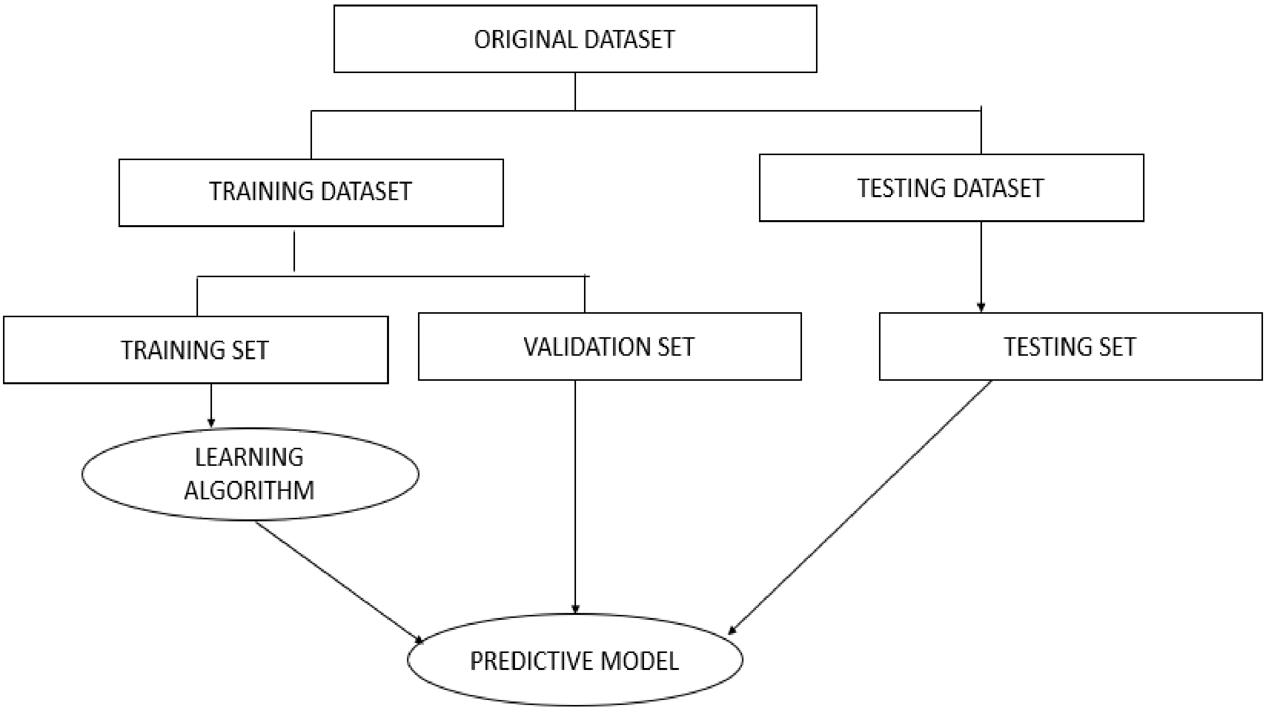
MODULE 2

TRAINING NEURAL NETWOK

DATA SET

- The dataset contains information for creating our model.
- It is a collection of data structured as a table in rows and columns.
- It is a collection of data arranged in some order.
- Dataset can be
    1) Numerical dataset (eg : temp ,prices)
    2) Categorical dataset (eg : Y/N, T/F)
    3) Odinal dataset (similar to categorical)
- We can represent dataset as table or marix or series of arrays.
- The data is usually stored in a data file. Data file include csv. ,excel. etc
- Dataset for the ML/DL application is divided into two:
    a) Training dataset
    b) Testing dataset



- TRAINING SET

- **Subset** of original dataset
- used **to learn the data’s behavior** or model

- The model learns the data and evaluates the **data repeatedly** to learn more about the data’s behavior and then adjust itself.
- Training dataset is used to fit the model.
- It trains images and labels
- Here, it adjusts weights and bias values by looking at the training data
- Accuracy is checked and reported after each iteration .(each iteration is loaded as batch) (aim to make error=0 and predict correct output)
- In each batch ,bias and weight value is adjusted . then check the accuracy and then again adjust the weight and bias in the next batch. The process is repeated until we get the correct predicted output(ie, error=0)

➢ VALIDATION SET

- Evaluate the model during training
- Accuracy gets checked and reported after each epoch(period of time)
- Here, the **model does not update any weight or bias value**
- ML/DL engineer monitor the validation error and tunes or adjust **the hyper parameter** to optimize the model for better performance.

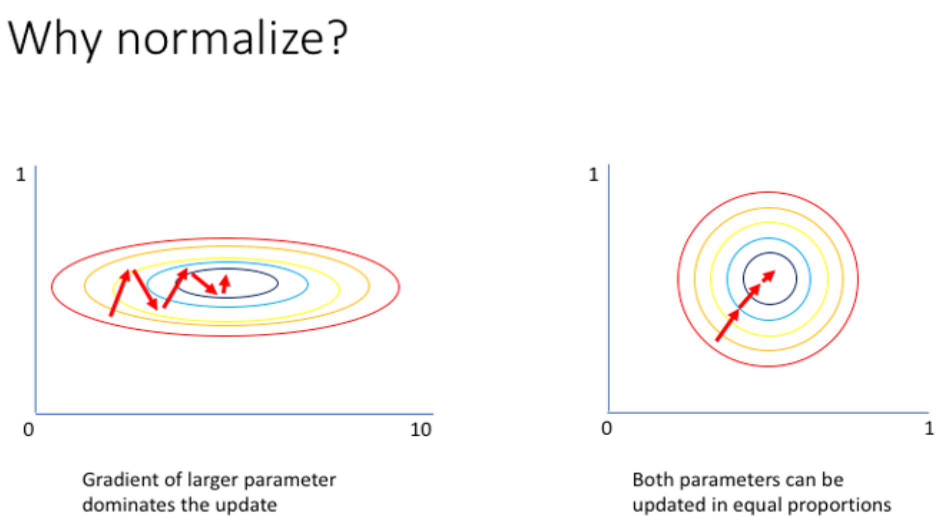
➢ TESTING SET

- Test data is used to evaluate the final trained model.
- It checks the accuracy after the entire training is done.
- It provides a final real world check of an unseen dataset to confirm that the DL/ML/ algorithm was trained affectively.

NORMALIZATION IN DL

- In a deep neural network, there is a phenomenon called **internal covariate shift**,
- which is a **change in the input distribution** to the network's layers due to the ever-changing network parameters during training.
- The input layer may have certain features which dominate the process, due to having high numerical values.
- This can create a bias in the network because only those **features contribute to the outcome of the training.**

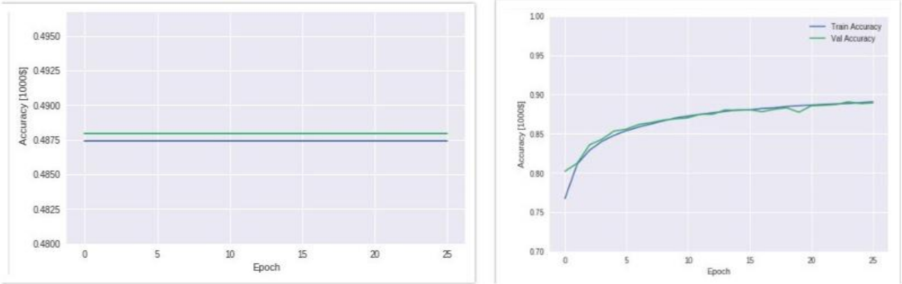
- For example, imagine feature 1 having values between 1 and 5, and feature 2 having values between 100 and 10000.
- During training, due to the difference in scale of both features, feature 2 would **dominate the network** and only that feature would have a contribution to the outcome of the model.
- Due to the reasons stated, a concept known as **normalization** was introduced.


Why normalize?
1
0
10
Gradient of larger parameter dominates the update
1
0
1
Both parameters can be updated in equal proportions

- When the features in the data have **different ranges**, Normalization is an approach used during data processing, to adjust the values of numeric columns in a dataset to a **similar scale**.
- Normalization has a lot of advantages, which includes

- **Reducing the internal covariate shift** to improve training
- Scaling each feature to a similar range to **prevent or reduce bias** in the network
- **Speeding up the optimization process** by preventing weights from exploding all over the place and limiting them to a specific range
- **Reducing overfitting** in the network by aiding in regularization

Left: Model Accuracy, without normalized data
Right: Model Accuracy with normalized data



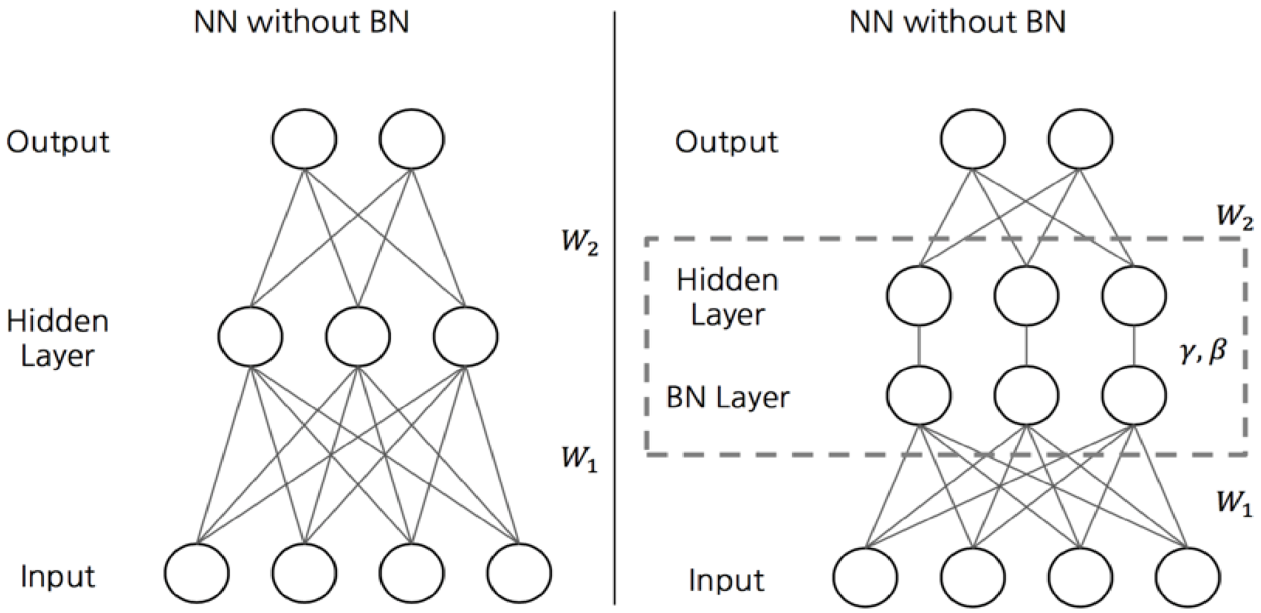
- From the figure above,
- we can see the benefits of normalization in action.
- Left : The **accuracy of the model is not improving** after every epoch because there is a big disparity in the numerical values of the features,
- Right : but when the features are normalized, **the accuracy tends to go high** after every epoch in response.

- **TYPES OF NORMALIZATION**

1. Batch normalization
2. Layer normalization
3. Instance normalization
4. Group normalization

BATCH NORMALIZATION

- Batch normalization is the most common form of normalization in deep learning.
- **It scales the inputs to a layer to a common value** for every **mini-batch** during the training of deep neural networks.
- This stabilizes the learning process and significantly reduces the number of epochs required to train deep networks, enabling the network to train faster.
- The way batch normalization works are by calculating the **mean** and **variance** of every feature in the mini-batch, then the mean is subtracted, and each feature is divided by the **standard deviation** of the mini-batch.
- The technique consists of **adding an operation** in the model just before the activation function of each layer, **simply zero-centering and normalizing the inputs**,
- Then **scaling and shifting** the result using two new parameters per layer (one for scaling, the other for shifting).
- In other words, this operation lets the model learn the optimal scale and mean of the inputs for each layer.



$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$ → mini-batch mean

$$\sigma^2{}_B = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$ → mini-batch variance

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma^2{}_B + \epsilon}}$$ → normalize

$$y_i = \gamma\hat{x}_i + \beta \equiv BN_{y,b}(x_i)$$ → scale and shift

μB is the empirical **mean**, evaluated over the whole mini-batch B.

σB is the empirical **standard deviation**, also evaluated over the whole mini-batch.

m is the number of instances in the mini-batch.

x(i) is the zero-centered and normalized input.

γ is the **scaling** parameter for the layer.

β is the shifting parameter (**offset**) for the layer.

ϵ is a tiny number to avoid division by zero (typically 10–5). This is called a smoothing term.

y(i) is the output of the BN operation: it is a scaled and shifted version of the inputs.

- ***At test time***, there is **no mini-batch** to compute the empirical mean and standard deviation,
- instead, simply use **the whole training set's** mean and standard deviation.
- These are typically efficiently computed during training using a moving average.
- So, in total, **four** parameters are learned for each batch normalized layer: γ (scale), β (offset), μ (mean), and σ (standard deviation).
- Batch Normalization does, however, **add some complexity** to the model (although it removes the need for normalizing the input data since the first hidden layer will take care of that, provided it is batch normalized).
- Moreover, there is a runtime penalty: the neural network makes **slower predictions** due to the extra computations required at each layer.

OVERFITTING

- Deep neural networks deal with a multitude of parameters for training and testing. With the increase in the number of parameters, neural networks have the freedom to fit multiple types of datasets which is what makes them so powerful.

- But, sometimes this power is what makes the neural network weak. The networks often lose control over the learning process and the model tries to **memorize each of the data points** causing it to perform **well on training data** but **poorly on the test dataset**. This is called **overfitting**.

- Overfitting occurs when the model tries to make predictions on data that is very **noisy**. A model that is overfitted is **inaccurate** because the trend does not reflect the reality present in the data.

- When the network tries to learn **too much or too many details in the training data along with the noise** from the training data which results in poor performance on unseen or test dataset.

- When this happens the **network fails to generalize the features/pattern** found in the training data.

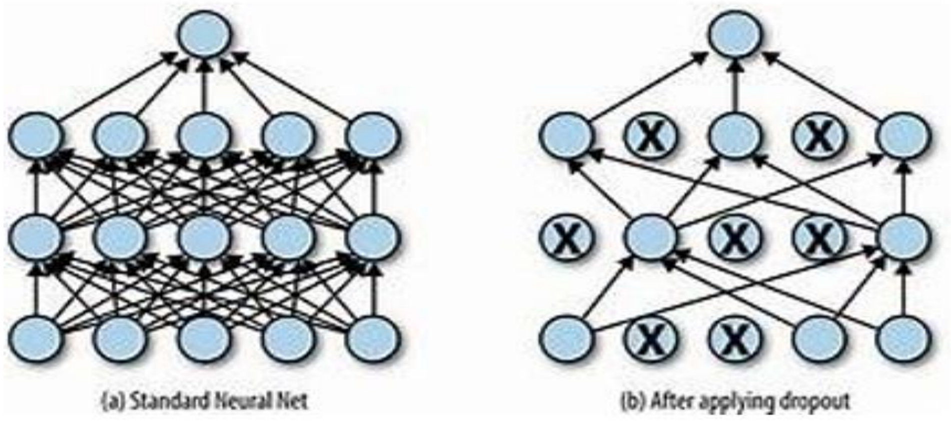
- Overfitting during training can be spotted when the error on training data decreases to a very small value but the error on the new data or test data increases to a large value.

- One of the main reasons for the network to overfit is if the **size of the training dataset is small.** When the network tries to learn from a small dataset it will tend to have greater control over the dataset & will make sure to satisfy all the datapoints exactly.

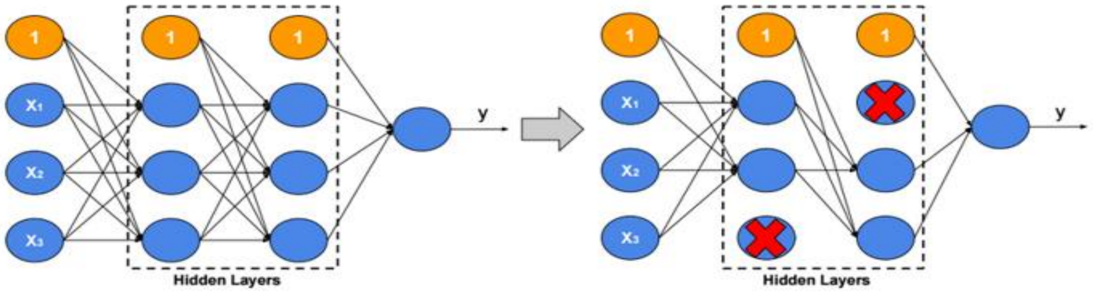
- It can be thought of as the network trying to memorize every single datapoint failing to capture the general trend in the data.

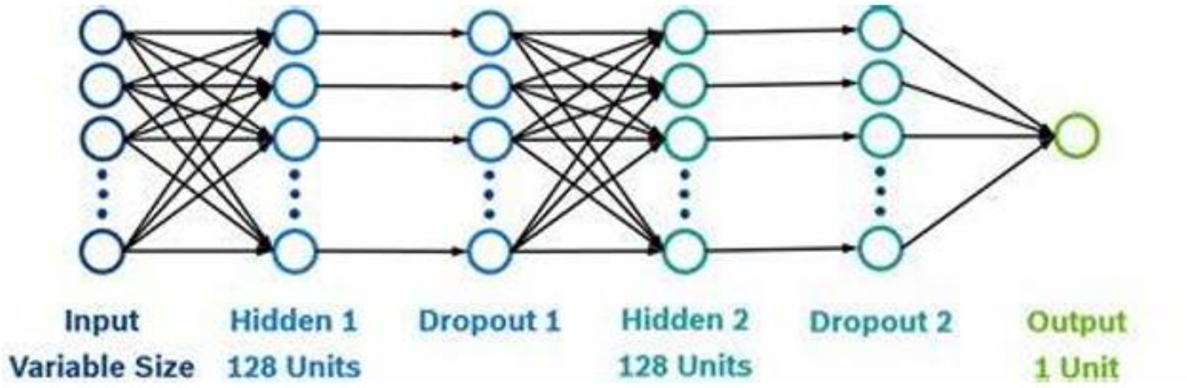
DROPOUT LAYER

- Dropout Layer is one of the most popular **regularization techniques to reduce overfitting** in the deep learning models.
- Overfitting in the model occurs when it shows more accuracy on the training data but less accuracy on the test data **or unseen data.**
- A first "deep" regularization technique is **dropout** .
- It consists of **removing** units at random during the forward pass on each sample and **putting them all back during test**.
- A key idea in deep learning is to engineer architectures to make them easier to train. So far, we saw that we can choose the architecture (number of layers, units, filters, filter sizes, etc.), the activation function(s), and the parameter initialization.
- We can go one step further by adding mechanisms specifically designed to facilitate the training, such as "dropout".
- As pictured on the right it removes at random some of the units during the forward pass. The units to remove are selected at random independently for every sample. **The backward pass is done consistently**, i.e. through the kept units alone.



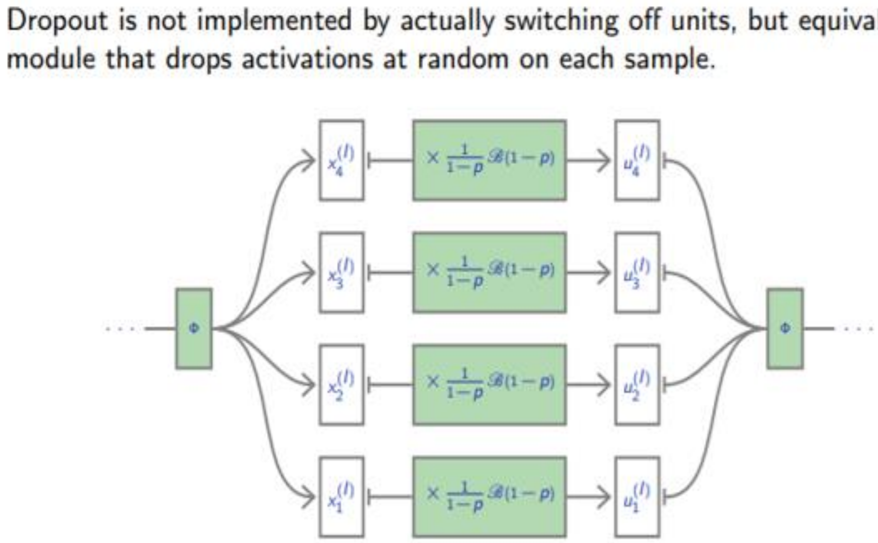
- There are many ways to regularize a machine learning algorithm, but for deep learning, one of the most common is by using dropout layers. This idea was introduced by **Geoffrey Hinton in 2012.**
- Dropout layers are very **simple**.
- During training, each dropout layer chooses a random set of units from the preceding layer and **sets their output to zero**.





- This simple addition drastically reduces overfitting, by ensuring that the network **doesn't** become **overdependent** on certain units or groups of units
- Just remember observations from the training set.
- If we use dropout layers, the **network cannot rely too much on any one unit** and therefore knowledge is more evenly spread across the whole network.
- This makes the model much better at generalizing to unseen data, because the network has been trained to produce accurate predictions even under **unfamiliar conditions**, such as those caused by dropping random units.

- There are **no weights to learn within a dropout layer**, as the units to drop are decided stochastically.
- At test time, the dropout layer doesn’t drop any units, so that the full network is used to make predictions.
- The Dropout layer in Keras implements this functionality, with the rate parameter specifying the proportion of units to drop from the preceding layer: Dropout(rate = 0.25)
- Dropout layers are used most after **Dense layers** since these are most prone to overfitting due to the **higher number of weights**, though you can also use them after convolutional layers.
- Dense layers are layers in neural network that is deeply connected with its preceding layer. It helps in **changing the dimentionality** of the output from the preceding layer.
- **Batch normalization** also has been shown to reduce overfitting, and therefore many modern deep learning architectures don’t use dropout at all and rely solely on batch normalization for regularization.
- Dropout is not implemented by actually switching off units, but equivalently as a module that drops activations at random on each sample.



Dropout is not implemented by actually switching off units, but equivalently as a module that drops activations at random on each sample.

Φ
$x_4^{(l)}$
$\times \frac{1}{1-p}\mathcal{B}(1-p)$
$u_4^{(l)}$
$x_3^{(l)}$
$\times \frac{1}{1-p}\mathcal{B}(1-p)$
$u_3^{(l)}$
$x_2^{(l)}$
$\times \frac{1}{1-p}\mathcal{B}(1-p)$
$u_2^{(l)}$
$x_1^{(l)}$
$\times \frac{1}{1-p}\mathcal{B}(1-p)$
$u_1^{(l)}$
Φ

- One has to **decide on which units/layers to use dropout**, and with what probability p units are dropped.
- During training, for each sample, as many Bernoulli variables as units are sampled independently to select units to remove.

- Let X be a unit activation, and D be an independent Boolean random variable of probability 1 – p.
- We have **E(D X) = E(D) E(X) = (1 – p)E(X)**
- To keep the means of the inputs to layers unchanged, the initial version of dropout was multiplying activations by **1 – p during test**.
- The standard variant in use is the "**inverted dropout**". It multiplies activations by 1 1–p during train and keeps the network untouched during test.
- To keep the means of the inputs to layers unchanged, the initial version of dropout was multiplying activations by 1 – p during test.
- Dropout is implemented in PyTorch as nn.DropOut, which is a torch.Module.
- **Default probability to drop is p = 0.5**, but other values can be specified.

UNDERFITTING

- A statistical model or a machine learning algorithm is said to have underfitting when it **cannot capture the underlying trend of the data.** *(It's just like trying to fit undersized pants!).*
- Underfitting **destroys the accuracy** of our machine learning model.
- Its occurrence simply means that our model or the algorithm **does not fit the data** well enough.
- It usually happens when we **have fewer data to build an accurate model** and also when we try to build a linear model with fewer non-linear data.
- In a nutshell, **Underfitting – High bias and low variance**

Techniques to reduce underfitting**:**

1. Increase model complexity
2. Increase the number of features, performing feature engineering
3. Remove noise from the data.
4. Increase the number of epochs or increase the duration of training to get better results.

- Underfitting happens when the network can neither model the training or test data which results in overall bad performance.
- A model is said to be underfitting when it's not able to classify the data it was trained on.
- We can tell that a model is underfitting when the metrics given for the training data are poor, meaning that the training accuracy of the model is low and/or the training loss is high.

MODEL BASICS

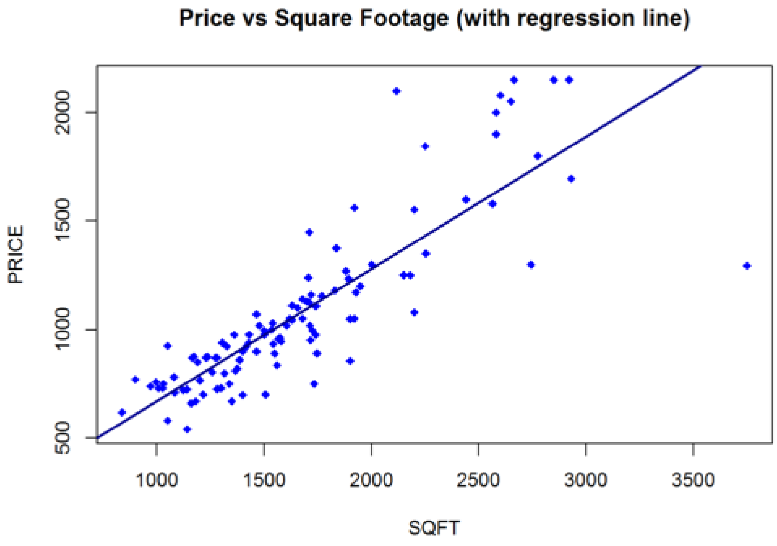
what is a model?

- A model is simply a **system for mapping inputs to outputs**.
- For example, if we want to predict house prices, we could make a model that takes in the square footage of a house and outputs a price.
- A model **represents a theory about a problem**: there is some connection between the square footage and the price and we make a model to learn that relationship.
- Models are useful because we can use them to predict the values of outputs for new data points given the inputs.
- A model learns relationships between the inputs, called **features**, and outputs, called **labels**, from a training dataset.
- During training, the model is **given both the features and the labels** and learns how to map the former to the latter.
- A trained model is evaluated on a **testing set, where we only give it the features and it makes predictions.**
- We compare the predictions with the known labels for the testing set to calculate accuracy.
- Models can take many shapes, from simple linear regressions to deep neural networks, but all supervised models are based on the fundamental idea of learning relationships between inputs and outputs from training data.

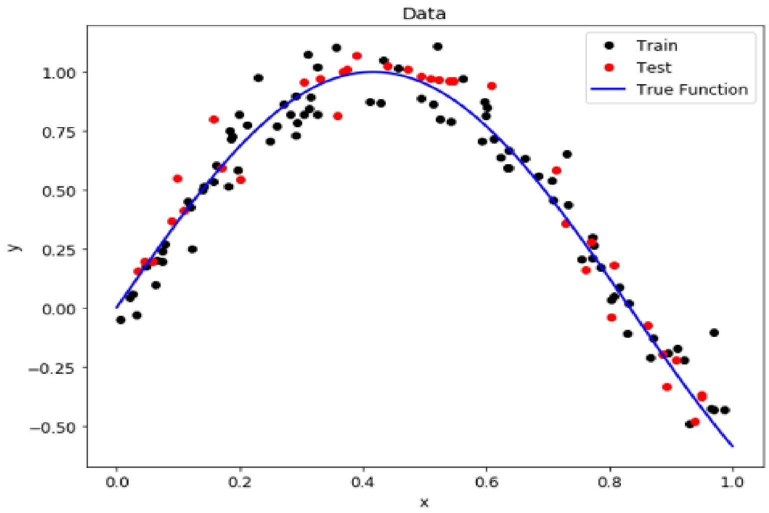
<u>Training and Testing Data</u>

- To make a model, we first need data that has an underlying relationship.

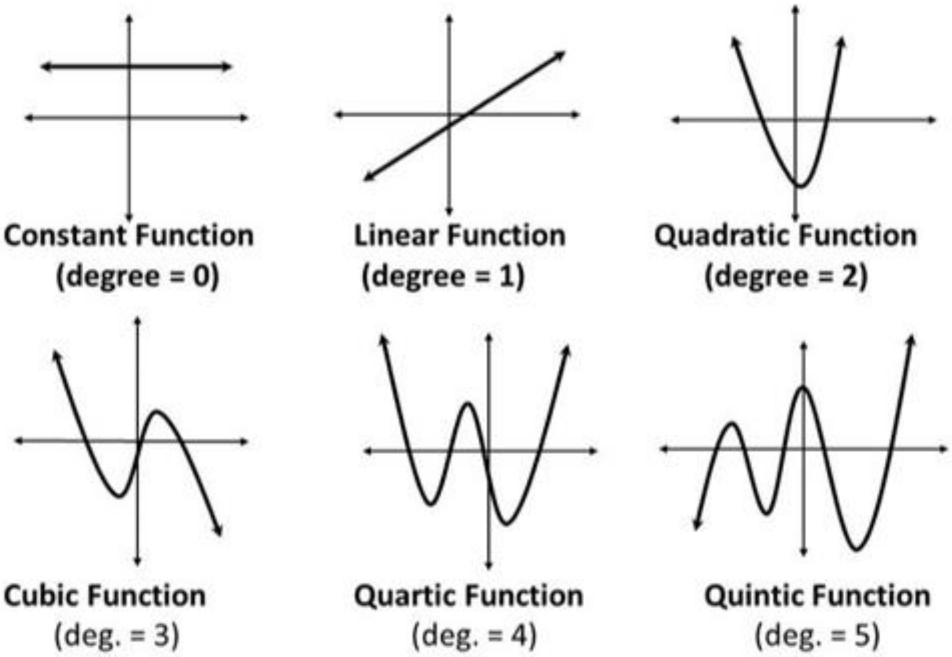
- For this example, we will create our own simple dataset **with x-values (features) and y-values (labels).**
- An important part of our data generation is **adding random noise** to the labels.
- In any real-world process, whether natural or man-made, the data does not exactly fit to a trend.
- There is always noise or other variables in the relationship we cannot measure.
- In the house price example, the trend between area and price is linear, but the prices do not lie exactly on a line because of other factors influencing house prices.
- Eg



- Our data similarly has a trend (which we call the true function) and random noise to make it more realistic.
- After creating the data, we split it into random training and testing sets. The model will attempt to learn the relationship on the training data and be evaluated on the test data.
- In this case, **70% of the data is used for training and 30% for testing.** The graph shows the data we will explore.
- We can see that our data are distributed with some variation around the true function (a partial sine wave) because of the random noise we added.
- During training, we want our model to learn the true function without being “distracted” by the noise.



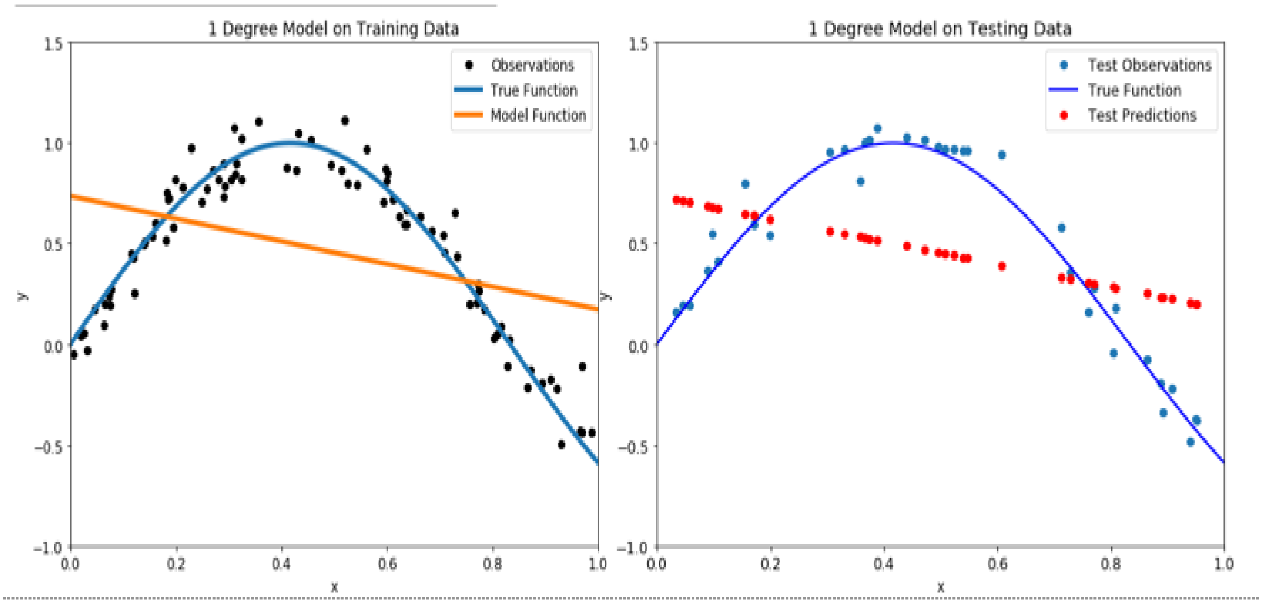
Polynomials of varying degree



- The degree represents how much flexibility is in the model, with a higher power allowing the model freedom to hit as many data points as possible.

UNDERFITTING:

- An underfit model will be less flexible and cannot account for the data.
- EG:--an underfit model with a 1 degree polynomial fit. In the image on the left, model function in orange is shown on top of the true function and the training observations. On the right, the model predictions for the testing data are shown compared to the true function and testing data points.



- Our model passes straight through the training set with no regard for the data!
- This is because an **underfit model has low variance and high bias.**
- **Variance** refers to how much the model is dependent on the training data.
- For the case of a 1 degree polynomial, the model depends very little on the training data because it barely pays any attention to the points.
- Instead, the model has high **bias**, which means it makes a strong assumption about the data.
- For this example, the assumption is that the data is linear, which is evidently quite wrong. When the model makes test predictions, the bias leads it to make inaccurate estimates.
- The model failed to learn the relationship between x and y because of this bias.
- **A low degree leads to underfitting.**
- A natural conclusion would be to learn the training data, we should **just increase the degree of the model to capture every change in the data**.

TENSORFLOW

➢ **TensorFlow** is **a free and open-source software library** for machine learning and artificial intelligence.

➢ It can be used across a range of tasks but has a particular focus **on training and inference of deep neural networks**.

- TensorFlow was developed by the **Google Brain team** for internal Google use in research and production.
- TensorFlow is Google Brain's second-generation system.
- It can be used in a **wide variety of programming languag**es, most notably Python, as well as Javascript, C++, and Java.
- TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.
- TensorFlow serves as the core platform and library for machine learning.
- TensorFlow’s APIs use **Keras** to allow users to make their own machine learning models.

History of TensorFlow

- A couple of years ago, deep learning started to outperform all other machine learning algorithms when giving a massive amount of data. Google saw it could use these deep neural networks to improve its services:
- Gmail
- Photo
- Google search engine
- They build a framework called **Tensorflow** to let researchers and developers work together on an **AI model**. Once developed and scaled, it allows lots of people to use it.
- It was first made public in late **2015**, while the first stable version appeared in 2017.
- It is open source under **Apache Open Source license**.
- You can use it, modify it and redistribute the modified version for a fee without paying anything to Google.
- In addition to **building** and **training** their model, TensorFlow can also **help load the data to train the model**, and deploy it using TensorFlow Serving.
- is an open-source end-to-end platform for creating Machine Learning applications.
- It is a symbolic math library that uses dataflow and differentiable programming to perform various tasks focused on training and inference of deep neural networks.
- It allows developers to create machine learning applications using various tools, libraries, and community resources.
- **Currently, the most famous deep learning library in the world is Google’s TensorFlow.**

How TensorFlow Works???

➢ TensorFlow enables you to build **dataflow graphs and structures** to define how data moves through a graph by **taking inputs as a multi-dimensional array called Tensor**.

➢ It allows you to **construct a flowchart of operations** that can be performed on these inputs, which goes at one end and comes at the other end as output.

TensorFlow Architecture

Tensorflow architecture works in three parts:

1. Preprocessing the data
2. Build the model
3. Train and estimate the model

→It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**.

- We can construct a sort of **flowchart** of operations (called a Graph) that we want to perform on that input.
- The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.
- This is why it is called TensorFlow because **the tensor goes in it flows through a list of operations, and then it comes out the other side**.

Where can Tensorflow run?

- TensorFlow hardware, and software requirements can be classified into

  1) **Development Phase**: This is when you **train** the mode. Training is usually done on your **Desktop or laptop**.

  2) **Run Phase or Inference Phase**: Once training is done Tensorflow can be run on many different platforms.
- You can run it on: Desktop running Windows, macOS or Linux
- Cloud as a web service
- Mobile devices like iOS and Android
- You **can train it on multiple machines** then you **can run it on a different machine, once you have the trained model.**

- TensorFlow is a library developed by the **Google Brain Team** to accelerate machine learning and deep neural network research.

TensorFlow Components

1. Tensor

- Tensorflow’s name is directly derived from its core framework: **Tensor**.
- In Tensorflow, all the computations involve **tensors**.
- **A tensor is a vector or matrix of n-dimensions that represents all types of data**.
- All values in a tensor hold **identical data type** with a known (or partially known) **shape**. The shape of the data is the dimensionality of the matrix or array.
- A tensor can be originated from the input data or the result of a computation.
- In TensorFlow, all the operations are conducted inside a **graph**.
- The graph is a set of computation that takes place successively. Each operation is called an **op node** and are connected to each other.
- The graph outlines the ops and connections between the nodes. However, it does not display the values. The **edge** of the nodes is the tensor, i.e., a way to populate the operation with data.

2. Graphs

- TensorFlow makes use of a graph framework. The graph gathers and describes all the series computations done during the training. The graph has lots of advantages:
- It was done to run on **multiple CPUs or GPUs** and even mobile operating system
- The **portability** of the graph allows to preserve the computations for immediate or **later** use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by **connecting tensors together**
  - A tensor has **a node and an edge.** The node carries the mathematical operation and produces an endpoints outputs. The edges explain the input/output relationships between nodes.

CPU AND GPU

Central Processing Unit (CPU):

- CPU is known as **brain** for every ingrained system.

- CPU comprises **the arithmetic logic unit** (ALU) accustomed quickly to store the information and perform calculations and **Control Unit** (CU) for performing instruction sequencing as well as branching.
- CPU interacts with more computer components such as memory, input and output for performing instruction.

Graphics Processing Unit (GPU):

- GPU is used to provide the **images** in computer games.
- GPU is **faster** than CPU's speed and it emphasis on **high throughput**.
- It's generally incorporated with electronic equipment for sharing RAM with electronic equipment that is nice for the foremost computing task.
- It contains **more ALU units** than CPU.

Why is TensorFlow Popular?

- TensorFlow is the best library of all because it is built to be **accessible for everyone.**
- Tensorflow library incorporates **different API** to built at scale deep learning architecture like CNN or RNN.
- TensorFlow is based on **graph computation**
- it allows the developer to visualize the construction of the neural network with Tensorboad.
- This tool is helpful to **debug** the program.
- Finally, Tensorflow is built to be deployed at scale.
- It runs on **CPU and GPU**.

TensorFlow Algorithms

Below are the algorithms supported by TensorFlow:

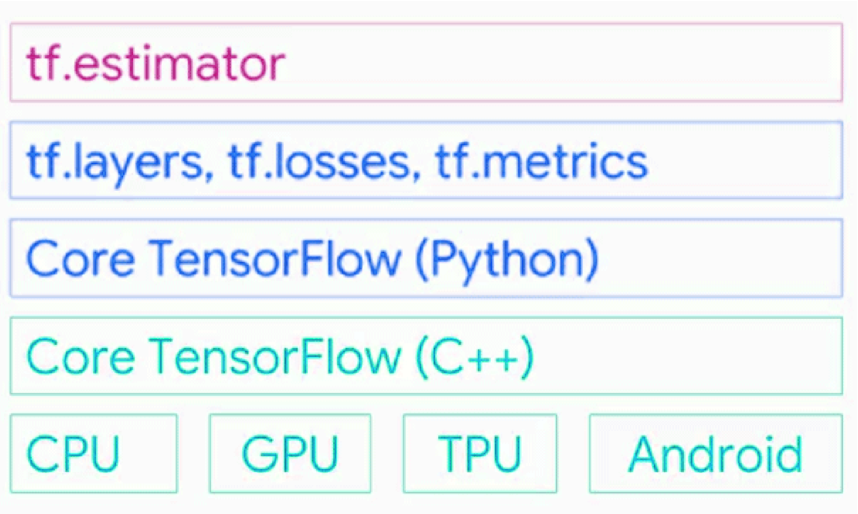
Currently, TensorFlow 1.10 has a built-in API for:

- Linear regression: tf.estimator.LinearRegressor
- Classification:tf.estimator.LinearClassifier
- Deep learning classification: tf.estimator.DNNClassifier
- Deep learning wipe and deep: tf.estimator.DNNLinearCombinedClassifier

- Booster tree regression: tf.estimator.BoostedTreesRegressor
- Boosted tree classification: tf.estimator.BoostedTreesClassifier

Tensorflow Basics

Tensors are objects that describe the linear relation between vectors, scalars, and other tensors. Tensors are nothing but multidimensional arrays. Tensorflow provides **support to write the code according to your requirements and access to different kinds of tools**. For example, we can write code in C++ and can call C++ code from python. Or we can write python code and call it by C++. tf.estimator.Estimator()



The lowest layer it supports two languages first is the Python language and the second C++ language. You can write it in any language in your comfort zone. It has a collection of different math libraries that help to create math functions easily.

It also provides support for processing like CPU, GPU, TPU and also runs on android mobiles.

Tf.layers:- tf.layers are used for method abstract so that you can customize the layers of neural networks.

```
pip install tensorflow / conda install tensorflow (Anaconda)
```

- This will install Tensorflow with GPU supported configurations.

```
          pip install Tensorflow-gpu
```

Basic Data types of Tensorflow

The basic data types in the Tensorflow framework (Tensors)

Below shows each dimension of tensors.

- **Scalar –** 0 Dimensional Array
- **Vector –** 1 Dimensional Array
- **Matrix –** 2 Dimensional Array
- **3D Tensor –** 3 Dimensional Array
- **N – D Tensor –** N-dimensional array

1. Vector

An array of numbers, which is either continuous or discrete, is defined as a vector. Machine learning algorithms deal with fixed length vectors for better output generation. Machine learning algorithms deal with multidimensional data so vectors play a crucial role.

2. Scalar

Scalar can be defined as one-dimensional vector. Scalars are those, which include only magnitude and no direction. With scalars, we are only concerned with the magnitude. Examples of scalar include weight and height parameters of children.

3 . Matrix

Matrix can be defined as multi-dimensional arrays, which are arranged in the format of rows and columns. The size of matrix is defined by row length and column length. Following figure shows the representation of any specified matrix.

Tensor Data Structure

Tensors are used as the basic data structures in TensorFlow language. **Tensors represent the connecting edges in any flow diagram called the Data Flow Graph.**

Tensors are defined as multidimensional array or list.

Tensors are identified by the following three parameters:

a) Rank
- Unit of dimensionality described within tensor is called rank.
- It identifies the number of dimensions of the tensor.
- A rank of a tensor can be described as the order or n-dimensions of a tensor defined.

b) Shape
- The number of rows and columns together define the shape of Tensor.

c) Type

Type describes the data type assigned to Tensor’s elements.

A user needs to consider the following activities for building a Tensor:

- Build an n-dimensional array
- Convert the n-dimensional array.

→Various Dimensions of TensorFlow

TensorFlow includes various dimensions. The dimensions are described in brief below:

- One dimensional Tensor

One dimensional tensor is a **normal array structure** which includes one set of values of the same data type. **Declaration**

```
import numpy as np

tensor_1d = np.array([1.3, 1, 4.0, 23.99])

print tensor_1d
```

- Two dimensional Tensors

Sequence of arrays are used for creating “two dimensional tensors”.

```
import numpy as np

tensor_2d=np.array([(1,2,3,4),(4,5,6,7),(8,9,10,11),(12,13,14,15)])

print(tensor_2d)

 [[ 1 2 3 4]

 [ 4 5 6 7]

[ 8 9 10 11]

[12 13 14 15]]
```

Types of Tensors

Import tensorflow as tf

Constants are exactly what their names refer to. They are the **fixed numbers** in your equation. To define a constant, we can do this:

```
a = tf.constant(1, name='a_var')
 b = tf.constant(2, name='b_bar')
```

Aside from the value 1, we can also provide a name such as “a_var” for the tensor which is separate from the Python variable name “a”. It’s optional.

After defining, if we print variable a, we’ll have:

```
<tf.Tensor 'a_var:0' shape=() dtype=int32>
```

Variables are **the model parameters** to be optimized, for example, the weights and biases in your neural networks. Similarly, we can also define a variable and show its contents like this:

```
  c = tf.Variable(a + b)
c
```

o/p `<tf.Variable 'Variable:0' shape=() dtype=int32_ref>`

all variables need to be initialized before use

```
init = tf.global_variables_initializer()
```

values for a and b, i.e., integers 1 and 2 are not showing up anywhere. That’s an important characteristic of TensorFlow — “lazy execution”, meaning **things are defined first, but not run. It’s only executed when we tell it to do, which is done through the running of a session**.

DATA VISUALIZATION

- The representation of information in the form of a chart, graph, diagram, picture, etc.
- "data visualization can also be used as a reporting tool“
- Data visualization is the **graphical representation of datasets** and information. Data visualization is an umbrella term for **visualizing all types** of data through charts, graphs, and maps.
- The ultimate goal is to visually represent your data in an accessible and **easy-to-understand** manner.
- Visualizing data is a fundamental step in understanding trends, uncovering patterns, and tracking outliers.
- Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, analyze.

What are the Different Types of Data Visualization?

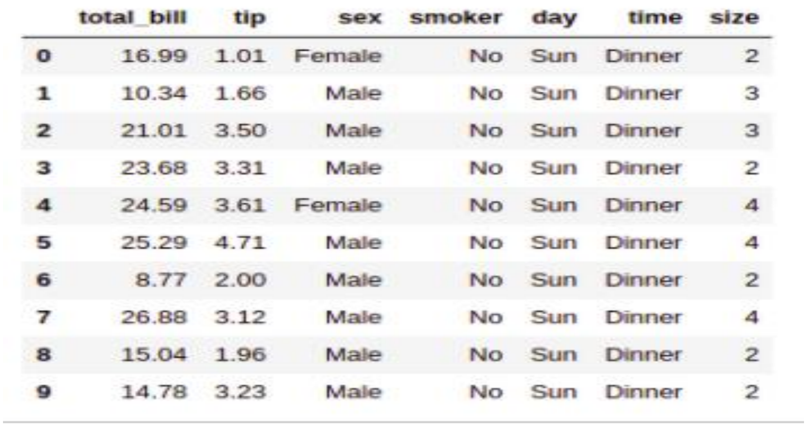
- Bar graphs
- Line graphs
- Pie charts
- Scatter plots: A slightly more advanced data visualization method is scatter plotting. Scatter plots are an effective way to explore the relationship between two variables and multiple sets of data.
- Histogram

- **Python provides various libraries** that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs.
- The four libraries are:

1. Matplotlib
2. Seaborn
3. Bokeh
4. Plotly

```
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

# printing the top 10 rows
print(data.head(10))
```

Tips database is the record of the tip given by the customers in a restaurant for two and a half months in the early 1990s. It contains 6 columns such as total_bill, tip, sex, smoker, day, time, size.



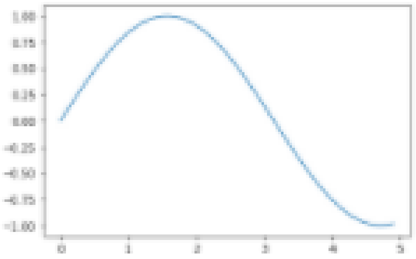
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2

1.MATPLOTLIB

- **Matplotlib** is an easy-to-use, low-level data visualization library that is built on **NumPy** arrays.
- It consists of various plots like **scatter plot, line plot, histogram**, etc.
- Matplotlib provides a lot of flexibility.

Pip install matplotlib

- **Scatter Plot**
- Scatter plots are used to observe relationships between variables and uses dots to represent the relationship between them. The **scatter()** method in the matplotlib library is used to draw a scatter plot.
- **matplotlib.pyplot**
- is a state-based interface to matplotlib. It provides an implicit, MATLAB-like, way of plotting. It also opens figures on your screen, and acts as the figure GUI manager.
- pyplot is mainly intended for interactive plots and simple cases of programmatic plot generation: eg:
- import numpy as np
- import matplotlib.pyplot as plt
- x = np.arange(0, 5, 0.1)
- y = np.sin(x)
- plt.plot(x, y)

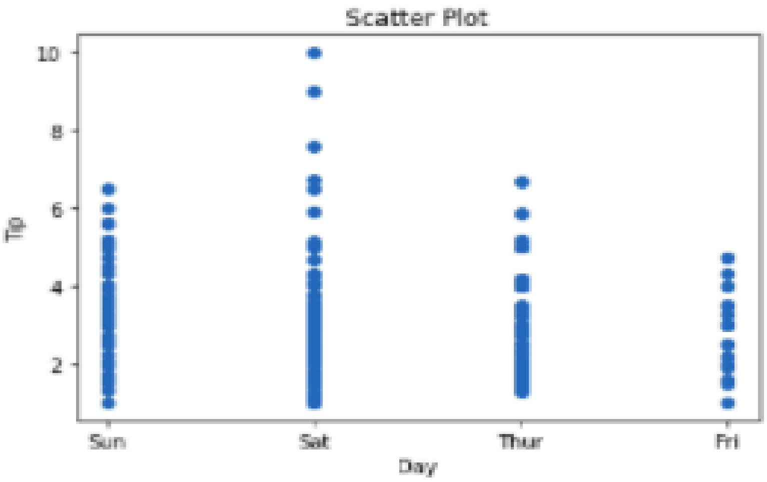


```
import pandas as pd
import matplotlib.pyplot as plt

# reading the database
data = pd.read_csv("tips.csv")

# Scatter plot with day against tip
plt.scatter(data['day'], data['tip'])
```

```
# Adding Title to the Plot
plt.title("Scatter Plot")

# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')

plt.show()
```

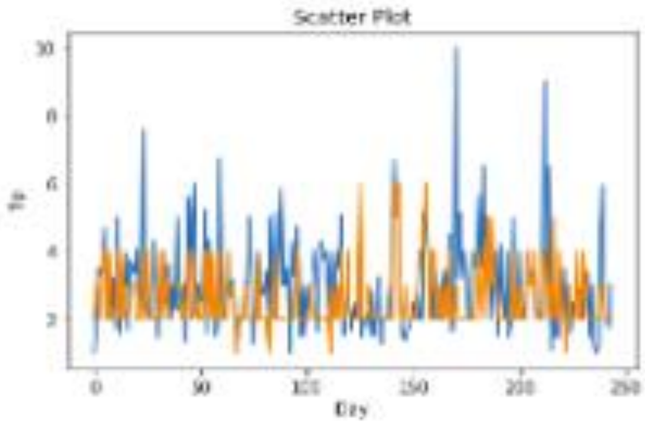
A) Scatter plot eg:

```
import pandas as pd
import matplotlib.pyplot as plt


# reading the database
data = pd.read_csv("tips.csv")

# Scatter plot with day against tip
plt.scatter(data['day'], data['tip'])

# Adding Title to the Plot
plt.title("Scatter Plot")

# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')

plt.show()
```



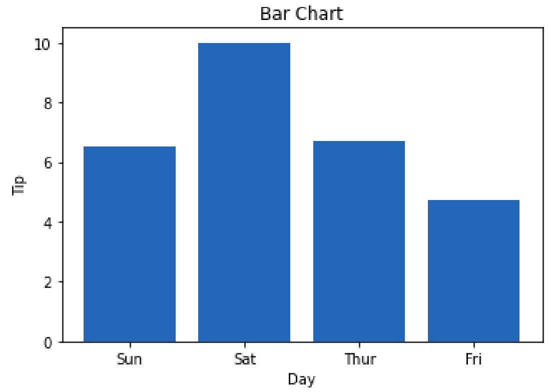
- **Line Chart**
- Line Chart is used to represent a relationship between two data X and Y on a different axis. It is plotted using the **plot()** function.
- import pandas as pd

```
import matplotlib.pyplot as plt
# reading the database
data = pd.read_csv("tips.csv")
# Scatter plot with day against tip
plt.plot(data['tip'])
plt.plot(data['size'])
# Adding Title to the Plot
plt.title("Scatter Plot")
# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')
plt.show()
```



C) **Bar Chart**

- A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent.
- It can be created using the **bar()** method.



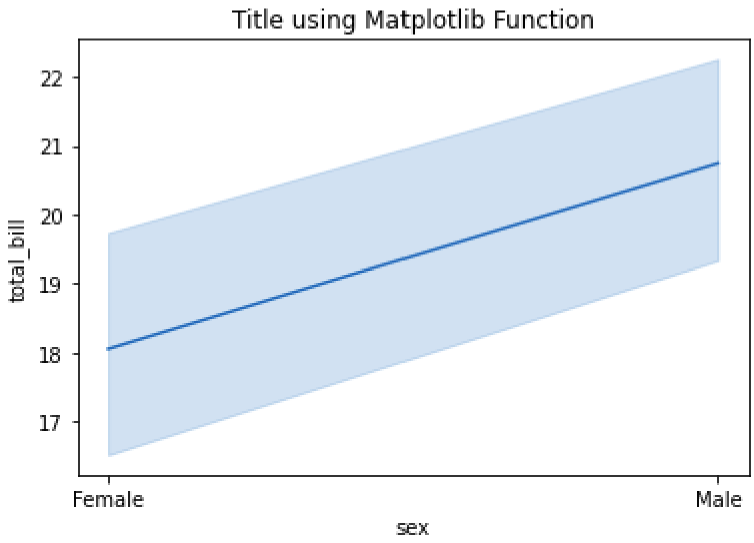
- **Histogram**
- A histogram is basically used to represent data in the form of some groups.
- It is **a type of bar plot** where the X-axis represents the bin ranges while the Y-axis gives information about frequency.
- The **hist()** function is used to compute and create a histogram.
- In histogram, if we pass categorical data then it will automatically compute the frequency of that data i.e. how often each value occurred.
- A histogram is a diagram involving rectangles whose area is proportional to the frequency of a variable and width is equal to the class interval

2) Seaborn

- **Seaborn** is a high-level interface built on top of the Matplotlib. It provides beautiful design styles and color palettes to make more attractive graphs.
- To install seaborn type the below command in the terminal.

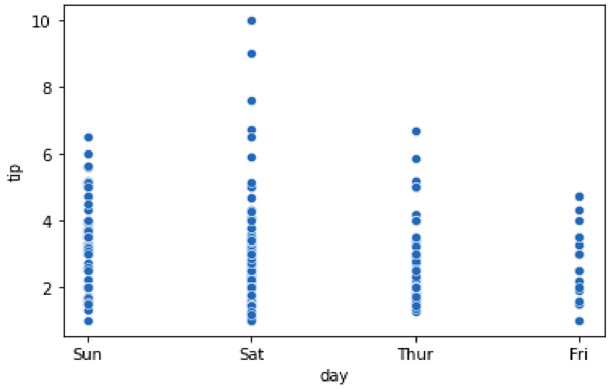
```
pip install seaborn
```

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
# reading the database
data = pd.read_csv("tips.csv")
# draw lineplot
sns.lineplot(x="sex", y="total_bill", data=data)
# setting the title using Matplotlib
plt.title('Title using Matplotlib Function')
plt.show()
```



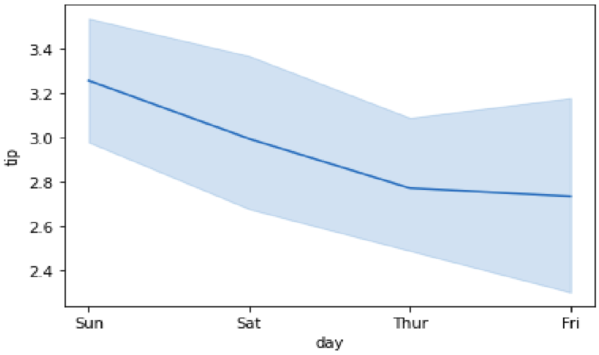
- Scatter plot is plotted using the **scatterplot()** method.
- This is similar to Matplotlib, but additional argument data is required.
- # importing packages

```
import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# reading the database

data = pd.read_csv("tips.csv")

sns.scatterplot(x='day', y='tip', data=data,)

plt.show()
```



- **Line Plot**
- Line Plot in Seaborn plotted using the **lineplot()** method. In this, we can pass only the data argument also.

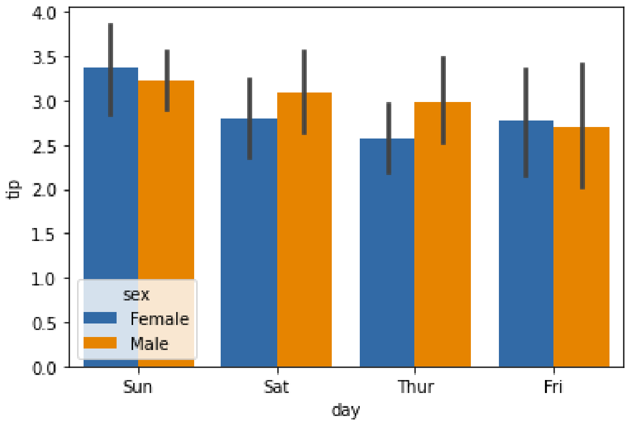
```
# importing packages

import seaborn as sns
import matplotlib.pyplot as plt

import pandas as pd

# reading the database

data = pd.read_csv("tips.csv")

sns.lineplot(x='day', y='tip', data=data)

plt.show()
```



- ## Bar Plot

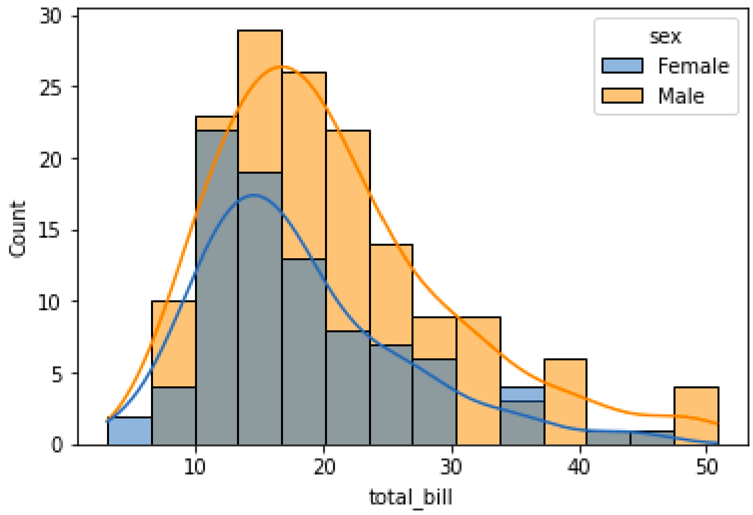
- Bar Plot in Seaborn can be created using the **barplot()** method.

```
# importing packages

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# reading the database

data = pd.read_csv("tips.csv")

sns.barplot(x='day',y='tip', data=data,hue='sex')

plt.show()
```



- **Histogram**
- The histogram in Seaborn can be plotted using the **histplot()** function.

```
# importing packages

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# reading the database

data = pd.read_csv("tips.csv")

sns.histplot(x='total_bill', data=data, kde=True, hue='sex')
```



3) Bokeh

- The third library is okeh is mainly famous for its interactive charts visualization.
- Bokeh renders its plots using **HTML and JavaScript** that uses modern web browsers for presenting elegant, concise construction of novel graphics with high-level interactivity.

```
Pip install bokeh
```

4) Plotly

- The last library **plotly**.
- Potly has hover tool capabilities that allow us to detect any outliers or anomalies in numerous data points.
- It allows more customization.
- It makes the graph visually more attractive.

- To install it type the below command in the terminal.

  Pip install plotly

CROSS VALIDATION IN MACHINE LEARNING

- In machine learning, we couldn't fit the model on the training data and can't say that the model will work accurately for the real data.
- For this, we must assure that our model got the **correct patterns** from the data, and it is not getting up too much noise.
- For this purpose, we use the cross-validation technique

Cross-validation is a technique in which we **train** our model using the **subset of the data-set** and then **evaluate using the complementary subset** of the data-set.

The **three** steps involved in cross-validation are as follows :

1. Reserve some portion of sample data-set.
2. Using the rest data-set train the model.
3. Test the model using the reserve portion of the data-set.

METHODS OF CROSS VALIDATION

a) Validation

b) LOOCV

c) K – Fold

a) Validation

- In this method, we perform **training on the 50%** of the given data-set and rest **50% is used for the testing purpose.**
- The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the **remaining 50% of the data contains some important information which we are leaving while training our model** i.e **higher bias.**

b) LOOCV (Leave One Out Cross Validation)

- In this method, we perform **training on the whole data-set** but **leaves only one data-point of the available data-set and then iterates for each data-point.**

- It has some advantages as well as disadvantages also.

- An advantage of using this method is that we make use of all data points and hence it is **low bias**.

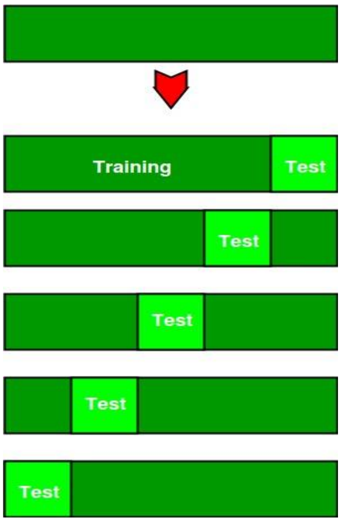
- The major drawback of this method is that it leads to **higher variation in the testing model** as we are testing against one data point.

- If the data point is an **outlier** it can lead to higher variation.

- Another drawback is it takes a lot of execution time as it iterates over 'the number of data points' times.

c) K-Fold Cross Validation

- In this method, we **split the data-set into k number of subsets**(known as folds) .
- then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model.
- In this method, we iterate k times with a different subset reserved for testing purpose each time.



- The diagram above shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total **25** instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training([5-10] testing and [1-5 and 10-25] training), and so on.

Advantages of train/test split:

1. This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times.

2. Simpler to examine the detailed results of the testing process.

Advantages of cross-validation:

1. More accurate estimate of out-of-sample accuracy.
2. More "efficient" use of data as every observation is used for both training and testing.

VALIDATION CURVE

- Model validation is used to determine how effective an estimator is on data that it has been trained on as well as how generalizable it is to new input.
- To measure a model's performance, we first split the dataset into training and test splits, fitting the model on the training data and scoring it on the reserved test data.
- **In order to maximize the score, the hyperparameters of the model must be selected which best allow the model to operate in the specified feature space**.
- Most models have multiple hyperparameters and the best way to choose a combination of those parameters is with a grid search.

  However, it is **sometimes useful to plot the influence of a single hyperparameter on the training and test data to determine if the estimator is underfitting or overfitting for some hyperparameter values**.

- validation is used to measure **the effectiveness of an ML model**.
- A good ML model not only fits the training data very well but also is generalizable to new input data.
- **Model hyperparameters** play an important role in determining the effectiveness of an ML model.
- The validation curve is a graphical technique that can be used to measure the influence of a single hyperparameter.
- **By looking at the curve, you can determine if the model is underfitting, overfitting or just-right for some range of hyperparameter values**.
- A **Validation Curve** is an important diagnostic tool that shows the sensitivity between to changes in a Machine Learning model's accuracy with change in some parameter of the model.
- A validation curve is typically drawn between some parameter of the model and the model's score.

- **Two curves** are present in a validation curve – one for **the training set score** and one for the **cross-validation score**.

- By default, the function for validation curve, present in the **scikit-learn library** performs 3-fold cross-validation.

- In Machine Learning one of the main task is to **model** the data and **predict** the output using various Classification and Regression Algorithms.

- But since there are so many Algorithms, it is really difficult to choose the one for predicting the final data. So we need to compare our models and choose the one with the **highest accuracy.**

- Using the sklearn library we can find out the scores of our ML Model and **thus choose the algorithm with a higher score to predict our output.**

- Another good way is to calculate errors such as mean absolute error and mean squared error and try to minimize them to better our models.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

- **A validation curve is used to evaluate an existing model based on hyper-parameters and is not used to tune a model**.

- This is because, if we tune the model according to the validation score, the model may be biased towards the specific data against which the model is tuned; thereby, not being a good estimate of the generalization of the model.

- Ideally, we **would want both the validation curve and the training curve to look as similar as possible.**

- **If both scores are low**, the model is likely to be ***underfitting***. This means either the model is too simple or it is informed by too few features. It could also be the case that the model is regularized too much.

- If the training curve reaches a **high score** relatively quickly and the validation curve is lagging behind, the model is ***overfitting.*** This means the model is very complex and there is too little data; or it could simply mean there is too little data.

- We would want the value of the parameter where the training and validation curves are closest to each other.

- Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python.

- It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Example ppt