DATE: __/__/____
PAGE: _____

# Data Analysis & Visualization

- **Data Science**
  Manipulation and analysis of data.

  Grouping of data can be of two types: structured and unstructured
  forms a class forms a cluster

- structured data is ordered.
- the data is classified & uses label
- eg: different cars like XUV, sedan, etc come under same class 'car' because they all are cars.

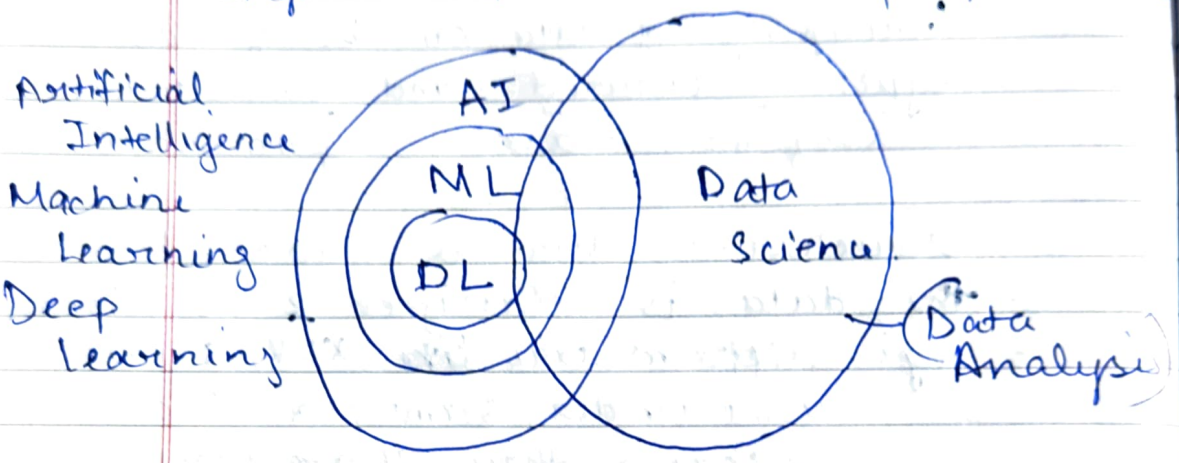
- Unstructured data is unordered.
- there are no labels.
- objects under unstructured data are ~~categ~~ grouped together in a cluster.

example:
consider a bread.
In structured data, we will have class of breads. (similar objects) like wholewheat, brown bread, etc.

unstructured data will group together the things that go together with bread like eggs, butter, milk. so it will form a cluster of things which go together with bread

DATE: / /
PAGE:

In structured data, we have labels like colour, shape, etc. but in unstructured data, we do not have labels, so we try to group together things based on similar properties.




Data science contains AI, ML, and DL but is not limited to it. It is vast and can be done with other tools as well.

DATE: __/__/____
PAGE: ____

- EAD (exploratory data analysis)
  - ↳ removal of unnecessary data
  - ↳ find corelation b/w 2 data.

$y = f(x)$

$f(x_1, x_2, x_3, \ldots)$

↳ feature vectors.

finding corelation b/w x and y

- data preprocessing: removal of noise (grainy) from photo comes under data preprocessing.

To reduce the calculation time, we can consider everything other than facial features to be noise and crop out.

actual: 200 x 200 x 100 x 60

cropped: 60 x 60 x 100 x 60

60 x 60 → less calculation time.

100 → no. of img.

60 → no. of people

→ suppose we have x values and y values. If any of the y values is not there, what do we do? we drop it. We can't take avg. why take headache of prediction if we'll take avg. only & we can find corelation only when prediction is done. SO, we drop it.

Good Write

That means, remove the data (remove whole line including x & y values).

- If any x value is missing, we can average or mod.

→ ~~Exploratory Data Analysis~~

→ What is data science?

Data science is a blend of various tools, algorithms and machine learning principles. Most simply, it involves obtaining meaningful information or insights from structured or unstructured data through a process of analyzing programming and business skills. It is a field containing many elements like mathematics, statistics, computer science, etc.

→ How data science works?

Data science is not a one step process, it passes from many stages and every element is important.

following are the steps for the development of a data science model

I. Problem statement.

- It is really important to declare or formulate your problem statement very clearly and it should be directed towards the pivotal goal to be achieved.

II Data Collection.

- After defining the problem statement, the next obvious step is to go in search of data that your model might require. Data can be in any form, i.e. unstructured or structured. It can be in various forms like, Videos, spreadsheets, coded form, etc.

III. Data cleaning

- Data cleaning is all about the removal of missing, redundant, unnecessary and duplicate data from your collection various tools such as numpy and pandas from python are used for data cleaning.

IV. Data analysis and exploration

- It's all about analysing the structure of data, finding hidden patterns in them, studying behaviours, visualising the effects of one variable over the others and then concluding. We can use mat.lib and seaborn libraries of python for visualization.

Good Write

DATE : __/__/____
PAGE : _____

## V Data Model

- Once the study have been performed and detection have been made for data visualization, a hypothesis model must be build that may yield a good prediction for the target value.

There are different types of algorithms from regression to classification:-
SPM,
decision tree, or
You can also form clusters based on unsupervised learning.
After creation of the model, the model must be trained using training data and its predictive analysis can be done using test data..

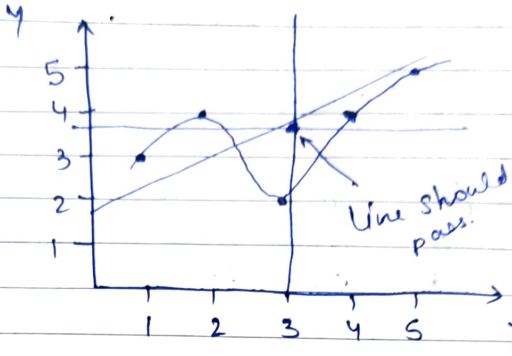
Good Write

DATE: / /
PAGE:

## Numerical :

$X = [1, 2, 3, 4, 5]$

$Y = [3, 4, 2, 4, 5]$




we want to draw a line which passes ~~straight~~ ~~thro~~ nearest to the plotted points.

$y = mx + c$.

$$m = \frac{\Sigma (x - \bar{x})(y - \bar{y})}{\Sigma (x - \bar{x})^2}$$

$\bar{x} = 3$, $\bar{y} = 3.6$

| | $x - \bar{x}$ | $y - \bar{y}$ | $(x - \bar{x}) \times (y - \bar{y})$ |
|---|---|---|---|
| 1 | $-2$ | $-0.6$ | $1.2$ |
| 2 | $-1$ | $0.4$ | $-0.4$ |
| 3 | $0$ | $-1.6$ | $0$ |
| 4 | $1$ | $0.4$ | $0.4$ |
| 5 | $2$ | $1.4$ | $2.8$ |
| | | | $4$ |

$$\Sigma (x - \bar{x})^2 = 4 + 1 + 1 + 4$$
$$= 10$$

$$m = \frac{\Sigma (x - \bar{x})(y - \bar{y})}{\Sigma (x - \bar{x})^2} = \frac{4}{10} = 0.4$$

Good Write

$y = mx + c.$

$\bar{y} = m\bar{x} + c.$

$3.6 = 0.4 \times 3 + c.$

$3.6 = 1.2 + c$

$\boxed{c = 2.4}$

| $x$ | | $\bar{y}$ | $y$ |
|---|---|---|---|
| 1 | | 2.8 | 3 |
| 2 | | 3.2 | 4 |
| 3 | | 3.6 | 2 |
| 4 | | 4 | 4 |
| 5 | | 4.4 | 5 |

↑ predicted

# $R_2$ score : $\dfrac{\sum (y_p - \bar{y})^2}{\sum (y - \bar{y})^2}$

if predicted value of $y$ ($y_p$) is equal to $y$, then it is best. so, it should be near 1.

$(y_p - \bar{y})$ :

$2.8 - 3.6 = -0.8$ Squ $0.64$

$3.2 - 3.6 = -0.4$ $0.16$

$3.6 - 3.6 = 0$ $0$

$4 - 3.6 = 0.4$ $0.16$

$4.4 - 3.6 = 0.8$ $0.64$

$1.60$

$(y - \bar{y})^2$ : $3 - 3.6 = -0.6 =$

$4 - 3.6 = 0.4 =$

$2 - 3.6 = -1.6 =$

$4 - 3.6 = 0.4 =$

$5 - 3.6 = 1.4 =$

$0.36 + 0.16 + 2.56 + 0.16 + 1.96$

$= 5.20$

$R^2$ score $= \frac{1.60}{5.20} = \frac{16}{52} = \frac{4}{13}$

$= 0.3$

→ play.tensorflow

→ Plt.scatter (x, y) is used to plot the true values corresponding to x

→ plt.plot : used to plot predicted value of y wrt x

→ Kagal. search cat and dog .csv da it will have series of x's and y. figure out which ones are importan find correlation b/w x & y.

methods : 1) corr.

2) plot.

# Pandas

Series vs Dataframe.

## * Series

- a series is a one-D array like an object, a sequence of values and an associated array of data labels called as its index.

- eg: ser_array = pd.Series([1,2,3,4,5])

- The string representation of a series shows the index on the left and the values on the right. since the index is not specified, a default one consisting of the integers 0 through n-1 is created.

- manual indexing eg:-

```
obj = pd.Series([2,-3,1,4],
                index = ['a', 'b',
                         'd'])
```

we can manually specify the index by writing the keyword 'index' and specifying the indices.

Also, names = ['1', '', '', '']
index = names

DATE: / /
PAGE:

## * Dataframe.

- A Dataframe represents a rectangular table of data and contains an ordered collections of columns, each of which can be a different value type. The Dataframe has both a row & column index. Under the hood, the data is stored as one or more 2-D blocks rather than a list, dictionary or some other collection of one-dimensional arrays.

- eg: obj = pd.Dataframe([1, 2, 3], [4, 5, 6])

```
data = {'state': ['Ohio', 'Delhi', 'Texas'],
        'year': [1996, 2000, 2004],
        'pop': [15, 17, 32]}

obj2 = pd.DataFrame(data)
```

- Columns have a name, rows are generally indexes.

- frame = pd.Dataframe(data, columns=['State', 'Year', 'Pop'], index=['A', 'B', 'C'])

DATE: / /
PAGE:

* Index objects

arange

obj = pd.Series(range(3), index = ['A', 'B', 'C'])

index = obj.index // reading ind

// reading particular index.

index[:1] → a

index[1:] → b c

// changing index.

index[1] = 'd' // not possible.

→ In pandas, indexes are immutable. We can not change them.

** Index objects are immutable and thus cannot be modified by the user.

labels = pd.Index(np.arange(3))

obj2 = pd.Series([1.5, 2.5, 3.0], index = labels)

DATE: / /
PAGE:

## * Essential functionality

1. Re-indexing: Reindex creates a new object with the data conformed to a new index

```
obj. = pd.Series([4.2, 1, 3.4, 6],
                 index=['d','b','c','a'])

obj2 = obj.reIndex(['d','b','a','c','e'],
                   method= ffill )
// index of obj originally is not
// changed. rather, a new object
// with new indexes is created.
```

obj2:

| | |
|---|---|
| d | 0.2 |
| b | 1.6 |
| a | 3.4 |
| c | 6.0 |
| e | NaN |

// To fill empty values, we can use functions:
1. fillNa
2. ffill
3. bfill.

- ~~Charles~~
  - → ffill - forward fill, fills upward value.
  - → bfill - ~~beg~~ backward fill, fills downward value.

## — Dropping entries from axis

DATE: / /
PAGE:

→ Dropping one or more entries from an axis is easy if you already have an index array or list without those entries. For doing this, the 'Drop' method will be used.

→ The Drop method will return a new object with the indicated value deleted from an axis.

```
obj = pd.Series([1, 2, 3, 4, 5],
                index=['a', 'b', 'c',
                       'd', 'e'])

obj2 = obj.drop('a')          // To drop index a
OR   = obj.drop(['a', 'd'])   // To drop two in

data = np.arange(16).reshape(4, 4)
obj3 = pd.DataFrame(data,
                    index=['a', 'b', 'c', 'd'],
                    columns=['one', 'two', 'three', 'four'])

// dropping column.

obj3.drop('three')
```

→ You can drop values from the column by passing axis = 1 or axis = 'columns'.

Good Write

→ iloc : index location
takes numerical value.

→ loc : takes name values.

→ data loading, storage and file formats :

- Handling dates and other custom types require extra efforts.
for eg: data saved in any format for example a.csv, a.txt, can be easily read using python's pandas library.
The only thing that is kept in mind is how the values are separated. They can be separated by a tab, ';', ',', space or even a ':'.
All formats are readable using pandas library.

- Reading a.csv file :

```
cars = pd.read_csv("cars.csv")
```

- only ',' separated files are read as .csv.
excel is just a way to access a csv file.

Good Write

DATE :
PAGE :

- we can also use 'read_table' for reading such values but we have to specify the delimitor.

```
cars = pd.read_table("cars.csv",
                     delimiter = ",")
                       ↓
                     sep = ","
```

→ if label is missing, then specify header = None.
otherwise first row will be considered as header names.

→ changing in file :

- ```
  cars = pd.read_csv("cars.csv",
         names = [" ", " ", " " ...])
  ```
  ↳ the columns available

- ```
  pd.read_csv("cars.csv", names = names,
              index_col = 'message')
  ```

- Suppose you want the message column to be the index of the returned data frame, you can indicate the column with a certain name using the index_col argument

if the separator between values is a variable space, we can use the regular expression "s+" writter

DATE: / /
PAGE:

as '\s+'

Suppose you want to skip some rows while reading a csv file use

skiprows = [2, 0, 3]

↑ these rows will be skipped.

→ Reading txt files into pieces.

df = pd.read_csv("cars.csv", nrows = 10)

↑ no. of rows from start.

- suppose we want to divide the given csv into several chunks :

chunksize = 5

↓

last chunk will contain all remaining rows.

→ storing data

- one of the easiest way to store data efficiently in a serialized manner, we can use python's built-in command, i.e. 'pickle'

data.to_pickle("data.csv")

↳ or any path.

DATE :
PAGE :

if you want to read retrieve it.

pd.read_pickle ("cars.csv")

→ Reading excel file.

for reading an excel file, we need the syntax: pd.read_excel.

data = pd.read_excel (" ")

→ Discretization and Binning.

- Binning is the process of creating bins or containers of specific ranges.

ages = [20, 21, 32, 29, 41, 18 . . ]

bins = [18, 25, 35, 60, 100]

cats = pd.cut (ages, bins)

cats

Output : (18, 25], (18, 25], (25, 35] . . .
(20 belongs) (21 belongs) (32 belongs)

each value of age is categorized according to the bin range.

Good Write

pd.value_counts (cats)

⟹ calculates values contained in each bin.

- // labeling the bins

group_names = ["young", "young-adult", "adult", "  "]

pd.cut (ages, bins, labels = group_names)

// to make 4 random bins.

~~data~~ pd.cut (data, 4, precision=2)
↑
4 random bins.

// To make bins with equal no. of // values in them.

data1 = np.random.randn (1000)
cats = pd.qcut (data1, 4).

→ qcut will fail if we have the same number in abundance. suppose we have the no. '2' 8 times, and the rest of the no.s a single time. In that case, qcut won't divide into equal bins & its purpose will fail.

Good Write

→ Permutation & random sampler

```
df = pd.Dataframe(np.arange(5*4).
                  reshape((5,4)))

sampler = np.random.permutation(5)

df.
```

// sampler will give random orders
// permutation is used for taking 5 numbers randomly.
// random.permutation is used for taking numbers randomly which are ordered randomly as well.

eg: instead of 0,1,2,3,4
we have 0,4,3,1,2

→ 1-HOT encoding.

Suppose we have ~~data~~ categorical data with strings and values. We cannot use numpy & panda library on it. So, we convert the categorical data to numerical data.
let us consider following eg.

Good Write

India
China
Japan

now, if data a
Ja - 3
library
value r
But it

Now
data
from
Jute

| | | |
|---|---|---|
| In | 1 | 1 |
| Ch | 2 | 0 |
| Ja | 3 | 0 |

this
0 is
no

Synt

df =

pd

Good

| | |
|---|---|
| India | Jute. |
| China | Silk. |
| Japan | cotton. |

now, if we convert it to numeric data and assign In. 1, Ch. 2, Ja - 3, Numpy or panda library will consider the value numerically like 1 < 2 < 3. But it is not that kind of value.

Now, we will encode the data into a matrix. In the form of true and false values

| | Jute | Silk | cotton. |
|---|---|---|---|
| In 1. | 1 | 0 | 0 |
| Ch 2 | 0 | 1 | 0 |
| Ja 3 | 0 | 0 | 1 |

this way, we can say that 0 is false & 1 is true, now we can use numeric libraries.

Syntax:

```
df = pd.DataFrame(
        'key': ['b', 'b', 'a', 'c', 'a', 'b'],
        'data2' : arange(6) ).

pd.get_dummies(df["key"])
```

Good Write

DATE: / /
PAGE:

## → Regular Expression.

```
import re
txt = 'on the horizon'
x = re.findall ("on", txt)
print (x)
// finding all occurences of "on" in txt
```

['on', 'on']

```
// checking if a string is present or not

txt1 = "_ _ "
x1 = re.search ("python", txt1)
print (x1.start ())
        ↳ // tells what is the posⁿ of the first occu
// separating words of a string
txt = " "
x2 = re.split ("\s", txt)
// x2 is a list which contains each word of the string.
```

DATE: __/__/____
PAGE: _____

// to check if a string starts with a particular word, use '^'.

```
x3 = re.findall ("^python", txt)
                  ↳ checking this word.

if x3 :
    print ("Yes")
else:
    print ("No match").
```

// to check if a string ends with a word, use '$'

```
x4 = re.findall ("tutorial$", txt)
if x4 :
    print ("Yes")
else :
    print ("No match").
```

// to check if any of the words exist in string, use OR '|'.

```
x5 = re.findall ("python|tutorial", txt)
if x5 :
    print ("yes")
else:
    print ("No match").
```

Good Write

// to check if there is one or more occurrence of the word, use '+'.

```
x7 = re.findall("thi+", txt)

if x7:
    print("yes")
else:
    print("No match")
```

→ Group by

To find the mean for a particular column,

```
df["  "].mean()
   ↳ column name.
```

Now to find mean of a particula column which has a particular value in another column.

→ df[df["symbols"]="RIVN"]["close"].m

∴ now, if we have different categori (RIVN, LCID ...), then we have to write the statement multiple times to get mean of each category separately.

→ df[df["symbols"]="LCID"]["close"].mean

Simplifying this expression using groupby

→ df.groupby("symbol")["close"].mean

⇒ titanic.csv file.
- to fill null age values.

titanic['age'] = titanic["age"].
replace(['?'], [None]).
astype['float']

∴ (float type for months).

reading csv and select 4 columns.

file → df = pd.read_csv("titanic")

data frame → titanic = df[["survived", "name",
"sex", "age"]].

gbo = titanic.groupby('sex')
↳ or (by='sex')

- gbo.ngroup (to mention no. of groups)

- gbo.get_group("female")
- to do any type of work with a particular category.