Chapter 2

Quick intro to - Pandas  python™

Contents:

- Pandas – Getting to know some useful data structures
- Reading data from CSV
- Using basic operations with Pandas
- Handling missing data
- Data merging and concatenation operations
- Annexure – A glimpse on Data scientist role! *[Not for your exams!]*

Pandas Python - Introduction

Why use Pandas? -

-- Essentially built on the Numpy package

-- Its basic & key data structure is referred to as DataFrame

-- DataFrames - Facilitate to store and manipulate tabular data in rows (number of observations) and columns of variables (features).

Pandas Python - Understanding the data

Tabular data – Visualization (DataFrames) -

Rows (number of observations); Columns of variables (features).

First Name	Gender	Start Date	Salary	Bonus %	Team
Douglas	Male	8/6/93	97308	6.945	Marketing
Thomas	Male	3/31/1996	61933		
Maria	Female	4/23/1993	130590	11.858	Finance
Jerry	Male	3/4/05	138705	9.34	Finance
Larry	Male		101004	1.389	Client Services
Dennis	Male	4/18/1987	115163	10.125	Legal
Ruby	Female	8/17/1987	65476	10.012	Product
	Female	7/20/2015	45906	11.598	Finance

Row →

Col →

Features – First Name, Gender, Start Date, Salary, Bonus %, Team (6 features);
So, each data point for us in the above dataset means a six-tuple entry
(a 6 dimensional vector); We have, 8 observations and 6 features here!

Pandas Introduction (Cont'd)

With Pandas Data Frames (DFs), it is easy to extract the required information;

If we have, say, *body fat data of individuals in a community*, we can easily answer questions like:

- *What is the average amount of fat in chest?*
- *What is the min amount of fat accumulated in the waist and who is having this min amount?*
- *Is there any correlation between the fat accumulated in thighs and waist?*
- *. . .*

Pandas Introduction (Cont'd)

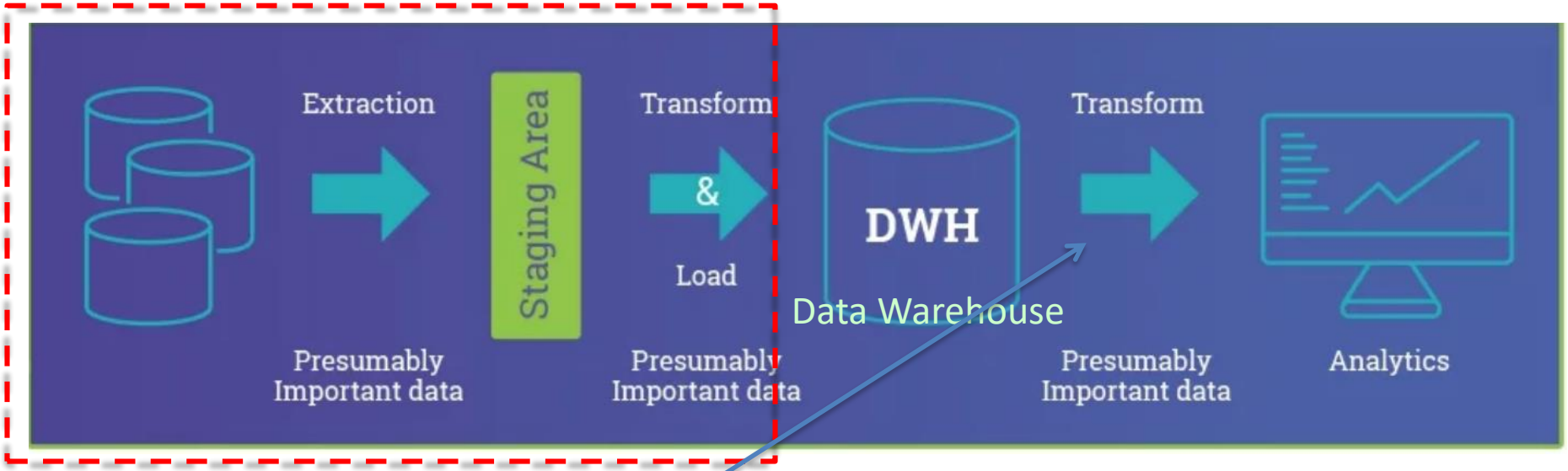
From Slide #1:

Note that you have some "gaps" or "missing data" in some of the observations; This is a very common problem in any raw data;

As a DE, your first job is to "*wrangle*" the raw data that is passed to you!

Data wrangling – Refers to the *process of cleaning, structuring and enriching raw data* into a desired format for quick and efficient processing in the analytics stage of the entire process! (*See Chapter 1 – Part1 for details*)

Data Extraction-Transform-Load (ETL)



Using Pandas, we can do this effectively

First, we will see how to extract data from a CSV file and use it for our needs.

ETL – *A practical approach – What it means in programming?*

Data is obtained, (extracted) cleaned, wrangled (transformed), and placed into a user-friendly data structure like a data frame (loaded).

Extract - The data requested via an API may not necessarily have come in a specific format we expect; This is the step in which we extract the data in the form we want!

One thing not to overlook when we observe the extracted data is in the way the data is accessed! This means application handling may be easy with certain way of indexing our data. Consequently we need to attend to this problem.

Consequently…..

ETL – *A practical approach! (Cont'd…)*

Transform – Issues *pertaining to data representations* conducive for our application needs to be taken care at this stage.

These include, NaNs, date-time formats, checking for duplicates and possibly grouping them using some metric like (mean, median, etc) that can give additional insights to the application processing, etc, This is a crucial step that can save lots of time later!

ETL – *A practical approach! (Cont'd…)*

Load – After transforming, we can now "load" our "new data" and any further transformations we make into a target database or data warehouse.

So, getting to know about the DWH you are using is imperative in this last step. Some examples of such repositories include, SQL, NoSQL, Data Frames(pandas), MongoDB, etc

Creating a Data Frame

One way to create a dataframe is via python *dictionary!*
(recall all your python data structures!)

To use pandas package, you need to import

```
>> import pandas as pd
```

Practice Problem 2.1a

```
>> mydict = {"Country": ["Brazil", "Russia", "India", "China",
"South Africa"],  "Capital": ["Brasilia", "Moscow", "New Dehli",
"Beijing", "Pretoria"],  "Currency": ["R$", "Russian Ruble",
"Indian Rupee", "Chinese Yuan", "South African Rand "],
"Currency_Sym": ["R", "₽", "₹", "¥", "R"] }

>> country_info = pd.DataFrame(mydict)
>>  print(country_info)   # What did you observe?
```

Data Frame…(cont'd)

The tabular output automatically puts indices for each row! We can change it as we like it using index() method. From Ex 2.1a

```
>> country_info.index = ["BR", "RU", "IN", "CH", "SA"]
# Print out country_info with new index values
print(country_info)
```

Data Frame is a multi-dimensional structure. *What if you want only one column information from a DF?* This is equivalent to extracting the information for all the data points (rows) for a given column (feature).

Pandas Series

Series: Series in Pandas is essentially a Column information; **DataFrame** in Pandas is a multi-dimensional table made up of a collection of *Series*. Elements in a *Series* can be accessed by our usual indexing style, as you do with single dim arrays.

For any specific feature, we can compute statistical quantities - summing (.sum()), averaging (.mean()), counting (.count()), median (.median()), determining unique values in a feature (.nunique())

You may extract a column for the purpose of concatenating with other columns!

TRY! In Tut 2.1a, extract a column and print specific entries!

Pandas Series

Extract country info and concatenate with currency symbol column

Practice Problem 2.1b

Making a copy of *Country feature* column

```
>>> new = country_info[“Country"].copy()
```

Concatenating Country column with Currency column by overwriting the current Currency column:

```
>>> country_info[“Currency"] =
country_info[“Currency_Sym "].str.cat(new, sep =", ")
```

Reading data from files & some basic operations with data frames

Large scale data is usually available in the form of files using different formats – csv, json, etc.

```
>> BFA = pd.read_json('bodyfat.json')
>> BFA = pd.read_csv('bodyfat.csv')
```

This means: BFA is a dataframe; You can access any row of the **df** using its index.

```
>> print(BFA[2:3])  # accessing 2nd row
>> print(BFA[3:6]) # accessing rows 3 to 5
```

Same as your slicing mechanism!

To get a general knowledge on your data, use:

```
>> BFA.head()        >> BFA.tail()
>> BFA.shape      BFA.describe()     >> BFA.info()
```

Reading data from files & some basic operations with data frames

Practice Problem 2.2

>> BFA.head() – By default extract the top 5 rows; we can change the # of rows to be displayed;

>> BFA.tail(2) – Last 2 rows; we can modify as above;

>> BFA.shape – Displays the size of DF as a tuple (rows,cols) – useful when we need to extract certain number of rows or columns from a dataframe!

>> BFA.describe() – Most important o/p; Gives all most common statistical measures – mean, std.dev, *25%,50%,75%,* count, min and max values for each of the features

Practice Problem 2.2

>> BFA.info() - provides the essential details about your dataset, such as # the number of rows and columns, the number of non-null values, what type of # data is in each column, and how much memory your DataFrame is using.

* percentiles

Example A:

```
mydata = {'product': ['A', 'B', 'C', 'C', 'D'],
          'price': [22000, 27000, 25000, 29000, 35000],
          'year': [2014, 2015, 2016, 2017, 2018] }
DF = pd.DataFrame(data)
print(DF)

# We can get a description specifically for certain categories;

stats_categorical = df['product'].describe()
print(stats_categorical)
```

Reading data from files & some basic operations with data frames

*Suppose you want to extract statistics **for specific features** in your dataset! How do you do this?*

You can group data **using a shared common value** and then summarize the values in another column using those groups.

>> Use: groupby() method Excellent way to extract an embedded info!

```
>> dataframe.groupby(['label_column'])[["value_column"]].method()
```

Note - Here the ''label_column" is the column is used to create the groups and the ''value_column" is the column that will be summarized for each group.

Reading data from files & some basic operations with data frames

>> PROBLEM 2.2 (Cont'd): Add a "category" to your bodyfat data and derive the statistics for 'bodyfat' feature and print. For 'category' generate random integers between 1 to 10; This category is equivalent to assigning an individual to a specific category.

Multiple aggregation of data as groups - If we want to group more than one quantity, we can do by listing them in the "*value col*" parameter w.r.t a specific "*label col*" parameter

Basic operations with data frames

- Many times each dataset will have several features. We may not know the names of those features obviously. We can extract them as:

>> DF.columns *This will return the features in a list and hence we can simply store as a list! (DF is your dataframe)*

Data Wrangling step here! We can "clean up" the way certain *column descriptors* appear.

Practice Problem 2.3

Consider the data EmpSmalldata0.csv. Try changing the features to all capital letters in your data frame; Try to extract data corresponding to specific features and specific rows, etc.

Basic operations with data frames

- Index and value_counts() - Very useful to know

if duplicate entries are there in your data;

- Alternatively, we can use *.duplicated()* method to extract all duplicates either w.r.t all columns or w.r.t specific columns; See the effect of "keep" parameter

- Accessing specific (single) rows and columns

  > Use loc() and iloc() methods

- Try sorting *w.r.t* specific column feature

  > Use sort_values()

Basic operations with data frames

Practice Problem 2.4

Demonstrate index(), value_counts(), and duplicated() methods with the given employees data.

- Extract specific rows of data using loc() and iloc() methods coupled with “at” and “iat”;
- For a set of specific rows, extract specific columns using loc() and also using the DF directly
- Consider a set of rows; Sort the data for a specific column and observe if other data values are preserved for a given row;

Basic operations with data frames

DIY Exercise!! (*No code or help will be provided*)

- Explore and try the following methods: Sum, mean, median, mode, std deviation, quantiles, for any dataset DF – *Search and use the corresponding methods! Easy!*
- Delete a set of given rows and columns from your dataset; *Useful to delete (how?) duplicate rows and columns!* - **Explore**: drop() method

Remember to make a copy of the original data and try the above exercise!

Handling Missing Data

- Most real-life data will not have all values captured and it is up to the DE to deal with them! ☹
- Missing values are handled differently by packages (Python – "None" and in Numpy – np.nan); *Some datasets assign some improbable values to missing numerical entries*
- Techniques that are used usually to handle such missing values include:

  (a) Remove the entire row

  (b) Replace the missing values using some relevant statistical quantities like – mean, median, and mode (*imputation*) - (method: .fillna(new_value)).

  (c) If "several" entries in a column are missing, then you may drop that entire column, if that feature does not add value to the application processing!

Missing data – an important remark

- In our practice problems above, we had counted the number of "missing data" and left the cells unfilled.

- In practice, you need to fill the "voids" with "NaN"s or using one of the ways mentioned above. If we do not fill we may encounter a problem due to "*data casting*" – other packages that are used in your application processing may not adapt to "voids" without any values or "NaN"s.

  Thus, it is customary that we fill the gaps right after any numerical data is extracted.

Missing data – an important remark

- Several schemes are available to capture the presence of missing data in a DF. Generally, they revolve around one of two strategies: (i) Using a *mask* that globally indicates missing values, or (ii) choosing a *sentinel value* that indicates a missing entry.

- In the masking approach, the mask might be an entirely separate Boolean array.

Missing data – an important remark

- In the sentinel approach, the sentinel value could be some data-specific convention, such as indicating a missing integer value with -9999 or some improbable values as far as application is concerned. Alternatively, indicating a missing floating-point value with NaN (*Not a Number*), a special value which is part of the IEEE floating-point specification could be used as a default.

Handling Missing Data

Practice Problem 2.5

Consider the dataset *EmpSmalldata.csv* given to you.

EmpSmalldata.csv has some missing data. Identify the missing data in each feature, count the number of missing data in each feature and report. Print this count information as a Pandas Series. Then, use the results captured in this series to drop all the columns that have more than 3 missing values.

(*As mentioned earlier, you can try to fill in the gaps and see the impact of results!)*

Methods in use: isnull(), drop()

Useful Data Pre-processing method - Converting categorical data to a column vector data

Where is this used? - Machine learning algorithms

When we are performing machine learning we need to work with algorithms that cannot process categorical variables.

For instance, we need to convert our column of labels (Ex: ['potato', 'carrot', 'butter', 'nuts']) into separate columns of 0s and 1s. This is called getting dummies() pandas columns used as inputs to a ML algo.

Pandas pd.get_dummies() will automatically convert our categorical column (column of labels) into indicator columns (columns of 0s and 1s).

Example B

Consider a dictionary U given as:

```
U = {'X': ['potato', 'carrot', 'nuts'], 'Y': ['carrot', 'butter', 'nuts'],
                'Q': ['potato','potato','potato'], 'M': ['nuts',np.NaN,'nuts'],
                'Z': [7,9,13]})
```

Use dummies() to derive vector data for all the categorical data in your DF

Useful Data Pre-processing method – Common computations on individual rows / columns

Given a dataframe, we may need to derive data that is not present in the DF and augment it to the original DF. This is one of the useful pre-processing steps during data extraction as well as in transformation steps of your ETL process.

One of the most useful methods is the *apply() method* of data preprocessing. It simplifies applying a function on each element in a pandas Series and each row or column in a pandas DataFrame.

Practice Problem 2.6

Consider a DF as shown below.

```
data = pd.DataFrame({'EmployeeName': ['Callen Dunkley', 'Sarah Rayner',
'Jeanette Sloan', 'Kaycee Acosta', 'Henri Conroy', 'Emma Peralta', 'Martin Butt',
'Alex Jensen', 'Kim Howarth', 'Jane Burnett'], 'Department': ['Accounting',
'Engineering', 'Engineering', 'HR', 'HR', 'HR', 'Data Science', 'Data Science',
'Accounting', 'Data Science'], 'HireDate': [2010, 2018, 2012, 2014, 2014, 2018,
2020, 2018, 2020, 2012],  'Sex': ['M', 'F', 'F', 'F', 'M', 'F', 'M', 'M', 'M', 'F'],
'Birthdate': ['04/09/1982', '14/04/1981', '06/05/1997', '08/01/1986',
'10/10/1988', '12/11/1992', '10/04/1991', '16/07/1995', '08/10/1992',
'11/10/1979'], 'Weight': [78, 80, 66, 67, 90, 57, 115, 87, 95, 57],
'Height': [176, 160, 169, 157, 185, 164, 195, 180, 174, 165], 'Kids': [2, 1, 0, 1, 1, 0,
2, 0, 3, 1] })
print(data)
```

1) Create two columns by separating the first and the last names of the employees
2) Compute the age of the employees and store as a separate column
3) Compute the average age of the employees in that organization.

Data concatenation & merging operations

- Two or more dataframes with identical features may need to be fused – *Concatenation operation; We will use concat()*
- *Often you may need to extract a column (feature) present in one dataframe A that is related to another dataset B and join this column (feature) to dataframe B. This is a merge() operation.*

Practice Problem 2.7 & 2.8

Demonstrate dataframes concatenation and merging operations; Show how you can access individual dataframes after concatenation/merging;

Data join() operations (Set arithmetic operations)

- How do we merge two DFs that have different column features and different sizes?

EXAMPLE

```
dfA = pd.DataFrame({'Name': [‘Mike’, ‘'Paul’,’ John’],
                    'Food': [‘fish’, ’beans’, ’bread’]},
                   columns= ([‘Name'', ’Food’])

dfB = pd.DataFrame({’’Name’:[’ John’, ‘Joseph’],
                    ‘Drink’’: [‘Wine’, ’Beer’]},
                   columns= ([‘Name'', ’Drink’])
```

Solution to this question can be thought of as a set intersection operation!

Question: What will John eat and drink?

Data join() operations (Set arithmetic operations)

- DIY!!! Try the following methods: Refer to slide #18

>> pd.merge(dfA, dfB, how='inner') # intersection operation

>> pd.merge(dfA, dfB, how='outer') # union operation

>> pd.merge(dfA, dfB, how='left')

>> pd.merge(dfA, dfB, how='right')

Print and see the results for yourself. *Under union operation, what do you observe?*

Annexure – A glimpse on Data Scientist role! ☺

Basic Multi-dimensional data analysis

Multi-dimensional data analysis is an informative analysis of data which takes many relationships into account.

Let us use a real-life multidimensional/multivariate data using open source libraries written in Python.

>> pip install **sklearn** # this is a useful ML package widely used

Clustering is one of the techniques to perform a MDD analysis. An example of clustering is to group customers w.r.t age, profession, interests, etc.

Note: It is beyond the scope of this module to go into ML techniques, but this attempt can serve as a first step to explore other techniques. Slides #37-end, Not for your exams! ☺

Basic Multi-dimensional data analysis

In ML, k-means clustering can be used to segment other data efficiently. This is one of the simplest *unsupervised machine learning* method.

Some quick facts about k-means clustering:

- Converges in a finite number of iterations.
- Computational cost of the k-means algorithm is O(k*n*d), where n is the number of data points, k the number of clusters, and d the number of attributes.
- Compared to other clustering methods, the k-means clustering technique is fast and efficient in terms of its computational cost.
- Difficult to predict an optimal number of clusters or the value of k. To find the number of clusters, we need to run the k-means clustering algorithm for a range of k values and compare the results.

Quick numerical example

Given data

	X1	X2
A	2	3
B	6	1
C	1	2
D	3	0

Step 1 – Choose 2 clusters; Say, AB and CD
Take the mean and use them as reference points

	X1	X2
AB	4	2
CD	2	1

Step 2: Calculate squared Euclidean distance between all data points to the centroids AB, CD. Note – Squared distance between A(2,3) and AB (4,2) is given by $s = (2–4)^2 + (3–2)^2$.

	A	B	C	D
AB	5	5	9	5
CD	4	16	2	2

Clusters formed are: **A,C, D** to CD cluster, **B** to AB cluster

Step 3 – Recompute the centroid coordinates based on clusters formed in Step 2: B and ACD

	X1	X2
B	6	1
ACD	2	1.67

Quick numerical example

Step 4 - As K-Means is an iterative procedure we have to calculate the distance of all points (A, B, C, D) to new centroids (B, ACD) similar to step 3.

	A	B	C	D
B	20	0	26	10
ACD	1.78	16.44	1.11	3.78

From the above result, we can see the assignment as:
Cluster ACD – points A,C,D and Cluster B – point B

All data points are assigned to clusters (B, ACD) based on their minimum distance. No shift in centroids positions.

Algorithm stops here for this example.

Basic Multi-dimensional data analysis

Samples from Iris dataset look like this:

sepal_length,	sepal_width,	petal_length,	petal_width,	species
5.1,	3.5,	1.4,	0.2,	setosa
7.1,	4.2,	1.8,	0.4,	virginica

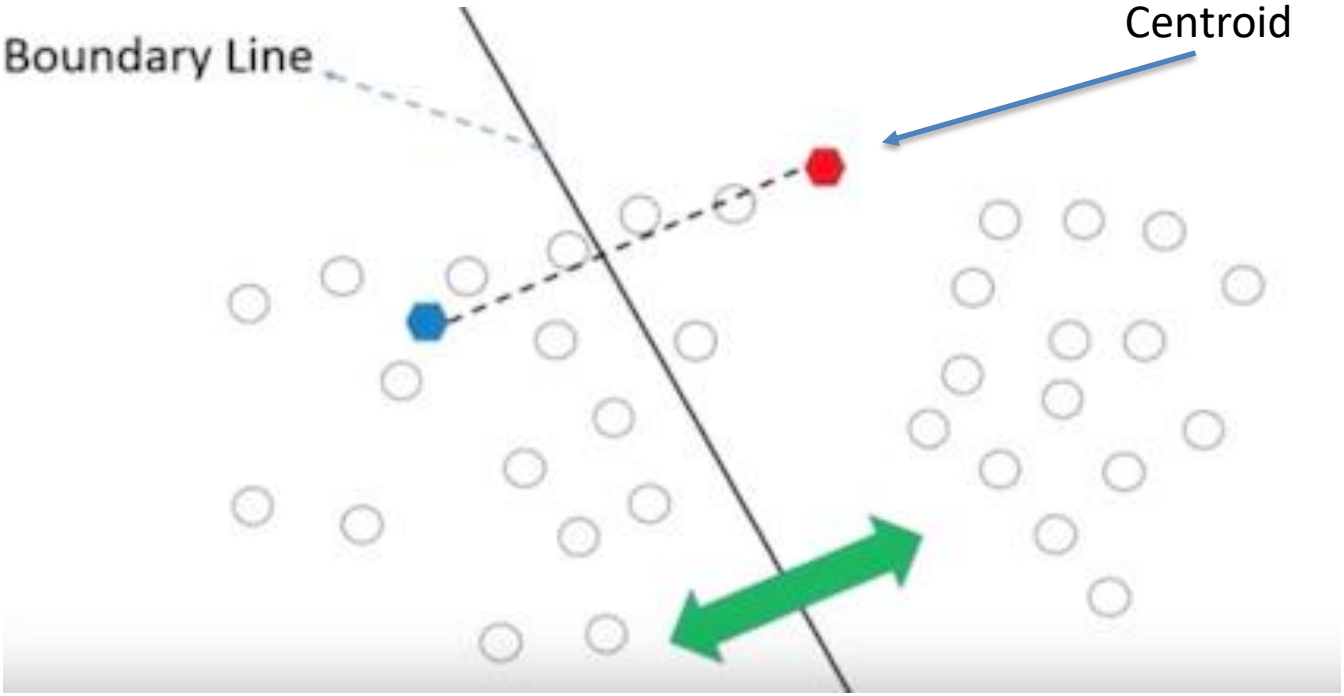
Annotation / labeled data

K-means works on unlabeled data; Following are the three steps used in the algorithm:

1) Initialization of cluster centers – called Centroids
2) All the data points that are the closest (similar) to a centroid will form a cluster.

Note that if we are using the Euclidean distance between data points and every centroid, a straight line is drawn between two centroids, then a perpendicular bisector (boundary line) divides this line into two clusters.

Basic Multi-dimensional data analysis



3) Move the centroid: A centroid's new value will be the mean of all the samples in a cluster. Iterate to determine the new place for the centroid and hence we can redefine the cluster. Process continues until there is hardly any further improvement.

Watch: https://www.youtube.com/watch?v=7Qv0cmJ6FsI

Basic Multi-dimensional data analysis

As the mean converges and settles down to a stable value, we say that the algorithm has converged.

Practice Problem 2.8

Use the Iris dataset given to you and perform K-means clustering. The dataset has 150 samples and 4 features. In addition, the dataset is also annotated/labeled.

You will need: Numpy, Pandas, sklearn packages

Happy Clustering! ☺

Thank you!