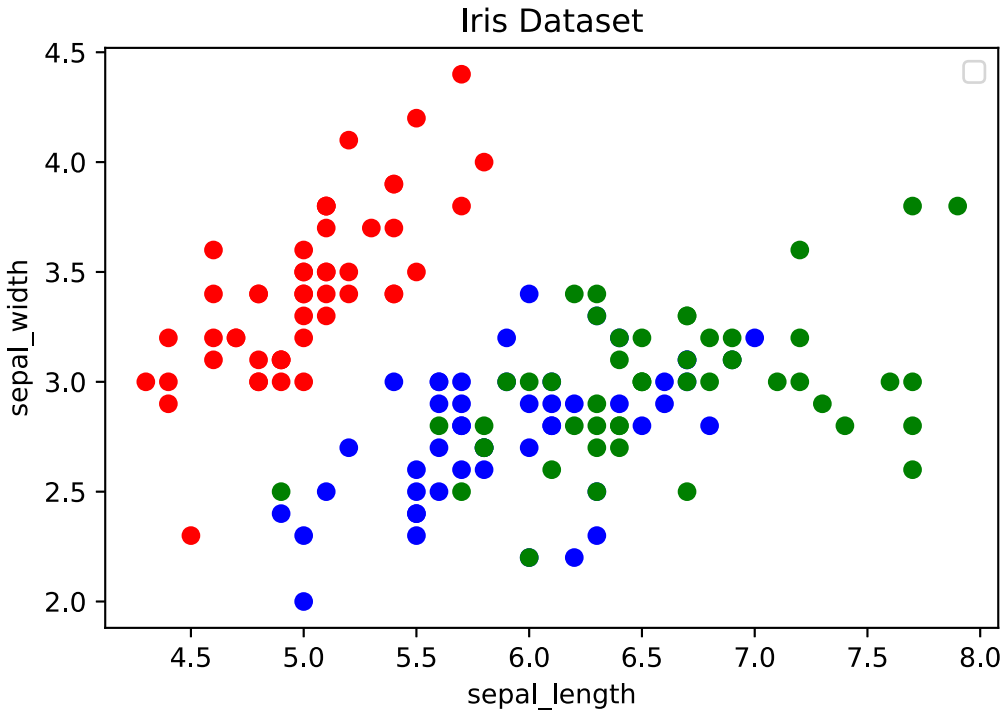
```
plt.scatter(x = iris_feat['sepal_length'],y = [iris_feat['petal_length']],alpha=1)

plt.title('Scatter Plot')
plt.xlabel('sepal_length')
plt.ylabel('petal_length')
#Show the plot
plt.show()
```



Scatter plot for Multivariate Analysis

```
# Multivariate Scatter Plot

colors = {'Iris-setosa':'r', 'Iris-virginica':'g', 'Iris-versicolor':'b'}

# create a figure and axis
fig, ax = plt.subplots()

# plot each data-point
for i in range(len(iris_feat['sepal_length'])):
    ax.scatter(iris_feat['sepal_length'][i], iris_feat['sepal_width']
[i],color=colors[iris_species[i]])

# set a title and labels
ax.set_title('Iris Dataset')
```

```
ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')

ax.legend()
plt.show()
```



Two common issues with the use of scatter plots are – overplotting and the interpretation of causation as correlation.
Overplotting occurs when there are too many data points to plot, which results in the overlapping of different data points and make it hard to identify any relationship between points
Correlation does not mean that the changes observed in one variable are responsible for changes in another variable. So any conclusions made by correlation should be treated carefully.

Return to the Top

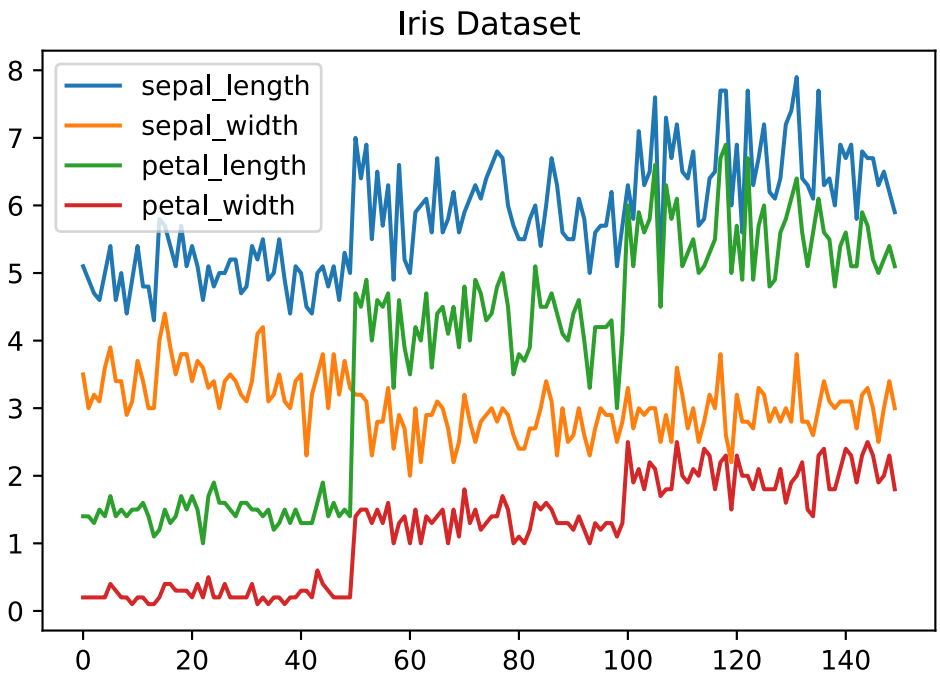
Line Plot

Line plots is a graph that is used for the representation of continuous data points on a number line. Line plots are created by first plotting data points on the Cartesian plane then joining those points with a number line. Line plots can help display data points for both single variable analysis as well as multiple variable analysis.

Line Plot in Matplotlib

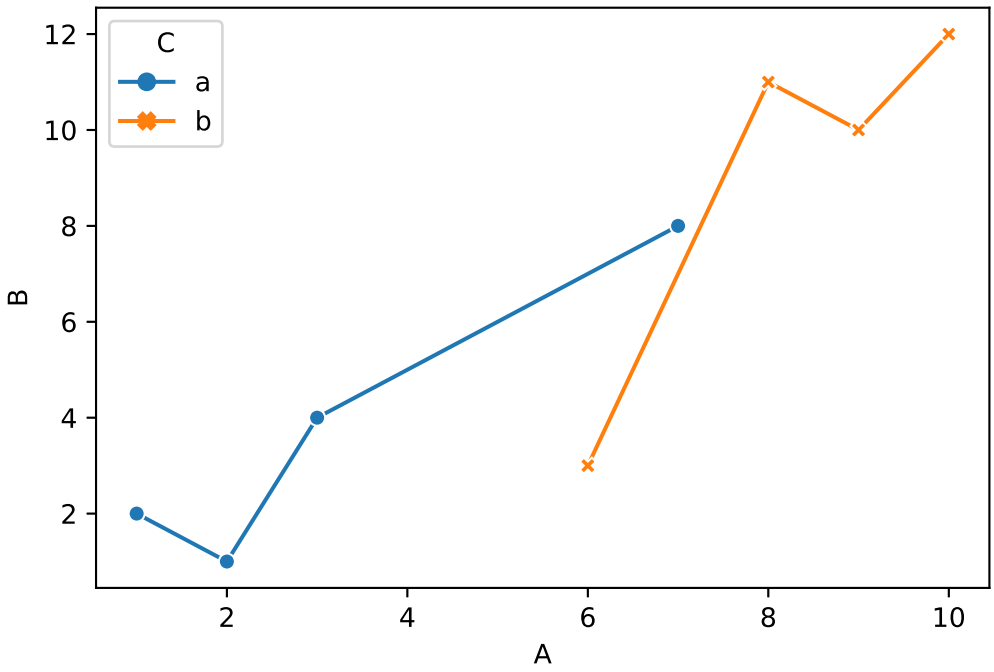
```
# get columns to plot
columns = iris_feat.columns
# create x data
```

```
x_data = range(0, iris.shape[0])
# create figure and axis
fig, ax = plt.subplots()
# plot each column
for column in columns:
    ax.plot(x_data, iris[column], label=column)
# set title and legend
ax.set_title('Iris Dataset')
ax.legend()
```



Seaborn Implementation Line Graphs

```
# Seaborn Implementation

df = pd.DataFrame({
    'A': [1,3,2,7,9,6,8,10],
    'B': [2,4,1,8,10,3,11,12],
    'C': ['a','a','a','a','b','b','b','b']
})

sns.lineplot(
    data=df,
    x="A", y="B", hue="C",style="C",
    markers=True, dashes=False
```

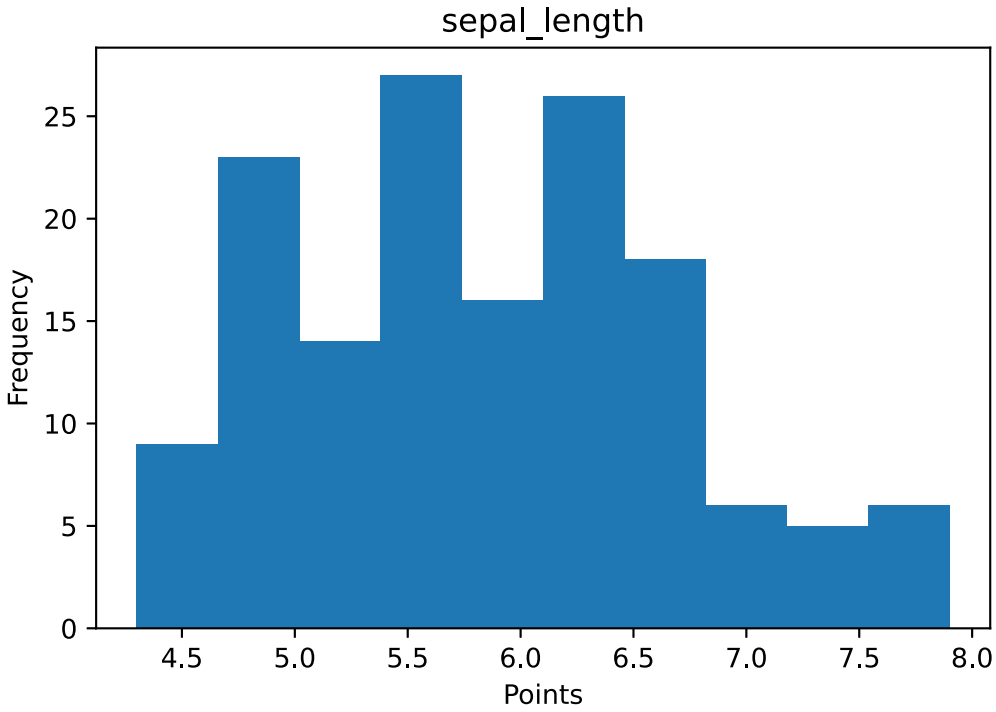


Return to the Top

Histograms

Histograms are used to represent the frequency distribution of continuous variables. The width of the histogram represents interval and the length represents frequency. To create a histogram you need to create bins of the interval which are not overlapping. Histogram allows the inspection of data for its underlying distribution, outliers, skewness.

```
**Histograms in Matplotlib**

# normal
fig, ax = plt.subplots()
# plot histogram
ax.hist(iris_feat['sepal_length'])
# set title and labels
ax.set_title('sepal_length')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
```



The values with longer plots signify that more values are concentrated there. Histograms help in understanding the frequency distribution of the overall data points for that feature.

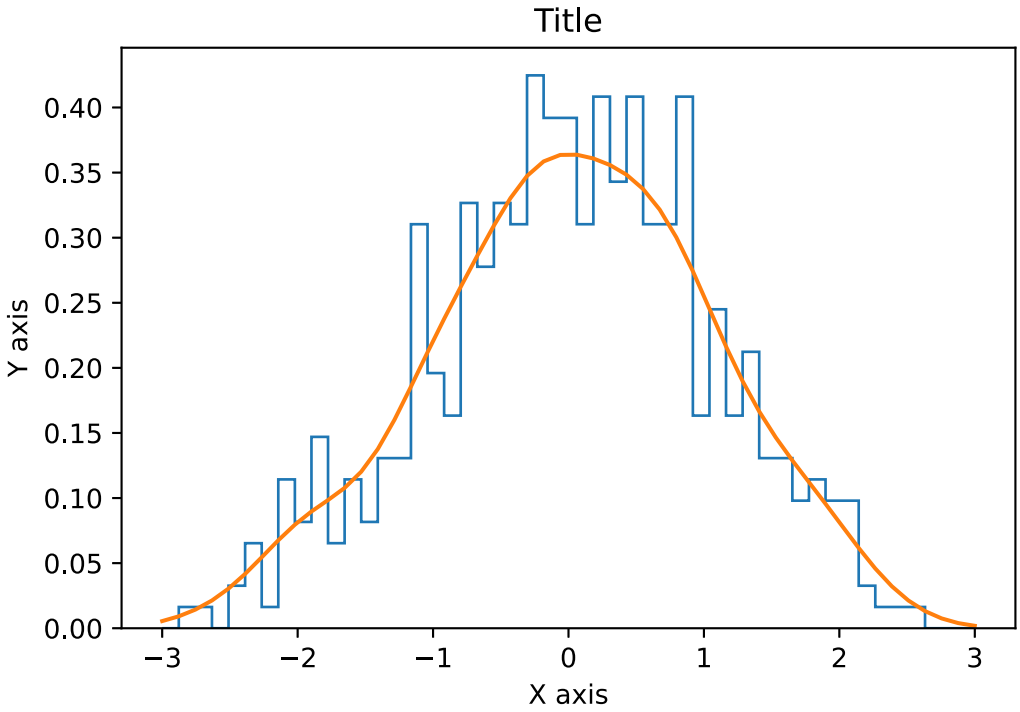
Return to the Top

LINE HISTOGRAMS (MODIFICATION TO HISTOGRAMS)

Line histograms are the modification to the standard histogram to understand and represent the distribution of a single feature with different data points. Line histograms have a density curve passing through the histogram.

Matplotlib Implementation

```
# line histogram

#Creating the dataset
test_data = np.random.normal(0, 1, (500, ))
density = stat.gaussian_kde(test_data)
#Creating the line histogram
n, x, _ = plt.hist(test_data, bins=np.linspace(-3, 3, 50), histtype=u'step',
density=True)
plt.plot(x, density(x))

plt.title('Title')
plt.xlabel('X axis')
plt.ylabel('Y axis')
#Show the plot
plt.show()
```



Normal Histogram is a bell-shaped histogram with most of the frequency counts focused in the middle with diminishing tails. Line with orange color passing through histogram represents Gaussian distribution for data points.

Return to the Top

Bar Chart

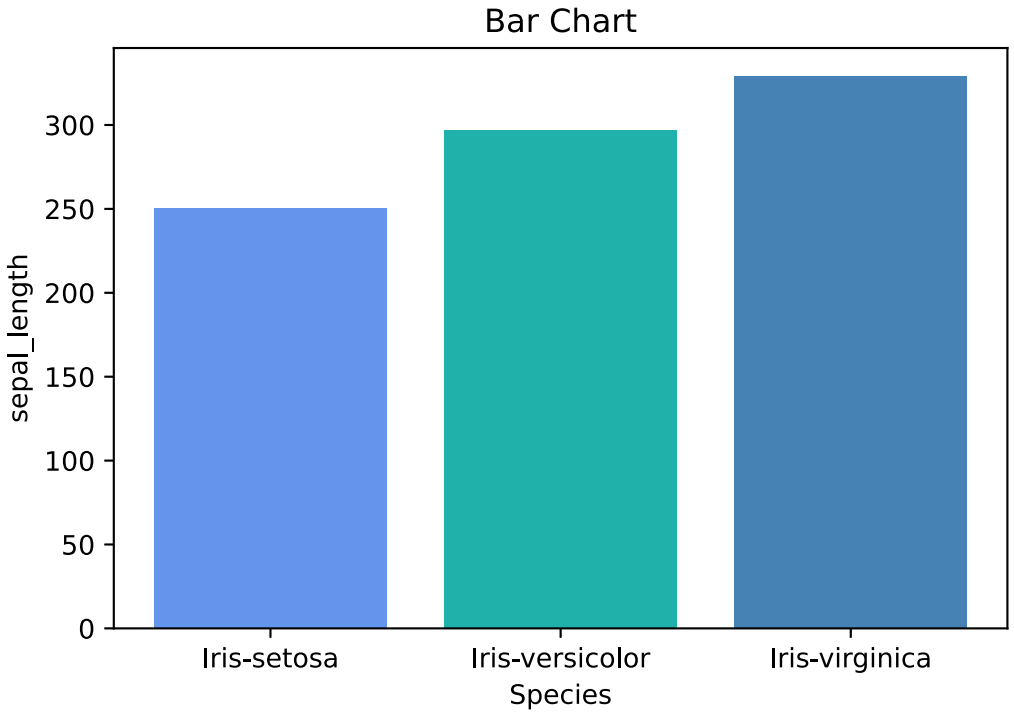
Bar charts are best suited for the visualization of categorical data because they allow you to easily see the difference between feature values by measuring the size(length) of the bars. There are 2 types of bar charts depending upon their orientation (i.e. vertical or horizontal). Moreover, there are 3 types of bar charts based on their representation that is shown below.

NORMAL BAR CHART

Matplotlib Implementation

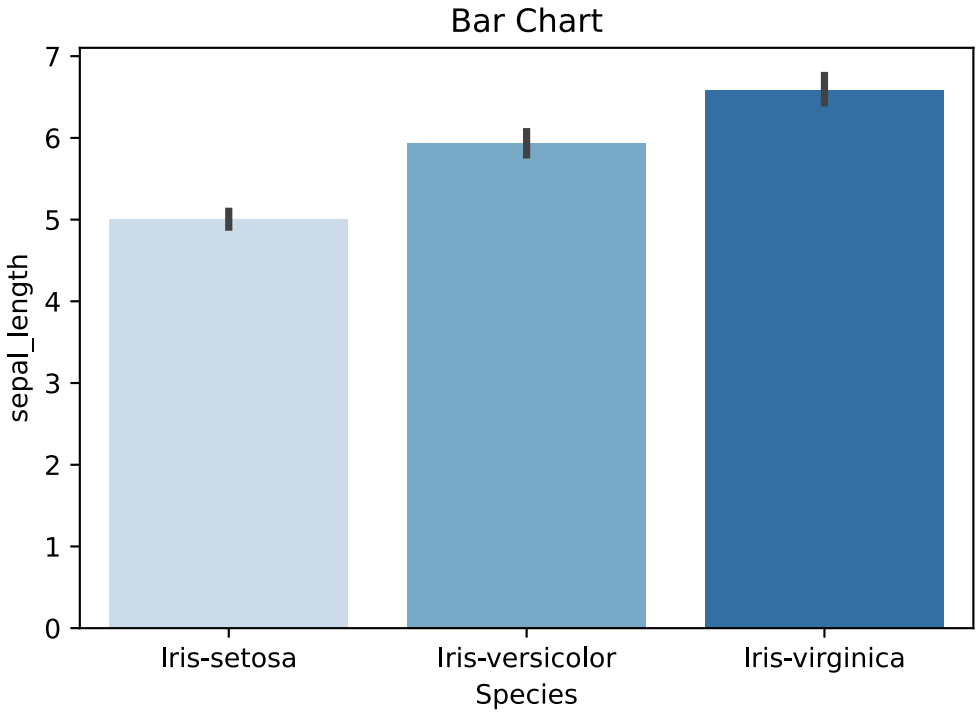
```
# matplotlib implementation

df = iris.groupby('species')['sepal_length'].sum().to_frame().reset_index()

#Creating the bar chart
plt.bar(df['species'],df['sepal_length'],color =
['cornflowerblue','lightseagreen','steelblue'])

plt.title('Bar Chart')
plt.xlabel('Species')
```

```
plt.ylabel('sepal_length')

#Show the plot
plt.show()
```



Sns Implementation

```
sns.barplot(x = 'species',y = 'sepal_length',data =iris,palette = "Blues")

plt.title('Bar Chart')
plt.xlabel('Species')
plt.ylabel('sepal_length')
# Show the plot
plt.show()
```



With the above image, we can clearly see the difference in the sum of sepal_length for each species of a leaf.

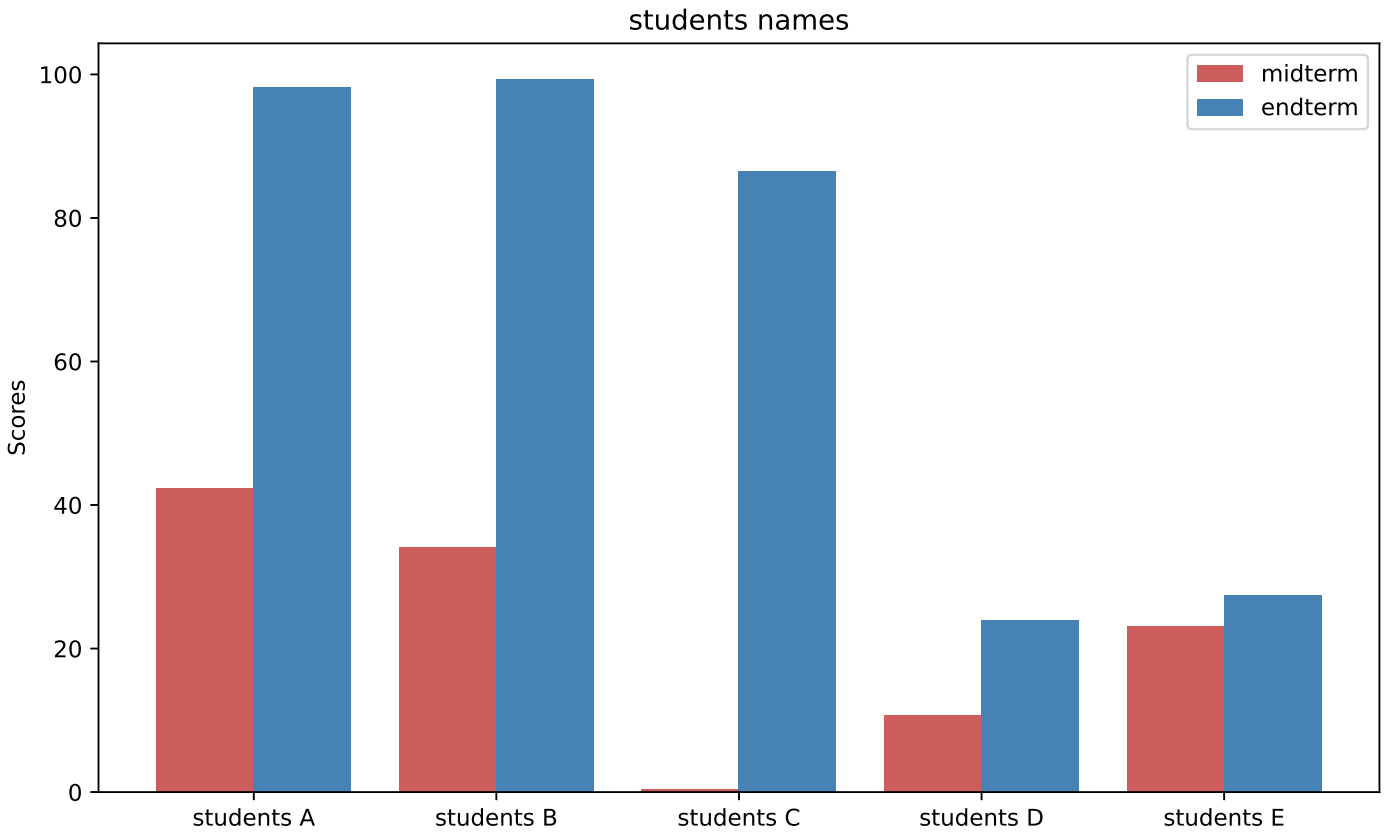
GROUPED BAR CHART

These bar charts allows us to compare multiple categorical features. Lets see an example.

Matplotlib Implementation

```
# Grouped Bar chart

mid_term_marks=[ random.uniform(0,50) for i in range(5)]
end_term_marks=[ random.uniform(0,100) for i in range(5)]

fig=plt.figure(figsize=(10,6))

students=['students A','students B','students C','students D','students E']
x_pos_mid=list(range(1,6))
x_pos_end=[ i+0.4 for i in x_pos_mid]

graph_mid_term=plt.bar(x_pos_mid,
mid_term_marks,color='indianred',label='midterm',width=0.4)
graph_end_term=plt.bar(x_pos_end,
end_term_marks,color='steelblue',label='endterm',width=0.4)

plt.xticks([i+0.2 for i in x_pos_mid],students)
plt.title('students names')
plt.ylabel('Scores')
```

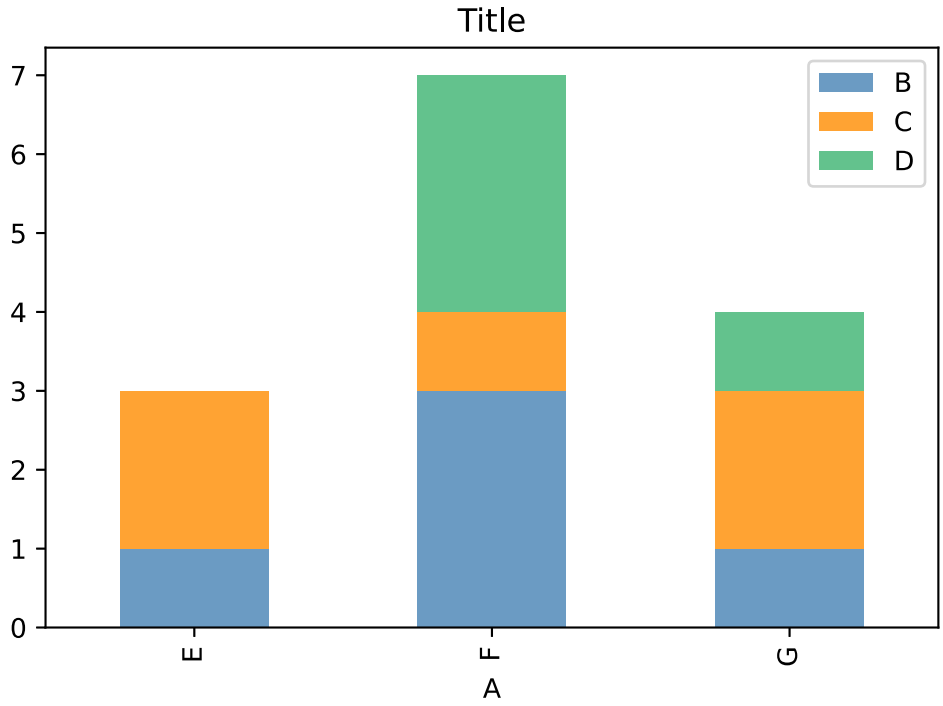
```
plt.legend()
plt.show()
```



STACKED BAR CHART

Matplotlib Implementation

```
# Stacked Bar chart

df = pd.DataFrame(columns=["A","B", "C","D"],
                  data=[["E",1,2,0],
                        ["F",3,1,3],
                        ["G",1,2,1]])

df.plot.bar(x='A', y=["B", "C","D"],  stacked=True, alpha=0.8 ,color=
['steelblue','darkorange' ,'mediumseagreen'])

plt.title('Title')

#Show the plot
plt.show()
```



Return to the Top

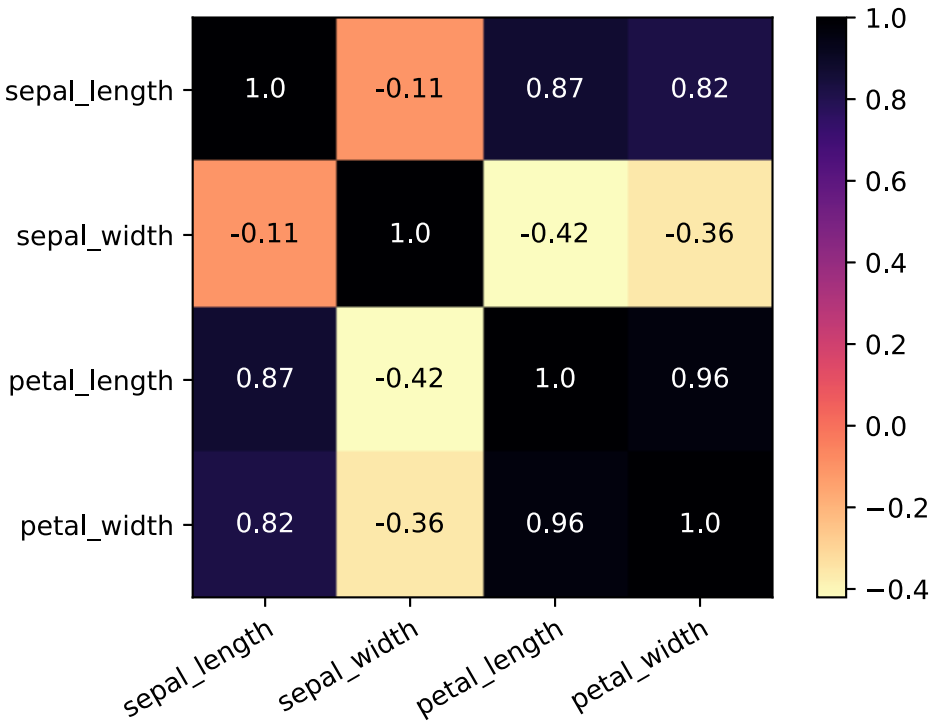
Heatmap

Heatmaps are the graphical representation of data where each value is represented in a matrix with different color coding. Mostly heatmaps are used to find correlations between various data columns in a dataset.

Matplotlib Implementation

```
corr = iris.corr()

fig, ax = plt.subplots()
img = ax.imshow(corr.values,cmap = "magma_r")

# set labels
ax.set_xticks(np.arange(len(corr.columns)))
ax.set_yticks(np.arange(len(corr.columns)))
ax.set_xticklabels(corr.columns)
ax.set_yticklabels(corr.columns)
cbar = ax.figure.colorbar(img, ax=ax ,cmap='')
plt.setp(ax.get_xticklabels(), rotation=30, ha="right",
         rotation_mode="anchor")

# text annotations.
for i in range(len(corr.columns)):
    for j in range(len(corr.columns)):
        if corr.iloc[i, j]<0:
            text = ax.text(j, i, np.around(corr.iloc[i, j], decimals=2),
```

```
                    ha="center", va="center", color="black")
        else:
            text = ax.text(j, i, np.around(corr.iloc[i, j], decimals=2),
                    ha="center", va="center", color="white")
```



You can see in the above heatmap that values high value that is close to 1 are represented with dark color and values with smaller value are represented with light color. This observation will always be the same for the heatmap. Dark values will always be greater than light-colored values.

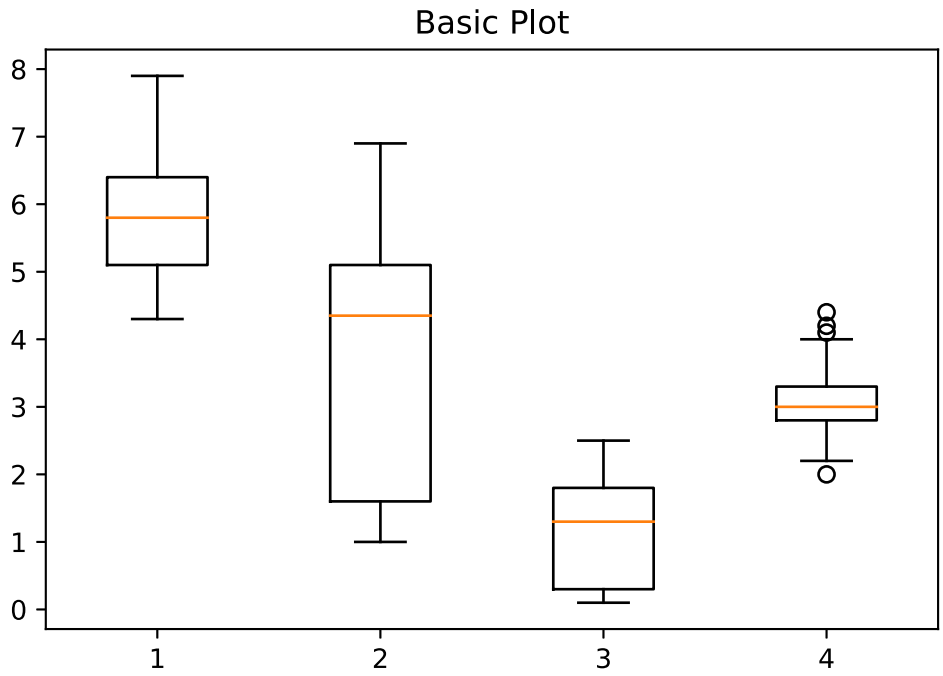
Return to the Top

Section - 2

Return to the Top

Box Plot

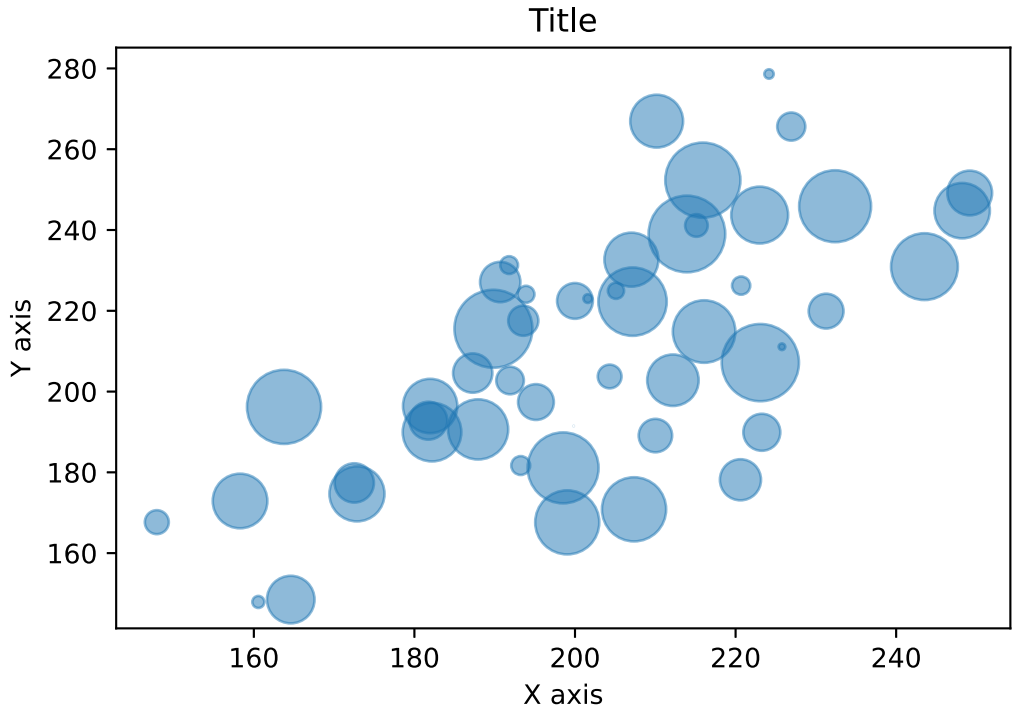
```
sepal_length = iris_feat['sepal_length']
petal_length = iris_feat['petal_length']
petal_width = iris_feat['petal_width']
sepal_width = iris_feat['sepal_width']

data = [sepal_length , petal_length , petal_width , sepal_width]
fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot')
ax1.boxplot(data)
plt.show()
```



Return to the Top

Bubble Plot

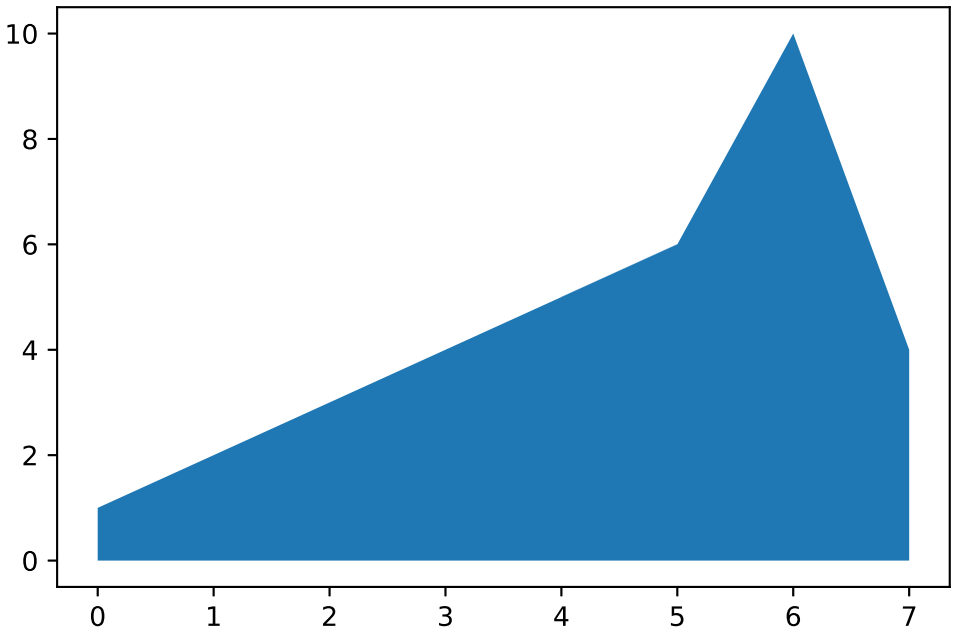
```
N = 50
x = np.random.normal(200, 20, N)
y = x + np.random.normal(5, 25, N)
area = (30 * np.random.rand(N))**2
df = pd.DataFrame({
    'X': x,
    'Y': y,
    "size":area})
plt.scatter('X', 'Y', s='size',alpha=0.5, data=df)

plt.title('Title')
plt.xlabel('X axis')
plt.ylabel('Y axis')

plt.show()
```



Return to the Top

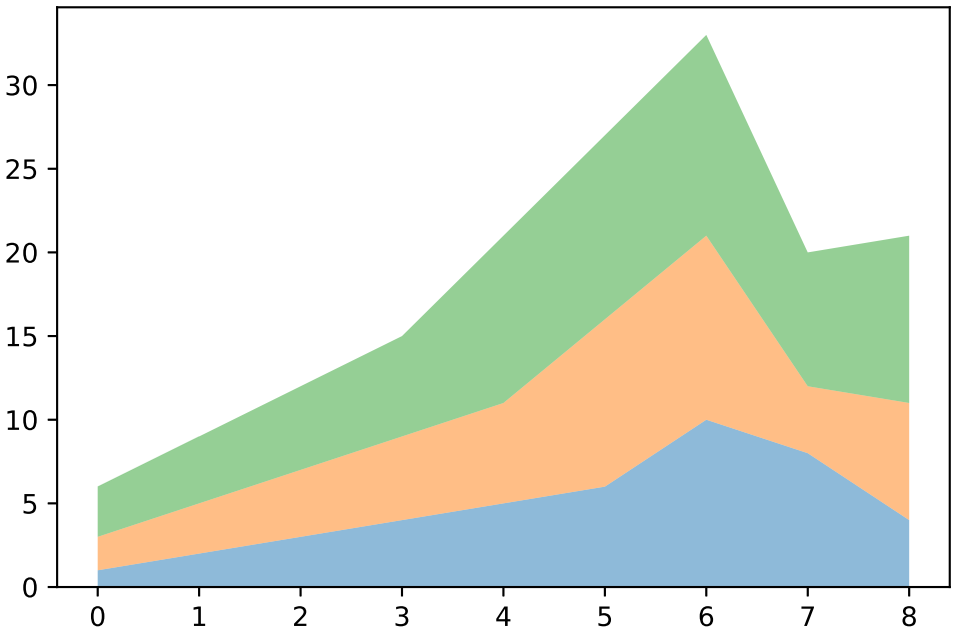
Area Plots

```
# Standard Area Plots
# Making some temporary data
y = [1,2,3,4,5,6,10,4]
x = list(range(len(y)))
#Creating the area chart
plt.fill_between(x, y)

#Show the plot
plt.show()
```

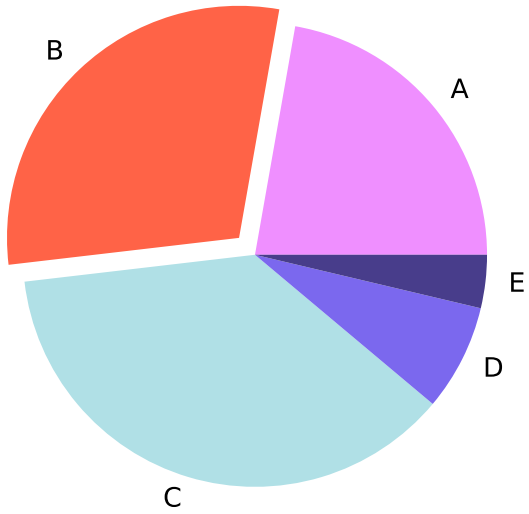


Stacked Area Plot

```
# Stacked Area Plot

y = [[1,2,3,4,5,6,10,8,4] , [2,3,4,5,6,10,11,4,7] , [3,4,5,6,10,11,12,8,10]]
x = list(range(len(y[0])))

ax = plt.gca()
ax.stackplot(x, y, labels=['A','B','c'],alpha=0.5)
plt.show()
```



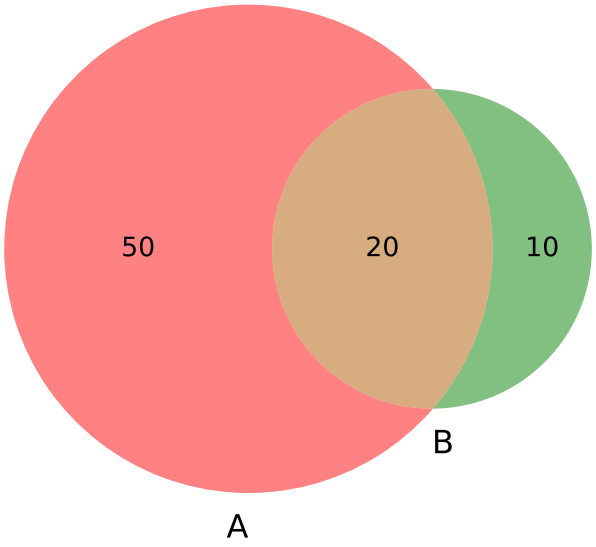
```
### Pie Chart

#Creating the dataset
students = ['A','B','C','D','E']
scores = [30,40,50,10,5]
#Creating the pie chart
plt.pie(scores, explode=[0,0.1,0,0,0], labels = students,colors =
['#EF8FFF','#ff6347','#B0E0E6','#7B68EE','#483D8B'])

#Show the plot
plt.show()
```



Return to the Top

Venn Diagram

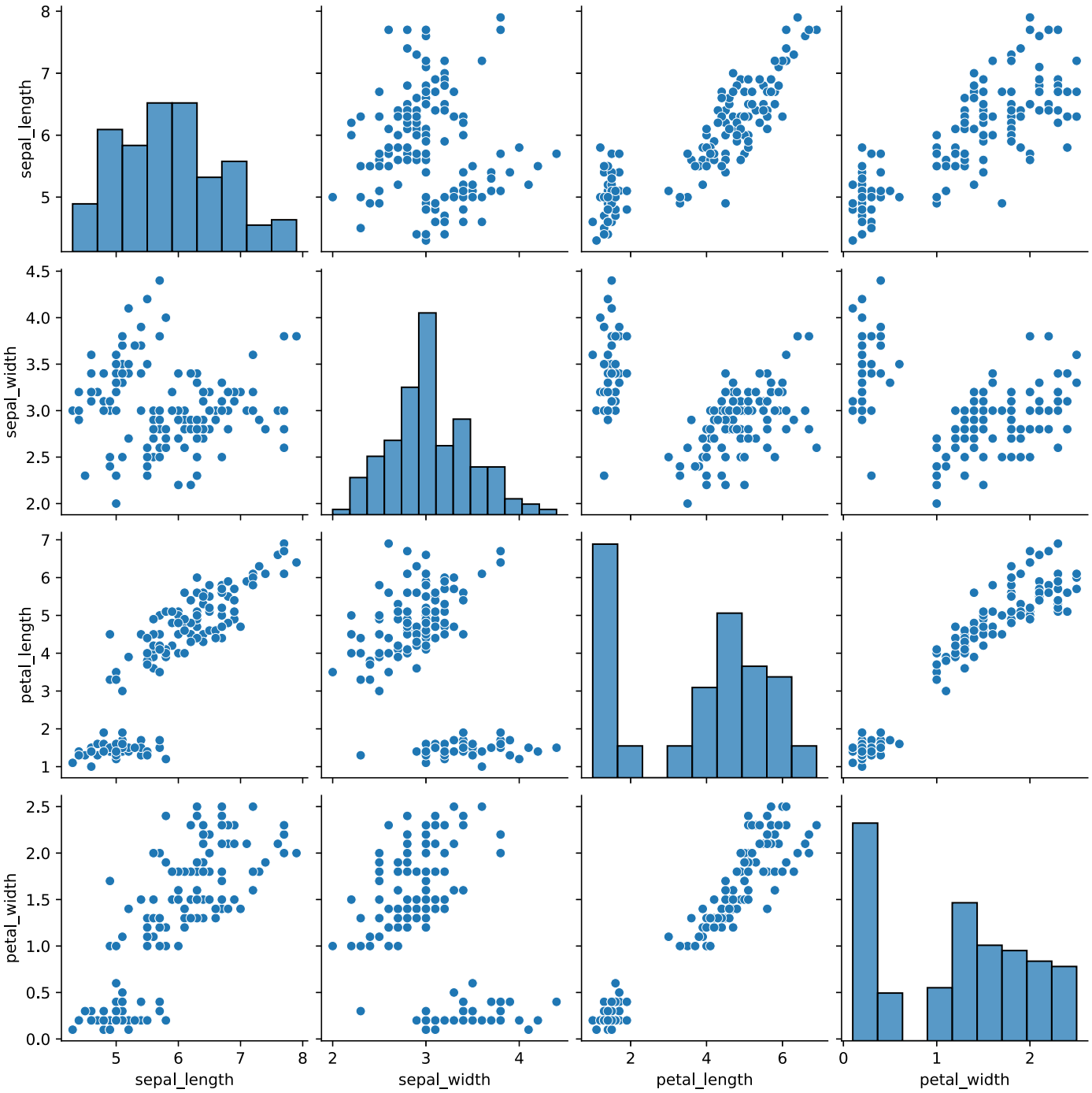
```
from matplotlib_venn import venn2 ,venn3
#Making venn  diagram

# Venn Diagram with 2 groups
venn2(subsets = (50, 10, 20), set_labels = ('A', 'B'),alpha=0.5)
plt.show()
```



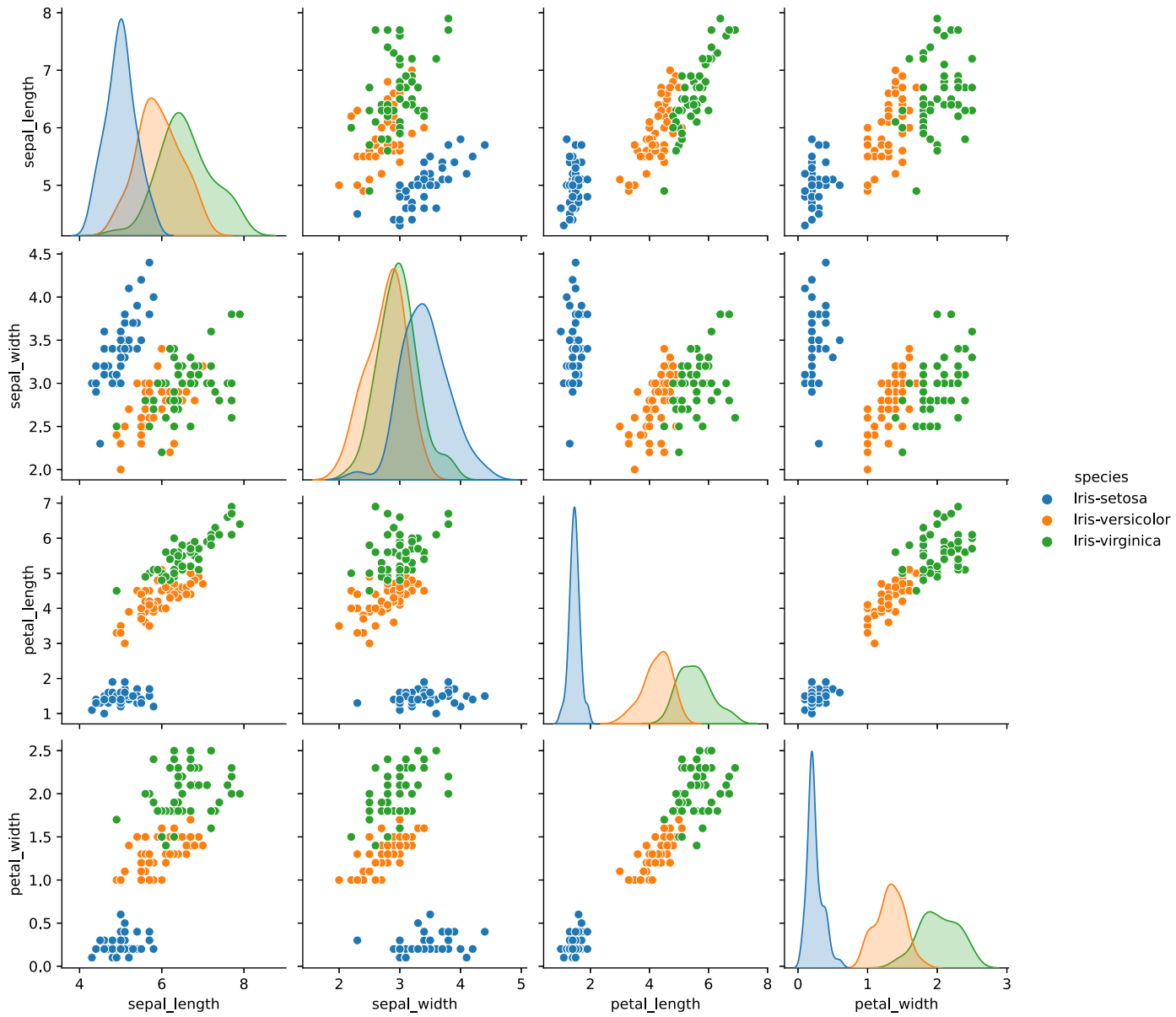
Return to the Top

Pair Plot

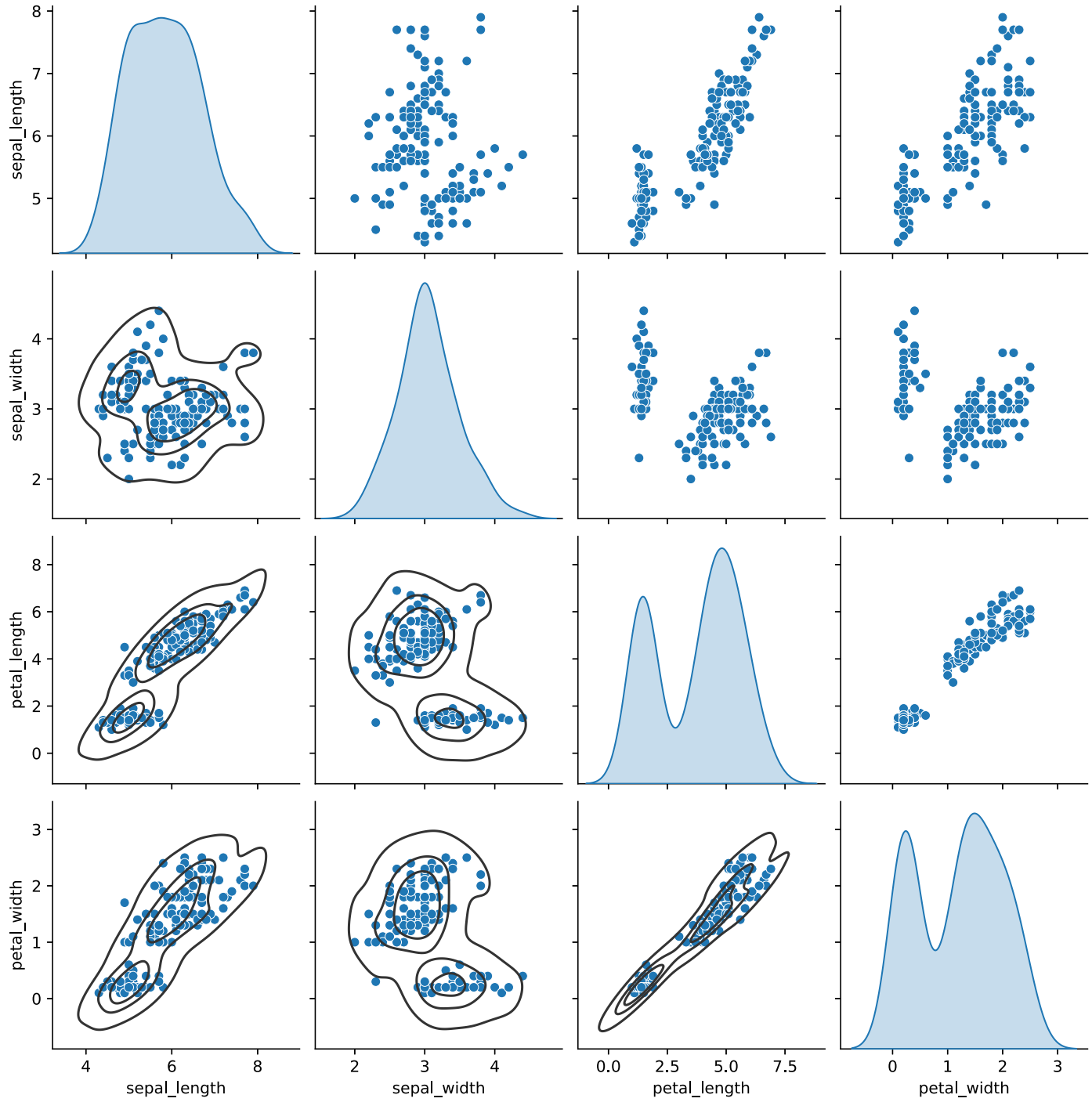
```
# type-1
sns.pairplot(iris)
```




```
# type-2
sns.pairplot(iris, hue="species")
```



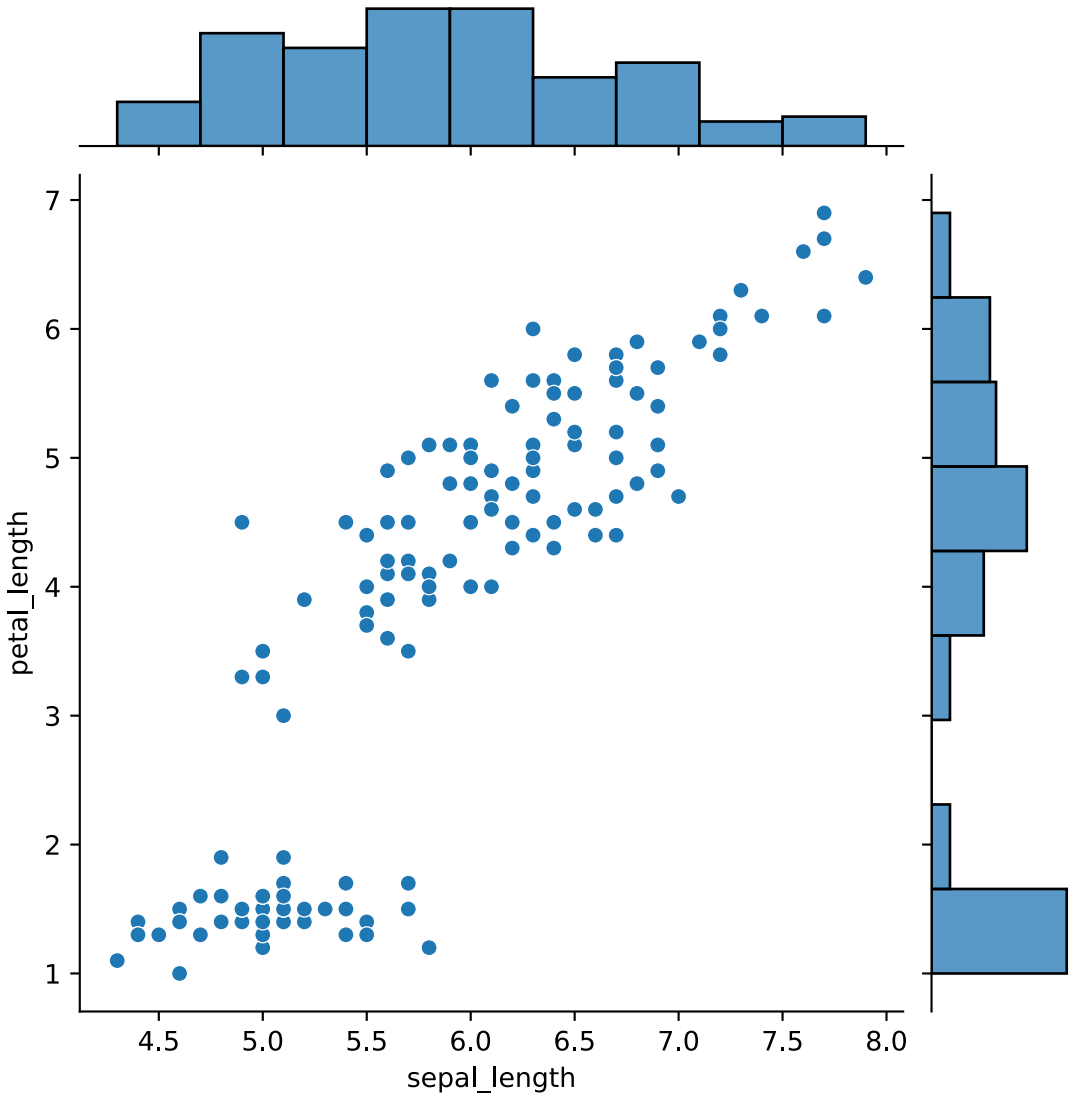

```
# type-3
g = sns.pairplot(iris, diag_kind="kde")
g.map_lower(sns.kdeplot, levels=4, color=".2")
```



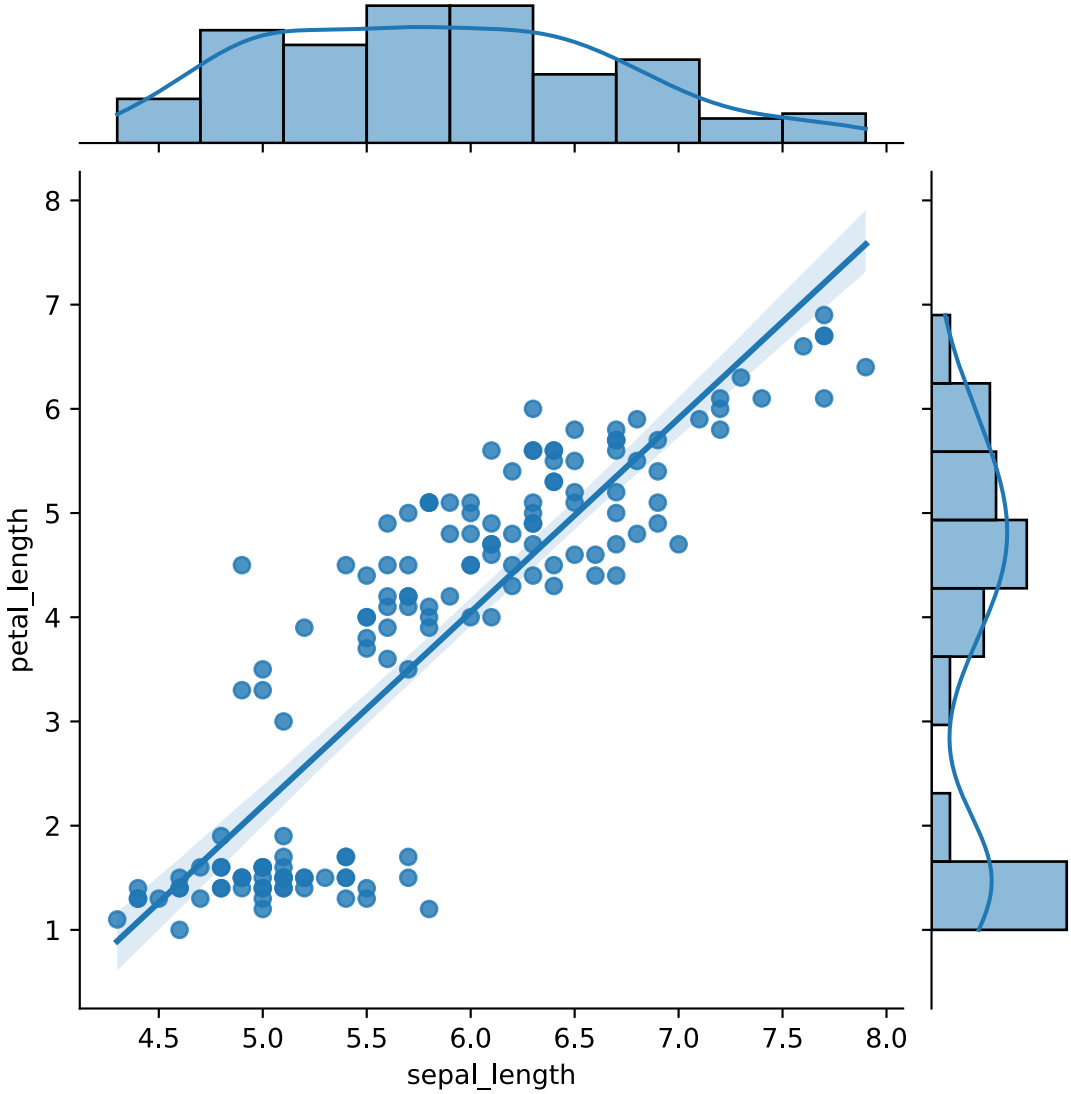
Return to the Top

Joint Plot

```
# Type -1
sns.jointplot(data=iris_feat, x="sepal_length", y="petal_length")
```




```
# type-2
sns.jointplot(data=iris, x="sepal_length", y="petal_length" , kind="reg")
```



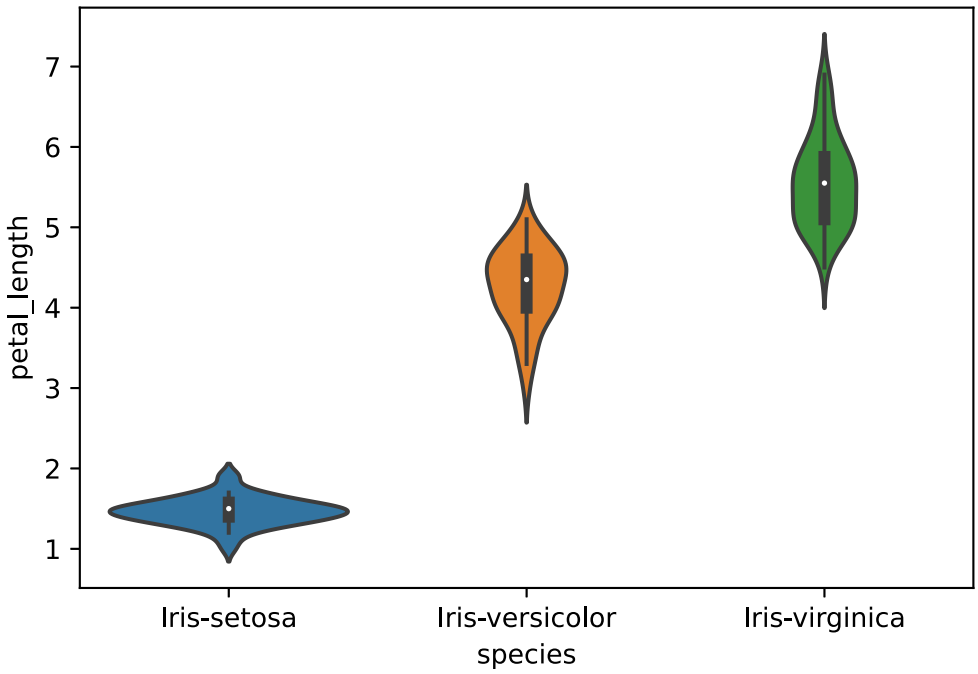
Return to the Top

Section - 3

Return to the Top

Violin Plots

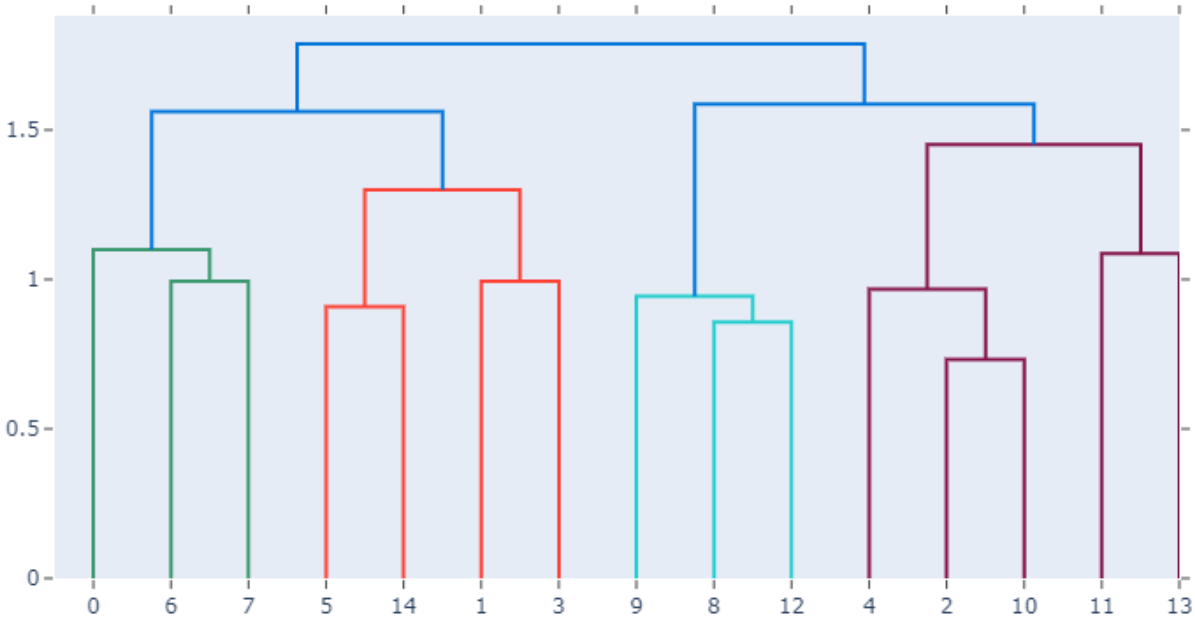
```
sns.violinplot(x="species", y="petal_length", data=iris, size=6)
```



Return to the Top

Dendrograms

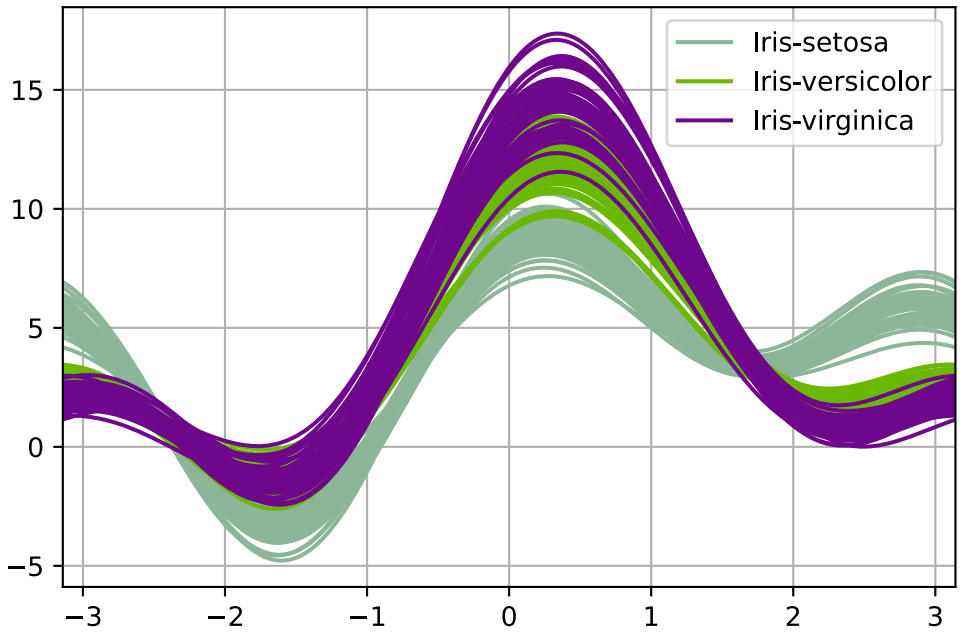
```
import plotly.figure_factory as ff

X = np.random.rand(15, 10)
fig = ff.create_dendrogram(X, color_threshold=1.5)
fig.update_layout(width=800, height=500)
fig.show()
```



Return to the Top

Andrew Curves

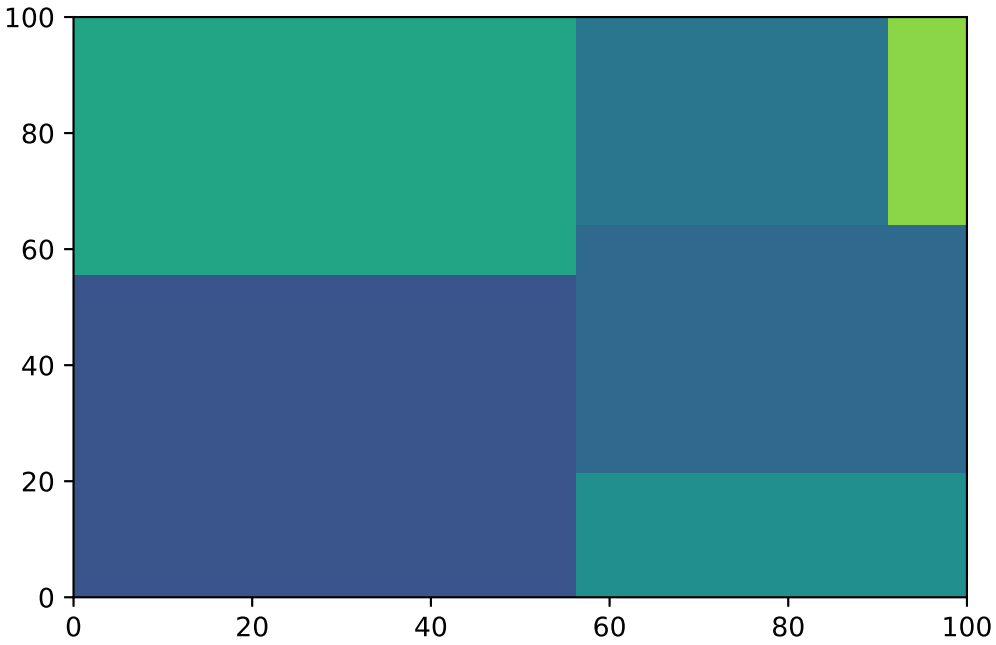
```
from pandas.plotting import andrews_curves
andrews_curves(iris, "species")
```



Return to the Top

Tree Maps

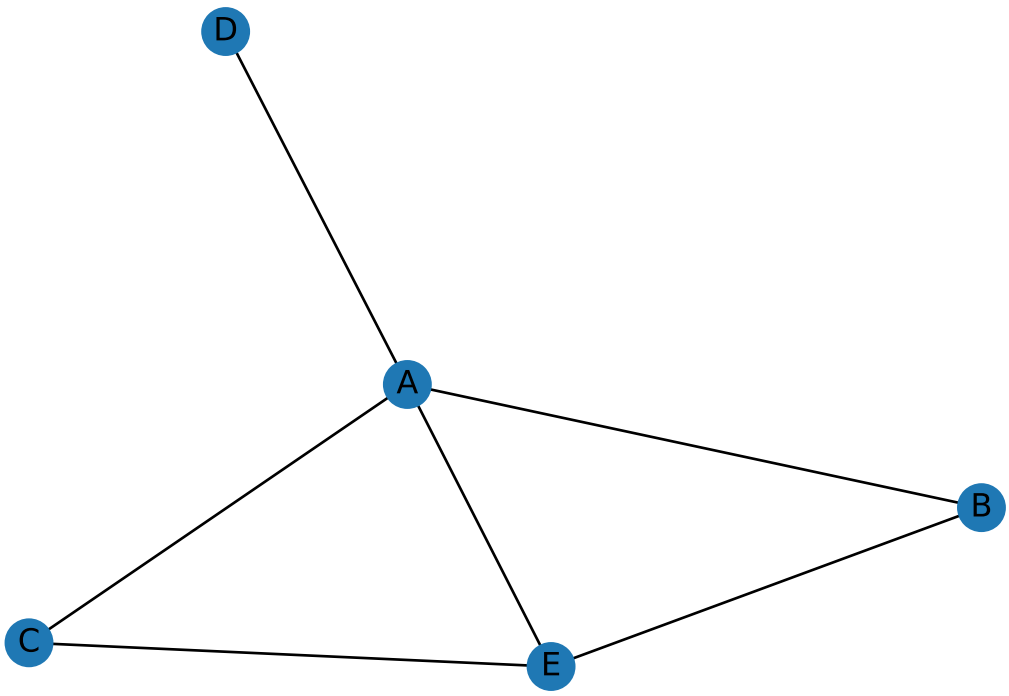
```
import squarify
sizes = [50, 40, 15, 30, 20,5]
squarify.plot(sizes)
# Show the plot
plt.show()
```



Return to the Top

Network Charts

```
import networkx as nx

# Build a dataframe with 4 connections
df = pd.DataFrame({ 'from':['A', 'B', 'C','A' ,'A' ,'E'], 'to':['D', 'A',
'E','C','E','B']})

# Build your graph
graph=nx.from_pandas_edgelist(df, 'from', 'to')

# Plot it
nx.draw(graph, with_labels=True)
plt.show()
```

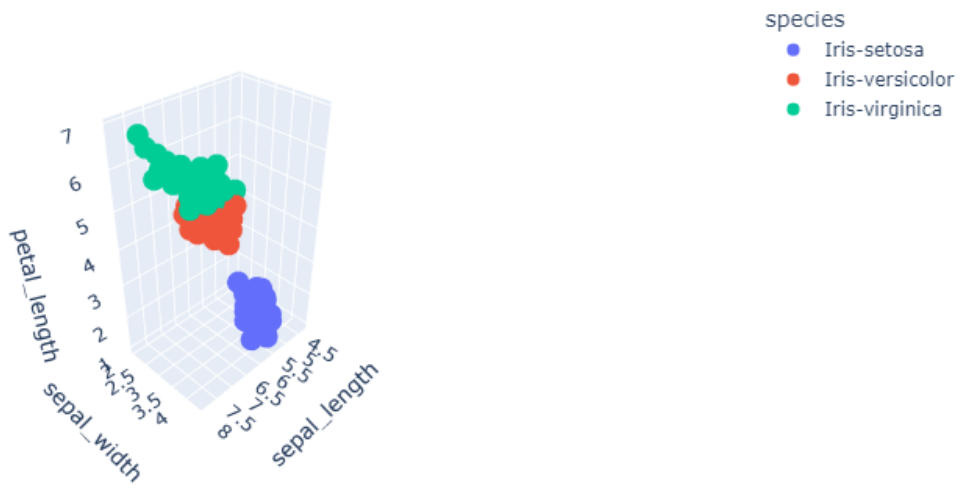


Return to the Top

3-d Plot

```
import plotly.express as px

fig = px.scatter_3d(iris, x='sepal_length', y='sepal_width', z='petal_length',
```

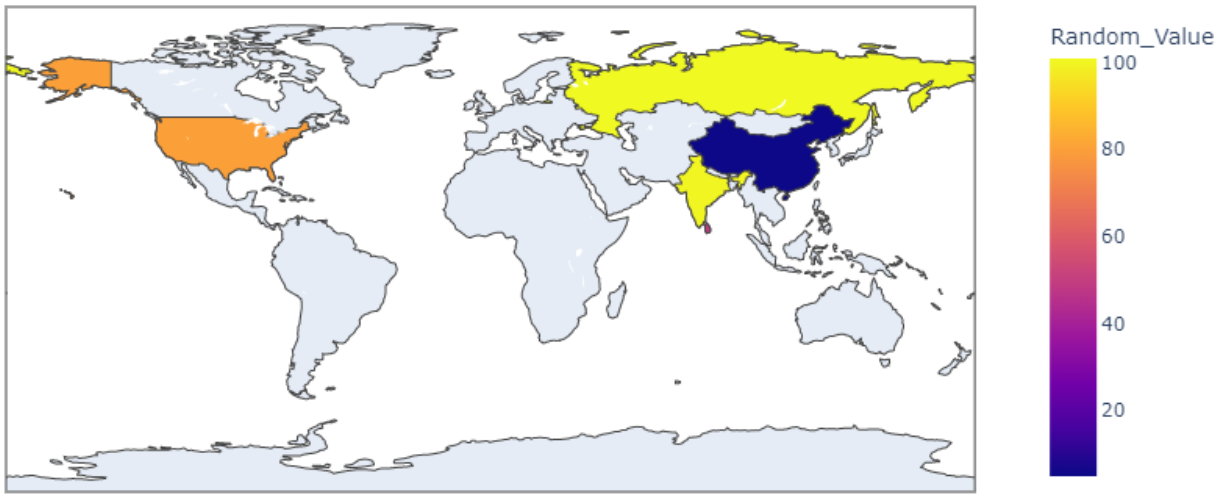
```
                color='species')
fig.show()
```



Return to the Top

Geographical Maps

```
df = pd.DataFrame({
    'Country': ['India','Russia','United States', 'China','Sri Lanka'],
    'Random_Value': [100,101,80,5,50]
})
fig = px.choropleth(df, locations="Country",
                    color="Random_Value",
                    locationmode='country names',
                    color_continuous_scale=px.colors.sequential.Plasma)
fig.show()
```



Return to the Top

Chapter 3 Basic Numpy

Return to the Top

```
import numpy as np
```

Creating Numpy Arrays

By converting lists to np arrays

```
x = np.array([1,2,3,4,5])
print(f'x={x}')
```

```
x=[1 2 3 4 5]
```

```
# We print information about x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x has dimensions: (5,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: int64
```

```
# We create a rank 1 ndarray that only contains strings
x = np.array(['Hello', 'World'])

# We print x
print()
print('x = ', x)
print()

# We print information about x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
```

```
x =  ['Hello' 'World']

x has dimensions: (2,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: <U5
```

```
# We create a rank 1 ndarray from a Python list that contains integers and strings
x = np.array([1, 2, 'World'])

# We print the ndarray
print()
print('x = ', x)
print()

# We print information about x
print('x has dimensions:', x.shape)
print('x is an object of type:', type(x))
print('The elements in x are of type:', x.dtype)
#Note the output whole array is of one single data type
```

```
x =  ['1' '2' 'World']

x has dimensions: (3,)
x is an object of type: <class 'numpy.ndarray'>
The elements in x are of type: <U21
```

```
# We create a rank 2 ndarray that only contains integers
Y = np.array([[1,2,3],[4,5,6],[7,8,9], [10,11,12]])

# We print Y
print()
print('Y = \n', Y)
print()

# We print information about Y
print('Y has dimensions:', Y.shape)
print('Y has a total of', Y.size, 'elements')
print('Y is an object of type:', type(Y))
print('The elements in Y are of type:', Y.dtype)
```

```
Y =
 [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]

Y has dimensions: (4, 3)
Y has a total of 12 elements
Y is an object of type: <class 'numpy.ndarray'>
The elements in Y are of type: int64
```

Even though NumPy automatically selects the dtype of the ndarray,
NumPy also allows you to specify the particular dtype you want to assign to the elements of the ndarray. You can specify the dtype when you create the ndarray using the keyword dtype in the np.array() function.

```
# We create a rank 1 ndarray of floats but set the dtype to int64
x = np.array([1.5, 2.2, 3.7, 4.0, 5.9], dtype = np.int64)

# We print x
print()
print('x = ', x)
```

```
print()

# We print the dtype x
print('The elements in x are of type:', x.dtype)
```

```
x =  [1 2 3 4 5]

The elements in x are of type: int64
```

Return to the Top

Saving numpy arrays into file and loading it(saved as .npy)

```
# We create a rank 1 ndarray
x = np.array([1, 2, 3, 4, 5])

# We save x into the current directory as
np.save('my_array', x)
```

```
# We load the saved array from our current directory into variable y
y = np.load('my_array.npy')

# We print y
print()
print('y = ', y)
print()

# We print information about the ndarray we loaded
print('y is an object of type:', type(y))
print('The elements in y are of type:', y.dtype)
```

```
y =  [1 2 3 4 5]

y is an object of type: <class 'numpy.ndarray'>
The elements in y are of type: int64
```

Return to the Top

Using Built-in Functions

numpy.zeros(shape, dtype=float, order='C')

```
X = np.zeros((3,4),dtype=int)
print(X)
```

```
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

Return to the Top

numpy.ones(shape, dtype=None, order='C')

```
X = np.ones((3,2))
print(X)
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

Return to the Top

numpy.full(shape, fill_value, dtype=None, order='C')

```
X = np.full((2,3), 5)#takes datatype of input value
print(X)
print()
Y= np.full((2,3), [1,'ishan',2]) #note number of columns must be equal to fill_value
array size
print(Y)
```

```
[[5 5 5]
 [5 5 5]]

[['1' 'ishan' '2']
 ['1' 'ishan' '2']]
```

Return to the Top

numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C')

```
X = np.eye(5,dtype=int)
print(X)
```

```
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]
```

Return to the Top

numpy.diag(v, k=0)

kint, optional

Diagonal in question. The default is 0. Use k>0 for diagonals above the main diagonal, and k<0 for diagonals below the main diagonal.

```
X = np.diag([10,20,30,50])
print(X)
```

```
[[10  0  0  0]
 [ 0 20  0  0]
 [ 0  0 30  0]
 [ 0  0  0 50]]
```

Return to the Top

numpy.arange([start,]stop, [step,]dtype=None)

```
x = np.arange(10)

print(x)
```

```
print()

y = np.arange(4,10)

print(y)
print()

z = np.arange(1,14,3)

print(z)
print()
```

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

[4 5 6 7 8 9]

[ 1  4  7 10 13]
```

even though the np.arange() function allows for non-integer steps, such as 0.3, the output is usually inconsistent, due to the finite floating point precision.

Return to the Top

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

```
# We create a rank 1 ndarray that has 10 integers evenly spaced between 0 and 25.
x = np.linspace(0,25,10)

# We print the ndarray
print()
print('x = \n', x)
print()
```

```
x =
 [ 0.          2.77777778  5.55555556  8.33333333 11.11111111 13.88888889
 16.66666667 19.44444444 22.22222222 25.        ]
```

Return to the Top

np.reshape(ndarray, new_shape)

```
x = np.reshape(x, (5,2))

print(x)
print()

Y = np.arange(20).reshape(4, 5)

print(Y)
```

```
[[ 0.          2.77777778]
 [ 5.55555556  8.33333333]
 [11.11111111 13.88888889]
 [16.66666667 19.44444444]
 [22.22222222 25.        ]]

[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

Return to the Top

np.random.random(shape)

with random floats in the half-open interval [0.0, 1.0)

```
X = np.random.random((3,3))

print(X)
```

```
[[0.61853248 0.32213218 0.39086456]
 [0.7180341  0.9293746  0.77943849]
 [0.57602494 0.69117078 0.07740787]]
```

Return to the Top

np.random.randint(start, stop, size = shape)

random integers in the half-open interval [start, stop)

```
X = np.random.randint(4,15,size=(3,2))
print(X)
```

```
[[11 10]
 [ 4  9]
 [ 5 14]]
```

In some cases, you may need to create ndarrays with random numbers that satisfy certain statistical properties. For example, you may want the random numbers in the ndarray to have an average of 0. NumPy allows you create random ndarrays with numbers drawn from various probability distributions. The function np.random.normal(mean, standard deviation, size=shape), for example, creates an ndarray with the given shape that contains random numbers picked from a normal (Gaussian) distribution with the given mean and standard deviation. Let's create a 1,000 x 1,000 ndarray of random floating point numbers drawn from a normal distribution with a mean (average) of zero and a standard deviation of 0.1.

```
X = np.random.normal(0, 0.1, size=(1000,1000))

print()
print('X = \n', X)
print()

# We print information about X
print('X has dimensions:', X.shape)
print('X is an object of type:', type(X))
print('The elements in X are of type:', X.dtype)
print('The elements in X have a mean of:', X.mean())
print('The maximum value in X is:', X.max())
print('The minimum value in X is:', X.min())
print('X has', (X < 0).sum(), 'negative numbers')
print('X has', (X > 0).sum(), 'positive numbers')
```

```
X =
 [[-0.04263308 -0.02294358  0.12837799 ...  0.07377007  0.18564738
  -0.12834024]
 [ 0.04318837 -0.14493155  0.01587642 ...  0.08378954 -0.1026855
  -0.11827792]
 [-0.0913915   0.14842814 -0.05293135 ... -0.01925517  0.06544787
  -0.11646424]
 ...
 [ 0.03422818 -0.09721159  0.06133823 ... -0.0273124   0.03284017
  -0.1017433 ]
```

```
 [-0.04687082 -0.09635745 -0.19078807 ...  0.08339463 -0.03262759
  -0.03174562]
 [-0.0440551  -0.00469147  0.02960465 ...  0.15276602  0.19640379
   0.1113167 ]]

X has dimensions: (1000, 1000)
X is an object of type: <class 'numpy.ndarray'>
The elements in X are of type: float64
The elements in X have a mean of: 9.73572160453743e-06
The maximum value in X is: 0.49855043341077726
The minimum value in X is: -0.47138648302414243
X has 499877 negative numbers
X has 500123 positive numbers
```

Return to the Top

Inserting/Deleting in numpy arrays

numpy arrays are mutable and hence can be changed after assignment

Accessing elements

```
x = np.array([1, 2, 3, 4, 5])

print(x[0])
print()
print(x[-1])
```

```
1

5
```

Return to the Top

Modifying Elements

```
x[0] = 6
print(x)
```

```
[6 2 3 4 5]
```

```
X = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(X)

print()

print(X[0][0])

print()

X[0][0] = 10

print(X)
```

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

1

[[10  2  3]
 [ 4  5  6]
 [ 7  8  9]]
```

Return to the Top

numpy.delete(arr, obj, axis=None)

axis=0 implies x-axis
axis=1 implies y-axis

```
x = np.array([1, 2, 3, 4, 5])
print(x)

x = np.delete(x,[0,4])#deletes first and last element
print()
print(x)
print()

Y = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(Y)
print()
```

```
# We delete the first row of y
w = np.delete(Y, 0, axis=0)
print(w)
print()

# We delete the first and last column of y
v = np.delete(Y, [0,2], axis=1)
print(v)
```

```
[1 2 3 4 5]

[2 3 4]

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

[[4 5 6]
 [7 8 9]]

[[2]
 [5]
 [8]]
```

Return to the Top

numpy.append(arr, values, axis=None)

```
x = np.array([1, 2, 3, 4, 5])

print(x,end='\n\n')

x = np.append(x, 6)

print(x)
print()

x = np.append(x, [7,8])

print(x)
print()

Y = np.array([[1,2,3],[4,5,6]])
print(Y)
```

```
print()

v = np.append(Y, [[7,8,9]], axis=0)
print(v)
print()

q = np.append(Y,[[9],[10]], axis=1)#note this
print(q)
```

```
[1 2 3 4 5]

[1 2 3 4 5 6]

[1 2 3 4 5 6 7 8]

[[1 2 3]
 [4 5 6]]

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

[[ 1  2  3  9]
 [ 4  5  6 10]]
```

Return to the Top

numpy.insert(arr, obj, values, axis=None)

```
x = np.array([1, 2, 5, 6, 7])
print(x)
print()

x = np.insert(x,2,[3,4])#at 2nd index
print(x)
print()

Y = np.array([[1,2,3],[7,8,9]])
print(Y)
print()

# We insert a row between the first and last row of y
w = np.insert(Y,1,[4,5,6],axis=0)
```

```
print(w)
print()

# We insert a column full of 5s between the first and second column of y
v = np.insert(Y,1,5, axis=1)
print(v)
```

```
[1 2 5 6 7]

[1 2 3 4 5 6 7]

[[1 2 3]
 [7 8 9]]

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

[[1 5 2 3]
 [7 5 8 9]]
```

NumPy also allows us to stack ndarrays on top of each other, or to stack them side by side. The stacking is done using either the np.vstack() function for vertical stacking, or the np.hstack() function for horizontal stacking. It is important to note that in order to stack ndarrays, the shape of the ndarrays must match.

```
x = np.array([1,2])

Y = np.array([[3,4],[5,6]])
print(Y)
print()
z = np.vstack((x,Y))
print(z)
print()

w = np.hstack((Y,x.reshape(2,1)))
print(w)
```

```
[[3 4]
 [5 6]]

[[1 2]
```

```
 [3 4]
 [5 6]]

[[3 4 1]
 [5 6 2]]
```

Return to the Top

slicing

ndarray[start:end]

ndarray[start:]

ndarray[:end]

```
X = np.arange(20).reshape(4, 5)
print(X)
print()

# We select all the elements that are in the 2nd through 4th rows and in the 3rd to
5th columns
Z = X[1:4,2:5]#row,cloumn
print(Z)
print()

# We select all the elements that are in the 1st through 3rd rows and in the 3rd to
4th columns
Y = X[:3,2:5]
print(Y)
print()

# We select all the elements in the 3rd row
v = X[2,:]
print(v)#note it prints 1d array
print()

# We select all the elements in the 3rd column but return a rank 2 ndarray
R = X[:,2:3]
print(R)
print()
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

[[ 7  8  9]
 [12 13 14]
 [17 18 19]]

[[ 2  3  4]
 [ 7  8  9]
 [12 13 14]]

[10 11 12 13 14]

[[ 2]
 [ 7]
 [12]
 [17]]
```

It is important to note that when we perform slices on ndarrays and save them into new variables, as we did above, the data is not copied into the new variable.

Z = X[1:4,2:5]

the slice of the original array X is not copied in the variable Z. Rather, X and Z are now just two different names for the same ndarray. We say that slicing only creates a view of the original array. This means that if you make changes in Z you will be in effect changing the elements in X as well.

```
X = np.arange(20).reshape(4, 5)
print(X)
print()

Z = X[1:4,2:5]
print(Z)
print()

Z[2,2] = 555

print(X)
```

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

```
 [10 11 12 13 14]
 [15 16 17 18 19]]

[[ 7  8  9]
 [12 13 14]
 [17 18 19]]

[[  0   1   2   3   4]
 [  5   6   7   8   9]
 [ 10  11  12  13  14]
 [ 15  16  17  18 555]]
```

Return to the Top

np.copy()

```
X = np.arange(20).reshape(4, 5)
print(X)
print()

W = X[1:4,2:5].copy()

W[2,2] = 444

print(X)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]
```

It is often useful to use one ndarray to make slices, select, or change elements in another ndarray. Let's see some examples:

```
# We create a 4 x 5 ndarray that contains integers from 0 to 19
X = np.arange(20).reshape(4, 5)
```

```
# We create a rank 1 ndarray that will serve as indices to select elements from X
indices = np.array([1,3])

# We print X
print()
print('X = \n', X)
print()

# We print indices
print('indices = ', indices)
print()

# We use the indices ndarray to select the 2nd and 4th row of X
Y = X[indices,:]

# We use the indices ndarray to select the 2nd and 4th column of X
Z = X[:, indices]

# We print Y
print()
print('Y = \n', Y)

# We print Z
print()
print('Z = \n', Z)
```

```
X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

indices =  [1 3]


Y =
 [[ 5  6  7  8  9]
 [15 16 17 18 19]]

Z =
 [[ 1  3]
 [ 6  8]
 [11 13]
 [16 18]]
```

**np.diag(arr,k=0)

np.unique(arr)**

Return to the Top

Boolean indexing

```
# We create a 5 x 5 ndarray that contains integers from 0 to 24
X = np.arange(25).reshape(5, 5)

# We print X
print()
print('Original X = \n', X)
print()

# We use Boolean indexing to select elements in X:
print('The elements in X that are greater than 10:', X[X > 10])
print('The elements in X that less than or equal to 7:', X[X <= 7])
print('The elements in X that are between 10 and 17:', X[(X > 10) & (X < 17)])

# We use Boolean indexing to assign the elements that are between 10 and 17 the
value of -1
X[(X > 10) & (X < 17)] = -1

# We print X
print()
print('X = \n', X)
print()
```

```
Original X =
 [[ 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]]

The elements in X that are greater than 10: [11 12 13 14 15 16 17 18 19 20 21 22 23
24]
The elements in X that less than or equal to 7: [0 1 2 3 4 5 6 7]
The elements in X that are between 10 and 17: [11 12 13 14 15 16]

X =
 [[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 -1 -1 -1 -1]
 [-1 -1 17 18 19]
 [20 21 22 23 24]]
```

Return to the Top

set operations

```
# We create a rank 1 ndarray
x = np.array([1,2,3,4,5])

# We create a rank 1 ndarray
y = np.array([6,7,2,8,4])

# We print x
print()
print('x = ', x)

# We print y
print()
print('y = ', y)

# We use set operations to compare x and y:
print()
print('The elements that are both in x and y:', np.intersect1d(x,y))
print('The elements that are in x that are not in y:', np.setdiff1d(x,y))
print('All the elements of x and y:',np.union1d(x,y))
```

```
x =  [1 2 3 4 5]

y =  [6 7 2 8 4]

The elements that are both in x and y: [2 4]
The elements that are in x that are not in y: [1 3 5]
All the elements of x and y: [1 2 3 4 5 6 7 8]
```

Return to the Top

sort

```
# We create an unsorted rank 1 ndarray
x = np.random.randint(1,11,size=(10,))

# We print x
print()
print('Original x = ', x)
```

```
# We sort x and print the sorted array using sort as a function.
print()
print('Sorted x (out of place):', np.sort(x))

# When we sort out of place the original array remains intact. To see this we print
x again
print()
print('x after sorting:', x)
```

```
Original x =  [4 9 2 9 1 9 9 6 8 6]

Sorted x (out of place): [1 2 4 6 6 8 9 9 9 9]

x after sorting: [4 9 2 9 1 9 9 6 8 6]
```

```
# We create an unsorted rank 1 ndarray
x = np.random.randint(1,11,size=(10,))

# We print x
print()
print('Original x = ', x)

# We sort x and print the sorted array using sort as a method.
x.sort()

# When we sort in place the original array is changed to the sorted array. To see
this we print x again
print()
print('x after sorting:', x)
```

```
Original x =  [5 2 4 6 8 1 1 6 3 9]

x after sorting: [1 1 2 3 4 5 6 6 8 9]
```

```
# We create an unsorted rank 2 ndarray
X = np.random.randint(1,11,size=(5,5))

# We print X
```

```
print()
print('Original X = \n', X)
print()

# We sort the columns of X and print the sorted array
print()
print('X with sorted columns :\n', np.sort(X, axis = 0))

# We sort the rows of X and print the sorted array
print()
print('X with sorted rows :\n', np.sort(X, axis = 1))
```

```
Original X =
 [[ 4  4  6  9  2]
 [ 1  1  1  6  9]
 [ 3  8  6  3  8]
 [ 4  6  3 10 10]
 [ 2  5  8  1  2]]


X with sorted columns :
 [[ 1  1  1  1  2]
 [ 2  4  3  3  2]
 [ 3  5  6  6  8]
 [ 4  6  6  9  9]
 [ 4  8  8 10 10]]

X with sorted rows :
 [[ 2  4  4  6  9]
 [ 1  1  1  6  9]
 [ 3  3  6  8  8]
 [ 3  4  6 10 10]
 [ 1  2  2  5  8]]
```

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

X = X[X%2!=0]

print(X)
```

```
[[ 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]]
[ 1  3  5  7  9 11 13 15 17 19 21 23 25]
```

Return to the Top

Arithmetic operations

```
# We create two rank 1 ndarrays
x = np.array([1,2,3,4])
y = np.array([5.5,6.5,7.5,8.5])

# We print x
print()
print('x = ', x)

# We print y
print()
print('y = ', y)
print()

# We perfrom basic element-wise operations using arithmetic symbols and functions
print('x + y = ', x + y)
print('add(x,y) = ', np.add(x,y))
print()
print('x - y = ', x - y)
print('subtract(x,y) = ', np.subtract(x,y))
print()
print('x * y = ', x * y)
print('multiply(x,y) = ', np.multiply(x,y))
print()
print('x / y = ', x / y)
print('divide(x,y) = ', np.divide(x,y))
```

```
x =  [1 2 3 4]

y =  [5.5 6.5 7.5 8.5]

x + y =  [ 6.5  8.5 10.5 12.5]
add(x,y) =  [ 6.5  8.5 10.5 12.5]
```

```
x - y =  [-4.5 -4.5 -4.5 -4.5]
subtract(x,y) =  [-4.5 -4.5 -4.5 -4.5]

x * y =  [ 5.5 13.  22.5 34. ]
multiply(x,y) =  [ 5.5 13.  22.5 34. ]

x / y =  [0.18181818 0.30769231 0.4        0.47058824]
divide(x,y) =  [0.18181818 0.30769231 0.4        0.47058824]
```

```
# We create a rank 1 ndarray
x = np.array([1,2,3,4])

# We print x
print()
print('x = ', x)

# We apply different mathematical functions to all elements of x
print()
print('EXP(x) =', np.exp(x))
print()
print('SQRT(x) =',np.sqrt(x))
print()
print('POW(x,2) =',np.power(x,2)) # We raise all elements to the power of 2
```

```
x =  [1 2 3 4]

EXP(x) = [ 2.71828183  7.3890561  20.08553692 54.59815003]

SQRT(x) = [1.         1.41421356 1.73205081 2.        ]

POW(x,2) = [ 1  4  9 16]
```

```
# We create a 2 x 2 ndarray
X = np.array([[1,2], [3,4]])

# We print x
print()
print('X = \n', X)
print()

print('Average of all elements in X:', X.mean())
print('Average of all elements in the columns of X:', X.mean(axis=0))
print('Average of all elements in the rows of X:', X.mean(axis=1))
```

```
print()
print('Sum of all elements in X:', X.sum())
print('Sum of all elements in the columns of X:', X.sum(axis=0))
print('Sum of all elements in the rows of X:', X.sum(axis=1))
print()
print('Standard Deviation of all elements in X:', X.std())
print('Standard Deviation of all elements in the columns of X:', X.std(axis=0))
print('Standard Deviation of all elements in the rows of X:', X.std(axis=1))
print()
print('Median of all elements in X:', np.median(X))
print('Median of all elements in the columns of X:', np.median(X,axis=0))
print('Median of all elements in the rows of X:', np.median(X,axis=1))
print()
print('Maximum value of all elements in X:', X.max())
print('Maximum value of all elements in the columns of X:', X.max(axis=0))
print('Maximum value of all elements in the rows of X:', X.max(axis=1))
print()
print('Minimum value of all elements in X:', X.min())
print('Minimum value of all elements in the columns of X:', X.min(axis=0))
print('Minimum value of all elements in the rows of X:', X.min(axis=1))
```

```
X =
 [[1 2]
 [3 4]]

Average of all elements in X: 2.5
Average of all elements in the columns of X: [2. 3.]
Average of all elements in the rows of X: [1.5 3.5]

Sum of all elements in X: 10
Sum of all elements in the columns of X: [4 6]
Sum of all elements in the rows of X: [3 7]

Standard Deviation of all elements in X: 1.118033988749895
Standard Deviation of all elements in the columns of X: [1. 1.]
Standard Deviation of all elements in the rows of X: [0.5 0.5]

Median of all elements in X: 2.5
Median of all elements in the columns of X: [2. 3.]
Median of all elements in the rows of X: [1.5 3.5]

Maximum value of all elements in X: 4
Maximum value of all elements in the columns of X: [3 4]
Maximum value of all elements in the rows of X: [2 4]

Minimum value of all elements in X: 1
```

```
Minimum value of all elements in the columns of X: [1 2]
Minimum value of all elements in the rows of X: [1 3]
```

```
# We create a 2 x 2 ndarray
X = np.array([[1,2], [3,4]])

# We print x
print()
print('X = \n', X)
print()

print('3 * X = \n', 3 * X)
print()
print('3 + X = \n', 3 + X)
print()
print('X - 3 = \n', X - 3)
print()
print('X / 3 = \n', X / 3)
```

```
X =
 [[1 2]
 [3 4]]

3 * X =
 [[ 3  6]
 [ 9 12]]

3 + X =
 [[4 5]
 [6 7]]

X - 3 =
 [[-2 -1]
 [ 0  1]]

X / 3 =
 [[0.33333333 0.66666667]
 [1.         1.33333333]]
```

.

.

Chapter 4 Basic Pandas

Return to the Top

Why Use Pandas?

The recent success of machine learning algorithms is partly due to the huge amounts of data that we have available to train our algorithms on. However, when it comes to data, quantity is not the only thing that matters, the quality of your data is just as important. It often happens that large datasets don't come ready to be fed into your learning algorithms. More often than not, large datasets will often have missing values, outliers, incorrect values, etc... Having data with a lot of missing or bad values, for example, is not going to allow your machine learning algorithms to perform well. Therefore, one very important step in machine learning is to look at your data first and make sure it is well suited for your training algorithm by doing some basic data analysis. This is where Pandas come in. Pandas Series and DataFrames are designed for fast data analysis and manipulation, as well as being flexible and easy to use. Below are just a few features that makes Pandas an excellent package for data analysis:

1. Allows the use of labels for rows and columns
2. Can calculate rolling statistics on time series data
3. Easy handling of NaN values
4. Is able to load data of different formats into DataFrames
5. Can join and merge different datasets together
6. It integrates with NumPy and Matplotlib

For these and other reasons, Pandas DataFrames have become one of the most commonly used Pandas object for data analysis in Python.

Note:pandas inherit from numpy and hence features like numpy boolean indexing and other operations of numpy arrays can be applied here too.

Pandas series

Return to the Top

pandas.Series(data,index)

important poitnts

1. one-dimensional array-like object
2. can hold data of different data types(but in numpy arrays all data types are converted into one)

```
import pandas as pd

groceries = pd.Series(data = [30, 6, 'Yes', 'No'], index = ['eggs', 'apples',
'milk', 'bread'])
```

```
groceries
```

```
eggs       30
apples      6
milk      Yes
bread      No
dtype: object
```

```
# We print some information about Groceries
print('Groceries has shape:', groceries.shape)
print('Groceries has dimension:', groceries.ndim)
print('Groceries has a total of', groceries.size, 'elements')
print()
# We print the index and data of Groceries
print('The data in Groceries is:', groceries.values)
print('The index of Groceries is:', groceries.index)
```

```
Groceries has shape: (4,)
Groceries has dimension: 1
Groceries has a total of 4 elements

The data in Groceries is: [30 6 'Yes' 'No']
The index of Groceries is: Index(['eggs', 'apples', 'milk', 'bread'],
dtype='object')
```

```
# We check whether bread is a food item (an index) in Groceries
print('bread' in groceries)#check index not data
```

```
True
```

Return to the Top

Manipulating pandas series

```
#Accessing elements using labes
print(groceries['eggs'])
print()
print(groceries[['eggs','milk']])
print()
print(groceries.loc[['eggs','apples']])
print()
#Accessing elemts using integer index
print(groceries[0])
print()
print(groceries[[0,2]])
print()
print(groceries.iloc[[0,1]])
```

```
30

eggs     30
milk    Yes
dtype: object

eggs      30
apples     6
dtype: object

30

eggs     30
milk    Yes
dtype: object

eggs      30
apples     6
dtype: object
```

to remove any ambiguity to whether we are referring to an index label or numerical index use loc and iloc

Return to the Top

pandas series are mutable and hence can be modified

```
# We display the original grocery list
print('Original Grocery List:\n', groceries)

# We change the number of eggs to 2
```

```
groceries['eggs'] = 2

# We display the changed grocery list
print()
print('Modified Grocery List:\n', groceries)
```

```
Original Grocery List:
 eggs       30
apples      6
milk      Yes
bread      No
dtype: object

Modified Grocery List:
 eggs        2
apples      6
milk      Yes
bread      No
dtype: object
```

Return to the Top

Series.drop(label, inplace=False)

```
# We display the original grocery list
print('Original Grocery List:\n', groceries)

# We remove apples from our grocery list. The drop function removes elements out of
place
print()
print('We remove apples (out of place):\n', groceries.drop('apples'))

# When we remove elements out of place the original Series remains intact. To see
this
# we display our grocery list again
print()
print('Grocery List after removing apples out of place is still intact:\n',
groceries)
```

```
Original Grocery List:
 eggs        2
apples      6
```

```
milk      Yes
bread      No
dtype: object

We remove apples (out of place):
 eggs       2
milk     Yes
bread     No
dtype: object

Grocery List after removing apples out of place is still intact:
 eggs        2
apples      6
milk      Yes
bread      No
dtype: object
```

if we set inplace to True than original list is also modified

```
groceries.drop('apples', inplace = True)
print(groceries)
```

```
eggs       2
milk     Yes
bread     No
dtype: object
```

Return to the Top

Arithmetic operations

```
fruits= pd.Series(data = [10, 6, 3,], index = ['apples', 'oranges', 'bananas'])
fruits
```

```
apples     10
oranges     6
bananas     3
dtype: int64
```

```
# We print fruits for reference
print('Original grocery list of fruits:\n ', fruits)

# We perform basic element-wise operations using arithmetic symbols
print()
print('fruits + 2:\n', fruits + 2) # We add 2 to each item in fruits
print()
print('fruits - 2:\n', fruits - 2) # We subtract 2 to each item in fruits
print()
print('fruits * 2:\n', fruits * 2) # We multiply each item in fruits by 2
print()
print('fruits / 2:\n', fruits / 2) # We divide each item in fruits by 2
print()
```

```
Original grocery list of fruits:
  apples     10
oranges     6
bananas     3
dtype: int64

fruits + 2:
 apples     12
oranges     8
bananas     5
dtype: int64

fruits - 2:
 apples     8
oranges    4
bananas    1
dtype: int64

fruits * 2:
 apples     20
oranges    12
bananas     6
dtype: int64

fruits / 2:
 apples     5.0
oranges    3.0
bananas    1.5
dtype: float64
```

Return to the Top

Applying mathemetical functions from numpy

```
import numpy as np

# We print fruits for reference
print('Original grocery list of fruits:\n', fruits)

# We apply different mathematical functions to all elements of fruits
print()
print('EXP(X) = \n', np.exp(fruits))
print()
print('SQRT(X) =\n', np.sqrt(fruits))
print()
print('POW(X,2) =\n',np.power(fruits,2))
```

```
Original grocery list of fruits:
 apples     10
oranges     6
bananas     3
dtype: int64

EXP(X) =
 apples     22026.465795
oranges      403.428793
bananas       20.085537
dtype: float64

SQRT(X) =
 apples     3.162278
oranges    2.449490
bananas    1.732051
dtype: float64

POW(X,2) =
 apples     100
oranges     36
bananas      9
dtype: int64
```

Return to the Top

On single elements

```
# We print fruits for reference
print('Original grocery list of fruits:\n ', fruits)
print()

# We add 2 only to the bananas
print('Amount of bananas + 2 = ', fruits['bananas'] + 2)
print()

# We subtract 2 from apples
print('Amount of apples - 2 = ', fruits.iloc[0] - 2)
print()

# We multiply apples and oranges by 2
print('We double the amount of apples and oranges:\n', fruits[['apples', 'oranges']]
* 2)
print()

# We divide apples and oranges by 2
print('We half the amount of apples and oranges:\n', fruits.loc[['apples',
'oranges']] / 2)
```

```
Original grocery list of fruits:
  apples     10
oranges     6
bananas     3
dtype: int64

Amount of bananas + 2 =  5

Amount of apples - 2 =  8

We double the amount of apples and oranges:
 apples     20
oranges    12
dtype: int64

We half the amount of apples and oranges:
 apples     5.0
oranges    3.0
dtype: float64
```

Return to the Top

operation on series with mixed data types

be carefull to check whether the operator applies to all selected elements

```
print(groceries)
print()
print(groceries*2)#multiplication on string doubles it
#if we apply '/' than there will be error as string dont have '/' operator
```

```
eggs       2
milk     Yes
bread     No
dtype: object

eggs          4
milk     YesYes
bread      NoNo
dtype: object
```

Return to the Top

Pandas DataFrames

1. think of Pandas DataFrames as being similar to a spreadsheet.
2. two-dimensional data structures with labeled rows and columns.
3. can hold many data types.

Creating By Dictionary

using pandas series in dictionary

```
# We import Pandas as pd into Python
import pandas as pd

# We create a dictionary of Pandas Series
items = {'Bob' : pd.Series(data = [245, 25, 55], index = ['bike', 'pants',
'watch']),
         'Alice' : pd.Series(data = [40, 110, 500, 45], index = ['book', 'glasses',
'bike', 'pants'])}

# We print the type of items to see that it is a dictionary
print(type(items))
```

```
<class 'dict'>
```

```
# We create a Pandas DataFrame by passing it a dictionary of Pandas Series
shopping_carts = pd.DataFrame(items)

# We display the DataFrame rows are alphabetically arranged
shopping_carts
```

	Bob	Alice
bike	245.0	500.0
book	NaN	40.0
glasses	NaN	110.0
pants	25.0	45.0
watch	55.0	NaN

NaN stands for Not a Number, and is Pandas way of indicating that it doesn't have a value for that particular row and column index.

If we don't provide index labels to the Pandas Series, Pandas will use numerical row indexes when it creates the DataFrame.

```
# We create a dictionary of Pandas Series without indexes
data = {'Bob' : pd.Series([245, 25, 55]),
        'Alice' : pd.Series([40, 110, 500, 45])}

# We create a DataFrame
df = pd.DataFrame(data)

# We display the DataFrame
df
```

	Bob	Alice
0	245.0	40
1	25.0	110
2	55.0	500

	Bob	Alice
3	NaN	45

```
# We print some information about shopping_carts
print('shopping_carts has shape:', shopping_carts.shape)
print('shopping_carts has dimension:', shopping_carts.ndim)
print('shopping_carts has a total of:', shopping_carts.size, 'elements')
print()
print('The data in shopping_carts is:\n', shopping_carts.values)
print()
print('The row index in shopping_carts is:', shopping_carts.index)
print()
print('The column index in shopping_carts is:', shopping_carts.columns)
```

```
shopping_carts has shape: (5, 2)
shopping_carts has dimension: 2
shopping_carts has a total of: 10 elements

The data in shopping_carts is:
 [[245. 500.]
 [ nan  40.]
 [ nan 110.]
 [ 25.  45.]
 [ 55.  nan]]

The row index in shopping_carts is: Index(['bike', 'book', 'glasses', 'pants',
'watch'], dtype='object')

The column index in shopping_carts is: Index(['Bob', 'Alice'], dtype='object')
```

```
# We Create a DataFrame that only has Bob's data
bob_shopping_cart = pd.DataFrame(items, columns=['Bob'])

# We display bob_shopping_cart
bob_shopping_cart
```

	Bob
bike	245

	Bob
pants	25
watch	55

```
# We Create a DataFrame that only has selected items for both Alice and Bob
sel_shopping_cart = pd.DataFrame(items, index = ['pants', 'book'])

# We display sel_shopping_cart
sel_shopping_cart
```

	Bob	Alice
pants	25.0	45
book	NaN	40

```
# We Create a DataFrame that only has selected items for Alice
alice_sel_shopping_cart = pd.DataFrame(items, index = ['glasses', 'bike'], columns =
['Alice'])

# We display alice_sel_shopping_cart
alice_sel_shopping_cart
```

	Alice
glasses	110
bike	500

You can also manually create DataFrames from a dictionary of lists (arrays). The procedure is the same as before, we start by creating the dictionary and then passing the dictionary to the pd.DataFrame() function. In this case, however, all the lists (arrays) in the dictionary must be of the same length. Let' see an example:

```
# We create a dictionary of lists (arrays)
data = {'Integers' : [1,2,3],
        'Floats' : [4.5, 8.2, 9.6]}

# We create a DataFrame
df = pd.DataFrame(data)
```

```
# We display the DataFrame
df
```

	Integers	Floats
0	1	4.5
1	2	8.2
2	3	9.6

```
# We create a dictionary of lists (arrays)
data = {'Integers' : [1,2,3],
        'Floats' : [4.5, 8.2, 9.6]}

# We create a DataFrame and provide the row index
df = pd.DataFrame(data, index = ['label 1', 'label 2', 'label 3'])

# We display the DataFrame
df
```

	Integers	Floats
label 1	1	4.5
label 2	2	8.2
label 3	3	9.6

Return to the Top

using list of dictionery

```
# We create a list of Python dictionaries
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35},
          {'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5}]

# We create a DataFrame
store_items = pd.DataFrame(items2)

# We display the DataFrame
store_items
```

	bikes	pants	watches	glasses

	bikes	pants	watches	glasses
0	20	30	35	NaN
1	15	5	10	50.0

```
# We create a list of Python dictionaries
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35},
          {'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5}]

# We create a DataFrame  and provide the row index
store_items = pd.DataFrame(items2, index = ['store 1', 'store 2'])

# We display the DataFrame
store_items
```

	bikes	pants	watches	glasses
store 1	20	30	35	NaN
store 2	15	5	10	50.0

Return to the Top

Manipulating elements

dataframe[column][row]

```
# We print the store_items DataFrame
print(store_items)

# We access rows, columns and elements using labels
print()
print('How many bikes are in each store:\n', store_items[['bikes']])
print()
print('How many bikes and pants are in each store:\n', store_items[['bikes',
'pants']])
print()
print('What items are in Store 1:\n', store_items.loc[['store 1']])
print()
print('How many bikes are in Store 2:', store_items['bikes']['store 2'])#[columns]
[rows]
```

```
         bikes  pants  watches  glasses
store 1     20     30       35      NaN
store 2     15      5       10     50.0

How many bikes are in each store:
          bikes
store 1     20
store 2     15

How many bikes and pants are in each store:
          bikes  pants
store 1     20     30
store 2     15      5

What items are in Store 1:
          bikes  pants  watches  glasses
store 1     20     30       35      NaN

How many bikes are in Store 2: 15
```

```
# We add a new column named shirts to our store_items DataFrame indicating the
number of
# shirts in stock at each store. We will put 15 shirts in store 1 and 2 shirts in
store 2
store_items['shirts'] = [15,2]

# We display the modified DataFrame
store_items
```

	bikes	pants	watches	glasses	shirts
store 1	20	30	35	NaN	15
store 2	15	5	10	50.0	2

```
# We make a new column called suits by adding the number of shirts and pants
store_items['suits'] = store_items['pants'] + store_items['shirts']

# We display the modified DataFrame
store_items
```

	bikes	pants	watches	glasses	shirts	suits

	bikes	pants	watches	glasses	shirts	suits
store 1	20	30	35	NaN	15	45
store 2	15	5	10	50.0	2	7

Return to the Top

Adding new row

append()

```
# We create a dictionary from a list of Python dictionaries that will number of
items at the new store
new_items = [{'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4}]

# We create new DataFrame with the new_items and provide and index labeled store 3
new_store = pd.DataFrame(new_items, index = ['store 3'])

# We display the items at the new store
print(new_store)

# We append store 3 to our store_items DataFrame
store_items = store_items.append(new_store)

# We display the modified DataFrame
store_items
```

```
         bikes  pants  watches  glasses
store 3     20     30       35        4
```

	bikes	pants	watches	glasses	shirts	suits
store 1	20	30	35	NaN	15.0	45.0
store 2	15	5	10	50.0	2.0	7.0
store 3	20	30	35	4.0	NaN	NaN

Return to the Top

Adding new column to specific rows

```
# We add a new column using data from particular rows in the watches column
# this adds watches to store  and store3
store_items['new watches'] = store_items['watches'][1:]

# We display the modified DataFrame
store_items
```

	bikes	pants	watches	glasses	shirts	suits	new watches
store 1	20	30	35	NaN	15.0	45.0	NaN
store 2	15	5	10	50.0	2.0	7.0	10.0
store 3	20	30	35	4.0	NaN	NaN	35.0

Return to the Top

Adds a new column

```
# We insert a new column with label shoes right before the column with numerical
index 4
store_items.insert(4, 'shoes', [8,5,0])

# we display the modified DataFrame
store_items
```

	bikes	pants	watches	glasses	shoes	shirts	suits	new watches
store 1	20	30	35	NaN	8	15.0	45.0	NaN
store 2	15	5	10	50.0	5	2.0	7.0	10.0
store 3	20	30	35	4.0	0	NaN	NaN	35.0

Return to the Top

Deleting elemets

```
# We remove the new watches column
store_items.pop('new watches')

# we display the modified DataFrame
store_items
```

	bikes	pants	watches	glasses	shoes	shirts	suits
store 1	20	30	35	NaN	8	15.0	45.0
store 2	15	5	10	50.0	5	2.0	7.0
store 3	20	30	35	4.0	0	NaN	NaN

Return to the Top

.drop() can delete both rows or columns using axis keyword

axis = 1 ->y-axis

axis = 0 ->x-axis

```
# We remove the watches and shoes columns
store_items = store_items.drop(['watches', 'shoes'], axis = 1)

# we display the modified DataFrame
store_items
```

	bikes	pants	glasses	shirts	suits
store 1	20	30	NaN	15.0	45.0
store 2	15	5	50.0	2.0	7.0
store 3	20	30	4.0	NaN	NaN

```
# We remove the store 2 and store 1 rows
store_items = store_items.drop(['store 2', 'store 1'], axis = 0)

# we display the modified DataFrame
store_items
```

	bikes	pants	glasses	shirts	suits
store 3	20	30	4.0	NaN	NaN

Return to the Top

Renaming labels

```
# We change the column label bikes to hats
store_items = store_items.rename(columns = {'bikes': 'hats'})

# we display the modified DataFrame
store_items
```

	hats	pants	glasses	shirts	suits
store 3	20	30	4.0	NaN	NaN

```
# We change the row label from store 3 to last store
store_items = store_items.rename(index = {'store 3': 'last store'})

# we display the modified DataFrame
store_items
```

	hats	pants	glasses	shirts	suits
last store	20	30	4.0	NaN	NaN

```
# We change the row index to be the data in the pants column
store_items = store_items.set_index('pants')

# we display the modified DataFrame
store_items
```

	hats	glasses	shirts	suits
pants				
30	20	4.0	NaN	NaN

Return to the Top

Dealing with NaN values

```
# We create a list of Python dictionaries
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35, 'shirts': 15, 'shoes':8,
'suits':45},
{'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5, 'shirts': 2, 'shoes':5,
'suits':7},
{'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4, 'shoes':10}]
```

```
# We create a DataFrame  and provide the row index
store_items = pd.DataFrame(items2, index = ['store 1', 'store 2', 'store 3'])

# We display the DataFrame
store_items
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	NaN
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	NaN	10	NaN	4.0

Return to the Top

Counting NaN values

```
# We count the number of NaN values in store_items
# break it to see how it works
x =  store_items.isnull().sum().sum()

# We print x
print('Number of NaN values in our DataFrame:', x)
```

```
Number of NaN values in our DataFrame: 3
```

Return to the Top

Counting non NaN values

```
store_items.count()
```

```
bikes      3
pants      3
watches    3
shirts     2
shoes      3
suits      2
```

```
glasses    2
dtype: int64
```

Return to the Top

deleting NaN value

The .dropna(axis) method eliminates any rows with NaN values when axis = 0 is used and will eliminate any columns with NaN values when axis = 1 is used. Let's see some examples

```
print(store_items)
print('-'*65)
print()
print(store_items.dropna(axis=0))
print('-'*65)
print()
print(store_items)
print('-'*65)
print()
print(store_items.dropna(axis=1))
```

```
         bikes  pants  watches  shirts  shoes  suits  glasses
store 1     20     30       35    15.0      8   45.0      NaN
store 2     15      5       10     2.0      5    7.0     50.0
store 3     20     30       35     NaN     10    NaN      4.0
-----------------------------------------------------------------

         bikes  pants  watches  shirts  shoes  suits  glasses
store 2     15      5       10     2.0      5    7.0     50.0
-----------------------------------------------------------------

         bikes  pants  watches  shirts  shoes  suits  glasses
store 1     20     30       35    15.0      8   45.0      NaN
store 2     15      5       10     2.0      5    7.0     50.0
store 3     20     30       35     NaN     10    NaN      4.0
-----------------------------------------------------------------

         bikes  pants  watches  shoes
store 1     20     30       35      8
store 2     15      5       10      5
store 3     20     30       35     10
```

Notice that the .dropna() method eliminates (drops) the rows or columns with NaN values out of place. This means that the original DataFrame is not modified. You can always remove the desired rows or columns in place by setting the keyword inplace = True inside the dropna() function.

Return to the Top

Filling NaN values

```
# We replace all NaN values with 0
store_items.fillna(0)
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	0.0
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	0.0	10	0.0	4.0

Return to the Top

.fillna(method = 'ffill', axis)

ffill method to replace NaN values using the previous known value along the given axis

```
store_items
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	NaN
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	NaN	10	NaN	4.0

```
store_items.fillna(method = 'ffill', axis = 0)#replaces with prevoius rows
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	NaN
store 2	15	5	10	2.0	5	7.0	50.0

	bikes	pants	watches	shirts	shoes	suits	glasses
store 3	20	30	35	2.0	10	7.0	4.0

```
store_items.fillna(method = 'ffill', axis = 1)#replaces with previous columns
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20.0	30.0	35.0	15.0	8.0	45.0	45.0
store 2	15.0	5.0	10.0	2.0	5.0	7.0	50.0
store 3	20.0	30.0	35.0	35.0	10.0	10.0	4.0

Return to the Top

.fillna(method = 'backfill', axis)

backfill method to replace NaN values using the next known value along the given axis

```
store_items
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	NaN
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	NaN	10	NaN	4.0

```
# We replace NaN values with the next value in the column
store_items.fillna(method = 'backfill', axis = 0)
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20	30	35	15.0	8	45.0	50.0
store 2	15	5	10	2.0	5	7.0	50.0
store 3	20	30	35	NaN	10	NaN	4.0

```

```

```
# We replace NaN values with the next value in the row
store_items.fillna(method = 'backfill', axis = 1)
```

	bikes	pants	watches	shirts	shoes	suits	glasses
store 1	20.0	30.0	35.0	15.0	8.0	45.0	NaN
store 2	15.0	5.0	10.0	2.0	5.0	7.0	50.0
store 3	20.0	30.0	35.0	10.0	10.0	4.0	4.0

Note we can always replace the NaN values in place by setting the keyword inplace = True inside the fillna() function.

Chapter 5 Exploratory Data Analysis

Exploratory Data Analysis is nothing but the complete film of the data. When we watch the characters in film, we get to understand the relations between them. How to link the characters to the story of film so that it will never go off track is very important to imagine for a writer. That curious imagination is required for the data engineer to explore the dataset.

What is Exploratory Data Analysis ?

When we collect the dataset from predefined source then that dataset may be imbalanced containing features that are not required, or null values. Then it's obviously become complex to process the data and predict results using machine learning model. EDA helps us to restructure the imbalanced data into balanced data using necessary functions. So when we have our structured balanced dataset then our machine learning model will preform very well and we can predict more accurate results.

Exploratory Data Analysis is the technique involving importation of data, analyzing data to obtain meaningful insights, data cleaning, sorting the data, finding correlation between data features, mapping of data feature values and data visualization. Later on, we will discuss everything one by one.

Return to the Top

Import Python Libraries

ou have to import all the necessary python libraries.

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

Load dataset

```
#Read csv file
df = pd.read_csv("Salaries.csv")
```

Note that in this case, you made use of read_csv() because the data happens to be in a comma-separated format. If you have files that have another separator, you can also consider using other functions to load in your data, such as read_table(), read_excel(), read_fwf() and read_clipboard, to read in general delimited files, Excel files, Fixed-Width Formatted data and data that was copied to the Clipboard, respectively.

Return to the Top

Display first five records

```
#Display a few first records
df.head()
```

```
 rank discipline phd service sex salary
0 Prof B 56 49 Male 186960.0
1 Prof A 12 6 Male 93000.0
2 Prof A 23 20 Male 110515.0
3 Prof A 40 31 Male NaN
4 Prof B 20 18 Male 104800.0
```

This head(n) function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it. By default it will show 5 rows.

Display the last 5 records

```
#Display the last 5 records
df.tail()
```

```
    rank discipline phd service sex salary
```

```
73 Prof     B 18 10 Female 105450.0
74 AssocProf B 19 6 Female 104542.0
75 Prof     B 17 17 Female 124312.0
76 Prof     A 28 14 Female 109954.0
77 Prof     A 23 15 Female 109646.0
```

This tail(n) function returns the last n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it. By default it will show 5 rows.

Return to the Top

Identify type of df object

```
#Identify the type of df object
type(df)
```

```
pandas.core.frame.DataFrame
```

This type(DataFrame) funtion is used to identify the type of dataframe object.

```
#Check the type of a column "salary"
df['salary'].dtype
```

```
dtype('float64')
```

```
#List the types of all columns
df.dtypes
```

```
rank           object
discipline     object
phd             int64
service         int64
sex            object
salary        float64
dtype: object
```

Using df.dtypes, we get the list of types of all the columns in our dataset.

Return to the Top

List the column names

```
#List the column names
df.columns
```

```
Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='object')
```

Using df.columns, we get the name of all columns(data features) in dataset.

Display dimensions of dataframe

```
#Number of rows and columns
df.shape
```

This .shape function gives us the number of rows and columns of the dataset.

Display statistics for numeric columns

```
#Output basic statistics for the numeric columns
df.describe()
```

```
         phd     service      salary
count 78.000000 78.000000 77.000000
mean 19.705128 15.051282 107722.727273
std   12.498425 12.139768 28353.167204
min     1.000000 0.000000 57800.000000
25%     10.250000 5.250000 88000.000000
50%     18.500000 14.500000 104542.000000
75%     27.750000 20.750000 126300.000000
max     56.000000 51.000000 186960.00000
```

When it comes to numerical attributes, we must start by observing several statistics like count, mean, standard deviation, etc. Here you will see at few of those for all our numerical categories.

Return to the Top

Display number of null values

```
#Display number of null values in every column in dataset
df.isnull().sum()
```

```
rank          0
discipline    0
phd           0
service       0
sex           0
salary        1
dtype: int6
```

Return to the Top

Find pairwise correlation

Pandas dataframe.corr() is used to find the pairwise correlation of all columns in the dataframe.Any na values are automatically excluded. For any non-numeric data type columns in the dataframe it is ignored.

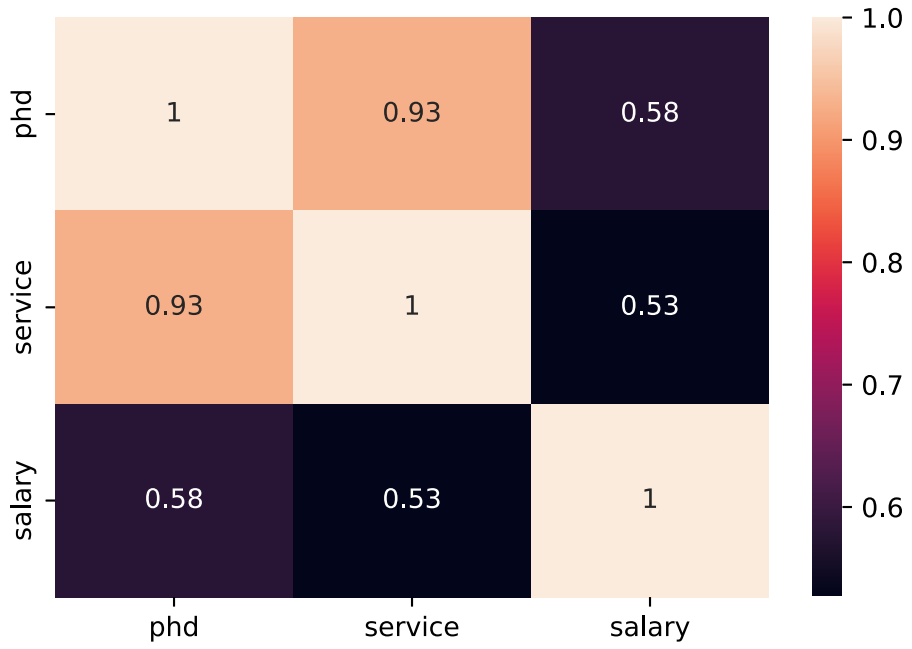
```
df.corr()
```

```
          phd      service salary
phd     1.000000 0.927170 0.576446
service 0.927170 1.000000 0.526343
salary 0.576446 0.526343 1.000000
```

Correlation analysis is used to quantify the degree to which two variables are related. Through the correlation analysis, you evaluate correlation coefficient that tells you how much one variable changes when the other one does. Correlation analysis provides you with a linear relationship between two variables.

Heatmap

```
corrMatrix = df.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```



Correlation analysis is used to quantify the degree to which two variables are related. Through the correlation analysis, you evaluate the correlation coefficient that tells you how much one variable changes when the other one does. Correlation analysis provides you with a linear relationship between two variables.

When we correlate feature variables with the target variable, we get to know that how much dependency is there between particular feature variables and the target variables.

Return to the Top

Calculate mean

```
#Calculate mean for all numeric columns
df.mean()
```

```
phd            19.705128
service        15.051282
salary     107722.727273
dtype: float64
```

As you know .describe() function gives all statistics but .mean() can also be used to calculate mean for all numeric columns.

value_counts() function

```
#Return a Series containing counts of unique values.
df['sex'].value_counts()
```

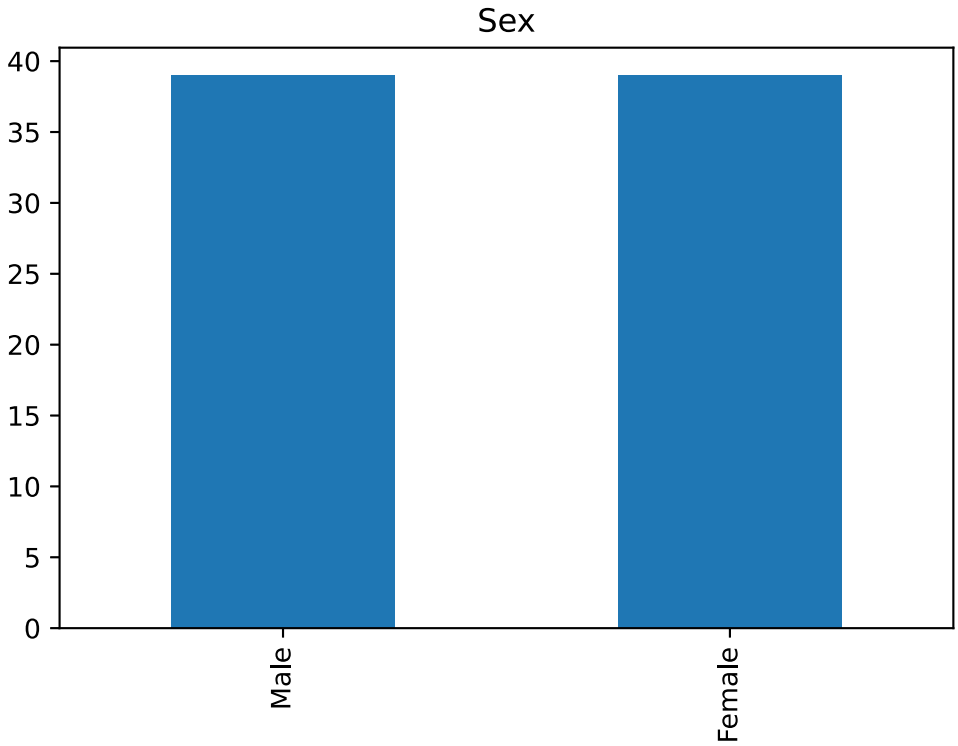
```
Male      39
Female    39
Name: sex, dtype: int64
```

```
#(normalize=True) means the object returned will contain the relative frequencies of
the unique values.
df['sex'].value_counts(normalize=True)
```

```
Male      0.5
Female    0.5
Name: sex, dtype: float64
```

Return to the Top

```
#plot.bar(title='') function is used to plot bargraph with title mentioned in
brackets with respect to the unique values.
df['sex'].value_counts().plot.bar(title='Sex')
```



1. This .value_counts() function returns a Series containing counts of unique values.
2. (normalize=True) means the object returned will contain the relative frequencies of the unique values.
3. plot.bar() function is used to plot bar graph with title mentioned in brackets with respect to the unique values.

Copy dataframe

```
#Using copy() function dataframe df is copied into dataframe df_copy
df_copy = df.copy()
#insert the rank column located in dataframe df_copy in temporary dataframe add
add = df_copy['rank']
df_copy.head()
```

```
    rank discipline phd service sex salary
0 Prof B 56 49 Male 186960.0
1 Prof A 12 6 Male 93000.0
2 Prof A 23 20 Male 110515.0
3 Prof A 40 31 Male NaN
4 Prof B 20 18 Male 104800.0
```

Using .copy() function, we copy one dataframe into another dataframe.

Return to the Top

Drop features

```
#By using drop() funtion, we drop column 'rank' & 'phd' from the dataframe df_copy
df_copy = df_copy.drop(['rank','phd'], axis = 1)
df_copy.head()
```

```
discipline service sex salary
0 B 49 Male 186960.0
1 A 6 Male 93000.0
2 A 20 Male 110515.0
3 A 31 Male NaN
4 B 18 Male 104800.0
```

When a feature has no significance or effect on the variation of the target variable, then obviously for better predictions we choose to drop that column. By using .drop() function we can drop the columns from current dataframe.

```
#As you can see 'rank' & 'phd' columns are not present.
df_copy.columns
```

```
Index(['discipline', 'service', 'sex', 'salary'], dtype='object')
```

Return to the Top

Replace values in column

```
#By replace() function we replace the value 'A' with 'D' in discipline column.
df_copy["discipline"] = df_copy["discipline"].replace('A','D')
df_copy.head()
```

```
  discipline service sex salary rank
0 B 49 Male 186960.000000 Prof
1 D 6 Male 93000.000000 Prof
2 D 20 Male 110515.000000 Prof
3 D 31 Male 107722.727273 Prof
4 B 18 Male 104800.000000 Prof
```

Fill null values with mean of column

```
#calculate mean of a specific column.
df_copy['salary'].mean()
```

107722.72727272726

```
#salary column has a NaN value in the 4th row,
#so using fillna() function we fill the value with mean by calculating mean value of
that column.
df_copy['salary'] = df_copy['salary'].fillna(df_copy['salary'].mean())
df_copy.head()
```

```
  discipline service sex salary rank
0 B 49 Male 186960.000000 Prof
1 D 6 Male 93000.000000 Prof
2 D 20 Male 110515.000000 Prof
3 D 31 Male 107722.727273 Prof
4 B 18 Male 104800.000000 Prof
```

When we have a certain missing numeric value in the dataset, we try to fill that values by taking the mean() of that particular column. You can fill with the missing values with median() also.

Return to the Top

Data slicing and grouping

A groupby operation involves some combination of splitting the object, applying a function, and combining the results. This can be used to group large amounts of data and compute operations on these groups.

```
df_sex = df.groupby('sex')
```

```
#Extract a column by name (method 1)
df['sex'].head()
```

```
0    Male
1    Male
2    Male
3    Male
4    Male
Name: sex, dtype: object
```

```
#Extract a column name (method 2)
df.sex.head()
```

```
0    Male
1    Male
2    Male
3    Male
4    Male
Name: sex, dtype: object
```

```
#Calculate the basic statistics for the salary column (used describe() method)
df['salary'].describe()
```

```
count        77.000000
mean     107722.727273
std       28353.167204
min       57800.000000
25%       88000.000000
50%      104542.000000
75%      126300.000000
max      186960.000000
Name: salary, dtype: float64
```

```
#Calculate how many values in the salary column (use count() method)
df['salary'].count()
```

77

```
#Calculate the average salary
df['salary'].mean()
```

107722.72727272728

```
#Group data using rank
df_rank = df.groupby('rank')
```

```
#Calculate mean of all numeric columns for the grouped object
df_rank.mean()
```

```
             phd     service      salary
rank
AssocProf 15.076923 11.307692 91786.230769
AsstProf 5.052632 2.210526 81362.789474
Prof        27.065217 21.413043 123456.355556
```

```
#Calculate the mean salary for men and women. The following produce Pandas Series
(single brackets around salary)
df.groupby('sex')['salary'].mean()
```

```
sex
Female    101002.410256
Male      114619.894737
Name: salary, dtype: float64
```

```
# If we use double brackets Pandas will produce a DataFrame
df.groupby('sex')[['salary']].mean()
```

```
         salary
sex
Female 101002.410256
Male 114619.894737
```

```
# Group using 2 variables - sex and rank:
df.groupby(['sex','rank'], sort=False)[['salary']].mean()
```

```
sex     rank       salary
Male Prof     124448.851852
        AssocProf 102697.666667
        AsstProf 85918.000000
Female Prof     121967.611111
        AssocProf 88512.800000
        AsstProf 78049.909091
```

```
# Group data by the discipline and find the average salary for each group
df.groupby('discipline')['salary'].mean()
```

```
discipline
A     97391.857143
B    116331.785714
Name: salary, dtype: float64
```

Return to the Top

Filtering

```
#Select observation with the value in the salary column > 120K
df_sub = df[ df['salary'] > 120000]
df_sub.head()
```

```
    rank discipline phd service sex salary
0 Prof B 56 49 Male 186960.0
5 Prof A 20 20 Male 122400.0
7 Prof A 18 18 Male 126300.0
10 Prof B 39 33 Male 128250.0
11 Prof B 23 23 Male 134778.0
```

```
#Select data for female professors
df_w = df[ df['sex'] == 'Female']
df_w.head()
```

```
    rank discipline phd service sex salary
39 Prof B 18 18 Female 129000.0
40 Prof A 39 36 Female 137000.0
41 AssocProf A 13 8 Female 74830.0
42 AsstProf B 4 2 Female 80225.0
43 AsstProf B 5 0 Female 77000.0
```

```
# Using filtering, find the mean value of the salary for the discipline A
df[df['discipline'] == 'A']['salary'].mean()
```

97391.85714285714

```
# Extract (filter) only observations with high salary ( > 100K) and find how many
female and male professors in each group
df[df['salary'] > 120000].groupby('sex')['salary'].count()
```

```
sex
Female     9
Male      15
Name: salary, dtype: int64
```

Return to the Top

More on slicing the dataset

```
#Select column salary
df1 = df['salary']
```

```
#Check data type of the result
type(df1)
```

pandas.core.series.Series

```
#Look at the first few elements of the output
df1.head()
```

```
0    186960.0
1     93000.0
2    110515.0
3         NaN
4    104800.0
Name: salary, dtype: float64
```

```
#Select column salary and make the output to be a data frame
df2 = df[['salary']]
```

```
#Check the type
type(df2)
```

pandas.core.frame.DataFrame

```
#Select a subset of rows (based on their position):
# Note 1: The location of the first row is 0
# Note 2: The last value in the range is not included
df[0:10]
```

```
 rank discipline phd service sex salary
0 Prof     B 56 49 Male 186960.0
1 Prof     A 12 6 Male 93000.0
2 Prof     A 23 20 Male 110515.0
3 Prof     A 40 31 Male NaN
4 Prof     B 20 18 Male 104800.0
5 Prof     A 20 20 Male 122400.0
6 AssocProf A 20 17 Male 81285.0
7 Prof     A 18 18 Male 126300.0
8 Prof     A 29 19 Male 94350.0
9 Prof     A 51 51 Male 57800.0
```

```
#If we want to select both rows and columns we can use method .loc
df.loc[10:20,['rank', 'sex','salary']]
```

```
    rank     sex     salary
10 Prof     Male 128250.0
11 Prof     Male 134778.0
12 AsstProf Male 88000.0
13 Prof     Male 162200.0
14 Prof     Male 153750.0
15 Prof     Male 150480.0
16 AsstProf Male 75044.0
17 AsstProf Male 92000.0
18 Prof     Male 107300.0
19 Prof     Male 150500.0
20 AsstProf Male 92000.0
```

```
#Let's see what we get for our df_sub data frame
# Method .loc subset the data frame based on the labels:
df_sub.loc[10:20,['rank','sex','salary']]
```

```
 rank sex      salary
10 Prof Male 128250.0
11 Prof Male 134778.0
13 Prof Male 162200.0
14 Prof Male 153750.0
15 Prof Male 150480.0
19 Prof Male 150500.0
```

```
#  Unlike method .loc, method iloc selects rows (and columns) by poistion:
df_sub.iloc[10:20, [0,3,4,5]]
```

```
 rank service sex salary
27 Prof 43 Male 155865.0
29 Prof 20 Male 123683.0
31 Prof 21 Male 155750.0
35 Prof 23 Male 126933.0
36 Prof 45 Male 146856.0
39 Prof 18 Female 129000.0
```

```
40 Prof 36 Female 137000.0
44 Prof 19 Female 151768.0
45 Prof 25 Female 140096.0
49 Prof 18 Female 122960.0
```

Return to the Top

Sorting the data

```
#Sort the data frame by yrs.service and create a new data frame
df_sorted = df.sort_values(by = 'service')
df_sorted.head()
```

```
 rank discipline phd service sex salary
55 AsstProf A 2 0 Female 72500.0
23 AsstProf A 2 0 Male 85000.0
43 AsstProf B 5 0 Female 77000.0
17 AsstProf B 4 0 Male 92000.0
12 AsstProf B 1 0 Male 88000.
```

```
#Sort the data frame by yrs.service and overwrite the original dataset
df.sort_values(by = 'service', ascending = False, inplace = True)
df.head()
```

```
   rank discipline phd service sex salary
9 Prof A 51 51 Male 57800.0
0 Prof B 56 49 Male 186960.0
36 Prof B 45 45 Male 146856.0
27 Prof A 45 43 Male 155865.0
40 Prof A 39 36 Female 137000.0
```

Here, you can see that we sort the dataframe by ‘service’ using the .sort_values() function. In the second code snippet, we give the attribute ‘ascending= False’ that means, it will show the records in the descending order as ‘service=prof’ is emerged at first and then the rest of all.

```
# Restore the original order (by sorting using index)
df.sort_index(axis=0, ascending = True, inplace = True)
df.head()
```

```
   rank discipline phd service sex salary
0 Prof B 56 49 Male 186960.0
1 Prof A 12 6 Male 93000.0
2 Prof A 23 20 Male 110515.0
3 Prof A 40 31 Male NaN
4 Prof B 20 18 Male 104800.0
```

```
# Sort data frame by the salary (in descending order) and display the first few
records of the output (head)
df.sort_values(by='salary', ascending=False).head()
```

```
    rank discipline phd service sex salary
0 Prof B 56 49 Male 186960.0
13 Prof B 35 33 Male 162200.0
72 Prof B 24 15 Female 161101.0
27 Prof A 45 43 Male 155865.0
31 Prof B 22 21 Male 155750.0
```

```
#Sort the data frame using 2 or more columns:
df_sorted = df.sort_values(by = ['service', 'salary'], ascending = [True,False])
df_sorted.head(10)
```

```
    rank discipline phd service sex salary
52 Prof A 12 0 Female 105000.0
17 AsstProf B 4 0 Male 92000.0
12 AsstProf B 1 0 Male 88000.0
23 AsstProf A 2 0 Male 85000.0
43 AsstProf B 5 0 Female 77000.0
55 AsstProf A 2 0 Female 72500.0
57 AsstProf A 3 1 Female 72500.0
28 AsstProf B 7 2 Male 91300.0
42 AsstProf B 4 2 Female 80225.0
68 AsstProf A 4 2 Female 77500.0
```

Return to the Top

Mapping of data feature values

```
df['discipline'] = df['discipline'].map({'A':1,'B':2})
df['discipline'].head(10)
```

```
0    2
1    1
2    1
3    1
4    2
5    1
6    1
7    1
8    1
9    1
Name: discipline, dtype: int64
```

If a certain column in the dataset has categorical values and for any model training we want to convert it into numeric type then we must use the .map() function and allocate the desired numeric values to the categorical values as you can see from the above example. It will help you in predicting better results.

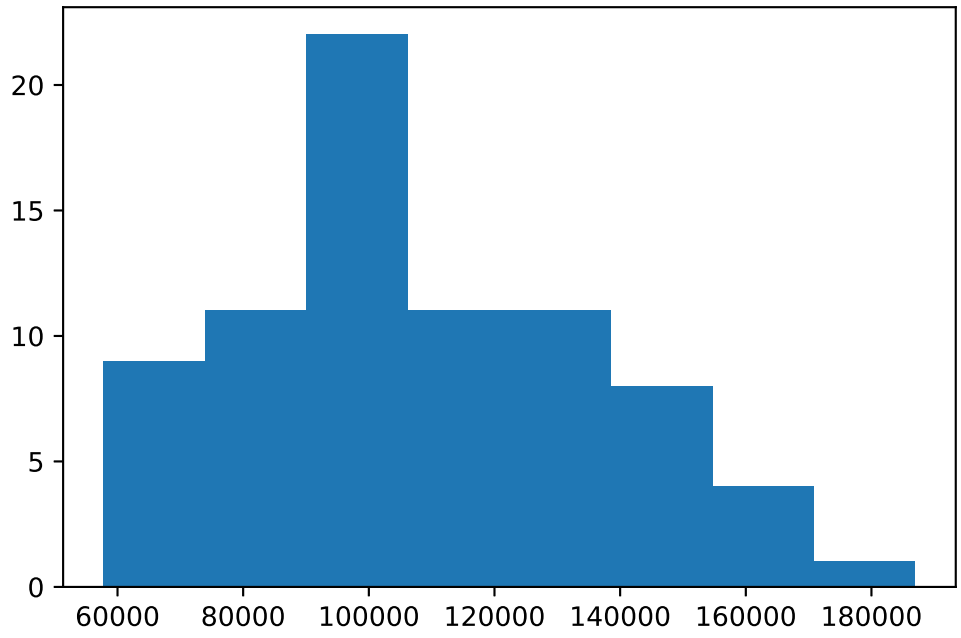
Return to the Top

Data Visualizations

The best way to gain insights from the data set is a visualization of that data in the form of histograms, bar graphs, scatter plots. Mainly in python, we have two libraries that are seaborn and matplotlib. Using these libraries we can visualize the data and can find the best patterns among the features in the data.

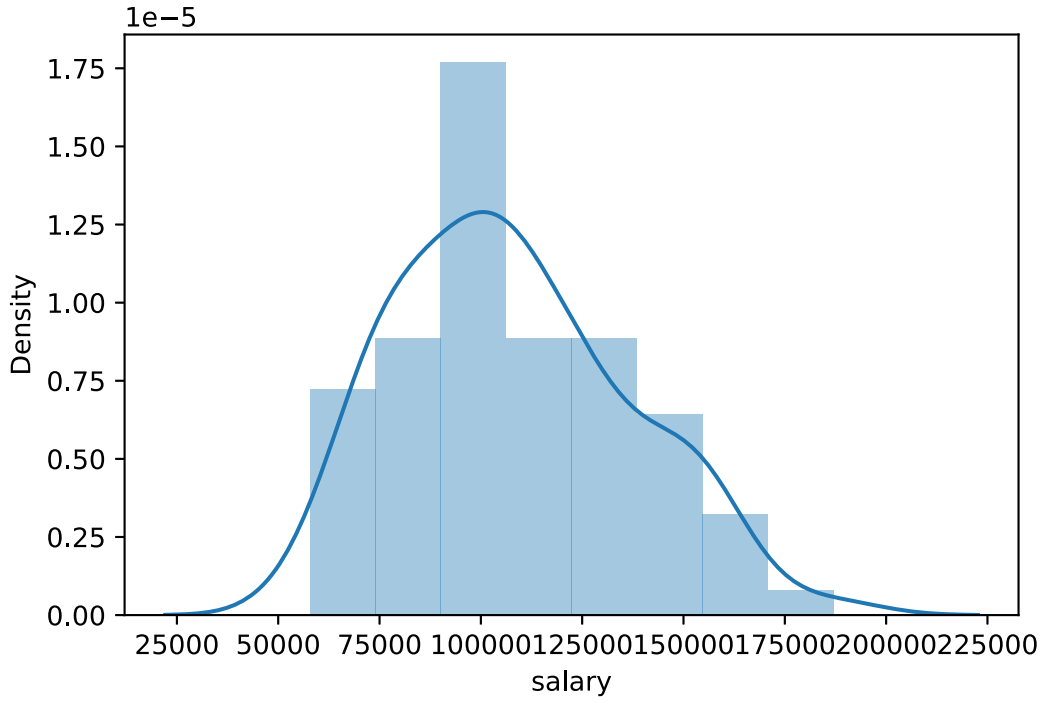
Let's start by plotting a histogram for the 'salary' attributes. As with the same syntax, we can plot histograms from every attribute.

```
#Use matplotlib to draw a histogram of a salary data
plt.hist(df['salary'],bins=8, )
```



Return to the Top

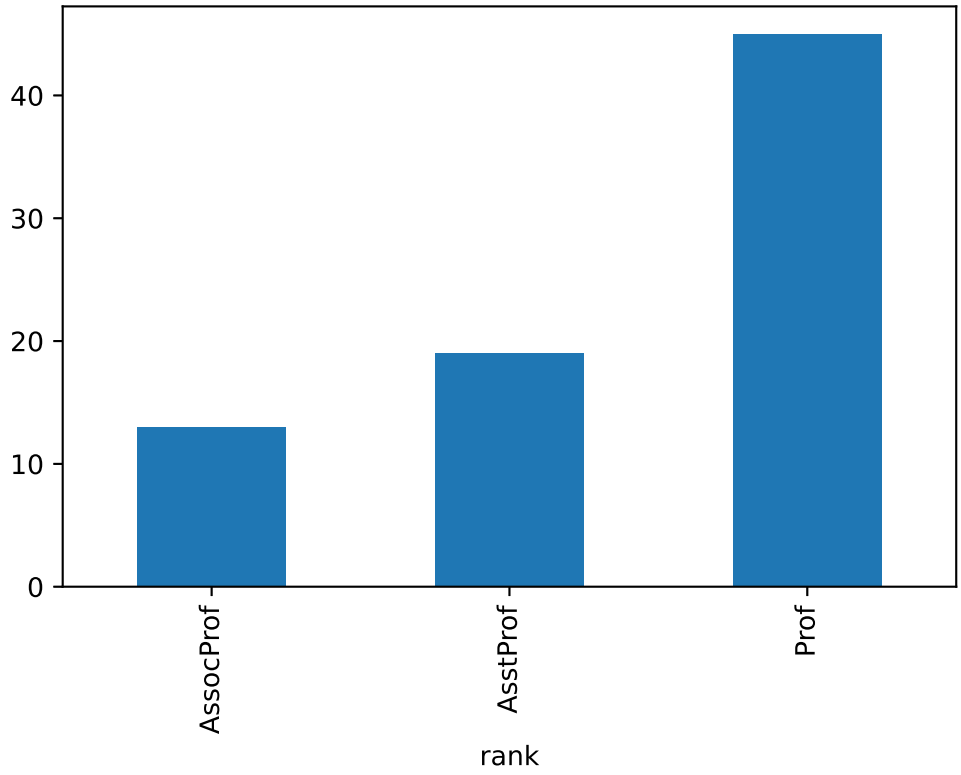
```
#Use seaborn package to draw a histogram
sns.distplot(df['salary']);
```



Pandas .groupby() function is used to split the data into groups based on the dependencies between them. In below we can see a bar plot showing the salary with respect to rank.

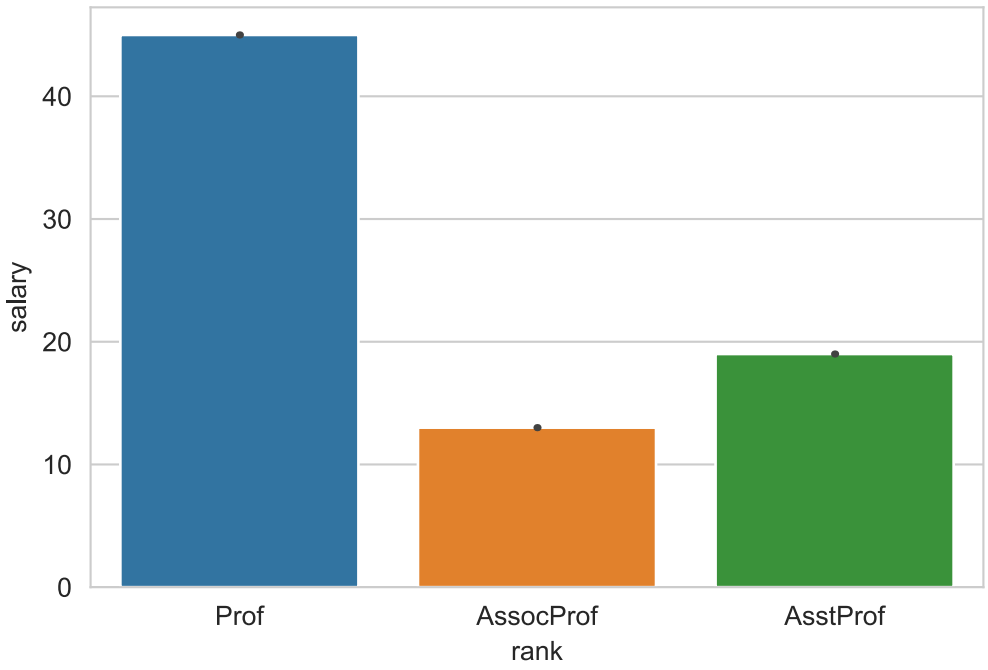
Return to the Top

```
# Use regular matplotlib function to display a barplot
df.groupby(['rank'])['salary'].count().plot(kind='bar')
```



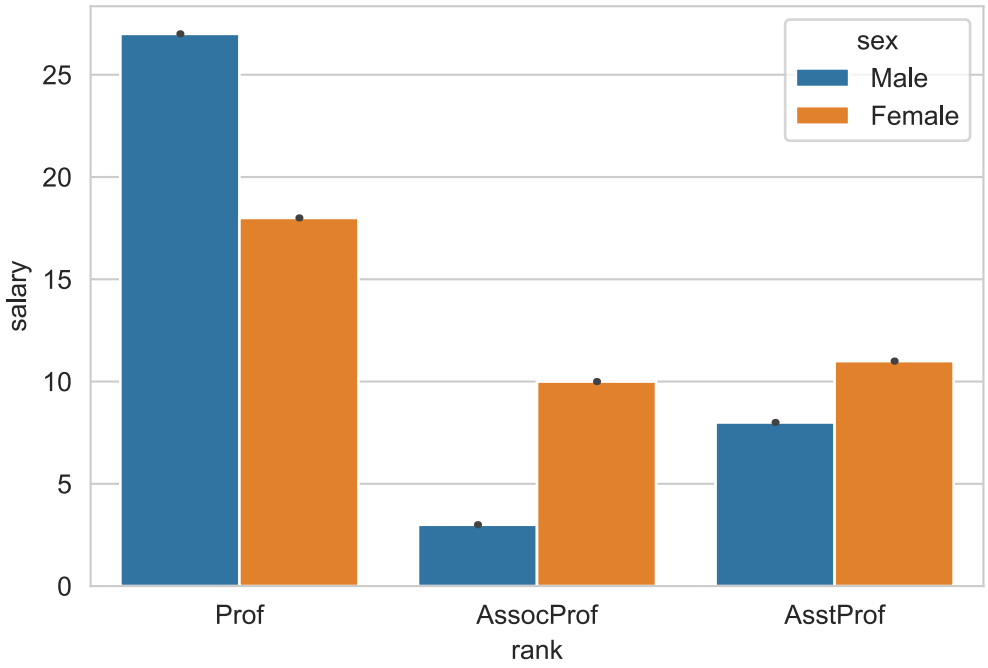
Return to the Top

```
# Use seaborn package to display a barplot
sns.set_style("whitegrid")

ax = sns.barplot(x='rank',y ='salary', data=df, estimator=len)
```



Return to the Top

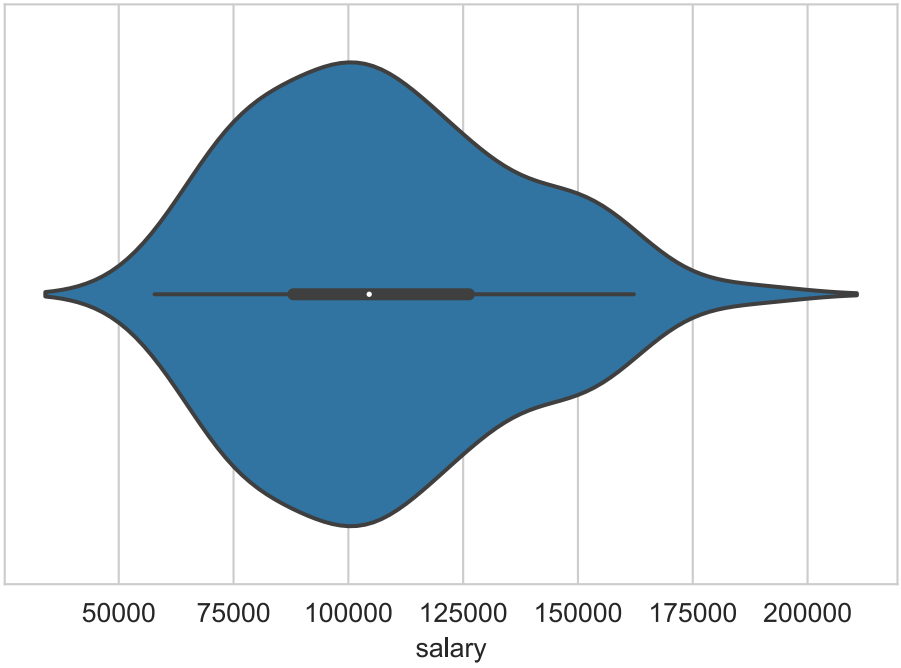
```
# Split into 2 groups:
ax = sns.barplot(x='rank',y ='salary', hue='sex', data=df, estimator=len)
```



When we are interested in a linear relationship between two numeric attributes we plot regplot which is best for future predictions.

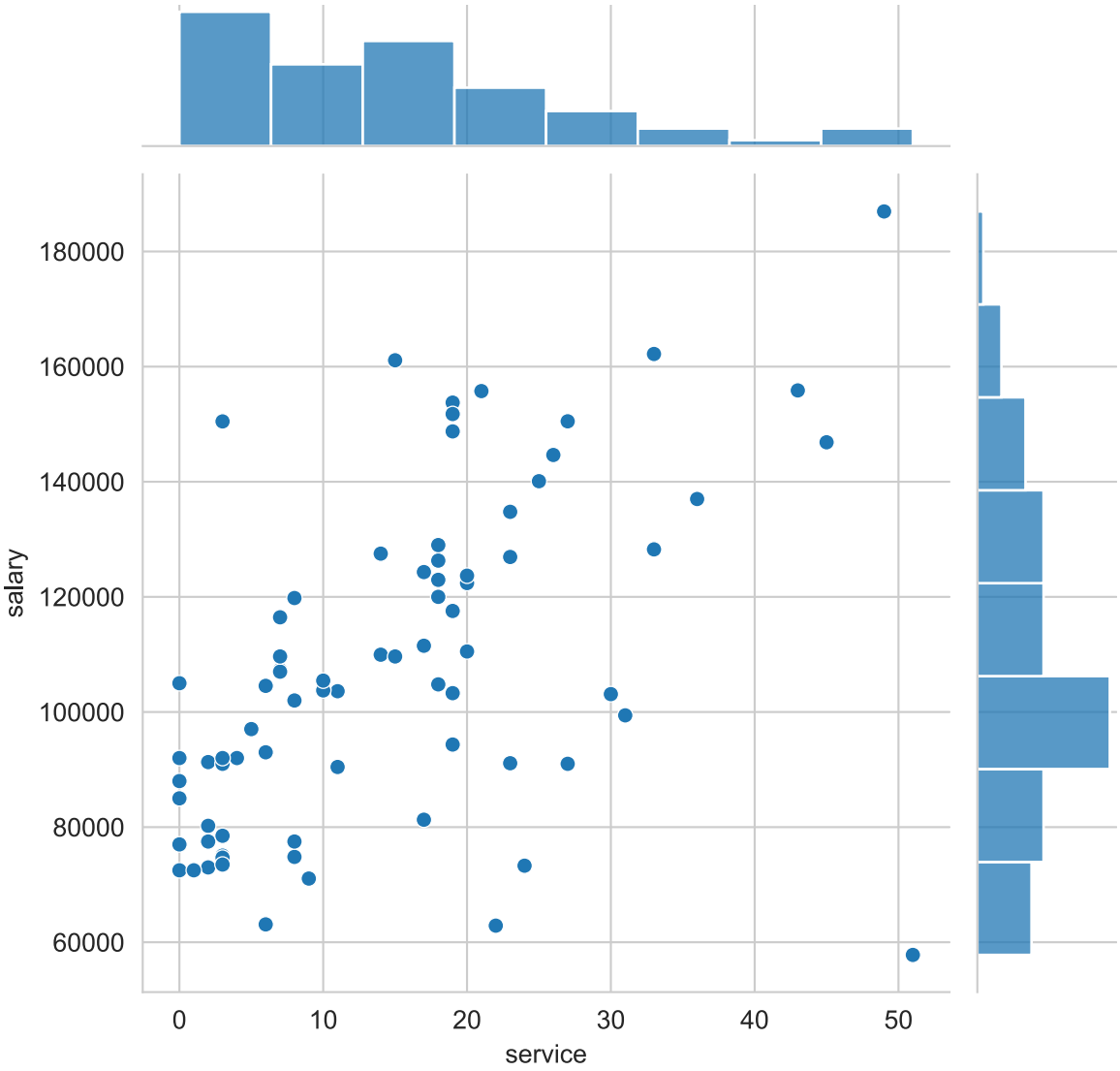
Return to the Top

```
#Violinplot
sns.violinplot(x = "salary", data=df)
```



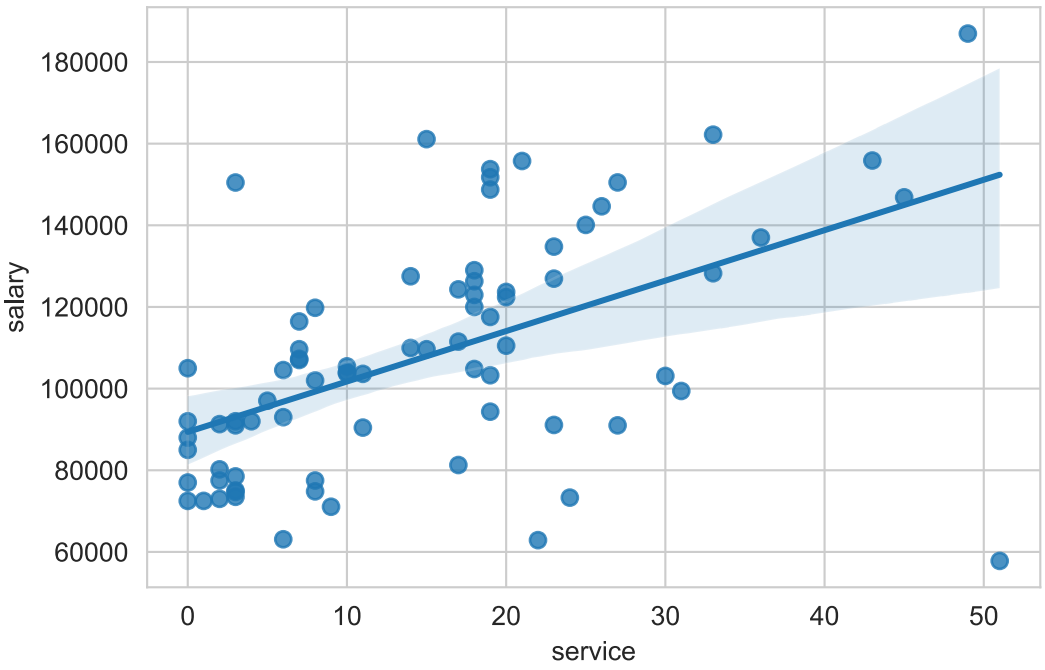
Return to the Top

```
#Scatterplot in seaborn
sns.jointplot(x='service', y='salary', data=df)
```



Return to the Top

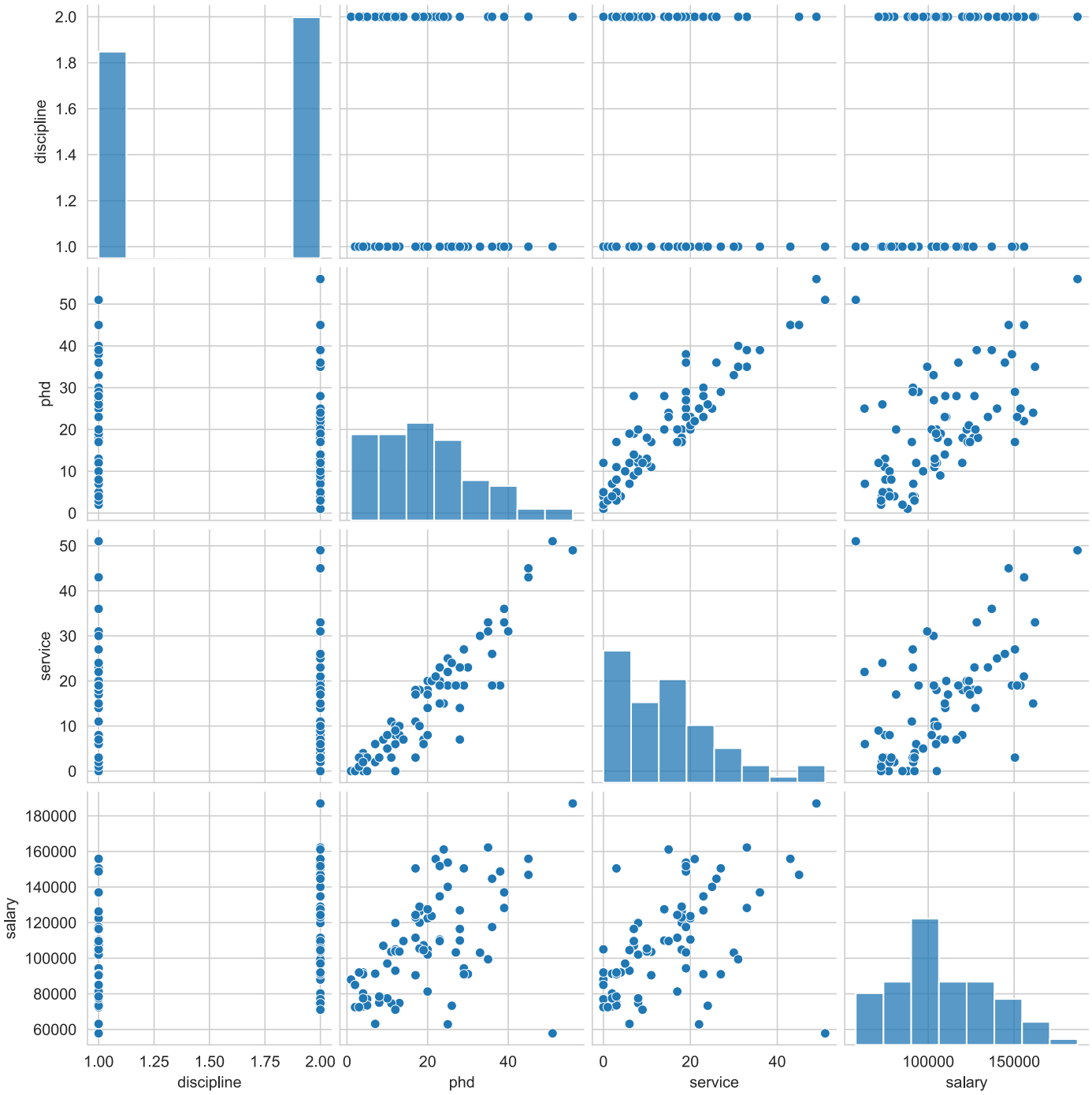
```
#If we are interested in linear regression plot for 2 numeric variables we can use regplot
sns.regplot(x='service', y='salary', data=df)
```



Let's complete the visualization by plotting the histogram for all the numeric features so that you can see all the variations in one feature with respect to other features. This visualization is very necessary for understanding the data feature's nature.

Return to the Top

```
# Pairplot
sns.pairplot(df)
```



Let's summarize, we performed many data processing and cleaning techniques from importing data to data slicing then filling missing data then mapping then finding statistics and data visualizations.

Exploratory Data Analysis concludes each and every function or you can say a step to gain powerful insights from the data. It is an important step in analyzing the data.

Chapter 6 Markdown

Headings

> To create a heading, add number signs (#) in front of a word or phrase. The number of number signs you use should correspond to the heading level. For example, to create a heading level three (< h3 >), use three number signs (e.g., ### My Header).

```
# Heading level 1
## Heading level 2
### Heading level 3
#### Heading level 4
##### Heading level 5
###### Heading level 6
```

Return to the Top

Paragraphs

> To create paragraphs, use a blank line to separate one or more lines of text.

I really like using Markdown.

I think I'll use it to format all of my documents from now on.

Return to the Top

Line Breaks

> To create a line break or new line (< br >), end a line with two or more spaces, and then type return.

This is the first line.
And this is the second line.

Return to the Top

Emphasis

> You can add emphasis by making text bold or italic.

Bold

> To bold text, add two asterisks or underscores before and after a word or phrase. To bold the middle of a word for emphasis, add two asterisks without spaces around the letters.

```
I just love **bold text**.
I just love __bold text__.
Love **is** bold
```

I just love **bold text**.
I just love **bold text**.
Love **is** bold

Italic

> To italicize text, add one asterisk or underscore before and after a word or phrase. To italicize the middle of a word for emphasis, add one asterisk without spaces around the letters.

```
Italicized text is the *cat's meow*.
Italicized text is the _cat's meow_.
A *cat* meow.
```

Italicized text is the *cat's meow*.
Italicized text is the *cat's meow*.
A *cat* meow.

Bold and Italic

> To emphasize text with bold and italics at the same time, add three asterisks or underscores before and after a word or phrase. To bold and italicize the middle of a word for emphasis, add three asterisks without spaces around the letters.

```
This text is __*really important*__.
This text is **_really important_**.
This is really ***very*** important text.
```

This text is ***really important***.
This text is ***really important***.
This is really ***very*** important text.

Return to the Top

Blockquotes

To create a blockquote, add a > in front of a paragraph.

> Dorothy followed her through many of the beautiful rooms in her castle.

Blockquotes with Multiple Paragraphs

Blockquotes can contain multiple paragraphs. Add a > on the blank lines between the paragraphs.

```
> Dorothy followed her through many of the beautiful rooms in her castle.
>
> The Witch bade her clean the pots and kettles and sweep the floor and keep the
fire fed with wood.
```

> Dorothy followed her through many of the beautiful rooms in her castle.
>
> The Witch bade her clean the pots and kettles and sweep the floor and keep the fire fed with wood.

Nested Blockquotes

Blockquotes can be nested. Add a >> in front of the paragraph you want to nest.

```
> Dorothy followed her through many of the beautiful rooms in her castle.
>
>> The Witch bade her clean the pots and kettles and sweep the floor and keep the
fire fed with wood.
```

> Dorothy followed her through many of the beautiful rooms in her castle.
>
> > The Witch bade her clean the pots and kettles and sweep the floor and keep the fire fed with wood.

Blockquotes with Other Elements

Blockquotes can contain other Markdown formatted elements. Not all elements can be used — you'll need to experiment to see which ones work.

```
> #### The quarterly results look great!
>
> - Revenue was off the chart.
> - Profits were higher than ever.
>
>  *Everything* is going according to **plan**.
```

> #### The quarterly results look great!
>
> - Revenue was off the chart.
> - Profits were higher than ever.
>
> *Everything* is going according to **plan**.

Return to the Top

Lists

> You can organize items into ordered and unordered lists.

Ordered Lists

> To create an ordered list, add line items with numbers followed by periods. The numbers don't have to be in numerical order, but the list should start with the number one.

```
1. First item
2. Second item
3. Third item
4. Fourth item
>
1. First item
1. Second item
1. Third item
1. Fourth item
>
1. First item
2. Second item
3. Third item
    1. Indented item
    2. Indented item
4. Fourth item
```

1. First item
2. Second item
3. Third item
4. Fourth item

1. First item
2. Second item
3. Third item
4. Fourth item

1. First item
2. Second item
3. Third item
    1. Indented item
    2. Indented item
4. Fourth item

Unordered Lists

> To create an unordered list, add dashes (-), asterisks (*), or plus signs (+) in front of line items. Indent one or more items to create a nested list.

```
- First item
- Second item
- Third item
- Fourth item
>
* First item
* Second item
* Third item
* Fourth item
>
+ First item
+ Second item
+ Third item
+ Fourth item
>
- First item
- Second item
- Third item
    - Indented item
    - Indented item
- Fourth item
```

- First item
- Second item
- Third item
- Fourth item

- First item
- Second item
- Third item
- Fourth item

- First item
- Second item
- Third item
- Fourth item

- First item
- Second item
- Third item
    - Indented item
    - Indented item
- Fourth item

Starting Unordered List Items With Numbers

> If you need to start an unordered list item with a number followed by a period, you can use a backslash (\ ) to escape the period.

```
- 1968\. A great year!
- I think 1969 was second best
```

- 1968. A great year!
- I think 1969 was second best.

Return to the Top

Adding Elements in Lists

> To add another element in a list while preserving the continuity of the list, indent the element four spaces or one tab, as shown in the following examples.

```
* This is the first list item.
* Here's the second list item.

    I need to add another paragraph below the second list item.

* And here's the third list item.
```

- This is the first list item.
- Here's the second list item.

  I need to add another paragraph below the second list item.

- And here's the third list item.

Return to the Top

Code Blocks

> Code blocks are normally indented four spaces or one tab. When they’re in a list, indent them eight spaces or two tabs.

```
1. Open the file.
2. Find the following code block on line 21:

        <html>
```

```
            <head>
              <title>Test</title>
            </head>

  3. Update the title to match the name of your website.
```

1. Open the file.
2. Find the following code block on line 21:

```
<html>
  <head>
    <title>Test</title>
  </head>
```

3. Update the title to match the name of your website.

Return to the Top

Images

```
1. Open the file containing the Linux mascot.
2. Marvel at its beauty.

    ![Tux, the Linux mascot](Markdown/tux.png)

3. Close the file.
```

1. Open the file containing the Linux mascot.
2. Marvel at its beauty.

3. Close the file.

Return to the Top

Code

> To denote a word or phrase as code, enclose it in backticks (`).

```
At the command prompt, type `nano`.
```

At the command prompt, type `nano`.

Escaping Backticks

> If the word or phrase you want to denote as code includes one or more backticks, you can escape it by enclosing the word or phrase in double backticks (``).

```
``Use `code` in your Markdown file.``
```

`Use `code` in your Markdown file.`

Return to the Top

Horizontal Rules

> To create a horizontal rule, use three or more asterisks (***), dashes (---), or underscores (___) on a line by themselves.

```
***

---

_________________
```

Return to the Top

Links

> To create a link, enclose the link text in brackets (e.g., [Duck Duck Go]) and then follow it immediately with the URL in parentheses (e.g., (https://duckduckgo.com)).

```
My favorite search engine is [Duck Duck Go](https://duckduckgo.com).
```

My favorite search engine is Duck Duck Go.

Adding Titles

> You can optionally add a title for a link. This will appear as a tooltip when the user hovers over the link. To add a title, enclose it in parentheses after the URL.

```
My favorite search engine is [Duck Duck Go](https://duckduckgo.com "The best search engine for privacy").
```

My favorite search engine is Duck Duck Go.

URLs and Email Addresses

> To quickly turn a URL or email address into a link, enclose it in angle brackets.

```
<https://www.markdownguide.org>
<fake@example.com>
```

https://www.markdownguide.org
fake@example.com

Formatting Links

> To emphasize links, add asterisks before and after the brackets and parentheses. To denote links as code, add backticks in the brackets.

```
I love supporting the **[EFF](https://eff.org)**.
This is the *[Markdown Guide](https://www.markdownguide.org)*.
See the section on [`code`](#code).
```

I love supporting the **EFF**.
This is the *Markdown Guide*.
See the section on `code`.

Reference-style Links

> Reference-style links are a special kind of link that make URLs easier to display and read in Markdown. Reference-style links are constructed in two parts: the part you keep inline with your text and the part you store somewhere else in the file to keep the text easy to read.

```
In a hole in the ground there lived a hobbit. Not a nasty, dirty, wet hole, filled
with the ends of worms and an oozy smell, nor yet a dry, bare, sandy hole with
nothing in it to sit down on or to eat: it was a [hobbit-hole]
(https://en.wikipedia.org/wiki/Hobbit#Lifestyle "Hobbit lifestyles"), and that means
comfort.

In a hole in the ground there lived a hobbit. Not a nasty, dirty, wet hole, filled
with the endsof worms and an oozy smell, nor yet a dry, bare, sandy hole with
nothing in it to sit down on or toeat: it was a [hobbit-hole][1], and that means
comfort.

[1]: <https://en.wikipedia.org/wiki/Hobbit#Lifestyle> "Hobbit lifestyles"
```

In a hole in the ground there lived a hobbit. Not a nasty, dirty, wet hole, filled with the ends of worms and an oozy smell, nor yet a dry, bare, sandy hole with nothing in it to sit down on or to eat: it was a hobbit-hole, and that means comfort.

In a hole in the ground there lived a hobbit. Not a nasty, dirty, wet hole, filled with the endsof worms and an oozy smell, nor yet a dry, bare, sandy hole with nothing in it to sit down on or toeat: it was a hobbit-hole, and that means comfort.

Return to the Top

Escaping Characters

> To display a literal character that would otherwise be used to format text in a Markdown document, add a backslash () in front of the character.

```
\* Without the backslash, this would be a bullet in an unordered list.

[_Return to the Top_](#python-with-arslan)
```

* Without the backslash, this would be a bullet in an unordered list.

Return to the Top

HTML

> Many Markdown applications allow you to use HTML tags in Markdown-formatted text. This is helpful if you prefer certain HTML tags to Markdown syntax. For example, some people find it easier to use HTML tags for images. Using HTML is also helpful when you need to change the attributes of an element, like specifying the color of text or changing the width of an image.

```
To use HTML, place the tags in the text of your Markdown-formatted file.

This **word** is bold. This <em>word</em> is italic.
```

To use HTML, place the tags in the text of your Markdown-formatted file.

This **word** is bold. This *word* is italic.

Return to the Top

Tables

Tables aren't part of the core Markdown spec, but they are part of GFM and Markdown Here supports them. They are an easy way of adding tables to your email -- a task that would otherwise require copy-pasting from another application.

```
Colons can be used to align columns.

| Tables        | Are           | Cool  |
| ------------- |:-------------:| -----:|
| col 3 is      | right-aligned | $1600 |
| col 2 is      | centered      |   $12 |
| zebra stripes | are neat      |    $1 |

The outer pipes (|) are optional, and you don't need to make the raw Markdown line
up prettily. You can also use inline Markdown.

Markdown | Less | Pretty
--- | --- | ---
*Still* | `renders` | **nicely**
1 | 2 | 3
```

Colons can be used to align columns.

Tables	Are	Cool
col 3 is	right-aligned	$1600

Tables	Are	Cool
col 2 is	centered	$12
zebra stripes	are neat	$1

The outer pipes (|) are optional, and you don't need to make the raw Markdown line up prettily. You can also use inline Markdown.

Markdown	Less	Pretty
Still	`renders`	**nicely**
1	2	3

Return to the Top

TeX Mathematical Formulae

```
$-b \pm \sqrt{b^2 - 4ac} \over 2a$
$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + a_4}}}$
$\forall x \in X, \quad \exists y \leq \epsilon$
```

$-b \pm \sqrt{b^2 - 4ac} \over 2a$
$x = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + a_4}}}$
$\forall x \in X, \quad \exists y \leq \epsilon$

The beginning and ending dollar signs ($) are the delimiters for the TeX markup.

Chapter 7 Statistics

Main objectives

1. choosing a right statical method
2. Do's and dont's of statistics
3. Reliable results
4. Paper revisions with proof of statistical test(WhQ s)
5. Making Data Visualization
6. Interpreting result

Return to the Top

Tests and their types

Parametric Tests

1. More Reliable results
2. First we have meet the assumptions

2	25
5	38
16	52
18	100
20	100

Not equal!

Return to the Top

Non-parametric Tests

1. Less reliable results
2. Calculates the rank of data
3. No need to make the assumptions

1	1
2	2
3	3
4	4
5	5

Return to the Top

Note: Before starting data analysis always check the normality of data

Normality test

Normality refers to a specific statistical distribution called a normal distribution, or sometimes the Gaussian distribution or bell-shaped curve. The normal distribution is a symmetrical continuous distribution defined by the mean and standard deviation of the data.

 bell curve

There are some properties of the normal distribution:

1. Bell-shaped
2. Symmetrical

3. Unimodal — it has one “peak”
   Mean and median are equal; both are located at the center of the distribution
4. About 68% of data falls within +/- one standard deviation of the mean
5. About 95% of data falls within +/- two standard deviations of the mean
6. About 99.7% of data falls within +/- three standard deviations of the mean

Tests to be used

1. Shapiro-wilk test
    - Specific (Reliable)
2. Kolmogorov-Smirnov test
    - General (less reliable)

Return to the Top

2. Homogeneity test

A data set is homogeneous if it is made up of things (i.e. people, cells or traits) that are similar to each other.

Test to be used

1. Levene's tets

Purpose

know the puorpose of your research question

Two type of purpose

1. Comparison
    - Difference
2. Relationship
    - Connection

Return to the Top

Comparison

if your purpose is comparison the compare atleast two groups
Examples:

1. Male vs Female
2. Control group vs treatment group

3. Grouping individual by color preference

Relationship

To find a connection
Examples:

1. Can food predict weight of a group of individuals
2. do fertilizer application increase crop growth?

We seek following here:

- Connection
- Correlayion
- Causation
- Prediction

Return to the Top

Data type

Know the type of data you are working with

CATEGORICAL

No numerical meaning represented in texts
(e.g：character,　factors)
Qualitative
EXAMPLES:

- Yes and No answers
  (Have you ever been to Lahore?)
  Which gene was expressed?
  Do you like Mangoes?" yes"or"No"

CONTINUOUS

Mostly represented in
(e.g：Numerical variable,
Quantitative
Numerical
number
int and float)
Quantitative
EXAMPLES：

- Amount

- Number
- Age
- Plant Height
- Number of bacterial colonies
- Chlorophyll content
- Fertilizer Amount

Return to the Top

Statistical tests

Choose a statistical test from three families

3 Families of statiscal tests

1. Chi-Squared
    - Purpose：Comparison
    - Data：Categorical only (Chi-Squared)
2. t-Test/ANOVA
    - Purpose：Comparison
    - Data：Categorical and continuous (t_Test)
3. Correlation
    - Purpose：Relationship
    - Data：continuous only (Corrrelation)

Chi-Squared

- Purpose：Comparison
- Data：Categorical only (Chi-Squared)

When and where to use?

Types:
1.Chi-Squared test of homogeneity
2.Chi-squared test of independence

When to use?

- Nothing effects this,
- Can be used with any number of levels or groups

You must remember the
purpose and datatype

t-Test/ANOVA

- Purpose：Comparison
- Data：Categorical and continuous
- (t_Test)

When and where to use?

Types：

1. One-sample t-Test(for one sample group with a know mean)
2. Two-sample t-Test：

- Un-paired t-Test(Two different groups)
- Paired t-Test (Same group Twice)

3. **ANOVA** (Analysis of Variance) [3+levels or groups are involved]

- **One-way ANOVA** (Even one of group is significant you will get significant results, but doesn't tell you which one)
- **Two-way ANOVA**
- **Repeated measures of ANOVA** (3+paired groups, scale up of Paired t-Test)

Correlation

- Purpose：Relationship
- Data：continuous only
(Corrrelation)

When and where to use?

Types：

1. **Pearson's Correlation** (one-Independent an One-Dependent Variable)
2. **Regression** (one-Independent and One-Dependent Variable)

Correlation: Tells us how closely cone nc ted two variables are?
"Is food a predictor of weight gain?"

Regression: Tells usa specific mathematical equation that describes the relationship.
(This helps us to find the data points not measured yet)
e.g：missing values can be predicted like this!

Return to the Top

Important Things

Assumptions about your data

These tests trusts you that:

- **Your data is Normally distributed**
- **or follow a Gaussian distribution**

If you do not follow the assumptions and break the trusts of 3-test families, they will not happy with you!

If Assumptions are not met! then

1. Normalize your Data
   a. Standardization
   b. Min-max scaling
   c. Log transformation
2. Use Alternative Non-Parametric Tests

Non-parametric test

1	2	3
Chi-Squared	t-Test/ANOVA	Correlation
Purpose：Comparison Data：Categorical only (Chi-Squared)	Purpose：Comparison Data：Categorical and continuous (t_Test)	Purpose：Relationship Data：continuous only (Corrrelation)
Chi-Squared	One-sample t-Test One-Sample Wilcoxon Signed rank test 2.Two-sample t-Test a.Unpaired t-Test(Mann Whitney'sU-Test) b.Paired t-Test(Wilcoxon) 3.ANOVA(Kruskal-Wallis test)	Pearson's Correlation (Spearman's Correlation) & (Kendall'sTau) Regression

Return to the Top

ANOVA

- Purpose：Comparison
- Data：Categorical and

Types of ANOVA

ANOVA (Analysis of Variance) [3+levels or groups are involved]

- **One-way ANOVA** (Even one of group is significant you will get significant results, but doesn't tell you which one)
- **Two-way ANOVA**
- **Repeated measures of ANOVA** (3+paired groups, scale up of Paired t-Test)

ANCOVA (Analysis of Co-variance)

- Compare the means of 3+iden pendent groups which can not be tested by ANOVA because the variables are affected by co-variance(pre-test and post-Test of class)

MANOVA (Multi-variate analysis of Variance)

MANCOVA (Multi-variate analysis of Co-variance)

Return to the Top

Some other test

Reliability tests

- K under-Richardson's Formula 20and 21 (KR20/21)
- Cronbach's Alpha

Inter-rater Reliability tests

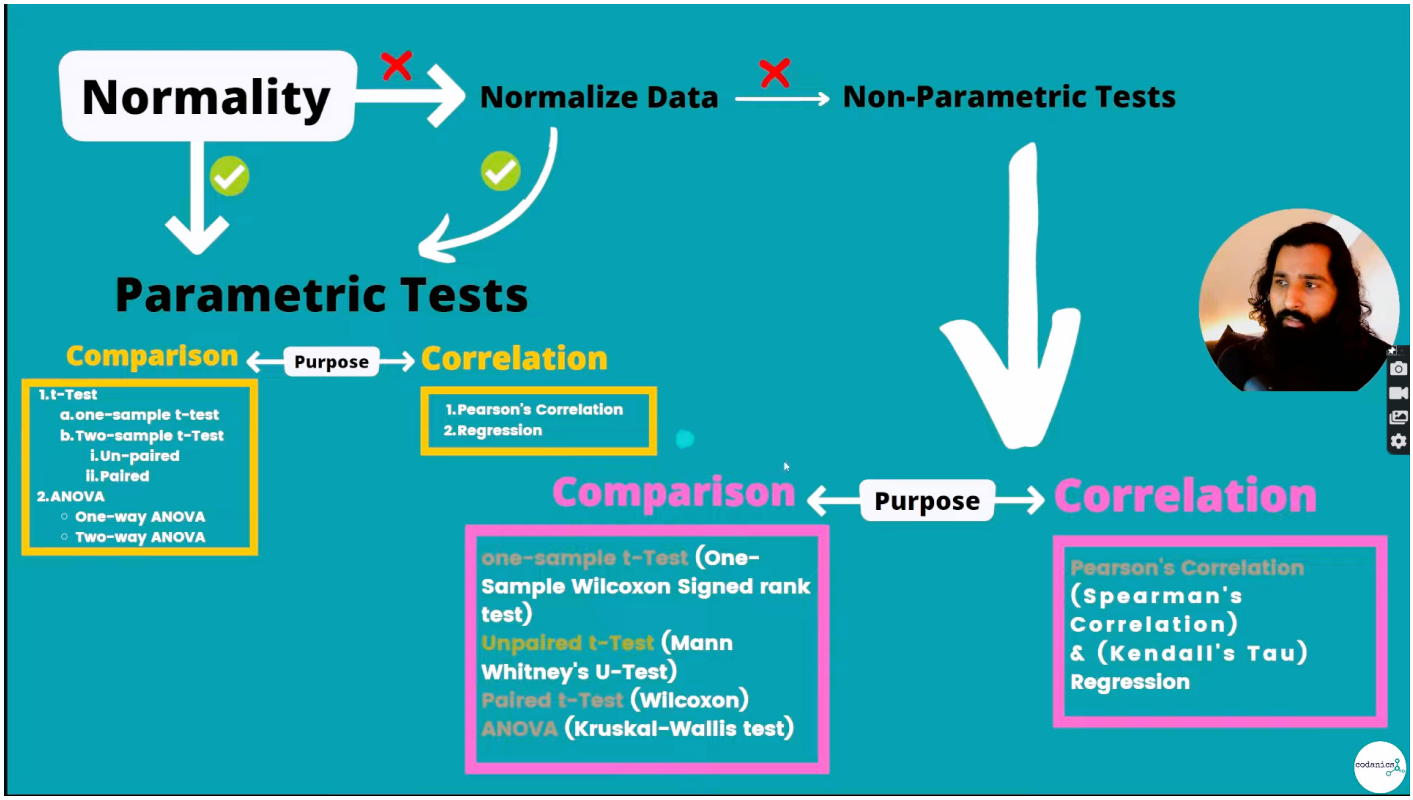
- Krippendorf's Alpha
    - (Categorical or continuous)
- Fleis's Kappa

    - (Only Categorical)

- Validity tests
    - Krippendorf's Alpha Test
    - Fle is's Kappa Test

- Sample size computation
  How to make sure how many samples are valid?
    - Cochran'sQ Test
    - Yamane's Test
    - many others.....

Whole Process diagram



Return to the Top

Statical hypotsis tests

Normality Tests

- Shapiro-Wilk Test
- D'Agostino's K^2 Test
- Anderson-Darling Test

Correlation Tests

- Pearson's Correlation Coefficient
- Spearman's Rank Correlation
- Kendall's Rank Correlation
- Chi-Squared Test

Stationary Tests

- Augmented Dickey-Fuller
- Kwiatkowski-Phillips-Schmidt-Shin

Parametric Statistical Hypothesis Tests

- Student's t-test
- Paired Student's t-test
- Analysis of Variance Test (ANOVA)
- Repeated Measures ANOVA Test

Nonparametric Statistical Hypothesis Tests

- Mann-Whitney U Test
- Wilcoxon Signed-Rank Test
- Kruskal-Wallis H Test
- Friedman Test

Return to the Top

Normality Tests

This section lists statistical tests that you can use to check if your data has a Gaussian distribution.

Shapiro-Wilk Test

- Tests whether a data sample has a Gaussian distribution.

Assumptions

- Observations in each sample are independent and identically distributed (iid).

Interpretation

- H0: the sample has a Gaussian distribution.
- H1: the sample does not have a Gaussian distribution.

```
# importing the required module
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

```
# Example of the Shapiro-Wilk Normality Test
from scipy.stats import shapiro
data = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
stat, p = shapiro(data)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably Gaussian')
else:
 print('Probably not Gaussian')
```

stat=0.895, p=0.193
Probably Gaussian

Return to the Top

D'Agostino's K^2 Test

Tests whether a data sample has a Gaussian distribution.

Assumptions

- Observations in each sample are independent and identically distributed (iid).

Interpretation

- H0: the sample has a Gaussian distribution.
- H1: the sample does not have a Gaussian distribution.

```
# Example of the D'Agostino's K^2 Normality Test
from scipy.stats import normaltest
data = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
stat, p = normaltest(data)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably Gaussian')
else:
 print('Probably not Gaussian')
```

```
stat=3.392, p=0.183
Probably Gaussian
```

Return to the Top

Anderson-Darling Test

Tests whether a data sample has a Gaussian distribution.

Assumptions

- Observations in each sample are independent and identically distributed (iid).

Interpretation

- H0: the sample has a Gaussian distribution.
- H1: the sample does not have a Gaussian distribution.

```
# Example of the Anderson-Darling Normality Test
from scipy.stats import anderson
data = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
result = anderson(data)
print('stat=%.3f' % (result.statistic))
for i in range(len(result.critical_values)):
 sl, cv = result.significance_level[i], result.critical_values[i]
 if result.statistic < cv:
  print('Probably Gaussian at the %.1f%% level' % (sl))
 else:
  print('Probably not Gaussian at the %.1f%% level' % (sl))
```

```
stat=0.424
Probably Gaussian at the 15.0% level
Probably Gaussian at the 10.0% level
Probably Gaussian at the 5.0% level
Probably Gaussian at the 2.5% level
Probably Gaussian at the 1.0% level
```

Return to the Top

Correlation Tests

This section lists statistical tests that you can use to check if two samples are related.

Pearson's Correlation Coefficient

Tests whether two samples have a linear relationship.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample are normally distributed.
- Observations in each sample have the same variance.

Interpretation

- H0: the two samples are independent.
- H1: there is a dependency between the samples.

```
# Example of the Pearson's Correlation test
from scipy.stats import pearsonr
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [0.353, 3.517, 0.125, -7.545, -0.555, -1.536, 3.350, -1.578, -3.537, -1.579]
stat, p = pearsonr(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably independent')
else:
 print('Probably dependent')
```

stat=0.688, p=0.028
Probably dependent

Return to the Top

Spearman's Rank Correlation

Tests whether two samples have a monotonic relationship.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample can be ranked.

Interpretation

- H0: the two samples are independent.
- H1: there is a dependency between the samples.

```
# Example of the Spearman's Rank Correlation Test
from scipy.stats import spearmanr
```

```
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [0.353, 3.517, 0.125, -7.545, -0.555, -1.536, 3.350, -1.578, -3.537, -1.579]
stat, p = spearmanr(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably independent')
else:
 print('Probably dependent')
```

stat=0.855, p=0.002
Probably dependent

Return to the Top

Kendall's Rank Correlation

Tests whether two samples have a monotonic relationship.

Assumptions

- bservations in each sample are independent and identically distributed (iid).
- Observations in each sample can be ranked.

Interpretation

- H0: the two samples are independent.
- H1: there is a dependency between the samples.

```
# Example of the Kendall's Rank Correlation Test
from scipy.stats import kendalltau
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [0.353, 3.517, 0.125, -7.545, -0.555, -1.536, 3.350, -1.578, -3.537, -1.579]
stat, p = kendalltau(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably independent')
else:
 print('Probably dependent')
```

stat=0.733, p=0.002
Probably dependent

Return to the Top

Chi-Squared Test

Tests whether two categorical variables are related or independent.

Assumptions

- Observations used in the calculation of the contingency table are independent.
- 25 or more examples in each cell of the contingency table.

Interpretation

- H0: the two samples are independent.
- H1: there is a dependency between the samples.

```
# Example of the Chi-Squared Test
from scipy.stats import chi2_contingency
table = [[10, 20, 30],[6,  9,  17]]
stat, p, dof, expected = chi2_contingency(table)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably independent')
else:
 print('Probably dependent')
```

stat=0.272, p=0.873
Probably independent

Return to the Top

Stationary Tests

This section lists statistical tests that you can use to check if a time series is stationary or not.

Augmented Dickey-Fuller Unit Root Test

Tests whether a time series has a unit root, e.g. has a trend or more generally is autoregressive.

Assumptions

- Observations in are temporally ordered.

Interpretation

- H0: a unit root is present (series is non-stationary).
- H1: a unit root is not present (series is stationary).

```
# Example of the Augmented Dickey-Fuller unit root test
from statsmodels.tsa.stattools import adfuller
data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
stat, p, lags, obs, crit, t = adfuller(data)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably not Stationary')
else:
 print('Probably Stationary')
```

stat=0.992, p=0.994
Probably not Stationary

Return to the Top

Kwiatkowski-Phillips-Schmidt-Shin

Tests whether a time series is trend stationary or not.

Assumptions

- Observations in are temporally ordered.

Interpretation

- H0: the time series is trend-stationary.
- H1: the time series is not trend-stationary.

```
# Example of the Kwiatkowski-Phillips-Schmidt-Shin test
from statsmodels.tsa.stattools import kpss
data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
stat, p, lags, crit = kpss(data)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably Stationary')
else:
 print('Probably not Stationary')
```

stat=0.410, p=0.073
Probably Stationary

Return to the Top

Parametric Statistical Hypothesis Tests

This section lists statistical tests that you can use to compare data samples.

Student's t-test

Tests whether the means of two independent samples are significantly different.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample are normally distributed.
- Observations in each sample have the same variance.

Interpretation

- H0: the means of the samples are equal.
- H1: the means of the samples are unequal.

```
# Example of the Paired Student's t-test
from scipy.stats import ttest_rel
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075,
-0.169]
stat, p = ttest_rel(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

stat=-0.334, p=0.746
Probably the same distribution

Return to the Top

Paired Student's t-test

Tests whether the means of two paired samples are significantly different.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample are normally distributed.
- Observations in each sample have the same variance.
- Observations across each sample are paired.

Interpretation

- H0: the means of the samples are equal.
- H1: the means of the samples are unequal.

```
# Example of the Paired Student's t-test
from scipy.stats import ttest_rel
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075,
-0.169]
stat, p = ttest_rel(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

```
stat=-0.334, p=0.746
Probably the same distribution
```

Return to the Top

Analysis of Variance Test (ANOVA)

Tests whether the means of two or more independent samples are significantly different.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample are normally distributed.
- Observations in each sample have the same variance.

Interpretation

- H0: the means of the samples are equal.
- H1: one or more of the means of the samples are unequal.

```
# Example of the Analysis of Variance Test
from scipy.stats import f_oneway
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075,
-0.169]
data3 = [-0.208, 0.696, 0.928, -1.148, -0.213, 0.229, 0.137, 0.269, -0.870, -1.204]
stat, p = f_oneway(data1, data2, data3)
print('stat=%.3f, p=%.3f' % (stat, p))
```

```
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

stat=0.096, p=0.908
Probably the same distribution

Return to the Top

Repeated Measures ANOVA Test

Tests whether the means of two or more paired samples are significantly different.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample are normally distributed.
- Observations in each sample have the same variance.
- Observations across each sample are paired.

Interpretation

- H0: the means of the samples are equal.
- H1: one or more of the means of the samples are unequal.

```
# Example of the repeated measures ANOVA Test
#create data
df = pd.DataFrame({'patient': np.repeat([1, 2, 3, 4, 5], 4),
                   'drug': np.tile([1, 2, 3, 4], 5),
                   'response': [30, 28, 16, 34,
                               14, 18, 10, 22,
                               24, 20, 18, 30,
                               38, 34, 20, 44,
                               26, 28, 14, 30]})

#perform the repeated measures ANOVA
print(AnovaRM(data=df, depvar='response', subject='patient', within=['drug']).fit())
```

```
            Anova
================================
     F Value Num DF  Den DF Pr > F
--------------------------------
```

```
drug 24.7589 3.0000 12.0000 0.0000
==================================
```

Return to the Top

Nonparametric Statistical Hypothesis Tests

Mann-Whitney U Test

Tests whether the distributions of two independent samples are equal or not.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample can be ranked.

Interpretation

- H0: the distributions of both samples are equal.
- H1: the distributions of both samples are not equal.

```
# Example of the Mann-Whitney U Test
from scipy.stats import mannwhitneyu
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075,
-0.169]
stat, p = mannwhitneyu(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

stat=40.000, p=0.473
Probably the same distribution

Return to the Top

Wilcoxon Signed-Rank Test

Tests whether the distributions of two paired samples are equal or not.

Assumptions

- Observations in each sample are independent and identically distributed (iid).

- Observations in each sample can be ranked.
- Observations across each sample are paired.

Interpretation

- H0: the distributions of both samples are equal.
- H1: the distributions of both samples are not equal.

```
# Example of the Wilcoxon Signed-Rank Test
from scipy.stats import wilcoxon
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075,
-0.169]
stat, p = wilcoxon(data1, data2)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

stat=21.000, p=0.557
Probably the same distribution

Return to the Top

Kruskal-Wallis H Test

Tests whether the distributions of two or more independent samples are equal or not.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample can be ranked.

Interpretation

- H0: the distributions of all samples are equal.
- H1: the distributions of one or more samples are not equal.

```
# Example of the Kruskal-Wallis H Test
from scipy.stats import kruskal
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075,
-0.169]
stat, p = kruskal(data1, data2)
```

```
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

stat=0.571, p=0.450
Probably the same distribution

Return to the Top

Friedman Test

Tests whether the distributions of two or more paired samples are equal or not.

Assumptions

- Observations in each sample are independent and identically distributed (iid).
- Observations in each sample can be ranked.
- Observations across each sample are paired.

Interpretation

- H0: the distributions of all samples are equal.
- H1: the distributions of one or more samples are not equal.

```
# Example of the Friedman Test
from scipy.stats import friedmanchisquare
data1 = [0.873, 2.817, 0.121, -0.945, -0.055, -1.436, 0.360, -1.478, -1.637, -1.869]
data2 = [1.142, -0.432, -0.938, -0.729, -0.846, -0.157, 0.500, 1.183, -1.075,
-0.169]
data3 = [-0.208, 0.696, 0.928, -1.148, -0.213, 0.229, 0.137, 0.269, -0.870, -1.204]
stat, p = friedmanchisquare(data1, data2, data3)
print('stat=%.3f, p=%.3f' % (stat, p))
if p > 0.05:
 print('Probably the same distribution')
else:
 print('Probably different distributions')
```

stat=0.800, p=0.670
Probably the same distribution

Return to the Top

chapter 8 Machine Learning Overview

What is Machine Learning?

Arthur Samuel coined the term Machine Learning in the year 1959. He was a pioneer in Artificial Intelligence and computer gaming, and defined Machine Learning as “Field of study that gives computers the capability to learn without being explicitly programmed”.

In simple terms, Machine Learning is an application of Artificial Intelligence (AI) which enables a program(software) to learn from the experiences and improve their self at a task without being explicitly programmed. For example, how would you write a program that can identify fruits based on their various properties, such as colour, shape, size or any other property?

One approach is to hardcode everything, make some rules and use them to identify the fruits. This may seem the only way and work but one can never make perfect rules that apply on all cases. This problem can be easily solved using machine learning without any rules which makes it more robust and practical. You will see how we will use machine learning to do this task in the coming sections.

Thus, we can say that Machine Learning is the study of making machines more human-like in their behaviour and decision making by giving them the ability to learn with minimum human intervention, i.e., no explicit programming. Now the question arises, how can a program attain any experience and from where does it learn? The answer is data. Data is also called the fuel for Machine Learning and we can safely say that there is no machine learning without data.

You may be wondering that the term Machine Learning has been introduced in 1959 which is a long way back, then why haven’t there been any mention of it till recent years? You may want to note that Machine Learning needs a huge computational power, a lot of data and devices which are capable of storing such vast data. We have only recently reached a point where we now have all these requirements and can practice Machine Learning.

How is it different from traditional programming?

Are you wondering how is Machine Learning different from traditional programming? Well, in traditional programming, we would feed the input data and a well written and tested program into a machine to generate output. When it comes to machine learning, input data along with the output associated with the data is fed into the machine during the learning phase, and it works out a program for itself.

Why do we need Machine Learning?

Machine Learning today has all the attention it needs. Machine Learning can automate many tasks, especially the ones that only humans can perform with their innate intelligence. Replicating this intelligence to machines can be achieved only with the help of machine learning.

With the help of Machine Learning, businesses can automate routine tasks. It also helps in automating and quickly create models for data analysis. Various industries depend on vast quantities of data to optimize their operations and make intelligent decisions. Machine Learning helps in creating models that can process and analyze large amounts of complex data to deliver accurate results. These models are precise and scalable and

function with less turnaround time. By building such precise Machine Learning models, businesses can leverage profitable opportunities and avoid unknown risks.

Image recognition, text generation, and many other use-cases are finding applications in the real world. This is increasing the scope for machine learning experts to shine as a sought after professionals.

In machine learning, there is a theorem called “no free lunch.” In short, it states that no single algorithm works for all problems, especially in supervised learning (ie, predictive modeling).

How Does Machine Learning Work?

A machine learning model learns from the historical data fed to it and then builds prediction algorithms to predict the output for the new set of data the comes in as input to the system. The accuracy of these models would depend on the quality and amount of input data. A large amount of data will help build a better model which predicts the output more accurately.

Suppose we have a complex problem at hand that requires to perform some predictions. Now, instead of writing a code, this problem could be solved by feeding the given data to generic machine learning algorithms. With the help of these algorithms, the machine will develop logic and predict the output. Machine learning has transformed the way we approach business and social problems. Below is a diagram that briefly explains the working of a machine learning model/ algorithm. our way of thinking about the problem.

History of Machine Learning

Nowadays, we can see some amazing applications of ML such as in self-driving cars, Natural Language Processing and many more. But Machine learning has been here for over 70 years now. It all started in 1943, when neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper about neurons, and how they work. They decided to create a model of this using an electrical circuit, and therefore, the neural network was born.

In 1950, Alan Turing created the “Turing Test” to determine if a computer has real intelligence. To pass the test, a computer must be able to fool a human into believing it is also human. In 1952, Arthur Samuel wrote the first computer learning program. The program was the game of checkers, and the IBM computer improved at the game the more it played, studying which moves made up winning strategies and incorporating those moves into its program.

Just after a few years, in 1957, Frank Rosenblatt designed the first neural network for computers (the perceptron), which simulates the thought processes of the human brain. Later, in 1967, the “nearest neighbour” algorithm was written, allowing computers to begin using very basic pattern recognition. This could be used to map a route for travelling salesmen, starting at a random city but ensuring they visit all cities during a short tour.

But we can say that in the 1990s we saw a big change. Now work on machine learning shifted from a knowledge-driven approach to a data-driven approach. Scientists began to create programs for computers to analyze large amounts of data and draw conclusions or “learn” from the results.

In 1997, IBM’s Deep Blue became the first computer chess-playing system to beat a reigning world chess champion. Deep Blue used the computing power in the 1990s to perform large-scale searches of potential moves

and select the best move. Just a decade before this, in 2006, Geoffrey Hinton created the term “deep learning” to explain new algorithms that help computers distinguish objects and text in images and videos.

Machine Learning at Present

The year 2012 saw the publication of an influential research paper by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever, describing a model that can dramatically reduce the error rate in image recognition systems. Meanwhile, Google’s X Lab developed a machine learning algorithm capable of autonomously browsing YouTube videos to identify the videos that contain cats. In 2016 AlphaGo (created by researchers at Google DeepMind to play the ancient Chinese game of Go) won four out of five matches against Lee Sedol, who has been the world’s top Go player for over a decade.

And now in 2020, OpenAI released GPT-3 which is the most powerful language model ever. It can write creative fiction, generate functioning code, compose thoughtful business memos and much more. Its possible use cases are limited only by our imaginations.

Features of Machine Learning

Automation: Nowadays in your Gmail account, there is a spam folder that contains all the spam emails. You might be wondering how does Gmail know that all these emails are spam? This is the work of Machine Learning. It recognises the spam emails and thus, it is easy to automate this process. The ability to automate repetitive tasks is one of the biggest characteristics of machine learning. A huge number of organizations are already using machine learning-powered paperwork and email automation. In the financial sector, for example, a huge number of repetitive, data-heavy and predictable tasks are needed to be performed. Because of this, this sector uses different types of machine learning solutions to a great extent.

Improved customer experience: For any business, one of the most crucial ways to drive engagement, promote brand loyalty and establish long-lasting customer relationships is by providing a customized experience and providing better services. Machine Learning helps us to achieve both of them. Have you ever noticed that whenever you open any shopping site or see any ads on the internet, they are mostly about something that you recently searched for? This is because machine learning has enabled us to make amazing recommendation systems that are accurate. They help us customize the user experience. Now coming to the service, most of the companies nowadays have a chatting bot with them that are available 24×7. An example of this is Eva from AirAsia airlines. These bots provide intelligent answers and sometimes you might even not notice that you are having a conversation with a bot. These bots use Machine Learning, which helps them to provide a good user experience.

Automated data visualization: In the past, we have seen a huge amount of data being generated by companies and individuals. Take an example of companies like Google, Twitter, Facebook. How much data are they generating per day? We can use this data and visualize the notable relationships, thus giving businesses the ability to make better decisions that can actually benefit both companies as well as customers. With the help of user-friendly automated data visualization platforms such as AutoViz, businesses can obtain a wealth of new insights in an effort to increase productivity in their processes.

Business intelligence: Machine learning characteristics, when merged with big data analytics can help companies to find solutions to the problems that can help the businesses to grow and generate more profit. From retail to financial services to healthcare, and many more, ML has already become one of the most effective technologies to boost business operations.

Types of Machine Learning

Machine learning has been broadly categorized into three categories

1. Supervised Learning
2. Unsupervised Learning
3. Semi-supervised learning
4. Reinforcement Learning

What is Supervised Learning?

Let us start with an easy example, say you are teaching a kid to differentiate dogs from cats. How would you do it?

You may show him/her a dog and say “here is a dog” and when you encounter a cat you would point it out as a cat. When you show the kid enough dogs and cats, he may learn to differentiate between them. If he is trained well, he may be able to recognise different breeds of dogs which he hasn’t even seen.

Similarly, in Supervised Learning, we have two sets of variables. One is called the target variable, or labels (the variable we want to predict) and features(variables that help us to predict target variables). We show the program(model) the features and the label associated with these features and then the program is able to find the underlying pattern in the data. Take this example of the dataset where we want to predict the price of the house given its size. The price which is a target variable depends upon the size which is a feature.

```
Number of rooms    Price
1                  $100
3                  $300
5                  $500
```

In a real dataset, we will have a lot more rows and more than one features like size, location, number of floors and many more.

Thus, we can say that the supervised learning model has a set of input variables (x), and an output variable (y). An algorithm identifies the mapping function between the input and output variables. The relationship is y = f(x).

The learning is monitored or supervised in the sense that we already know the output and the algorithm are corrected each time to optimise its results. The algorithm is trained over the data set and amended until it achieves an acceptable level of performance.

We can group the supervised learning problems as:

Regression problems – Used to predict future values and the model is trained with the historical data. E.g., Predicting the future price of a house.

Classification problems – Various labels train the algorithm to identify items within a specific category. E.g., Dog or cat(as mentioned in the above example), Apple or an orange, Beer or wine or water.

What is Unsupervised Learning?

This approach is the one where we have no target variables, and we have only the input variable(features) at hand. The algorithm learns by itself and discovers an impressive structure in the data.

The goal is to decipher the underlying distribution in the data to gain more knowledge about the data.

We can group the unsupervised learning problems as:

Clustering: This means bundling the input variables with the same characteristics together. E.g., grouping users based on search history

Association: Here, we discover the rules that govern meaningful associations among the data set. E.g., People who watch 'X' will also watch 'Y'.

what is Semi-supervised learning?

Semi-supervised machine learning is a combination of supervised and unsupervised learning. It uses a small amount of labeled data and a large amount of unlabeled data, which provides the benefits of both unsupervised and supervised learning while avoiding the challenges of finding a large amount of labeled data. That means you can train a model to label data without having to use as much labeled training data.

What is Reinforcement Learning?

In this approach, machine learning models are trained to make a series of decisions based on the rewards and feedback they receive for their actions. The machine learns to achieve a goal in complex and uncertain situations and is rewarded each time it achieves it during the learning period.

Reinforcement learning is different from supervised learning in the sense that there is no answer available, so the reinforcement agent decides the steps to perform a task. The machine learns from its own experiences when there is no training data set present.

Return to the Top

List of Common Machine Learning Algorithms

Linear Regression

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a *X + b.

Logistic Regression

It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logit regression. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

Decision Tree

It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible.

SVM (Support Vector Machine)

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

Naive Bayes

It is a classification technique based on Bayes' theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

kNN (k- Nearest Neighbors)

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors.

K-Means

It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.

Random Forest

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Dimensionality Reduction Algorithms

In the last 4-5 years, there has been an exponential increase in data capturing at every possible stages. Corporates/ Government Agencies/ Research organisations are not only coming with new sources but also they are capturing data in great detail.

Return to the Top

According to the function to divide, machine learning, including

Regression algorithm

- Linear regression
- Logistic regression
- Multiple Adaptive Regression (MARS)
- Local scatter smoothing estimate (LOESS)

Instance-based learning algorithm

- K — proximity algorithm (kNN)
- Learning vectorization (LVQ)
- Self-Organizing Mapping Algorithm (SOM)
- Local Weighted Learning Algorithm (LWL)

Regularization algorithm

- Ridge Regression
- LASSO（Least Absolute Shrinkage and Selection Operator）
- Elastic Net
- Minimum Angle Regression (LARS)

Decision tree algorithm

- Classification and Regression Tree (CART)
- ID3 algorithm (Iterative Dichotomiser 3)
- C4.5 and C5.0
- CHAID（Chi-squared Automatic Interaction Detection()
- Random Forest
- Multivariate Adaptive Regression Spline (MARS)
- Gradient Boosting Machine (GBM)

Bayesian algorithm

- Naive Bayes
- Gaussian Bayes
- Polynomial naive Bayes

- AODE（Averaged One-Dependence Estimators）
- Bayesian Belief Network

Kernel-based algorithm

- Support vector machine (SVM)
- Radial Basis Function (RBF)
- Linear Discriminate Analysis (LDA)

Clustering Algorithm

- K — mean
- K — medium number
- EM algorithm
- Hierarchical clustering

Association rule learning

- Apriori algorithm
- Eclat algorithm

Neural Networks

- sensor
- Backpropagation algorithm (BP)
- Hopfield network
- Radial Basis Function Network (RBFN)

Deep learning

- Deep Boltzmann Machine (DBM)
- Convolutional Neural Network (CNN)
- Recurrent neural network (RNN, LSTM)
- stacked Auto-Encoder

Dimensionality reduction algorithm

- Principal Component Analysis (PCA)
- Principal component regression (PCR)
- Partial least squares regression (PLSR)
- Salmon map
- Multidimensional scaling analysis (MDS)
- Projection pursuit method (PP)
- Linear Discriminant Analysis (LDA)
- Mixed Discriminant Analysis (MDA)

- Quadratic Discriminant Analysis (QDA)
- Flexible Discriminant Analysis (FDA)

Integrated algorithm

- Boosting
- Bagging
- AdaBoost
- Stack generalization (mixed)
- GBM algorithm
- GBRT algorithm
- Random forest

Other algorithms

- Feature selection algorithm
- Performance evaluation algorithm
- Natural language processing
- Computer vision
- Recommended system
- Reinforcement learning
- Migration learning

Return to the Top

Steps in Machine Learning

I wish Machine learning was just applying algorithms on your data and get the predicted values but it is not that simple. There are several steps in Machine Learning which are must for each project.

Gathering Data

This is perhaps the most important and time-consuming process. In this step, we need to collect data that can help us to solve our problem. For example, if you want to predict the prices of the houses, we need an appropriate dataset that contains all the information about past house sales and then form a tabular structure. We are going to solve a similar problem in the implementation part.

Preparing that data

Once we have the data, we need to bring it in proper format and preprocess it. There are various steps involved in pre-processing such as data cleaning, for example, if your dataset has some empty values or abnormal values(e.g, a string instead of a number) how are you going to deal with it? There are various ways in which we can but one simple way is to just drop the rows that have empty values. Also sometimes in the dataset, we might have columns that have no impact on our results such as id’s, we remove those columns as well.We usually use Data Visualization to visualise our data through graphs and diagrams and after analyzing the graphs, we decide which

features are important.Data preprocessing is a vast topic and I would suggest checking out this article to know more about it.

Choosing a model

Now our data is ready is to be fed into a Machine Learning algorithm. In case you are wondering what is a Model? Often “machine learning algorithm” is used interchangeably with “machine learning model.” A model is the output of a machine learning algorithm run on data. In simple terms when we implement the algorithm on all our data, we get an output which contains all the rules, numbers, and any other algorithm-specific data structures required to make predictions. For example, after implementing Linear Regression on our data we get an equation of the best fit line and this equation is termed as a model.The next step is usually training the model incase we don’t want to tune hyperparameters and select the default ones.

valuation

You may be wondering, how can you know if the model is performing good or bad.What better way than testing the model on some data. This data is known as testing data and it must not be a subset of the data(training data) on which we trained the algorithm. The objective of training the model is not for it to learn all the values in the training dataset but to identify the underlying pattern in data and based on that make predictions on data it has never seen before. There are various evaluation methods.

The results of predictive models can be viewed in various forms such as by using confusion matrix, root-mean-squared error(RMSE), AUC-ROC etc.

A confusion matrix used in classification problems is a table that displays the number of instances that are correctly and incorrectly classified in terms of each category within the attribute that is the target class

TP (True Positive) is the number of values predicted to be positive by the algorithm and was actually positive in the dataset. TN represents the number of values that are expected to not belong to the positive class and actually do not belong to it. FP depicts the number of instances misclassified as belonging to the positive class thus is actually part of the negative class. FN shows the number of instances classified as the negative class but should belong to the positive class.

Now in Regression problem, we usually use RMSE as evaluation metrics. In this evaluation technique, we use the error term.

In a good model, the RMSE should be as low as possible and there should not be much difference between RMSE calculated over training data and RMSE calculated over the testing set.

Prediction

Now that our model has performed well on the testing set as well, we can use it in real-world and hope it is going to perform well on real-world data.

Return to the Top

Advantages of Machine Learning

1. Easily identifies trends and patterns

Machine Learning can review large volumes of data and discover specific trends and patterns that would not be apparent to humans. For instance, for e-commerce websites like Amazon and Flipkart, it serves to understand the browsing behaviors and purchase histories of its users to help cater to the right products, deals, and reminders relevant to them. It uses the results to reveal relevant advertisements to them.

2. Continuous Improvement

We are continuously generating new data and when we provide this data to the Machine Learning model which helps it to upgrade with time and increase its performance and accuracy. We can say it is like gaining experience as they keep improving in accuracy and efficiency. This lets them make better decisions.

3. Handling multidimensional and multi-variety data

Machine Learning algorithms are good at handling data that are multidimensional and multi-variety, and they can do this in dynamic or uncertain environments.

4. Wide Applications

You could be an e-tailer or a healthcare provider and make Machine Learning work for you. Where it does apply, it holds the capability to help deliver a much more personal experience to customers while also targeting the right customers.

Return to the Top

Disadvantages of Machine Learning

1. Data Acquisition

Machine Learning requires a massive amount of data sets to train on, and these should be inclusive/unbiased, and of good quality. There can also be times where we must wait for new data to be generated.

2. Time and Resources

Machine Learning needs enough time to let the algorithms learn and develop enough to fulfill their purpose with a considerable amount of accuracy and relevancy. It also needs massive resources to function. This can mean additional requirements of computer power for you.

3. Interpretation of Results

Another major challenge is the ability to accurately interpret results generated by the algorithms. You must also carefully choose the algorithms for your purpose. Sometimes, based on some analysis you might select an algorithm but it is not necessary that this model is best for the problem.

4. High error-susceptibility

Machine Learning is autonomous but highly susceptible to errors. Suppose you train an algorithm with data sets small enough to not be inclusive. You end up with biased predictions coming from a biased training set. This

leads to irrelevant advertisements being displayed to customers. In the case of Machine Learning, such blunders can set off a chain of errors that can go undetected for long periods of time. And when they do get noticed, it takes quite some time to recognize the source of the issue, and even longer to correct it.

Return to the Top

Chapter 9 Supervised Learning

Implementation of a Regression problem

We have a problem of predicting the prices of the house given some features such as size, number of rooms and many more. So let us get started:

Return to the Top

1. Gathering data

We don't need to manually collect the data for past sales of houses. Luckily there are some good people who do it for us and make these datasets available for us to use.

We are going to import all the necessary libraries such as Pandas and NumPy.Next, we will load the dataset directly from the sklearn library into a pandas DataFrame.

Importing require libraries

```
# import required library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.feature_selection import f_regression, SelectKBest
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from scipy import stats
from sklearn.linear_model import LinearRegression
import warnings
warnings.filterwarnings('ignore')
```

```
#### Reading Dataset

# load dataset from sklearn
boston_dataset = datasets.load_boston()
```

```
# load datset as pandas datsframe
data = pd.DataFrame(boston_dataset.data,columns=boston_dataset.feature_names)
data['target'] = boston_dataset.target
# print 10 sample rows
data.sample(10)
```

```
     CRIM ZN    INDUS CHAS NOX RM     AGE      DIS RAD  TAX PTRATIO   B    LSTAT target
6 0.08829 12.5 7.87 0.0  0.524 6.012 66.6 5.5605 5.0  311.0 15.2 395.60 12.43 22.9
363 4.22239 0.0     18.10 1.0  0.770 5.803 89.0 1.9047 24.0 666.0 20.2 353.04 14.64
16.8
246 0.33983 22.0 5.86 0.0  0.431 6.108 34.9 8.0555 7.0  330.0 19.1 390.18 9.16 24.3
137 0.35233 0.0     21.89 0.0  0.624 6.454 98.4 1.8498 4.0  437.0 21.2 394.08 14.59
17.1
236 0.52058 0.0     6.20 1.0  0.507 6.631 76.5 4.1480 8.0  307.0 17.4 388.45 9.54
25.1
245 0.19133 22.0 5.86 0.0  0.431 5.605 70.2 7.9549 7.0  330.0 19.1 389.13 18.46 18.5
315 0.25356 0.0     9.90 0.0  0.544 5.705 77.7 3.9450 4.0  304.0 18.4 396.42 11.50
16.2
140 0.29090 0.0     21.89 0.0  0.624 6.174 93.6 1.6119 4.0  437.0 21.2 388.08 24.16
14.0
399 9.91655 0.0     18.10 0.0  0.693 5.852 77.8 1.5004 24.0 666.0 20.2 338.16 29.97
6.3
73 0.19539 0.0     10.81 0.0  0.413 6.245 6.2     5.2873 4.0  305.0 19.2 377.17 7.54
23.4
```

Defining the problem statement:

Create a ML model which can predict the price(target) of a house

- Target Variable: target
- features: CRIM, ZN, INDUS, CHAS, NOX, RM, etc.

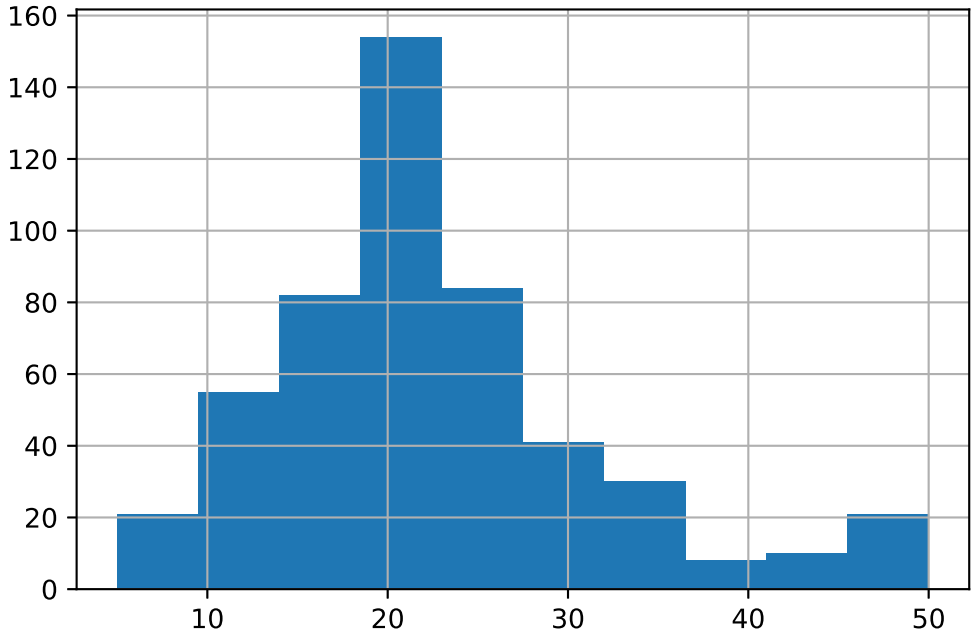
Determining the type of Machine Learning

Based on the problem statement you can understand that we need to create a supervised ML Regression model, as the target variable is Continuous.

Looking at the distribution of Target variable

- If target variable's distribution is too skewed then the predictive modeling will not be possible.
- Bell curve is desirable but slightly positive skew or negative skew is also fine
- When performing Regression, make sure the histogram looks like a bell curve or slight skewed version of it. Otherwise it impacts the Machine Learning algorithms ability to learn all the scenarios.

```
# Creating histogram as the Target variable is Continuous
# This will help us to understand the distribution of the MEDV values
data['target'].hist()
```



The data distribution of the target variable is satisfactory to proceed further. There are sufficient number of rows for each type of values to learn from.

Return to the Top

2. Preprocess data

Basic Data Exploration

This step is performed to gauge the overall data. The volume of data, the types of columns present in the data. Initial assessment of the data should be done to identify which columns are Quantitative, Categorical or Qualitative.

This step helps to start the column rejection process. You must look at each column carefully and ask, **does this column affect the values of the Target variable?** For example in this case study, you will ask, does this column affect the price of the house? If the answer is a clear "No", then remove the column immediately from the data, otherwise keep the column for further analysis.

There are four commands which are used for Basic data exploration in Python

- **head()** : This helps to see a few sample rows of the data
- **info()** : This provides the summarized information of the data
- **describe()** : This provides the descriptive statistical details of the data

- **nunique()**: This helps us to identify if a column is categorical or continuous

```
#print basic info
print(f"Contains {data.shape[0]} rows and {data.shape[1]-1} feature and contimous
target variable")
```

Contains 506 rows and 13 feature and contimous target variable

```
# describing the feature name and attribute information
print(boston_dataset.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people by
town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None
```

Thoughts...

1. No missing value are given
2. As target value is represnted as median , there are chances that dataset contain outliers.
3. Most of the feature correlates the target value.
4. Feature have different scales.

Data wrangling

Find missing values

```
# checking for any missing value

py
print("Missing values for every features.")
data.isna().sum()
```

```
Missing values for every features.
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
target     0
dtype: int64
```

As given there are no missing values

```
# check datatype and na values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
```

```
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  target   506 non-null    float64
dtypes: float64(14)
```

All data is in numeric(float64)

```
#describe dataset
data.describe().T
```

```
         count  mean       std      min       25%      50%       75%      max
CRIM     506.0 3.613524  8.601545  0.00632  0.082045  0.25651  3.677083  88.9762
ZN       506.0 11.363636 23.322453 0.00000  0.000000  0.00000  12.500000 100.0000
INDUS    506.0 11.136779 6.860353  0.46000  5.190000  9.69000  18.100000 27.7400
CHAS     506.0 0.069170  0.253994  0.00000  0.000000  0.00000  0.000000  1.0000
NOX      506.0 0.554695  0.115878  0.38500  0.449000  0.53800  0.624000  0.8710
RM       506.0 6.284634  0.702617  3.56100  5.885500  6.20850  6.623500  8.7800
AGE      506.0 68.574901 28.148861 2.90000  45.025000 77.50000 94.075000 100.0000
DIS      506.0 3.795043  2.105710  1.12960  2.100175  3.20745  5.188425  12.1265
RAD      506.0 9.549407  8.707259  1.00000  4.000000  5.00000  24.000000 24.0000
TAX      506.0 408.23715 168.53711 187.0000 279.00000 330.0000 666.00000 711.0000
PTRATIO 506.0 18.455534 2.164946  12.60000 17.400000 19.05000 20.200000 22.0000
B        506.0 356.67403 91.294864 0.32000  375.37750 391.4400 396.22500 396.9000
LSTAT    506.0 12.653063 7.141062  1.73000  6.950000  11.36000 16.955000 37.9700
target   506.0 22.532806 9.197104  5.00000  17.025000 21.20000 25.000000 50.000
```

```
# check if rows are duplicate or not
print(f"There are {data.duplicated().sum()} rows in dataset.")
```

There are 0 rows in dataset.

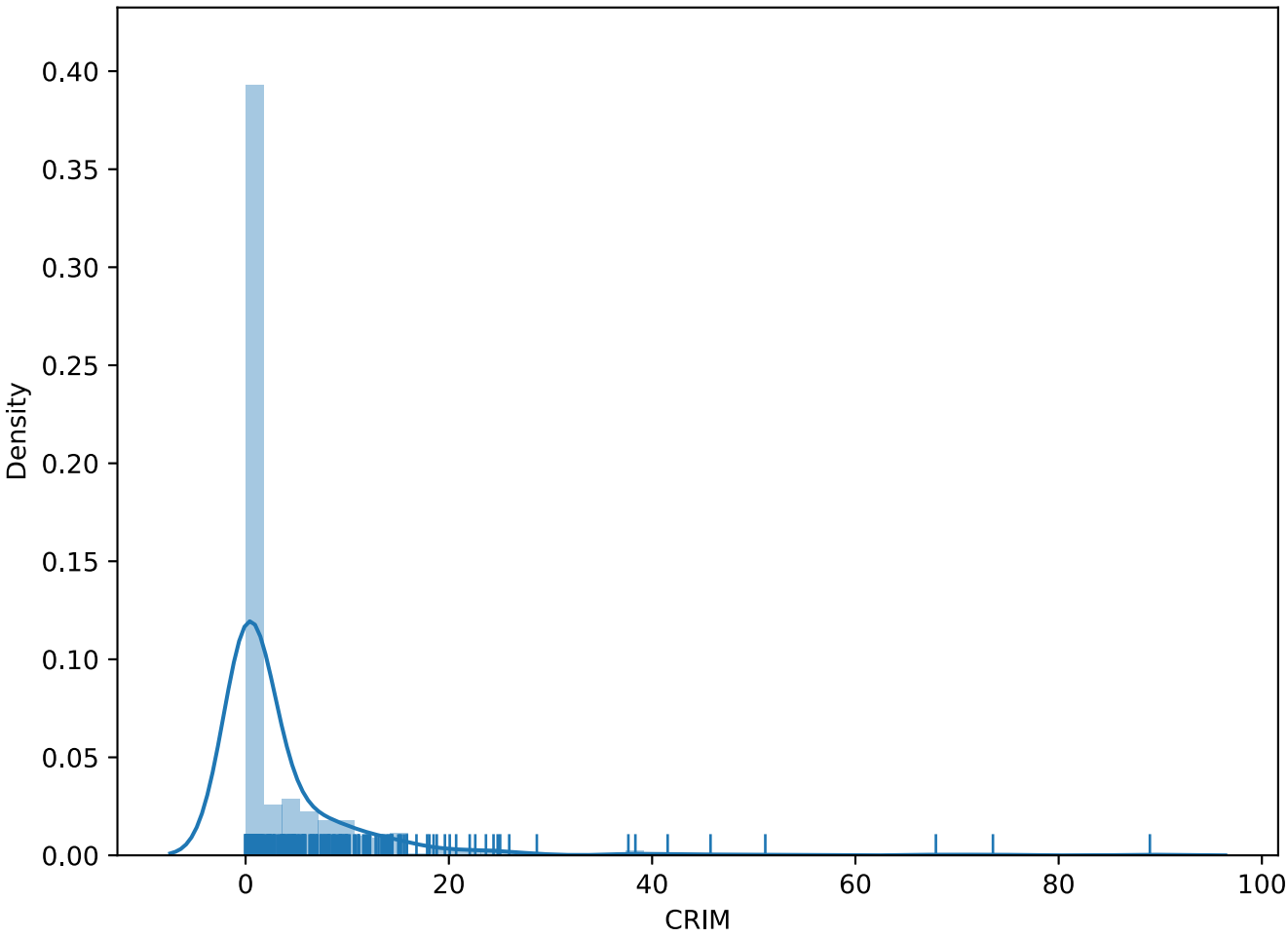
Thoughts...

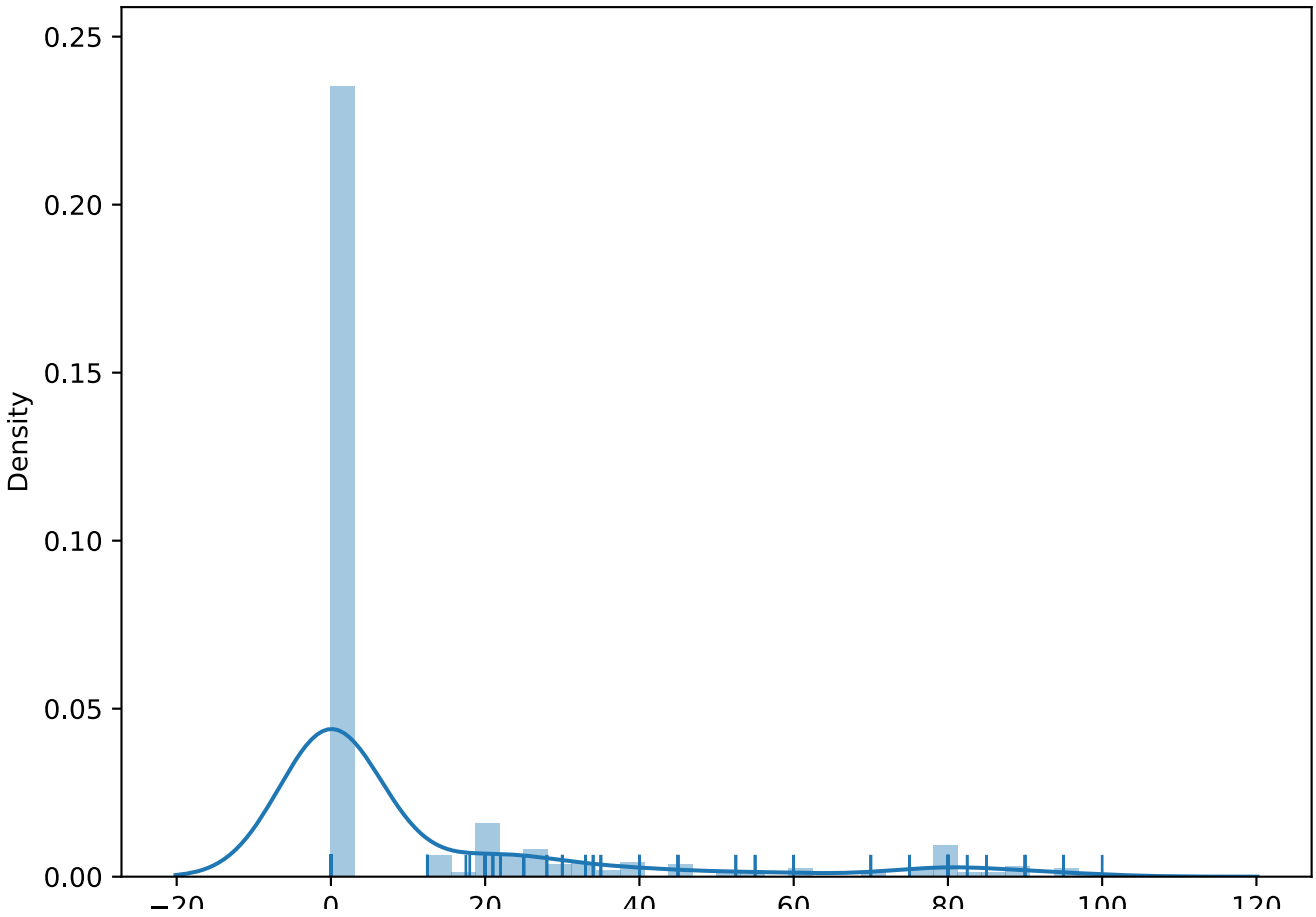
1. Dataset dosent contain any null values.
2. All features and target vatriable are numeric(float64), `CHAS` is an categorical varible.
3. Features `CHAS`, `NOX` and `ZN` have more than 50% value as zero.
4. Most of the feature contain outliers that need to fixed.
5. There are no duplicated rows

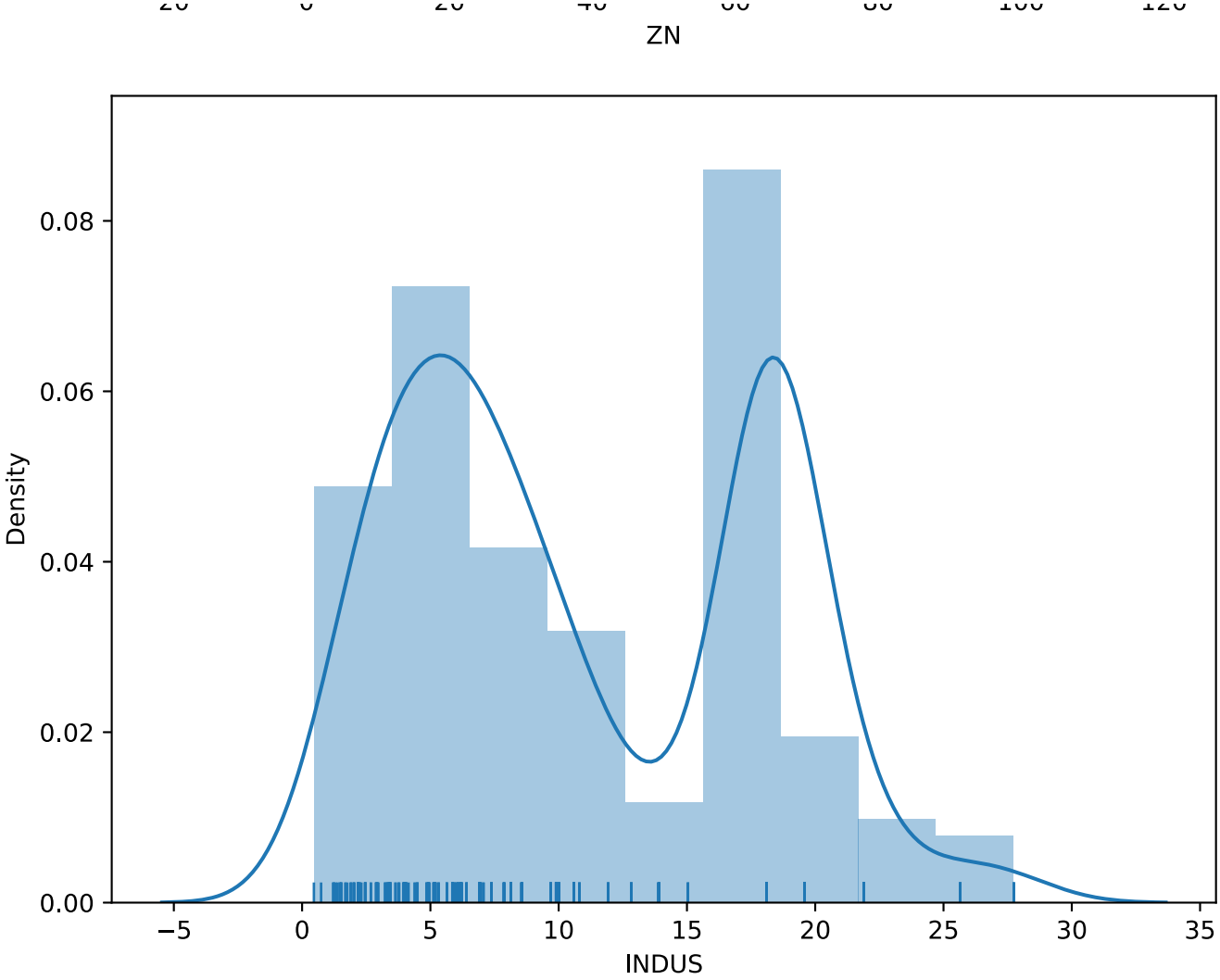
Visual Exploratory Data Analysis

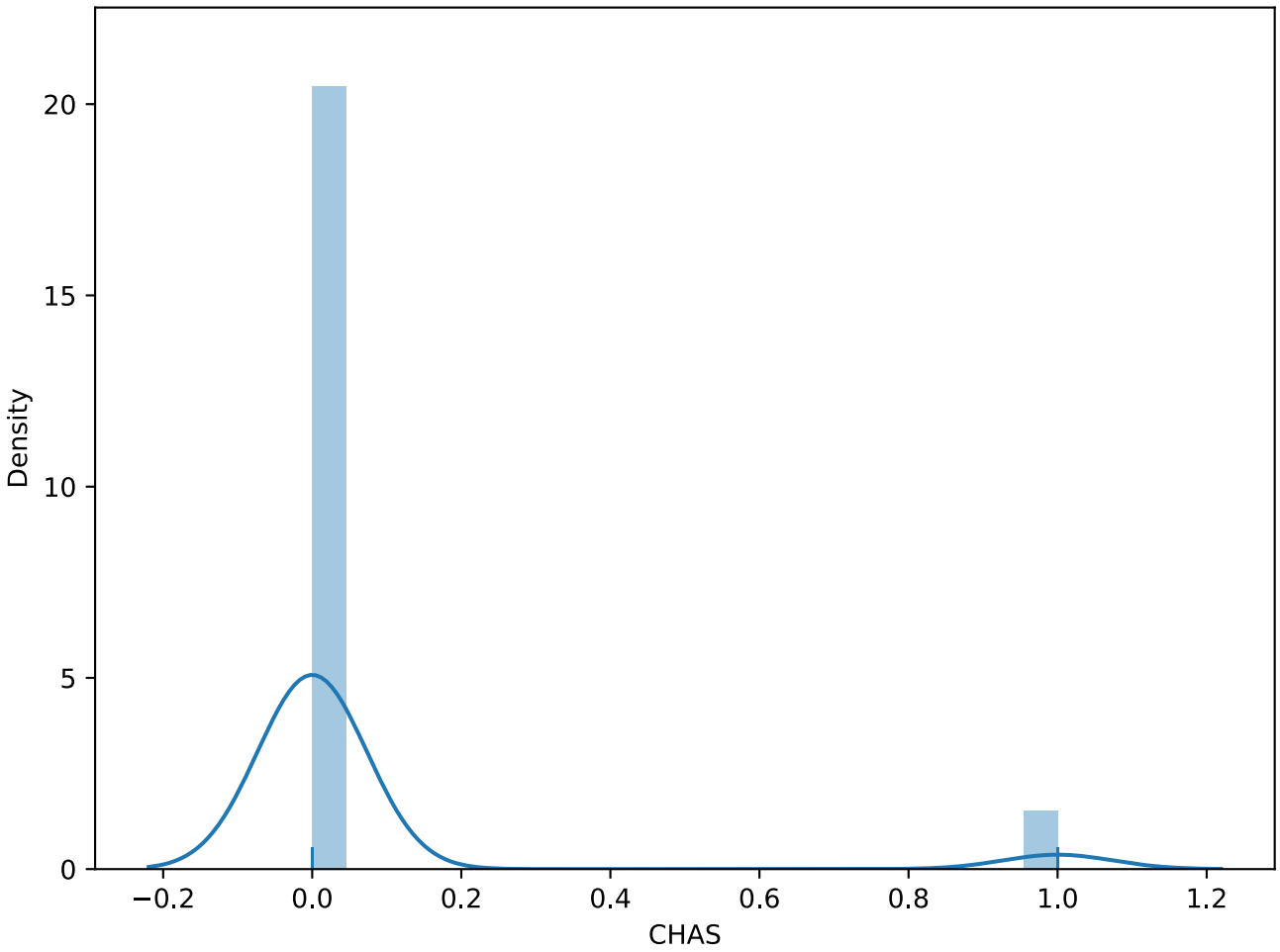
```
import seaborn as sb
plt.figure(figsize=(8, 6))
sb.distplot(data['CRIM'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['ZN'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['INDUS'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['CHAS'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['NOX'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['RM'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['AGE'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['DIS'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['RAD'], rug = True)
```

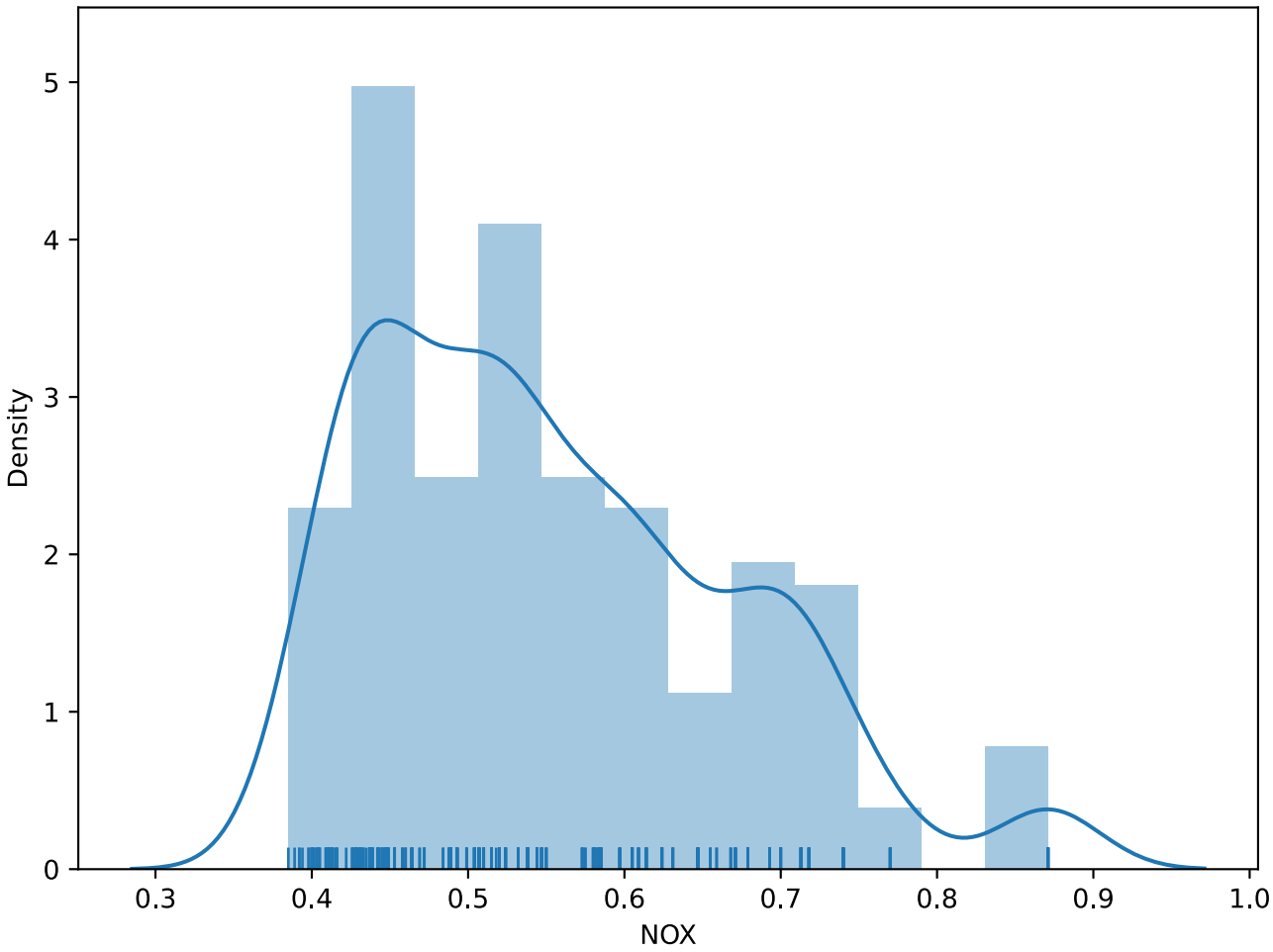
```
plt.figure(figsize=(8, 6))
sb.distplot(data['TAX'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['PTRATIO'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['B'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['LSTAT'], rug = True)


plt.figure(figsize=(8, 6))
sb.distplot(data['target'], rug = True)
```

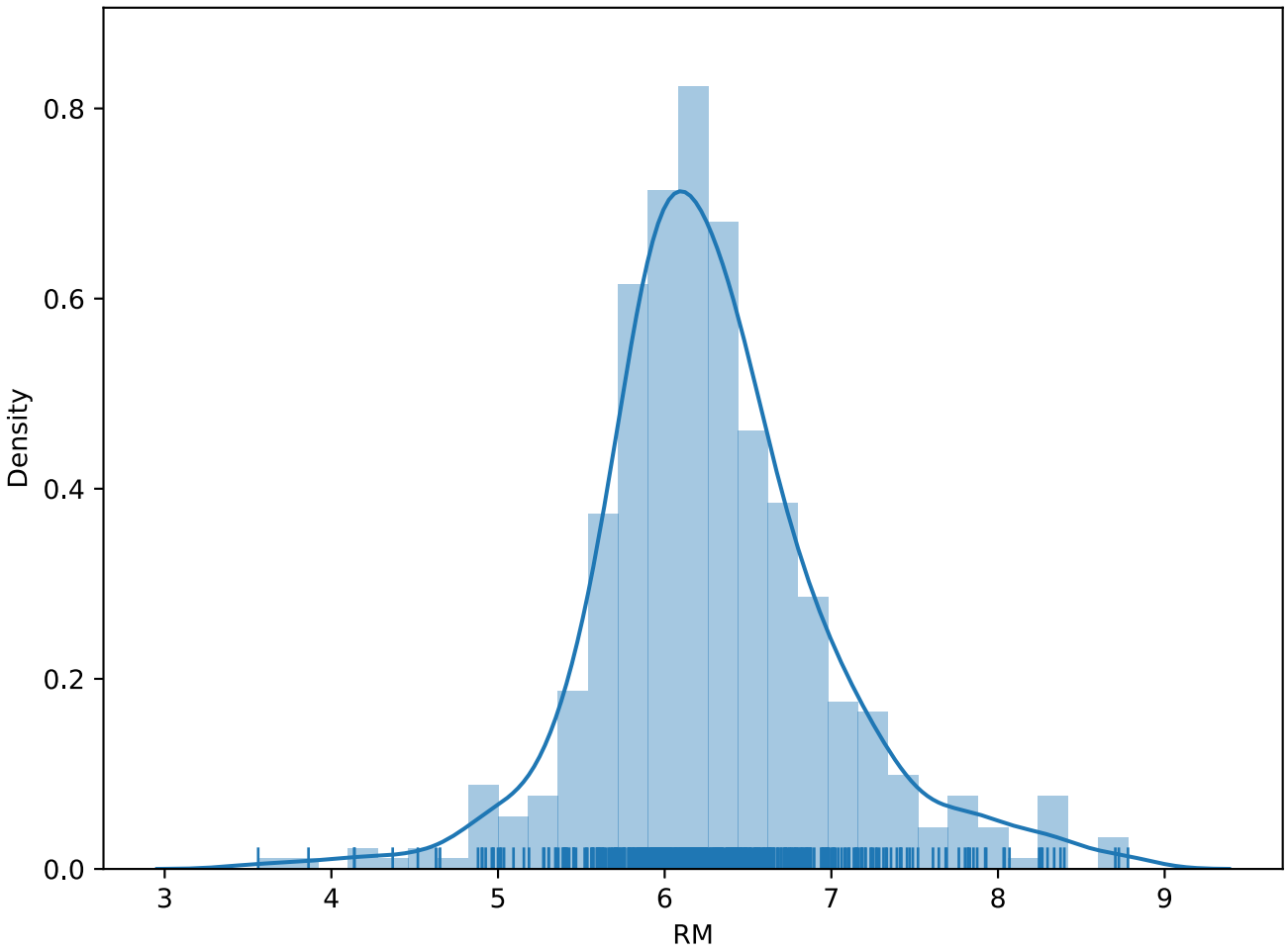

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
0
20
40
60
80
100
CRIM

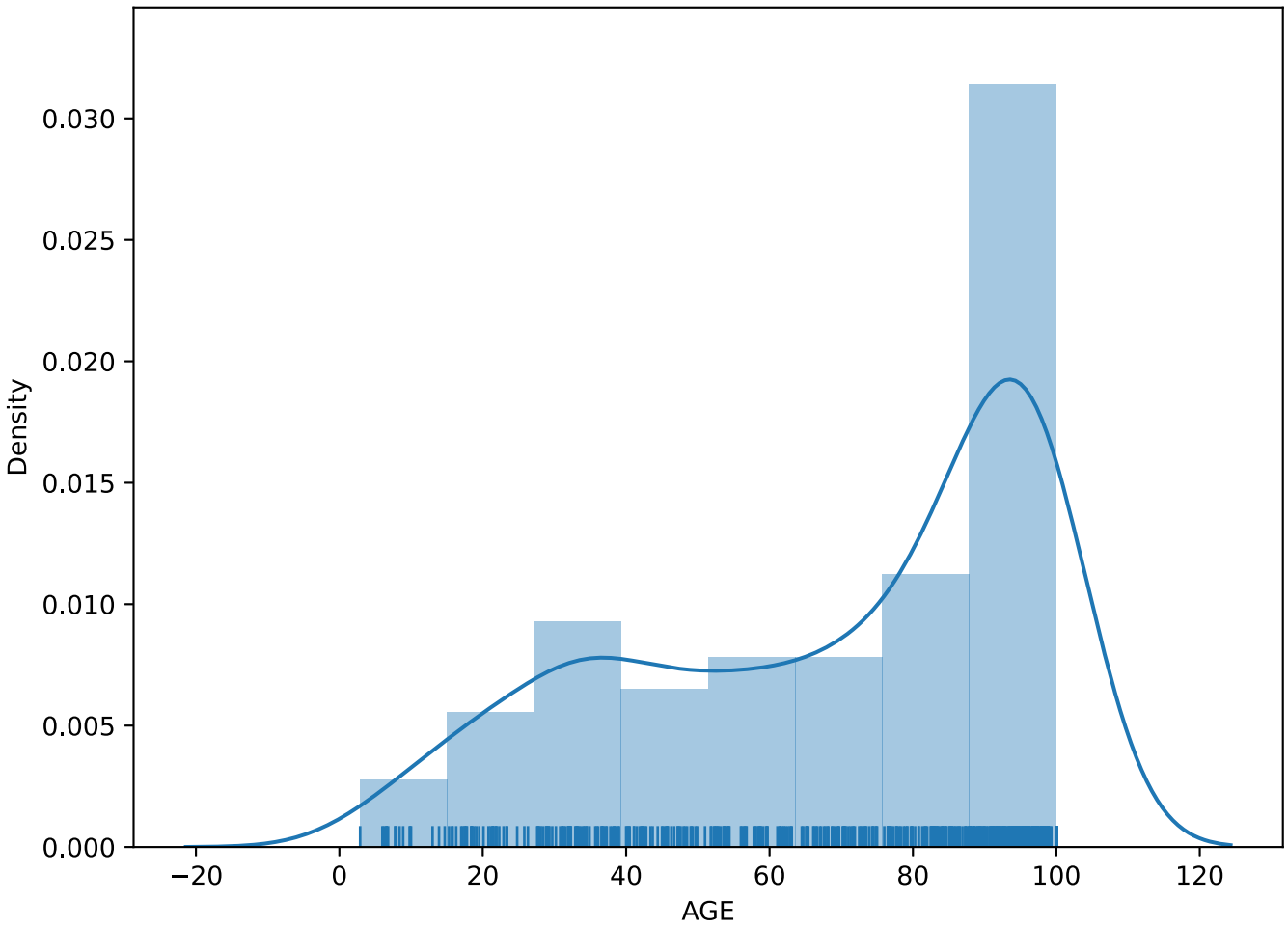

0.25
0.20
0.15
0.10
0.05
0.00
Density
−20
0
20
40
60
80
100
120

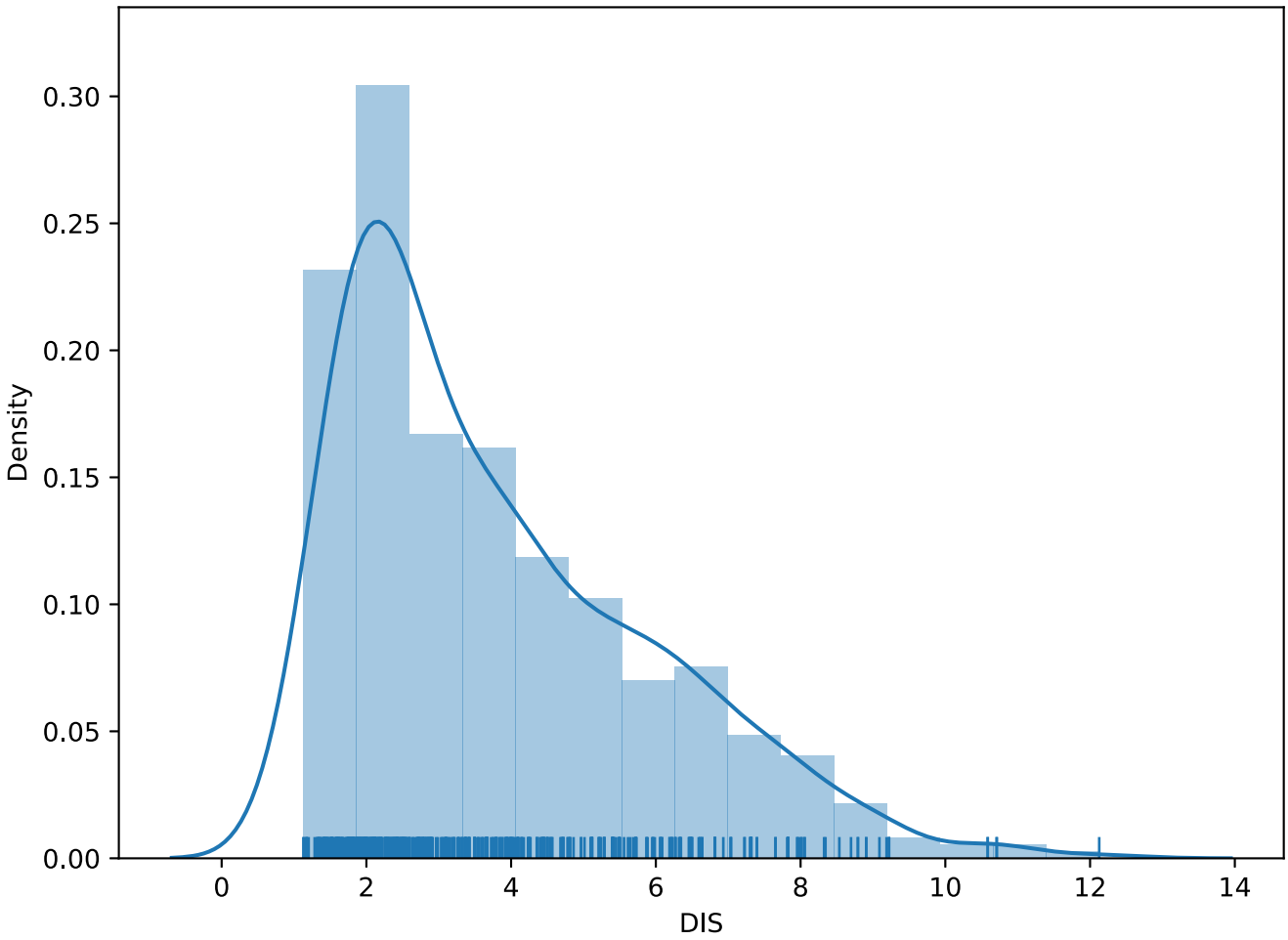

ZN
Density
0.08
0.06
0.04
0.02
0.00
−5
0
5
10
15
20
25
30
35
INDUS

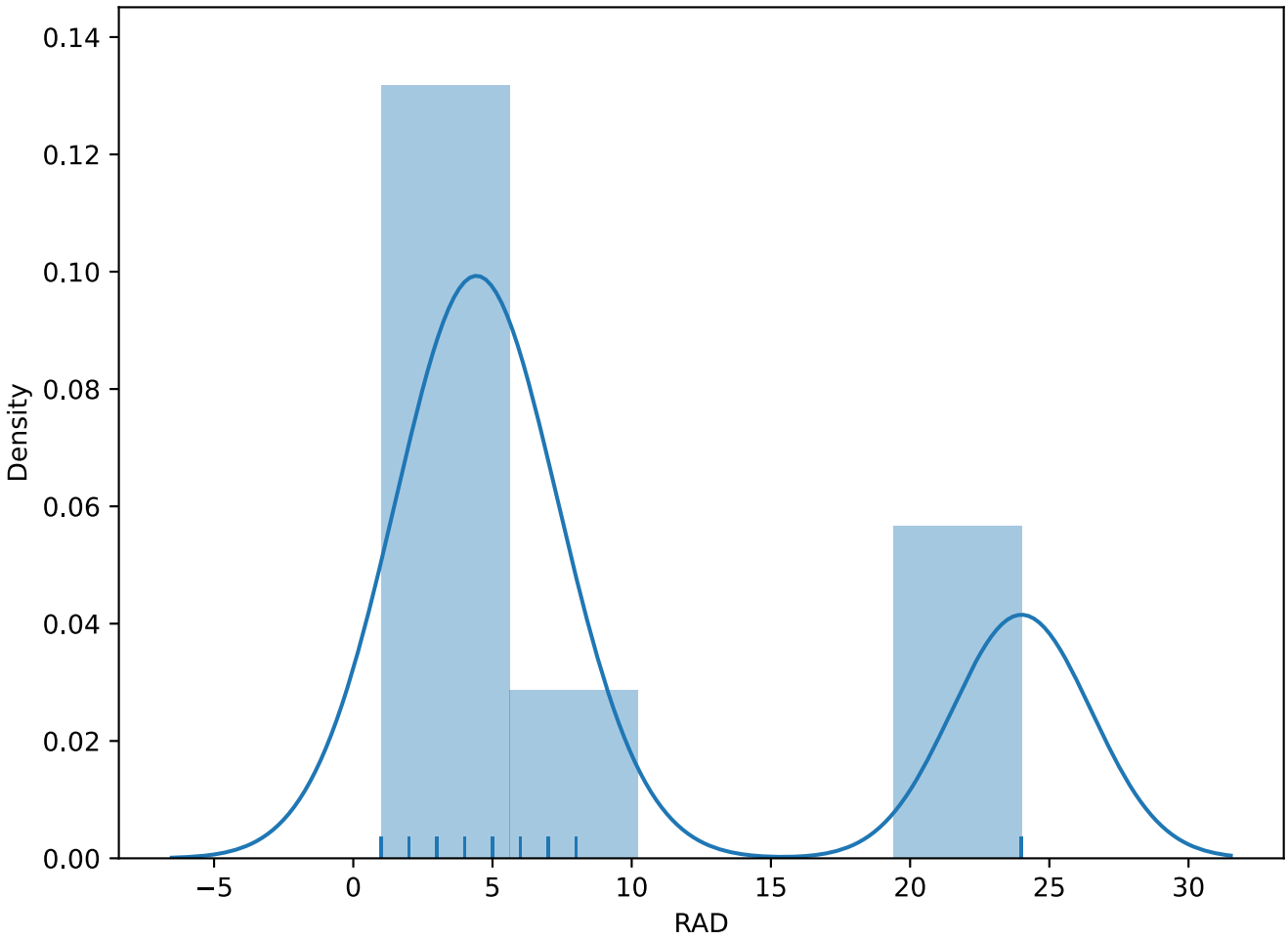

Density
20
15
10
5
0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
CHAS

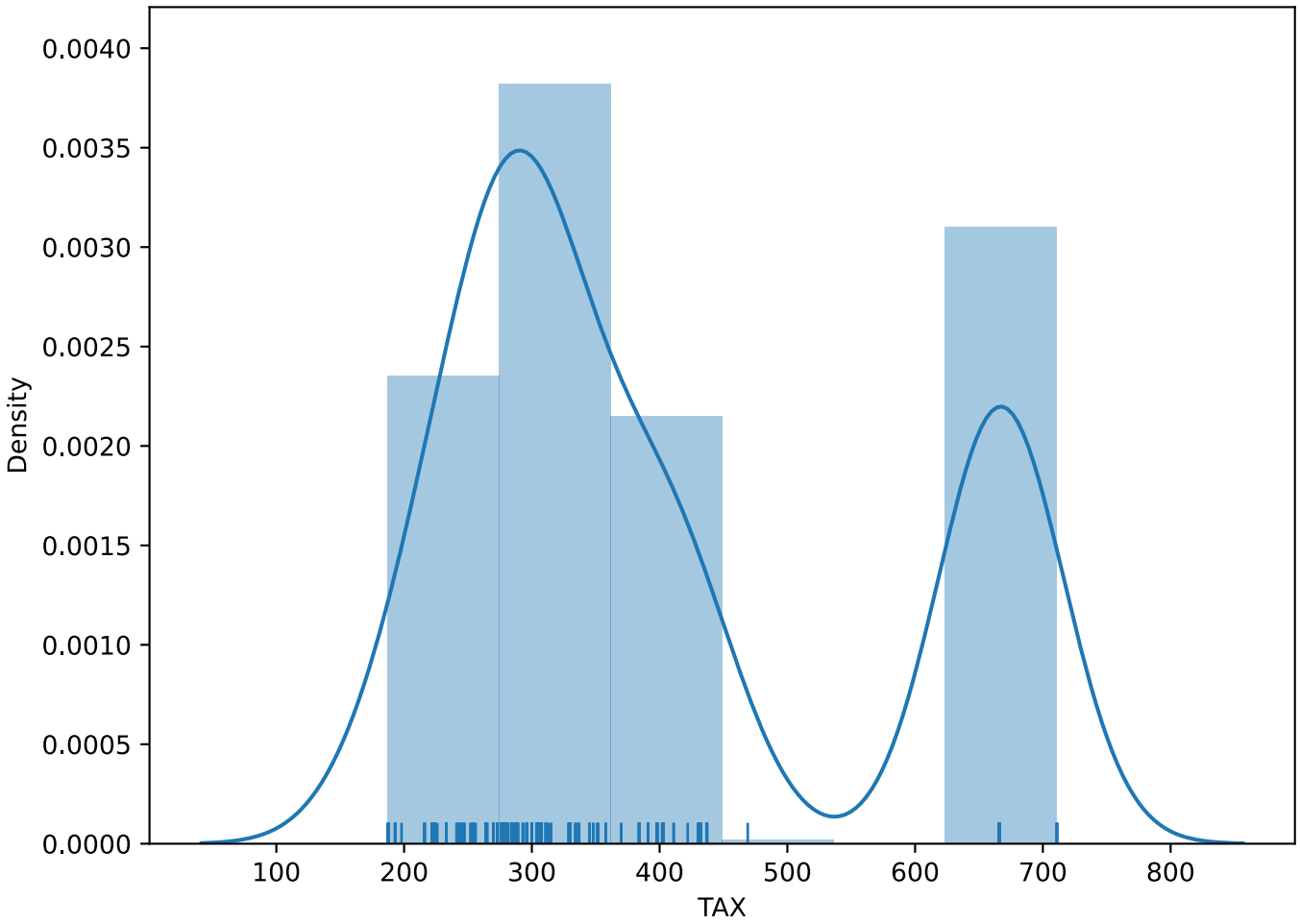

Density
NOX
0
1
2
3
4
5
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

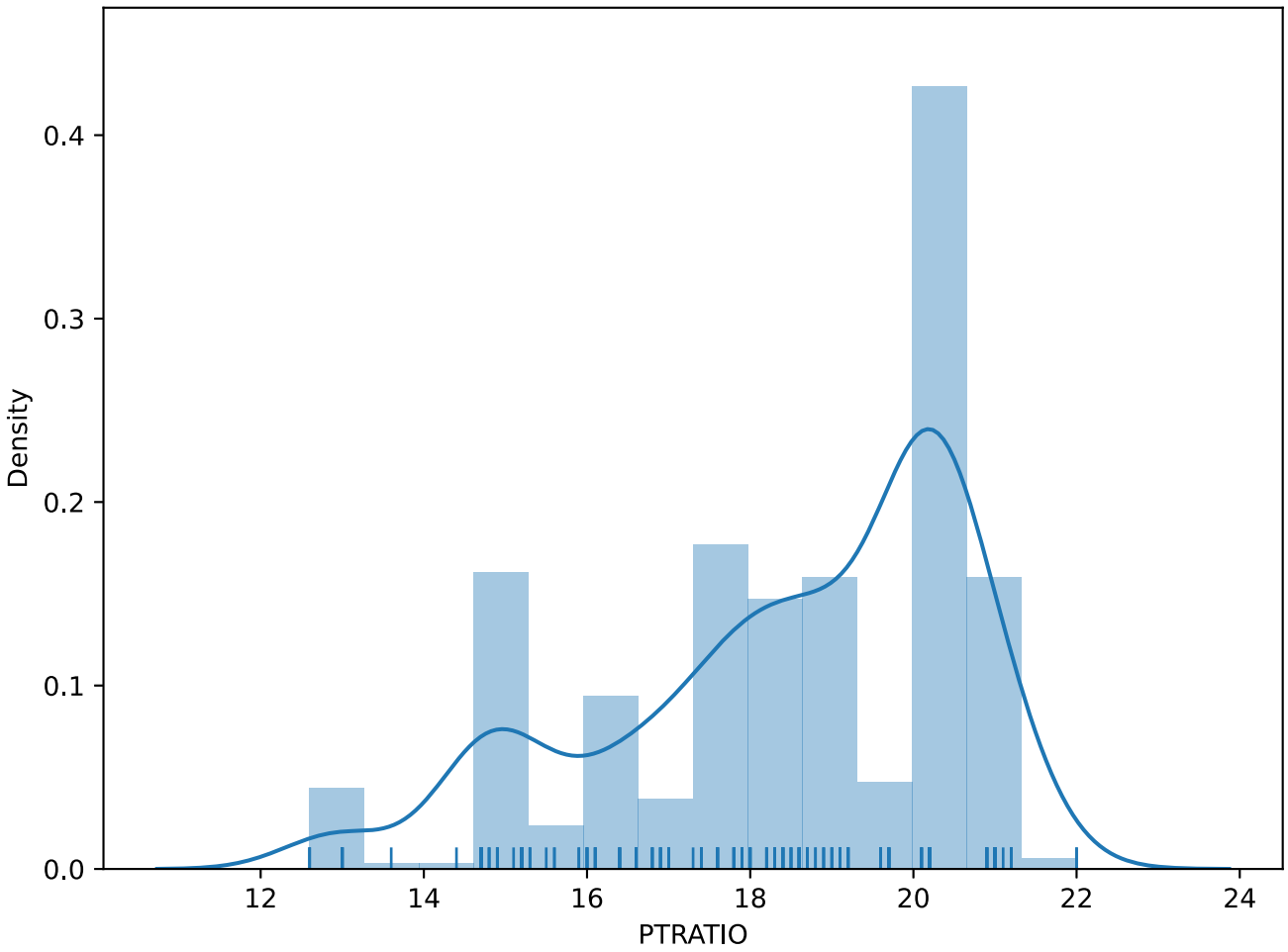

0.8
0.6
0.4
0.2
0.0
Density
3
4
5
6
7
8
9
RM

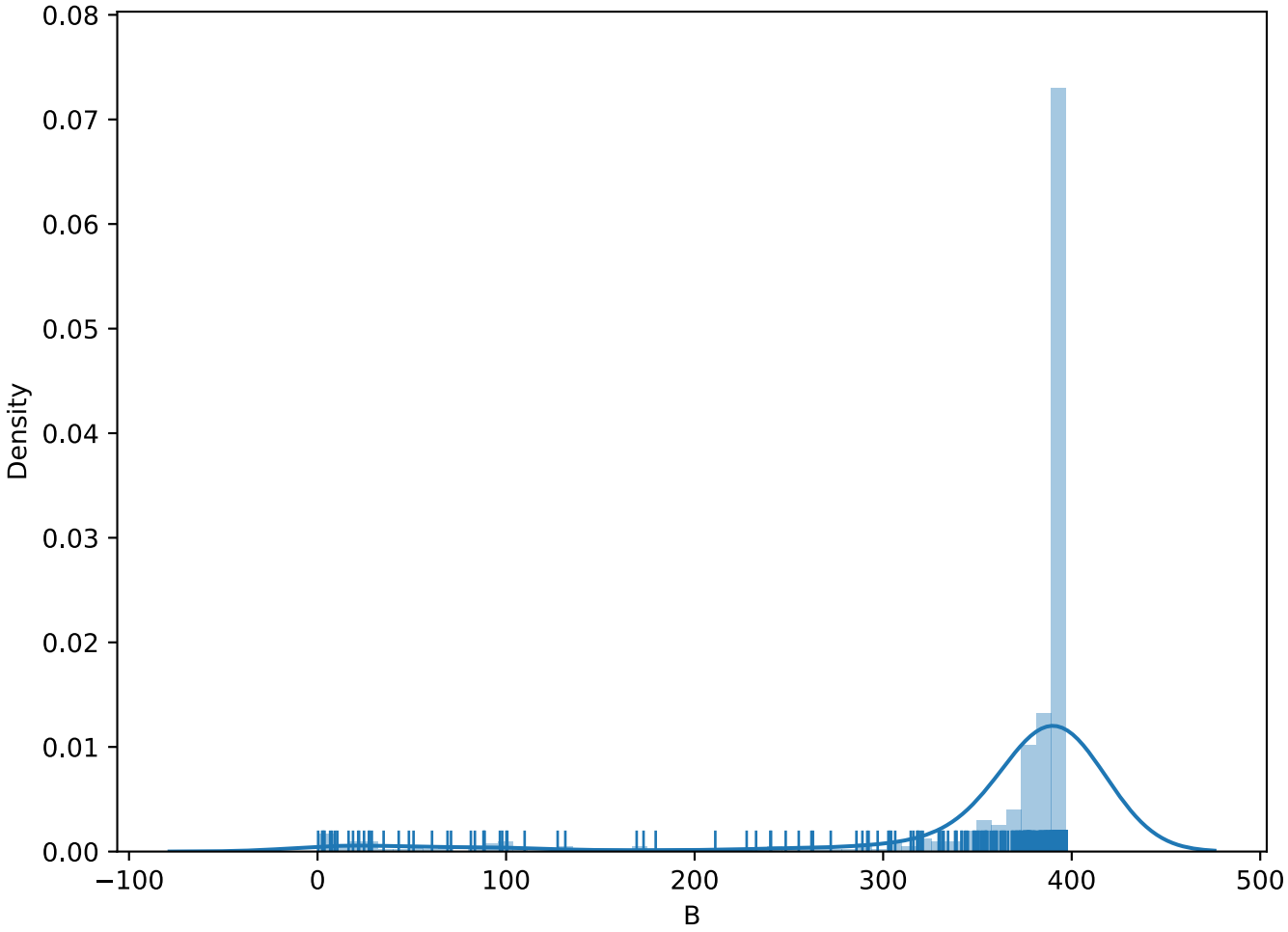

0.030
0.025
0.020
0.015
0.010
0.005
0.000
Density
−20
0
20
40
60
80
100
120
AGE

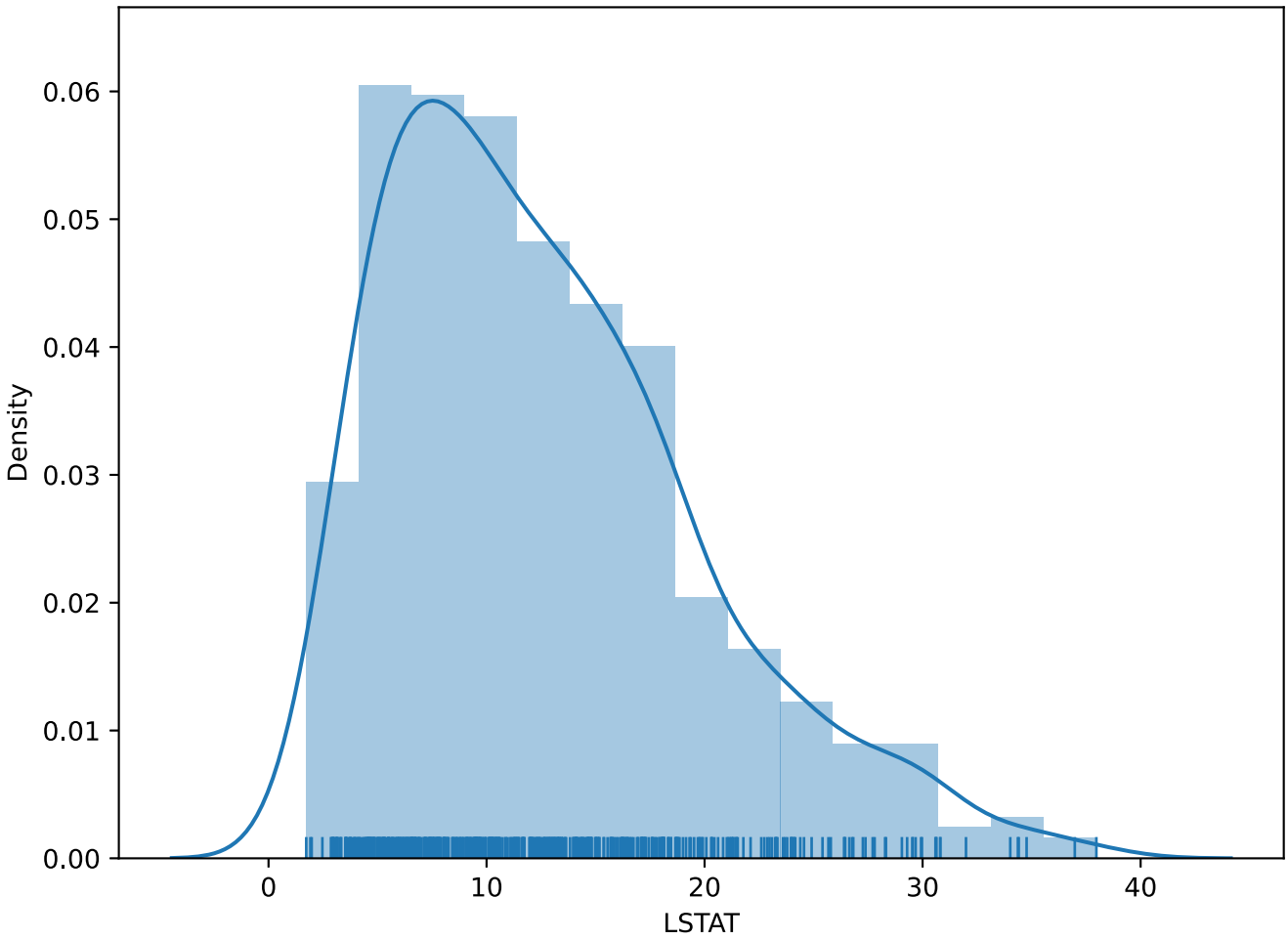

0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
0
2
4
6
8
10
12
14
DIS

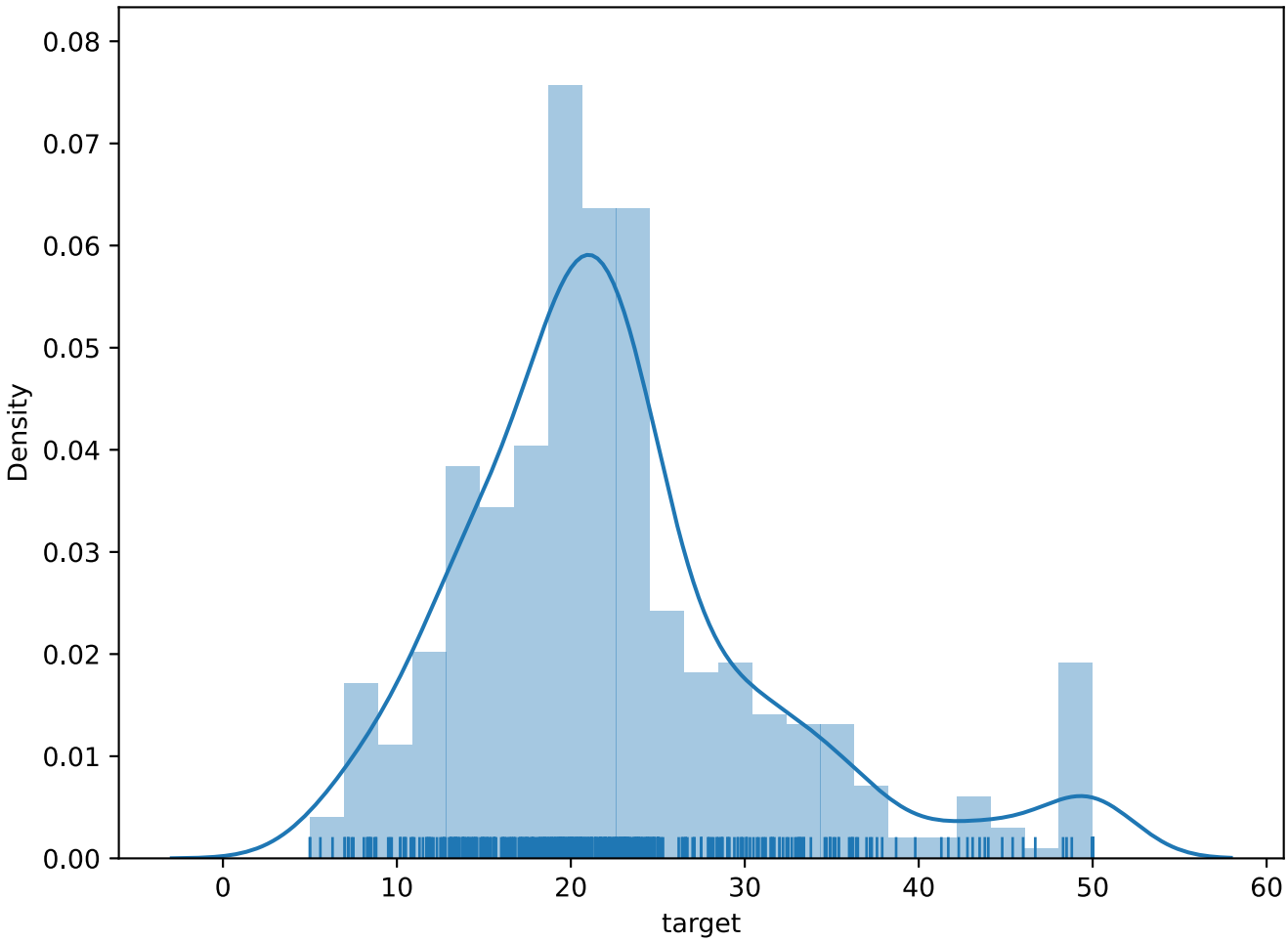

0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
−5
0
5
10
15
20
25
30
RAD


0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Density
100
200
300
400
500
600
700
800
TAX


0.4
0.3
0.2
0.1
0.0
Density
12
14
16
18
20
22
24
PTRATIO


0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
−100
0
100
200
300
400
500
B


0.06
0.05
0.04
0.03
0.02
0.01
0.00
Density
0
10
20
30
40
LSTAT



- `CRIM`, `ZN` and features are `left skewed`.
- `B` is `right skewed`.
- `target` is normally distributed.
- `CRIM`, `ZN` ,`CHAS`, `RN` ,`B` and `Target` seems to have outlier.

Removing Outliers and dropping columns

```
# removing Outliers
data = data[(np.abs(stats.zscore(data))<3).all(axis=1)]
```

```
#checking stats
data.describe()
```

```
          CRIM       ZN      INDUS     CHAS    NOX        RM       AGE       DIS
RAD        TAX    PTRATIO        B       LSTAT      target
count 415.000000 415.000000 415.000000 415.0 415.000000 415.000000 415.000000
```

```
415.000000 415.000000 415.000000 415.000000 415.000000 415.000000 415.000000
mean 2.452461 9.602410 10.829687 0.0     0.547168 6.267554 67.384337 3.852640
8.684337 393.853012 18.467952 374.235373 12.275976 22.353976
std     4.897241 19.677404 6.798613 0.0     0.110428 0.611602 28.072043 1.964763
8.196802 161.196133 2.137611 51.582784 6.474602 8.077366
min     0.006320 0.000000 0.460000 0.0     0.385000 4.368000 2.900000 1.169100
1.000000 188.000000 12.600000 83.450000 1.730000 5.600000
25%     0.081005 0.000000 5.130000 0.0     0.449000 5.886000 43.550000 2.203200
4.000000 279.000000 17.400000 377.880000 7.210000 17.550000
50%     0.211610 0.000000 8.560000 0.0     0.524000 6.193000 73.900000 3.377900
5.000000 315.000000 19.000000 392.630000 11.220000 21.200000
75%     2.152115 12.500000 18.100000 0.0     0.609000 6.559500 93.700000 5.287300
8.000000 437.000000 20.200000 396.900000 16.215000 24.800000
max     28.655800 80.000000 27.740000 0.0     0.871000 8.337000 100.000000 9.222900
24.000000 711.000000 21.200000 396.900000 31.990000 50.000000
```

CHAS feture is only containing '0'.

```
#droping CHAS feature as it only contains 0
data.drop('CHAS',axis=1, inplace=True)
```

Checking corelation with heatmap

```
# analyzing how features correlates with one and orther
corr_matrix = data.corr()
# plot heatmap
plt.figure(figsize=(12,8))
ax = sns.heatmap(corr_matrix, annot=True)
ax.set_title("Correlation Heatmap")
plt.plot()
```

- features like `LSTAT`, `PTRATIO`, `TAX`, `INDUS` and `RM` highly correlates with target variable. (IMP features).
- feature `TAX` and `RAD` have strong correlation index.
- feature `DIS` and `NOX` have strong correlation index.
- feature `CHAS` dosen't contribute much to prediction. Need to verify.

```
# analyzing how features correlates with one and orther
corr_matrix = data.corr()
# plot correation heatmap
plt.figure(figsize=(12,8))
ax = sns.heatmap(corr_matrix, annot=True)
ax.set_title("Correlation Heatmap")
plt.plot()
```

- Features like `INDUS`, `RM`, `TAX`, `PTRATIO` and `LSTAT` have high correlation(>.45) with target variable.

```
# Removing feature either correlation less tha 0.40 with target feature
features_to_remove = ['ZN', 'AGE', 'B']
data.drop(features_to_remove, axis=1, inplace=True)
```

```
# Removing skewness
for feature_name in data.columns[:-1]:
    data[feature_name] = data[feature_name] if -0.3 < data[feature_name].skew() <
0.3 else data[feature_name].apply(np.log1p)
```

```
# reset index of dataframe
data = data.reset_index(drop=True)
```

Scaling

```
#standerlizing feature into same scale

## creating feature and target datframe
features = data.drop('target',axis=1)
target = data['target']

# feature Dataframe after Standarlizaion
features =
pd.DataFrame(StandardScaler().fit_transform(features),columns=features.columns)
# print top 5 rows of new feature dataframe
features.head()
```

```
  CRIM       INDUS        NOX       RM      DIS      RAD       TAX      PTRATIO
LSTAT
0 -0.759508 -1.705866 -0.050211 0.541925 0.320472 -1.800543 -0.549683 -1.471021
-1.351689
1 -0.736617 -0.307677 -0.710817 0.293975 0.725112 -1.234989 -1.076992 -0.244291
-0.297911
2 -0.736638 -0.307677 -0.710817 1.476888 0.725112 -1.234989 -1.076992 -0.244291
-1.696918
3 -0.731169 -1.768726 -0.818990 1.197887 1.153962 -0.833722 -1.302687 0.157723
-2.184306
4 -0.692454 -1.768726 -0.818990 1.420712 1.153962 -0.833722 -1.302687 0.157723
-1.238184
```

```
target.head()
```

```
0    24.0
1    21.6
2    34.7
3    33.4
4    36.2
Name: target, dtype: float64
```

```
#save feature and target dataframe
features.to_csv('./data/features_init.csv', index=False)
target.to_csv('./data/target.csv', index=False)
```

dividing data into training and testing data

Now for this dataset, we can see that there are no NaN (missing) values and also all the data is in numbers rather than strings so we won't face any errors when training the model. So let us just divide our data into training data and testing data such that 70% of data is training data and the rest is testing data.

```
# Shuffle and split dataset into train and val
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=.3,
random_state=2)
```

Return to the Top

3. Choose a Model

For this particular problem, we are going to use four algorithms of supervised learning that can solve regression problems.

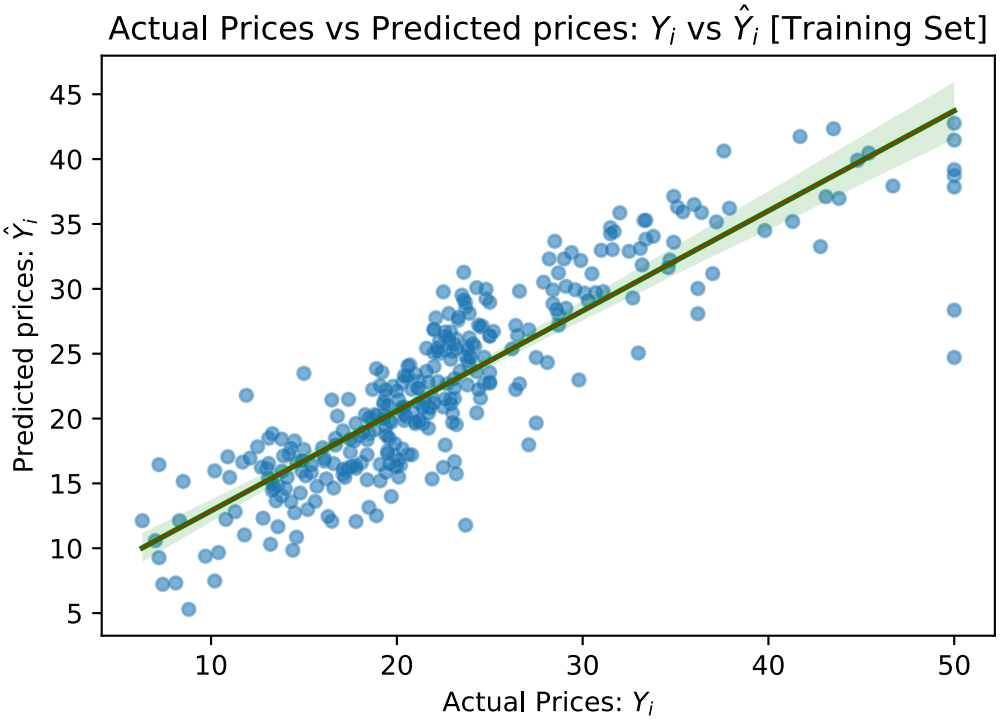
Linear Regression

```
lm = LinearRegression()
lm.fit(X_train, y_train)

print('Linear Regression coefficients: {}'.format(lm.coef_))
print('Linear Regression intercept: {}'.format(lm.intercept_))

# model evaluation for training set
y_train_predict = lm.predict(X_train)


# calculating the intercept and slope for the regression line
b, m = np.polynomial.polynomial.polyfit(y_train, y_train_predict, 1)
```

Linear Regression coefficients: [-1.19237058 -0.43461219 -1.28249355 1.99532108 -2.82830797 2.03729459 -1.46873254 -1.72330057 -4.43306807]
Linear Regression intercept: 22.37939746081968

```
sns.scatterplot(y_train, y_train_predict, alpha=0.4)
sns.regplot(y_train, y_train_predict, truncate=True, scatter_kws={'s': 20,
'alpha':0.3}, line_kws={'color':'green', 'linewidth': 2})
sns.lineplot(np.unique(y_train), np.unique(np.poly1d(b + m * np.unique(y_train))),
linewidth=0.5, color='r')
```

```
plt.xlabel("Actual Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Actual Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$ [Training Set]")

plt.show()
```



Return to the Top

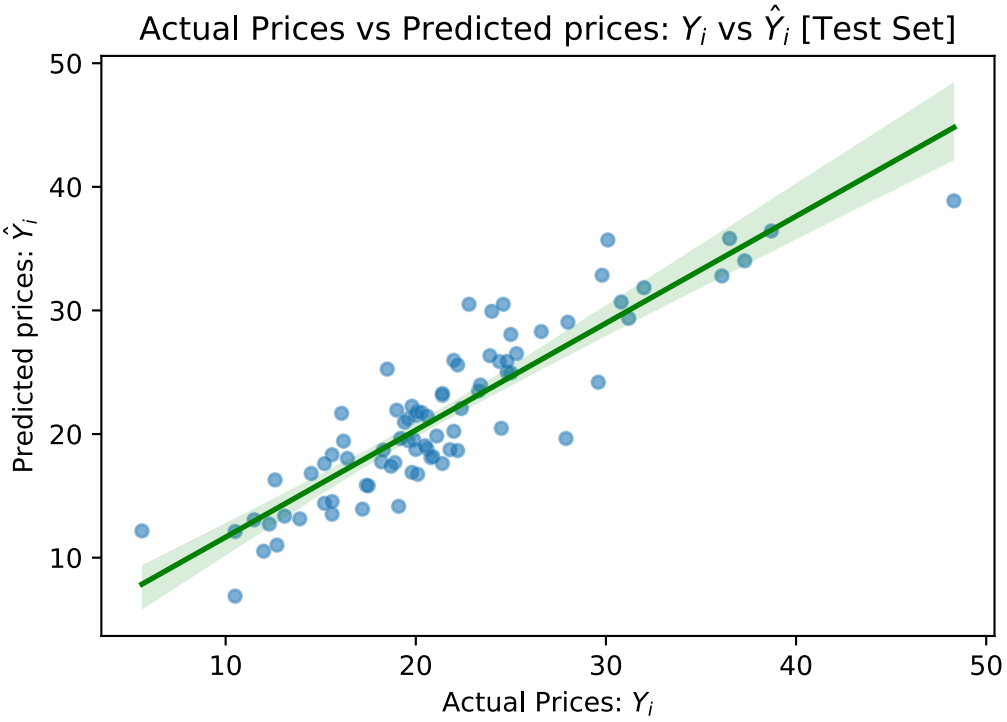
4. Evaluating the model

For evaluating the model we are going to use the R_square and mean_squared_error() method from the scikit-learn library.

```
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = r2_score(y_train, y_train_predict)

print("The linear model performance for training set")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

The linear model performance for training set
RMSE is 3.980665646517504
R2 score is 0.7710482780699368

```
# model evaluation for testing set
y_test_predict = lm.predict(X_test)

sns.scatterplot(y_test, y_test_predict, alpha=0.4)
sns.regplot(y_test, y_test_predict, truncate=True, scatter_kws={'s': 20,
'alpha':0.3}, line_kws={'color':'green', 'linewidth': 2})

plt.xlabel("Actual Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Actual Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$ [Test Set]")

plt.show()
```

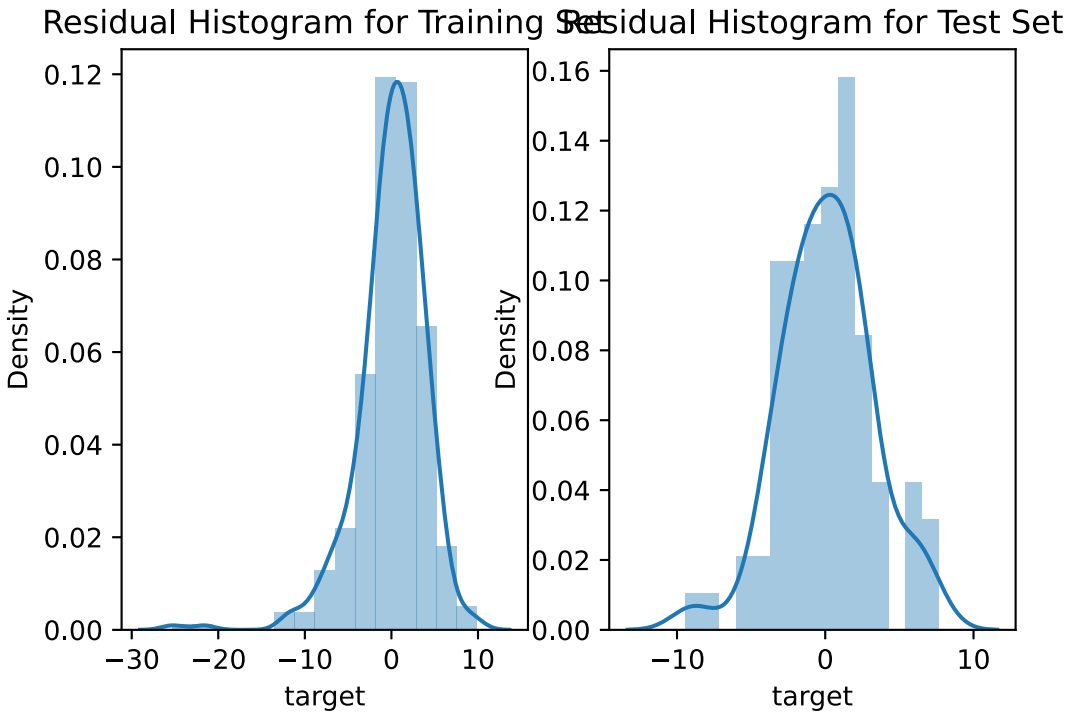



```
# root mean square error of the model
rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))

# r-squared score of the model
r2 = r2_score(y_test, y_test_predict)

print("\nThe linear model performance for testing set")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

The linear model performance for testing set
RMSE is 3.125971553207074

R2 score is 0.7940285885971254

```
y_train_residual = y_train_predict - y_train
y_test_residual = y_test_predict - y_test

plt.subplot(1, 2, 1)
sns.distplot(y_train_residual, bins=15)
plt.title('Residual Histogram for Training Set')

plt.subplot(1, 2, 2)
sns.distplot(y_test_residual, bins=15)
plt.title('Residual Histogram for Test Set')

plt.show()
```

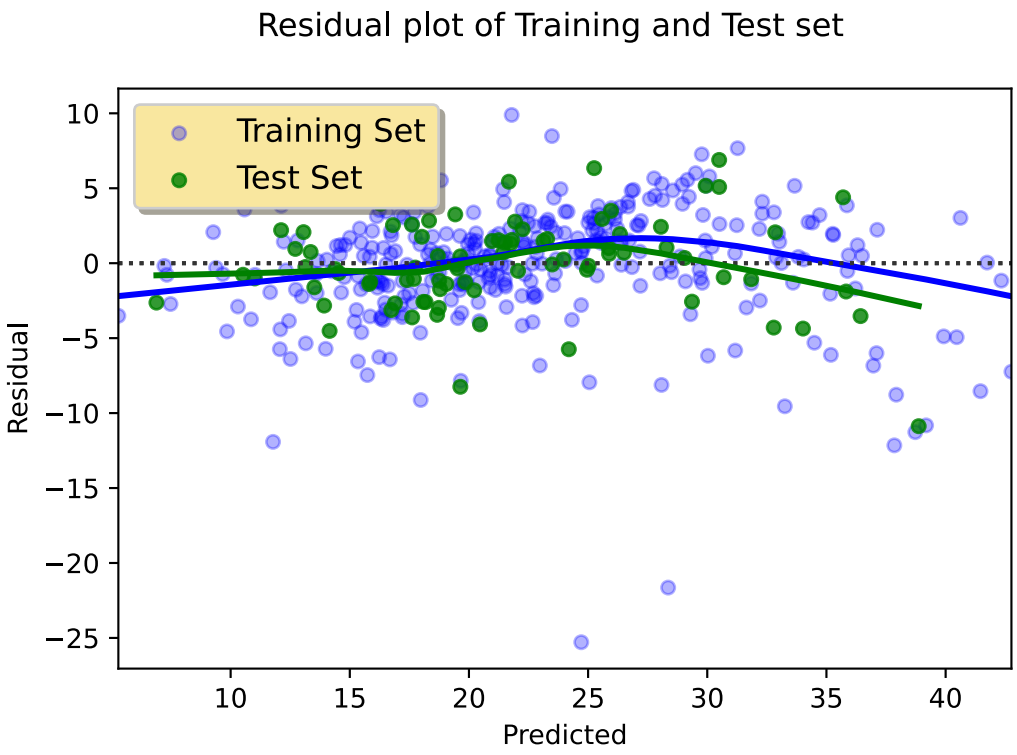


fig, axes = plt.subplots()

fig.suptitle('Residual plot of Training and Test set')

```
# Plot the residuals after fitting a linear model
sns.residplot(y_train_predict, y_train_residual, lowess=True, color="b", ax=axes,
label='Training Set',
              scatter_kws={'s': 25, 'alpha':0.3})

sns.residplot(y_test_predict, y_test_residual, lowess=True, color="g", ax=axes,
label='Test Set',
              scatter_kws={'s': 25})
```

```
legend = axes.legend(loc='upper left', shadow=True, fontsize='large')
legend.get_frame().set_facecolor('#f9e79f')

plt.xlabel('Predicted')
plt.ylabel('Residual')
plt.show()
```



We can conclude that the straight regression line is unable to capture the patterns in the data. This is an example of underfitting. To overcome underfitting, we need to increase the complexity of the model. This could be done by converting the original features into their higher order polynomial terms by using the PolynomialFeatures class provided by scikit-learn. Next, we train the model using Linear Regression.

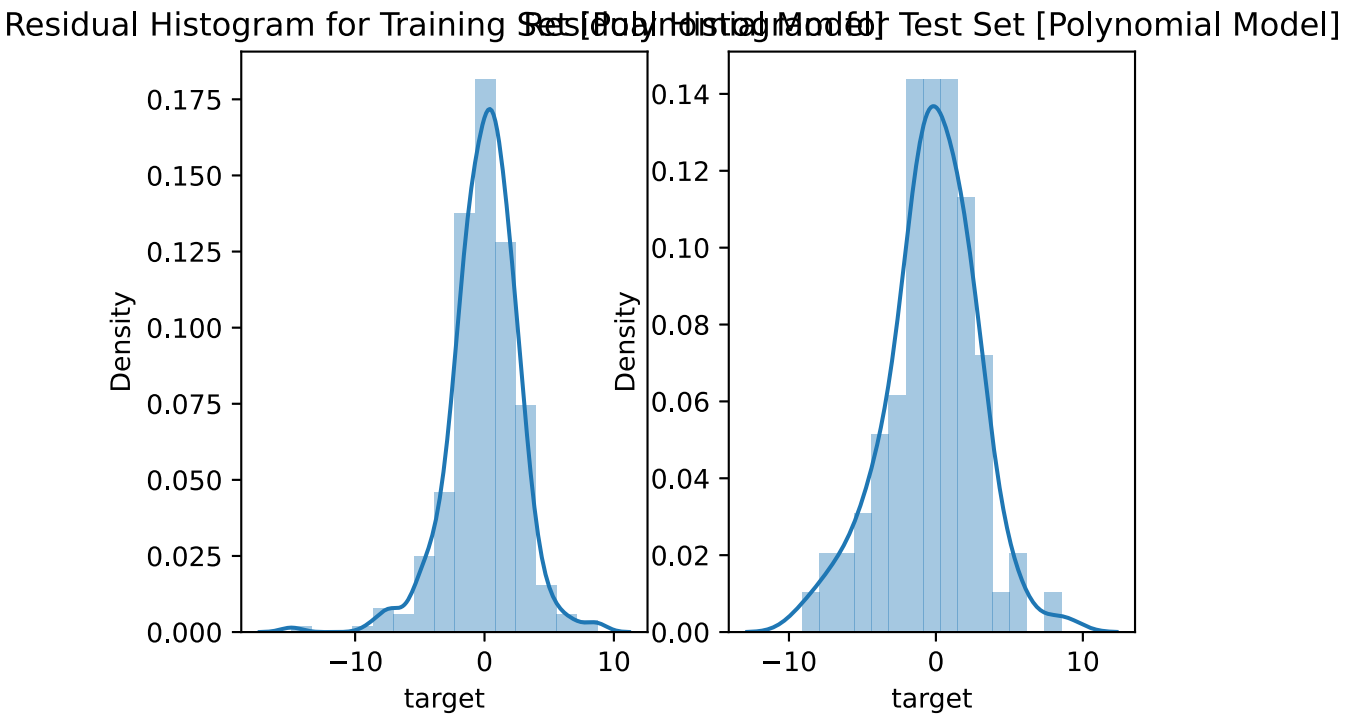
Return to the Top

polynomial regression

```
"Creates a polynomial regression model for the given degree"
poly_features = PolynomialFeatures(degree=2)

# transform the features to higher degree features.
X_train_poly = poly_features.fit_transform(X_train)

# fit the transformed features to Linear Regression
poly_model = LinearRegression()
```

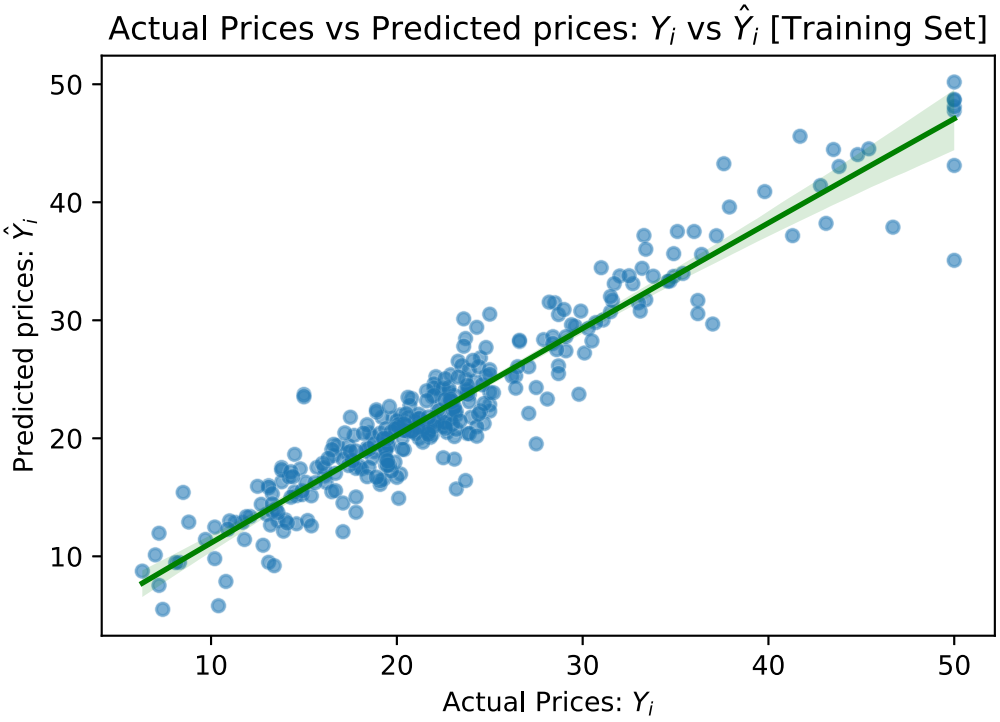
```
poly_model.fit(X_train_poly, y_train)

# predicting on training data-set
y_train_predicted = poly_model.predict(X_train_poly)

# predicting on test data-set
y_test_predicted = poly_model.predict(poly_features.fit_transform(X_test))
```

```
y_train_residual = y_train_predicted - y_train
y_test_residual = y_test_predicted - y_test

plt.subplot(1, 2, 1)
sns.distplot(y_train_residual, bins=15)
plt.title('Residual Histogram for Training Set [Polynomial Model]')

plt.subplot(1, 2, 2)
sns.distplot(y_test_residual, bins=15)
plt.title('Residual Histogram for Test Set [Polynomial Model]')

plt.show()
```

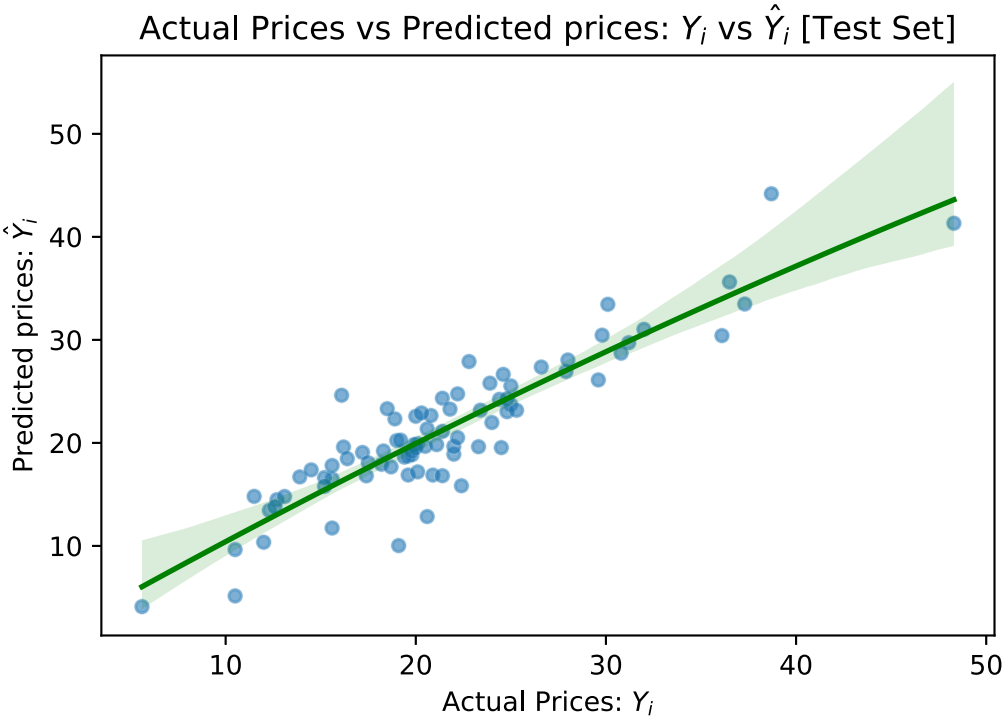



```
sns.scatterplot(y_train, y_train_predicted, alpha=0.4)
sns.regplot(y_train, y_train_predicted, scatter_kws={'s': 20, 'alpha':0.3},
line_kws={'color':'green', 'linewidth': 2}, order=2)
```

```
plt.xlabel("Actual Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Actual Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$ [Training Set]")

plt.show()
```




```
sns.scatterplot(y_test, y_test_predicted, alpha=0.4)
sns.regplot(y_test, y_test_predicted, scatter_kws={'s': 20, 'alpha':0.3}, line_kws=
{'color':'green', 'linewidth': 2}, order=2)

plt.xlabel("Actual Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Actual Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$ [Test Set]")

plt.show()
```




```
# evaluating the model on training data-set
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_predicted))
r2_train = r2_score(y_train, y_train_predicted)

print("The polynomial model performance for the training set")
print("RMSE of training set is {}".format(rmse_train))
print("R2 score of training set is {}".format(r2_train))
```

The polynomial model performance for the training set
RMSE of training set is 2.6106219823373875
R2 score of training set is 0.9015262461391217

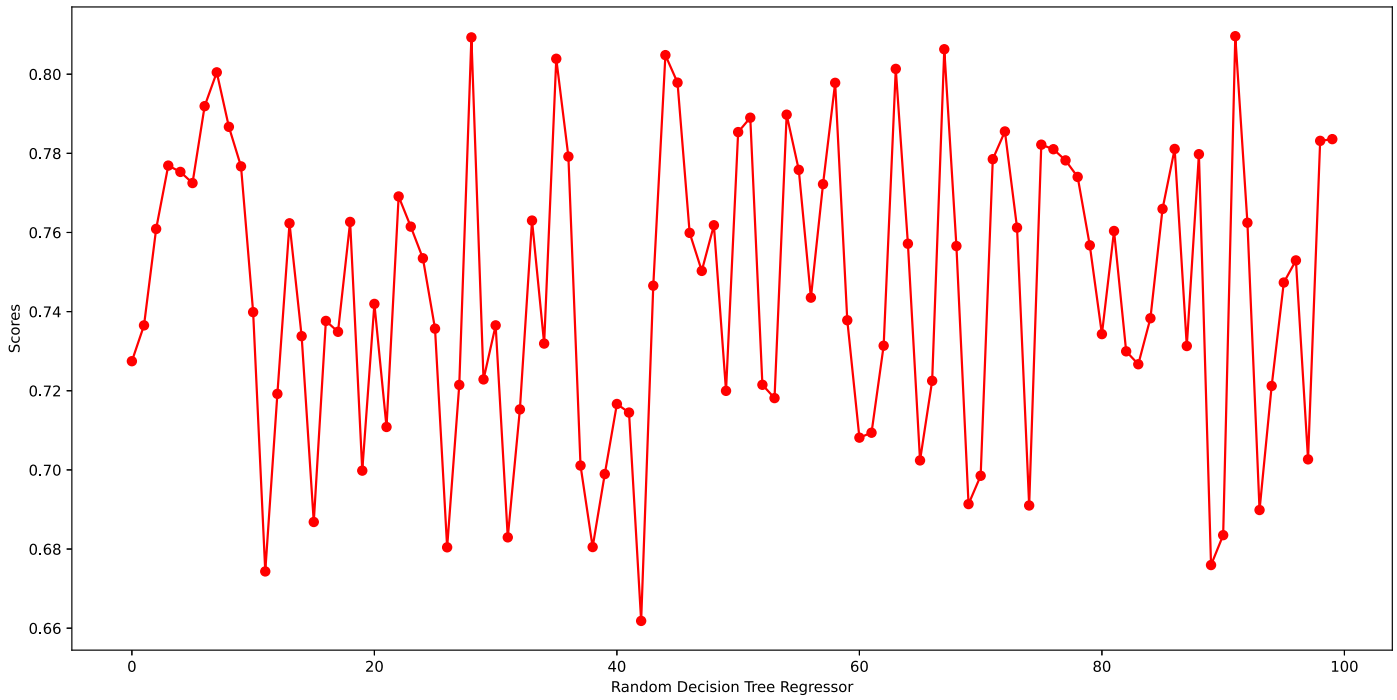
```
# evaluating the model on test data-set
rmse_test = np.sqrt(mean_squared_error(y_test, y_test_predicted))
r2_test = r2_score(y_test, y_test_predicted)

print("The polynomial model performance for the test set")
print("RMSE of test set is {}".format(rmse_test))
print("R2 score of test set is {}".format(r2_test))
```

The polynomial model performance for the test set
RMSE of test set is 3.0403295716752656
R2 score of test set is 0.805159951409818

Return to the Top

Decision Tree Regression

```
from sklearn.tree import DecisionTreeRegressor
scores = []
for i in range(100):

    dtr_model = DecisionTreeRegressor(max_depth=None, random_state=i)
    dtr_model.fit(X_train, y_train)
    scores.append(r2_score(y_test, dtr_model.predict(X_test)))

plt.figure(figsize = (16, 8))
plt.plot(list(range(100)), scores, 'ro-')
plt.xlabel('Random Decision Tree Regressor')
plt.ylabel('Scores')
plt.show()
```



See how the decision tree score changes for different random states

```
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size =
0.20, random_state = 42)

dtr_model = DecisionTreeRegressor(max_depth=23, random_state=3)
dtr_model.fit(X_train, y_train)
```

```
print('Training Score : ', dtr_model.score(X_train, y_train))
print('Testing Score  : ', dtr_model.score(X_test, y_test))

print('R2 Score : ', r2_score(y_test, dtr_model.predict(X_test)))
print('MSE : ', mean_squared_error(y_test, dtr_model.predict(X_test)))
```

Training Score : 1.0
Testing Score : 0.623187418263587
R2 Score : 0.623187418263587
MSE : 17.876746987951808

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.pipeline import make_pipeline
adtr_model = make_pipeline(MinMaxScaler(), DecisionTreeRegressor(max_depth = 12,
random_state = 92))
adtr_model.fit(X_train, y_train)

print('Training Score : ', adtr_model.score(X_train, y_train))
print('Testing Score  : ', adtr_model.score(X_test, y_test))

print('R2 Score : ', r2_score(y_test, adtr_model.predict(X_test)))
print('MSE : ', mean_squared_error(y_test, adtr_model.predict(X_test)))
```

Training Score : 0.9985709928931329
Testing Score : 0.6364905646801227
R2 Score : 0.6364905646801227
MSE : 17.245618957310725

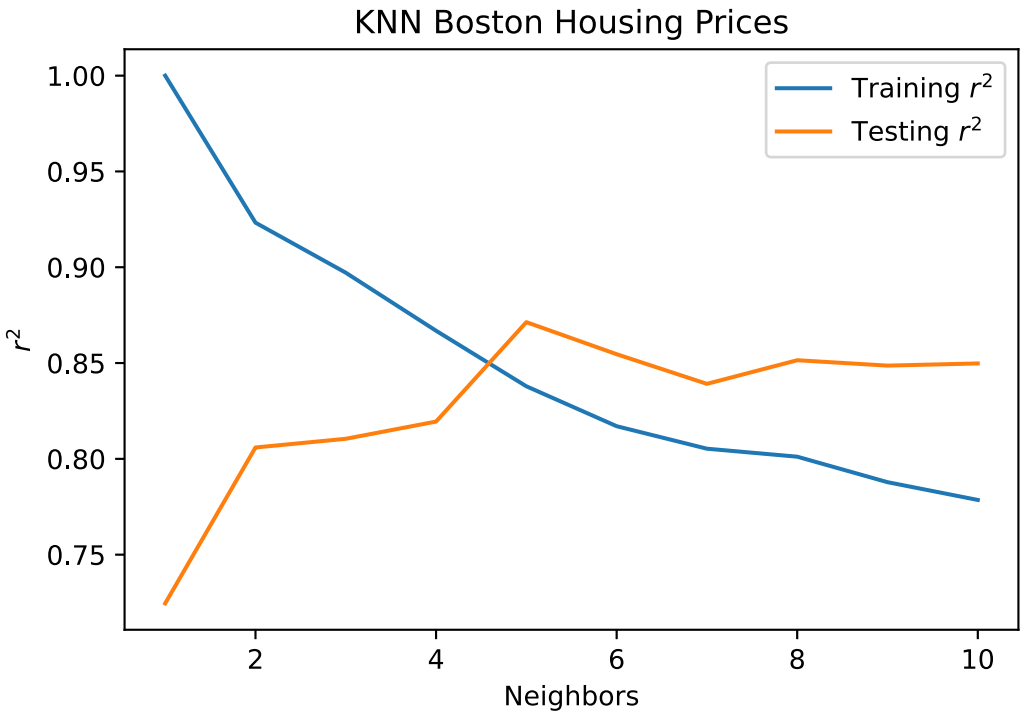
Return to the Top

K Nearest Neighbors Regression

```
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as pyplot
def get_r_squared(max_neighbors=10):
    train_score = []
    test_score = []
    models = []
    for neighbors in range(1, max_neighbors+1):
        model = KNeighborsRegressor(n_neighbors=neighbors)
        model.fit(X_train, y_train)
        train_score.append(model.score(X_train, y_train))
        test_score.append(model.score(X_test, y_test))
```

```
            models.append(model)
    return train_score, test_score, models
```

```
train_score, test_score, models = get_r_squared()
neighbors = range(1, 11)
pyplot.plot(neighbors, train_score, label="Training $r^2$")
pyplot.plot(neighbors, test_score, label="Testing $r^2$")
pyplot.xlabel("Neighbors")
pyplot.ylabel("$r^2$")
pyplot.title("KNN Boston Housing Prices")
pyplot.legend()
```




```
print("Training r2 for 2 neigbors: {:.2f}".format(train_score[1]))
print("Testing r2 for 2 neighbors: {:.2f}".format(test_score[1]))
```

Training r2 for 2 neigbors: 0.92
Testing r2 for 2 neighbors: 0.81

Return to the Top

5. Prediction

Now we can use the models to predict the prices of the houses using the predict function as we did above. Make sure when predicting the prices that we are given all the features that were present when training the model.

Conclusion

We achieved a r2 score of above .80, which is a fairly a good number. Analysed different algorithm and chooses best that generalized the data in a good manner.

The data in which it is trained is of 1978 and guessing the price of house, from the data that is from 1970's is not going to be good prediction.

The features present in dataset are not enough to describe or predict the price of house, as price of houses depend on several other parameter such as area, furnishing, education services,etc

As the data we trained is only from urban city like Boston, and price of house of urban city and rural city are way to different so predicting the value of rural area house from this model might not be a good idea.

Return to the Top

Implementation of a Classification problem

This data set consists of the physical parameters of three species of flower — Versicolor, Setosa and Virginica. The numeric parameters which the dataset contains are Sepal width, Sepal length, Petal width and Petal length. In this data we will be predicting the classes of the flowers based on these parameters.The data consists of continuous numeric values which describe the dimensions of the respective features. We will be training the model based on these features. I choose Iris dataset because:

- Famous dataset for machine learning because prediction is easy
- Framed as a supervised learning problem: Predict the species of an iris using the measurements
- It already come with scikit-learn For more detail about the dataset visit: http://archive.ics.uci.edu/ml/datasets/Iris. I will download the data and load as csv file.

Return to the Top

1. Gathering data:

Import libraries

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix, classification_report
```

Load dataset Iris and get info

```
iris = pd.read_csv('Iris.csv')

iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Return to the Top

2. Preprocess data

Drop column 'Id'

```
iris.drop(columns='Id', inplace=True) # delete column'Id'
iris.head(10)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

Identify the shape of datatset

```
iris.shape # View the shape/dimension of the dataset (rows, columns)
```

(150, 5)

Get the list of columns

```
iris.columns # View list of column names
```

```
Index(['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

Missing values

```
iris.isna().values.any() # Check missing value
```

False

Get new information from dataset

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Identify duplicate entries/rows

```
iris[iris.duplicated(keep=False)] # show whole row with duplication
```

```
    SepalLengthCm    SepalWidthCm    PetalLengthCm   PetalWidthCm    Species
34      4.9             3.1             1.5             0.1        Iris-setosa
37      4.9             3.1             1.5             0.1        Iris-setosa
101     5.8             2.7             5.1             1.9        Iris-virginica
142     5.8             2.7             5.1             1.9        Iris-virginica
```

```
iris.duplicated().value_counts() # count the number of data duplication
```

False 147
True 3
dtype: int64

Drop duplicate entries/rows

```
iris.drop_duplicates(inplace=True) # remove duplication of data
iris.shape
```

(147, 5)

Describe the dataset

```
iris.describe() # data description
```

```
       SepalLengthCm SepalWidthCm PetalLengthCm  PetalWidthCm
count    147.000000  147.000000   147.000000     147.000000
mean     5.856463    3.055782     3.780272       1.208844
std      0.829100    0.437009     1.759111       0.757874
min      4.300000    2.000000     1.000000       0.100000
25%      5.100000    2.800000     1.600000       0.300000
50%      5.800000    3.000000     4.400000       1.300000
75%      6.400000    3.300000     5.100000       1.800000
```

```
max     7.900000    4.400000     6.900000       2.500000
```

```
iris.corr() # correlation between columns
```

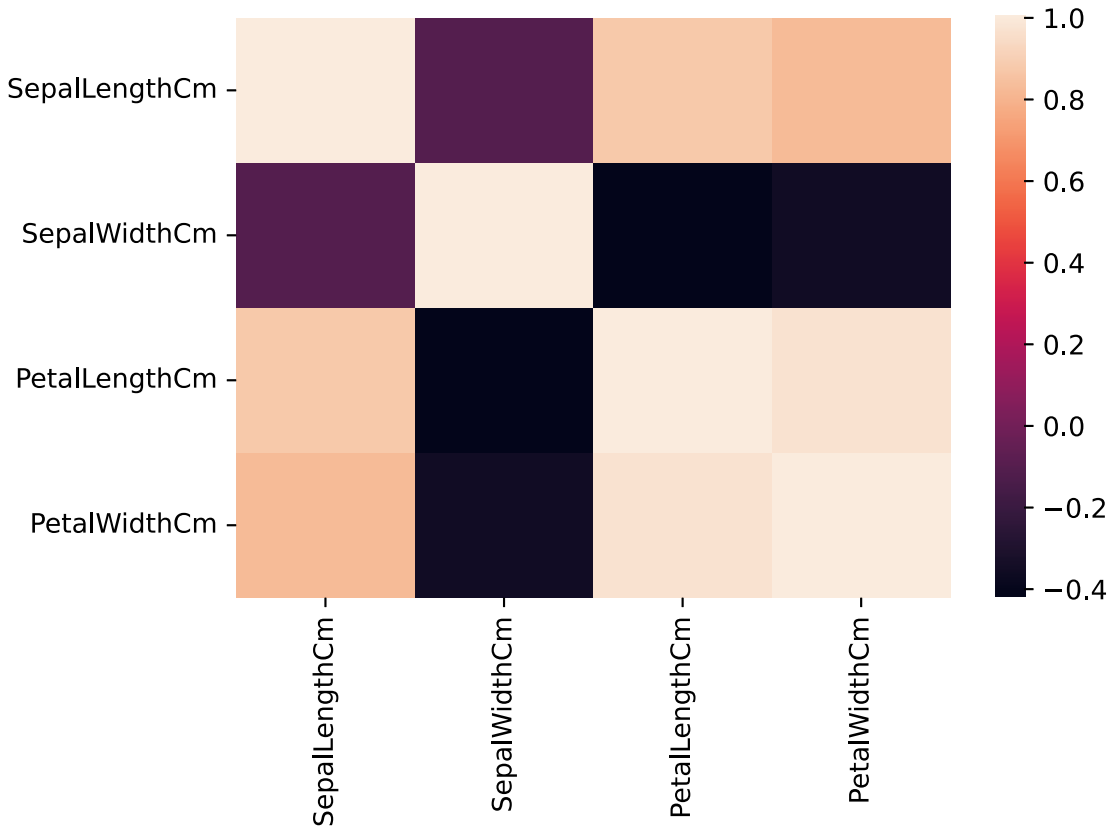
```
              SepalLengthCm  SepalWidthCm    PetalLengthCm   PetalWidthCm
SepalLengthCm     1.000000    -0.109321         0.871305       0.817058
SepalWidthCm     -0.109321     1.000000        -0.421057      -0.356376
PetalLengthCm     0.871305    -0.421057         1.000000       0.961883
PetalWidthCm      0.817058    -0.356376         0.961883       1.000000
```

Return to the Top

Data Visualisation

Heatmap

```
sns.heatmap(data=iris.corr())
```

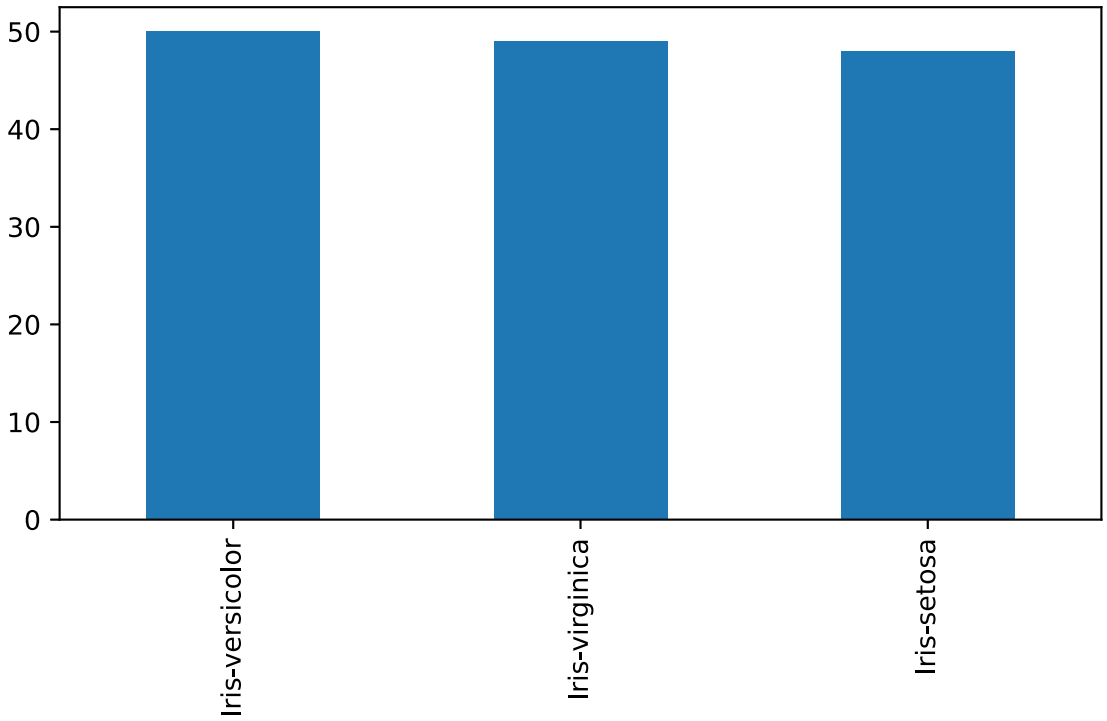


Bar Plot

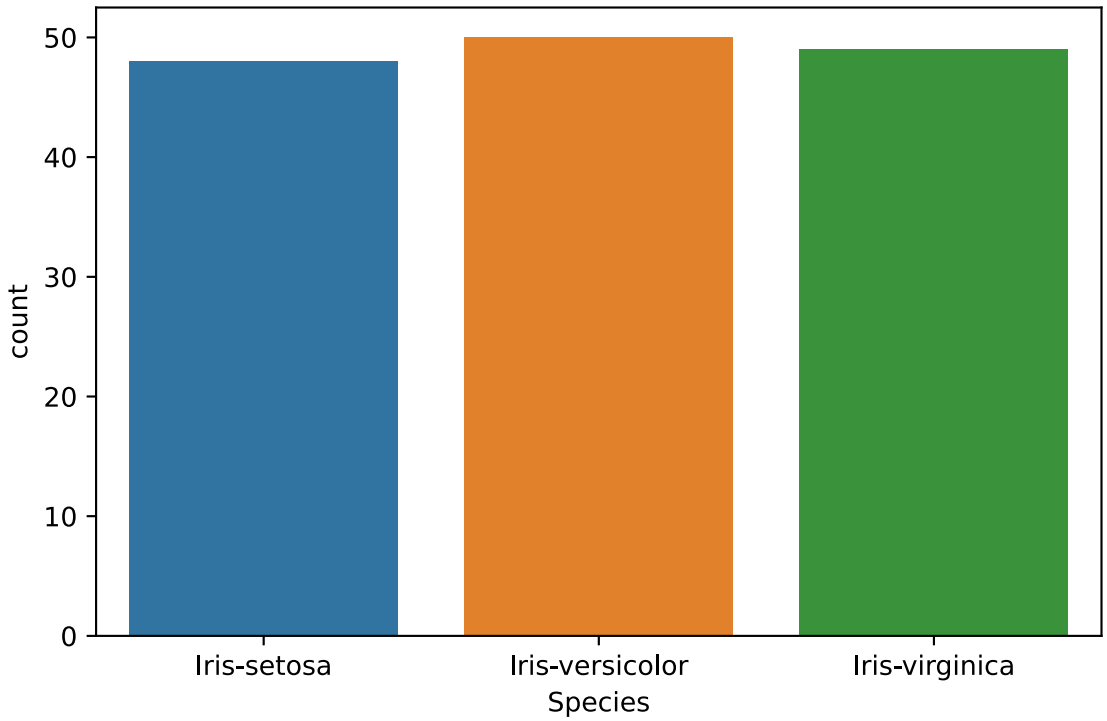
```
iris['Species'].value_counts() # count the number of each species
```

```
Iris-versicolor    50
Iris-virginica     49
Iris-setosa        48
Name: Species, dtype: int64
```

```
iris['Species'].value_counts().plot.bar()
plt.tight_layout()
plt.show()
```

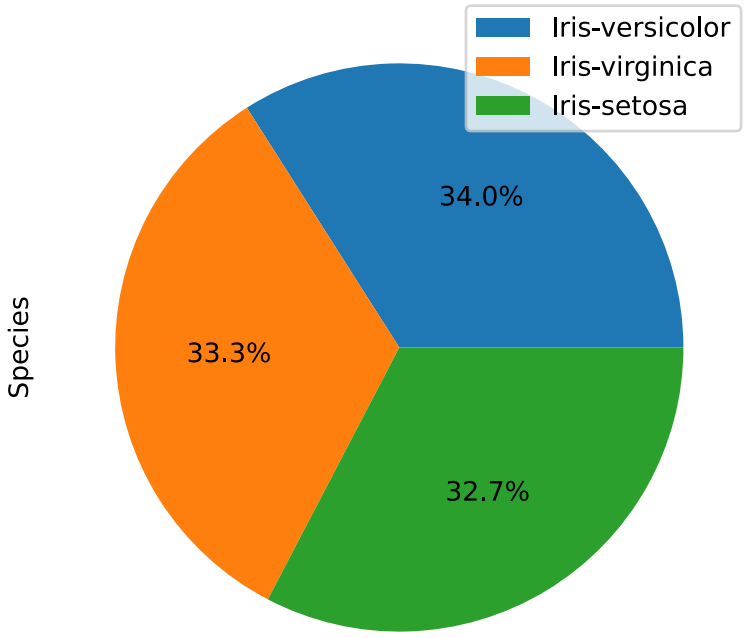



```
sns.countplot(data=iris, x='Species')
plt.tight_layout()
```



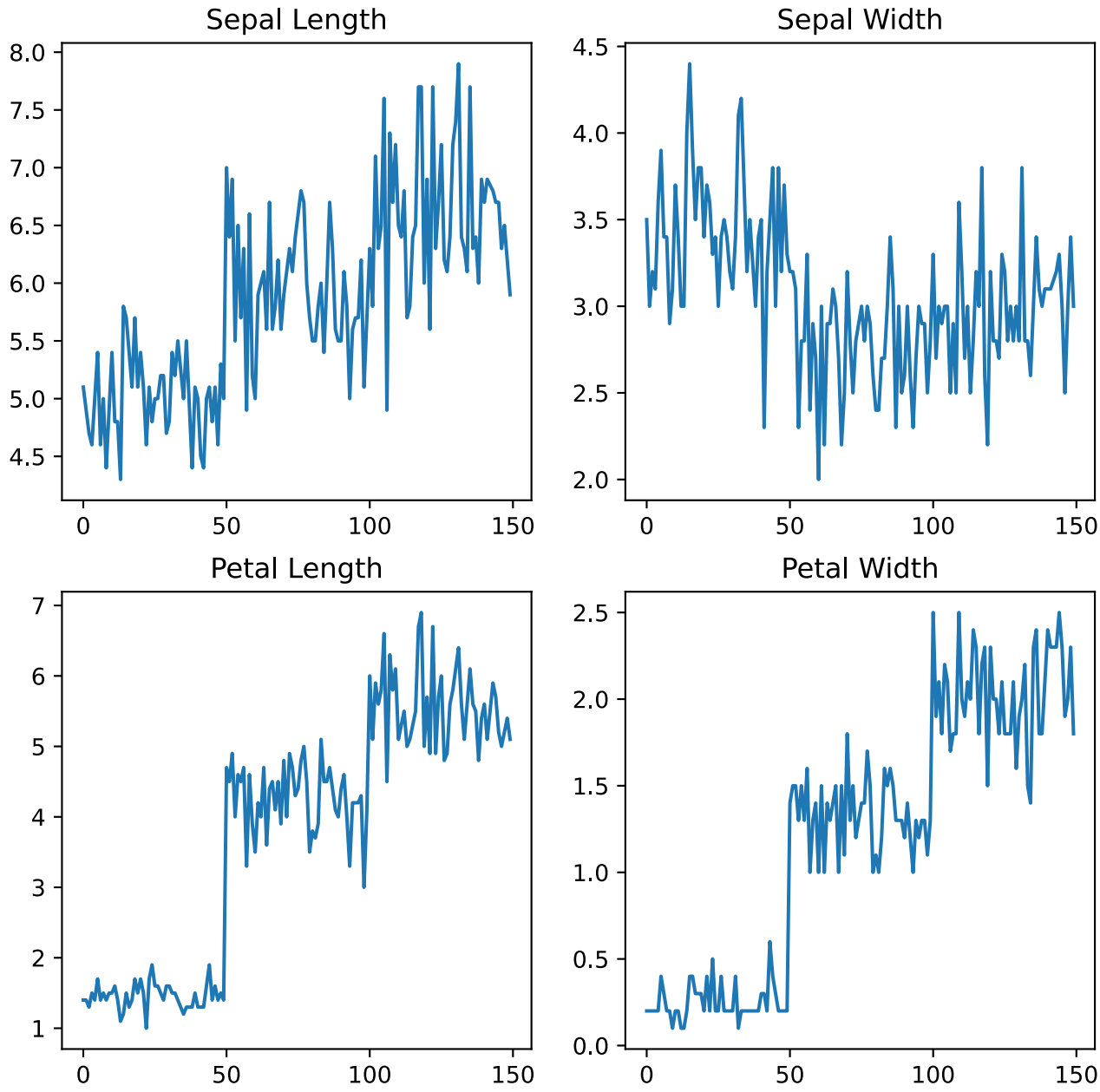
Pie Chart

```
iris['Species'].value_counts().plot.pie(autopct='%1.1f%%', labels=None, legend=True)
plt.tight_layout()
```

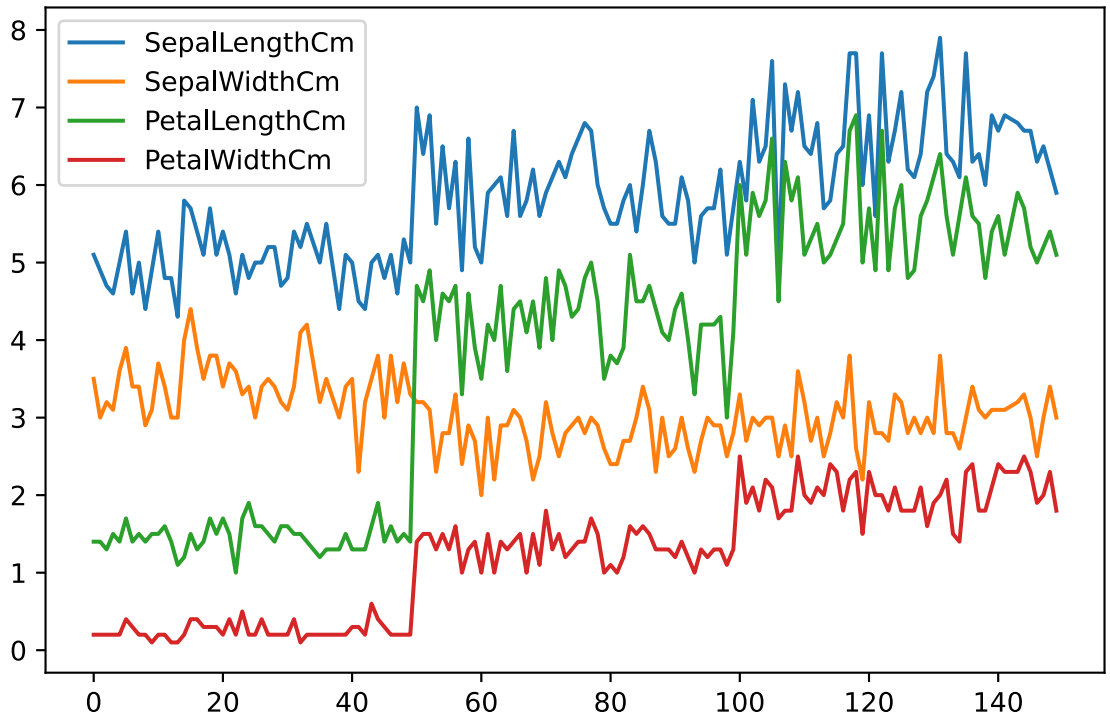


Line Plot

```
fig,ax = plt.subplots(nrows=2, ncols=2, figsize=(8,8))

iris['SepalLengthCm'].plot.line(ax=ax[0][0])
ax[0][0].set_title('Sepal Length')

iris['SepalWidthCm'].plot.line(ax=ax[0][1])
ax[0][1].set_title('Sepal Width')

iris.PetalLengthCm.plot.line(ax=ax[1][0])
ax[1][0].set_title('Petal Length')

iris.PetalWidthCm.plot.line(ax=ax[1][1])
ax[1][1].set_title('Petal Width')
```

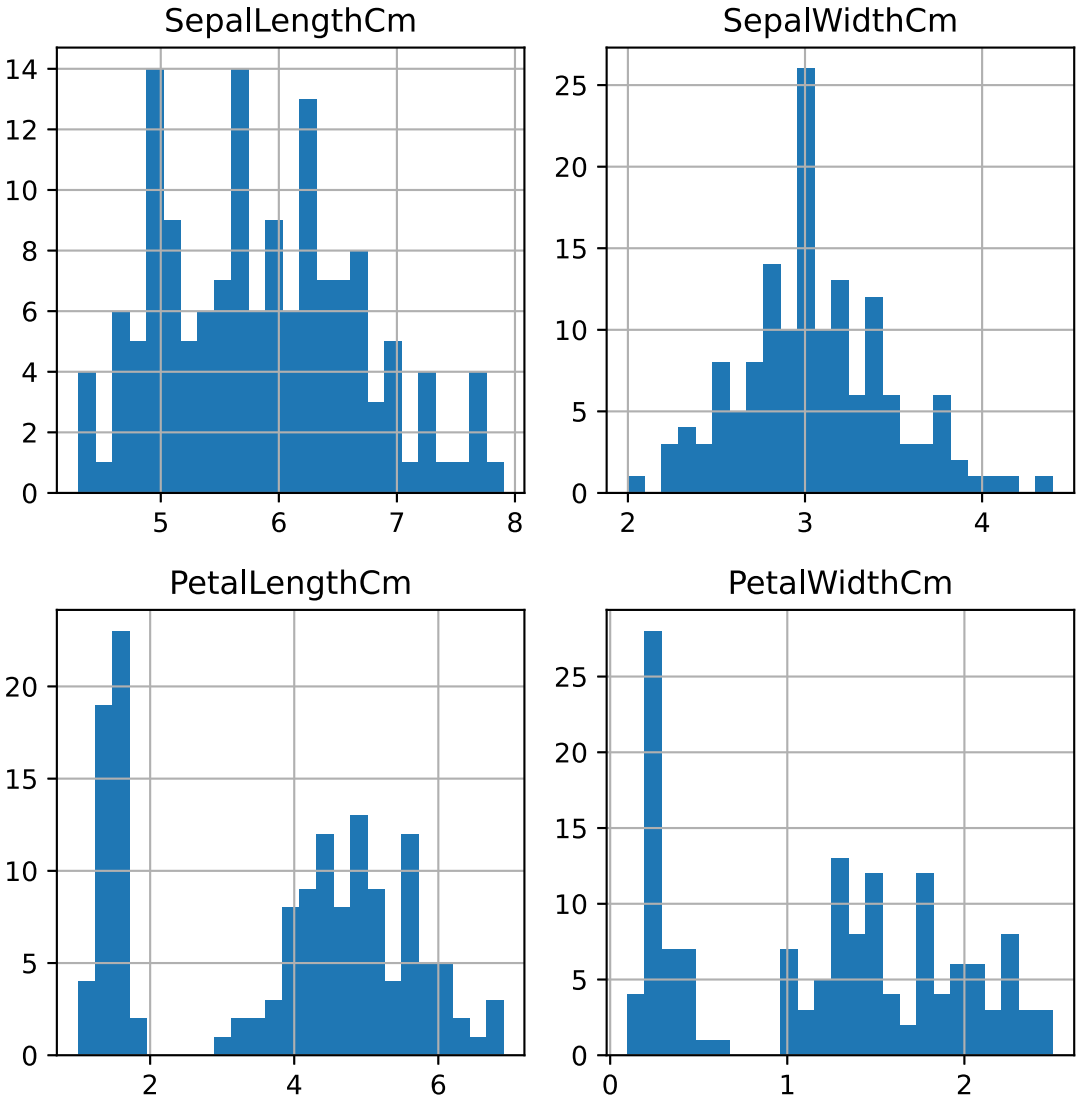



```
iris.plot()
plt.tight_layout()
```



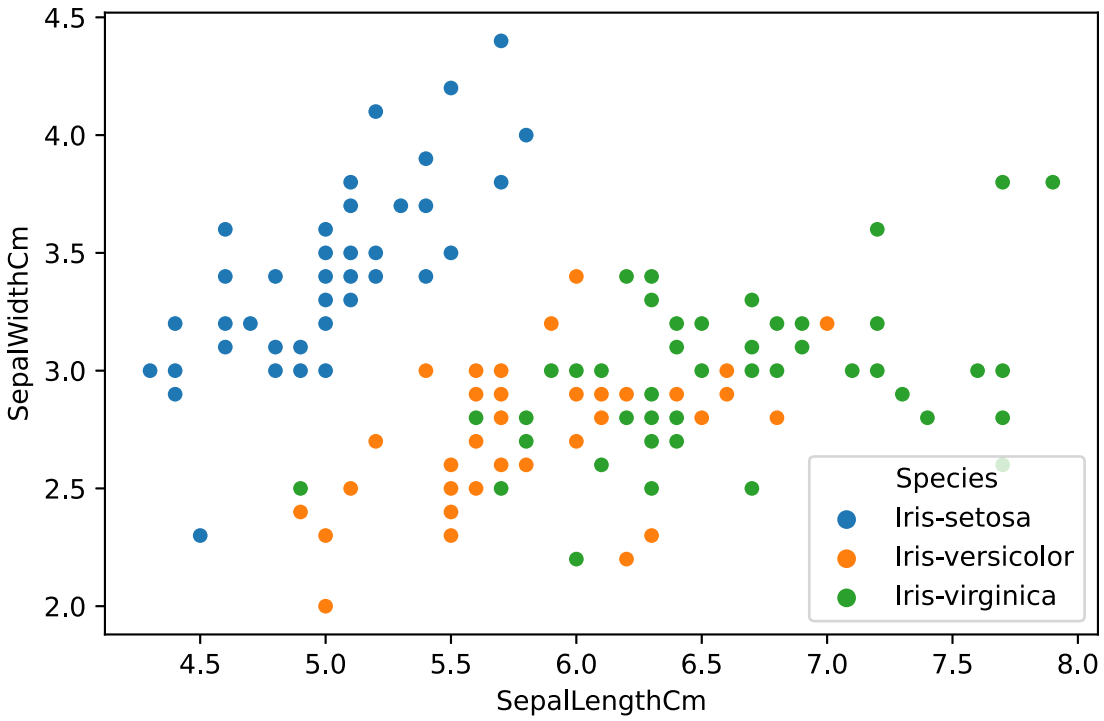
Histogram

```
iris.hist(figsize=(6,6), bins=25)
plt.tight_layout()
```



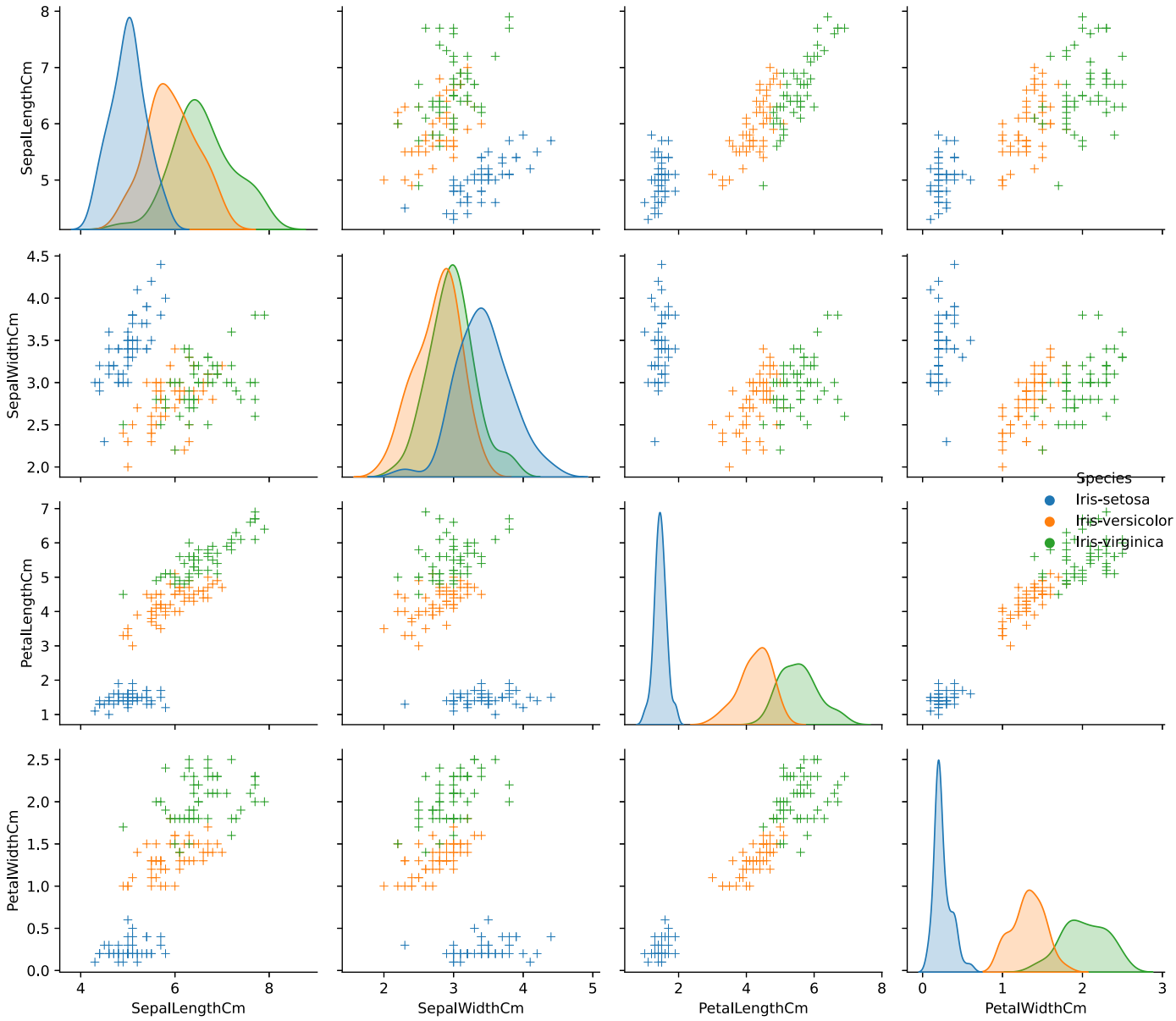
Scatter Plot

```
sns.scatterplot(x='SepalLengthCm', y='SepalWidthCm', data=iris, hue='Species')
plt.tight_layout()
```



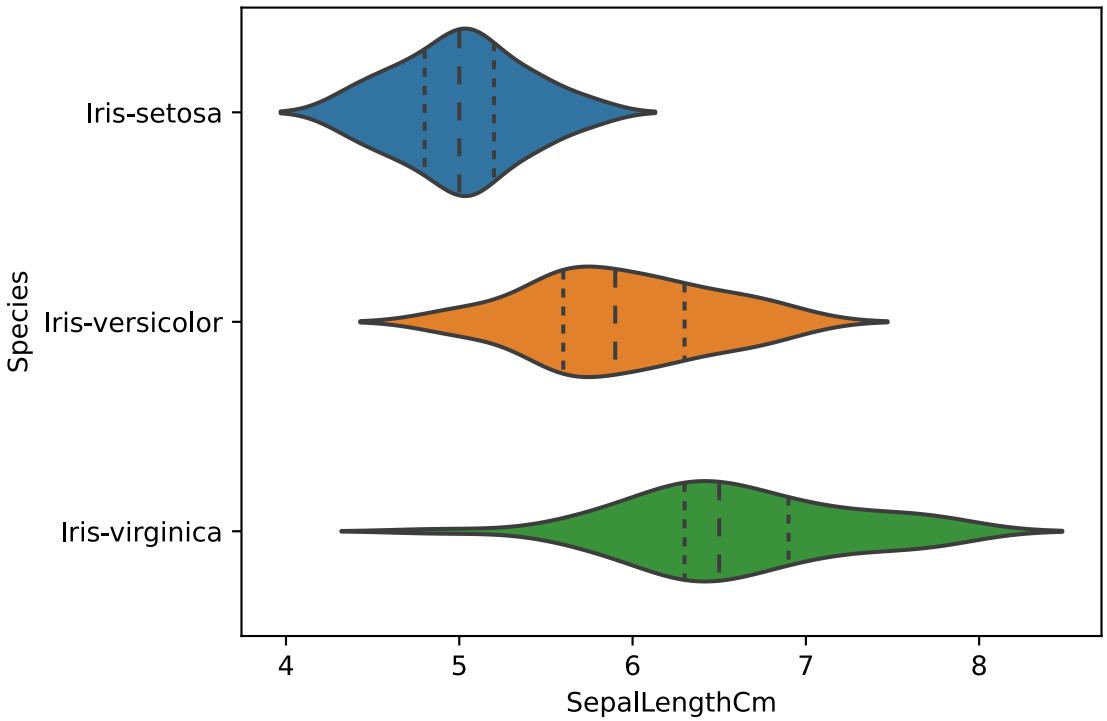
Pair Plot

```
sns.pairplot(iris, hue='Species', markers='+')
plt.tight_layout()
```



Violin Plot

```
sns.violinplot(data=iris, y='Species', x='SepalLengthCm', inner='quartile')
plt.tight_layout()
```



Dataset: Features & Class Label

```
X = iris.drop(columns='Species') # put features into variable X
X.head(5)
```

```
   SepalLengthCm    SepalWidthCm    PetalLengthCm   PetalWidthCm
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
```

```
y = iris['Species'] # put features into variable y
y.head(10)
```

```
0    Iris-setosa
1    Iris-setosa
2    Iris-setosa
3    Iris-setosa
4    Iris-setosa
5    Iris-setosa
```

```
6    Iris-setosa
7    Iris-setosa
8    Iris-setosa
9    Iris-setosa
Name: Species, dtype: object
```

Return to the Top

Split the dataset into a training set and a testing set

```
# Split the dataset into a training set and a testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

print('training dataset')
print(X_train.shape)
print(y_train.shape)
print()
print('testing dataset:')
print(X_test.shape)
print(y_test.shape)
```

```
training dataset
(117, 4)
(117,)

testing dataset:
(30, 4)
(30,)
```

Return to the Top

3. Choose a Model

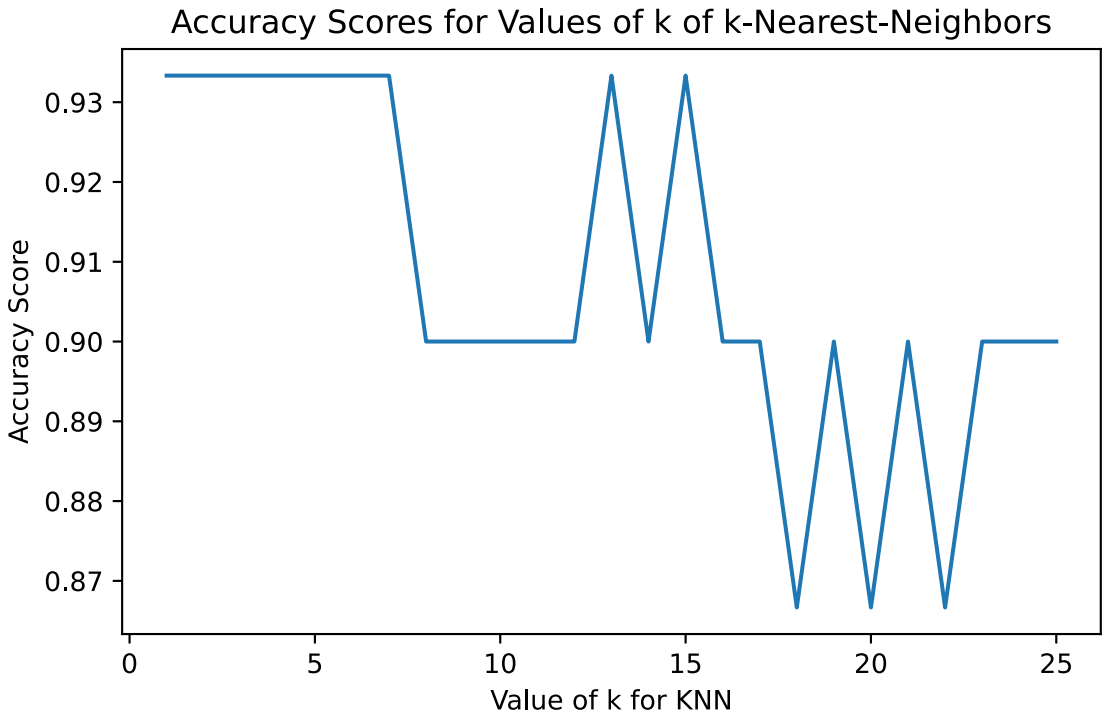
For this particular problem, we are going to use 5 algorithms of supervised learning that can solve classification problems.

K Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier
k_range = list(range(1,26))
scores = []
for k in k_range:
```

```
    model_knn = KNeighborsClassifier(n_neighbors=k) # Algorithm configuration
    model_knn.fit(X_train, y_train) # training model/classifier
    y_pred = model_knn.predict(X_test) # make predictions
    scores.append(accuracy_score(y_test, y_pred)) # performance evaluation
```

```
plt.plot(k_range, scores)
plt.xlabel('Value of k for KNN')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors')
plt.tight_layout()
plt.show()
```




```
model_knn = KNeighborsClassifier(n_neighbors=3)
model_knn.fit(X_train,y_train) # training model/classifier
y1_pred = model_knn.predict(X_test) # make predictions
```

4. Evaluating the model

Accuracy Score

```
print(accuracy_score(y_test, y1_pred))
```

0.9333333333333333

Confusion Matrix

```
print(confusion_matrix(y_test, y1_pred)) # Confusion matrix
```

```
[[11  0  0]
 [ 0  9  1]
 [ 0  1  8]]
```

Classification Report

```
print(classification_report(y_test, y1_pred))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.90      0.90      0.90        10
 Iris-virginica       0.89      0.89      0.89         9

       accuracy                           0.93        30
      macro avg       0.93      0.93      0.93        30
   weighted avg       0.93      0.93      0.93        30
```

Return to the Top

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

# model_logreg = LogisticRegression()
model_logreg = LogisticRegression(solver='lbfgs', multi_class='auto')
model_logreg.fit(X_train,y_train)
y2_pred = model_logreg.predict(X_test)
```

Accuracy Score

```
print(accuracy_score(y_test, y2_pred))
```

0.9333333333333333

```
print(confusion_matrix(y_test, y2_pred))
```

```
[[11  0  0]
 [ 0  9  1]
 [ 0  1  8]]
```

Classification Report

```
print(classification_report(y_test, y2_pred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.90      0.90      0.90        10
 Iris-virginica       0.89      0.89      0.89         9

       accuracy                           0.93        30
      macro avg       0.93      0.93      0.93        30
   weighted avg       0.93      0.93      0.93        3
```

Return to the Top

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
model_dt = DecisionTreeClassifier()
model_dt.fit(X_train,y_train)
y4_pred = model_dt.predict(X_test)
```

Accuracy Score

```
print(accuracy_score(y_test, y4_pred))
```

0.9666666666666667

Confusion Matrix

```
print(classification_report(y_test, y4_pred))
```

Classification Report

```
print(classification_report(y_test, y4_pred))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.90      0.95        10
 Iris-virginica       0.90      1.00      0.95         9

       accuracy                           0.97        30
      macro avg       0.97      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

Return to the Top

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
# model_rf = RandomForestClassifier()
model_rf = RandomForestClassifier(n_estimators=100)
model_rf.fit(X_train,y_train)
pred_rf = model_rf.predict(X_test)
```

Accuracy Score

```
print(accuracy_score(y_test, pred_rf))
```

0.9333333333333333

Confusion Matrix

```
print(confusion_matrix(y_test, pred_rf))
```

```
[[11  0  0]
 [ 0  9  1]
 [ 0  1  8]]
```

Classification Report

```
print(classification_report(y_test, pred_rf))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       0.90      0.90      0.90        10
 Iris-virginica       0.89      0.89      0.89         9

       accuracy                           0.93        30
      macro avg       0.93      0.93      0.93        30
   weighted avg       0.93      0.93      0.93        30
```

Return to the Top

Accuracy comparision for various models.

```
models = [model_knn, model_logreg, model_svc, model_dt, model_rf]
accuracy_scores = []
for model in models:
```

```
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)
# 'KNN'- 'LogReg' - 'SVC' - 'DT' - 'RF'
print(accuracy_scores)
```

[0.9333333333333333, 0.9333333333333333, 0.9333333333333333, 0.9666666666666667, 0.9333333333333333]

Return to the Top

5. Prediction

Now we can use the models to predict the prices of the houses using the predict function as we did above. Make sure when predicting the prices that we are given all the features that were present when training the model.

Chapter 10 Unsupervised learning

K-Means Algorithm Introduction

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by 'K' in K-means.

In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

Working of K-Means Algorithm

We can understand the working of K-Means clustering algorithm with the help of following steps −

Step 1 − First, we need to specify the number of clusters, K, need to be generated by this algorithm.

Step 2 − Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.

Step 3 − Now it will compute the cluster centroids.

**Step 4 **− Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more

- 4.1 − First, the sum of squared distance between data points and centroids would be computed.
- 4.2 − Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).

- 4.3 – At last compute the centroids for the clusters by taking the average of all data points of that cluster.

K-means follows **Expectation-Maximization** approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

While working with K-means algorithm we need to take care of the following things –

- hile working with clustering algorithms including K-Means, it is recommended to standardize the data because such algorithms use distance-based measurement to determine the similarity between data points.
- Due to the iterative nature of K-Means and random initialization of centroids, K-Means may stick in a local optimum and may not converge to global optimum. That is why it is recommended to use different initializations of centroids.

Implementation in Python

It is a simple example to understand how k-means works. In this example, we are going to first generate 2D dataset containing 4 different blobs and after that will apply k-means algorithm to see the result.

First, we will start by importing the necessary packages –

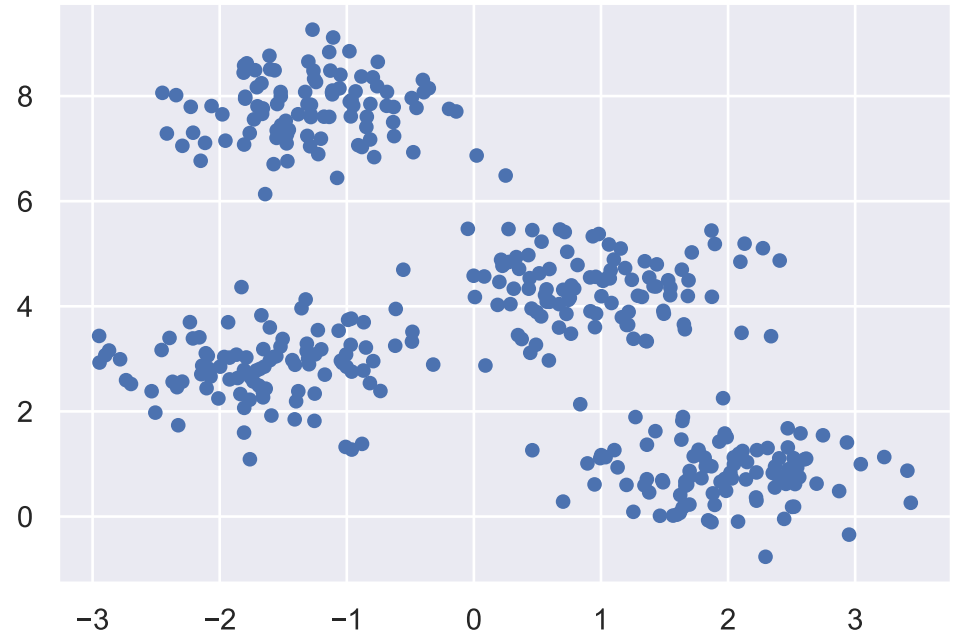
```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
import sklearn.datasets._samples_generator
```

The following code will generate the 2D, containing four blobs –

```
from sklearn.datasets._samples_generator  import make_blobs
X, y_true = make_blobs(n_samples = 400, centers = 4, cluster_std = 0.60,
random_state = 0)
```

Next, the following code will help us to visualize the dataset –

```
plt.scatter(X[:, 0], X[:, 1], s = 20);
plt.show()
```



Next, make an object of KMeans along with providing number of clusters, train the model and do the prediction as follows −

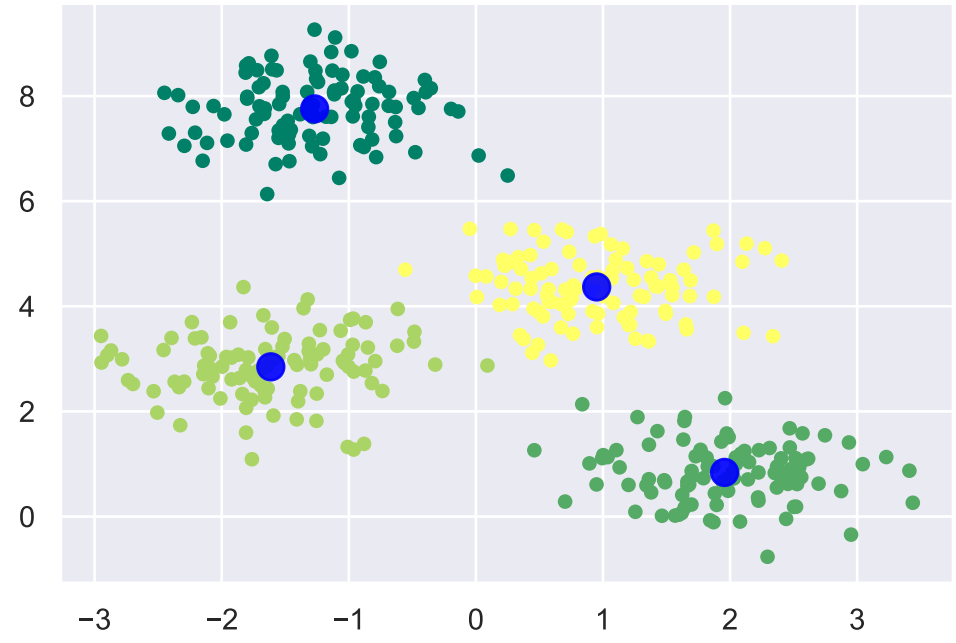
```
kmeans = KMeans(n_clusters = 4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

Now, with the help of following code we can plot and visualize the cluster's centers picked by k-means Python estimator −

```
from sklearn.datasets._samples_generator  import make_blobs
X, y_true = make_blobs(n_samples = 400, centers = 4, cluster_std = 0.60,
random_state = 0)
```

Next, the following code will help us to visualize the dataset −

```
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 20, cmap = 'summer')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'blue', s = 100, alpha = 0.9);
plt.show()
```



Advantages and Disadvantages

Advantages

The following are some advantages of K-Means clustering algorithms −

- It is very easy to understand and implement.
- If we have large number of variables then, K-means would be faster than Hierarchical clustering.
- On re-computation of centroids, an instance can change the cluster.
- Tighter clusters are formed with K-means as compared to Hierarchical clustering.

Disadvantages

The following are some disadvantages of K-Means clustering algorithms −

- It is a bit difficult to predict the number of clusters i.e. the value of k.
- Output is strongly impacted by initial inputs like number of clusters (value of k)
- Order of data will have strong impact on the final output.
- It is very sensitive to rescaling. If we will rescale our data by means of normalization or standardization, then the output will completely change.
- It is not good in doing clustering job if the clusters have a complicated geometric shape.

Applications of K-Means Clustering Algorithm

The main goals of cluster analysis are −

- To get a meaningful intuition from the data we are working with.
- Cluster-then-predict where different models will be built for different subgroups.

To fulfill the above-mentioned goals, K-means clustering is performing well enough. It can be used in following applications −

- Market segmentation
- Document Clustering
- Image segmentation
- Image compression
- Customer segmentation
- Analyzing the trend on dynamic data

Chapter 11 Computer vision

In our era, we're trying to train machines to make them do what we're doing in daily basis. Not with the perfect rate but we are trying to develop machines to help us, makes us better, and faster at some important tasks. Computer vision is one of the most important task we're working on. Seeing is a gift for human race. We can see, interpret, and analyze everything around us. When we look at a picture with dog chasing a ball eagerly, we can say just like I wrote. We can perceive the emotions in the picture, what is happening at that exact moment, or what can happen just after the shot. Computer vision task is trying to be developed for this purpose. Because sometimes, human vision may be inadequate or delayed. Sometimes instant responses may be required. Like, tumor diagnoses, intervention at the time of crime.

Computer vision is the science that enables computers to analyze and elaborate videos, and images like human brain does.

All idea behind computer vision is what computers can tell from a digital video or an image. It is a field that aims to automate tasks that the human vision can do with. Computer vision operations require some methods, like processing, analyzing, and extraction of the images. We cannot feed the model with direct images obviously. As you know, computers only understand numbers and in order to train the model, we must convert the pictures to matrices or tensors. We can also make changes in the images to make the operations easier.

What is OpenCV library?

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human.

Return to the Top

Reading and displaying image

Before whatever we want to do to our image, we first need to read our image.

```
# import library
import cv2 as cv

# read image
img = cv.imread("resourses/image.jpg")

# show image
cv.imshow("Pehli_Image", img)
cv.waitKey(0)
```

cv.waitKey(0) will display the window infinitely until any keypress. You can shut the image window by pressing one random key on your keyboard, like 'q' or escape key. If you put any other number in waitKey, it will wait that long milliseconds, and shut down itself.

cv.destroyAllWindows() simply destroys all the windows we created.

Return to the Top

Resizing image

```
# import library
import cv2 as cv

# read image
img2 = cv.imread("resourses/image.jpg")

# resize image
img2 = cv.resize(img2, (800,600))

# show image
cv.imshow("Pehli_Image", img2)
cv.waitKey(0)
cv.destroyAllWindows()
```

Return to the Top

grayscale conversion

```
# import library
import cv2 as cv

# read and resize image
img3 = cv.imread("resourses/image.jpg")
img3 = cv.resize(img3, (800,600))

# grayscale conversion
gray_img3 = cv.cvtColor(img3, cv.COLOR_BGR2GRAY)

# display image
cv.imshow("orignal_Image", img3)
cv.imshow("gray_Image", gray_img3)

# delay code
cv.waitKey(0)
cv.destroyAllWindows()
```

Return to the Top

Image into Black and white image

```
# import library
import cv2 as cv

# read and resize image
img4 = cv.imread("resourses/image.jpg")
img4 = cv.resize(img4, (800,600))

# grayscale and Black and white  conversion
gray = cv.cvtColor(img4, cv.COLOR_BGR2GRAY)
(thresh, binary)= cv.threshold(gray, 127, 255, cv. THRESH_BINARY)

# display image
cv.imshow('original', img4 )
cv.imshow('gray', gray)
cv.imshow('Black and White', binary)
cv. waitKey(0)
cv.destroyAllWindows()
```

Return to the Top

Writing image

```
# import library
import cv2 as cv

# read and resize image
img5 = cv.imread("resourses/image.jpg")
img5 = cv.resize(img5, (800,600))

# grayscale and Black and white  conversion
gray = cv.cvtColor(img5, cv.COLOR_BGR2GRAY)
(thresh, binary)= cv.threshold(gray, 127, 255, cv. THRESH_BINARY)

# saving image
cv.imwrite('resourses/Image_gray.png', gray)
cv.imwrite('resourses/Image_bw.png', binary)
```

Return to the Top

Reading video

```
# import library
import cv2 as cv

# reading video
cap2 = cv.VideoCapture('resourses/video.mp4')

#converting video to gray or Black and white
while (True):
    ret, frame = cap2.read()
    grayframe = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    (thresh, binary) = cv.threshold(grayframe, 127, 255, cv.THRESH_BINARY)
    if ret == True:
       cv.imshow("Video", grayframe)
       if cv.waitKey(1) & 0XFF == ord('q'):
            break
    else:
        break

cap.release()
cv.destroyAllWindows()
```

Return to the Top

Writing Video

```
# import library
import cv2 as cv

# reading video
cap=cv.VideoCapture('resourses/video.mp4')


# writing format, codec, video writer object and file output
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
out = cv.VideoWriter("resourses/Out_video.avi", cv.VideoWriter_fourcc('M', "j", 'P',
'G'), 10, (frame_width, frame_height))

#indicator
if (cap.isOpened () == False):
    print("error in reading video")

    #reading and playing
while(cap.isOpened ()):
    ret, frame = cap.read()
    if ret == True:
        out.write(frame)
        cv.imshow("Video", frame)
        if cv.waitKey(25) & 0XFF == ord('q'):
            break
    else:
        break

cap.release()
cv.destroyAllWindows ()
```

Return to the Top

Capture a webcam

```
# how to capture a webcam inside python

# Step-1 Import libraries
import cv2 as cv
import numpy as np

# Step-2 Read the frames from Camera
```

```
cap = cv.VideoCapture(0) #webcam no.1

# read until the end
# Step-3 Display frame by frame
while(cap.isOpened()):
     #capture frame by frame
     ret, frame = cap.read ()
     if ret == True:
           # to display frame
           cv. imshow("Frame", frame)
           # to quit with q key
           if cv.waitKey(1) & 0XFF == ord('q'):
                break
     else:
           break

# Step-4 release or close windows easily
cap.release()
cv.destroyAllWindows ()
```

Return to the Top

change color of webcam

```
# Import libraries
import cv2 as cv
import numpy as np

# Read the frames from Camera
cap = cv.VideoCapture(0)

# converting frames from Camera into gray or Black and white
while(True):
    (ret, frame) = cap.read()
    gray_frame = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    (thresh, binary)= cv.threshold(gray_frame, 127, 255, cv. THRESH_BINARY)

    cv.imshow("OriginalCam", frame)
    cv.imshow("GrayCam", gray_frame)
    cv.imshow("BinaryCam", binary)
    if cv.waitKey(1) & 0XFF == ord('q'):
        break


cap.release()
cv.destroyAllWindows ()
```

Return to the Top

Writing videos from cam

```
# Writing videos from cam

# Import libraries
import cv2 as cv
import numpy as np

# Read the frames from Camera
cap=cv.VideoCapture(0)

# writing format, codec, video writer object and file output
# Obtain frame size information using get() method
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
frame_size = (frame_width,frame_height)
fps = 15

out = cv.VideoWriter("resourses/cam_video.avi", cv.VideoWriter_fourcc('M', "j", 'P',
'G'), 15, frame_size)

# fourcc: 4-character code of codec, used to compress the frames (fourcc)
# fps: Frame rate of the created video stream


if (cap.isOpened () == False):
    print("error in reading video")

    #reading and playing
while(cap.isOpened ()):
    ret, frame = cap.read()
    if ret == True:
        out.write(frame)
        cv.imshow("Video", frame)
        if cv.waitKey(25) & 0XFF == ord('q'):
            break
    else:
        break


cap.release()
out.release()
cv.destroyAllWindows ()
```

Return to the Top

Setting of camera or video

```
# Setting of camera or video

# Import libraries
import cv2 as cv
import numpy as np

# Read the frames from Camera
cap = cv.VideoCapture(0)


cap.set(10, 100) # 10 is the key to set brightness
cap.set(3, 640) # width key 3
cap.set(4, 480) #height key 4


while (True):
     ret, frame = cap.read()
     if ret == True:
           cv.imshow("frame", frame)
           if cv.waitKey(1) & 0XFF == ord('q'):
                break
     else:
           break


cap.release()
cv.destroyAllWindows ()
```

Return to the Top

Basic functions in opencv

```
# basic functions or manipulation in opencv
# Import libraries
import cv2 as cv

# read image
img1 = cv.imread("resourses/image.jpg")

#resize
img = cv.resize(img1, (800,600))
```

```
# gray
gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# binary
(thresh, binary)= cv.threshold(gray_img, 127, 255, cv.THRESH_BINARY)

# blurred image
blurr_img = cv.GaussianBlur(img, (7,7), 0)

# edge detection
edge_img = cv.Canny(img, 53,53)

# thickness of lines
mat_kernel = np.ones((3,3), np.uint8)
dilated_img= cv.dilate(edge_img, (mat_kernel), iterations=1)

# Make thinner outline
ero_img = cv.erode(dilated_img, mat_kernel, iterations=1)

#cropping we will use numpy library not open cv
#print("The size of our image is: ", img.shape)
cropped_img = img[0:500, 150:400]


cv.imshow("Original", img)
cv.imshow("Resized", img)
cv.imshow("Gray", gray_img)
cv.imshow('Black and White', binary)
cv.imshow("Blurr", blurr_img)
cv.imshow("edge", edge_img)
cv.imshow("Dilated", dilated_img)
cv.imshow("Erosion", ero_img)
cv.imshow("Cropped", cropped_img)


cv.waitKey(0)
cv.destroyAllWindows ()
```

Return to the Top

Draw lines, and shapes

```
# How to draw lines, and shapes in python
# Import libraries
import cv2 as cv
import numpy as np
```

```
# Draw a canvas e is for Black
img = np.zeros((600,600)) # black
img1 = np.ones((600,600))

# print size
print("The size of our canvas is: ", img.shape)
# print(img)

#adding colors to the canvas
colored_img = np.zeros( (600,600, 3), np.uint8)   #color channel addition

colored_img[:] = 255,0, 255 #color complete image

colored_img[150:230, 180:500] = 255,168, 10 # part of image to be colored

#adding line
cv.line(colored_img, (0,0), (colored_img.shape[0], colored_img.shape[1]), (255,0,0)
,3) #croosed line
cv.line(colored_img, (100,100), (300, 300), (255,255,50) ,3) # part line

#adding rectangles
#cv.rectangle(colored_img, (200, 200), (300, 600), (255,255,255), 3) # draw
cv.rectangle(colored_img, (300, 500), (300, 400), (255,255,255), cv.FILLED) # draw

# adding circle
cv.circle(colored_img, (400,300), 50, (255,100,0), 5) # draw
cv.circle(colored_img, (400,300), 50, (255,100,0), cv.FILLED) # draw

#adding text
y0, dy, text = 500,50, "Python ka Chilla \non Codanics Youtube Channel"
for i, line in enumerate(text.split('\n')):
    y = y0 + i*dy
    cv.putText(colored_img, line, (50, y ), cv.FONT_HERSHEY_SIMPLEX, 1, (255,255,
0), 2, cv.LINE_AA, False)


# cv.imshow("Black", img)
# cv.imshow("White", img1)
cv.imshow("Colored", colored_img)

cv.waitKey(0)
cv.destroyAllWindows ()
```

Return to the Top

Resolution of cam

```
# Import libraries
import cv2 as cv
import numpy as np

# Read the frames from Camera
cap = cv.VideoCapture(0)


#resolutions
def hd_resolution():
    cap.set (3, 1280)
    cap.set(4, 720)
```

```
def sd_resolution():
    cap.set(3, 640)
    cap.set(4, 480)
def fhd_resolution():
    cap.set(3, 1920)
    cap.set(4, 1080)

cap.set(cv.CAP_PROP_FPS, 30)

#sd_resolution()
hd_resolution()
#fhd_resolution()

while(True):
    ret, frame = cap.read()
    if ret ==True:
        cv. imshow("Camera", frame)
        if cv.waitKey(1) & 0XFF == ord('q'):
             break

    else:
       break
cap.release()
cv.destroyAllWindows ()
```

Return to the Top

Saving HD recording of Cam steaming

```
import cv2 as cv
import numpy as np
cap = cv.VideoCapture(0)

cap = cv.VideoCapture(0)

#resolutions
def hd_resolution():
    cap.set (3, 1280)
    cap.set(4, 720)
def sd_resolution():
    cap.set(3, 640)
    cap.set(4, 480)
def fhd_resolution():
    cap.set(3, 1920)
    cap.set(4, 1080)


#sd_resolution()
```

```
hd_resolution()
#fhd_resolution()

# Obtain frame size information using get() method
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
frame_size = (frame_width,frame_height)
fps = 30

out = cv.VideoWriter("resourses/cam_video.avi", cv.VideoWriter_fourcc('M', "j", 'P',
'G'), 30, frame_size)

# fourcc: 4-character code of codec, used to compress the frames (fourcc)
# fps: Frame rate of the created video stream


if (cap.isOpened () == False):
    print("error in reading video")

    #reading and playing
while(cap.isOpened ()):
    ret, frame = cap.read()
    if ret == True:
        out.write(frame)
        cv.imshow("Video", frame)
        if cv.waitKey(25) & 0XFF == ord('q'):
            break
    else:
        break
cap.release()
out.release()
cv.destroyAllWindows ()
```

Return to the Top

Joining two images

```
# Joining two images

import cv2 as cv
import numpy as np

img = cv.imread("resourses/image.jpg")
img = cv.resize(img, (800,600))

#stacking same image

#1- Horizontal stack
```

```
hor_img = np.hstack((img, img))

# 2- Vertical stack
ver_img = np.vstack((img, img))

cv.imshow("Hoizontal", hor_img)

cv.imshow("vertical", ver_img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Return to the Top

Combine 2 images with two different sizes

```
import numpy as np
import cv2 as cv

img1 = cv.imread("resourses/image.jpg")
img1 = cv.resize(img1, (800,600))
img2 = cv.imread("resourses/01.jpg")


#combine 2 images  with two different sizes
h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
```

```
#create empty martrix (Mat)
res = np.zeros(shape=(max(h1, h2), w1 + w2, 3), dtype=np.uint8)

# assign BGR values to concatenate images
for i in range(res.shape[2]):
    # assign img1 colors
    res[:h1, :w1, i] = np.ones([img1.shape[0], img1.shape[1]]) * img1[:, :, i]
    # assign img2 colors
    res[:h2, w1:w1 + w2, i] = np.ones([img2.shape[0], img2.shape[1]]) * img2[:, :,
i]

output_img = res.astype('uint8')
cv.imshow("output", output_img )
cv.waitKey(0)
cv.destroyAllWindows()
```

Return to the Top

Resize + join image + change color

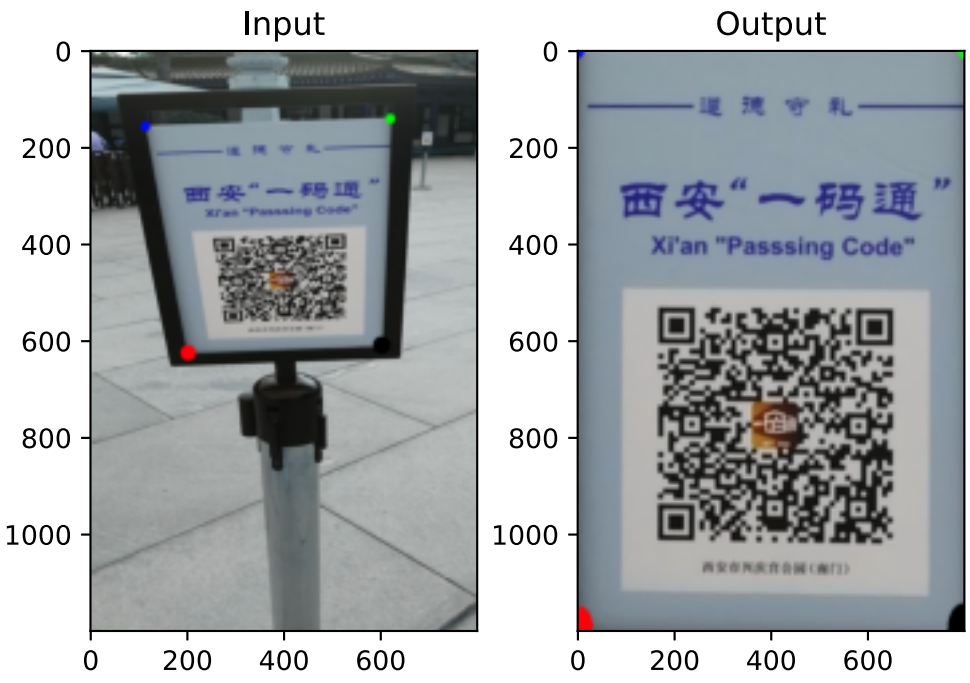
```
import numpy as np, cv2

img1 = cv2.imread("resourses/image.jpg", 0)
img1 = cv.resize(img1, (800,600))
img2 = cv2.imread("resourses/01.jpg", 0)
img2 = cv2.resize(img2, (800,600))
h1, w1 = img1.shape[:2]
h2, w2 = img2.shape[:2]
vis = np.zeros((max(h1, h2), w1+w2), np.uint8)
vis[:h1, :w1] = img1
vis[:h2, w1:w1+w2] = img2
vis = cv2.cvtColor(vis, cv2.COLOR_GRAY2BGR)

cv2.imshow("test", vis)
cv2.waitKey()
```

Return to the Top

Change the perspective of an image

```
# how to change the perspective of an image

import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
width, height = 800, 1200
img = cv.imread("resourses/pass.jpg")
red_point = [114 , 159]
green_point = [619 , 143]
blue_point =[202 , 626]
black_point = [601 , 610]
# Create point matrix
```

```
point_matrix = np.float32([red_point, green_point,blue_point, black_point])

# Draw circle for esch point
cv.circle(img,(red_point[0], red_point[1]) ,10,(0,8, 255),cv.FILLED)
cv.circle(img, (green_point[0],green_point[1]),10, (0 ,255,0), cv.FILLED)
cv.circle(img, (blue_point[0],blue_point[1]),16,(255,0,8) ,cv.FILLED)
cv.circle(img,(black_point[0] ,black_point[1]), 18, (0,0.0), cv.FILLED)

converted_points = np.float32([[0,0], [800,0], [0, height], [width, height]])
perspective_transform = cv.getPerspectiveTransform(point_matrix ,converted_points)
img_Output = cv.warpPerspective(img,perspective_transform ,(width, height))

cv.imshow("Original Image", img)
cv.imshow("Output Image", img_Output)
plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(out_img),plt.title('Output')
plt.show()
cv.waitKey(0)
cv.destroyAllWindows ()
```



Return to the Top

Coordinates of an image or video

```
# coordinates of an image or video
# BGR color codes from an image

# Step-1: Import libraries
```

```
import cv2 as cv
import numpy as np

# Step-2: define a function
def find_coord(event, x, y, flags, params):
    if event == cv.EVENT_LBUTTONDOWN:
        #left mouse click
        print(x, '', y)
        #how to define or print on the same image or window
        font = cv. FONT_HERSHEY_PLAIN
        cv.putText(img, str(x) + ','+ str(y), (x, y), font, 1, (255,0,0),
thickness=2)
        #show the text on image and img itself
        cv.imshow("image", img)
    # for color finding
    if event == cv.EVENT_RBUTTONDOWN:
     print(x, '', y)
     font = cv.FONT_HERSHEY_SIMPLEX
     b = img[y,x,0]
     g = img[y,x,1]
     r = img[y,x,2]
     cv. putText(img, str(b) + ',' + str(g) + ',' + str(r), (x,y), font, 1,
(255,0,0), 2)
     cv.imshow('image', img)


# Step-3: final function to read an display

if __name__ == "__main__":
    #reading an image
    img = cv.imread("resourses/pass.jpg", 1)
    #display the image
    cv.imshow('image', img)
    # setting call back function
    cv.setMouseCallback("image", find_coord)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

Return to the Top

Split video into frames

```
# split your video into frames or image sequence
import cv2 as cv
cap = cv.VideoCapture('resourses/video.mp4')
frameNr = 0
while (True):
    success, frame = cap.read()
```

```
    if success:
          cv.imwrite(f"resourses/frames/frame_{frameNr}.jpg", frame)
    else:
          break
    frameNr = frameNr+1
cap.release()
```

Return to the Top

Color detection using opencv

```
import cv2 as cv
import numpy as np

img = cv.imread("resourses/01.jpg")
img = cv.resize(img, (600,500))
def empty(a):
    pass

cv.namedWindow("HSV")
cv.resizeWindow("HSV",640,240)
cv.createTrackbar("HUE Min","HSV",0,179,empty)
cv.createTrackbar("HUE Max","HSV",179,179,empty)
cv.createTrackbar("SAT Min","HSV",0,255,empty)
cv.createTrackbar("SAT Max","HSV",255,255,empty)
cv.createTrackbar("VALUE Min","HSV",0,255,empty)
cv.createTrackbar("VALUE Max","HSV",255,255,empty)

while True:


    imgHsv = cv.cvtColor(img,cv.COLOR_BGR2HSV)

    h_min = cv.getTrackbarPos("HUE Min","HSV")
    h_max = cv.getTrackbarPos("HUE Max", "HSV")
    s_min = cv.getTrackbarPos("SAT Min", "HSV")
    s_max = cv.getTrackbarPos("SAT Max", "HSV")
    v_min = cv.getTrackbarPos("VALUE Min", "HSV")
    v_max = cv.getTrackbarPos("VALUE Max", "HSV")
    print(h_min)

    lower = np.array([h_min,s_min,v_min])
    upper = np.array([h_max,s_max,v_max])
    mask = cv.inRange(imgHsv,lower,upper)
    result = cv.bitwise_and(img,img, mask = mask)

    mask = cv.cvtColor(mask, cv.COLOR_GRAY2BGR)
    hStack = np.hstack([img,mask,result])
```

```
    #cv.imshow('Original', img)
    #cv.imshow('HSV Color Space', imgHsv)
    #cv.imshow('Mask', mask)
    #cv.imshow('Result', result)
    cv.imshow('Horizontal Stacking', hStack)
    if cv.waitKey(1) & 0xFF == ord('q'):
        break

cv.destroyAllWindows()
```

Return to the Top

face detection in images

```
# face detection in images

import cv2 as cv

face_cascade = cv.CascadeClassifier('resourses/haarcascade_frontalface_default.xml')

img = cv.imread("resourses/01.jpg")

img= cv.resize(img, (800, 900))


gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray_img, 1.1, 4)

#draw a rectangle

for (x, y, w, h) in faces:

    cv.rectangle(img, (x,y), (x+w, y+h), (255,0,0), 2)

cv.imshow("image", img)
cv.imwrite("resourses/arsl.png", img)
cv.waitKey(0)
cv.destroyAllWindows ()
```

Return to the Top

Bonus chapter

LinkedIn

To optimize your LinkedIn profile as a Data Scientist, you need to build a strong LinkedIn presence.

LinkedIn Profile Photo: Keep a high-definition, well-lit professional headshot as your LinkedIn profile picture. People use informal selfies and group pictures which does not have a good impact on those viewing your profile.

Headline: Here's where you can get more creative. LinkedIn allows you to pitch yourself briefly and effectively. You can use minimum words to make maximum impact on recruiters.

Mention specific skills that grab attention. For example, if you're proficient in NLP and CNN, your headline should clearly reflect that.

Write a simple, results-driven headline which will make your profile visitors spend some time on your profile and connect with you.

Professional Summary: If you have relevant experience as a Data Scientist, do write a clear and informative career journey. Mention your Designation, Organization and Dates of employment along with a quick summary of your job role.

Write in pointers. Include results, numbers and metrics wherever possible. When writing Data Science projects that you have worked on start with the problem statement and then mention the result your project had. If including algorithms, ensure to mention the ones you have a certain level of proficiency in.

Collect and write recommendations: Make sure you ask for recommendation notes from your previous peers, colleagues, superiors and collaborators. A good way to ensure that is to recommend people yourself.

Focus on your skill-set.

Do not try to get too many recommendations by endorsing and recommending your connections randomly or sending random requests. If you're an expert on Naive Bayes, you can focus on collecting recommendations from people who have worked with you on a related project or recognized your skills for that particular algorithm.

If you have a quality recommendation from 2–3 people along with instances just Naive Bayes, it will have a stronger impact than having your peers validating you randomly for multiple ML algorithms.

Endorse for skills: Get your technical skills highlighted by collecting endorsements from colleagues and managers.

Focus on getting endorsed for popular tools and applications that are relevant in the industry. For instance, getting endorsed for Python proficiency adds much more value to your profile than getting endorsed for a programming tool like Java.

Since Python is a more efficient and time-saving tool for Machine Learning than Java, it is automatically a more relevant skill than Java. Weed out such unimportant additions from your profile.

LinkedIn Skill Assessments: LinkedIn is one of the leading platforms for job searching in the world and thousands of companies and job recruiters list over 1000+ jobs daily. While its size makes it crucial to create and maintain a LinkedIn profile, it can also make it harder for you to stand out, as there are just too many applicants.

Which is where LinkedIn Skill Assessments come into play. Taking one of these tests (and passing) can help confirm you're the real deal and help you stand out from the crowd as a result. Recruiters will use any tool at their disposal to zero in on the best candidates.

Taking a test or getting a LinkedIn skill badge is not a guarantee that you will land your dream job. However, you have nothing to lose and everything to gain by taking a LinkedIn Skill Assessment.

Accomplishments & Interests: Make this area reflect your interest in Data Science. Write down your accomplishments, certifications and courses.

Again, if you are mentioning Data Science tools and applications here, also mention your proficiency level in each. For example, you might write ''Highly proficient in…" and follow that with a list of ML algorithms. A more reliable way of doing that would be to write something like this:

-Linear Regression: Proficiency- High. Successfully built # number of models using Linear Regression…

-Logistic Regression: Proficiency- Intermediate. Worked on # number of algorithms with medium exposure to Logistic Regression…

Update your credentials: If you have taken a data science course online/offline, include them under Education. List all the topics you have learned in the course duration. If you have worked on a data project at work, include them under responsibilities.

Post project links in your projects section: It is likely that you would have done a capstone project as a part of your course. You would have presented it to an audience. Don't just save it to your computer. Share it online. I suggest uploading them to Github and Slideshare and copy-paste the links to your individual projects. Include a well written project summary that details the work you have done. You must specify if the project was done as a team or solo. If it is a team - include the names of your project partners and detail the tasks executed by each of you. There should be mention of the tools and techniques you have used in solving the problem.

If you have not worked on a data science project, then i ask of you to just spend a week working on one data science problem. Create presentations that detail how you approached the problem. Read about CRISP-DM. This is all you need to explain your approach one step at a time. Build your project portfolio because end of the day your activities count as experience. Employers want a tangible proof of your experience. Personal projects are the best to way to do it.

Share data science related articles on Linkedin: If you have read an interesting article or tutorial related to Data science. Post them on your profile. This helps your peers recognize the value in the domain and also acknowledge your dedication to learning.

Follow Data science influencers and Companies: The more people you follow, the more you learn about the trends in the field. This will complement the above tip.

Create an "Independent activities" section under Education: Taking a course is one thing, but applying the knowledge you get from it is a whole new ball game. I believe in this phrase - "Knowledge is wasted when not expressed". This section should list all your independent activities - whether it is additional courses, attending workshops and webinars, or participating in hackathons, you must list them.

Upload professional certificates: Don’t post all of them separately, because this will make your profile look like a movie credits section. Not all are Marvel fans. Nobody has the patience to go through them all. I prefer to keep the profile minimalistic. There are websites that let you collate all your certificates and showcase it on a single dashboard. Utilize it and you will only have to post the link to the site. All your certificates are a click away.

Profile URL: Create a profile URL. Allow others to quickly identify you in search results by changing or customising your public profile URL. Just go to your ‘Profile’ then click on the profile URL which appears on the bottom-left corner on the window. It should be something like – https://www.linkedin.com/in/xyz-abc-245b5b42 by default. Just click on it, add your name which is simple to read.

Twitter

LinkedIn might be the first social media platform you consider when looking for work. But Twitter offers plenty of opportunities for job seekers and resercher too but in a more informal setting. It's common to see large companies announce vacancies on Twitter, and you're able to follow recruiters and hiring managers without needing to make a formal connection as you would on LinkedIn.

Twitter can be useful for:

- Filtering, accessing science stories relevant to your field of study (e.g., EurekAlert!, news media, science writers)
- Assisting with your career (job ads, getting to know potential colleagues/supervisors)
- Creating a research network
- Doing research
- Forging collaborations
- Attending conferences virtually
- Engaging with a broader audience (e.g. Directly or through journalists, media offices, science writers)
- Social justice, political change, activism
- Being inspired by great thinkers, innovators, writers, scientists etc
- Seeng the human side of science
- Becoming a better writer and science communicator
- Track the reach of your work (analytics)

Twitter for a Successful Job Search

Optimizing Your Twitter Bio to Find Work
Step back from the idea you always need to apply for a job actively and consider that recruiters also search Twitter to headhunt their candidates. So, if you want to show up in their results, ensure your Twitter bio is thoroughly optimized.

You only have 160 characters, so make them count. You might choose to mention your qualifications, the industry you work in, or your past work experience. Remember to use keywords that employers might search for, like "finance graduates."

It's also a good idea to be easily contactable by opening your DMs. You can switch these on through Messages > Settings > Direct Messages.

Use Twitter Lists to Find Valuable Content
If you already follow hundreds of accounts on Twitter, how can you sift through all the content to find relevant job vacancies? The answer is Twitter Lists. This is a category of accounts you group together to make it easier to find the information you're looking for.

Instead of relying on the general Twitter feed to present the tweets you want to read, you can set up lists to help manage your job search. For example, you might want to create a list of companies that offer remote work or those that are hiring graduates with your degree.

Start by going to Lists and Start a New List.

You can give your list a name, a description, and choose to add an image too. These are all essential fields if you want to make your list Public, but for a job-seeking list, you'll probably want to set your list to Private so no one can see it.

Organize Your Job Search With TweetDeck
TweetDeck is a dashboard feature that provides you with an at-a-glance overview of your Twitter feed. You can view lists, trending topics, messages, and account information in a single hit to supercharge your experience of Twitter.

It's easier to work with TweetDeck on a desktop than on your phone. But from here, you can keep an eye on industry news, job openings, and important updates.

Once you're logged in to TweetDeck using your Twitter sign-in, you can customize the columns to provide you with the most relevant information.

Building a Relevant Twitter Network
If you're new to Twitter, don't expect to attract thousands of followers overnight — it does take time. To build a solid professional profile, start by following the accounts of:

- Companies you might be interested in working with
- Recruitment firms
- Career fairs
- Past colleagues
- College professors
- Thought leaders within your industry.

Remember, you can create Twitter Lists based on your connections. It's important to go beyond just following these accounts. Engage with them and respond to their tweets regularly to build an ongoing relationship.

Participating in Twitter Chats
Twitter Chats are recurring conversations hosted by the same account. They might happen weekly or monthly, but they'll usually be at the same time. Each Twitter chat has a hashtag to follow, which you should include in every message you write during the chat.

Start looking for Twitter Chats to join — for example, #JobHuntChat takes place on Mondays at 9 pm EST. This offers tips on networking, interviewing, and creating a winning resume.

Maximize Your Use of Twitter Search

With so much information available on Twitter, you must know how to use the search tool correctly, starting with the terms you use.

Use a combination of keywords in this sequence: location + job level + job + industry.

Your result can look like this: New York + graduate + job + accountancy.

You can switch out the word "job" with "hiring." "vacancy," or "position" as needed.

After choosing your search terms, use the filter to narrow down your field further. For example, you can filter by From Anyone or People You Follow and by Location.

By clicking Advanced Search, you can include hashtags, accounts, and engagement fields.

Narrowing Down Job Opportunities

We've already discussed Twitter Chats, Twitter Lists, and how to build your Twitter network, but there are yet more ways to seek out job opportunities on Twitter. The first is to use hashtags, so you can follow tags like #hiring, #gradjobs, #salesjobs, or whatever terms are relevant to you.

It's also worth creating a list of dream companies you would like to work for. If it's a giant corporation, they will often promote a separate Twitter account just for recruitment. For example, KPMG Careers Twitter profiles are different from the main KPMG account.

Present a Professional Image

If you're using Twitter for professional purposes, think about the image you're portraying. Your account doesn't need to be as formal as LinkedIn, but it needs to present you positively. So use a clear and crisp headshot as your Twitter profile picture, and remember to optimize your bio.

You can also link to your professional website or LinkedIn profile so recruitment teams can conduct further background research on you.

If you already have a Twitter account for non-work purposes, review the tweets you send out to check if they would be suitable for hirers to read. This isn't LinkedIn, so you don't need to be 100% professional. But you might be more comfortable setting up a new Twitter account, so there aren't any blurred lines between your work and personal life.

Using Twitter for Interview Prep

Congratulations, you've secured an interview! Now, it's time to do your prep work, so you can go into that interview and win over the recruiters. Twitter is a great way to learn every last detail about the company you'd like to work for. In fact, you can use Twitter without an account to snoop around.

Be careful not to start actively following any of the interviewers at this stage. Still, you can certainly do a lot of digging to learn about their past work experience, projects, and outside interests.

When using Twitter as a scientist, here are some things to think about:

- What you might want to do on Twitter? (Learn? Engage? Have fun? Grow a following? Do research? Promote your work?). Craft your profile and approach based on these objectives (Note: this can change!)
- If your objectives are about science, find a balance between professional/personal (actually: ALWAYS think about this… And remember that "personal" is seldom completely private with social media tools)
- Don't overwhelm your followers with self-promotion
- When tweeting try to "Be professional, and Be positive" (note: I learned this advice from Adam Taylor who runs #SciStuChat)
- You don't have to Tweet to be on Twitter: Watch and learn before jumping in (many months, perhaps!)
- Curate who you follow carefully (Don't be afraid to unfollow people)
- Don't obsess about growing your own following: this will happen over time
- Don't feel you have to read your entire feed: important and interesting content appears multiple times
- If you per objective is to share content, aim for information-rich tweets (links/photos etc)
- Use "draft" features – sometimes it's good to write Tweets without sending them right away.
- Learn how to use Hashtags effectively (they are, essentially the "magnets" of the Internet)
- Own up to mistakes / apologize
- Give credit where it's due, especially when thinking about sharing photos or art: ask permission before sharing!
- Curate content! (e.g. "Like" button, or better yet, another program – Pocket, Evernote) – it's easy to forget about neat things you have seen on Twitter, so it's important to find ways to save the things you may wish to find later on.

Caveats:

- Twitter can become a time-waster and great procrastination tool: learn to be careful with your use
- Often, your community ends up being limited to like-minded people
- It's easy to get embroiled in debates and controversy: be careful
- Trolls can ruin everything; people can be jerks.
- Twitter is certainly not for everyone