Python

Python Environment

Types Of Variables

- Integer
- Float / double
- String
- Logical / Boolean

Operators

- **Comparison opr.**

  <   >   <=   >=   != <> ==

- **Logical Operator**

  and or not

- **Arithmetic opr.**

  + - / ( ) %

While Loop

No { } brackets

Indentation is important

```
while condition:
        executable code1
        executable code2
        executable code3
executable code4
```

```
while condition:
        executable code1
        executable code2
executable code3
executable code4
```

For Loop

```
for i in range(5):
    print(‘Hello ’)
```

```
for j in range(1,10):
    print(‘Hello :’, j)
```

range(begin,end,step)

```
for k in range(10,100,5):
            print( k )
```

If stmt

```
if condition1:
        executable code
elif condition2:
        executable code
else:
        executable code
```

Assignment

Law Of Large Numbers

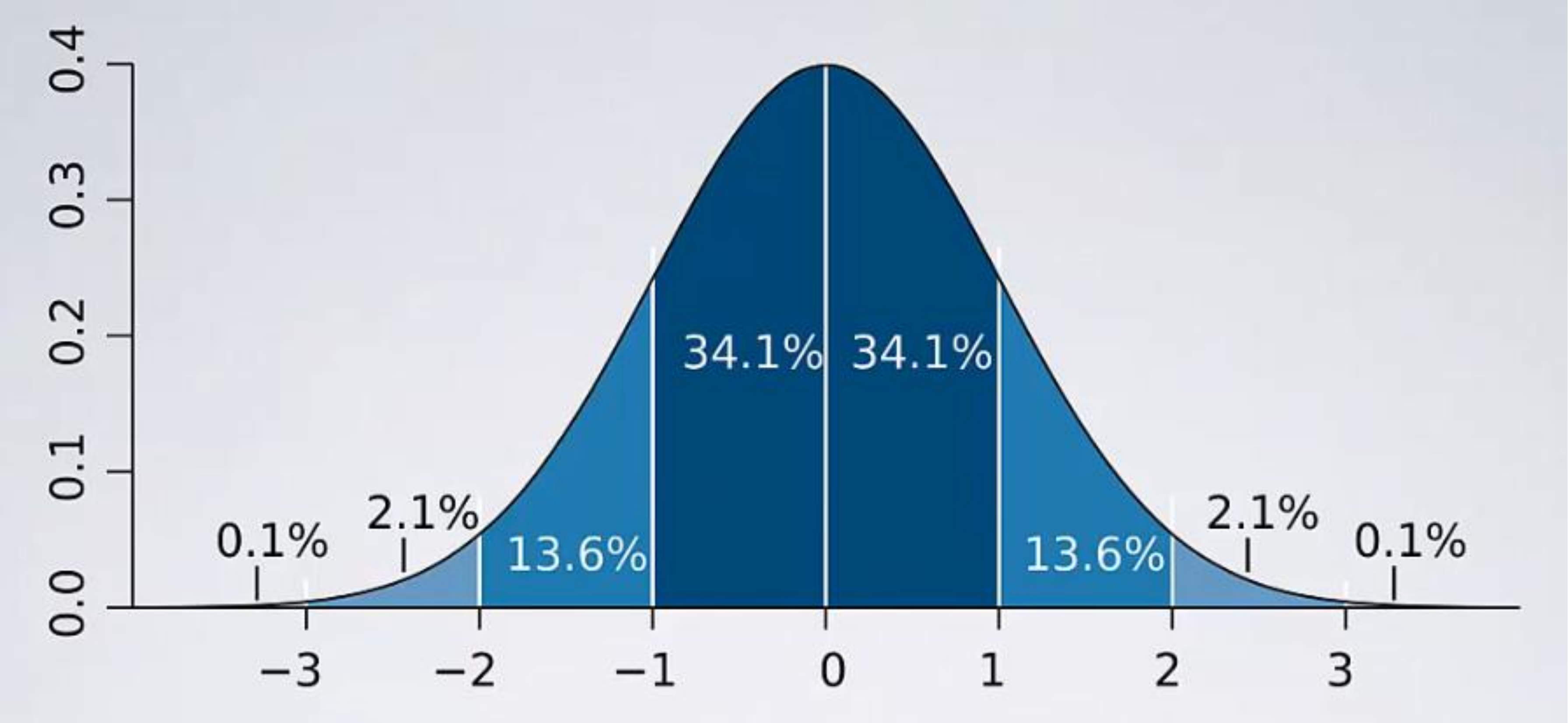
Test The Law OF Large Numbers for N random normally distributed numbers with mean = 0, stdev = 1

Create a Python Script that will count how many of these numbers fall between -1 and 1 and divide by the total quantity of N

E(X) = 68.2%

Check that Mean(Xn) -> E(X) as you rerun your script while increasing N

Assignment

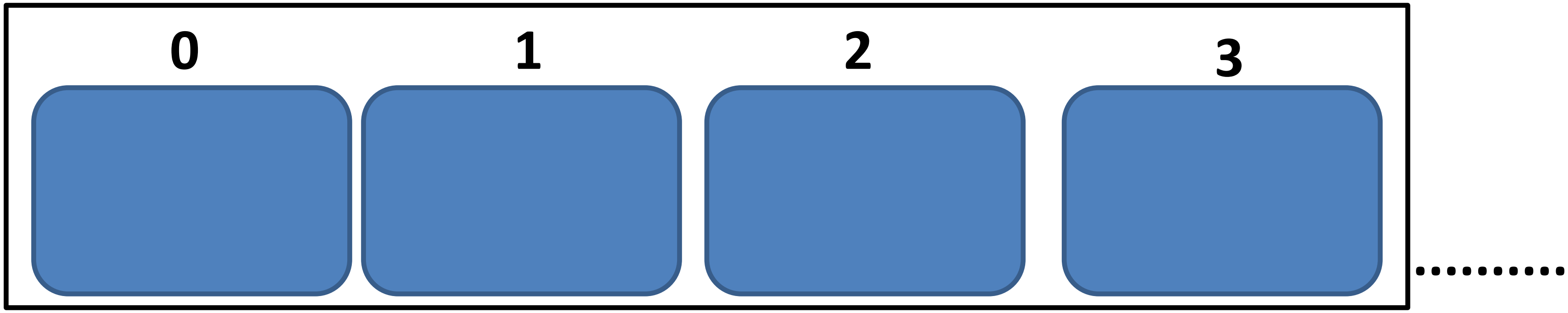


Coin Toss

10 :	7/3	70%**H**	30%**T**
100:	52/48	52%**H**	48%**T**
1000:	502/498	50.2%**H**	49.8%**T**

List

- Like Arrays
- Ordered Sequence of values
- Enumerated starting with zero
- Can be of mixed datatype



List

- list1 = [1,2,3,4,5,6]
- list2 = [‘a’, 55.5, ‘b’,2000]
- list3 = [‘123’,’how are you?’, list2]

list1.append(55)

- range(15)
- myList = list(range(10))

list1[2] =55

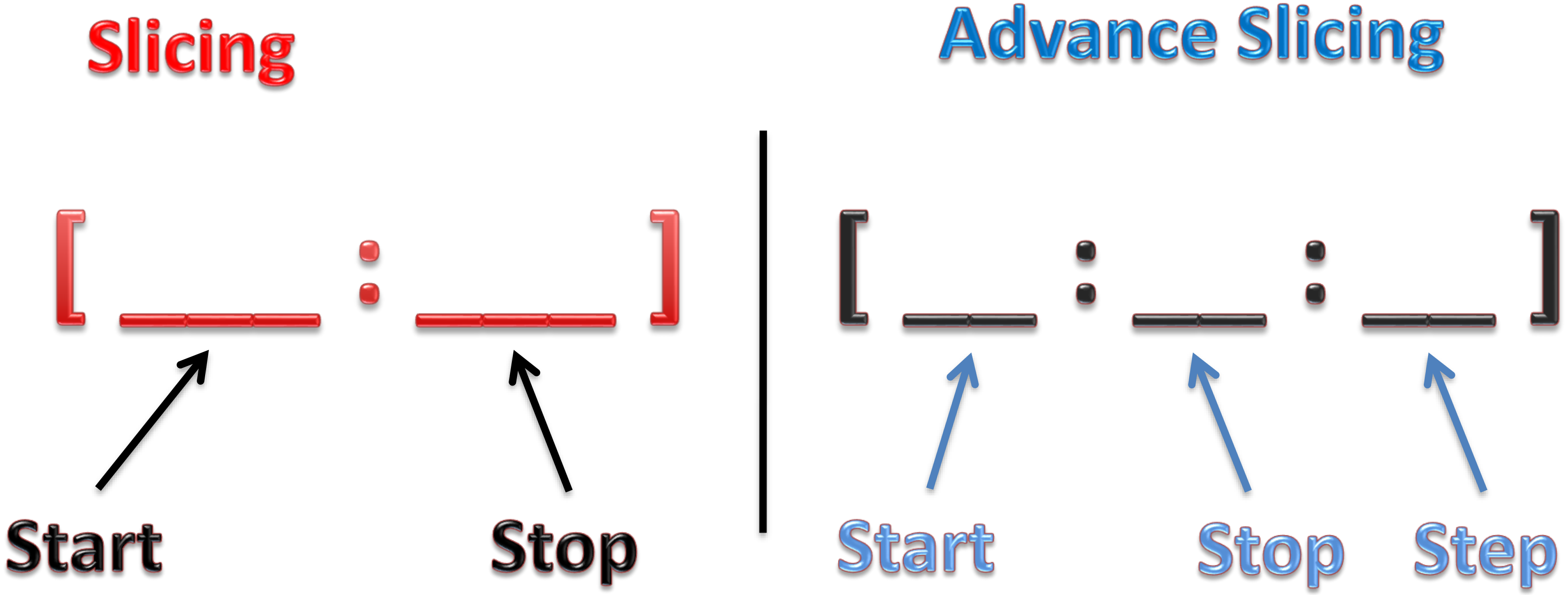
list1.sort()

list1.reverse()

list1.extend(list3)

Slicing

- Subset the list



Slicing

[___ : ___]

Start Stop

Advance Slicing

[__ : __ : __]

Start Stop Step

Slicing

letters

0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I	J
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

letters[:]

letters[: 7]

letters[2 :]

letters[2 : 7]

letters[2: 9 : 2]

letters[-8 : 7]

letters[: : 3]

letters[: : -1]

Tuples

- **Immutable list of values**
  - myTuple = (123, 456, 343)
  - myTuple[:]
  - type(myTuple)
  - len(myTuple)
  - myTuple[1] = 777   --error

Assignment

FINANCIAL STATEMENT ANALYSIS

Packages & Modules

- **Modules** in Python are simply **Python files** with a .py extension.
- The name of the module will be the name of the file.
- A Python **module** can have a set of **functions**, **classes** or **variables** defined and implemented.

e.g. Module color (color.py)
Function red()
Function blue()
Function green()

```
import color
          color.red
          color.green
OR
from color import red

from color import *
```

Packages & Modules

- **Packages** are **namespaces** which contain **multiple packages** and **modules** themselves. They are simply **directories**.

- We create a directory **drawing**

  Include modules in it:

  color, line, rectangle, square, circle

- To use line module from drawing package

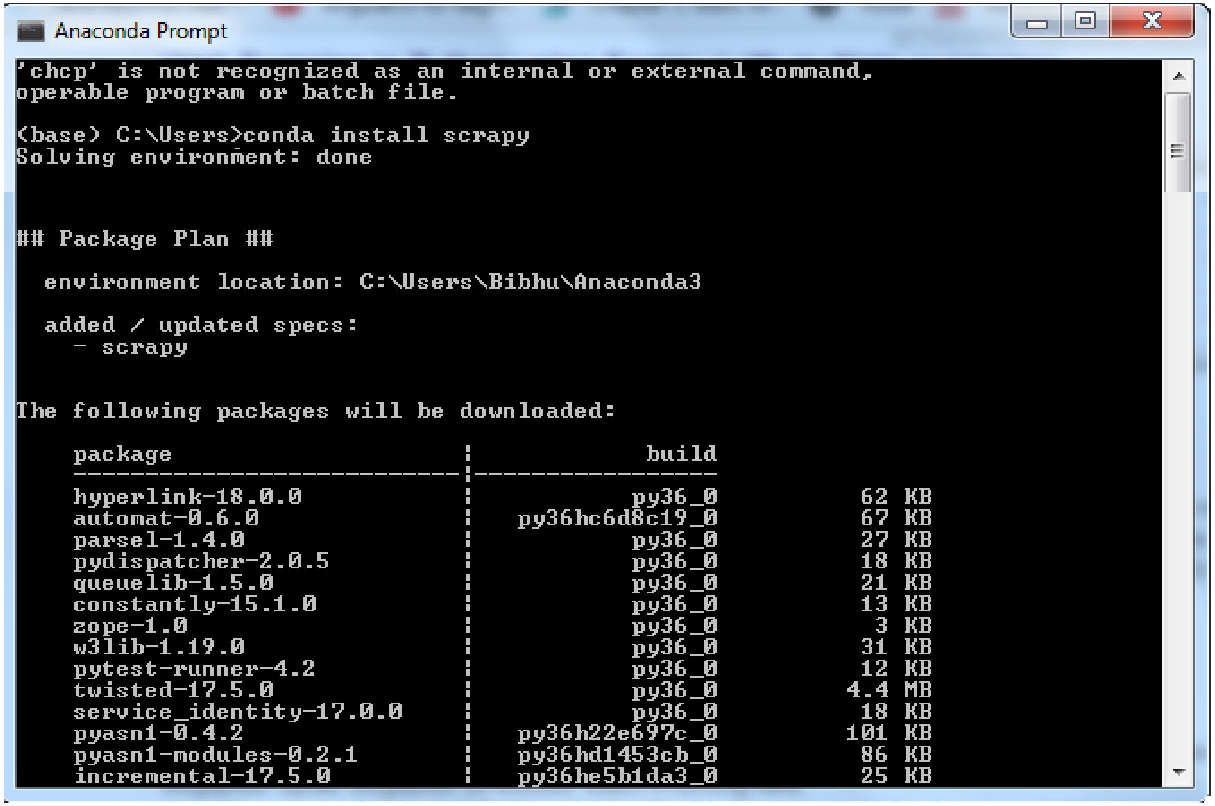
```
import drawing.line
from drawing import circle


import matplotlib.pyplot as plt
from matplotlib import pyplot as plt2
```

Packages & Modules

Install a New Package

conda install packg_name   OR   **pip install** packg_name



```
'chcp' is not recognized as an internal or external command,
operable program or batch file.

(base) C:\Users>conda install scrapy
Solving environment: done


## Package Plan ##

  environment location: C:\Users\Bibhu\Anaconda3

  added / updated specs:
    - scrapy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    hyperlink-18.0.0           |           py36_0          62 KB
    automat-0.6.0              |   py36hc6d8c19_0          67 KB
    parsel-1.4.0               |           py36_0          27 KB
    pydispatcher-2.0.5         |           py36_0          18 KB
    queuelib-1.5.0             |           py36_0          21 KB
    constantly-15.1.0          |           py36_0          13 KB
    zope-1.0                   |           py36_0           3 KB
    w3lib-1.19.0               |           py36_0          31 KB
    pytest-runner-4.2          |           py36_0          12 KB
    twisted-17.5.0             |           py36_0         4.4 MB
    service_identity-17.0.0    |           py36_0          18 KB
    pyasn1-0.4.2               |   py36h22e697c_0         101 KB
    pyasn1-modules-0.2.1       |   py36hd1453cb_0          86 KB
    incremental-17.5.0         |   py36he5b1da3_0          25 KB
```

Numpy Arrays

- Can hold Same Datatype values only
- Contains very powerful and versatile set of methods

```
e.g. import numpy as np
       a = np.array([1,2,3,4,5,6])
       a.min()
               a.mean()
       len(a)
               np.append(a, 55)
```

Slicing Numpy Arrays

- When we slice a list it creates new list
- When we slice a Numpy Array it doesnt create a new array, saving memory

e.g

a = numpy.array([1,2,3,4,5])

b = a[2:]

⇒ **b** is like a **view** pointing to **original array**

⇒ changes to **b** reflect in **a** and **vice versa**

c = a.copy() => creates a new array c

Dictionaries

- A dictionary is an associative array
- Any key of the dictionary is associated (or mapped) to a value.
- The values of a dictionary can be any Python data type
- Dictionaries are unordered key-value-pairs.
- Dictionaries can easily be changed, can be shrunk and grown at run time

Operators on Dictionaries

Operator	**Explanation**
len(d)	returns the number of stored entries, i.e. the number of (key,value) pairs.
del d[k]	deletes the key k together with his value
k in d	True, if a key k exists in the dictionary d
k not in d	True, if a key k doesn't exist in the dictionary d

Dictionaries

```
d1 = {‘key1’ : ‘val1’ , ‘key2’ : ‘val2’, ‘key3’ : ‘val3’ }
d1[‘key1’]
```

Two lists get combined like a zipper

```
 dishes = ["pizza", “pretzel", "
 countries = ["Italy", "Germany", "Sp
country_specialities = zip(countries
```

convert the zipped list to dictionary

```
country_specialities_dict = dict(country_specialities)
```

Matrices

A lot of data used for processing is stored in tabular format and **Matrices** is one solution in Python to manage such type of data

A[0,:] A[:,4] A[2,3] A[row,col]

A =

	0	1	2	3	4
0	21	31	41	51	61
1	22	32	42	52	62
2	23	33	43	53	63

Matrix Operations

- matrix1 + matrix2
- matrix1 - matrix2
- matrix1 * matrix2
- matrix1 / matrix2
- np.matrix.round(matrix1 / matrix2)
- np.nan_to_num(myMatrix)
- for index, item in enumerate(myMatrix)

Visualisation (matplotlib)

import matplotlib.pyplot as plt

%matplotlib inline

- plt.plot()
- plt.legend()
- plt.xlabel()
- plt.ylabel()
- plt.title()
- plt.show()

DataFrames

- A Data frame is a two-dimensional data structure
- Data is aligned in a tabular fashion in rows and columns

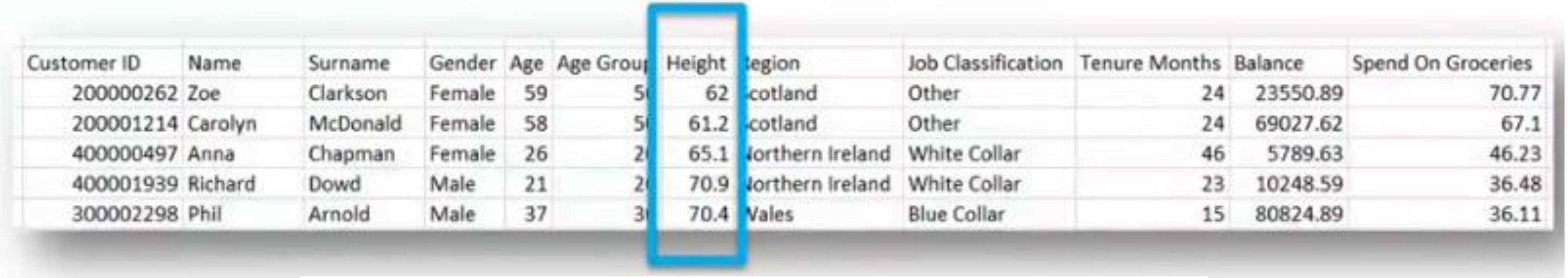
```
import pandas as pd
statsDF = pd.read_csv(‘C:\\......\\file1.csv’)
```

DataFrames

- A Data frame is a two-dimensional data structure
- Data is aligned in a tabular fashion in rows and columns

```
import pandas as pd
statsDF = pd.read_csv(‘C:\\......\\file1.csv’)
```

Standard Deviation



Customer ID	Name	Surname	Gender	Age	Age Grou	Height	egion	Job Classification	Tenure Months	Balance	Spend On Groceries
200000262	Zoe	Clarkson	Female	59	5	62	cotland	Other	24	23550.89	70.77
200001214	Carolyn	McDonald	Female	58	5	61.2	cotland	Other	24	69027.62	67.1
400000497	Anna	Chapman	Female	26	2	65.1	orthern Ireland	White Collar	46	5789.63	46.23
400001939	Richard	Dowd	Male	21	2	70.9	orthern Ireland	White Collar	23	10248.59	36.48
300002298	Phil	Arnold	Male	37	3	70.4	ales	Blue Collar	15	80824.89	36.11

$$\{\ 61.2,\ 62,\ 65.1,\ 70.4,\ 70.9\ \}$$

$$\text{Mean} = \frac{61.2 + 62 + 65.1 + 70.4 + 70.9}{5} = 65.92$$

$$\{\, 61.2,\ 62,\ 65.1,\ 70.4,\ 70.9 \,\}$$

$$\mu \quad \text{Mean} = \frac{61.2 + 62 + 65.1 + 70.4 + 70.9}{5} = 65.92$$

$$\text{Variance} = \frac{(61.2 - 65.92)^2 + (62 - 65.92)^2 + (65.1 - 65.92)^2 + (70.4 - 65.92)^2 + (70.9 - 65.92)^2}{5}$$

$$\text{Variance} = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{\text{N}} = 16.64$$

$$\sigma^2$$

$$\text{Std. Dev.} = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \mu)^2}{\text{N}}} = 4.08$$

$$\sigma$$

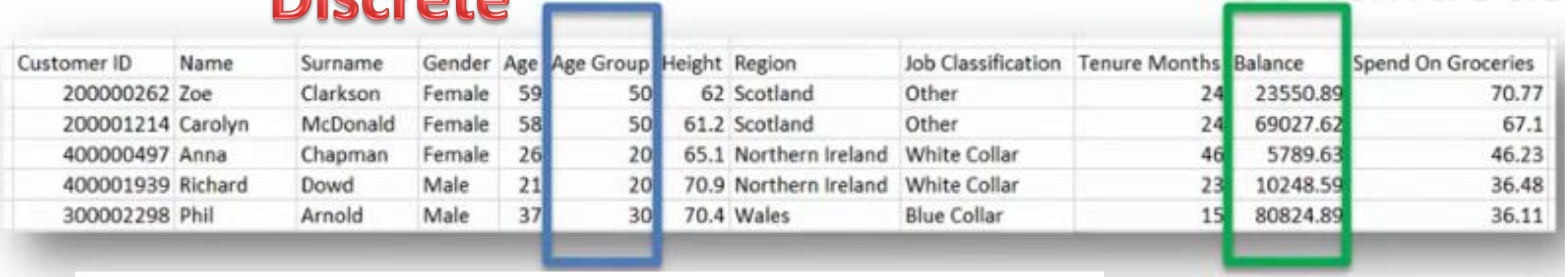
What is Distribution?

In probability theory and statistics, a probability distribution is a mathematical function that provides the probabilities of occurrence of different possible outcomes in an experiment.

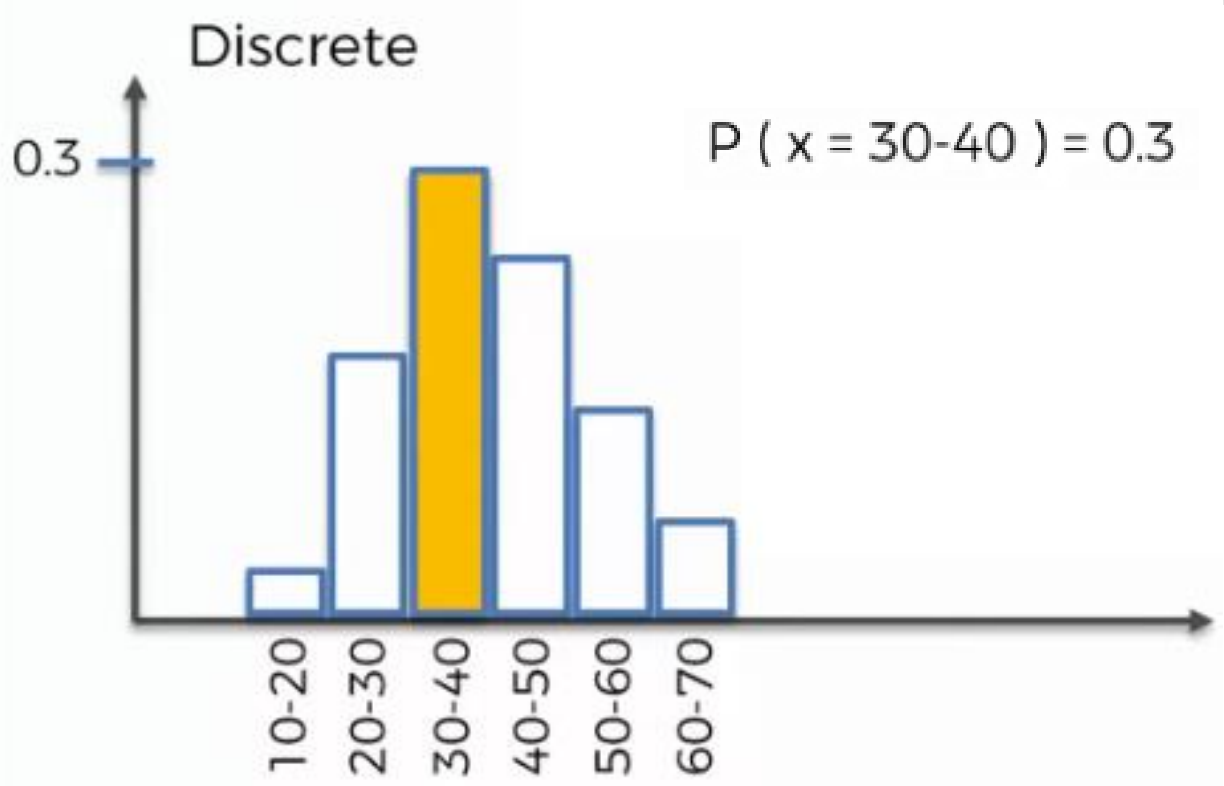
What is Distribution?

Discrete

Continuous



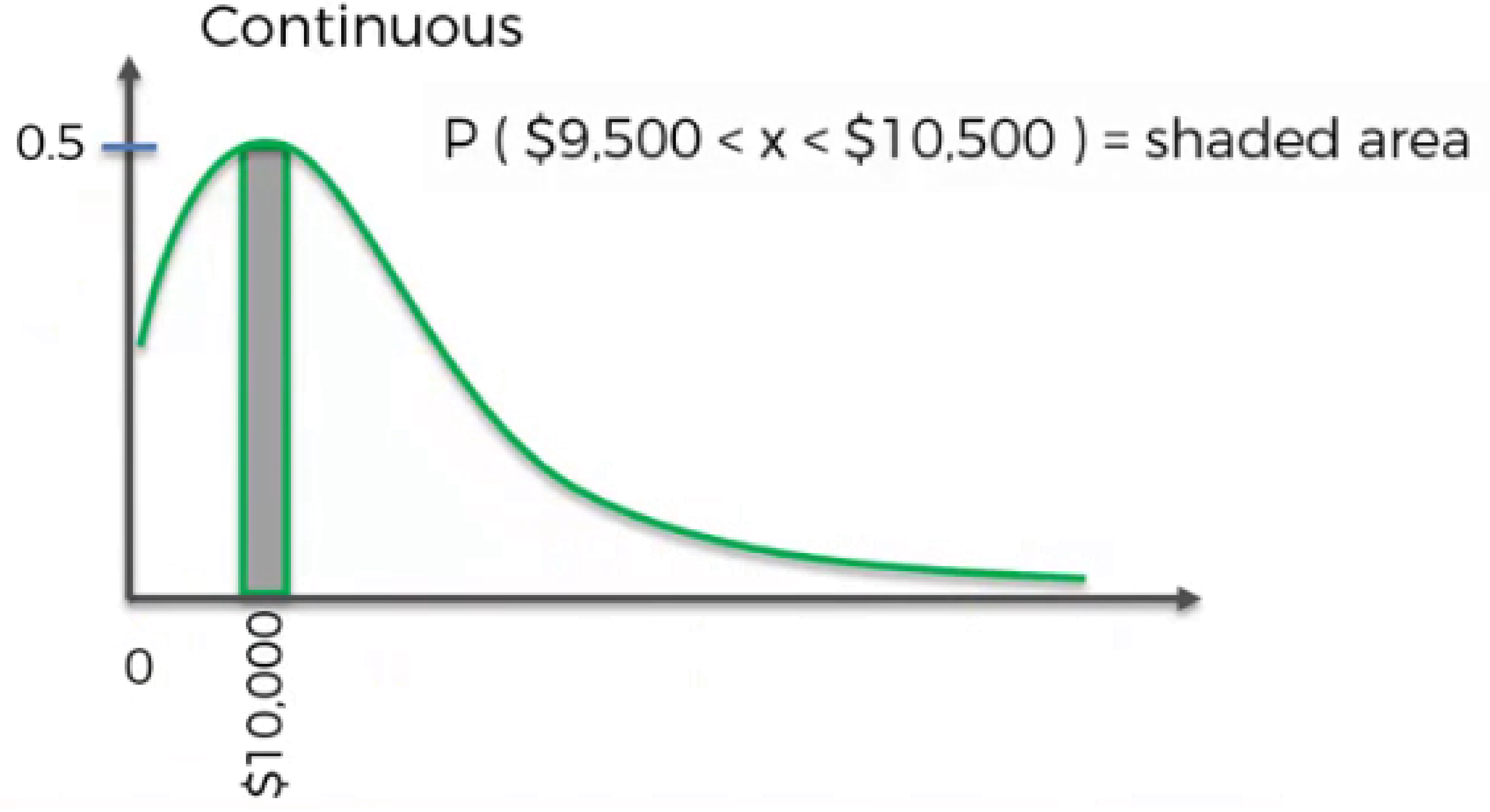
Customer ID	Name	Surname	Gender	Age	Age Group	Height	Region	Job Classification	Tenure Months	Balance	Spend On Groceries
200000262	Zoe	Clarkson	Female	59	50	62	Scotland	Other	24	23550.89	70.77
200001214	Carolyn	McDonald	Female	58	50	61.2	Scotland	Other	24	69027.62	67.1
400000497	Anna	Chapman	Female	26	20	65.1	Northern Ireland	White Collar	46	5789.63	46.23
400001939	Richard	Dowd	Male	21	20	70.9	Northern Ireland	White Collar	23	10248.59	36.48
300002298	Phil	Arnold	Male	37	30	70.4	Wales	Blue Collar	15	80824.89	36.11



Discrete

P (x = 30-40) = 0.3

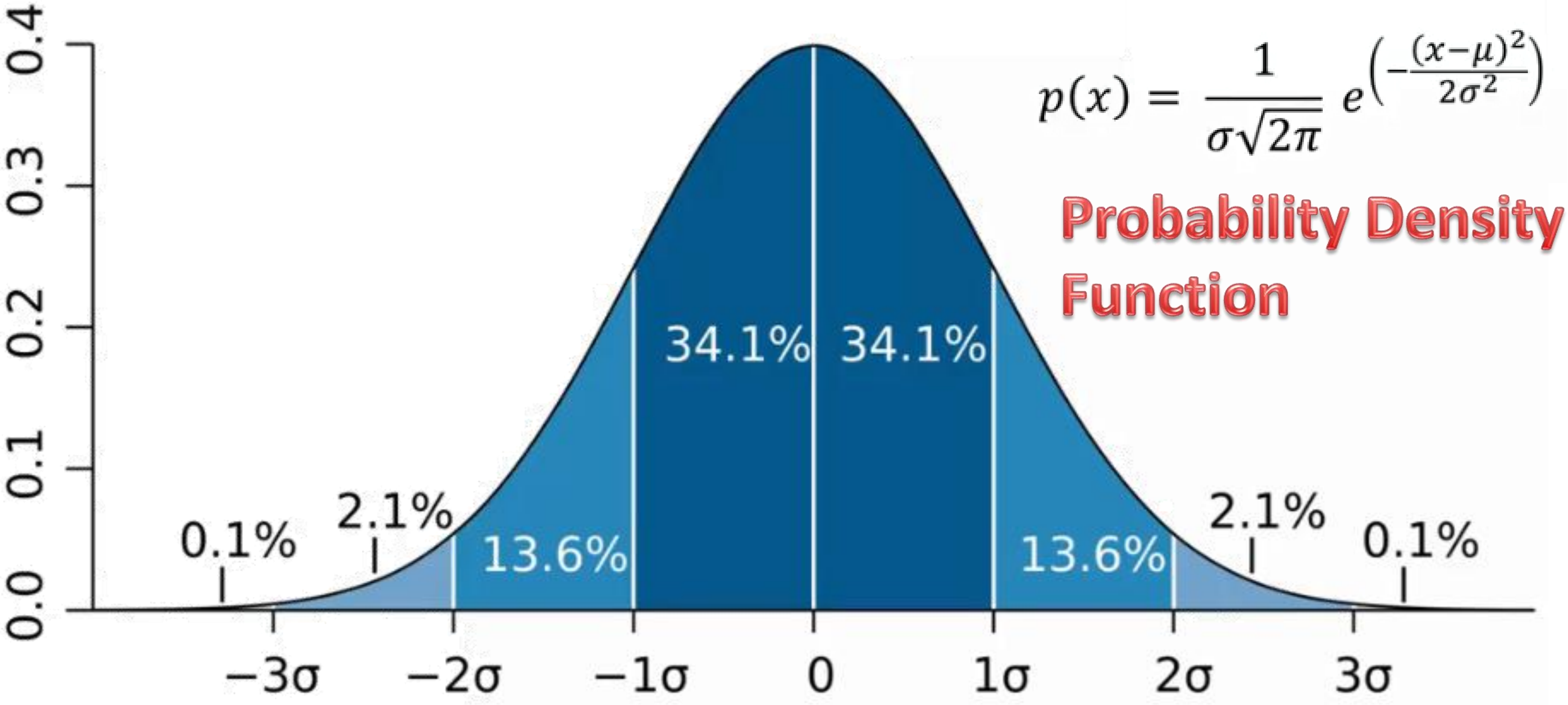
....



Continuous

P ($9,500 < x < $10,500) = shaded area

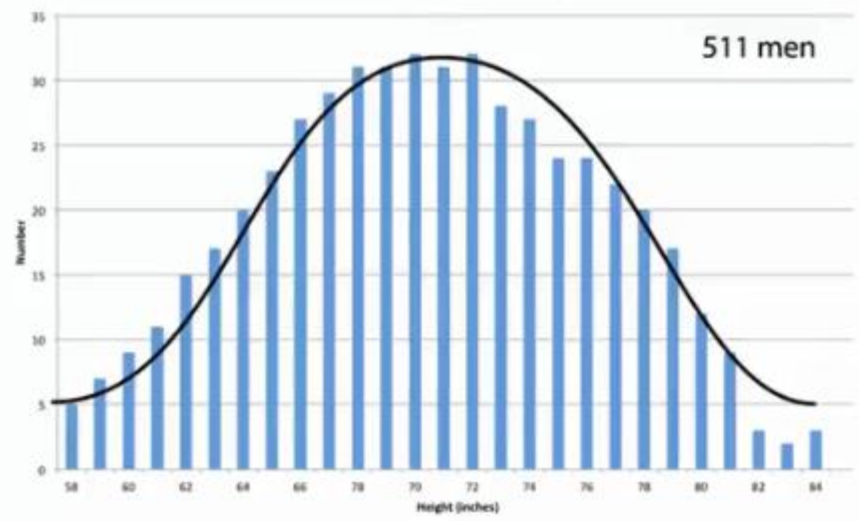
Normal Distribution

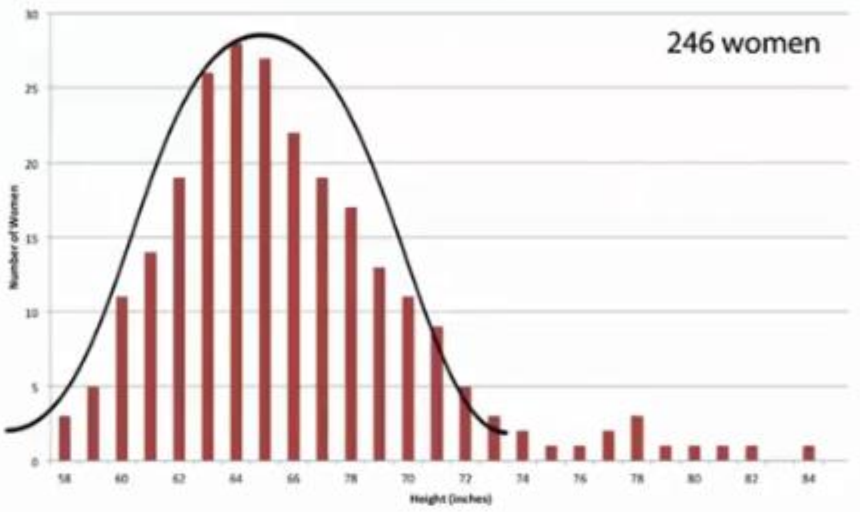


$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

Probability Density Function

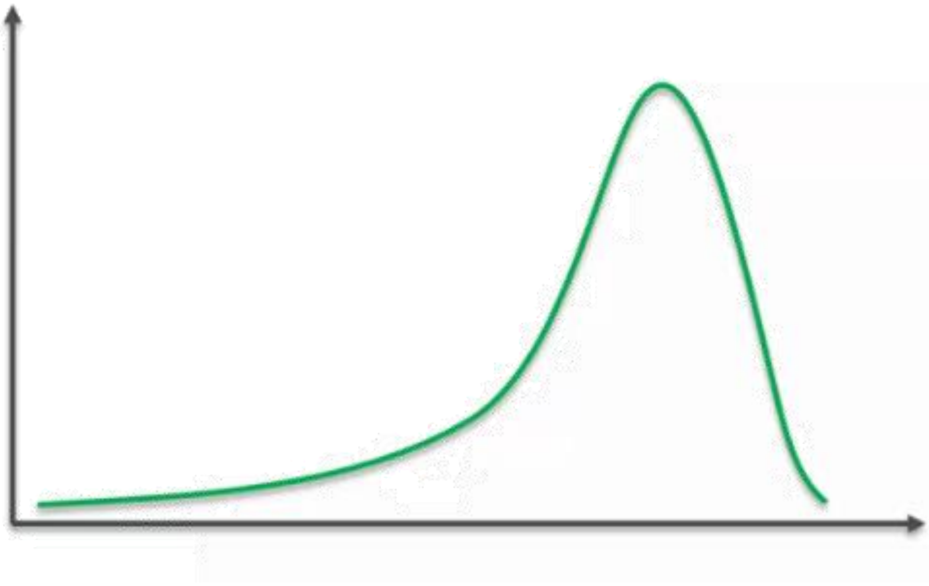
Height distribution of 20-yr old men and women in India



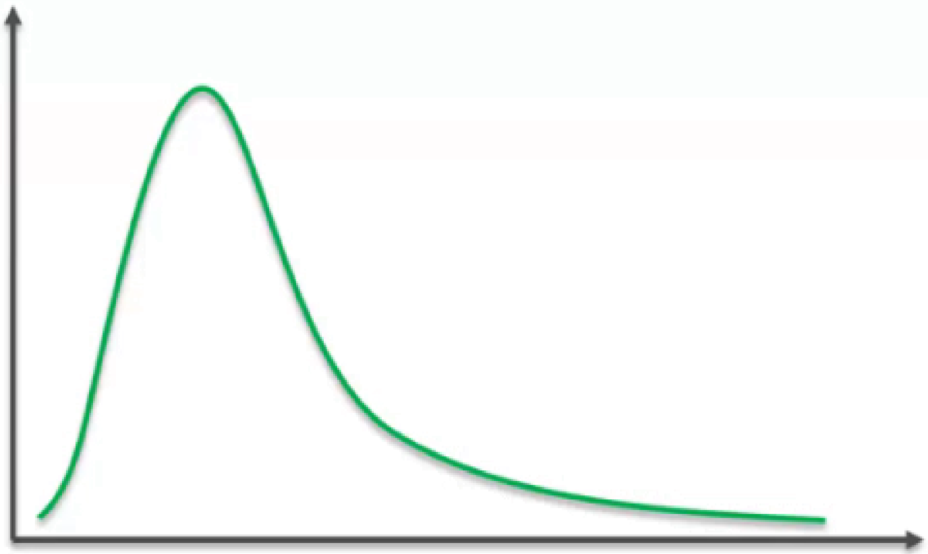


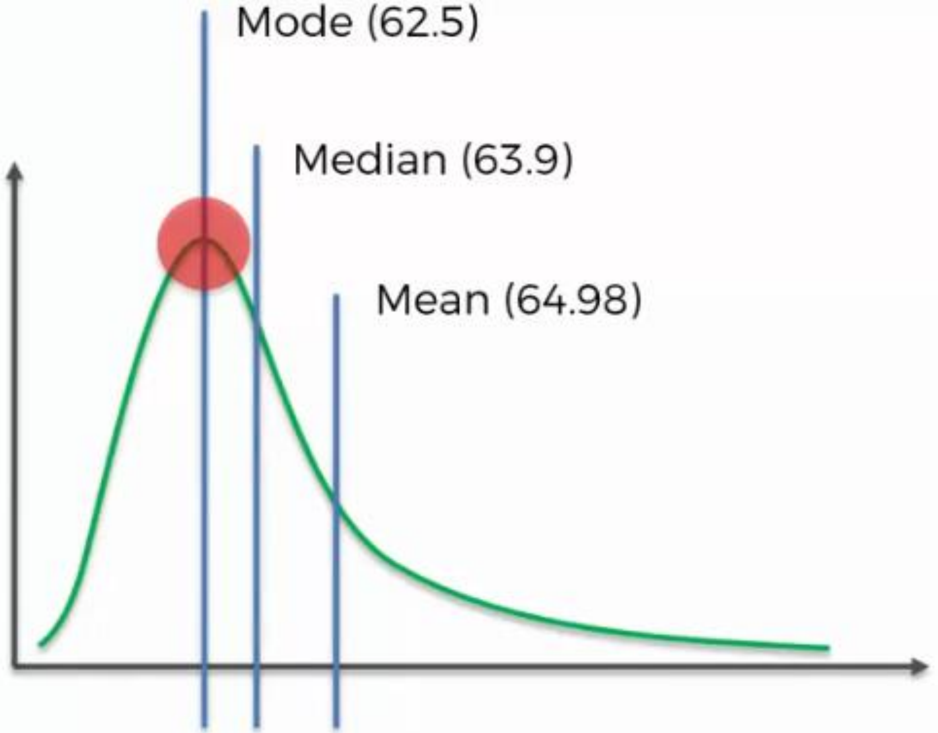
Skweness

Left (Negative) Skew



Right (Positive) Skew



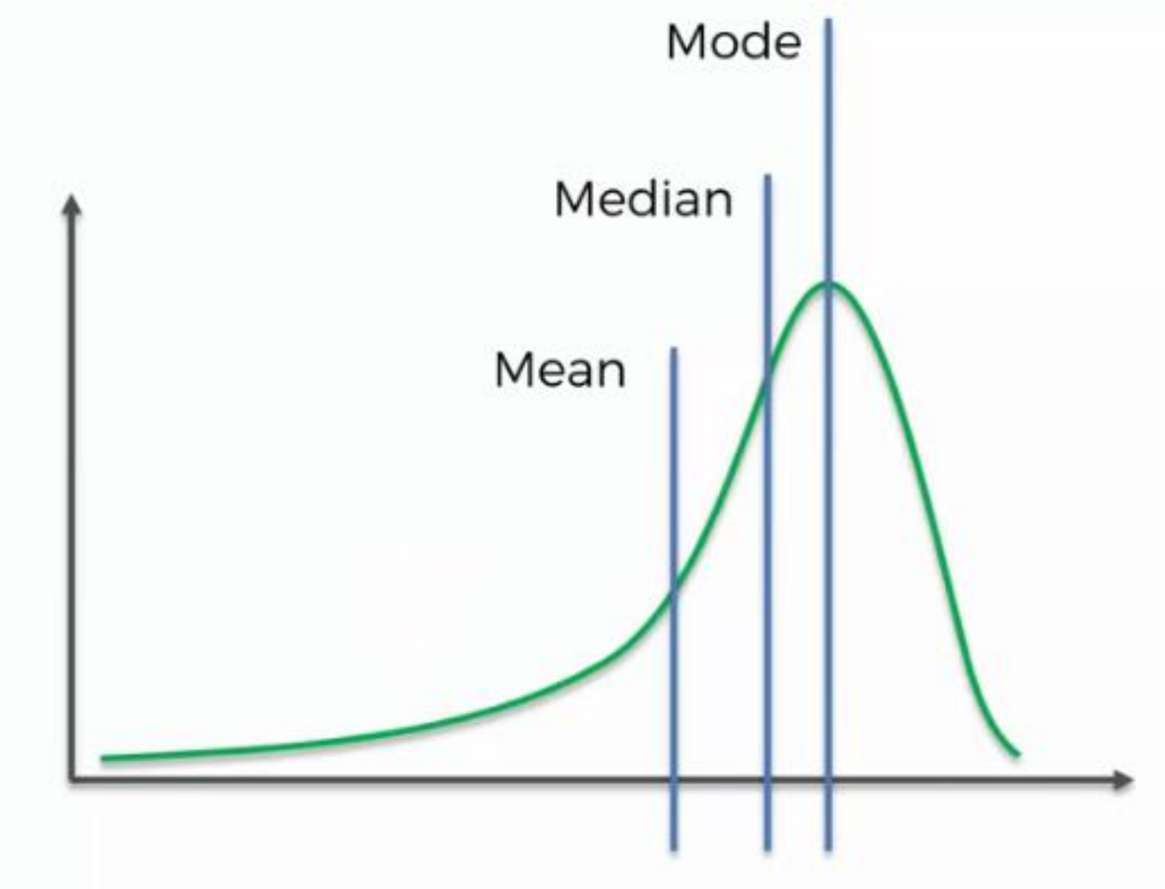


{ 58.8 59.9 61.2 61.3 61.5 62 62.5 62.5 62.5 63 63 63.7 63.9 64 64 65.1 65.2 66.7 67.8 68.3 69.9 70.7 71.8 72.2 73 }

{ 58.8 59.9 61.2 61.3 61.5 62 62.5 62.5 62.5 63 63 63.7

63.9

64 64 65.1 65.2 66.7 67.8 68.3 69.9 70.7 71.8 72.2 73 }


Mode
Median
Mean

Seaborn Package

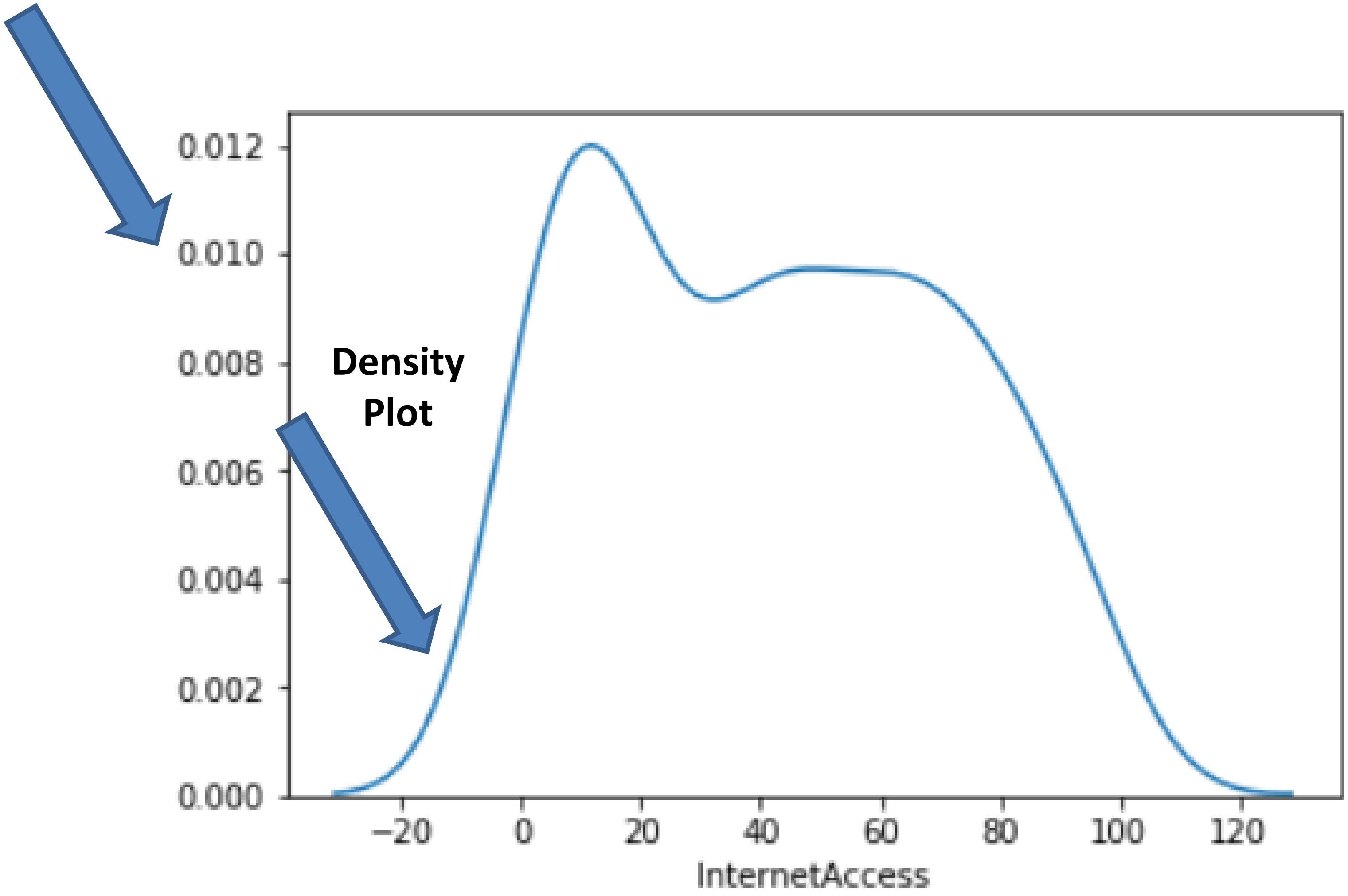
- Seaborn is a Python visualization library based on matplotlib.
- It provides a high-level interface for drawing attractive statistical graphics.

```
import seaborn
seaborn.distplot()
```

Creating a univariate distribution in seaborn with distplot()

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
plt.figure(figsize=(3,4))
sns.distplot(DF.InternetAccess )
plt.show()
```

Probability density : Probability per unit on the x-axis



Density Plot

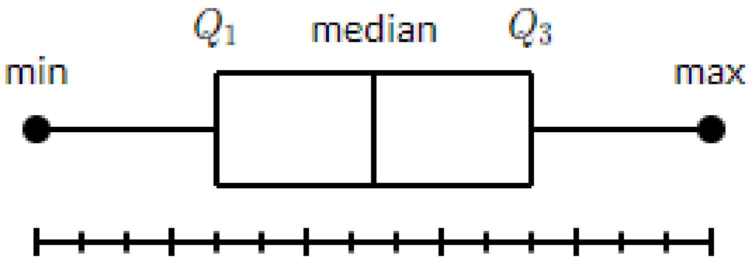
```
sns.distplot(df["Colunm"] ,
      hist=True,
      kde=True,
  kde_kws = {'shade':True,
                  'linewidth': 3,"color":"Red"},
  hist_kws={'edgecolor':'black'} )
```

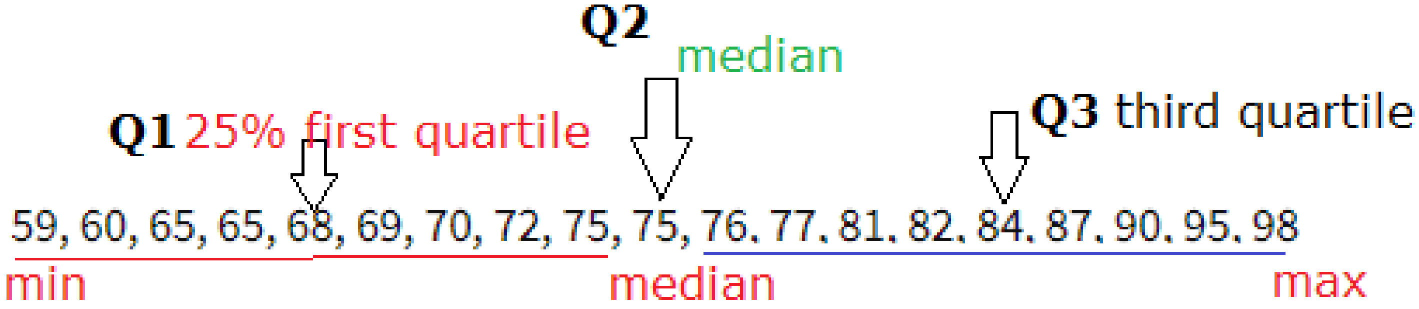
BoxPlot

- boxplot(), shows the distribution of quantitative data in a way that facilitates comparisons between variables

```
vis2 = sns.boxplot(data=DF, x="IncomeGroup", y="BirthRate")
```

Quartiles





Q2

median

Q1 25% first quartile

Q3 third quartile

59, 60, 65, 65, 68, 69, 70, 72, 75, 75, 76. 77. 81. 82. 84. 87. 90. 95. 98

min

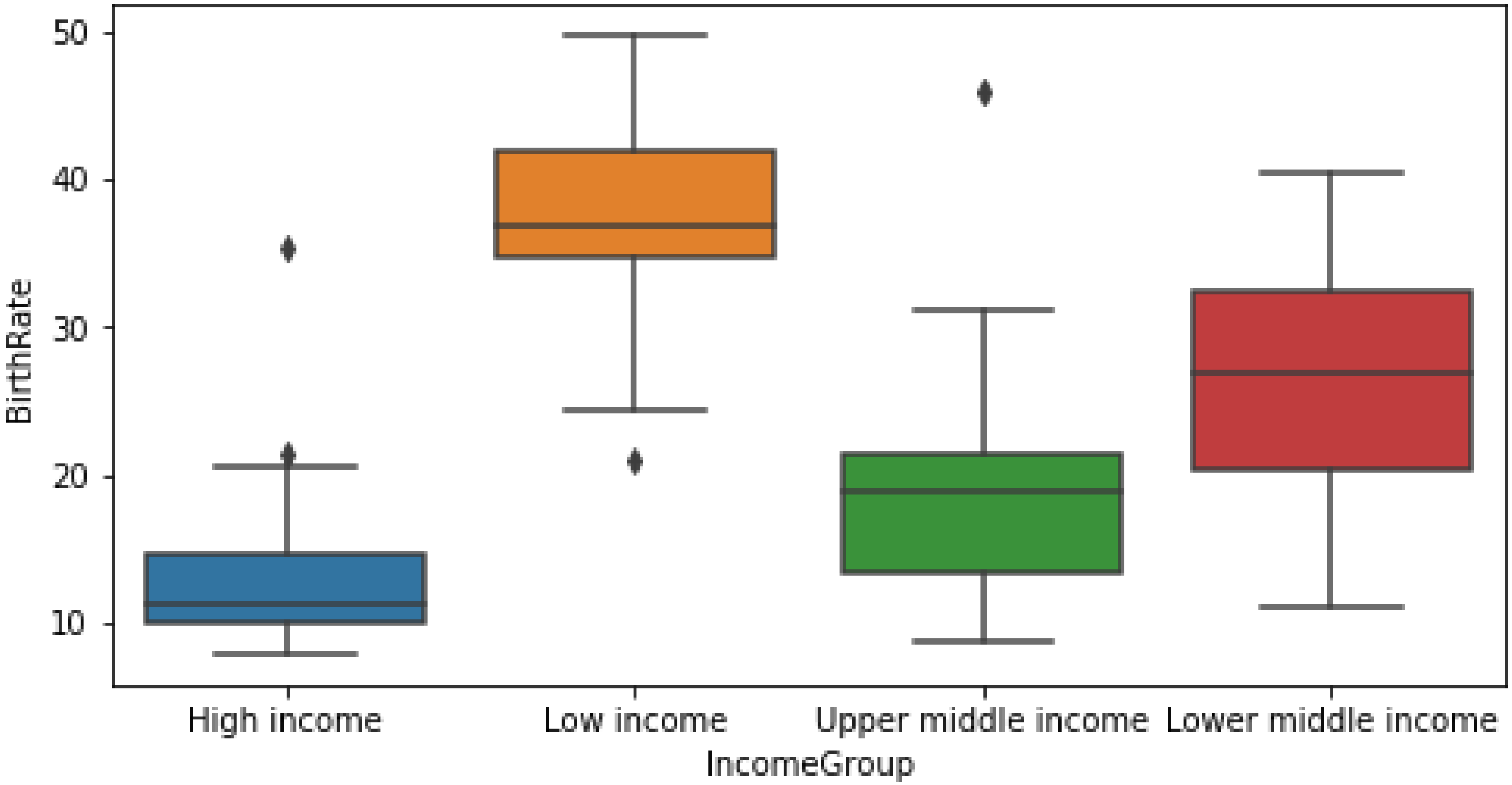
median

max

Total 19 values

The extra pointers plotted indicate outliers
(e.g. few high income rate class having high birthrate)

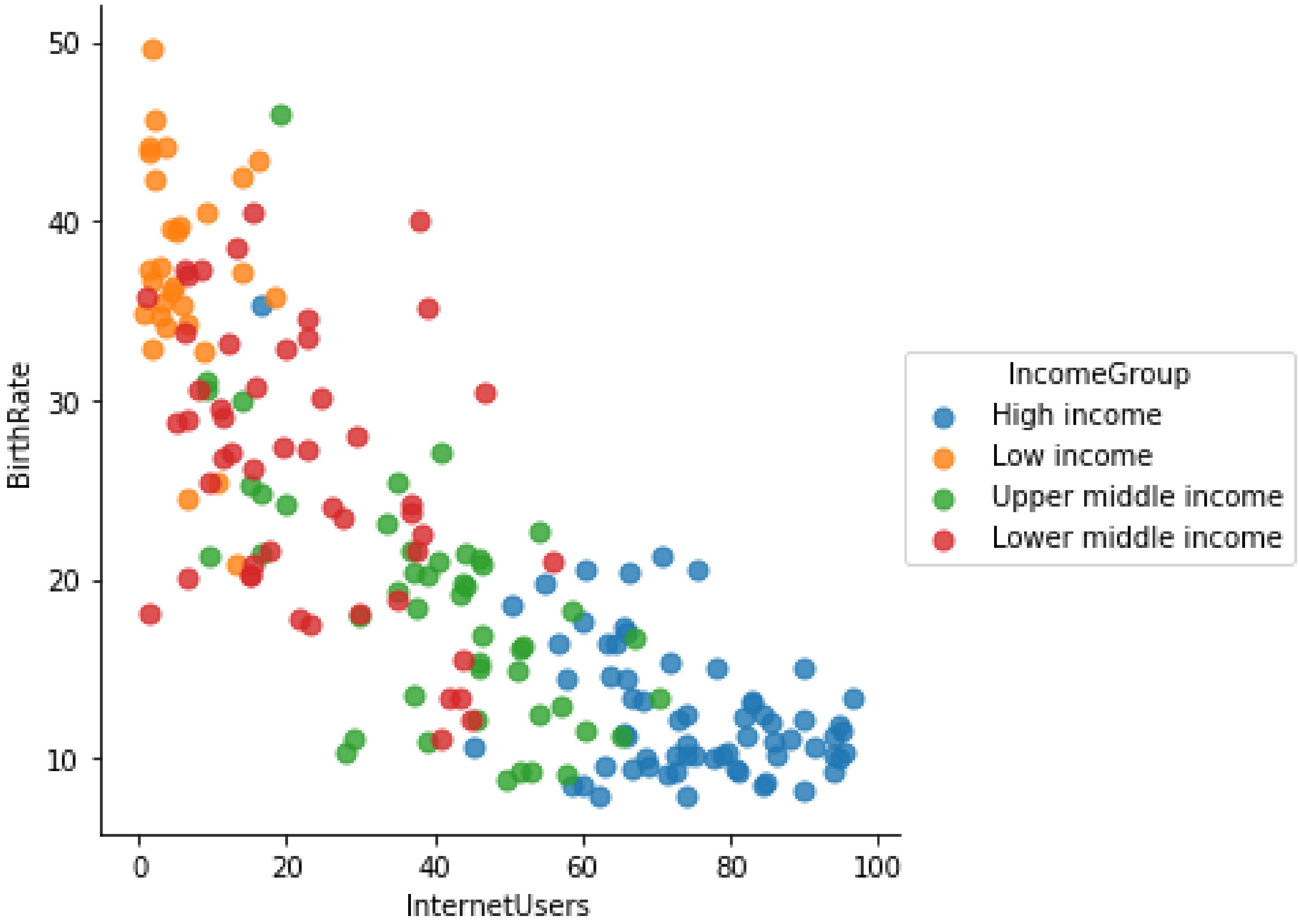
Task : Confirm the outliers and the plotted density values by using appropriate functions



lmplot

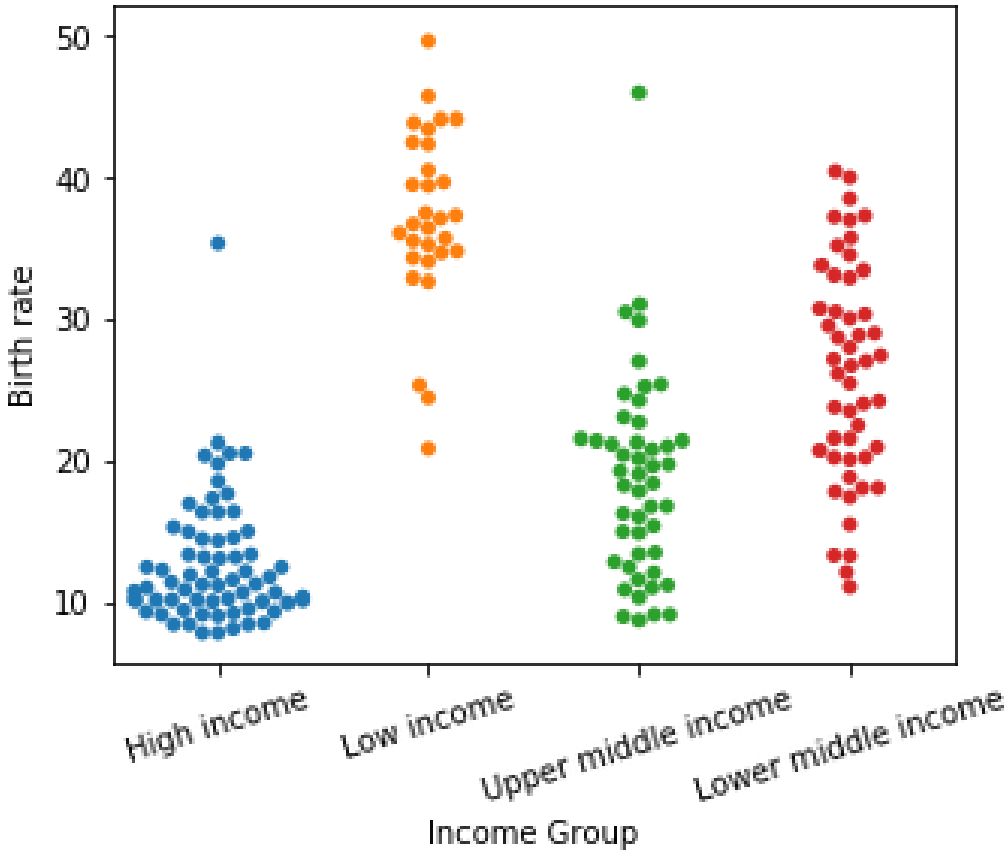
- **BirthRate Vs Internet Users**
- scatter_kws is a wrapper for plt.scatter (matplotlib.pyplot.scatter), so to size the markers we need to pass value to the scatter_kws as a dictionary(key:value) , where s is the size of the marker

```
vis3 = sns.lmplot(
          data = DF, x="InternetAccess“, y="BirthRate",
          fit_reg=False,  hue="IncomeGroup", size=5)
          scatter_kws={"s":50})
```


BirthRate
50
40
30
20
10
0
20
40
60
80
100
InternetUsers
IncomeGroup
High income
Low income
Upper middle income
Lower middle income

swarmplot

```
s=sns.swarmplot(data=DF,
        x="IncomeGroup", y="BirthRate")
```



Pearson Coefficient

- **The Pearson correlation coefficient measures the linear relationship between two datasets**
- **Varies between -1 and +1 with 0 implying no correlation**
- **Correlations of -1 or +1 imply an exact linear relationship**
- **Positive correlations imply that as x increases, so does y**
- **Negative correlations imply that as x increases, y decreases**

Pearson Coefficient

```
                N *sum(xy) - sum(x)*sum(y)
Pearsonr =      -------------------------------------------------------------
                sqrt([N*sum(x^2)- sum(x)^2]*[N*sum(y^2)-sum(y)^2])
```

p-Value

- The p-value roughly indicates the probability of an uncorrelated system
- The p-values are not entirely reliable but are probably reasonable for datasets larger than 500 or so
- p-value is measured with a significance level of 0.05
- p-value below 0.05 indicate correlation
- p-value above 0.05 indicate no correlation

The p-value for Pearson's correlation coefficient uses the t-distribution.

The T distribution, also known as the Student's t-distribution, is a type of probability distribution that is similar to the normal distribution with its bell shape but has heavier tails.
T distributions have a greater chance for extreme values than normal distributions.

$$t = \frac{r \times \sqrt{n-2}}{\sqrt{1-r^2}}$$

The p-value is => tDistribution(Value of T, degree of freedom)

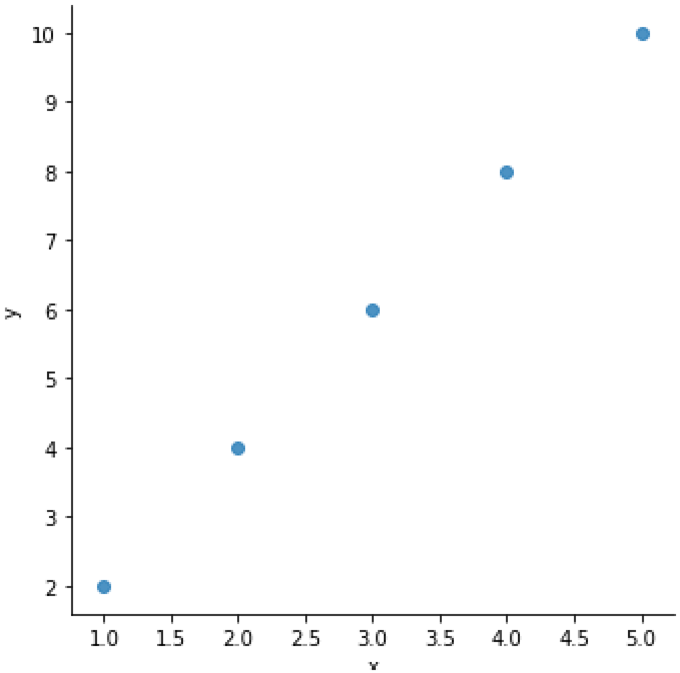
t distribution with degrees of freedom= n – 2

In statistics, the number of degrees of freedom is the number of values in the final calculation of a statistic that are free to vary.

pearsonr

```
from scipy.stats.stats import pearsonr
        pearsonr([1,2,3,4,5], [2,4,6,8,10])
```

Result => (1.0)



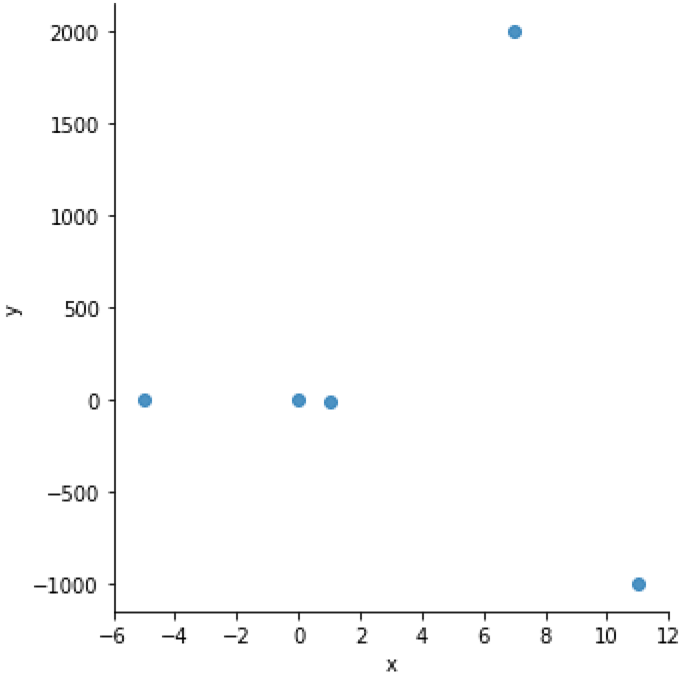
There is a **perfect linear rel**

x & y

pearsonr

pearsonr([0,7,11,1,-5],[-2,2000,-1000,-11,0])

Result => (0.008211472)



No linear relationship

between x & y