COMP4616 – Fundamentals of Machine Learning

ASST. PROF. TUĞBA ERKOÇ

SPRING 2025

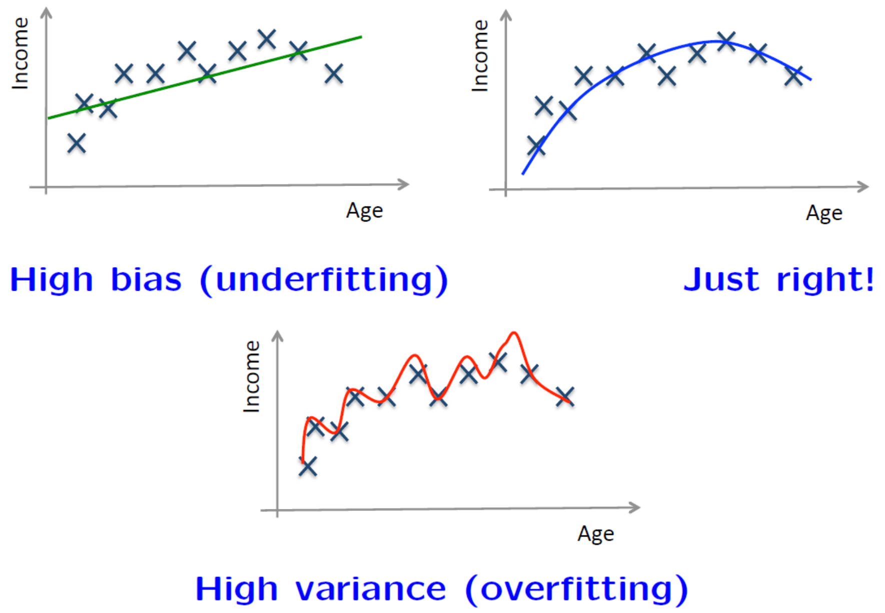
WEEK 3: DECISION TREES

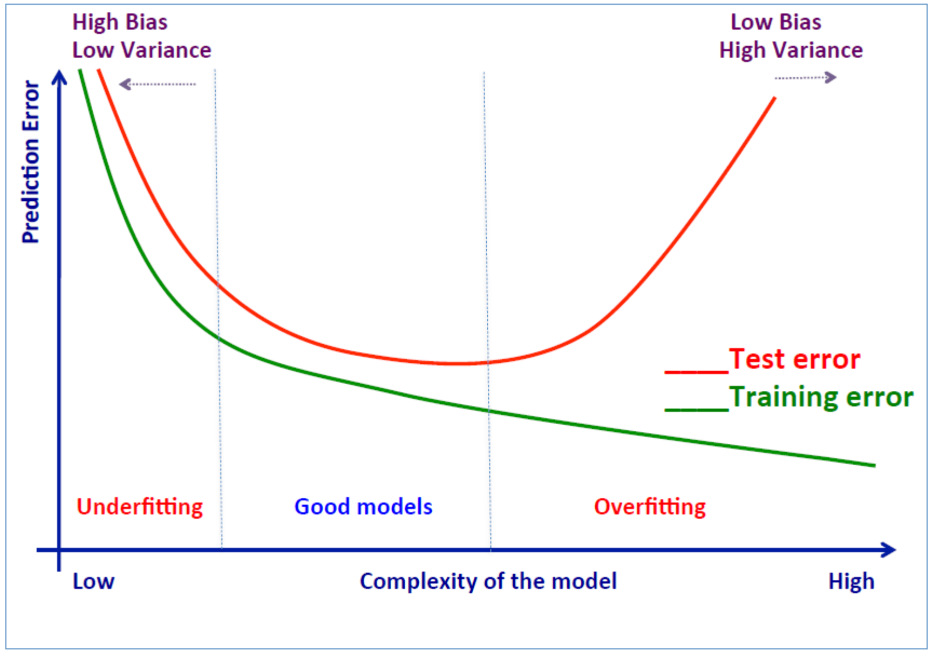
Generalization and Overfitting

- We want our learning algorithms to find a hypothesis that fits the training data, and we want it to generalize well for previously unseen data.
- We can evaluate the hypothesis with a second set of (x_i, y_i) pairs called **test set**.
- We say that h **generalizes** well if it accurately predicts the outputs of the test set.
- We say a function is **overfitting** the data when it pays too much attention to the particular data set it is trained on, causing it to perform poorly on unseen data.
- We say that a hypothesis is **underfitting** when it fails to find a pattern in the data.
- **Bias** is the difference between the average prediction of our model and the correct value which we are trying to predict. A model with high bias pays very little attention to the training data and oversimplifies the model.
- **Variance** is the variability of model prediction for a given data point or a value that tells us the spread of our data. A model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn’t seen before.

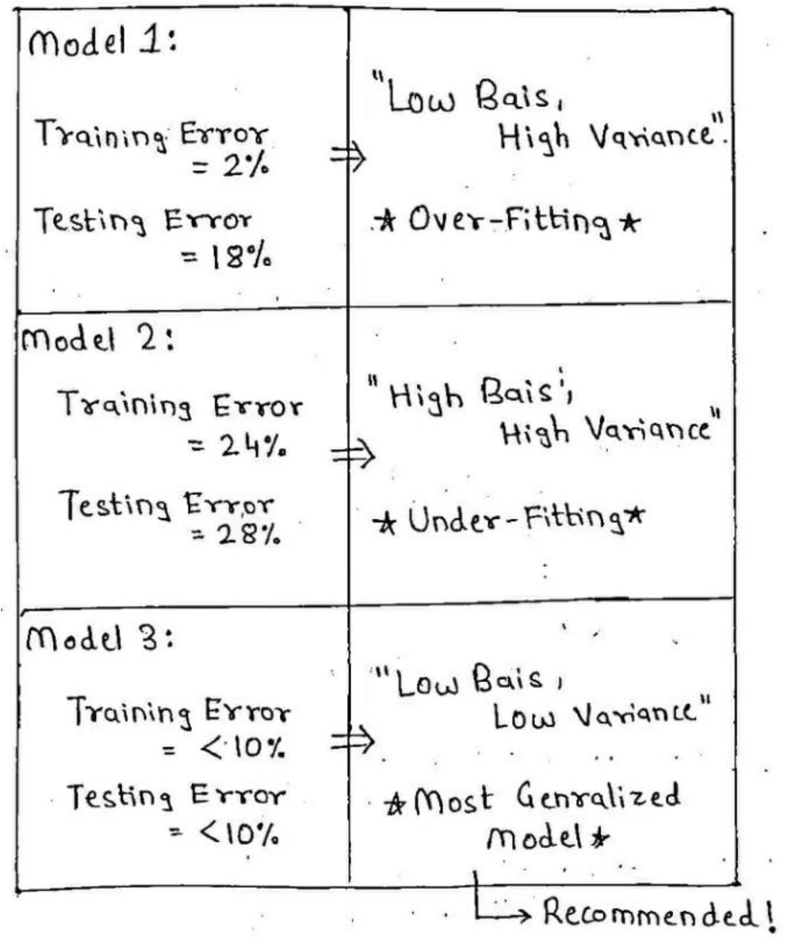
Generalization and Overfitting

- Overfitting becomes more likely as the number of attributes grows, and less likely as we increase the number of training examples.
- Larger hypothesis spaces (e.g., decision trees with more nodes) have more capacity both to fit and to overfit.





Generalization and Overfitting

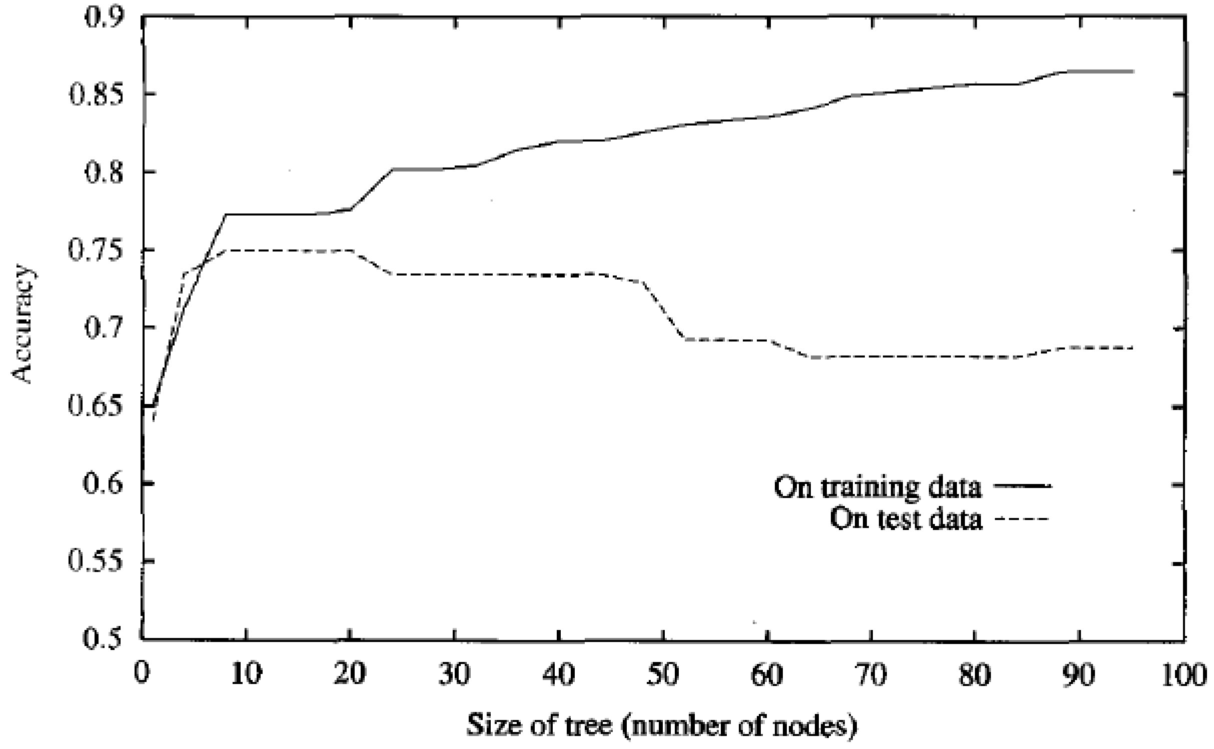


Model 1: Training Error = 2% Testing Error = 18%	⇒ "Low Bais, High Variance". ★ Over-Fitting ★
Model 2: Training Error = 24% Testing Error = 28%	⇒ "High Bais", High Variance" ★ Under-Fitting★
Model 3: Training Error = <10% Testing Error = <10%	⇒ "Low Bais, Low Variance" ★Most Genralized Model★

→ Recommended!

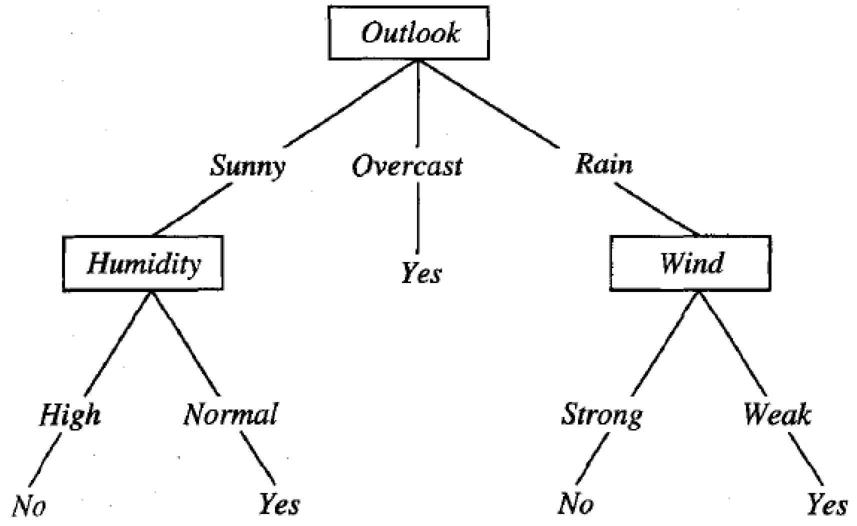
Generalization and Overfitting

- Impact of overfitting in a typical application of decision tree learning.
  - the accuracy of the tree over the training examples increases monotonically as the tree is grown
  - the accuracy measured over the independent test examples first increases, then decreases
- When the tree size exceeds approximately 25 nodes, further elaboration of the tree decreases its accuracy over the test examples despite increasing its accuracy on the training examples



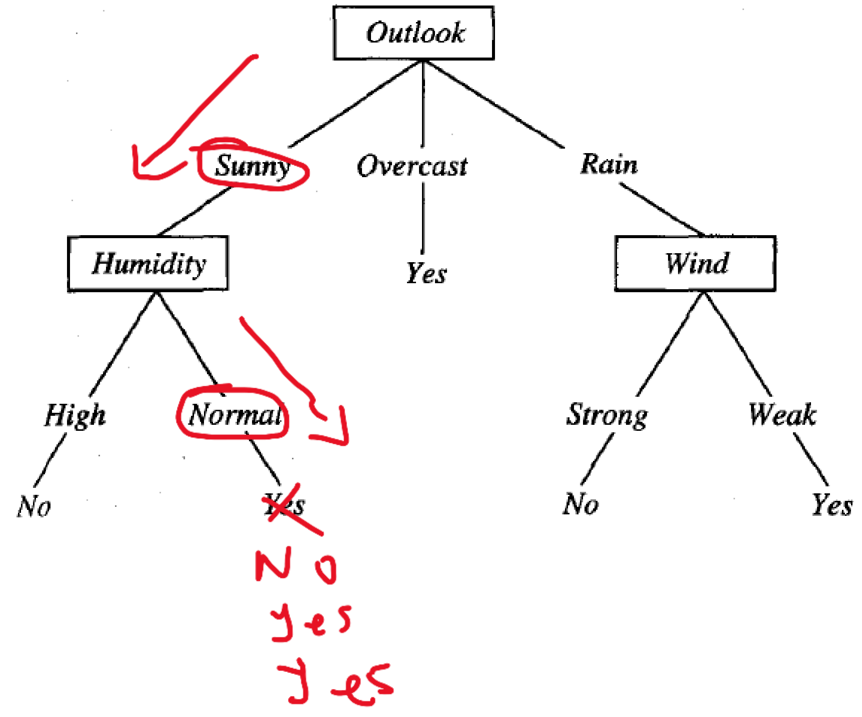
Generalization and Overfitting

- Why this happens?
  - It can occur when the training examples contain random errors or noise
- Let’s consider the PlayTennis example.
- Assume that we have this example with incorrectly labelled as negative:
  - $\langle Outlook = Sunny, Temperature = Hot, Humidity = Normal, Wind = Strong, PlayTennis = No \rangle$
- Without this error, we would produce the tree on the right side.
- The addition of this incorrect example will now cause ID3 to construct a more complex tree.



Generalization and Overfitting

- The new example will be sorted into the second leaf node from the left in the learned tree along with the previous positive examples D9 and D11.
- Because the new example is incorrectly labeled as a negative example, ID3 will search for further refinements to the tree below this node.
- ID3 will split this node further via an appropriate attribute, so we will get more complex tree.
- In one experimental study of ID3 involving five different learning tasks with noisy, nondeterministic data (Mingers 1989b), overfitting was found to decrease the accuracy of learned decision trees by 10-25% on most problems



Generalization and Overfitting

- There are several approaches to avoiding overfitting in decision tree learning. These can be grouped into two classes:
  - approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,
  - approaches that allow the tree to overfit the data, and then post-prune the tree.
- Although the first of these approaches might seem more direct, the second approach of post-pruning overfit trees has been found to be more successful in practice.
  - This is due to the difficulty in the first approach of estimating precisely when to stop growing the tree.

Generalization and Overfitting

- Regardless of whether the correct tree size is found by stopping early or by post-pruning, a key question is **what criterion is to be used to determine the correct final tree size**.

1. Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree.
2. Use all the available data for training, but apply a **statistical test** to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set.
   - Quinlan (1986) uses a **chi-square** test to estimate whether further expanding a node is likely to improve performance over the entire instance distribution, or only on the current sample of training data.
3. Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized.

Generalization and Overfitting

- **Reduced-error pruning** considers each of the decision nodes in the tree to be candidates for pruning.
- Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node.
- Nodes are removed only if the resulting pruned tree performs no worse than the original over the validation set.
- Nodes are pruned iteratively, always choosing the node whose removal most increases the decision tree accuracy over the validation set.
- Pruning of nodes continues until further pruning is harmful (i.e., decreases accuracy of the tree over the validation set).
- Using a separate set of data to guide pruning is an effective approach provided a large amount of data is available.
- The major drawback of this approach is that when data is limited, withholding part of it for the validation set reduces even further the number of examples available for training.

Generalization and Overfitting

- **Rule Post Pruning**:
  - Start with a full tree, as generated by ID3 algorithm allowing overfitting to occur
  - Look at a test node that has only leaf nodes as descendants
  - If the test appears to be irrelevant (i.e. detecting only noise in the data) then eliminate the test, replacing it with a leaf node.
- Repeat this process, considering each test with only leaf descendants, until each one has either been pruned or accepted as is.

Generalization and Overfitting

- A **low information gain** is a good clue that the attribute is **irrelevant**.
- Now the question is, how large a gain should we require in order to split on a particular attribute?
- Use statistical **significance test**.
  - Test begins by assuming that there is no underlying pattern (i.e. null hypothesis)
  - Actual data are analyzed to calculate the extent to which they deviate from a perfect absence of pattern
  - If the degree of deviation is **statistically unlikely** (5% probability or less), then that is considered to be good evidence for the presence of a **significant pattern** in the data

Generalization and Overfitting

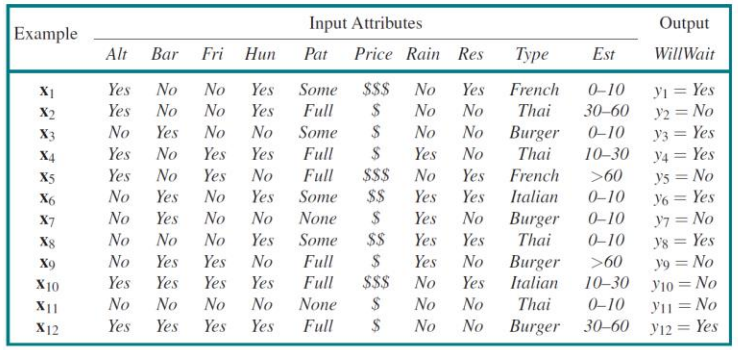
- The null hypothesis is that the attribute is irrelevant and, hence, that the information gain for an infinitely large sample would be zero.
- We need to calculate the probability that, under the null hypothesis, a sample of size $v = n + p$ would exhibit the observed deviation from the expected distribution of positive and negative examples.
- The total deviation is given by:

# of expected values

# of actual pos/neg values

$$\hat{p}_k = p \times \frac{p_k + n_k}{p + n} \qquad \hat{n}_k = n \times \frac{p_k + n_k}{p + n}$$

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$



Example	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
	Input Attributes										Output
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 19.2 Examples for the restaurant domain.

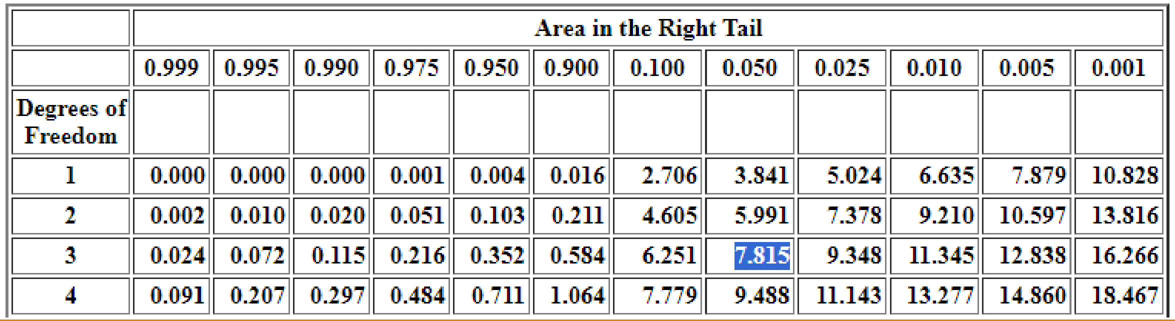
Generalization and Overfitting

Under the null hypothesis, the value of Δ is distributed according to the χ^2 (chi-squared) distribution with d − 1 degrees of freedom.

In short, we can use a χ^2 statistics function to see if a particular Δ value confirms or rejects the null hypothesis.

Example: consider the restaurant $Type$ attribute, with four values and thus three degrees of freedom.

- If Δ >= 7.82 reject null hypothesis at 5% level
- Else accept the null hypothesis that the attribute is irrelevant, thus the associated branch of the tree should be pruned away.
- This is called χ^2 **pruning**.



	Area in the Right Tail											
	0.999	0.995	0.990	0.975	0.950	0.900	0.100	0.050	0.025	0.010	0.005	0.001
Degrees of Freedom												
1	0.000	0.000	0.000	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879	10.828
2	0.002	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597	13.816
3	0.024	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838	16.266
4	0.091	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860	18.467

Python

Python - Basics

- In other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.
- Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.
- Variables do not need to be declared with any particular type, and can even change type after they have been set
- Variable names are case-sensitive.
- Comments start with a #, and Python will render the rest of the line as a comment or you can add a multiline string in your code by using triple quotes(“””)

Python - Numpy

- NumPy is a Python library used for working with arrays.
- Lists serve the purpose of arrays, but they are slow to process.
- NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
- The array object in NumPy is called **ndarray**, it provides a lot of supporting functions that make working with **ndarray** very easy.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
- The indexes in NumPy arrays start with 0.

Python - Pandas

- **Pandas** is a Python library used for working with data sets
- Import Pandas in your applications by adding the **import** keyword
- Data sets in Pandas are usually multi-dimensional tables, called **DataFrame**s.
- If your data sets are stored in a file, Pandas can load them into a **DataFrame**.
- The **head()** method returns the headers and a specified number of rows, starting from the top.