

Python

Python Environment



ANACONDA®

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

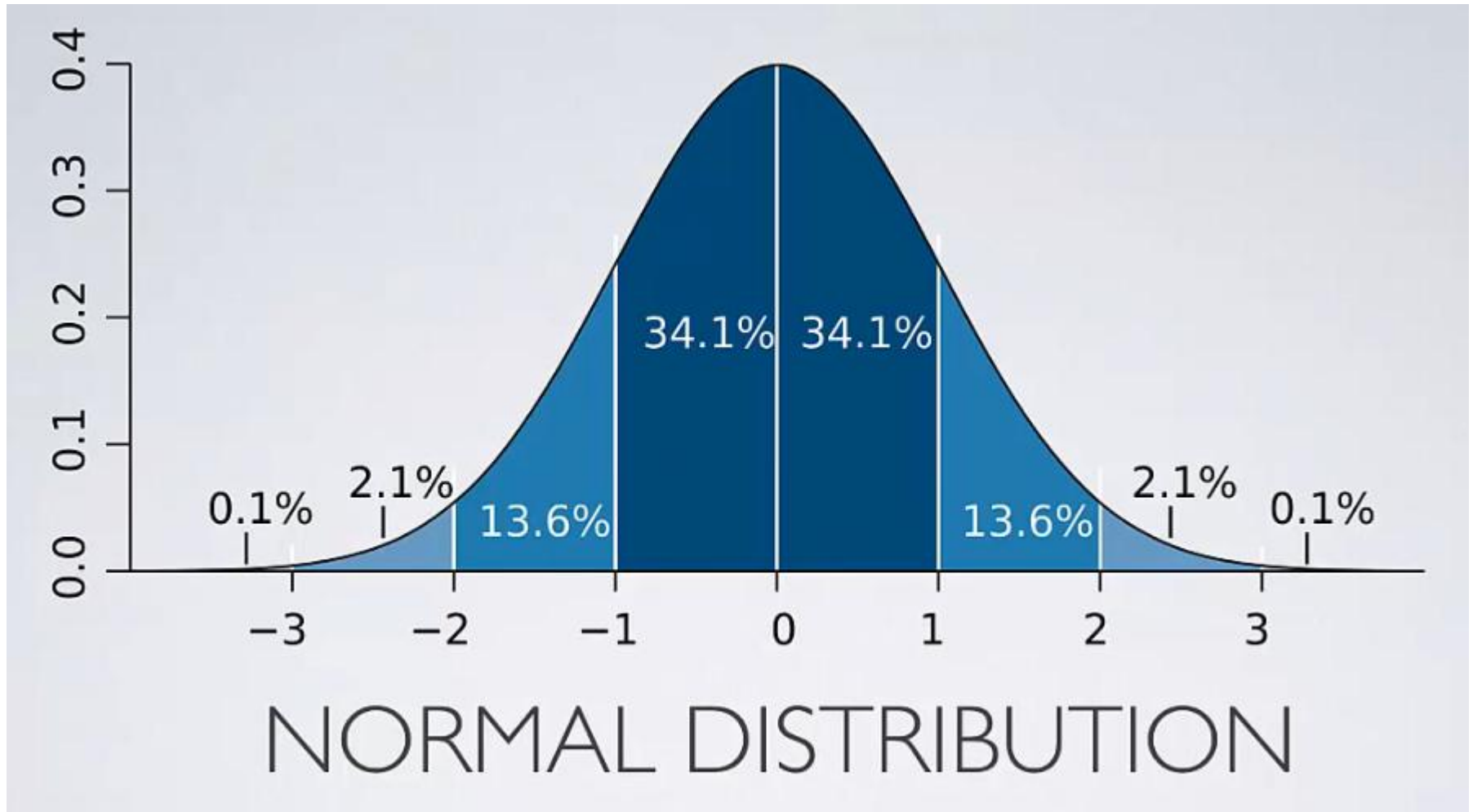
Test The Law OF Large Numbers for N random normally distributed numbers with mean = 0, stdev = 1

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 $\text{Mean}(X_n) \rightarrow E(X)$ as you rerun your script while increasing N

Assignment

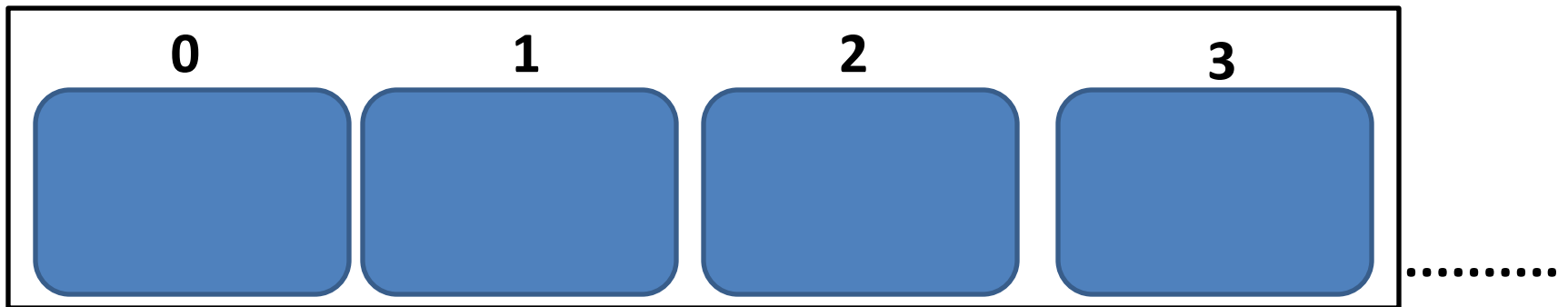


Coin Toss

- 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



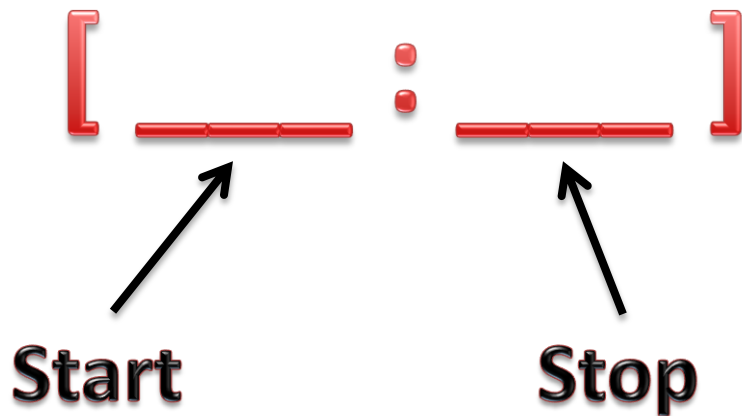
List

- `list1 = [1,2,3,4,5,6]`
- `list2 = ['a', 55.5, 'b',2000]`
- `list3 = ['123','how are you?', list1]`
 - `list1.append(55)`
- `range(15)`
 - `list1[2] = 55`
- `myList = list(range(10))`
 - `list1.sort()`
 - `list1.reverse()`
 - `list1.extend(list3)`

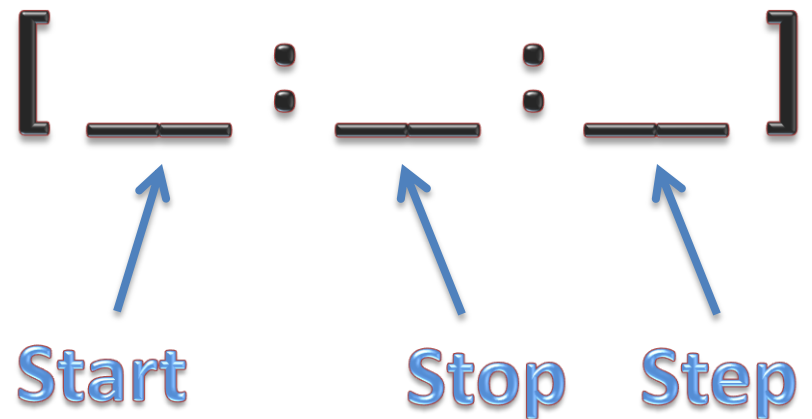
Slicing

- Subset the list

Slicing



Advance Slicing



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

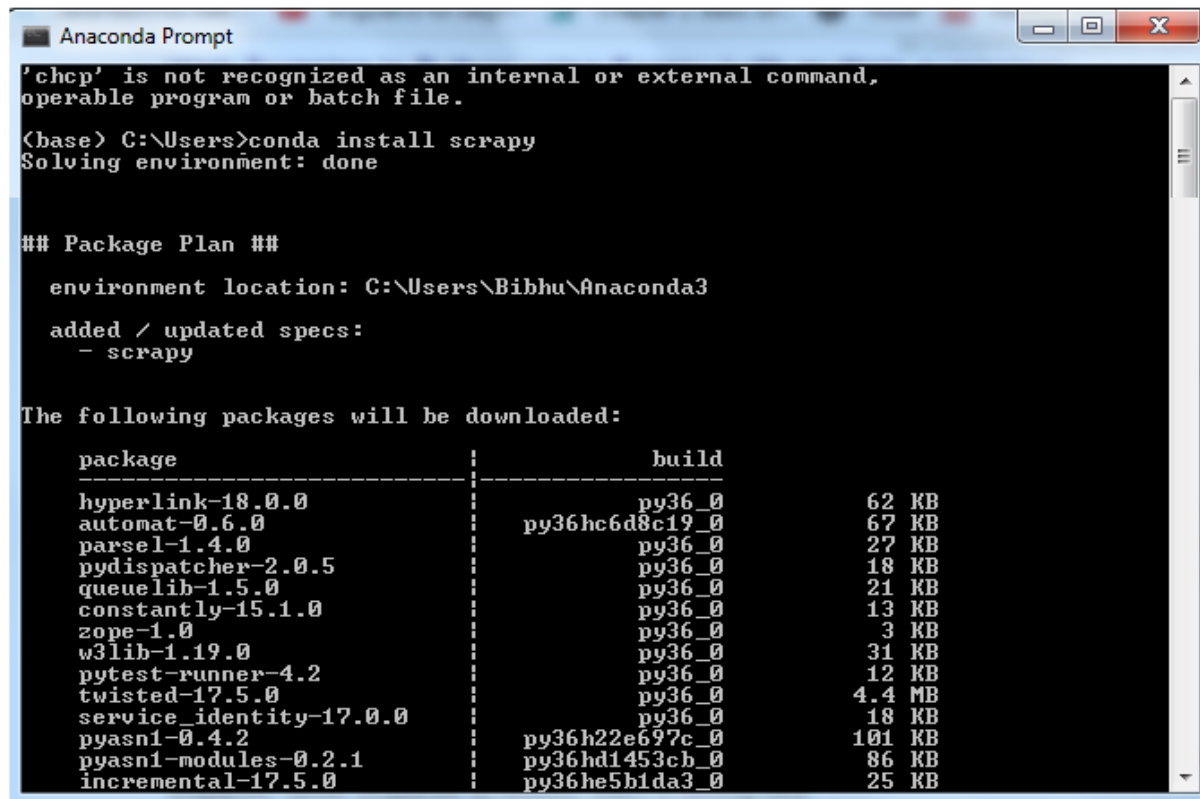
- **Packages** are **namespaces** which contain **multiple packages** and **modules** themselves. They are simply **directories**.
- We create a directory **drawing**
Include modules in it:
color, line, rectangle, square, circle
- To use line module from drawing package
import drawing.line
from drawing **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**



```
Anaconda Prompt

'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	py36hd1453ch_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 doesn't 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

`len(d)`

Explanation

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", "sushi"]  
countries = ["Italy", "Germany", "Spain"]  
country_specialities = zip(countries, dishes)
```

**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()

Functions

def function_name(formal_param....):

 function_body

 executable line of code

calling the function

function_name(actual_param....)

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')
```

```
import os
```

```
os.getcwd() => get current directory
```

```
os.chdir()      => change directory
```

```
statsDF = pd.read_csv('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')
```

Exploring DataSet

`len(df)`

`df.head()`

`len(df.columns)`

`df.columns`

`df.tail()`

`df.info()`

`df.describe()`

Rename Columns

SubSet DataFrames

```
df[4:8]['BirthRate']
```

```
df[::-1]
```

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

```
df.drop([0,100])
```

```
df[21:26]
```

```
df.drop(list(range(0,100)))
```

```
df["myCalc"] = df.BirthRate*df.InternetUsers
```

```
df.BirthRate
```

```
df[['CountryName','BirthRate']]
```


Filtering DataFrame

```
df.rating < 2
```

```
df2.iat[10,0]
```

```
df[(df.rating < 2) & (df.crowd > 40)]
```

```
df[df.product == 'veggies']
```

```
df.designations.unique()
```

```
df2.at[10,'CountryName']
```

Drop Row / Column From DataFrame

```
df.drop("colname",axis=1,inplace=True)
```

```
df.drop(df[condition] ,inplace=True)
```

Advance Actions on DataFrame

- Creating a DataFrame From Array

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

```
DfFromArray = pd.DataFrame(InputArray)
```

- Creating DataFrame from Dictionary

```
InputDictn = {'Col1': ['1', '3'], 'Col2': ['1', '2'], 'Col3': ['2', '4']}
```

```
DfFromDictn = pd.DataFrame(InputDictn)
```

Advance Actions on DataFrame

- Creating DataFrame from list

```
DfFromList = pd.DataFrame(  
    data=[40,50,60,70], columns=['ColA'] )
```

- Creating DataFrame from DataFrame

```
DfFromDF = DfFromDictn[:,:]  
DfFromDF = DfFromDictn.copy()
```

Merging DataFrames

- Join the two dataframes along rows

```
pd.concat([DF1 , DF2 ])
```

- Join the two dataframes along columns

```
pd.concat([DF1 , DF2 ], axis=1)
```

- Merge two dataframes

```
pd.merge(DF1 , DF2 , on='common_col')
```

- Merge with inner join

```
pd.merge(DF1 , DF2, on= 'common_col ', how='inner')
```

Standard Deviation

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

{ 61.2, 62, 65.1, 70.4, 70.9 }

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

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

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

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

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

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

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.

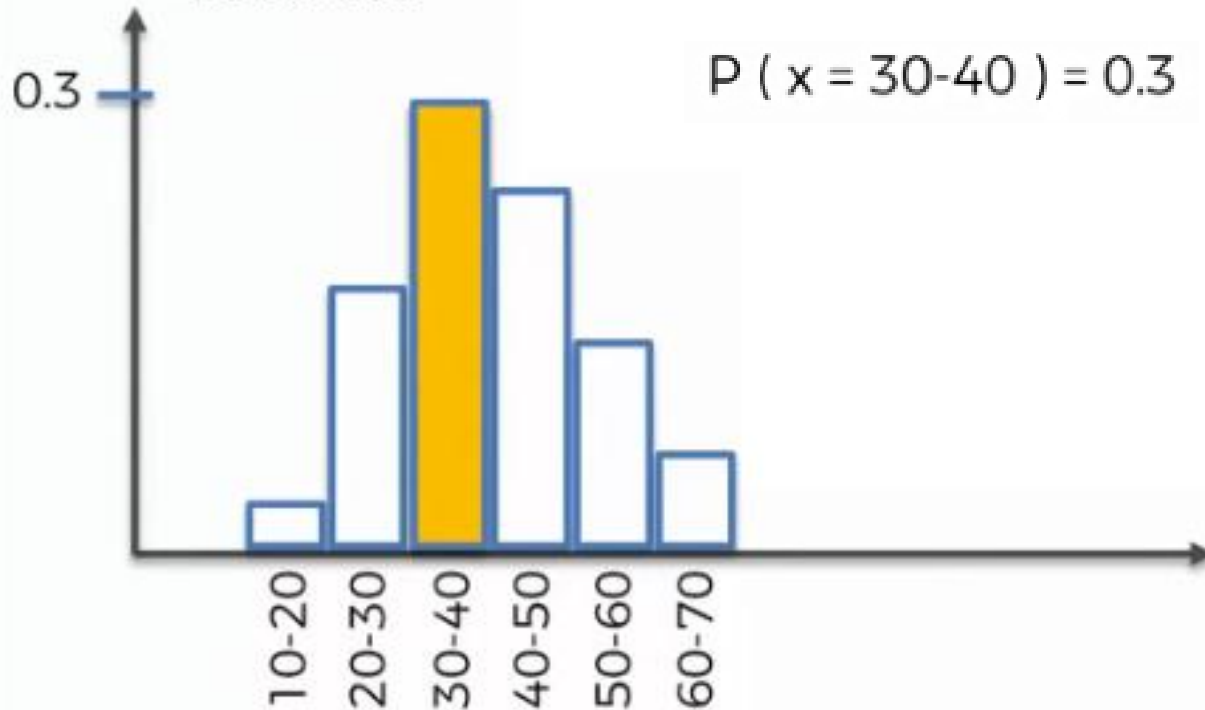
What is Distribution?

Continuous

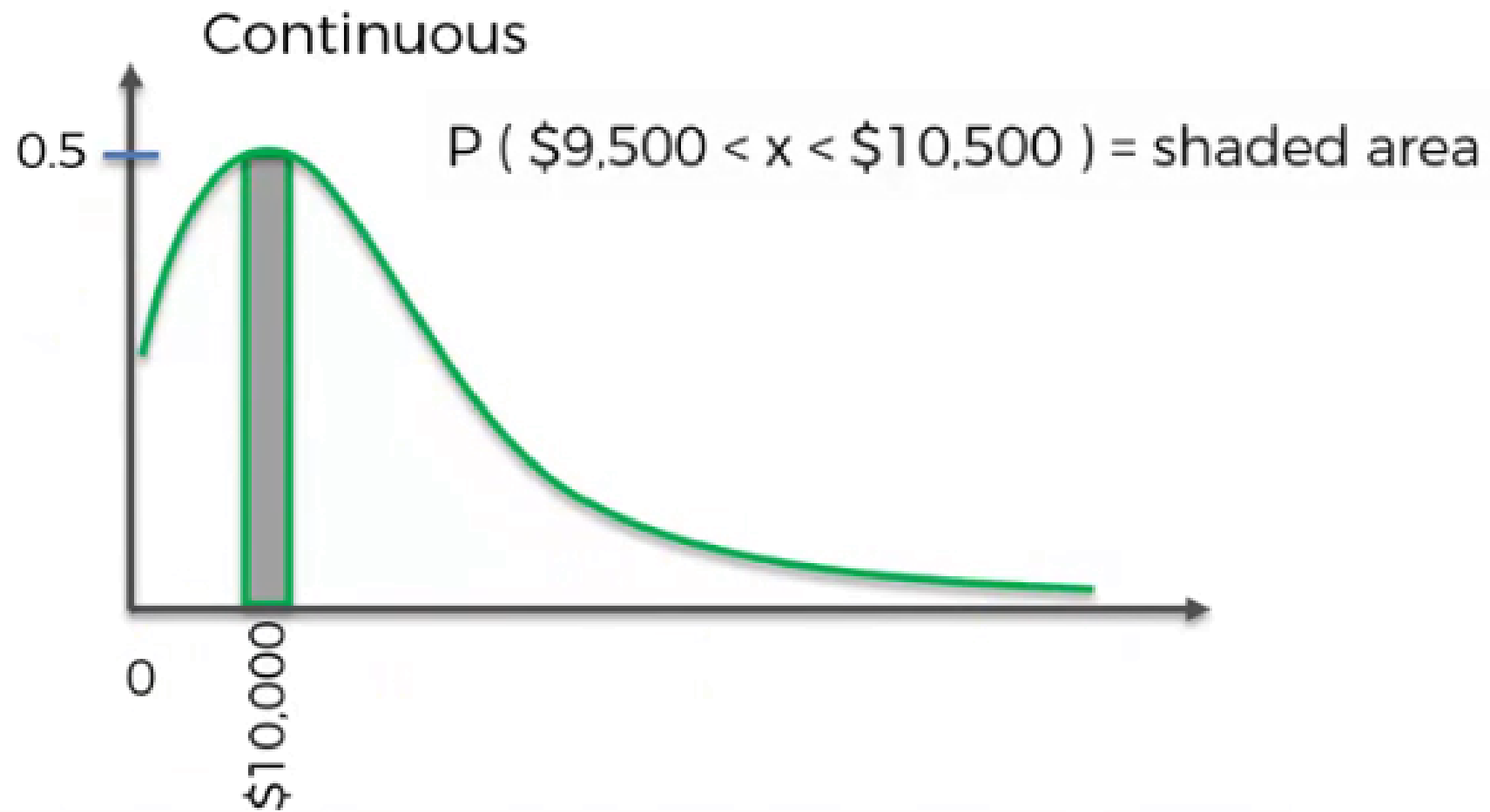
Discrete

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

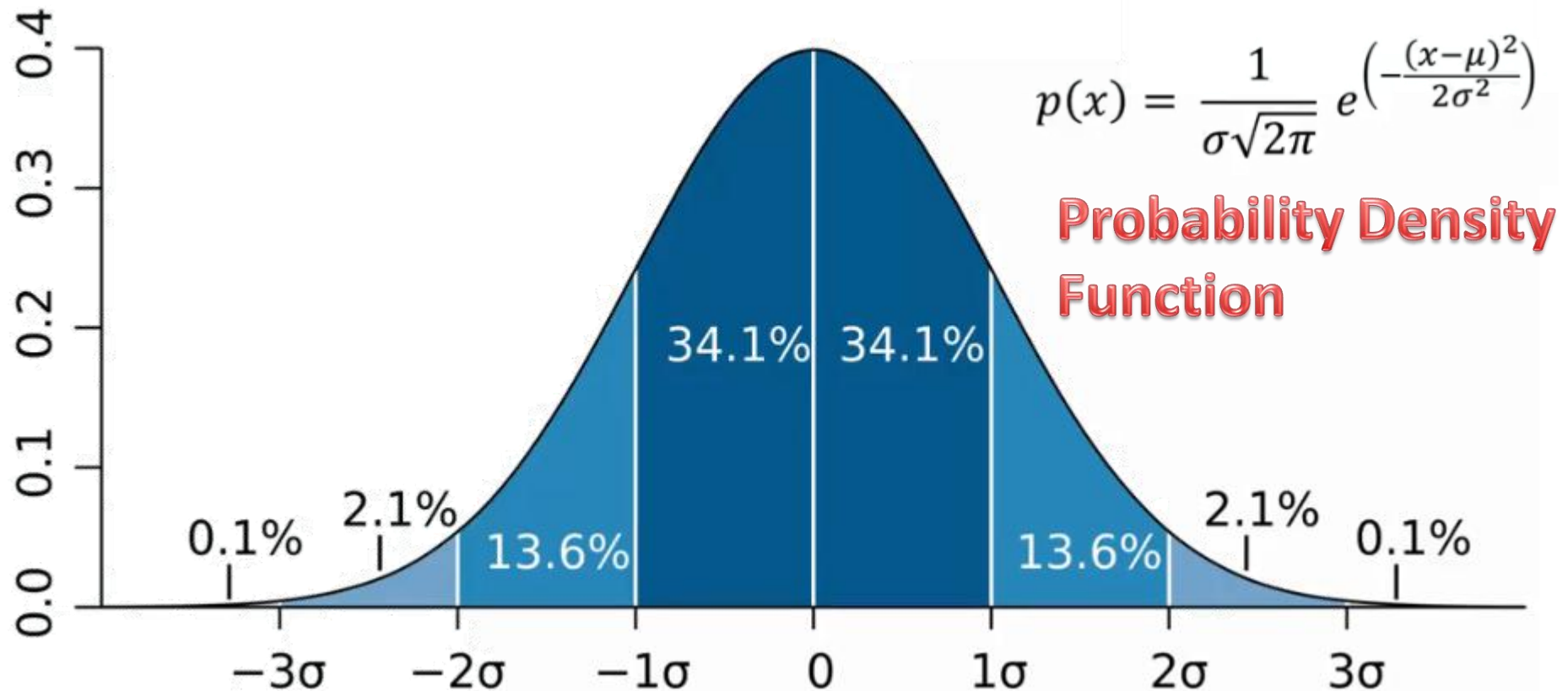
Discrete



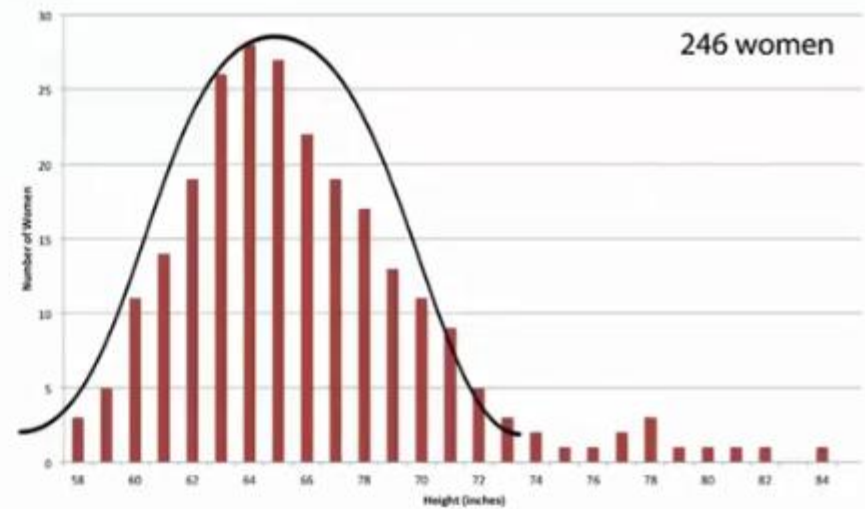
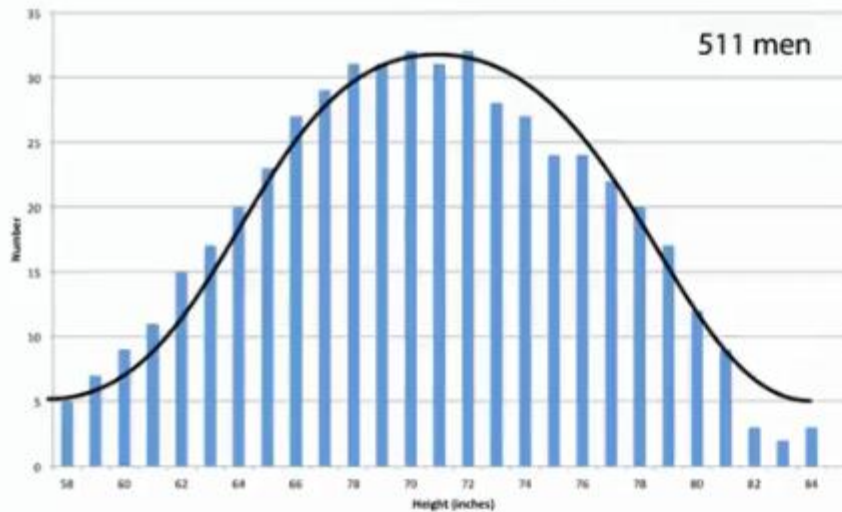
....



Normal Distribution

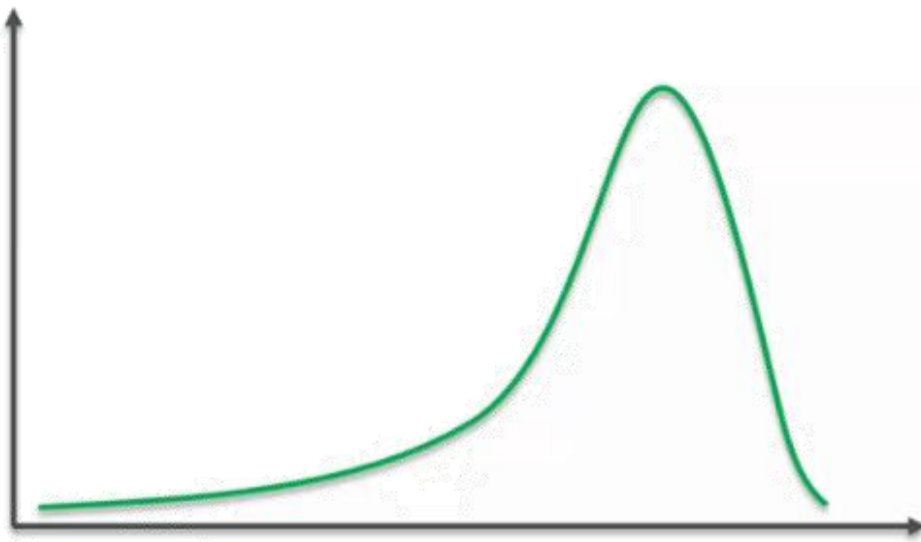


Height distribution of 20-yr old men and women in India



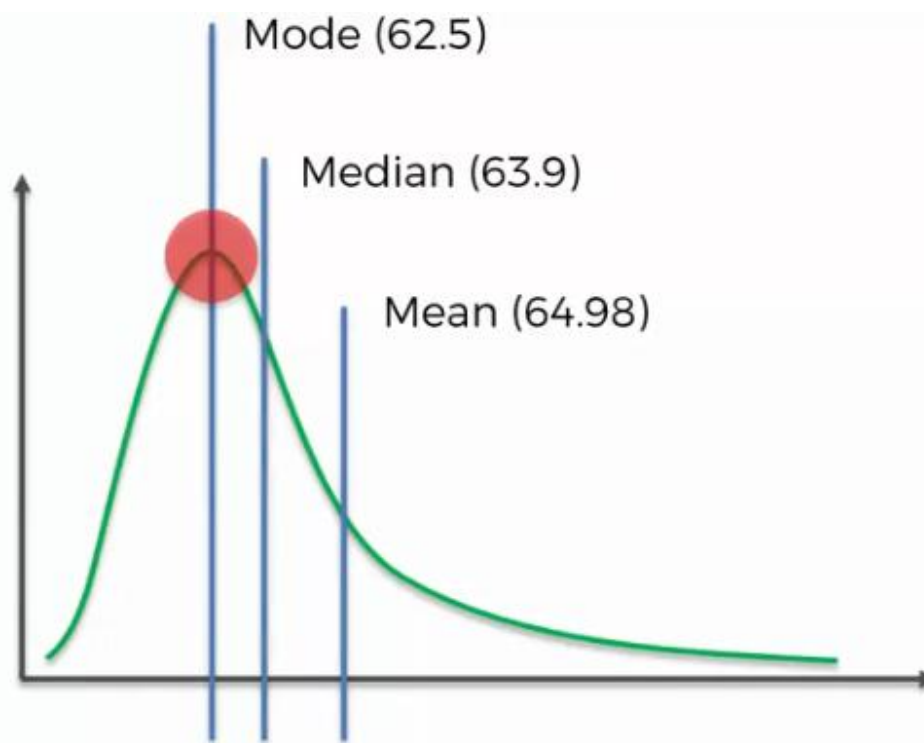
Skweness

Left (Negative) Skew



Right (Positive) Skew



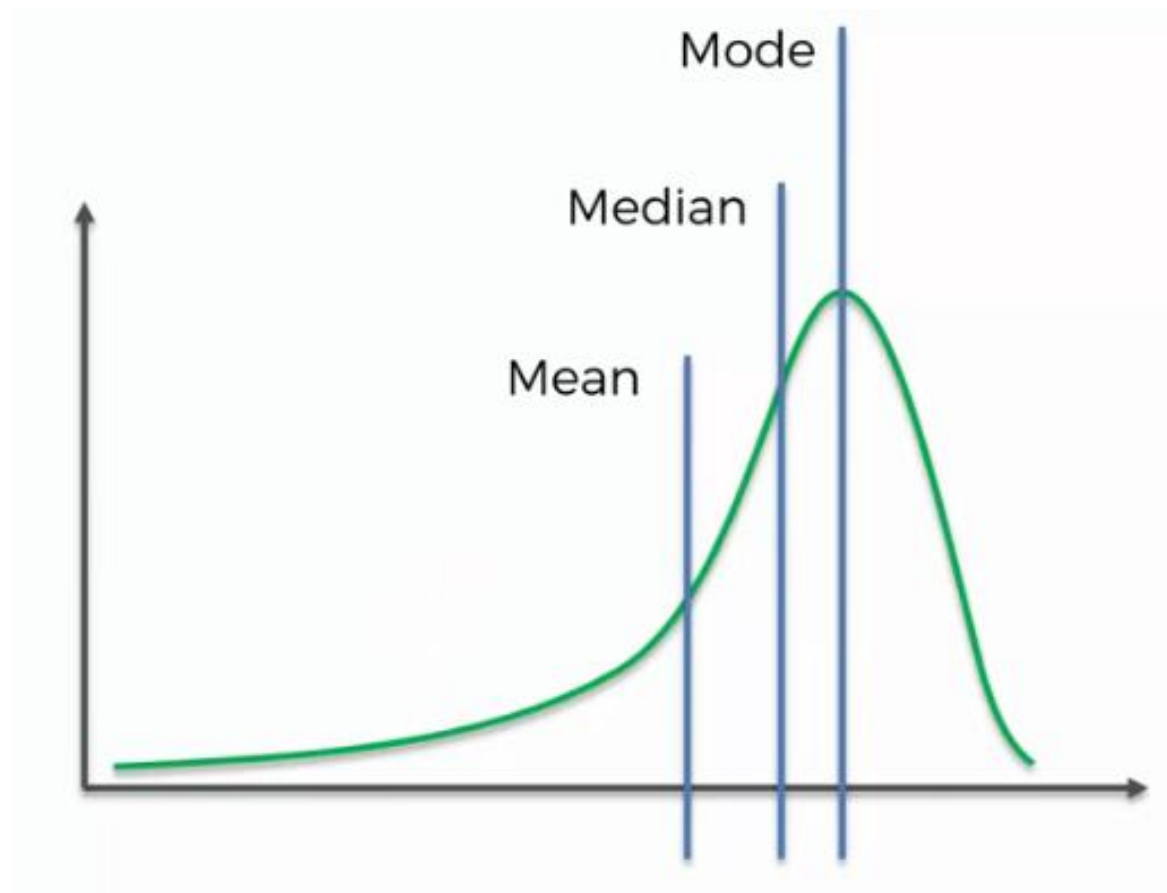


{ 58.8 59.9 61.2 61.3 61.5 62 62.5 62.5 62.5 63 63 63.7 63.9 64 64 65.1 65.2 66.7 67.8 68.3 69.9 70.7 71.8 72.2 73 }

{ 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 }



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()
```

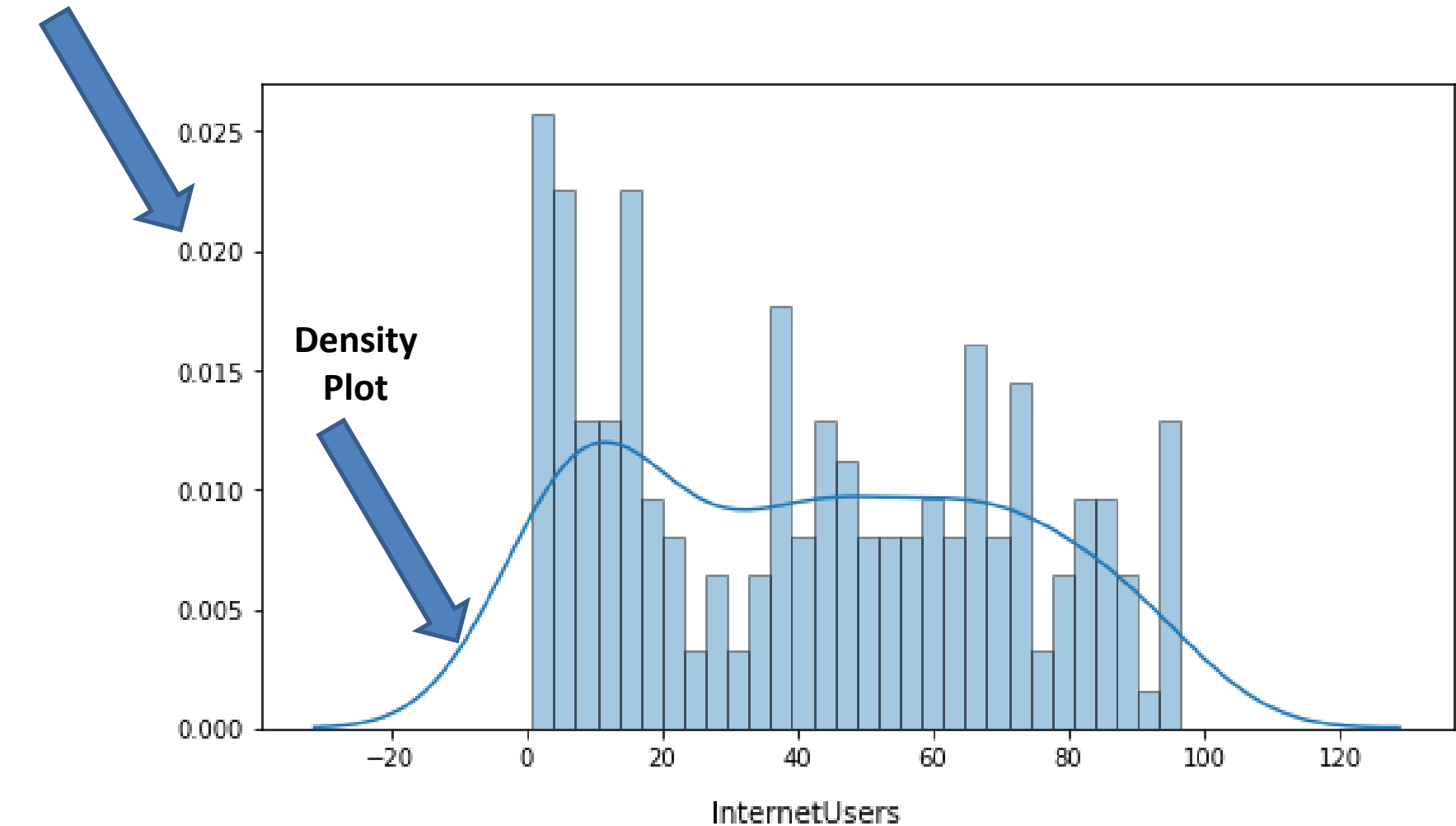

Creating a univariate distribution in seaborn with distplot()

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
plt.rcParams['figure.figsize']=8,4
plt.rcParams["patch.force_edgecolor"] = True
```

```
sns.distplot(DF.InternetAccess , bins=30)
```

```
plt.show()
```

Probability density : Probability per unit on the x-axis



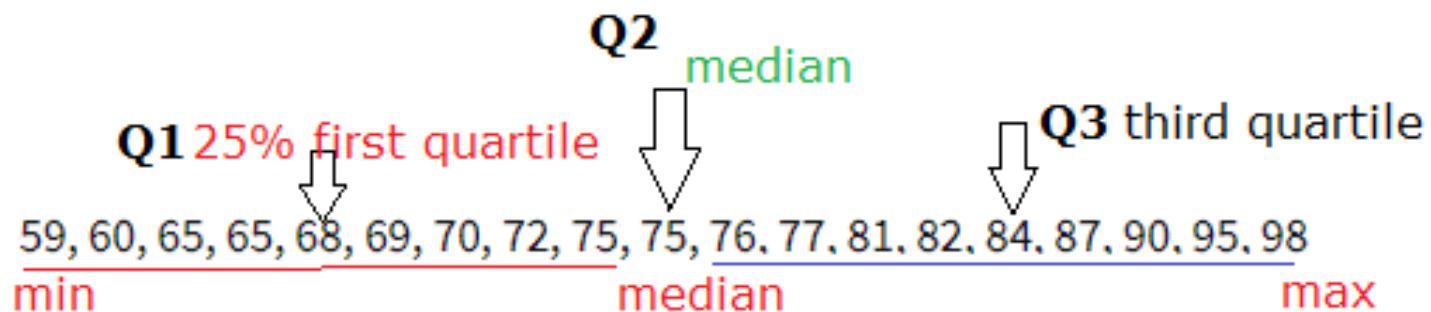
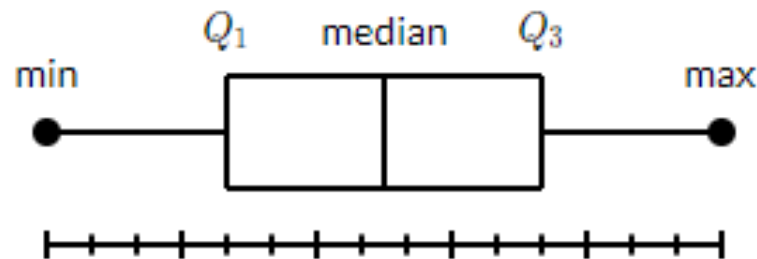
```
sns.distplot(df["Column"] ,  
             hist=True,  
             kde=True,  
             kde_kws = {'shade':True,  
                         'linewidth': 3,"color":"Red"},  
             hist_kws={'edgecolor':'black'},  
             bins=10)
```

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")
```

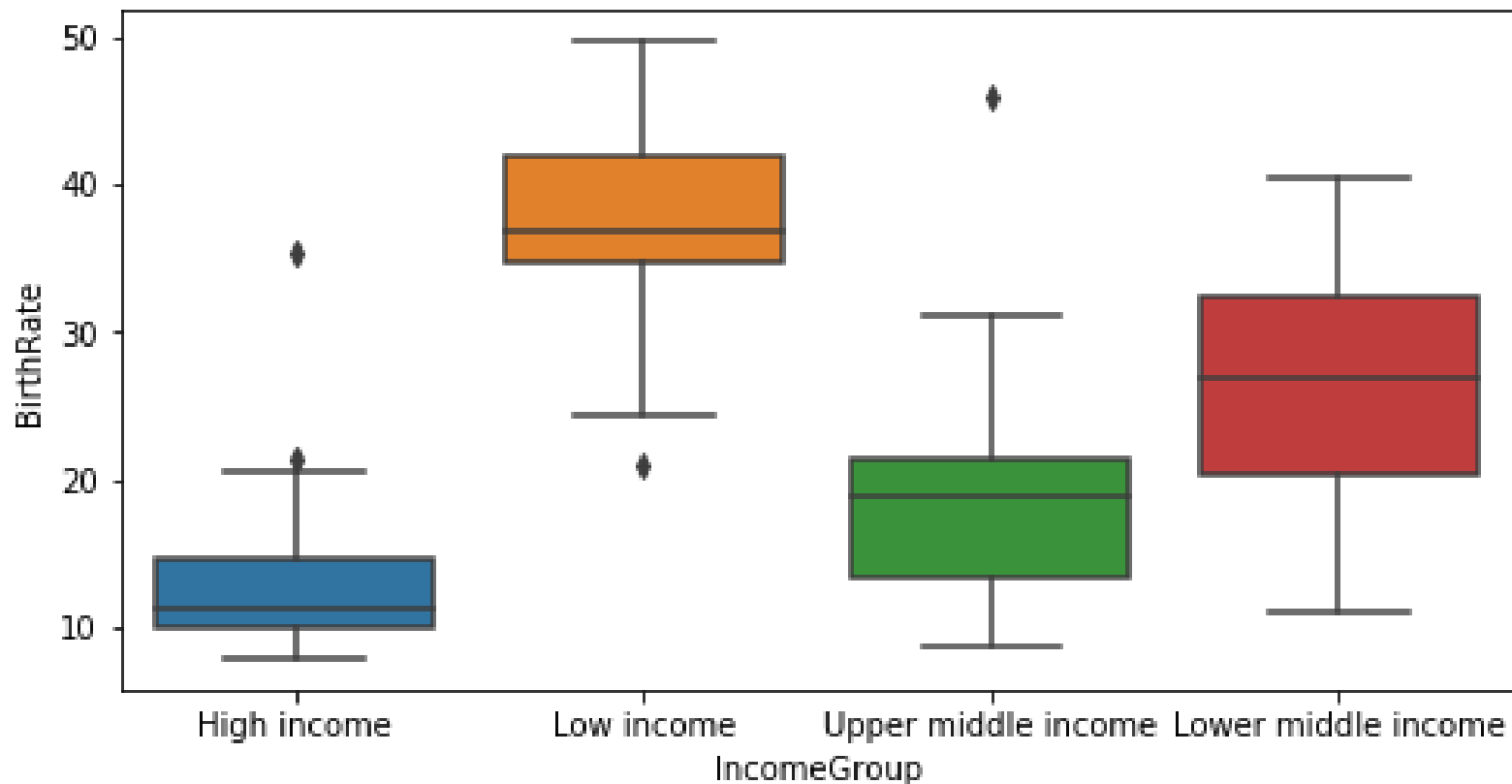
Quartiles



Total 19 values

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

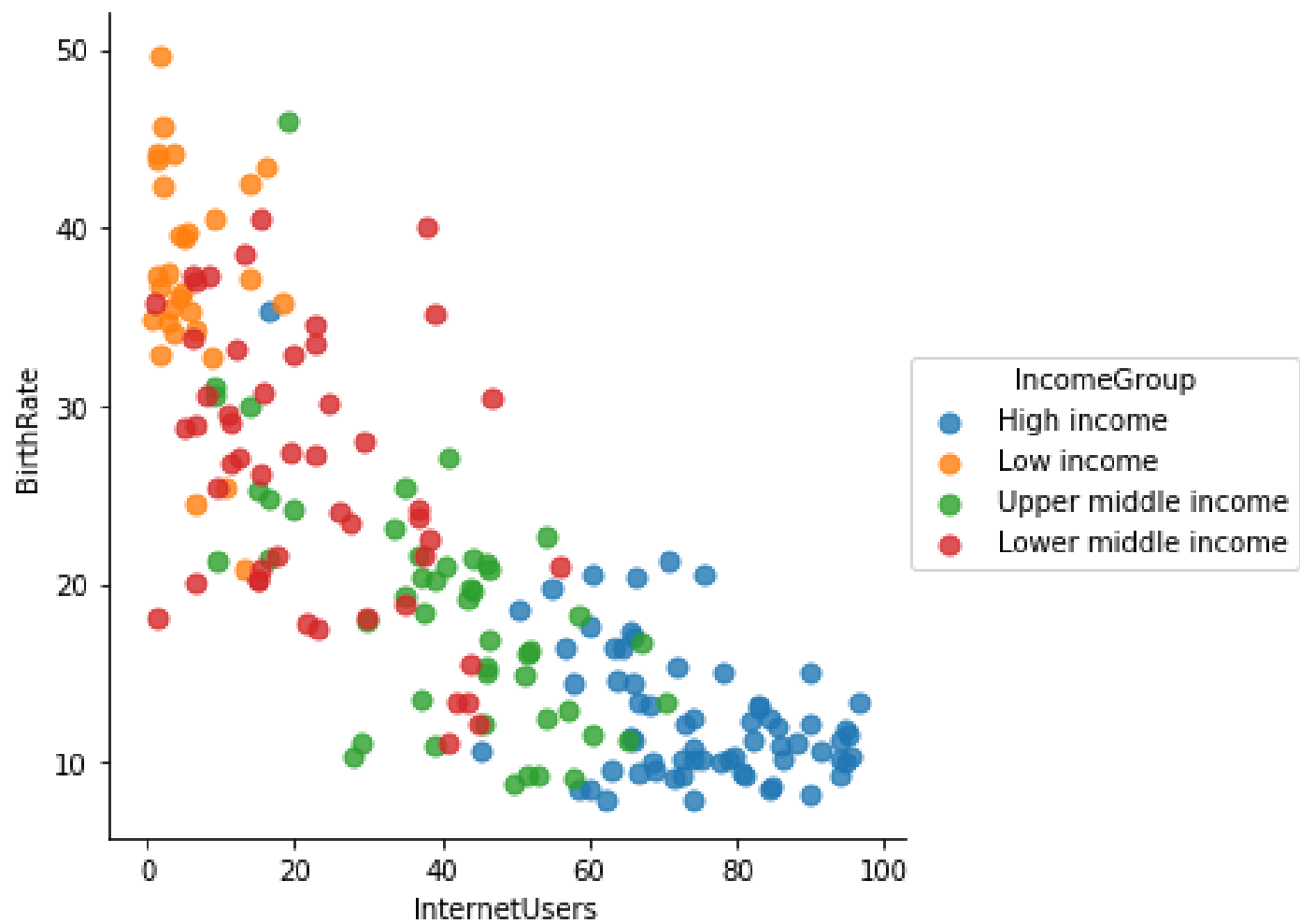
Task : Confirm the outliers and the plotted density values by using appropriate functions



Implot

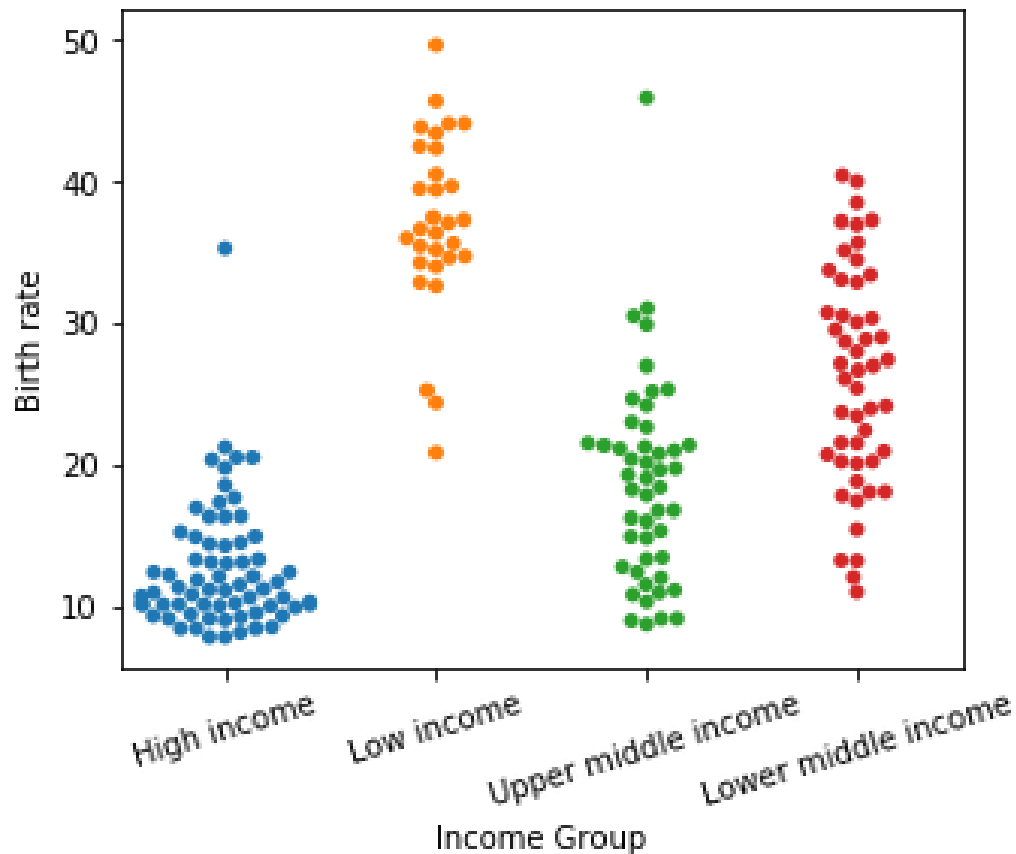
- **BirthRate Vs Internet Users**
- scatter_kws is a wrapper for plt.scatter (matplotlib.pyplot.scatter), so to size the markers we need to pass value to the scatter_kws as a dictionary(key:value) , where s is the size of the marker

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



swarmplot

```
s=sns.swarmplot(data=demog,  
x="Income Group", y="Birth rate")
```



Joint Plot

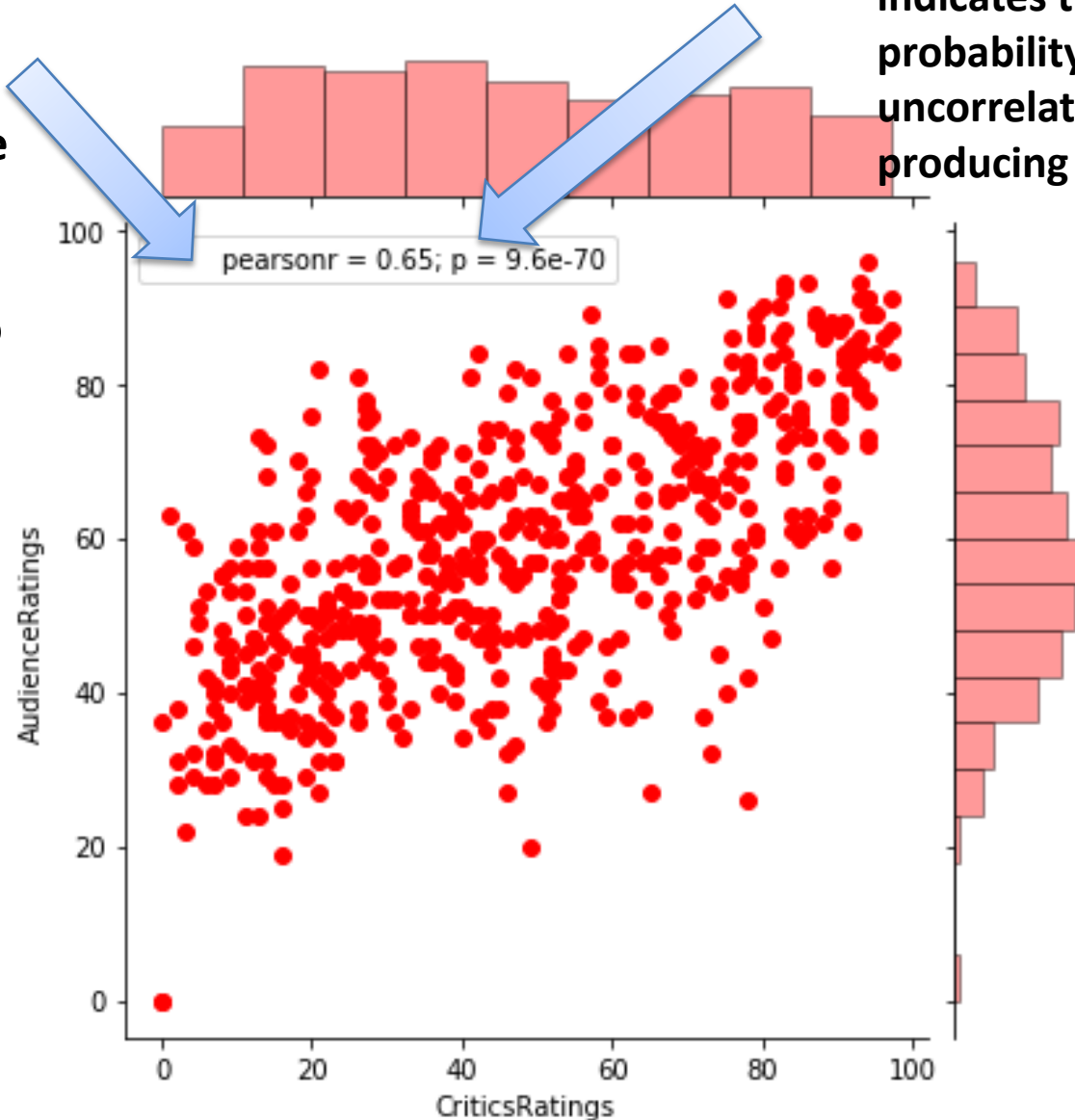
CriticsRatings Vs AudienceRatings

```
j = sns.jointplot(  
    data=movies,  
    x='CriticsRatings',y='AudienceRatings',  
    color='Red')
```

JointPlot

Pearsonr:
correlation
coefficient
measures the
linear
relationship
between two
datasets.

The p-value roughly
indicates the
probability of an
uncorrelated system
producing datasets

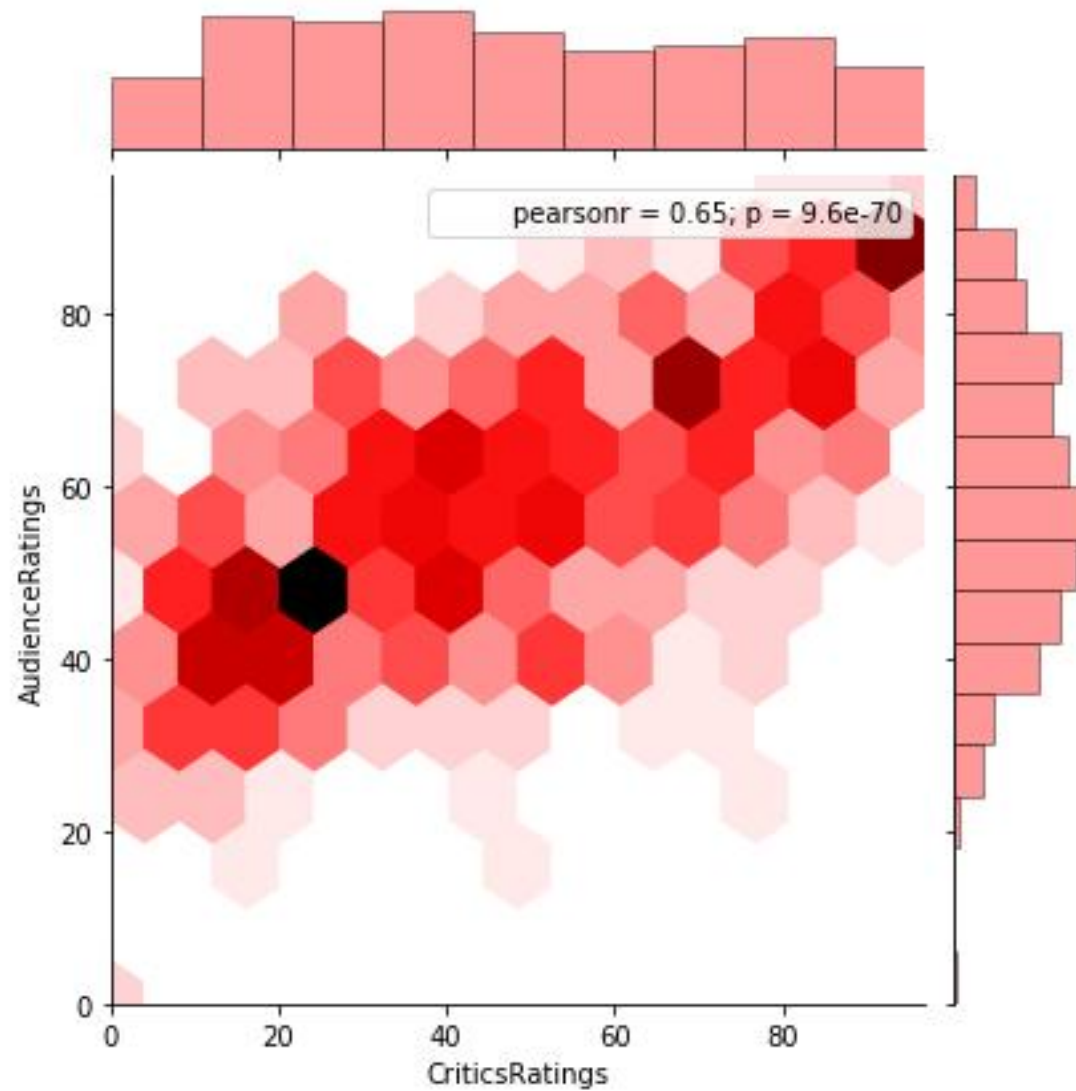


JointPlot

Joint plot for CriticsRatings Vs AudienceRatings

```
j = sns.jointplot(data=movies,  
                  x='CriticsRatings', y='AudienceRatings',  
                  color='Red',kind="hex")
```

JointPlot



Pearson Coefficient

- The Pearson correlation coefficient measures the linear relationship between two datasets
- Pearson's correlation requires that each dataset be normally distributed
- 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

p-value

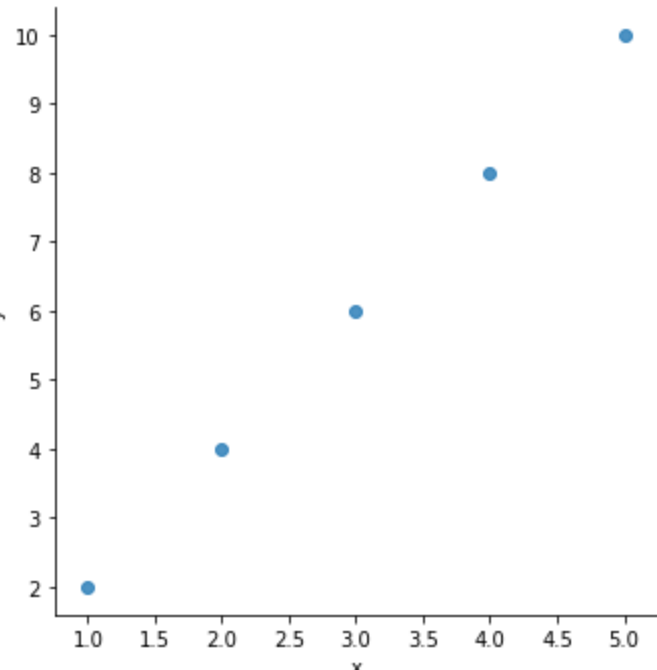
- The p-value roughly indicates the probability of an uncorrelated system
- The p-values are not entirely reliable but are probably reasonable for datasets larger than 500 or so
- p-value is measured with a significance level of 0.05
- p-value below 0.05 indicate correlation
- p-value above 0.05 indicate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

pearsonr

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

Result => (1.0, 0.0)

There is a **perfect linear rel**
x & y and the **probability c**
uncorrelated is 0.0



pearsonr

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

Result => (0.008211472, 0.98954494)

**No linear relationship bet
probability of dataset u**

