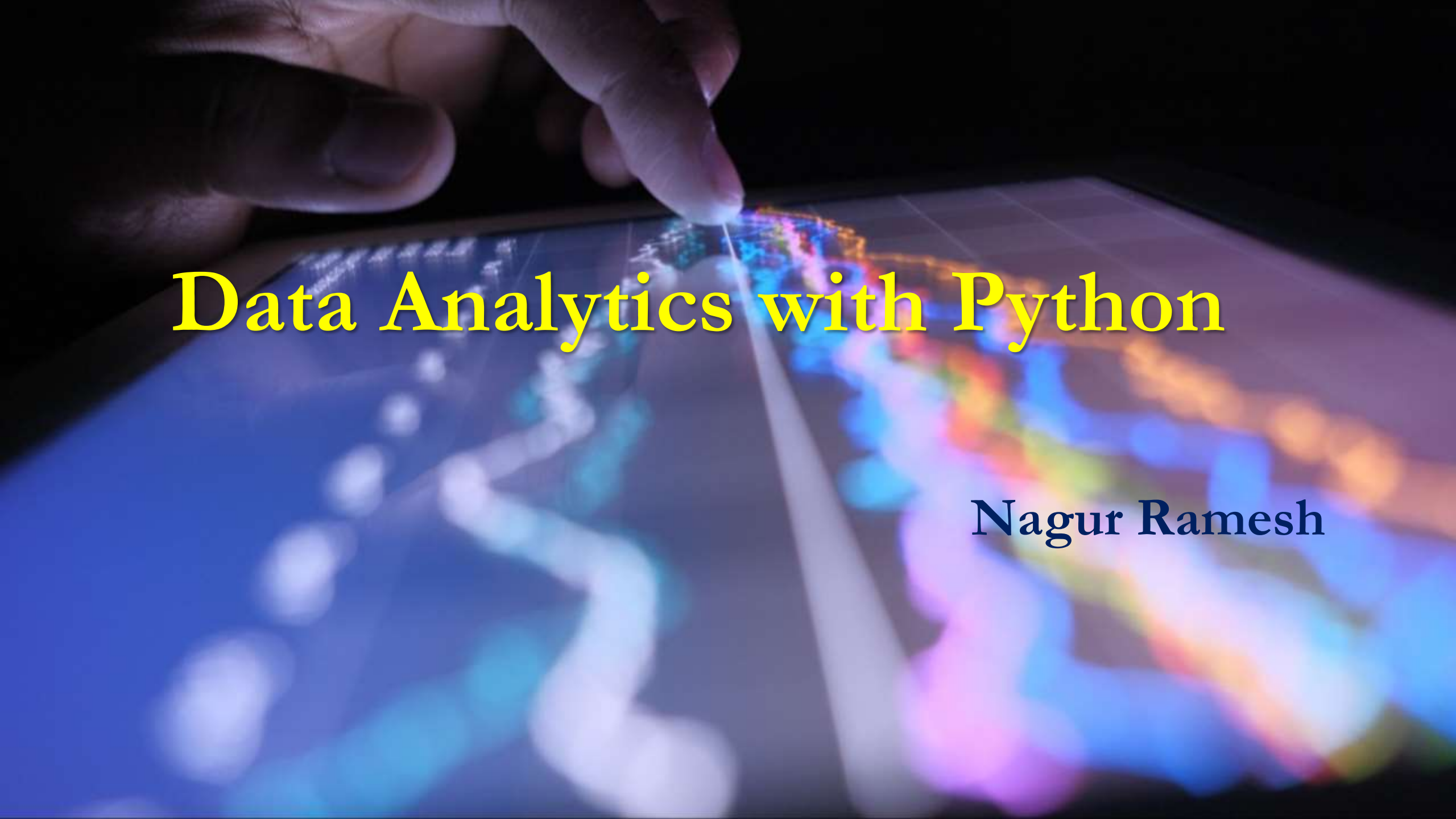


Data Analytics with Python

Nagur Ramesh

Prerequisites

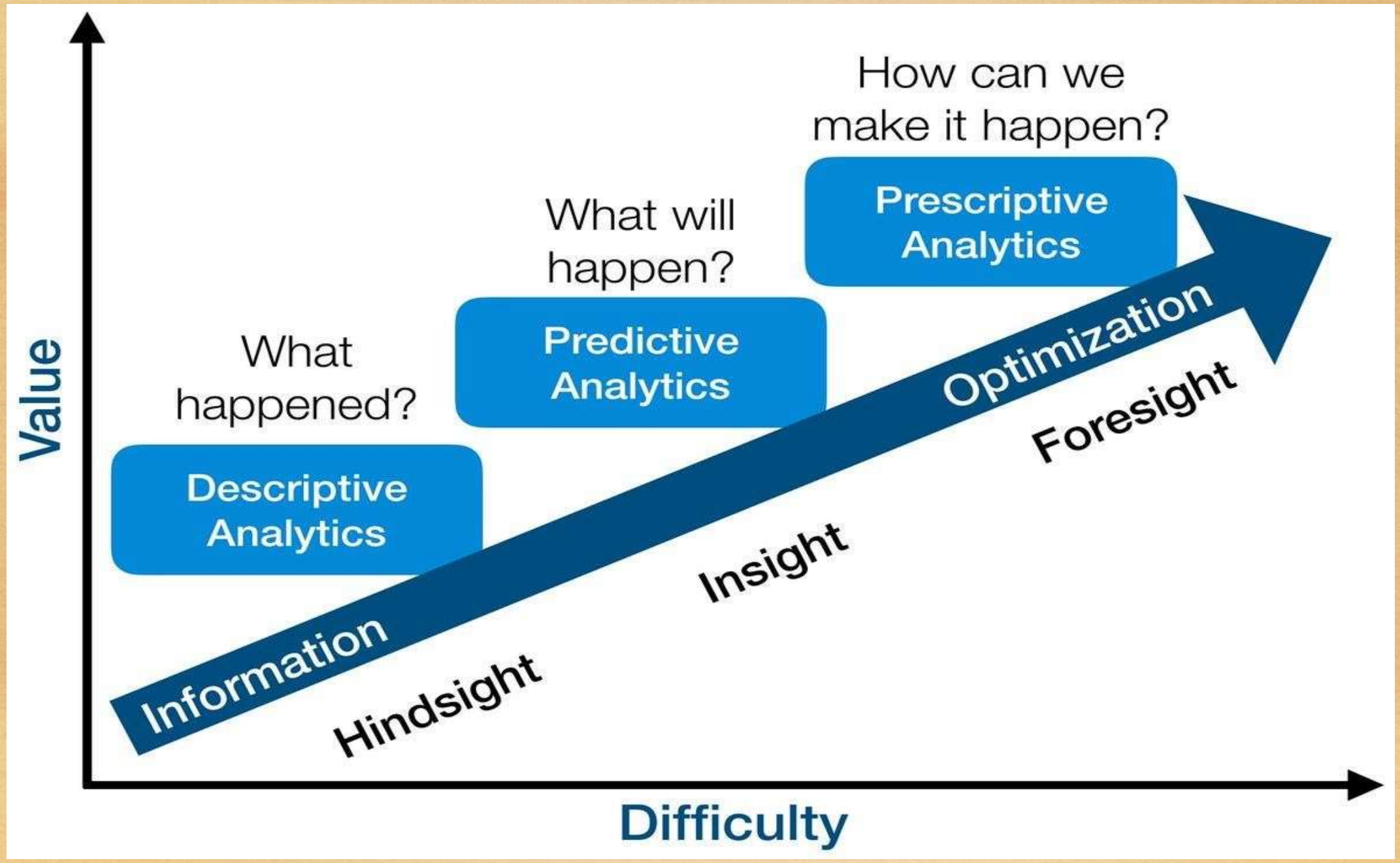
- **Python programming**
- **Basic mathematics**

Analytics ?

Types of Analytics ?

Descriptive **Predictive** Prescriptive

Descriptive	Predictive	Prescriptive
What has happened?	What could happen in the future?	What should a business do?


How can we make it happen?
Prescriptive Analytics
What will happen?
Predictive Analytics
What happened?
Descriptive Analytics
Optimization
Foresight
Insight
Information
Hindsight
Value
Difficulty

Basic Statistics

Synopsis

1. Statistics
2. Descriptive and Inferential Statistics
3. Data Types
4. Summarizing the Data
5. Frequency Distribution

1. Statistics

Statistics consists of principles and method for

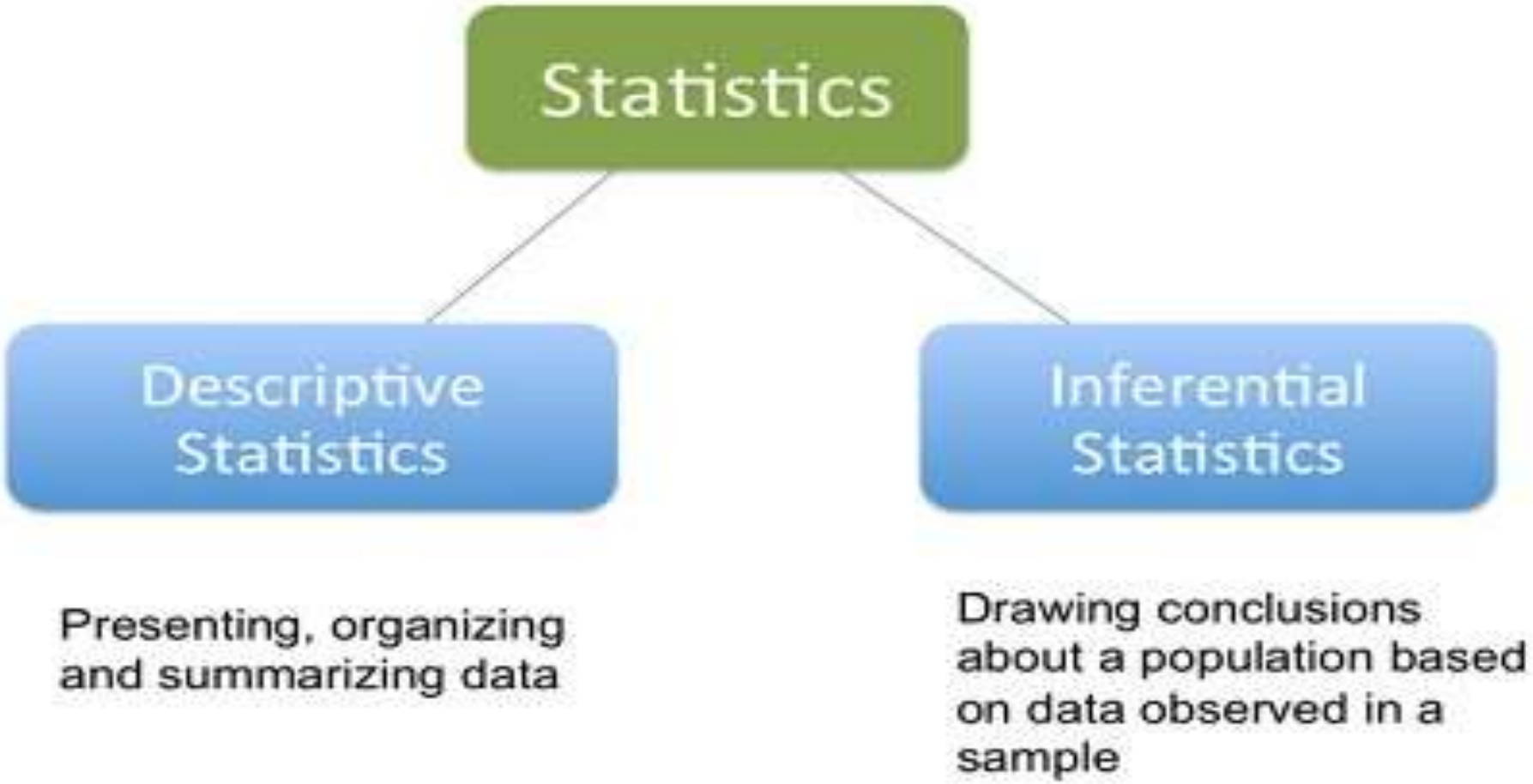
1. **Collecting** Data
2. **Analyzing** Data
3. **Interpreting/Explaining** the results
4. **Presentation** of Data

1. Statistics

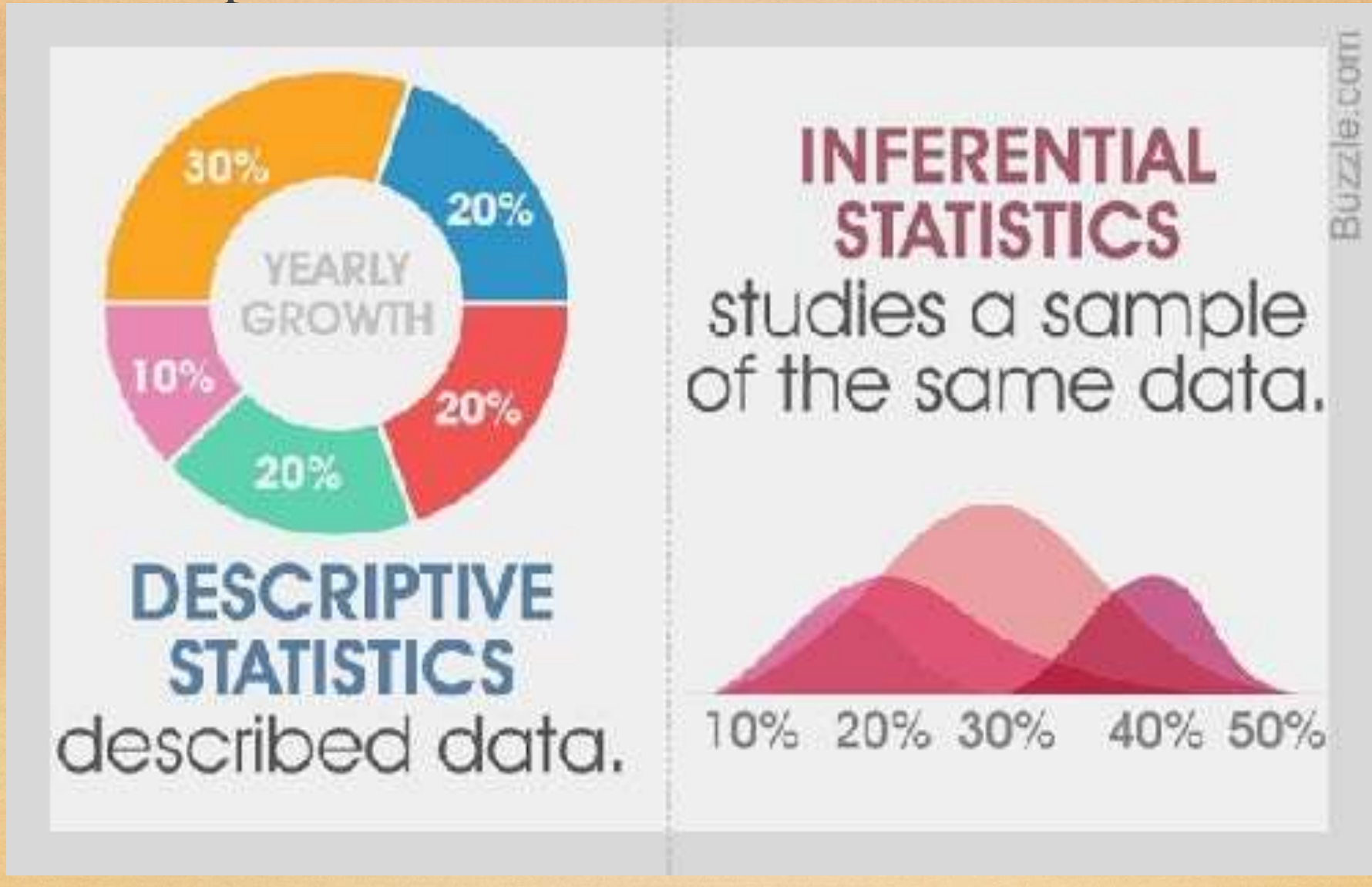
Statistics has been described as

1. Turning **Data** into **information**
2. Data Based **decision** making

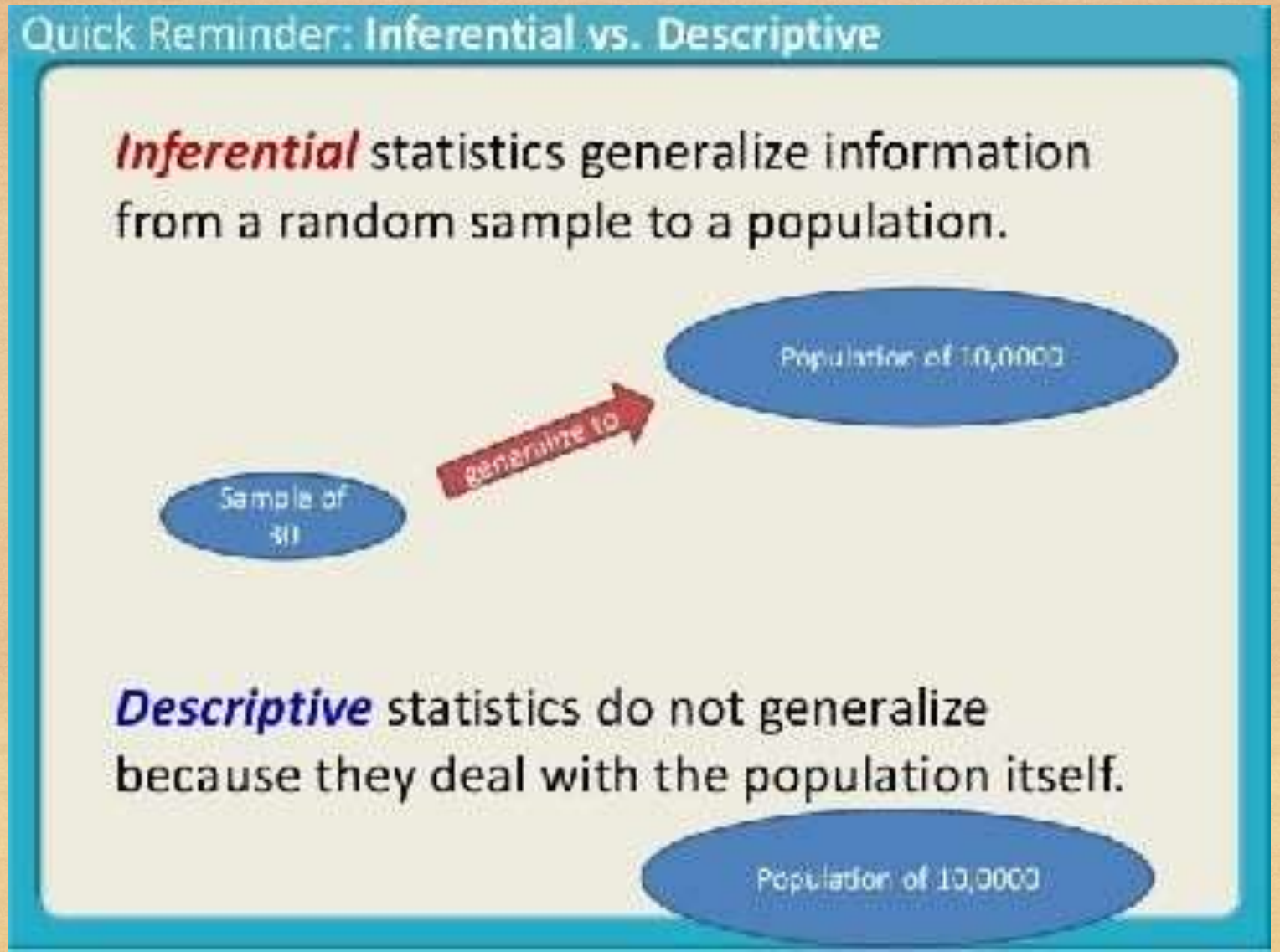
2. Descriptive and Inferential Statistics



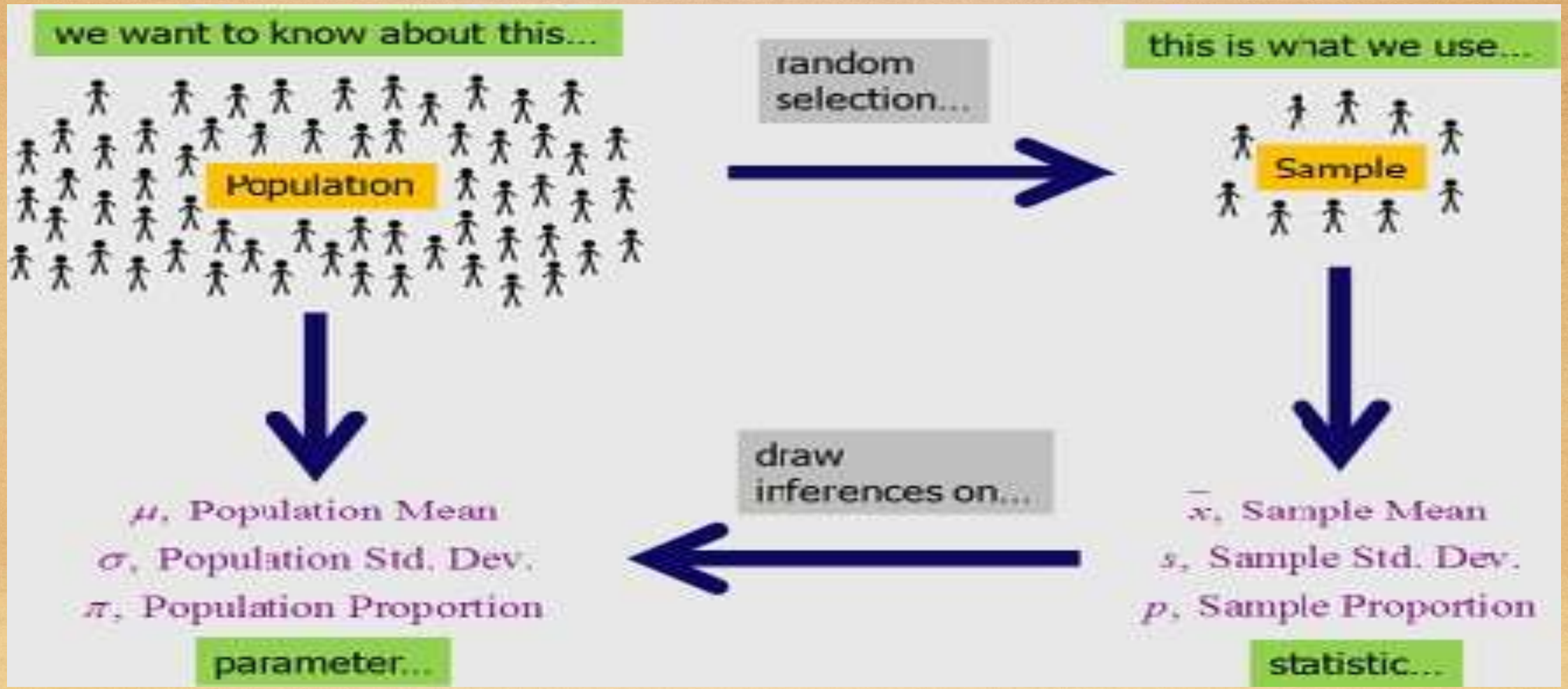
2. Descriptive and Inferential Statistics



2. Descriptive and Inferential Statistics



Population and Sample



Reference : http://www.dxbydt.com/the-size-of-your-sample/

Census and Survey

Census:

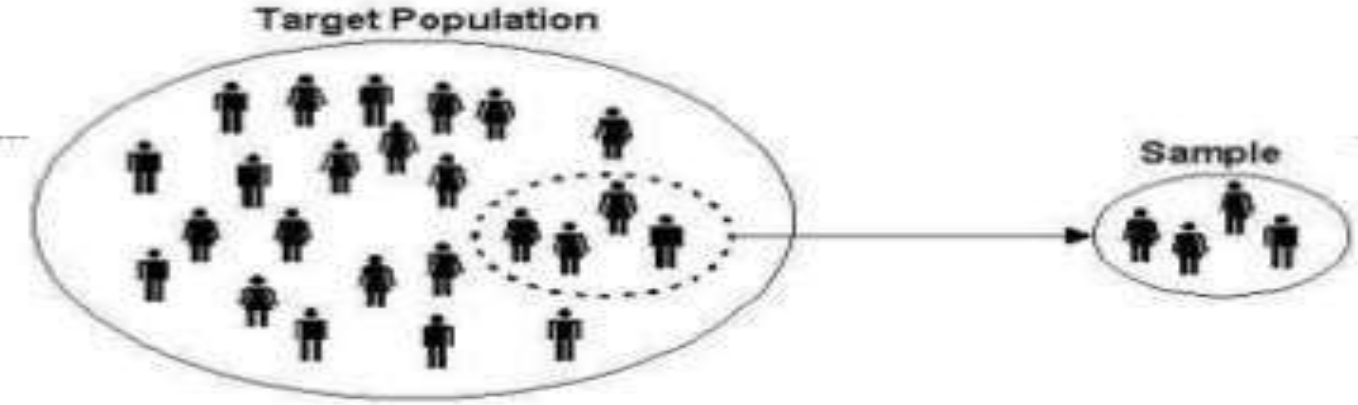
Gathering data from the whole population of interest. For example, elections, 10-year census, etc.

Survey:

Gathering data from the sample in order to make conclusions about the population.

For example, opinion polls, quality control checks in manufacturing units, etc.

WHEN DATA IS TO BE COLLECTED FROM EACH MEMBER OF THE POPULATION, IT IS KNOWN AS **CENSUS SURVEY**



WHEN DATA IS TO BE COLLECTED ONLY FROM SOME MEMBERS OF THE POPULATION, IT IS KNOWN AS **SAMPLE SURVEY**

References : https://www.slideshare.net/SunilChichra/introduction-28293803

Parameter and Statistic

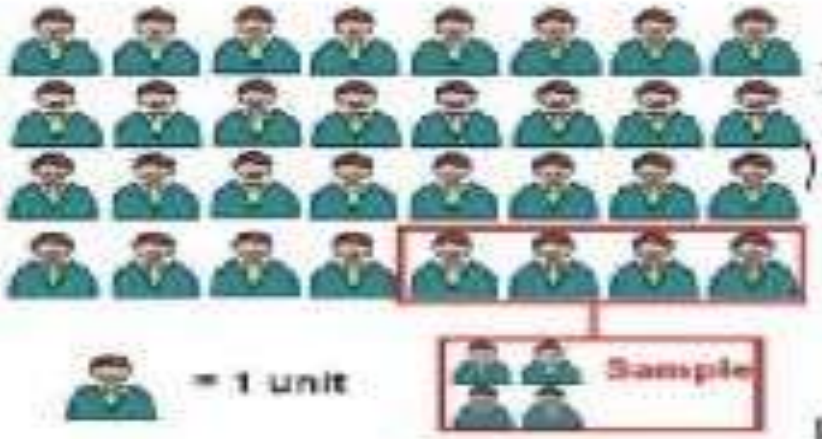
Parameter: A descriptive measure of the population.

For example, population mean, population variance, population standard deviation, etc.

Statistic: A descriptive measure of the sample.

For example, sample mean, sample variance, sample standard deviation, etc.

Population Vs. Sample



Population ← Sample

↓ ↓

Parameter ← Statistic

We measure the sample using statistics in order to draw inferences about the population and its parameters.

Reference : https://www.slideshare.net/drbhushankamble/sampling-and-its-variability

Identify Population Data or Sample Data?

- The US Government takes a census of its citizens every 10 years to gather information.

  a) Population b) Sample

- You want to know what sports teens prefer so you send out a survey to all the students in your high school.

  a) Population b) **Sample**

- You want data on the shoe size of all West students, so you interview every student at school.

  a) Population b) Sample

Identify as Parameter or Statistics?

- You want to know the mean income of the people who subscribe to People magazine, so you question 100 subscribers.

  a) Parameter b) **Statistic**

- You want to know the average height of the students in this math class, so you have everyone in the class write their height on a sheet of paper.

  a) Parameter b) Statistic

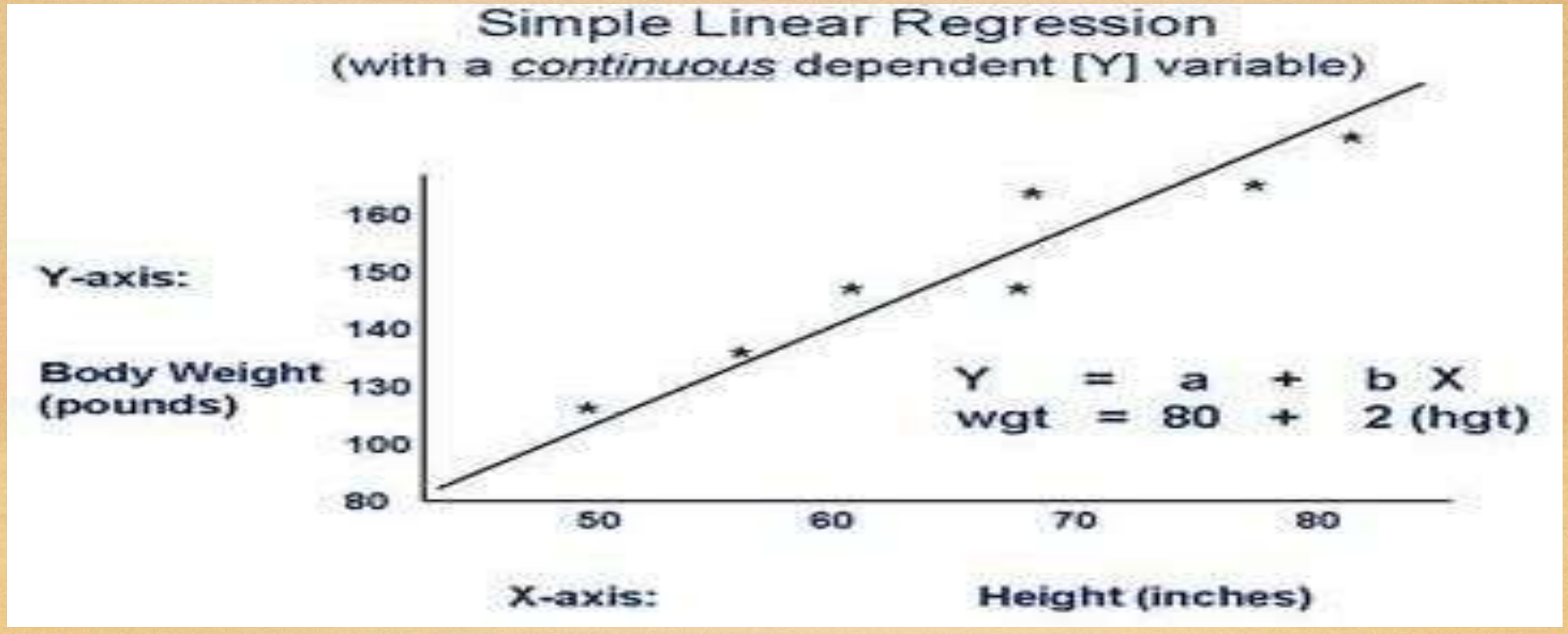
Parameter and Statistics

- Greek – Population Parameter
- Mean – μ
- Variance – σ^2
- Standard Deviation - σ

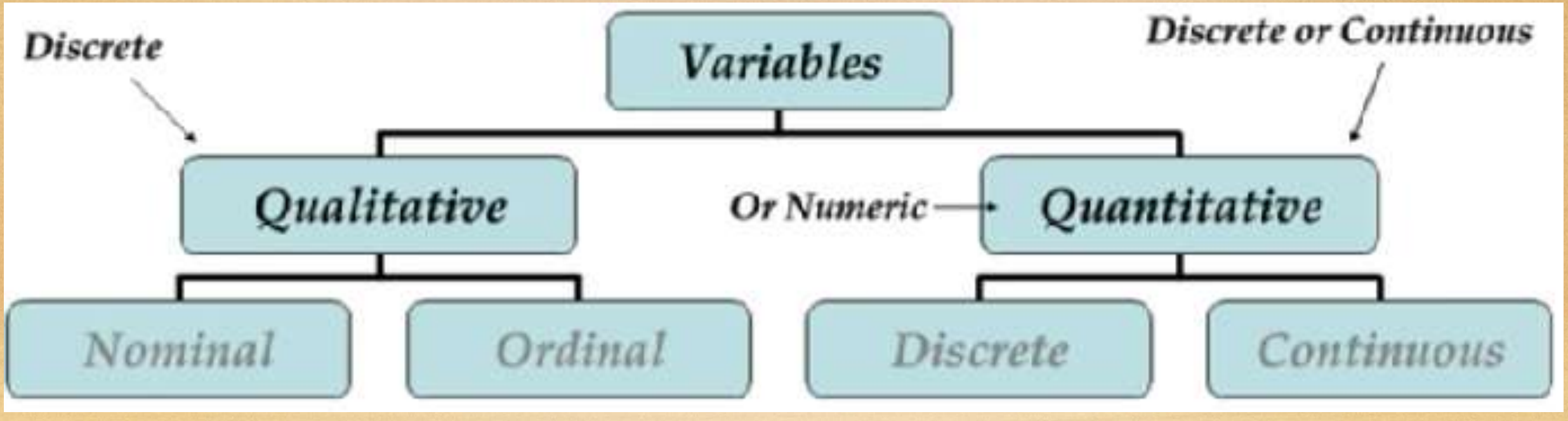
- Roman – Sample Statistic
- Mean – $\bar{x}$
- Variance – s^2
- Standard Deviation - s

Variables – Dependent and Independent

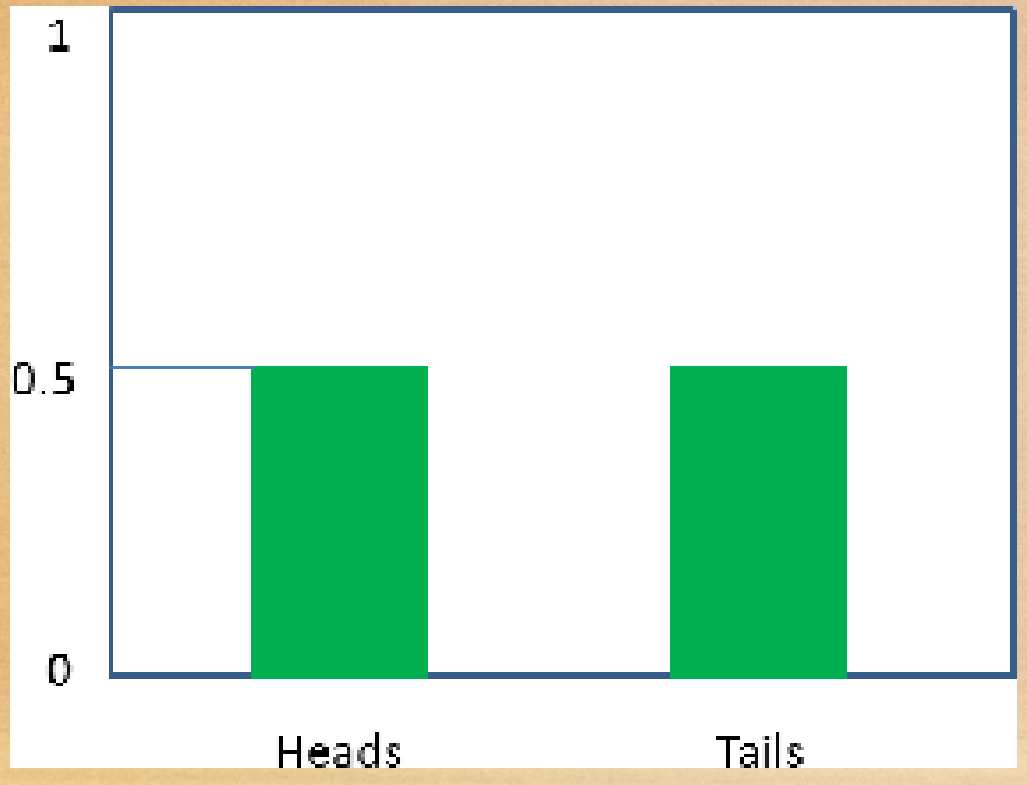
- Dependent variables on y-axis and Independent on x-axis.
- Dependent variable also called Target variable or Class variable.



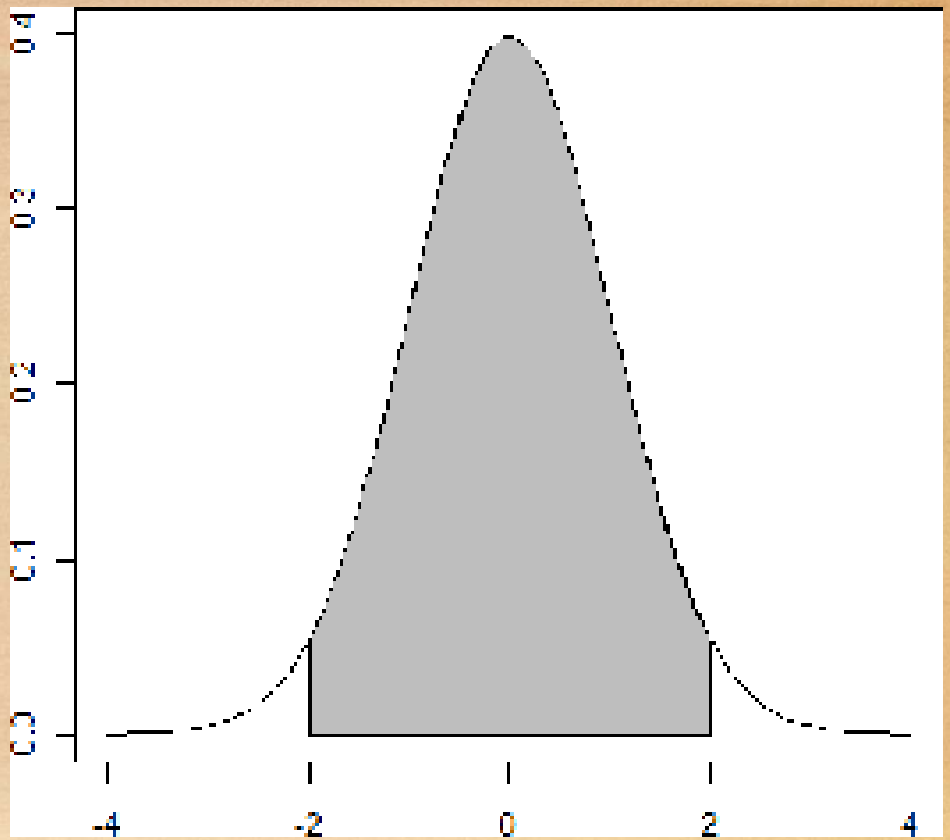
3. Data Types


Discrete
Variables
Discrete or Continuous
Qualitative
Or Numeric
Quantitative
Nominal
Ordinal
Discrete
Continuous

Discrete and Continuous



Countable



Measurable

Numerical or Categorical?

Age	Gender	Major	Units	Housing	GPA
18	Male	Psychology	16	Dorm	3.6
21	Male	Nursing	15	Parents	3.1
20	Female	Business	16	Apartment	2.8

- Numerical
  - Age
  - Units
  - GPA
- Categorical
  - Gender
  - Major
  - Housing

Discrete or Continuous?

- Time between customer arrivals at a retail outlet
  Continuous
- Sampling 100 voters in an exit poll and determining how many voted for the winning candidate

  Discrete
- Lengths of newly designed automobiles -

  Continuous
- No. of customers arriving at a retail outlet during a five- minute period

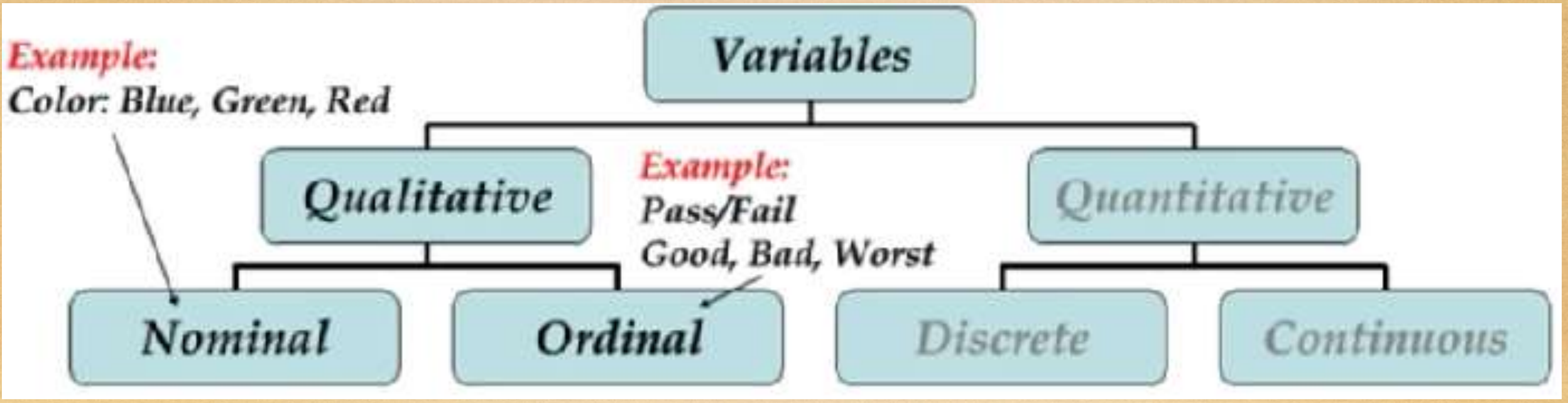
  Discrete
- No. of defects in a batch of 50 items

  Discrete

Numeric Data (Quantitative) Examples

- Height
- Weight
- Time
- Volume
- Number of iPads sold
- Number of complaints received at the call centre
- Number of employees
- Percentage return on a stock
- Rupee change in stock price

3. Data Types


Example:
Color: Blue, Green, Red
Variables
Qualitative
Example:
Pass/Fail
Good, Bad, Worst
Quantitative
Nominal
Ordinal
Discrete
Continuous

Categorical Data (Qualitative)

- Nominal Examples
- Employee ID
- Gender
- Religion
- Ethnicity
- Pin codes
- Place of birth
- Aadhaar numbers

Ordinal

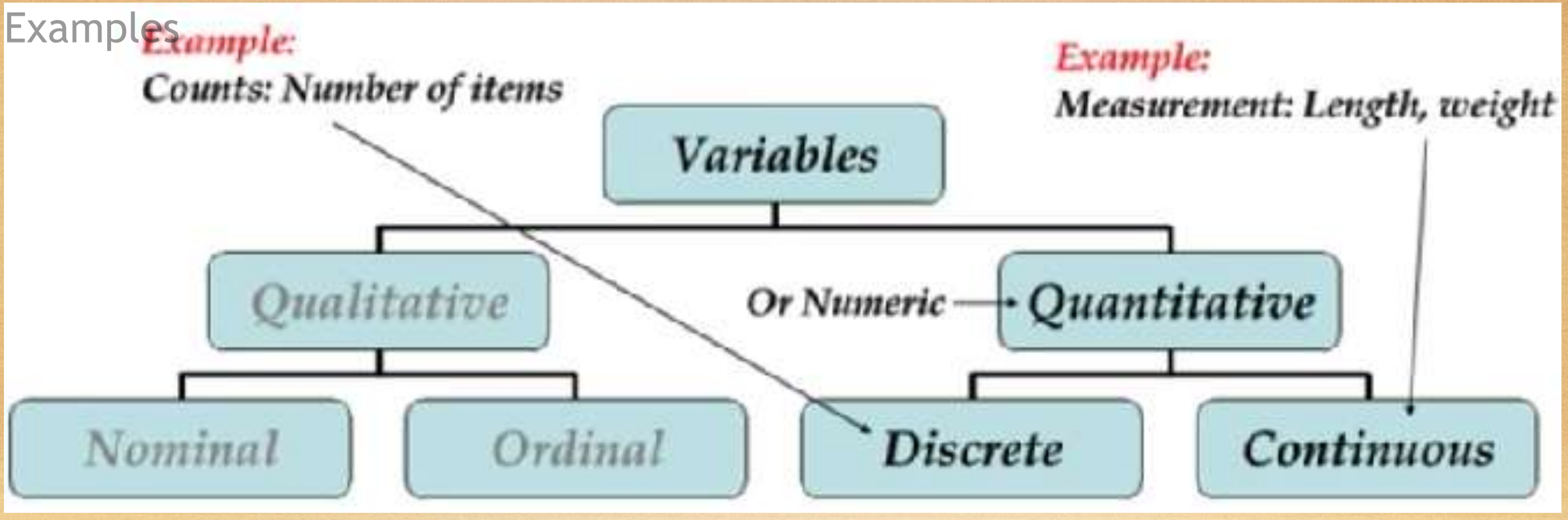
Examples

•Mutual fund risk ratings
Fortune 50 rankings

- Movie ratings
- While there is an order, difference between consecutive levels are not always equal.

3. Data Types

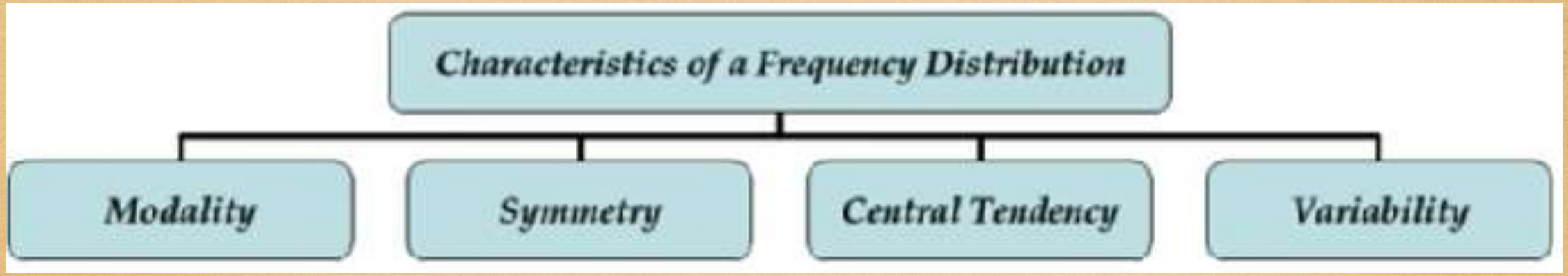
Examples



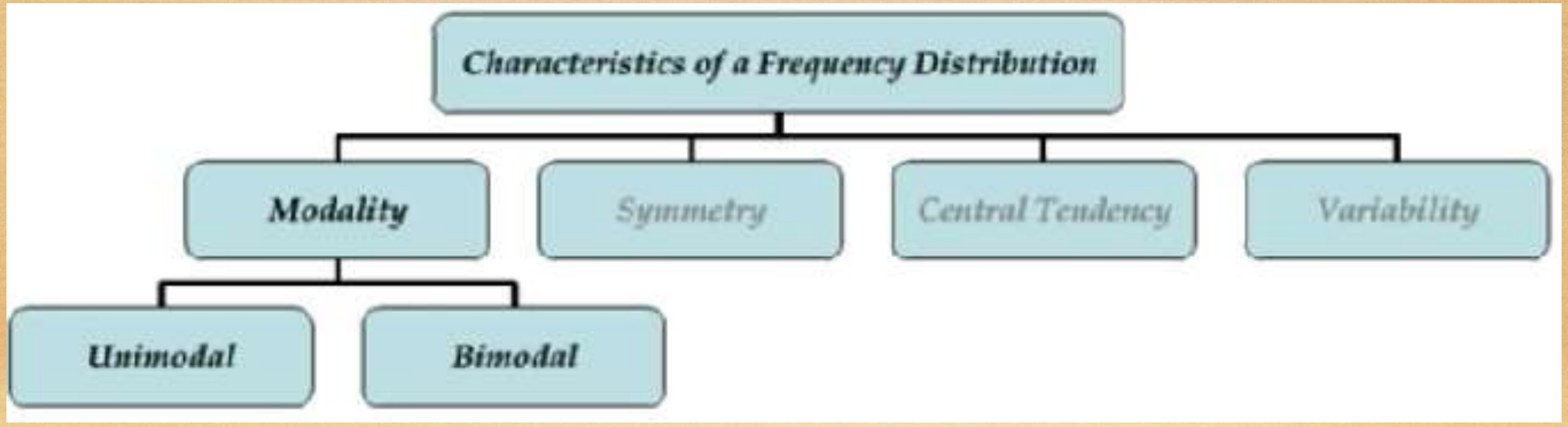
3. Summarizing the Data

1. Frequency Distribution
2. Bar Chart
3. Histograms

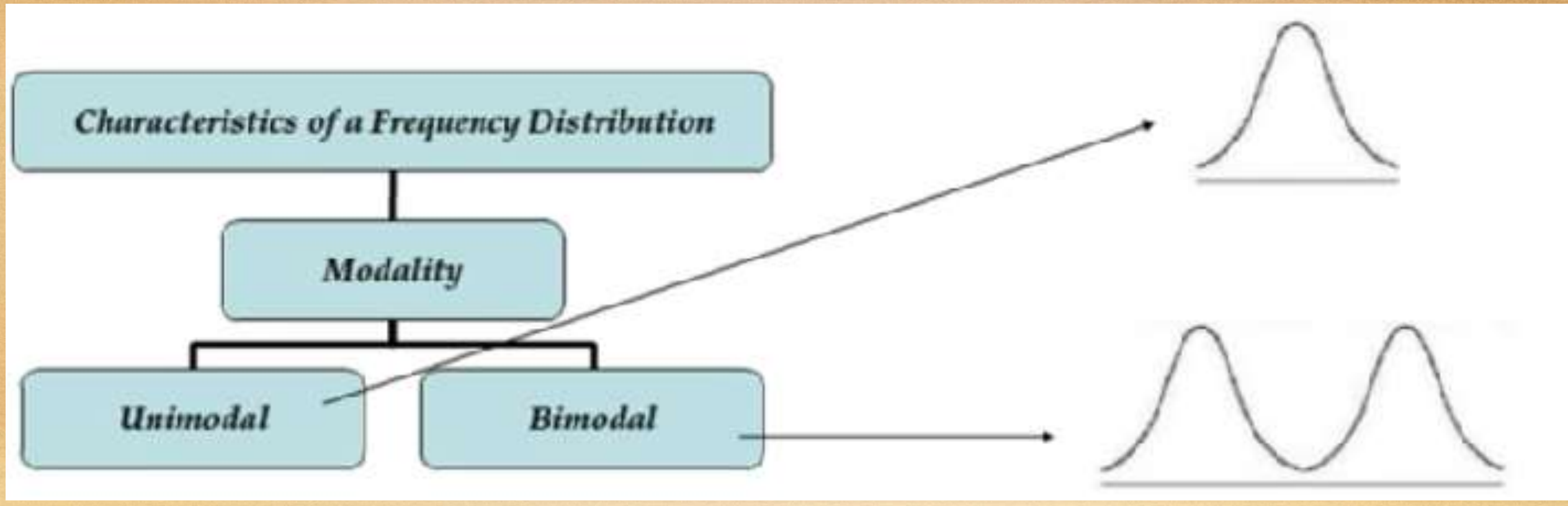
3. Summarizing the Data



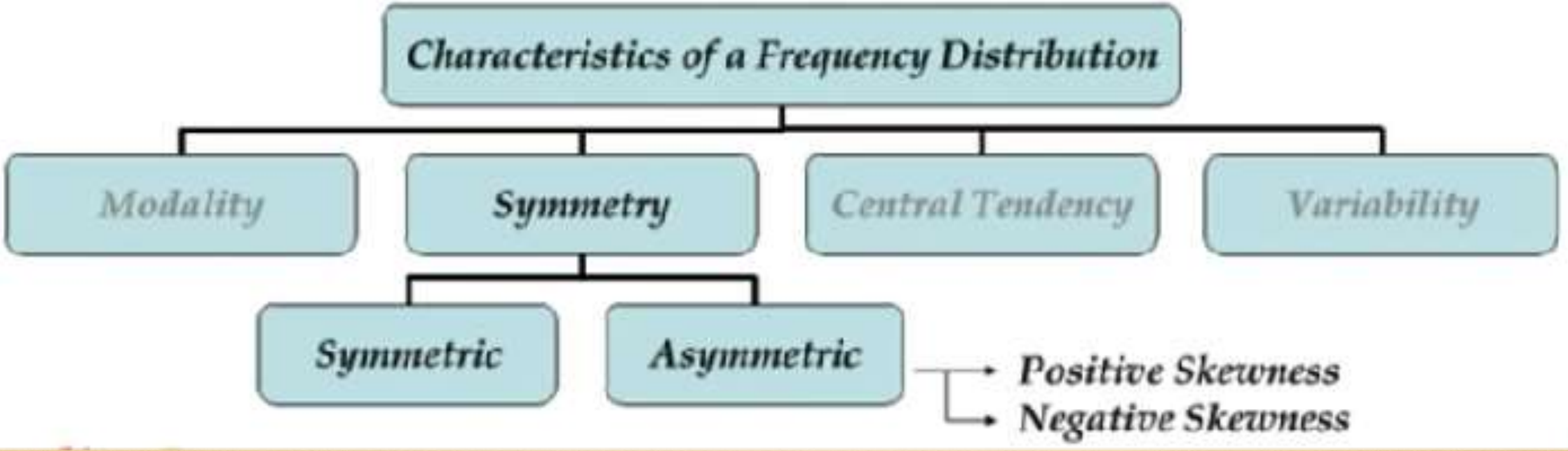
3. Summarizing the Data



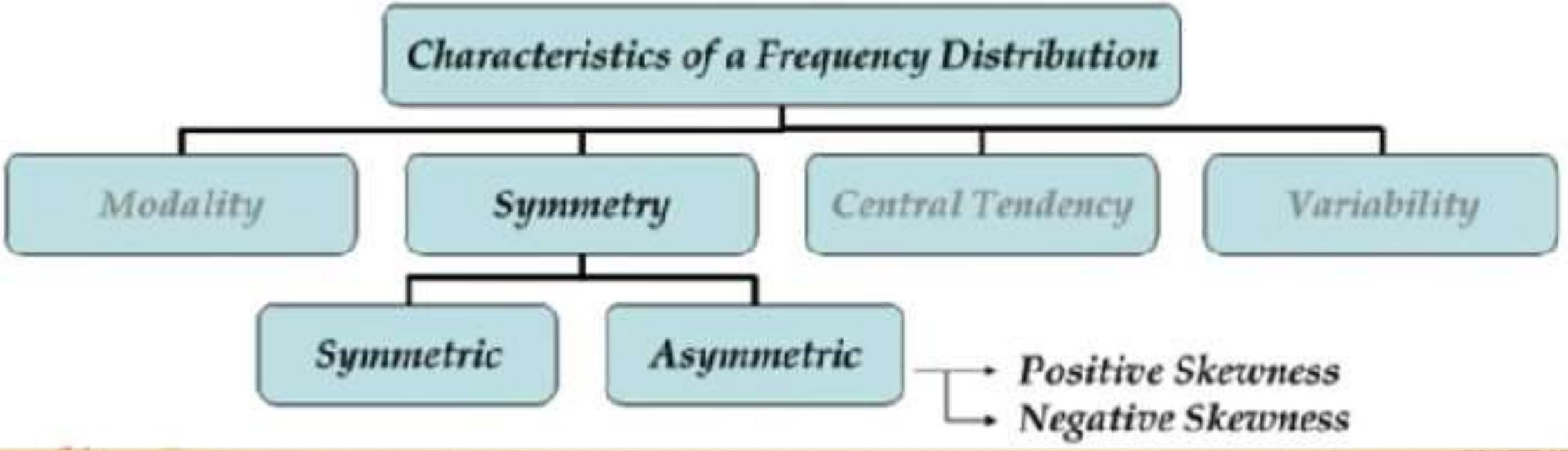
3. Summarizing the Data



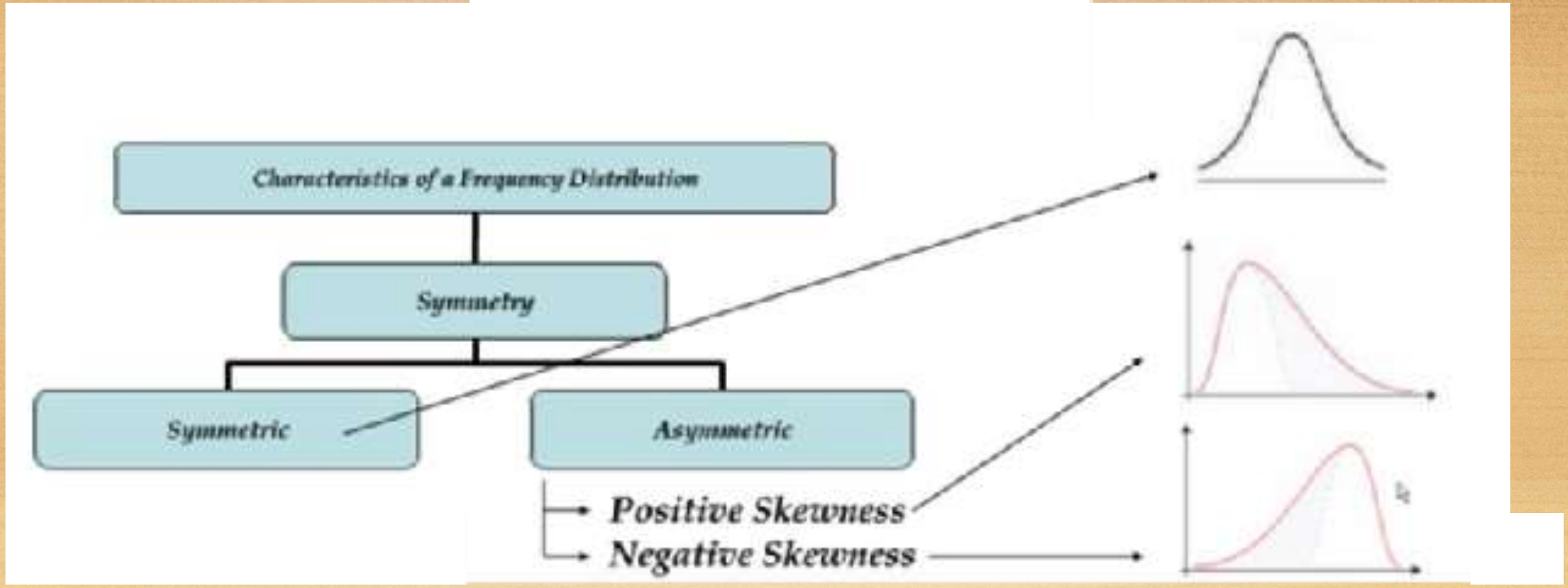
3. Summarizing the Data



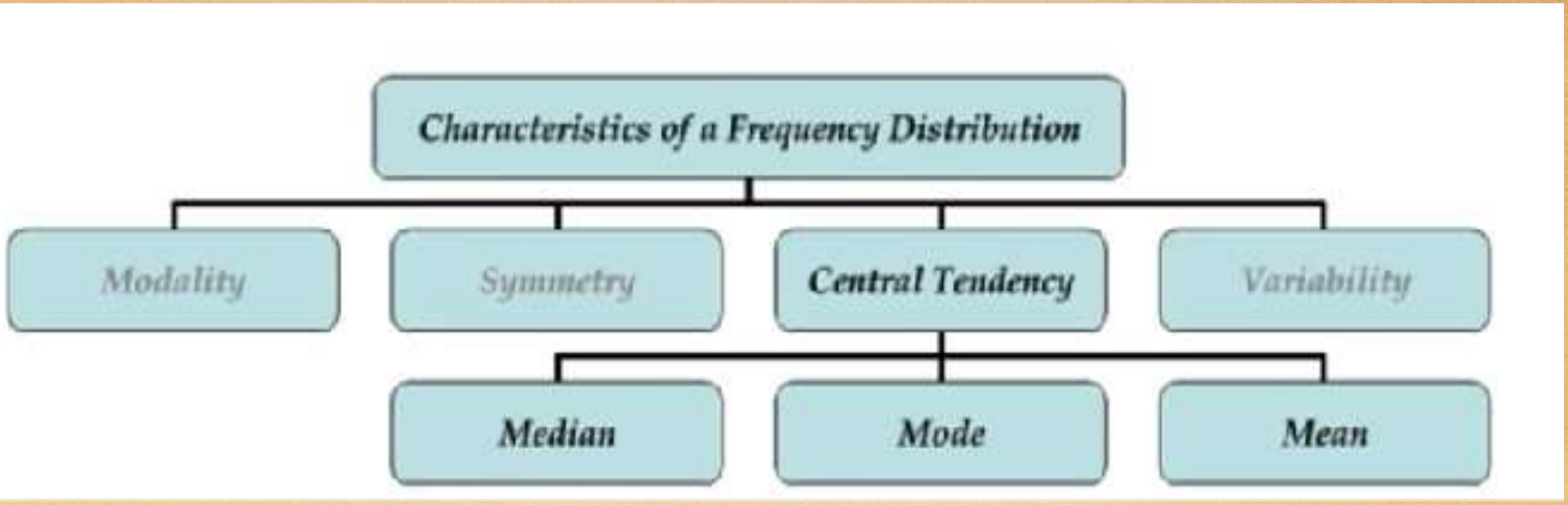
3. Summarizing the Data



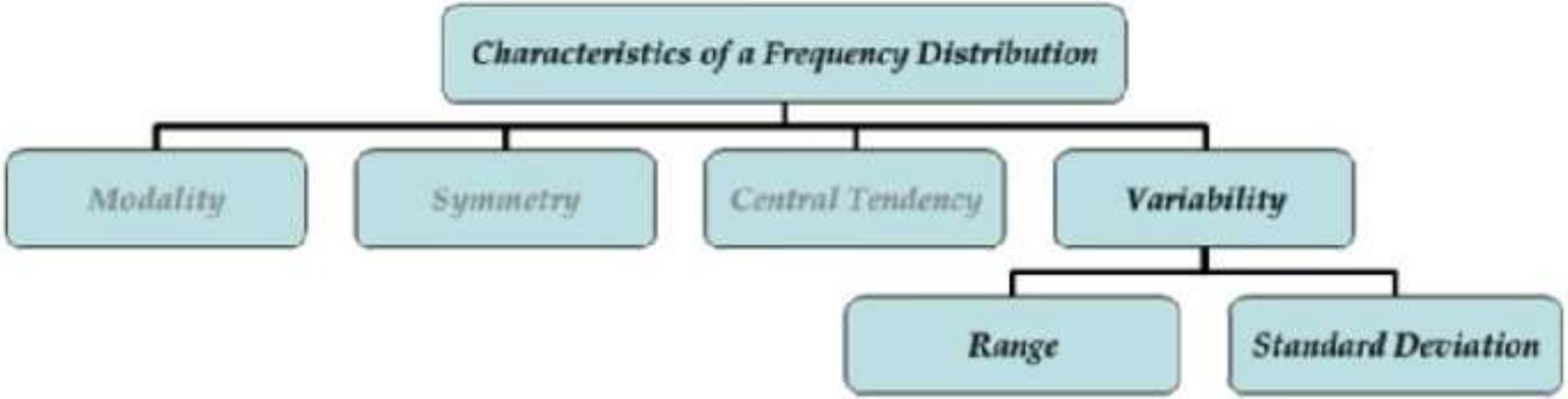
3. Summarizing the Data



3. Summarizing the Data



3. Summarizing the Data



Match	Player A	Player B
1	40	40
2	40	35
3	7	45
4	40	52
5	0	30
6	90	40
7	3	29
8	11	43
9	120	37

Match	Player A	Player B
1	40	40
2	40	35
3	7	45
4	40	52
5	0	30
6	90	40
7	3	29
8	11	43
9	120	37
SUM	351	351

Match	Player A	Player B
1	40	40
2	40	35
3	7	45
4	40	52
5	0	30
6	90	40
7	3	29
8	11	43
9	120	37
SUM	351	351
MEAN	39	39

Match	Player A	Player B
1	40	40
2	40	35
3	7	45
4	40	52
5	0	30
6	90	40
7	3	29
8	11	43
9	120	37
SUM	351	351
MEAN	39	39
MEDIAN	40	40

Match	Player A	Player B
1	40	40
2	40	35
3	7	45
4	40	52
5	0	30
6	90	40
7	3	29
8	11	43
9	120	37
SUM	351	351
MEAN	39	39
MEDIAN	40	40
RANGE	120	23

Standard Deviation

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{n}}$$

$$s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$$

S.No	x	x^2	x-mean	Abs(x-mean)	(x-mean)^2	
1	5	25	0.3333333333	0.3333333333	0.1111111111	
2	7	49	2.333333333	2.333333333	5.444444444	
3	4	16	-0.6666666667	0.6666666667	0.4444444444	
4	2	4	-2.666666667	2.666666667	7.111111111	
5	6	36	1.333333333	1.333333333	1.777777778	
6	2	4	-2.666666667	2.666666667	7.111111111	
7	8	64	3.333333333	3.333333333	11.11111111	
8	5	25	0.3333333333	0.3333333333	0.1111111111	
9	3	9	-1.666666667	1.666666667	2.777777778	
SUM	42	232	0	15.33333333		36
Average	4.666666667	25.77777778				

Match	Player A	Player B
1	40	40
2	40	35
3	7	45
4	40	52
5	0	30
6	90	40
7	3	29
8	11	43
9	120	37
SUM	351	351
MEAN	39	39
MEDIAN	40	40
STANDARD DEVIATION	41.5180683558376	7.28010988928052

Basketball coach Statson is in a dilemma choosing between 3 players all having the same average scores.

Points scored per game	7	8	9	10	11	12	13
Frequency, f	1	1	2	2	2	1	1

Points scored per game	7	9	10	11	13
Frequency, f	1	2	4	2	1

Points scored per game	3	6	7	10	11	13	30
Frequency, f	2	1	2	3	1	1	1

Basketball coach Statson is in a dilemma choosing between 3 players all having the same average scores.

Points scored per game	7	8	9	10	11	12	13
Frequency, f	1	1	2	2	2	1	1

Points scored per game	7	9	10	11	13
Frequency, f	1	2	4	2	1

Points scored per game	3	6	7	10	11	13	30
Frequency, f	2	1	2	3	1	1	1

MEAN = ? MEDIAN = ? MODE = ?

Basketball coach Statson is in a dilemma choosing between 3 players all having the same average scores.

Points scored per game	7	8	9	10	11	12	13
Frequency, f	1	1	2	2	2	1	1

Points scored per game	7	9	10	11	13
Frequency, f	1	2	4	2	1

Points scored per game	3	6	7	10	11	13	30
Frequency, f	2	1	2	3	1	1	1

MEAN = MEDIAN = MODE = 10

Basketball coach Statson is in a dilemma choosing between 3 players all having the same average scores.

Points scored per game	7	8	9	10	11	12	13
Frequency, f	1	1	2	2	2	1	1

Points scored per game	7	9	10	11	13
Frequency, f	1	2	4	2	1

Points scored per game	3	6	7	10	11	13	30
Frequency, f	2	1	2	3	1	1	1

MEAN = MEDIAN = MODE = 10 RANGE = 5 , 5 , 27

Basketball coach Statson is in a dilemma choosing between 3 players all having the same average scores.

Points scored per game	7	8	9	10	11	12	13
Frequency, f	1	1	2	2	2	1	1

Points scored per game	7	9	10	11	13
Frequency, f	1	2	4	2	1

Points scored per game	3	6	7	10	11	13	30
Frequency, f	2	1	2	3	1	1	1

MEAN = MEDIAN = MODE = 10 RANGE = 5 , 5 , 27 Reject Player 3

Basketball coach Statson is in a dilemma choosing between 3 players all having the same average scores.

Points scored per game	7	8	9	10	11	12	13
Frequency, f	1	1	2	2	2	1	1

Points scored per game	7	9	10	11	13
Frequency, f	1	2	4	2	1

STANDARD DEVIATION

Player 1 = **1.7873008824606**
Player 2 = **3.30823887354653**

What is your Decision?????????

CONSISTENCY INDEX: TOP EIGHT RUN-SCORERS

Player	For	Ci	Sd	Ave	M	I	No	Runs
Ricky Ponting	Aus	**1.05**	**59.47**	**56.88**	128	215	26	10750
Sunil Gavaskar	Ind	**1.06**	**54.42**	**51.12**	125	214	16	10122
Allan Border	Aus	**1.08**	**54.45**	**50.37**	156	265	44	11174
Rahul Dravid	Ind	**1.17**	**61.07**	**52.28**	131	227	26	10509
Sachin Tendulkar	Ind	**1.18**	**64.28**	**54.28**	156	256	27	12429
Jacques Kallis	Sa	**1.19**	**64.91**	**54.58**	128	216	33	9988
Brian Lara	Wi	**1.24**	**65.33**	**52.89**	131	232	6	11953
Steve Waugh	Aus	**1.26**	**64.16**	**51.06**	168	260	46	10927

Data Preprocessing

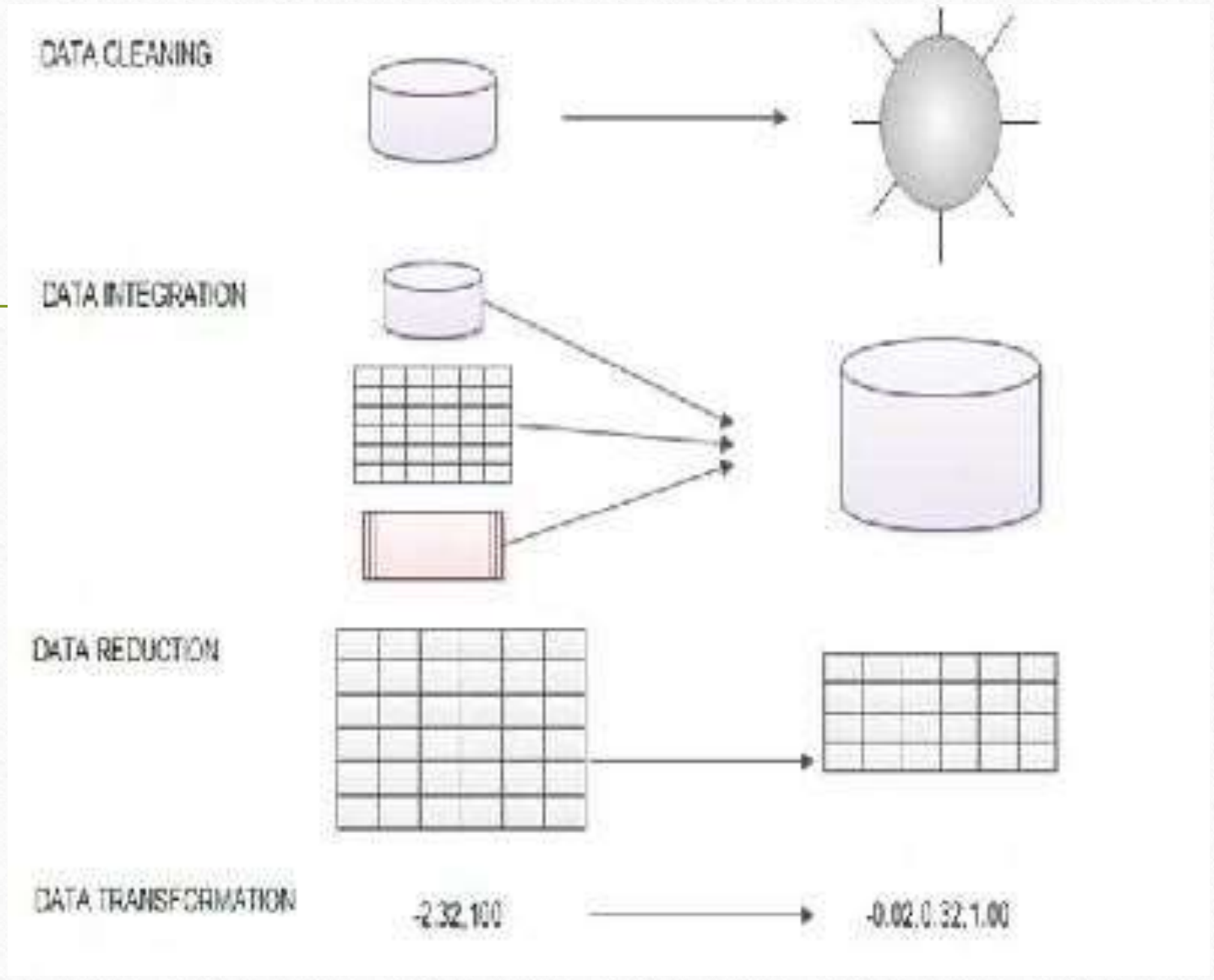
Real World Data

Do you see Any Problem?

S.No	Credit_rating	Age	Income	Credit_cards
1	0.00	21	10000	y
2	1.0		2500	n
3	2.0	62	-500	y
4	100.012	42		n
5	yes	200	1	y
6	30	0	Seventy thousand	No

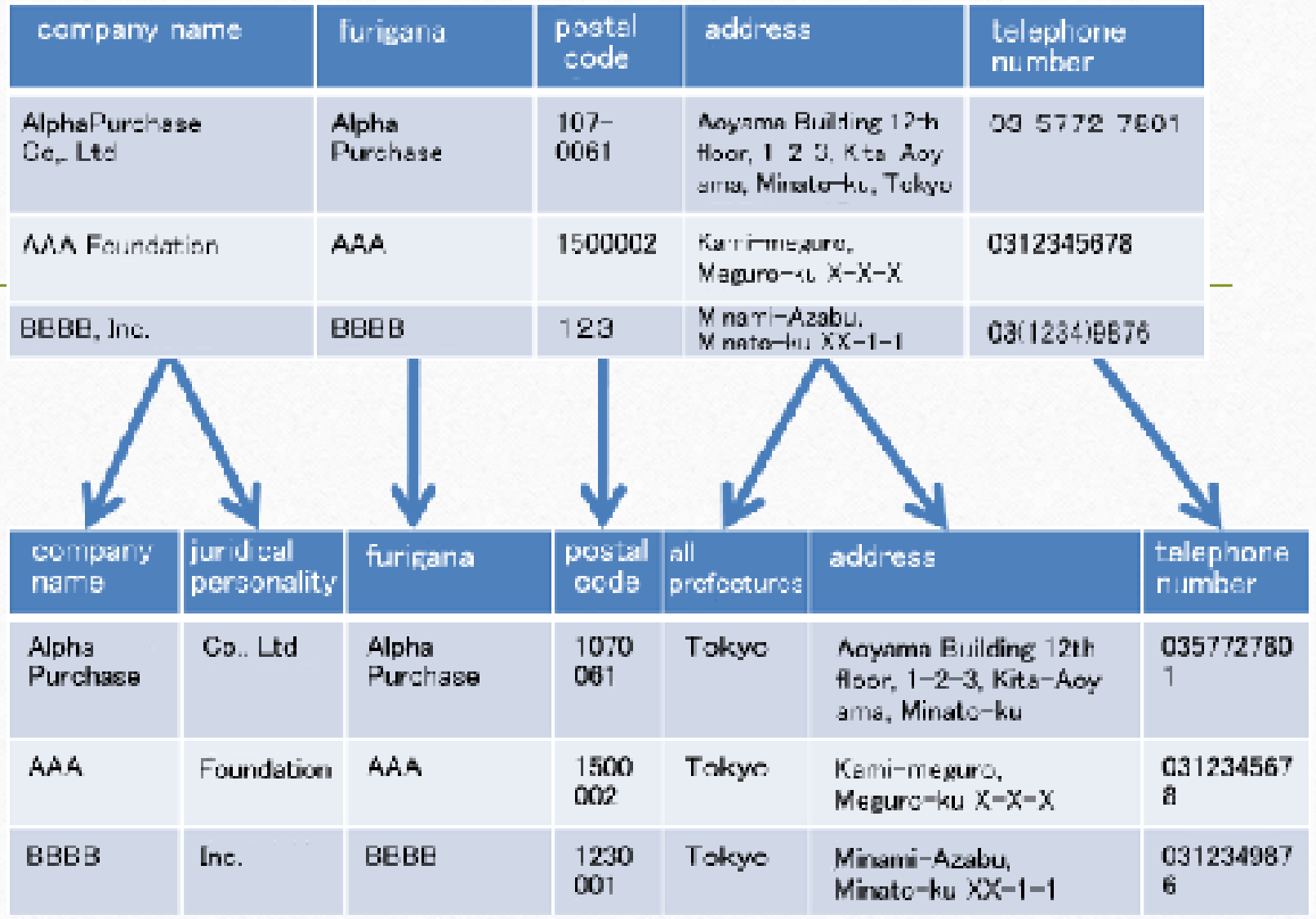
Data Preprocessing

- Data Cleaning
- Data Integration
- Data Reduction
- Data Transformation



Data Cleaning

1. ~~Missing Data~~
- Central Imputation
- KNN Imputation
- 2. Noisy Data
- Smoothing
- Clustering
1. Outlier Removal
- Using Boxplot



company name	furigana	postal code	address	telephone number
AlphaPurchase Co., Ltd	Alpha Purchase	107-0061	Aoyama Building 12th floor, 1 2 3, Kita Aoyama, Minato-ku, Tokyo	03 5772 7801
AAA Foundation	AAA	1500002	Kami-meguro, Meguro-ku X-X-X	0312345678
BBBB, Inc.	BBBB	123	Minami-Azabu, Minato-ku XX-1-1	03(12349876

company name	juridical personality	furigana	postal code	all prefectures	address	telephone number
Alpha Purchase	Co., Ltd	Alpha Purchase	1070061	Tokyo	Aoyama Building 12th floor, 1-2-3, Kita-Aoyama, Minato-ku	0357727801
AAA	Foundation	AAA	1500002	Tokyo	Kami-meguro, Meguro-ku X-X-X	0312345678
BBBB	Inc.	BBBB	1230001	Tokyo	Minami-Azabu, Minato-ku XX-1-1	0312349876

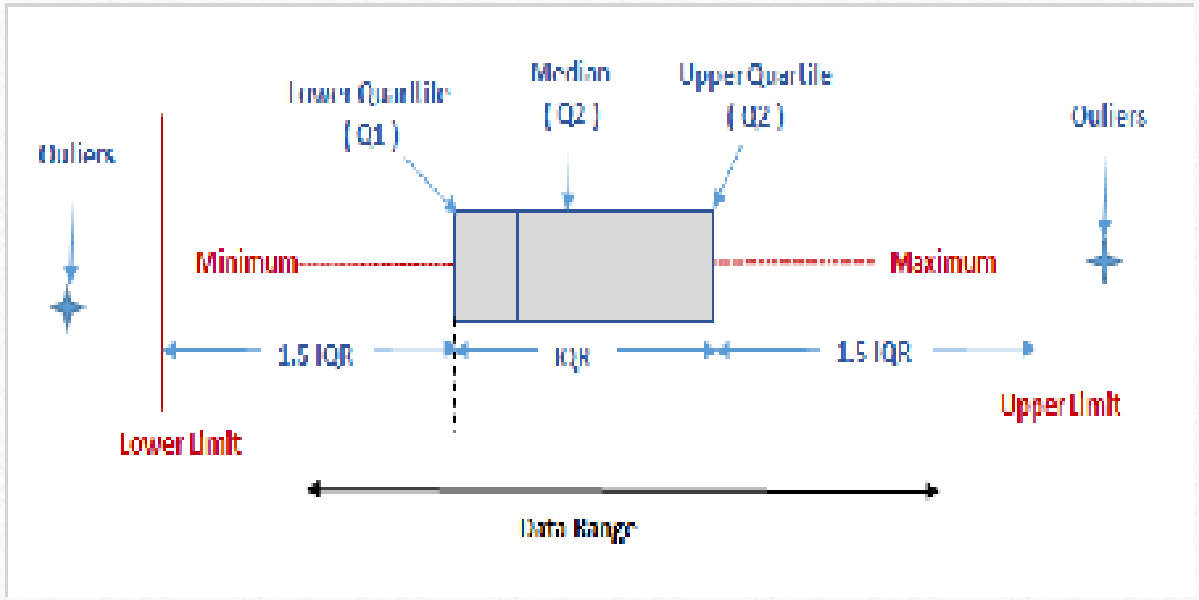
Imputation

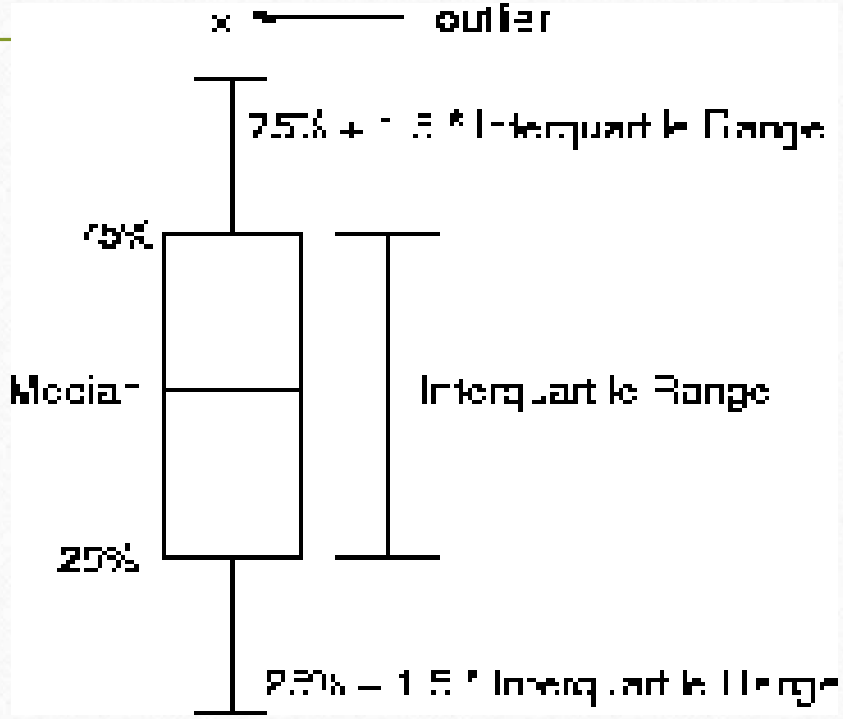
- Replace with mean or a median
- When to use mean?
- Replace with nearest neighbour
- How much nearest to see?

S.No	Qualification	Age	Income
1	B.Tech	25	30k
2	M.Tech	30	50k
3	B.Tech	26	32k
4	B.Tech	25	?
5	M.Tech	29	60k
6	B.Tech	?	30k

Outlier

- BoxPlot





Data Transformation

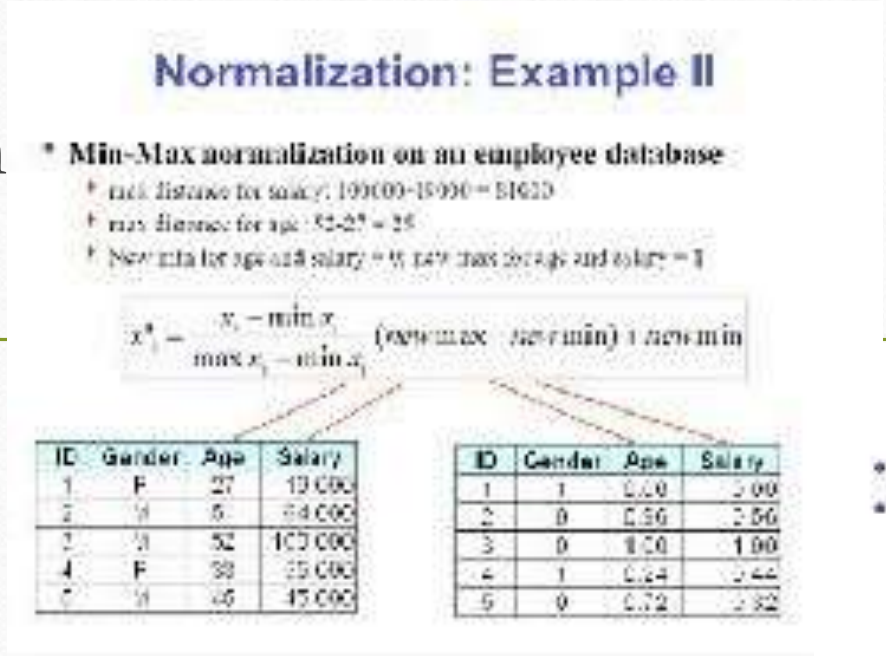
- Normalization

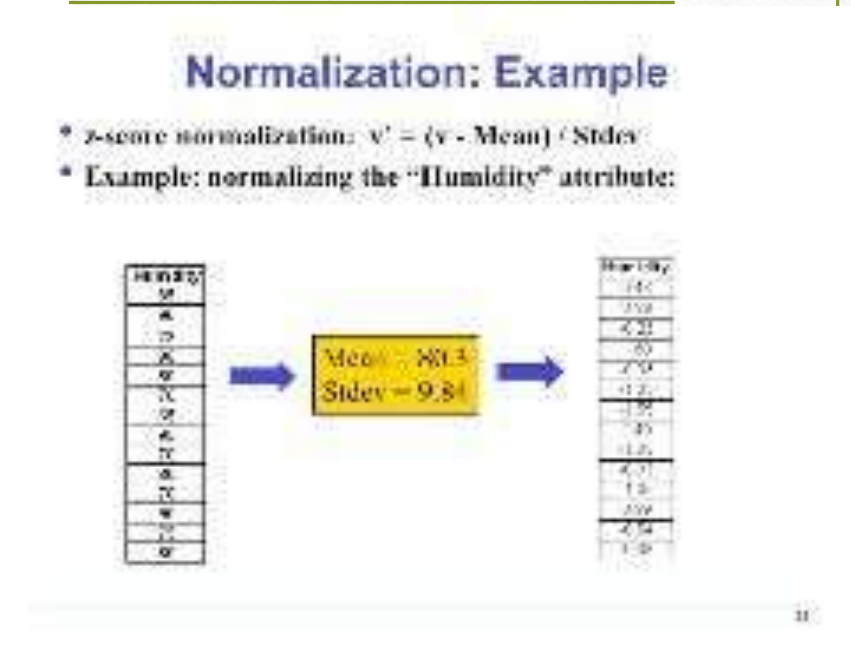
Min-max normalization

1. Min Max Normalization
2. Z - Score Normalization
3. Decimal scaling

Decimal scaling

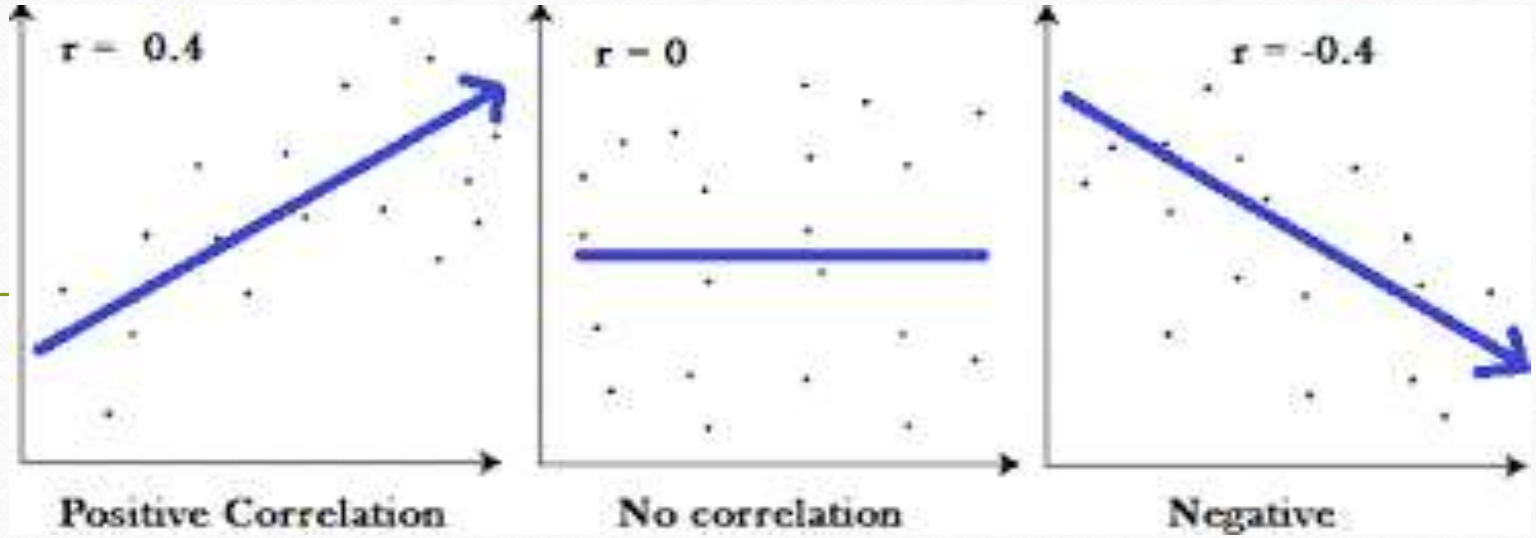
v= v/10^j





Data Integration

- Check for correlation
- Remove uncorrelated data



$$r = \frac{\sum (x - \overline{x})(y - \overline{y})}{\sqrt{\sum (x - \overline{x})^2 \sum (y - \overline{y})^2}}$$

Data Reduction

- Data Cube Aggregation

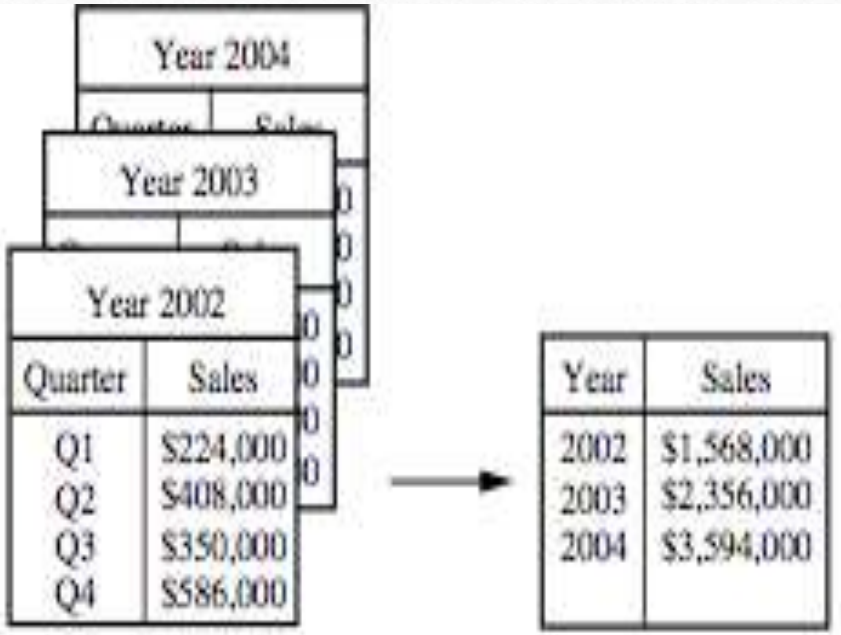


Figure 2.13 Sales data for a given branch of *AllElectronics* for the years 2002 to 2004. On the left, the sales are shown per quarter. On the right, the data are aggregated to provide the annual sales.

Introduction to Jupyter notebooks
Numpy
Pandas
Data Analysis

Notebook Basics

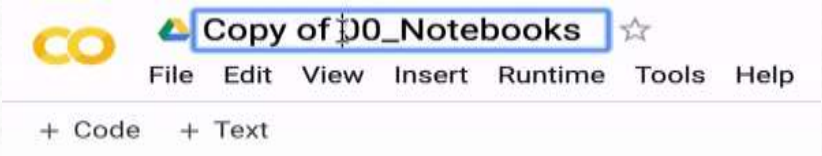
colab

In this lesson we'll learn how to work with **notebooks**. Notebooks allow us to do interactive and visual computing which makes it a great learning tool. We'll use notebooks to code in Python and learn the basics of machine learning.

Set Up

1. Sign into your Google account to start using the notebook. If you don't want to save your work, you can skip the steps below.
2. If you do want to save your work, click the **COPY TO DRIVE** button on the toolbar. This will open a new notebook in a new tab.

3. Rename this new notebook by removing the words `Copy of` from the title (change "`Copy of 00_Notebooks`" to "`00_Notebooks`").



4. Now you can run the code, make changes and it's all saved to your personal Google Drive.

Types of cells

Notebooks are made up of cells. Each cell can either be a **code cell** or a **text cell**.

code cell: used for writing and executing code.

text cell: used for writing text, HTML, Markdown, etc.

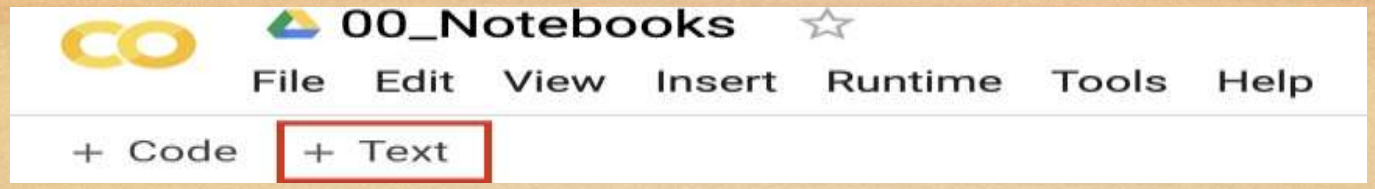
Creating cells

Add Shortcuts:

1. *New Code Cell: $Alt+N$*
2. *New text Cell: $Alt+T$*

First, let's create a text cell. Click on a desired location in the notebook and create the cell by clicking on the

+ **TEXT** (located in the top left corner).



Once you create the cell, click on it and type the following inside it:

```
### This is a header  Hello world!
```

This is a header

Hello world!

Markdown Formatting

Headers

Texts in order of large to small.

```
# Header 1
## Header 2
### Header 3
#### Header 4
##### Header 5
###### Header 6
```

- **Bold** - `**Text**`

  Bold Text

- **Italics** - `*Text*`

  Italicised Text

- **Links** - `[link text](https://)`

  A link

- **Image** - `![alt text](https://)`

NumPy

In this lesson we will learn the basics of numerical analysis using the NumPy package.

Set up

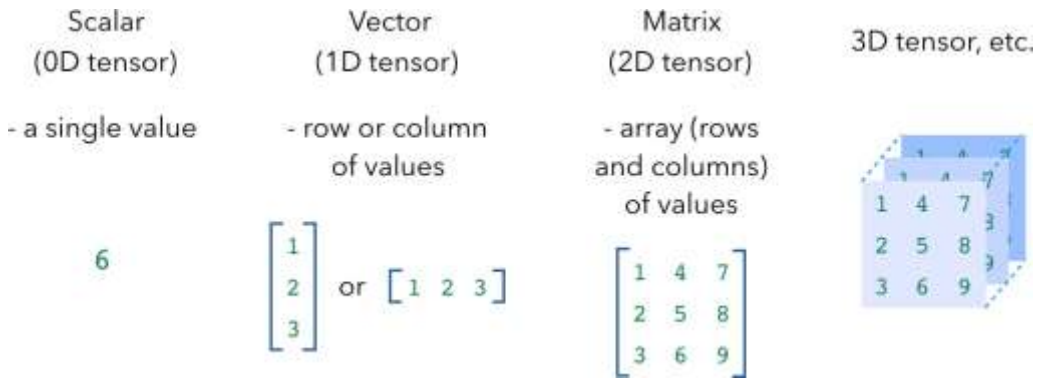
```
import numpy as np
```

```
# Set seed for reproducibility
np.random.seed(seed=1234)
```

Basics

Let's take a took at how to create tensors with NumPy.

- **Tensor**: collection of values




```
# Scalar
x = np.array(6) # scalar
print ("x: ", x)
# Number of dimensions
print ("x ndim: ", x.ndim)
# Dimensions
print ("x shape:", x.shape)
# Size of elements
print ("x size: ", x.size)
# Data type
print ("x dtype: ", x.dtype)
```

```
x:  6
x ndim:  0
x shape: ()
x size:  1
x dtype:  int64
```

```
# Vector
x = np.array([1.3 , 2.2 , 1.7])
print ("x: ", x)
print ("x ndim: ", x.ndim)
print ("x shape:", x.shape)
print ("x size: ", x.size)
print ("x dtype: ", x.dtype) # notice the float datatype
```

```
x:  [1.3 2.2 1.7]
x ndim:  1
x shape: (3,)
x size:  3
x dtype:  float64
```

```
# Matrix
x = np.array([[1,2], [3,4]])
print ("x:\n", x)
print ("x ndim: ", x.ndim)
print ("x shape:", x.shape)
print ("x size: ", x.size)
print ("x dtype: ", x.dtype)
```

```
x:
 [[1 2]
 [3 4]]
x ndim:  2
x shape: (2, 2)
x size:  4
x dtype:  int64
```

```
# 3-D Tensor
x = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
print ("x:\n", x)
print ("x ndim: ", x.ndim)
```

```
print ("x shape:", x.shape)
print ("x size: ", x.size)
print ("x dtype: ", x.dtype)
```

```
x:
 [[[1 2]
  [3 4]]

 [[5 6]
  [7 8]]]
x ndim:  3
x shape: (2, 2, 2)
x size:  8
x dtype:  int64
```

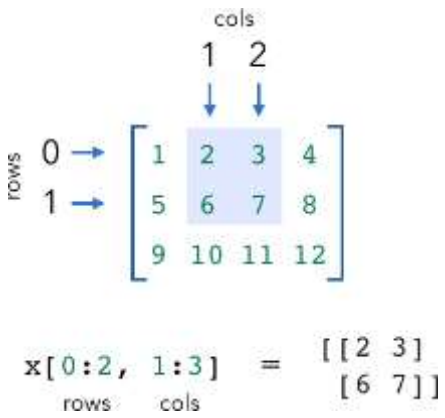
NumPy also comes with several functions that allow us to create tensors quickly.

```
# Functions
print ("np.zeros((2,2)):\n", np.zeros((2,2)))
print ("np.ones((2,2)):\n", np.ones((2,2)))
print ("np.eye((2)):\n", np.eye((2))) # identity matrix
print ("np.random.random((2,2)):\n", np.random.random((2,2)))
```

```
np.zeros((2,2)):
 [[0. 0.]
 [0. 0.]]
np.ones((2,2)):
 [[1. 1.]
 [1. 1.]]
np.eye((2)):
 [[1. 0.]
 [0. 1.]]
np.random.random((2,2)):
 [[0.19151945 0.62210877]
 [0.43772774 0.78535858]]
```

Indexing

Keep in mind that when indexing the row and column, indices start at 0. And like indexing with lists, we can use negative indices as well (where -1 is the last item).




```
# Indexing
x = np.array([1, 2, 3])
print ("x: ", x)
print ("x[0]: ", x[0])
x[0] = 0
print ("x: ", x)
```

```
x:  [1 2 3]
x[0]:  1
x:  [0 2 3]
```

```
# Slicing
x = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print (x)
print ("x column 1: ", x[:, 1])
print ("x row 0: ", x[0, :])
print ("x rows 0,1 & cols 1,2: \n", x[0:2, 1:3])
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
x column 1:  [ 2  6 10]
x row 0:  [1 2 3 4]
x rows 0,1 & cols 1,2:
 [[2 3]
 [6 7]]
```

```
# Integer array indexing
print (x)
rows_to_get = np.array([0, 1, 2])
```

```
print ("rows_to_get: ", rows_to_get)
cols_to_get = np.array([0, 2, 1])
print ("cols_to_get: ", cols_to_get)
# Combine sequences above to get values to get
print ("indexed values: ", x[rows_to_get, cols_to_get]) # (0, 0), (1, 2), (2, 1)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
rows_to_get:  [0 1 2]
cols_to_get:  [0 2 1]
indexed values:  [ 1  7 10]
```

```
# Boolean array indexing
x = np.array([[1, 2], [3, 4], [5, 6]])
print ("x:\n", x)
print ("x > 2:\n", x > 2)
print ("x[x > 2]:\n", x[x > 2])
```

```
x:
 [[1 2]
 [3 4]
 [5 6]]
x > 2:
 [[False False]
 [ True  True]
 [ True  True]]
x[x > 2]:
 [3 4 5 6]
```

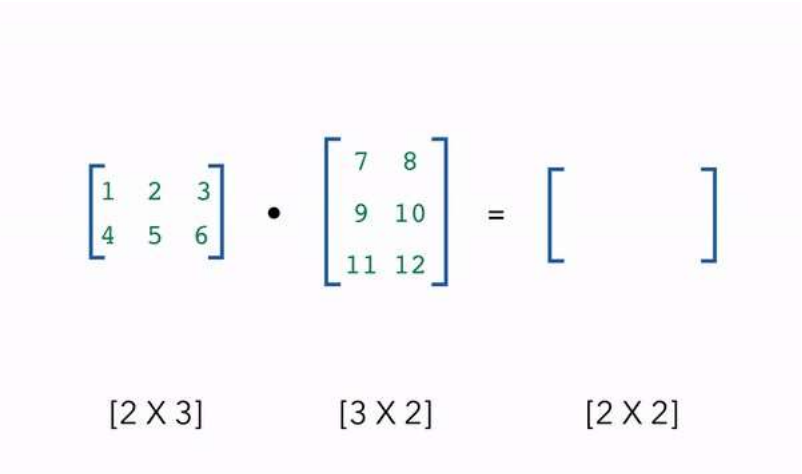
Arithmetic

```
# Basic math
x = np.array([[1,2], [3,4]], dtype=np.float64)
y = np.array([[1,2], [3,4]], dtype=np.float64)
print ("x + y:\n", np.add(x, y)) # or x + y
print ("x - y:\n", np.subtract(x, y)) # or x - y
print ("x * y:\n", np.multiply(x, y)) # or x * y
```

```
x + y:
 [[2. 4.]
 [6. 8.]]
x - y:
 [[0. 0.]
 [0. 0.]]
x * y:
 [[ 1.  4.]
 [ 9. 16.]]
```

Dot product

One of the most common NumPy operations we'll use in machine learning is matrix multiplication using the dot product. We take the rows of our first matrix (2) and the columns of our second matrix (2) to determine the dot product, giving us an output of `[2 X 2]`. The only requirement is that the inside dimensions match, in this case the frist matrix has 3 columns and the second matrix has 3 rows.

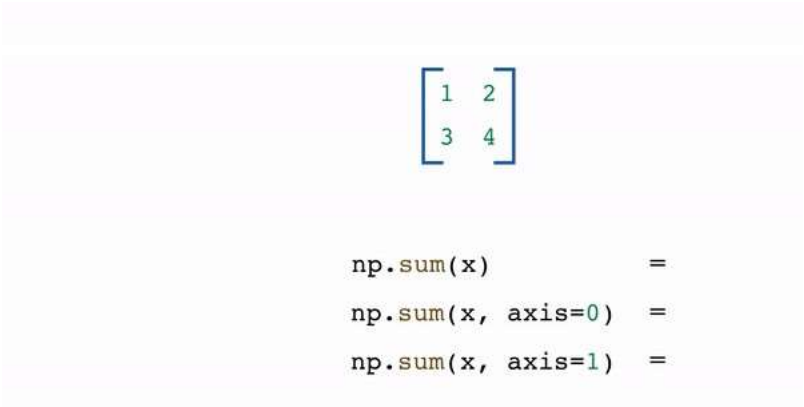



```
# Dot product
a = np.array([[1,2,3], [4,5,6]], dtype=np.float64) # we can specify dtype
b = np.array([[7,8], [9,10], [11, 12]], dtype=np.float64)
c = a.dot(b)
print (f"{a.shape} · {b.shape} = {c.shape}")
print (c)
```

```
(2, 3) · (3, 2) = (2, 2)
[[ 58.  64.]
 [139. 154.]]
```

Axis operations

We can also do operations across a specific axis.




```
# Sum across a dimension
x = np.array([[1,2],[3,4]])
print (x)
print ("sum all: ", np.sum(x)) # adds all elements
print ("sum axis=0: ", np.sum(x, axis=0)) # sum across rows
print ("sum axis=1: ", np.sum(x, axis=1)) # sum across columns
```

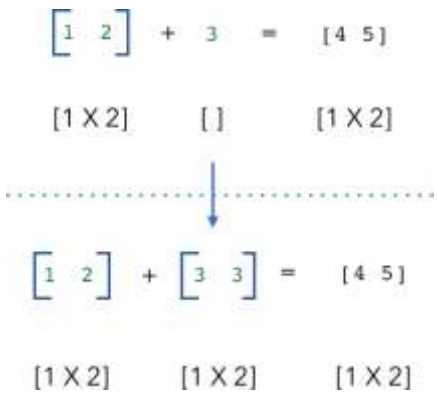
```
[[1 2]
 [3 4]]
sum all:  10
sum axis=0:  [4 6]
sum axis=1:  [3 7]
```

```
# Min/max
x = np.array([[1,2,3], [4,5,6]])
print ("min: ", x.min())
print ("max: ", x.max())
print ("min axis=0: ", x.min(axis=0))
print ("min axis=1: ", x.min(axis=1))
```

```
min:  1
max:  6
min axis=0:  [1 2 3]
min axis=1:  [1 4]
```

Broadcasting

Here, we're adding a vector with a scalar. Their dimensions aren't compatible as is but how does NumPy still gives us the right result? This is where broadcasting comes in. The scalar is *broadcast* across the vector so that they have compatible shapes.



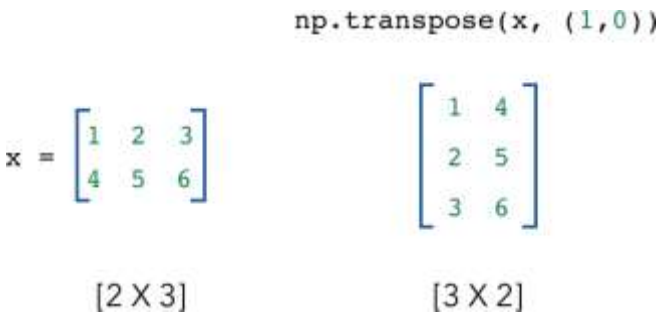

```
# Broadcasting
x = np.array([1,2]) # vector
y = np.array(3) # scalar
z = x + y
print ("z:\n", z)
```

```
z:
 [4 5]
```

Advanced

Transposing

We often need to change the dimensions of our tensors for operations like the dot product. If we need to switch two dimensions, we can transpose the tensor.




```
# Transposing
x = np.array([[1,2,3], [4,5,6]])
print ("x:\n", x)
print ("x.shape: ", x.shape)
y = np.transpose(x, (1,0)) # flip dimensions at index 0 and 1
print ("y:\n", y)
print ("y.shape: ", y.shape)
```

```
x:
 [[1 2 3]
 [4 5 6]]
x.shape:  (2, 3)
y:
 [[1 4]
 [2 5]
 [3 6]]
y.shape:  (3, 2)
```

```
# Transposing - Method 2
z = x.T
print ("z:\n", z)
print ("z.shape: ", z.shape)
```

```
z:
 [[1 4]
 [2 5]
 [3 6]]
z.shape:  (3, 2)
```

Reshaping

Sometimes, we'll need to alter the dimensions of the matrix. Reshaping allows us to transform a tensor into different permissible shapes -- our reshaped tensor has the same amount of values in the tensor. (1X6 = 2X3). We can also use `-1` on a dimension and NumPy will infer the dimension based on our input tensor.

The way reshape works is by looking at each dimension of the new tensor and separating our original tensor into that many units. So here the dimension at index 0 of the new tensor is 2 so we divide our original tensor into 2 units, and each of those has 3 values.

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \qquad [1 \times 6]$$

$$\texttt{np.reshape(x, (2, 3))} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad [\mathbf{2} \times 3]$$

$$\texttt{np.reshape(x, (2, -1))} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad [2 \times 3]$$

```
# Reshaping
x = np.array([[1,2,3,4,5,6]])
print (x)
print ("x.shape: ", x.shape)
y = np.reshape(x, (2, 3))
print ("y: \n", y)
print ("y.shape: ", y.shape)
z = np.reshape(x, (2, -1))
print ("z: \n", z)
print ("z.shape: ", z.shape)
```

```
[[1 2 3 4 5 6]]
x.shape:  (1, 6)
y:
 [[1 2 3]
 [4 5 6]]
y.shape:  (2, 3)
z:
 [[1 2 3]
 [4 5 6]]
z.shape:  (2, 3)
```

Unintended reshaping

Though reshaping is very convenient to manipulate tensors, we must be careful of their pitfalls as well. Let's look at the example below. Suppose we have `x`, which has the shape `[2 X 3 X 4]`.

```
[[[ 1  1  1  1]
  [ 2  2  2  2]
  [ 3  3  3  3]]
 [[10 10 10 10]
  [20 20 20 20]
  [30 30 30 30]]]
```

We want to reshape x so that it has shape `[3 X 8]` which we'll get by moving the dimension at index 0 to become the dimension at index 1 and then combining the last two dimensions. But when we do this, we want our output

to look like: ✅

```
[[ 1  1  1  1 10 10 10 10]
 [ 2  2  2  2 20 20 20 20]
 [ 3  3  3  3 30 30 30 30]]
```

and not like: ✗

```
[[ 1  1  1  1  2  2  2  2]
 [ 3  3  3  3 10 10 10 10]
 [20 20 20 20 30 30 30 30]]
```

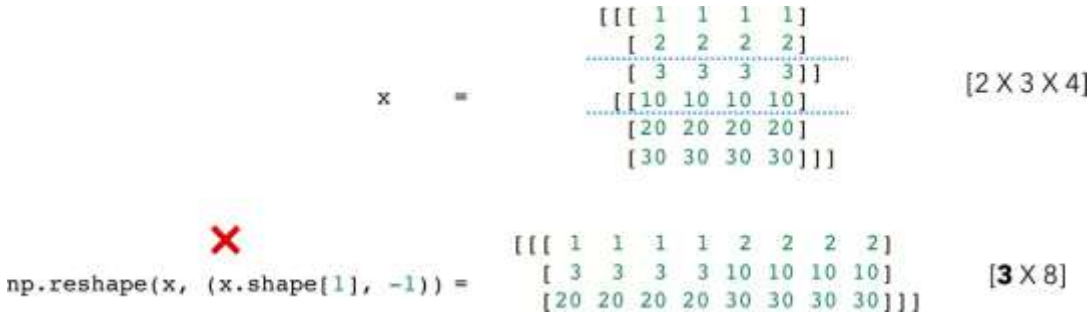
even though they both have the same shape `[3X8]`.

```
x = np.array([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3]],
              [[10, 10, 10, 10], [20, 20, 20, 20], [30, 30, 30, 30]]])
print ("x:\n", x)
print ("x.shape: ", x.shape)
```

```
x:
 [[[ 1  1  1  1]
  [ 2  2  2  2]
  [ 3  3  3  3]]

 [[10 10 10 10]
  [20 20 20 20]
  [30 30 30 30]]]
x.shape:  (2, 3, 4)
```

When we naively do a reshape, we get the right shape but the values are not what we're looking for.




```
# Unintended reshaping
z_incorrect = np.reshape(x, (x.shape[1], -1))
print ("z_incorrect:\n", z_incorrect)
print ("z_incorrect.shape: ", z_incorrect.shape)
```

```
z_incorrect:
 [[ 1  1  1  1  2  2  2  2]
 [ 3  3  3  3 10 10 10 10]
```

```
 [20 20 20 20 30 30 30 30]]
z_incorrect.shape:  (3, 8)
```

Instead, if we transpose the tensor and then do a reshape, we get our desired tensor. Transpose allows us to put our two vectors that we want to combine together and then we use reshape to join them together. Always create a dummy example like this when you're unsure about reshaping. Blindly going by the tensor shape can lead to lots of issues downstream.


```
# Intended reshaping
y = np.transpose(x, (1,0,2))
print ("y:\n", y)
print ("y.shape: ", y.shape)
z_correct = np.reshape(y, (y.shape[0], -1))
print ("z_correct:\n", z_correct)
print ("z_correct.shape: ", z_correct.shape)
```

```
y:
 [[[ 1  1  1  1]
  [10 10 10 10]]

 [[ 2  2  2  2]
  [20 20 20 20]]

 [[ 3  3  3  3]
  [30 30 30 30]]]
y.shape:  (3, 2, 4)
z_correct:
[[ 1  1  1  1 10 10 10 10]
 [ 2  2  2  2 20 20 20 20]
 [ 3  3  3  3 30 30 30 30]]
z_correct.shape:  (3, 8)
```

Adding/removing dimensions

We can also easily add and remove dimensions to our tensors and we'll want to do this to make tensors compatible for certain operations.

```
# Adding dimensions
x = np.array([[1,2,3],[4,5,6]])
print ("x:\n", x)
print ("x.shape: ", x.shape)
y = np.expand_dims(x, 1) # expand dim 1
print ("y: \n", y)
print ("y.shape: ", y.shape)   # notice extra set of brackets are added
```

```
x:
 [[1 2 3]
 [4 5 6]]
x.shape:  (2, 3)
y:
 [[[1 2 3]]

 [[4 5 6]]]
y.shape:  (2, 1, 3)
```

```
# Removing dimensions
x = np.array([[[1,2,3]],[[4,5,6]]])
print ("x:\n", x)
print ("x.shape: ", x.shape)
y = np.squeeze(x, 1) # squeeze dim 1
print ("y: \n", y)
print ("y.shape: ", y.shape)  # notice extra set of brackets are gone
```

```
x:
 [[[1 2 3]]

 [[4 5 6]]]
x.shape:  (2, 1, 3)
y:
 [[1 2 3]
 [4 5 6]]
y.shape:  (2, 3)
```

Pandas

In this notebook, we'll learn the basics of data analysis with the Python Pandas library.

Set up

```
import numpy as np
import pandas as pd
```

```
# Set seed for reproducability
np.random.seed(seed=1234)
```

Download data

We're going to work with the Titanic dataset which has data on the people who embarked the RMS Titanic in 1912 and whether they survived the expedition or not. It's a very rich dataset which makes it very apt for exploratory data analysis with Pandas.

Load data

Let's load the data from the CSV file into a Pandas dataframe. The `header=0` signfies that the first row (0th index) is a header row which contains the names of each column in our dataset.

```
# Read from CSV to Pandas DataFrame
df = pd.read_csv('https://bit.ly/2qZoqRH', header=0)
```

```
# First five items
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	1
1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	1

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	0
3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	0
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	0

```
# Last five items
df.tail()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
1304	3	Zabour, Miss. Hileni	female	14.5	1	0	2665	14.4542	NaN	C	0
1305	3	Zabour, Miss. Thamine	female	NaN	1	0	2665	14.4542	NaN	C	0
1306	3	Zakarian, Mr. Mapriededer	male	26.5	0	0	2656	7.2250	NaN	C	0
1307	3	Zakarian, Mr. Ortin	male	27.0	0	0	2670	7.2250	NaN	C	0
1308	3	Zimmerman, Mr. Leo	male	29.0	0	0	315082	7.8750	NaN	S	0

These are the diferent features:

- **pclass**: class of travel
- **name**: full name of the passenger
- **sex**: gender
- **age**: numerical age
- **sibsp**: # of siblings/spouse aboard
- **parch**: number of parents/child aboard
- **ticket**: ticket number
- **fare**: cost of the ticket
- **cabin**: location of room
- **emarked**: port that the passenger embarked at (C - Cherbourg, S - Southampton, Q = Queenstown)
- **survived**: survial metric (0 - died, 1 - survived)

Exploratory analysis

```
import matplotlib.pyplot as plt
```

We can use `.describe()` to extract some standard details about our numerical features.

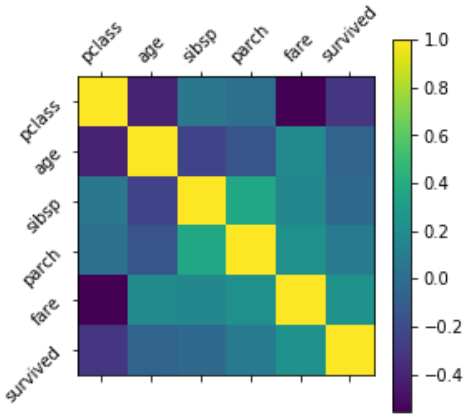
```
# Describe features
df.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	pclass	age	sibsp	parch	fare	survived
count	1309.000000	1046.000000	1309.000000	1309.000000	1308.000000	1309.000000
mean	2.294882	29.881135	0.498854	0.385027	33.295479	0.381971
std	0.837836	14.413500	1.041658	0.865560	51.758668	0.486055
min	1.000000	0.166700	0.000000	0.000000	0.000000	0.000000
25%	2.000000	21.000000	0.000000	0.000000	7.895800	0.000000
50%	3.000000	28.000000	0.000000	0.000000	14.454200	0.000000
75%	3.000000	39.000000	1.000000	0.000000	31.275000	1.000000
max	3.000000	80.000000	8.000000	9.000000	512.329200	1.000000

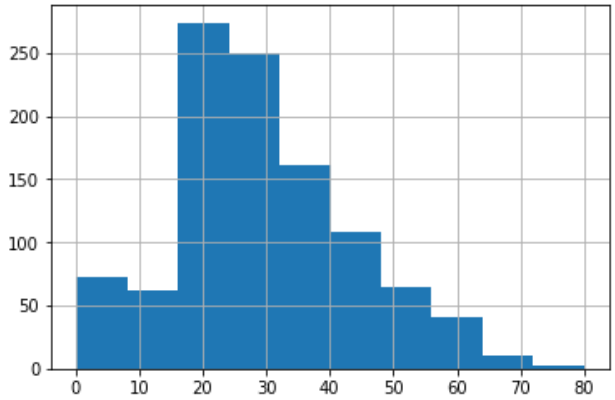
```
# Correlation matrix
plt.matshow(df.corr())
continuous_features = df.describe().columns
plt.xticks(range(len(continuous_features)), continuous_features, rotation='45')
plt.yticks(range(len(continuous_features)), continuous_features, rotation='45')
plt.colorbar()
plt.show()
```



We can also use `.hist()` to view the histrogram of values for each feature.

```
# Histograms
df['age'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fabd938d198>
```




```
# Unique values
df['embarked'].unique()
```

```
array(['S', 'C', nan, 'Q'], dtype=object)
```

Filtering

```
# Selecting data by feature
df['name'].head()
```

```
0                      Allen, Miss. Elisabeth Walton
1                     Allison, Master. Hudson Trevor
2                       Allison, Miss. Helen Loraine
3               Allison, Mr. Hudson Joshua Creighton
4    Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
Name: name, dtype: object
```

```
# Filtering
df[df['sex']=='female'].head() # only the female data appear
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```

```
.dataframe thead th {
    text-align: right;
}
```

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
0	1	Allen, Miss. Elisabeth Walton	female	29.0	0	0	24160	211.3375	B5	S	1
2	1	Allison, Miss. Helen Loraine	female	2.0	1	2	113781	151.5500	C22 C26	S	0
4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0	1	2	113781	151.5500	C22 C26	S	0
6	1	Andrews, Miss. Kornelia Theodosia	female	63.0	1	0	13502	77.9583	D7	S	1
8	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0	2	0	11769	51.4792	C101	S	1

Sorting

```
# Sorting
df.sort_values('age', ascending=False).head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
14	1	Barkworth, Mr. Algernon Henry Wilson	male	80.0	0	0	27042	30.0000	A23	S	1
61	1	Cavendish, Mrs. Tyrell William (Julia Florence...	female	76.0	1	0	19877	78.8500	C46	S	1
1235	3	Svensson, Mr. Johan	male	74.0	0	0	347060	7.7750	NaN	S	0
135	1	Goldschmidt, Mr. George B	male	71.0	0	0	PC 17754	34.6542	A5	C	0
9	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	PC 17609	49.5042	NaN	C	0

Grouping

```
# Grouping
survived_group = df.groupby('survived')
survived_group.mean()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	pclass	age	sibsp	parch	fare
survived					
0	2.500618	30.545369	0.521632	0.328801	23.353831
1	1.962000	28.918228	0.462000	0.476000	49.361184

Indexing & Slicing

We can use `loc` to get rows or columns at particuluar positions using locations(name of row/column) in the dataframe.

Syntax:

```
df.loc[rows [, columns]]
```

We can use `iloc` to get rows or columns at particuluar positions using integer locations in the dataframe.

```
df.iloc[irows [, icolumns]]
```

```
# Selecting row with index name 0 - loc
df.loc[0]
```

```
pclass                                  1
name         Allen, Miss. Elisabeth Walton
sex                                female
age                                    29
sibsp                                   0
parch                                   0
ticket                              24160
fare                               211.338
cabin                                   B5
embarked                                S
survived                                1
Name: 0, dtype: object
```

```
# Selecting row at location 0 - iloc
df.iloc[0]
```

```
pclass                                    1
name         Allen, Miss. Elisabeth Walton
sex                                  female
age                                      29
sibsp                                     0
parch                                     0
ticket                                24160
fare                                211.338
cabin                                    B5
embarked                                  S
survived                                  1
Name: 0, dtype: object
```

```
# Selecting a specific value - loc
df.loc[0, 'name']
```

```
'Allen, Miss. Elisabeth Walton'
```

```
# Selecting a specific value - iloc
df.iloc[0, 1]
```

```
'Allen, Miss. Elisabeth Walton'
```

```
# Selecting a specific value without loc or iloc - Method 1
df['name'][0]
```

```
'Allen, Miss. Elisabeth Walton'
```

```
# Selecting a specific value without loc or iloc - Method 2
df.name[0]
```

```
'Allen, Miss. Elisabeth Walton'
```

```
# Selecting a column - loc
df.loc[:, 'name']
```

```
0                          Allen, Miss. Elisabeth Walton
1                        Allison, Master. Hudson Trevor
2                          Allison, Miss. Helen Loraine
3                  Allison, Mr. Hudson Joshua Creighton
4        Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
                              ...
1304                             Zabour, Miss. Hileni
1305                           Zabour, Miss. Thamine
1306                        Zakarian, Mr. Mapriededer
1307                             Zakarian, Mr. Ortin
1308                              Zimmerman, Mr. Leo
Name: name, Length: 1309, dtype: object
```

```
# Selecting a column - iloc
df.iloc[:, 1]
```

```
0                          Allen, Miss. Elisabeth Walton
1                        Allison, Master. Hudson Trevor
2                          Allison, Miss. Helen Loraine
3                  Allison, Mr. Hudson Joshua Creighton
4        Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
                              ...
1304                             Zabour, Miss. Hileni
1305                           Zabour, Miss. Thamine
1306                        Zakarian, Mr. Mapriededer
1307                             Zakarian, Mr. Ortin
1308                              Zimmerman, Mr. Leo
Name: name, Length: 1309, dtype: object
```

```
# Selecting a column without loc or iloc - Method 1
df['name']
```

```
0                          Allen, Miss. Elisabeth Walton
1                        Allison, Master. Hudson Trevor
2                          Allison, Miss. Helen Loraine
3                  Allison, Mr. Hudson Joshua Creighton
4        Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
                              ...
1304                             Zabour, Miss. Hileni
1305                           Zabour, Miss. Thamine
1306                        Zakarian, Mr. Mapriededer
1307                             Zakarian, Mr. Ortin
1308                              Zimmerman, Mr. Leo
Name: name, Length: 1309, dtype: object
```

```
# Selecting a column without loc or iloc - Method 2
df.name
```

```
0                          Allen, Miss. Elisabeth Walton
1                        Allison, Master. Hudson Trevor
2                          Allison, Miss. Helen Loraine
3                  Allison, Mr. Hudson Joshua Creighton
4        Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
                             ...
1304                             Zabour, Miss. Hileni
1305                           Zabour, Miss. Thamine
1306                        Zakarian, Mr. Mapriededer
1307                              Zakarian, Mr. Ortin
1308                               Zimmerman, Mr. Leo
Name: name, Length: 1309, dtype: object
```

```
# Selecting multiple columns and rows - loc
df.loc[[0, 1, 2], ['name', 'age', 'parch']]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	age	parch
0	Allen, Miss. Elisabeth Walton	29.0000	0
1	Allison, Master. Hudson Trevor	0.9167	2
2	Allison, Miss. Helen Loraine	2.0000	2

```
# Selecting multiple columns and rows - iloc
df.iloc[[0, 1, 2], [1, 3, 5]]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	age	parch
0	Allen, Miss. Elisabeth Walton	29.0000	0
1	Allison, Master. Hudson Trevor	0.9167	2
2	Allison, Miss. Helen Loraine	2.0000	2

```
# Selecting multiple columns
df[['name', 'age', 'parch']]

# Note that you cannot access multiple rows this way however!
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	age	parch
0	Allen, Miss. Elisabeth Walton	29.0000	0
1	Allison, Master. Hudson Trevor	0.9167	2
2	Allison, Miss. Helen Loraine	2.0000	2
3	Allison, Mr. Hudson Joshua Creighton	30.0000	2
4	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	25.0000	2
...	...	...	...
1304	Zabour, Miss. Hileni	14.5000	0
1305	Zabour, Miss. Thamine	NaN	0
1306	Zakarian, Mr. Mapriededer	26.5000	0
1307	Zakarian, Mr. Ortin	27.0000	0
1308	Zimmerman, Mr. Leo	29.0000	0

1309 rows × 3 columns

```
# Slicing over multiple rows and columns - loc
df.loc[0: 10, 'name':'ticket']

# Note here that the index values are also names of the row.
# Hence, all the rows and columns which are sliced are returned.
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	sex	age	sibsp	parch	ticket
0	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160

	name	sex	age	sibsp	parch	ticket
1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781
2	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781
3	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781
4	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781
5	Anderson, Mr. Harry	male	48.0000	0	0	19952
6	Andrews, Miss. Kornelia Theodosia	female	63.0000	1	0	13502
7	Andrews, Mr. Thomas Jr	male	39.0000	0	0	112050
8	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0000	2	0	11769
9	Artagaveytia, Mr. Ramon	male	71.0000	0	0	PC 17609
10	Astor, Col. John Jacob	male	47.0000	1	0	PC 17757

```
# Slicing over multiple rows and columns - iloc
df.iloc[0: 11, 1:7]

# Note here that slicing happens by indices (integer locations).
# Hence, observe that the slicing behaves in the usual way.
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	name	sex	age	sibsp	parch	ticket
0	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160
1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781
2	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781
3	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781
4	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781
5	Anderson, Mr. Harry	male	48.0000	0	0	19952
6	Andrews, Miss. Kornelia Theodosia	female	63.0000	1	0	13502
7	Andrews, Mr. Thomas Jr	male	39.0000	0	0	112050
8	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0000	2	0	11769
9	Artagaveytia, Mr. Ramon	male	71.0000	0	0	PC 17609
10	Astor, Col. John Jacob	male	47.0000	1	0	PC 17757

Preprocessing

After exploring, we can clean and preprocess our dataset.

```
# Rows with at least one NaN value
df[pd.isnull(df).any(axis=1)].head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
9	1	Artagaveytia, Mr. Ramon	male	71.0	0	0	PC 17609	49.5042	NaN	C	0
13	1	Barber, Miss. Ellen "Nellie"	female	26.0	0	0	19877	78.8500	NaN	S	1
15	1	Baumann, Mr. John D	male	NaN	0	0	PC 17318	25.9250	NaN	S	0
23	1	Bidois, Miss. Rosalie	female	42.0	0	0	PC 17757	227.5250	NaN	C	1
25	1	Birnbaum, Mr. Jakob	male	25.0	0	0	13905	26.0000	NaN	C	0

```
# Drop rows with Nan values
df = df.dropna() # removes rows with any NaN values
df = df.reset_index() # reset's row indexes in case any rows were dropped
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	index	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
0	0	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	1
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	1

	index	pclass	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	survived
2	2	1	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	0
3	3	1	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	0
4	4	1	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	0

```
# Dropping multiple columns
df = df.drop(['name', 'cabin', 'ticket'], axis=1) # we won't use text features for our initial basic
models
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	index	pclass	sex	age	sibsp	parch	fare	embarked	survived
0	0	1	female	29.0000	0	0	211.3375	S	1
1	1	1	male	0.9167	1	2	151.5500	S	1
2	2	1	female	2.0000	1	2	151.5500	S	0
3	3	1	male	30.0000	1	2	151.5500	S	0
4	4	1	female	25.0000	1	2	151.5500	S	0

```
# Map feature values
df['sex'] = df['sex'].map( {'female': 0, 'male': 1} ).astype(int)
df['embarked'] = df['embarked'].dropna().map( {'S':0, 'C':1, 'Q':2} ).astype(int)
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	index	pclass	sex	age	sibsp	parch	fare	embarked	survived
0	0	1	0	29.0000	0	0	211.3375	0	1
1	1	1	1	0.9167	1	2	151.5500	0	1
2	2	1	0	2.0000	1	2	151.5500	0	0
3	3	1	1	30.0000	1	2	151.5500	0	0
4	4	1	0	25.0000	1	2	151.5500	0	0

Feature engineering

We're now going to use feature engineering to create a column called `family_size`. We'll first define a function called `get_family_size` that will determine the family size using the number of parents and siblings.

```
# Lambda expressions to create new features
def get_family_size(sibsp, parch):
    family_size = sibsp + parch
    return family_size
```

Once we define the function, we can use `lambda` to `apply` that function on each row (using the numbers of siblings and parents in each row to determine the family size for each row).

```
df["family_size"] = df[["sibsp", "parch"]].apply(lambda x: get_family_size(x["sibsp"], x["parch"]),
axis=1)
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	index	pclass	sex	age	sibsp	parch	fare	embarked	survived	family_size
0	0	1	0	29.0000	0	0	211.3375	0	1	0
1	1	1	1	0.9167	1	2	151.5500	0	1	3
2	2	1	0	2.0000	1	2	151.5500	0	0	3
3	3	1	1	30.0000	1	2	151.5500	0	0	3
4	4	1	0	25.0000	1	2	151.5500	0	0	3

```
# Reorganize headers
df = df[['pclass', 'sex', 'age', 'sibsp', 'parch', 'family_size', 'fare', 'embarked', 'survived']]
df.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	pclass	sex	age	sibsp	parch	family_size	fare	embarked	survived
0	1	0	29.0000	0	0	0	211.3375	0	1
1	1	1	0.9167	1	2	3	151.5500	0	1
2	1	0	2.0000	1	2	3	151.5500	0	0
3	1	1	30.0000	1	2	3	151.5500	0	0
4	1	0	25.0000	1	2	3	151.5500	0	0

Save data

Finally, let's save our preprocessed data into a new CSV file to use later.

```
# Saving dataframe to CSV
df.to_csv('processed_titanic.csv', index=False)
```

```
# See the saved file
!ls
```

```
processed_titanic.csv sample_data
```

Additional resources

- **Pandas reference manual**: There's so much more we can do with Pandas and we'll see even more in later notebooks. But if you're curious, checkout the Pandas user guide for more information.

Note: This notebook has been copied from this kaggle kernel. You can use this reliable coding implementation for your reference.

California Housing EDA & Data Cleaning

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
print(os.listdir("sample_data/"))
```

```
['housing.csv']
```

Loading our data , usually done with **Pandas** lib

```
housing = pd.read_csv('sample_data/california_housing_train.csv')
```

```
housing.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	o
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	N
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	N
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	N
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	N
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	N

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age  20640  non-null  float64
total_rooms           20640 non-null float64
total_bedrooms    20433    non-null    float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

> The count for each value in ocean_proximity column.

```
```

```
housing.ocean_proximity.value_counts()
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

```
housing.describe()
```

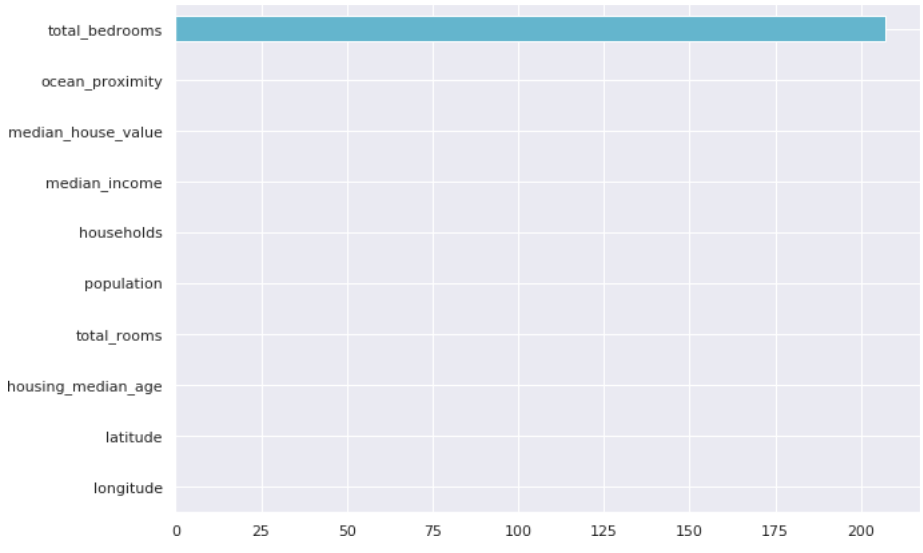
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

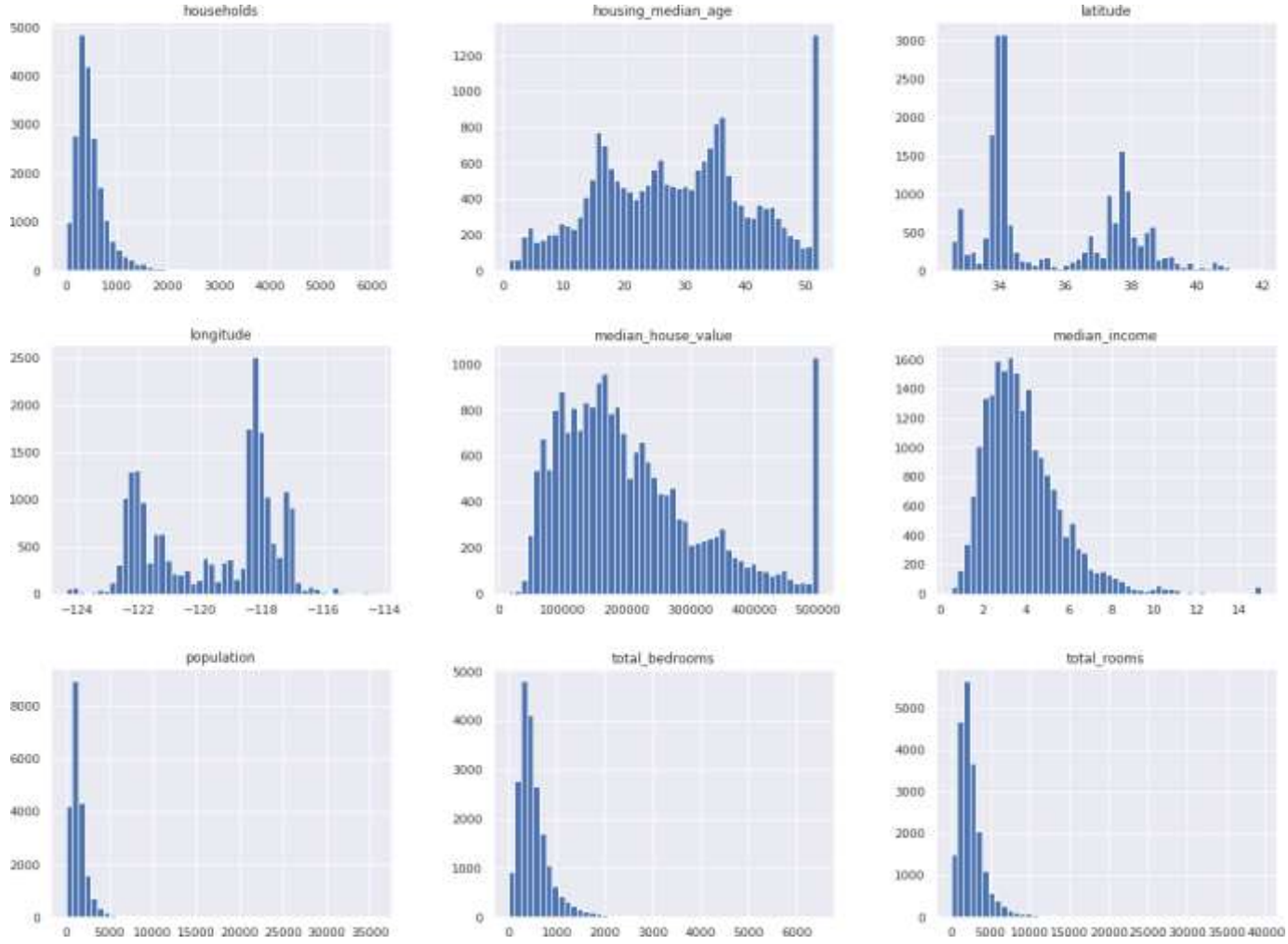
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	media
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001

```
sns.set()
housing.isna().sum().sort_values(ascending=True).plot(kind='barh',figsize=(10,7))#Quick peak into the missing columns values
#Let's deal with that later on the cleaning part with various methods !
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18f4f789b0>
```



```
housing.hist(bins=50,figsize=(20,15))#The bins parameter is used to custom the number of bins shown on the
plots.  plt.show()
```




```
from sklearn.model_selection import train_test_split
train_, test_ = train_test_split(housing,test_size=0.2,random_state=1)
```

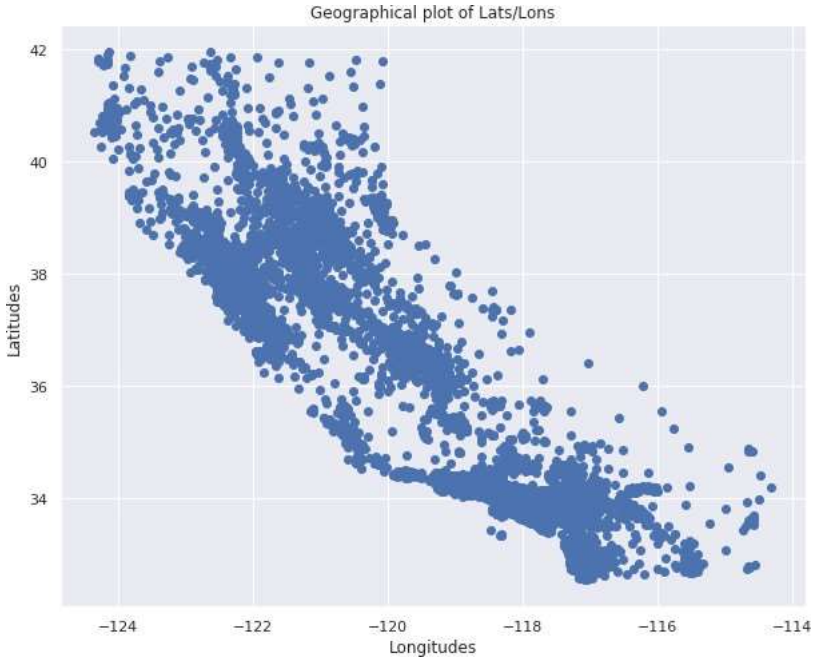
EDA Time to have a look on our Data

```
One good practice is to do EDA on the full data and creating a copy of it for not harming our test and training data.
```

```
plotter = housing.copy()
```

Since there is geographical information (latitude and longitude), it is a good idea to create a scatterplot of all districts to visualize the data

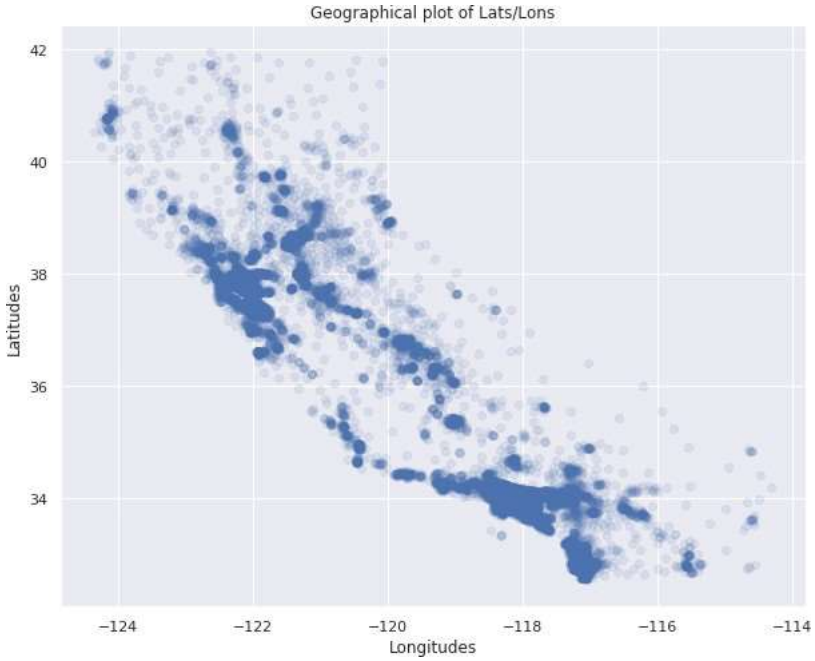
```
sns.set()
plt.figure(figsize=(10,8))#Figure size
plt.scatter('longitude','latitude',data=plotter)
plt.ylabel('Latitudes')
plt.xlabel('Longitudes')
plt.title('Geographical plot of Lats/Lons')
plt.show()
```



The plot above look like california RIGHT ?

But we don't have a **informative** look on the plot since we need to know the density for each point, let's do a simple modification.

```
sns.set()
plt.figure(figsize=(10,8))#Figure size
plt.scatter('longitude','latitude',data=plotter,alpha=0.1)
plt.ylabel('Latitudes')
plt.xlabel('Longitudes')
plt.title('Geographical plot of Lats/Lons')
plt.show()
```



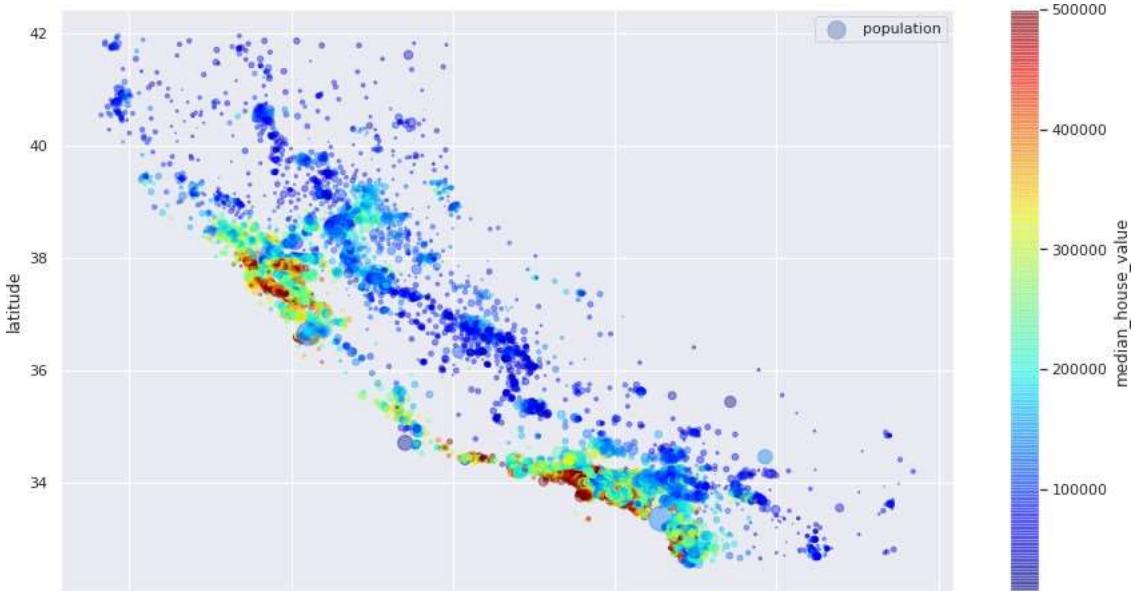
Now it's much better , and if we're familiar with Californias map we can see clearly that the high-density areas , namely the Bay Area and all around Los Angeles & San diego More generally our brains can spot patterns visually , but we always need to play around with the vizualisations to make the patterns stands out.

```
plt.figure(figsize=(10,7))
plotter.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
        s=plotter["population"]/100, label="population", figsize=(15,8),
        c="median_house_value", cmap=plt.get_cmap("jet"),colorbar=True,
    )
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f18f14920f0>



<Figure size 720x504 with 0 Axes>
```



Now we can say that the house price is a bit related to the location (e.g close to ocean) and to the density of the population.

```
corr_matrix=plotter.corr()
corr_matrix.median_house_value.sort_values(ascending=False)
```

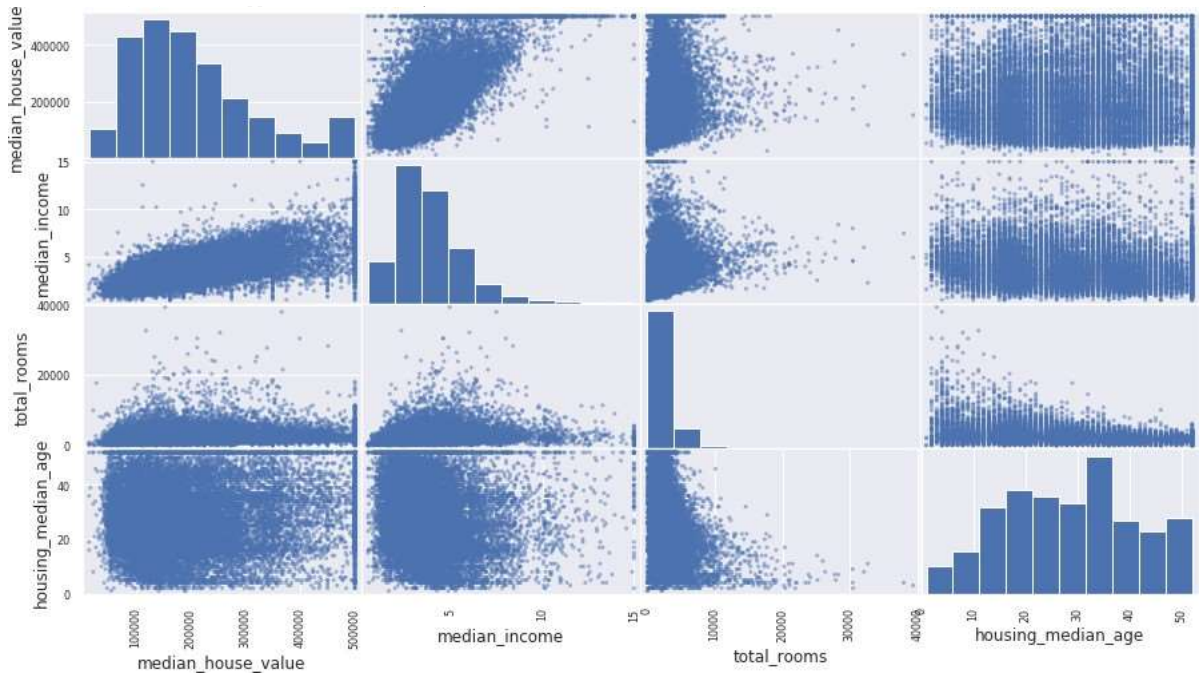
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

Checking the correlation between the main features with the Pandas function (Scatter_matrix) wich shows linear correlations between the features

```
from pandas import scatter_matrix
sns.set()
feat = ['median_house_value','median_income','total_rooms','housing_median_age']
scatter_matrix(plotter[feat],figsize=(15,8))
```

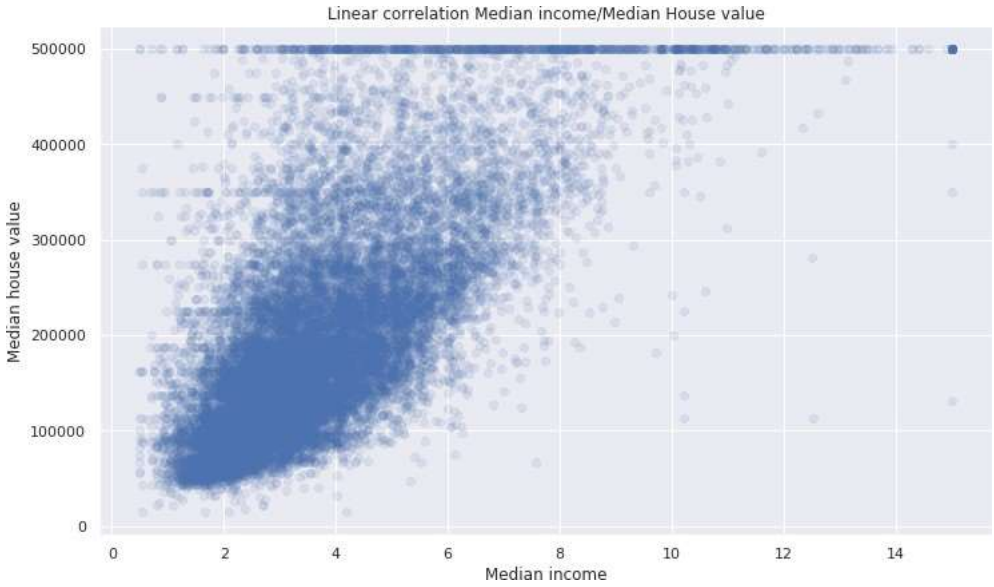
```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:4: FutureWarning: pandas.scatter_matrix is deprecated,
use  pandas.plotting.scatter_matrix instead
  after removing the cwd from sys.path.
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f18f4f50e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f1677f28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f167e7f0>,
        <matplotlib.axes._subplots.AxesSubplot object at
       0x7f18f0a1c198>],  [<matplotlib.axes._subplots.AxesSubplot object
       at 0x7f18f0a50b00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f0b7d4a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f0b8ee10>,
        <matplotlib.axes._subplots.AxesSubplot object at
       0x7f18f0ba47f0>],  [<matplotlib.axes._subplots.AxesSubplot object
       at 0x7f18f0ba4828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f1510ac8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f1560470>,
        <matplotlib.axes._subplots.AxesSubplot object at
       0x7f18f15b2dd8>],  [<matplotlib.axes._subplots.AxesSubplot object
       at 0x7f18f159f780>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f1641128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f18f15cda90>,
        <matplotlib.axes._subplots.AxesSubplot object at
```



```
plt.figure(figsize=(12,7))
plt.scatter('median_income','median_house_value',data=plotter,alpha=0.1)
plt.xlabel('Median income')
plt.ylabel('Median house value')
plt.title('Linear correlation Median income/Median House value')
```

```
Text(0.5, 1.0, 'Linear correlation Median income/Median House value')
```



NB: One last thing you may want to do before actually preparing the data for Machine Learning algorithms is to try out various attribute combinations. For example, the total number of rooms in a district is not very useful if you don't know how many households there are. What you really want is the number of rooms per household.

```
plotter['rooms_per_household']= plotter.total_rooms/housing.households
```

```
plotter.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	o
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	N
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	N
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	N
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	N
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	N

```
corr_matrix1=plotter.corr()
corr=corr_matrix1.median_house_value.sort_values(ascending=False)
d= pd.DataFrame({'Column':corr.index,
                 'Correlation with median_house_value':corr.values})
d
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Column	Correlation with median_house_value
0	median_house_value	1.000000
1	median_income	0.688075
2	rooms_per_household	0.151948
3	total_rooms	0.134153
4	housing_median_age	0.105623
5	households	0.065843
6	total_bedrooms	0.049686
7	population	-0.024650
8	longitude	-0.045967
9	latitude	-0.144160

> Not bad haha ! The number of rooms per household is now more informative than the total number of rooms in a district

Data cleaning

> Most Machine Learning algorithms cannot work with **missing features**, so let's create a few functions to take care of them. You noticed earlier that the total_bedrooms attribute has some missing values, so let's fix this. You have three options:

1. Get rid of the corresponding districts.
2. Get rid of the whole attribute.
3. Set the values to some value (zero, the mean, the median, etc.)

Since we don't have a lot of data the first option won't be the best , the second one too because we need that feature , the wisest choice could be the median , we can't affect the mean because we have some outliers this will affect our training model.

> I'm commenting those options just to show you how to do them i won't use them in this tutorial

```
#plotter.dropna(subset=["total_bedrooms"]) # option 1
#plotter.drop("total_bedrooms", axis=1) # option 2
#median = plotter["total_bedrooms"].median() # option 3
#plotter["total_bedrooms"].fillna(median, inplace=True)
```

> Scikit learn have a handy class to compute median , mean... strategies. We'll use that !

```
from sklearn.impute import SimpleImputer
imputer =SimpleImputer(strategy='median')#In this case its better to use the median to replace missing values
```

> If we run the code (imputer.fit(data)) we'll have an error since the imputer doesn't work on objects, and as shown at the very beginning we have a categorical attribute which is **"Ocean_proximity"** so we need to drop that.

```
ft_data = plotter.drop('ocean_proximity',axis=1)
```

```
imputer.fit(ft_data)
```

```
SimpleImputer(copy=True, fill_value=None, missing_values=nan,
       strategy='median', verbose=0)
```

```
imputer.statistics_ #Here's the median of every attribute in our data !
```

```
array([-1.18490000e+02, 3.42600000e+01, 2.90000000e+01, 2.12700000e+03,
       4.35000000e+02, 1.16600000e+03, 4.09000000e+02,  3.53480000e+00,
       1.79700000e+05,  5.22912879e+00])
```

```
ft_data.total_bedrooms.median()
```

```
435.0
```

Now you can use this “trained” imputer to transform the training set by replacing missing values by the learned medians:

```
X = imputer.transform(ft_data)
```

The result is a plain NumPy array containing the transformed features. We want to put it back into a Pandas DataFrame, it’s simple:

```
ft_transformed = pd.DataFrame(X,columns=ft_data.columns)
ft_transformed.tail() #The missing values in total_bedrooms were replaced by the median value
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_valu
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0

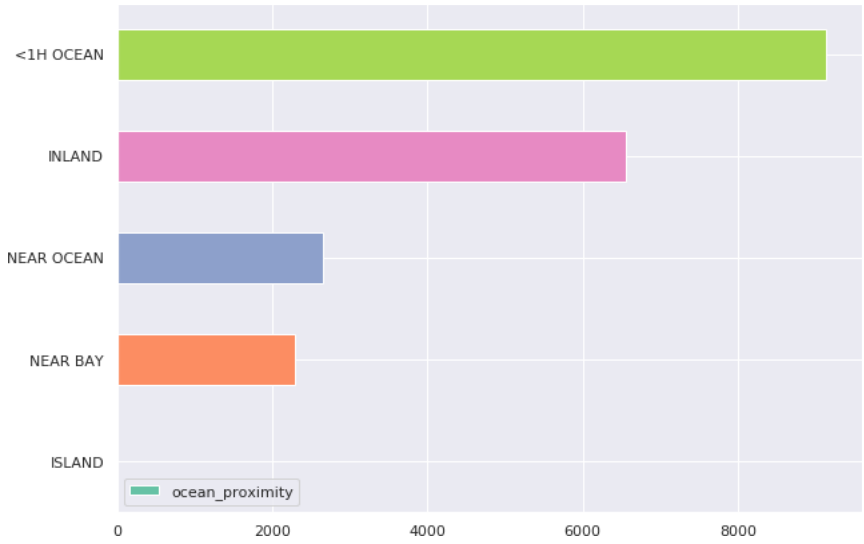
Let's handle our categorical data issue

```
obj_cols = housing.dtypes
obj_cols[obj_cols=='object']
```

```
ocean_proximity    object
dtype: object
```

```
sns.set(palette='Set2')
housing.ocean_proximity.value_counts().sort_values(ascending=True).plot(kind='barh',figsize=(10,7))
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f18f0acc940>
```



In this case i will one hot encode the labels, we got various encoders for categorical objects, label encoding, ordinal encoder...

```
from sklearn.preprocessing import OneHotEncoder
lab_encoder = OneHotEncoder()
cat_house = housing[['ocean_proximity']]
cat_enc = lab_encoder.fit_transform(cat_house)
```

One of the most important transformation step to apply to your data is **Feature Scaling** Because with some few exceptions, Machine learning algorithm won't perform well since we have different attributes scales, so what we want to do is to scale them , **note that target attribute doesn't have to be scaled**

We have two common ways to get all the attributes to have the same scale

1. Min-Max Scaling. Many people call it Normalization and its quite simple , values are shifted and rescaled to be in a range of 0 and 1

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}.$$

2. Standardization is a bit different, first it substracts the mean value so standardized values always have a zero mean, then it divides by the standard deviation so that the resulting distribution has unit variance, this is how we calculate standard deviation (Écart Type)

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2},$$

N is the number of our samples, We sum the Squared difference from mean which means (X(i) - $\bar{X}$) squared then we have our standard deviation, but dont worry we have a lot of ways compute all this, but it's always good to know what your computing. To compute STD (standard deviation) we use numpy , exemple : to compute the STD for the median_income we only have to do this --> np.std(data['median_income'])

Anyway as we showed we need a lot of transformations but thanks to scikit learn that provides a **Pipeline** class to help with such transformations link here : Pipeline doc'

NOW you're ready to go and start training your model on the train set and test it's accuracy on the test set that we created with the train_test_split function !

Thank you for reading !

```
If you found this helpful an upvote would be very much appreciated
```