MACHINE LEARNING

With Python

Harman Abdul Waheed

Fundamentals of Machine learning

Fundamentals
Machine Learning / Artificial Intelligence with Python

Compiled by: Harman

2025

harmanwaheed@gmail.com

Cell: +92 3284884150

Instagram: fly_fazaia

Answer these questions by the time you read this book. You don't necessarily need to answer these in one sitting.

Write your short-term goals:

1. _____
2. _____
3. _____
4. _____
5. _____

Write your long-term goals:

1. _____
2. _____
3. _____

I feel fulfilled and successful when I reach/accomplish/achieve:

-
-
-
-
-

Contents

GETTING STARTED – INTRODUCTION TO BOOK

Hi, welcome!!! In this book we will be learning fundamentals of machine learning and we will dive into the details of one of the types of machine learning: **supervised machine learning**.

Programming language used in this course: **Python**

Introduction

By following this book, you will develop a clear understanding of pandas for data manipulation, machine learning, and in particular, supervised learning. There are 7 sections in this course with total 35 lectures.

In first section we will discuss machine learning and types of machine learning. I have tried my best to explain the concept in simple and understandable language.

In second section we will train our first machine learning model. This section contains 20 lectures and we assure that if you take those all 20 lectures, you will have no confusion in rudimentary machine learning concepts. There is coding, written lectures and quizzes in this section.
we have also covered all the fundamental steps of data science for your clarity, including exploring datasets, creating custom datasets, and understanding datasets for machine learning applications

In third section we have trained model on another machine learning algorithm (linear regression). Working of linear regression, graphical understanding and basic math behind this algorithm is briefly explained. There are assignments for you, that will increase your confidence on your understandings.

In fourth section we will train a ML model on famous dataset, iris dataset, containing feature of flowers. We will train a model which predicts category of a flower based on some features of flower (petal and sepal dimensions).

In fifth section: Until now, accuracy of model was unknown, we were not having any idea how well our model works. We will discuss technique to check accuracy of our models in this section. I have explained very comprehensively how to evaluate performance of any of our machine learning model.

We have checked the accuracy of our iris dataset machine learning model (which we trained in previous section), which is amazingly 97%.

In sixth and last section we will discuss another widely used machine learning algorithm (logistic regression). Using logistic regression we will train a model to predict whether a patient is suffering from diabetes or not. We will create our model that predicts whether a person have diabetes or not, with accuracy of almost 79%.

In seventh section we have assembled all codes of this book in a single place where you can easily access code of any section from this book.

How much time you need to complete this course?

Take this book slow and steady. If you give one hour a day, you can complete this book within 20-30 days. When you start, make sure that you are consistent with your learning.

Time per day	**time to complete this course**
0.5 - 1 hour	*20 - 35 days*
1 - 2 hours	*15 - 25 days*
2 - 3 hours	*7 - 15 days*

Spending more than 1.5 hours is not suggested if this is beginning of your machine learning journey. Don't make any commitment related to time with yourself. Study as much as you can easily digest in one go.

Suggestion: Practice all codes of this course in your own compiler along with.

SECTION 1

Lecture 1: Welcome

Peace be upon you.

"If you really want to do something, you have to get out of your comfort zone."

Contact me if you need any help.

harmanwaheed@gmail.com

Good luck with your journey!

Lecture 2: What is Machine learning

Machine Learning, as the name suggests, refers to a machine acquiring knowledge or skills. But what exactly does this mean?

In traditional programming, humans used to write instructions for a computer to follow, but in machine learning, the computer learns to perform a task by itself.

Humans still write instructions. **Let suppose** a property dealer has given these instructions in his website:

- If the house is in area, where gas and electricity is sufficient, then the price of house will be 50,000 rupees
- If house is in area where gas and electricity is not sufficient for living, then the price of house will be 30,000 rupees.

One day you were exploring his website and tried to check price of house where gas is sufficient, but electricity is not… boom! His website won't be able to show you the price for this house, because he didn't consider this such criteria for a house.

Till now, his website was using set of conditional codes (if this, do this, if that, do that) to show a price for different areas. His website can't handle the unseen data (scenarios where houses have features, which were not addressed when he made the website).

To make his website more robust, so that we can deal well with unseen data, we can use machine learning tools.
I hope this will make sense now:

'In Machine learning, machine learns to deal with unseen data'

I hope you got the basic idea of machine learning.

To explain the concept, I have given a very simple example; real world problems are a bit different. We will deal with those soon.

Lecture 3: How Machine learns

Machines… think of them as young children, around 5 to 9 years old, who learn by observing their surroundings.

For example, my 2-year-old nephew saw a dog on the ground and ran to me. I reassured him not to be afraid of this creature and told him that it was a "doggy."

Now, whenever I take him to the ground, he looks for dogs. When he finds one, he runs to me, points at the dog, and says, "dugi... dugi..." Even when he sees a different dog, he still recognizes it as "dugi... dugi..."

Human's machine is pretty fast and accurate. He (my nephew) saw a dog in ground, and now he knows that a dog can appear in any size (1-3 feet maybe) and color. He recognizes any dog immediately.

Let's get back to machines now, as we mentioned in the first line, take machine learning as a kid, who learns from his surrounding and his elders. Similarly, machines require something from which they can learn. That thing is data.

For example,

Let say you are making a machine learning model that will tell you whether an image has cat in it or not. For this, we have to give our model some images of real cats and we will tell our model that:

'See this! This is what a Cat looks like!'

After this, we will give our model another image of cat and we will ask our model whether there is a cat in this image or not?

In this example, firstly we provided our model with some data (images of cats) from which model will learn. From the given data, machine learn how to make decisions (deciding whether the image has cat in it or not). That's why to make a good and accurate model, we have to provide bunch of situations (large, extensive data).

Why?

Let say I have given my model an image of orange cat. Model can learn that everything orange is cat. After that, if we input image of an orange (fruit)… Our model will predict that image has a cat in it. RIP model.

To overcome this,

We will upload bunch of cat images of different colors and sizes and we will tell our model that

'*These all are cats! Cats looks like these*'

Now, machine will understand that cat can be of different colors and size. Model will try to figure out something common in all images. These can be:

- 4 legs of each cat
- 2 ears
- Small height
- One tail

These things are likely to be common in each image, so model can learn that anything having 4 legs, 2 ears, a tail and small height is a cat.

And this model is a lot better than previous model, which was classifying orange (fruit) as a cat.

That's why we have to train our model on large and extensive data.

Lecture 4: Two types of Machine learning

In this section, we will be discussing two types of machine learning.

Revise previous example (lecture 3) in which we provided our model with some images of cats. We told our model that '*these are the cats! learn from these!'*

We performed two important tasks here.

1 - Giving images of cats.

2 - Telling our model that these are cat images **(labeling our data).**

Such scenario in which we label our data is called **'Supervised Machine learning'**.

Now let’s consider a different example.

LECTURE 5: TERMS WE WILL BE USING

Let say you have 2000 images of different animals. Your task is to separate images of each animal in different folders. For example, all images of monkeys should be in one folder, and each lion images should be in other, and so on.

It can be a tedious task for you to separate 2000 images in different folders.

You, a data scientist, will feed all these images in a machine and machine will learn something from it. **How? Notice that we didn't label our data. (We did in previous example)**

Well, in this case step 2 (labelling our data) is missing. So how and what machine will learn?

We know that there are 2000 images of different animals. Let say there are 500 images of elephant**,** 500 images of lion**,** 500 images of monkey and 500 images of cats.

All images of elephants will have something in common.

All images of lion will have something in common.

All images of monkeys will also have something in common.

And all cats will also have something in common.

Based on these patterns, our model may learn that there are 4 types of things in our data:

- First one is large, fat, have trunk, grey (elephants are mostly grey).
- Second one has normal size, yellow (Lions are mostly yellow) in color, hairs around neck
- Third one is thin, two feet, two hands, prominent bones (monkeys)
- Fourth one is small, four legs, small face and different colors. (cats)

Our model will separate all the images based on such patterns

Such scenario is **'Unsupervised Machine learning'**.

In supervised machine learning, data is already labeled.
In unsupervised machine learning, data isn't labeled.

Have a glass of water, understand upper concept, here is so called definition:

- In unsupervised machine learning, the algorithm is provided with unlabeled data. Model find patterns within the data.
- In supervised machine learning, the algorithm is provided with labeled data. (relate to example from, *Lecture 3: How machine learns*)

I hope things are making sense.

Lecture 5: Terms we will be using

Till now, I have tried to maintain language as simple as possible. I will try this for the rest of the book as well.

Here are some of the technical terms we will be using in this course:

Classification:

Recall the example from previous lectures, where I explained an example of unsupervised machine learning. I wrote sufficient text to deliver the concept that our model will simply separate pictures of animals.

Term 'classification' makes it easier to deliver such messages.

To summarize, we can say:

'Our model will classify animal images.'

Classification can be explained as making classes.

In our example (from *lecture 4*), there will be four classes (though any number is possible).

- First class contains elephants.
- Second class contains lions.
- Third class contains monkeys.
- Fourth class contains cats.

'Male and female images are classified through machine learning.' (I hope this makes sense now).

Features

Well, features sound not a very technical term, but in terms of machine learning, it is!

What features are?

Features are something, based on which our model will make predictions.

You remember the example, where we had 2000 images of different animals (*lecture 4*).

There, we discussed four categories of animals:

- First one is large, fat, have trunk, grey. (elephants are mostly grey)
- Second one has normal size, yellow (Lions are mostly yellow), hairs around neck.
- Third one is thick, two feet, two hands, and prominent bones. (monkeys)
- Fourth one is small, four legs, small face and different colors. (cats)

We see that each category have some specifications of each animal. These specifications are: Size, color, thickness and number of feet.

These are specifications on the basis of which our model will make predictions (for simplicity I haven't discussed other attributes for now)

These specifications are Features of our model! Try to understand this sentence:

'In the task of classification of cats and sparrows, our features are: size, wings, tail, color etc.'

Labels

In terms of machine learning, what our machine learning model predicts, are labels.

In the example of images of animals, our labels were: Elephant, Lion, Monkey and cat. (because for each image, model will predict one of these)

Now try to understand this sentence:

'In task of classification, our labels are cats and sparrow.'

Now this:

'In task of image classification, our labels are cats and sparrow. In addition, our features are: color, tail, wings, size'

Based on color, tail, wings, and size… Our model will predict whether the image contain sparrow or a cat.

Solve the next quiz; let's see how much you grabbed.

Quiz 1: Fundamentals about ML, multiple choice questions

1. **What is the main characteristic of supervised learning?**

- Unlabeled data
- Learning without guidance
- Labeled data

2. **What is main purpose of machine learning?**

- To make computer work with unseen data
- To make computer work with already seen data in more robust way

3. **My model can classify patients of blood pressure and diabetes. What does it mean?**

- My model can differ between blood pressure patients and diabetes patients and can separate them (it will create two classes: blood pressure patients and diabetes patients).

- My model can make a single class for all patients suffering from blood pressure and diabetes.

4. **What are features in terms of machine learning?**

- Features are different elements through which we make our model perform well.

- Features are the object that our model predicts. From previous mcq, 'blood pressure patient' and 'diabetes patients' are two features.

- Features are the attributes or specifications, from which our model learn to predict something.

5. What are labels in term of machine learning?

- The attributes of a dataset are called labels.

- The output or target variable that the model predicts.

SECTION 2: SUPERVISED MACHINE LEARNING

Lecture 7: Exploring the dataset

Let us see how a dataset looks like.

	A	B	C	D	E
1	Study hours	Marks in previous grade	class performance out of 10	Marks scored	
2	3	72%	7.2	75%	
3	1	69%	6.3	72%	
4	5	87%	8.6	84%	
5	2	75%	7.9	78%	
6	3	78%	7.9	76%	
7	5	85%	9.2	94%	
8	4	82%	9.1	89%	
9	2	71%	7.3	78%	
10					

- This data is in excel format.
- There are 4 columns in this data. Each column has appropriate name.
- There are 9 rows in this dataset.
- Each row represents data of one student.

- Looking at the second row of our dataset, we see a student who studies 3 hours a day, scored 72% in his previous grade, has a class performance rating of 7.2 out of 10, and achieved 75% marks in the annual exam.

- Each row contains detailed information about a student.

- Look at the sixth row. What information can you extract from it? Open your notebook and write down your observations.

- Each row is called a data point, as it represents the complete information of an instance.

Lecture 8: Understanding data for machine learning

Hi, let's now understand our data for machine learning.

What does it mean?

Have a look at our dataset:

	A	B	C	D	E
1	Study hours	Marks in previous grade	class performance out of 10	Marks scored	
2	3	72%	7.2	75%	
3	1	69%	6.3	72%	
4	5	87%	8.6	84%	
5	2	75%	7.9	78%	
6	3	78%	7.9	76%	
7	5	85%	9.2	94%	
8	4	82%	9.1	89%	
9	2	71%	7.3	78%	
10					

In supervised machine learning, we try to predict something....that is called *'label'*.

What is the label in our dataset? Well, that depends on us—what do we want the computer to predict based on the given data?

Let us make a model to predict the annual marks scored (last column). Hence, the label is last column, 'Marks Scored'. Our model will learn from this dataset, and later on, it will predict on unseen data. Let's dive deeper.

In this case, the features (or attributes) are the first three columns: *Study Hours, Previous Grade Marks, and Class Performance*. The fourth column is not a feature because we have selected it as the label.

Don't worry if you didn't get previous 6-7 lines. Let us have a breakdown:

What we expect from our model to do:

Once we have trained our model, we expect it to predict marks of a student. Those predictions will be based on the features.

We will tell our model that there exists a student who study 8 hours a day, scored 95% in previous exams and performed 9.6 out of 10 in class. After telling these things to our model, we expect it to tell us how much marks that student will score in annual exams.

- Features: study hours, marks in previous grade, class performance.
- Label: Marks scored by student.

To train the model:

We will feed our data in a model.

	A	B	C	D
1	Study hours	Marks in previous grade	class performance out of 10	Marks scored
2	3	72%	7.2	75%
3	1	69%	6.3	72%
4	5	87%	8.6	84%
5	2	75%	7.9	78%
6	3	78%	7.9	76%
7	5	85%	9.2	94%
8	4	82%	9.1	89%
9	2	71%	7.3	78%
10				

In this image, I have colored different columns for better understanding. Last column (orange one) is the label. This is what we want our model to predict later. Green columns (first three columns) are features.

In short:

- We want to make a model to predict students annual exam marks
- We will predict this on the basis of study hours, marks in previous grade, and class performance.

(Continue to lecture 10 for now, skip this one)

Lecture 9: Two types supervised machine learning

Before going any further, let me introduce you to the types of supervised machine learning.

These two types are based on kind of labels of our dataset.

Let's suppose we are trying to predict whether an image contains a cat or a dog. There are two possible outcomes: the model can classify the image as either a cat or a dog. This type of model in which model's prediction are based on specific number of labels is called **Classification**.

Classification problems are not limited to just two labels; they can have three, four, or even hundreds. The key point is that the number of labels is fixed, and the model will always make predictions within those predefined labels.

In simple words, you're trying to categorize or classify input data into already defined classes. (For this scenario, there are two classes: dogs and cats)

Regression is another type of machine learning task where the model predicts continuous numerical values rather than discrete categories (classes are not defined). In this case, you're not predicting classes; instead, you're predicting a quantity.

For example, if you are building a machine learning model to predict house prices, the predicted prices will vary continuously rather than fitting into predefined categories. Since house prices have no fixed range, the model's predictions will be continuous values.

Different algorithms are used for classification problems and regression problems, depending on the nature of the task.

In both cases, the machine learning model learns from data to make predictions on new, unseen data

Lecture 10: Building our model

Continued from lecture 8, now, you know about dataset, labels and features, it's time to make our first machine learning model.

Let suppose *Professor Ahmad* teaches math to 10th grade and he want to predict, how much each student will score in his/her annual exams. *Professor Ahmad* contacted you, a data scientist, and requested you to make a machine learning model for this task (**a model to predict annual marks of students**).

As we know, Data is always required to train a machine learning model, you will ask *Professor Ahmad* to provide you with some previous data of students. He will take some time and gather data of students.

Let say he provide us this data:

	A	B	C	D	E
1	Study hours	Marks in previous grade	class performance out of 10	Marks scored	
2	3	72%	7.2	75%	
3	1	69%	6.3	72%	
4	5	87%	8.6	84%	
5	2	75%	7.9	78%	
6	3	78%	7.9	76%	
7	5	85%	9.2	94%	
8	4	82%	9.1	89%	
9	2	71%	7.3	78%	
10	3	75%	6.9	73%	
11	1	65%	6.2	64%	
12	4	80%	8.1	82%	
13	6	91%	9.5	93%	
14					
15					

(We have discussed the same data in lecture 8)

There are four columns, one of them is label, and other three are features. Overall, there are 12 data points.

Upper line requires your attention!

This is data that professor has provided us, and we have to construct our model based on this data!

Goin good? It's time to code now.

Lecture 11: Building our model, part 2: Loading data

As we have our data, label and features, we are all done with the requirements.

Loading our data in python environment:

To load data in python, we will use Pandas. We have to import pandas to access its functions of data processing.

```
import pandas as pd
data = pd.read_excel('path to your data.xlsx')
```

For example, if you have saved your data in local disk D, you will write:

```
data = pd.read_excel('D://file_name.xlsx')
```

'file_name' will be replaced with the name of excel file that contain our data

In my case: To make the path and format more clearly to you, I will show you where I have saved my data and how I loaded that data in python

I have local disk D and within that, I have a folder named 'datasets'. Within that folder, I have saved my excel file. And my excel file is named 'professor_ahmad_class'.

So, to load the dataset, my code will look like this:

```
data = pd.read_excel('D://datasets/professor_ahmad_class.xlsx')
```

.xlsx is the format of excel file, which we are loading (all excel files have format .xlsx)

After loading dataset, let's see how it looks like in python.

For that, we can use command:

```
data.head()
```

This command will show first 5 rows of data (data points).

See the outputs on my compiler:

```
In [5]: import pandas as pd
        data = pd.read_excel('D://datasets/professor_ahmad_class.xlsx')

In [6]: data.head()
```

Out[6]:

	Study hours	Marks in previous grade	class performance out of 10	Marks scored
0	3	0.72	7.2	0.75
1	1	0.69	6.3	0.72
2	5	0.87	8.6	0.84
3	2	0.75	7.9	0.78
4	3	0.78	7.9	0.76

```
In [ ]:
```

Compare the data in excel file and data in python.

You may have noticed that python automatically removed '%' sign from our data and calculated percentages of those values (See column number 2 and 4). Don't worry; we will deal with this while constructing our model.

Great, now we have loaded our dataset in python environment.

Lecture 12: Building our model, part 3: features and labels

We have loaded our data in python. Now we have to specify features and labels. Model can't decide it automatically of course.

As we discussed earlier, our **features** are: first 3 columns of dataset. And our **labels** are: last column of dataset. (We want to predict annual score of students, which is in last column). If you are getting confused in features and labels, it is advised to read: *lecture 8 Understanding data for machine learning.*

Now let's separate features and labels from our data.

We will store features and labels in two different variables:

Features:

```
features = data[['Study hours', 'Marks in previous grade', 'class performance out of 10']]
```

This line of code is specifying features as first three columns of our data.

Let's see how our features variable looks like. For that, write:

```
features.head()
```

And our output will be:

```
In [32]: features = data[['Study hours', 'Marks in previous grade', 'class performance out of 10']]
         features.head()
```

Out[32]:

	Study hours	Marks in previous grade	class performance out of 10
0	3	0.72	7.2
1	1	0.69	6.3
2	5	0.87	8.6
3	2	0.75	7.9
4	3	0.78	7.9

```
In [ ]:
```

Great, we see that '*features'* variable contain only first three columns. This is what we wanted. We wanted these three columns as features for our model.

Label:

For label, we want only one column (the last one).

```
label = data[['Marks scored']]
final_label = label * 100
```

First line of code is specifying the last column of our data as label. As we saw earlier that each value of this column was divided by 100 automatically due to '%' sign. Now, in second line of code, we have multiplied each value of label with 100 to make it normal again.

Now our labels are stored in a variable *'final_label'*.

This is how it looks like:

```
final_label.head()
```

```
In [43]: final_label.head()
```

Out[43]:

	Marks scored
0	75.0
1	72.0
2	84.0
3	78.0
4	76.0

```
In [ ]:
```

Now, we see that our variable *'final_label'* contain only last column of dataset that Professor Ahmad has provided us.

As features and labels are ready, let's train our model.

Coding done in this lecture:

```
features = data[['Study hours', 'Marks in previous grade', 'class performance out of 10']]
```

```
label = data[['Marks scored']]
final_label = label * 100
```

'features' Looks like:

LECTURE 12: BUILDING OUR MODEL, PART 3: FEATURES AND LABELS

```
In [32]: features = data[['Study hours', 'Marks in previous grade', 'class performance out of 10']]
         features.head()
```

Out[32]:

	Study hours	Marks in previous grade	class performance out of 10
0	3	0.72	7.2
1	1	0.69	6.3
2	5	0.87	8.6
3	2	0.75	7.9
4	3	0.78	7.9

```
In [ ]:
```

'final_label' looks like:

```
In [43]: final_label.head()
```

Out[43]:

	Marks scored
0	75.0
1	72.0
2	84.0
3	78.0
4	76.0

```
In [ ]:
```

- *'features'* contain features of our data.
- *'final_label'* contain labels of our data.

Lecture 13: Building our model, part 4

Till now, we have loaded our dataset in python environment, separated features and labels from our dataset.

We are all ready to construct a machine learning model. You may be thinking that this example is very simple... This is how we start learning complex things.

There are different machine learning algorithms, and for now we will be using '*nearest neighbors classifier*'.

This is one of the widely used ml (machine learning) algorithm.

Later in this section, we will explore how this algorithm actually makes prediction.

Ohkayyy, only thing we need is scikit-learn for this task. **scikit-learn** is a Python library for machine learning. To install this, we open our command prompt (or anaconda prompt) and write:

```
pip install -U scikit-learn
```

Now let's continue our coding. To make everything clear, here is the whole code we are done with till now:

```
import pandas as pd
data = pd.read_excel('D://datasets/professor_ahmad_class.xlsx')

features = data[['Study hours', 'Marks in previous grade', 'class performance
out of 10']]

label = data[['Marks scored']]
final_label = label * 100
```

To construct our model, we have to import our '*nearest neighbors classifier*'

Step 1

We will import our classifier this way:

```
from sklearn.neighbors import KNeighborsClassifier
```

Step 2

Initialize our model. Initializing a model will actually create a model, but it is not trained on any data yet. We will train it afterwards.

To create/initialize model:

```
mdl = KNeighborsClassifier()
```

Here we have created our model which is stored in variable *'mdl'*.

Step 3

Here, we will train our model. And it's pretty very simple!

Recall that we have to train our model on features and labels, and our features are stored in variable *'features'* and our labels are stored in variable *'final_label'* (coding is done in lecture 12).

To train our model on features and labels:

```
mdl.fit(features, final_label)
```

We pass features as first argument and labels as second argument.

Now our model is ready. Yes, it's very simple. Our model is ready to make predictions.

We will give this model to Professor Ahmad and for sure he will ask how can I use this thing? How to make prediction from this model?

For that, we need to give three pieces of information to our model, and those are features.

Let suppose there is a student of Professor Ahmad and he (professor) want to know how much marks that student can score according to our model. For that, we must provide the model with these pieces of information:

1 - Study hours of that student.

2 - Marks he scored in previous exams.

3 - Class performance of that student out of 10.

Let suppose, the new student study 7 hours a day, have scored 83% in his previous exams, and class performance of that student is 8.0 out of 10. Let us make our model predict marks of this student.

Let's feed this data in the model:

```
mdl.predict([[7, 0.83, 8.0]])
```

This is how we feed our data. You may have noticed that second argument (marks) is 0.83 instead of 83. Well, while loading data, marks were divided by 100 by python automatically. To make prediction, we will give our model data in same format in which model was trained.

What will be the output of upper code? (In which we made prediction)

```
In [25]: mdl.predict([[7, 0.83, 8.0]])

C:\Users\dell\anaconda3\lib\site-packages\sklearn\base.py:45
Classifier was fitted with feature names
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\sklearn\neighbors\
ions (e.g. `skew`, `kurtosis`), the default behavior of `mod
s behavior will change: the default value of `keepdims` will
eliminated, and the value None will no longer be accepted. S
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

Out[25]: array([82.])

In [ ]:
```

Greaaat, our model has predicted that the student will score 82% marks in his annual exams.

That pink box in image is not part of code, we can ignore this for now.

Let's talk about model performance...

We had a student who study 7 hours a day (too much). He performed 83% in previous exams, and class performance is 8.0 out of 10.

It is clear from this data that this student is extraordinary and will score well in exams. And the same thing is predicted by our model.

This is what Professor Ahmad needed, now he can give data of student and see how much the student will score.

Overall coding in this lecture (we trained our model in this lecture):

```
from sklearn.neighbors import KNeighborsClassifier

#initializing our model

mdl = KNeighborsClassifier()

#training our model
```

```
mdl.fit(features,final_label)

#making predictions

mdl.predict([[7, 0.83, 8.0]])
```

Note that the lines containing # are not parts of code; these are here just for instruction purpose.

Now one thing Professor may request you is to make your model user friendly. For that, I can design inputs statements that will take inputs from Professor Ahmad and then feed those inputs to the model.

This way:

```
study_hours = int(input('How many hours your student study per day: '))

marks_previous = int(input('How much he scored in his previous grade give
your answer in percentage: '))

mp = marks_previous / 100

class_performance = int(input('What is class performance of student out of
10: '))

prediction = mdl.predict([[study_hours, mp, class_performance]])

print('Predicted marks: ')
print(prediction)
```

Explanation:

There are two ways of giving test data, either you can give list containing features (the way we are doing yet) or you can define your variables. In the above code, I am giving this privilege to the professor that the model will one by one ask him about the features.

Defining variables in Python

1) Variables are like containers that hold values, and you can assign different types of data to them.

2) User can define the value of variable by writing it in input form

```
study_hours = int(input('How many hours your student study per day: '))
```

Here study_hours is our variable and by giving **input** as a key word, when the code is run, it asks the user the question, **'How many hours your student study per day: '**

3) Outside the key word input, we wrote int, this is because we want data type of our feature to be integer, not string!

```
mp = marks_previous / 100
```

Here, we are dividing previous marks by 100, because they are in percentage form. For example: 50%, 85% and so on but we want our data to be in same format as it was in the training data. It will make valid predictions when test data and training data are in same format.

Predicting on variables!

After we have defined our variables, we now make predictions with our variables.

The code will ask previous marks, study hours and class performance of a student. Professor Ahmad will answer values one by one. And then model will predict the score based on these variables (answers from professor).

The benefit of defining variables is that the model now becomes a lot user friendly.

Test run:

```
In [28]: study_hours = int(input('How many hours your student study per day: '))
         marks_previous = int(input('How much he scored in his previous grade give you answer in percentage: '))
         mp = marks_previous / 100

         class_performance = int(input('What is class performance of student out of 10: '))

         prediction = m.predict([[study_hours, mp, class_performance]])

         print('Predicted marks: ')
         print(prediction)

         How many hours your student study per day: 7
         How much he scored in his previous grade give you answer in percentage: 83
         What is class performance of student out of 10: 8
         Predicted marks:
         [82.]
```

Congratulations! Now our model is ready for Professor Ahmad.

Quiz 2

1. **What keyword is used to train model?**

- model.train()
- model.fit()

2. **What keyword is used to make prediction from our model?**

- model.predict()
- model.guess()

3. `int(input('write your age: '))` **What is role *'int'* in this code?**

- Takes input from user
- Specifies that our answer should be in form of integer.

Lecture 14: Predictions at once

Let us now give data of bunch of students to our model and let's see how well our model predicts.

Note that we can also pass data of more than one student at once to our model. Professor Ahmad won't have to input data of each student one by one, he can input data of as many students as he wants in a single go.

Until now, we were having predictions using this line of code:

```
mdl.predict([[7, 0.83, 8.0]])
```

Note that there are two [ ] brackets. This mean it is a list within list.
Square brackets at the edges can be referred as parent list, and square bracket inside can be referred as kid list.

We always give data points in the child list. What will happen if we increase our child lists? Each child list contains data of one student so more child lists will contain data of more students. By doing this, we can predict marks of multiple students in one go.

Look at upper code,

First argument is number of study hours, 7.

Second argument is percentage of previous class, 0.83.

Third argument is performance in class out of 10, which is 8.

We will have to give data in the same arrangement.

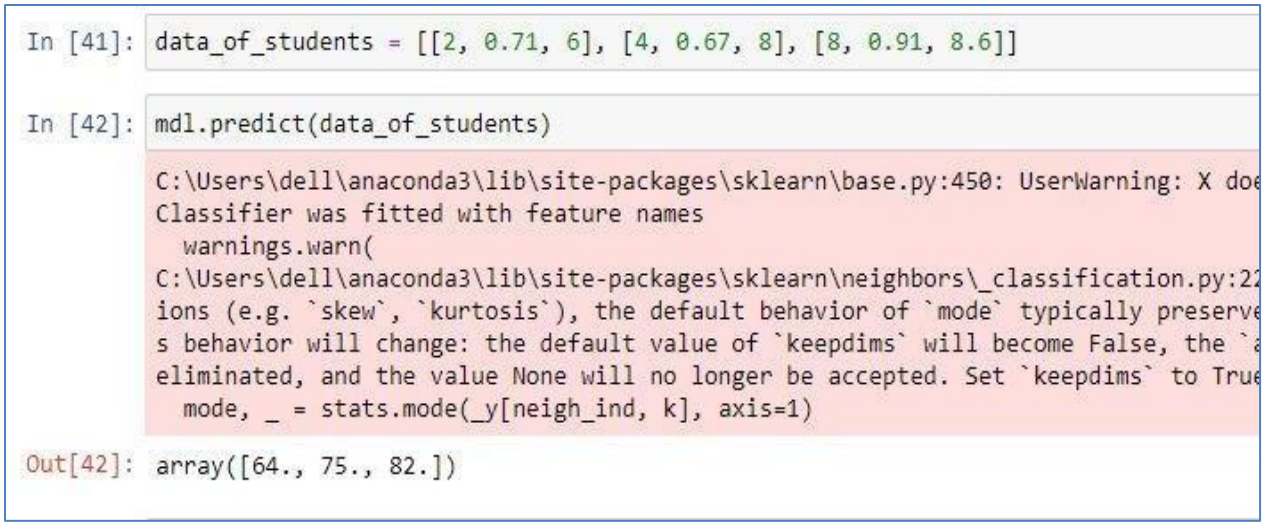
```
data_of_students =  [[2, 0.71, 6], [4, 0.67, 8], [8, 0.91, 8.6]]
```

Here I have defined a variable *'data_of_students'* in which there are lists within list. Each list contains data of a specific student.

We already have a model, let's now pass this data into our model and let's see how predictions are made.

```
mdl.predict(data_of_students)
```

Output is:



There are three outputs for our three data points (list of data). And this makes sense.

Have a look at data of first student, who study 2 hours a day, scored 71% in previous exams and class performance is 6. Our model is saying that this student will score 64% marks in annual exams.

And for the third student, who is a book freak, studies 8 hours a day, have scored 91% in previous exams and class performance is 8.6 out of 10. And our model predict that he will score 82% in annual exams, which is pretty good prediction. Note that these predictions are approximate marks, which the student will score in annual exams.

Lecture 15: How our KNN model work:

We have the model which is making good predictions so far. But we don't know how this model works and make predictions. Let's discuss that now. As mentioned before and seen in the code, we are using **KNeighbors Classifier.**

Kneighbors classifier or nearest neighbors classifier is one of the major machine learning algorithm and also considered as simplest one.

To make a prediction for a new data point (student data), the algorithm finds the closest data points (closest student data) in the training dataset.

Now, read the bold line again.

Let's breakdown this algorithm.

Let suppose I am a bad person, and you are a good person. And then comes another person who is your best friend. That new person is closer to you than to me. Now any other person may think that this new person, who is your friend, is a good person. Well, the logic is clear, you are a good person so your friend will be considered good and likewise my friend may be considered as a bad person because I am a bad person.

Now take you and me as data points. You are a data point and your label is *'good person'* and I am another data point and my label is *'bad person'.* After this, take that new person as the data point that we want to predict. As that new person is closer to you, we will classify him as

'good person'. This is how nearest neighbors work. It finds the nearest data point to the new data point. Then it classifies new data point same label as of nearest data point's label.

In our case, it finds you more close to that new friend of yours and hence classified that friend same label as yours, which is *'good person'*.

This is how nearest neighbors classifier works. 'neighbors' refer to the data points.

Let's now see how it implements on our dataset.

Our dataset was:

	A	B	C	D	E
1	Study hours	Marks in previous grade	class performance out of 10	Marks scored	
2	3	72%	7.2	75%	
3	1	69%	6.3	72%	
4	5	87%	8.6	84%	
5	2	75%	7.9	78%	
6	3	78%	7.9	76%	
7	5	85%	9.2	94%	
8	4	82%	9.1	89%	
9	2	71%	7.3	78%	
10	3	75%	6.9	73%	
11	1	65%	6.2	64%	
12	4	80%	8.1	82%	
13	6	91%	9.5	93%	
14					
15					

This was the data on which we trained our model. Recall that column 2 and 4 was converted into point values. Each value was divided by 100. For example: 72% will be considered as 0.72 and same way 69% will be considered as 0.69.

We already have trained a model on this data and have made predictions through it. This was one of the codes we used in this section.

```
mdl.predict([[7, 0.83, 8.0]])
```

Now let's see any closest data point to this data. Have a look at dataset and try to find any student who has approximately same features as this new student on which we are doing prediction.

We have to find any student from our data, who approximately:

Study 7 hours a day

Have scored 83% marks in previous class

Class performance is 8

Take some time and find that data point from upper image of data

.

.

.

We may find many students close to this new data.

For this case I think we can consider two students, *row 12* and *row 4*. Data of these two students is very close to our new student. These are not very close, but closer than others. Well python won't have this confusion of *'approximately'*. Python will measure exact distance between data points mathematically and will have exact one answer.

Based on our observation, row 4 and row 12 students can be considered as *'nearest neighbor'*. Row 4 student have scored 84% marks and row 12 student have scored 82% marks. Our model will predict one of these marks for new student.

```
In [25]: mdl.predict([[7, 0.83, 8.0]])

C:\Users\dell\anaconda3\lib\site-packages\sklearn\base.py:45
Classifier was fitted with feature names
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\sklearn\neighbors\
ions (e.g. `skew`, `kurtosis`), the default behavior of `mod
s behavior will change: the default value of `keepdims` will
eliminated, and the value None will no longer be accepted. S
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

Out[25]: array([82.])

In [ ]:
```

As expected, 82% marks is one of the marks we thought model will predict.

What model has done? Model has calculated mathematical distances of new data point to every data point in our training dataset, the most least distant data point is nearest data point. For our case it looks like 12th row has least distance to our new data point and hence its label (82) is given to new data point (Predicted value).

Now go at the top of this lecture and again read that bold statement. And if possible, read this whole lecture again after 15-20 minutes.

"To make a prediction for a new data point (student data), the algorithm finds the closest data points (closest student data) in the training dataset"

We haven't discussed how our model calculates the mathematical distances. Here is the brief information about how those calculations are made:

- We plot data points in 3 dimensional planes (as there are three features).
- We plot all training data points in the three-dimensional plane.

- When a new data point is added (test data point), we plot that in the same coordinate system.
- Now we have coordinates of test point and training points. We find the distance of test point to every training point, and the point with least distance is selected as nearest neighbors.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Model calculates distance using this formula which you may remember from your high school (formula shown is for 2-dimensional plane).

Lecture 16: Training model on more than one neighbor

Until now we didn't specified number of neighbors (by default model automatically select 1 neighbors) but we will do that in next example of nearest neighbor classifier.

When training for more than one neighbors, model will not look for only one nearest neighbor, it will gather n number of nearest data points and will do voting. The majority label will be selected as final label.

For example, if task of my model is to classify you as a good or a bad person, it will find 10 close friends of yours. Then it will analyze number of good friends and number of bad friends. If you have more good friends, then you will be classified as good person and vice versa.

It’s not mandatory that you understand things in the first attempt. It can take time and that's the boring part of learning something. Take it slowly and steady. If this lecture doesn’t make sense to you now, leave it.

Lecture 17: All in once

We are done with basic knowledge of machine learning and have trained our first ML model.

Let's have a look what we coded in this section. I will not be explaining whole code now. You will take a look at each line of code and make sure that you understand everything.

Dataset:

	A	B	C	D	E
1	Study hours	Marks in previous grade	class performance out of 10	Marks scored	
2	3	72%	7.2	75%	
3	1	69%	6.3	72%	
4	5	87%	8.6	84%	
5	2	75%	7.9	78%	
6	3	78%	7.9	76%	
7	5	85%	9.2	94%	
8	4	82%	9.1	89%	
9	2	71%	7.3	78%	
10	3	75%	6.9	73%	
11	1	65%	6.2	64%	
12	4	80%	8.1	82%	
13	6	91%	9.5	93%	
14					
15					

Coding to train a model on this data:

```
import pandas as pd
data = pd.read_excel('D://Professor ahmad class.xlsx')
features = data[['Study hours', 'Marks in previous grade', 'class performance
out of 10']]
label = data[['Marks scored']]
final_label = label * 100

from sklearn.neighbors import KNeighborsClassifier
mdl = KNeighborsClassifier()
```

```
mdl.fit(features,final_label)

mdl.predict([[7, 0.83, 8.0]])
```

Last line of code is to make prediction. For more than 1 predictions at once, we used this code:

```
data_of_students = [[2, 0.71, 6], [4, 0.67, 8], [8, 0.91, 8.6]]
mdl.predict(data_of_students)
```

To make this code more user friendly by taking inputs from user, we updated our code this way:

```
study_hours = int(input('How many hours your student study per day: '))
marks_previous = int(input('How much he scored in his previous grade give you
answer in percentage: '))
mp = marks_previous / 100

class_performance = int(input('What is class performance of student out of
10: '))

prediction = mdl.predict([[study_hours, mp, class_performance]])

print('Predicted marks: ')
print(prediction)
```

Have a good grip on this lecture and practice the codes along with.

Assignment 1: Machine learning assignment

Make neat notes of nearest neighbor classifier in your notebook. This practice will you confidence on your understandings.

Discuss how it works. Don't focus on decorations. Make neat notes so that your notes will be of advantage for you in future.

Also try to add short code snippets.

Assignment Instructions
Get your paper and pen and create notes. This may sound childish but this will make your concept clearer.
Also try to find someone like your sibling or friend and explain to them how nearest neighbor classifier works and show them your machine learning model. If you don't find anybody, create a video with your screen or notes or whatever, with your voice over (any way you are comfortable with) explaining how this model works, and send that video to me. I will reply I guarantee. (harmanwaheed@gmail.com) or ping me on my instagram *@fly_fazaia*.

Also discuss how neighbor classifier is logically correct for predictions for our dataset?

Quiz 3: Predictions from our model

	A	B	C	D	E
1	Study hours	Marks in previous grade	class performance out of 10	Marks scored	
2	3	72%	7.2	75%	
3	1	69%	6.3	72%	
4	5	87%	8.6	84%	
5	2	75%	7.9	78%	
6	3	78%	7.9	76%	
7	5	85%	9.2	94%	
8	4	82%	9.1	89%	
9	2	71%	7.3	78%	
10	3	75%	6.9	73%	
11	1	65%	6.2	64%	
12	4	80%	8.1	82%	
13	6	91%	9.5	93%	
14					
15					

This is our data. A new data point, a student, is: (3, 0.74, 6.8).

To which data point, our new data point is closer? Answer in terms of row number.

3 13

6 10

If we pass the data point, same as in previous question, to our model, what will be the output?

```
mdl.predict([[3, 0.74, 6.8]])
```

73 78

```
Correct answers: 10, 73
```

Lecture 18: Drawbacks of neighbors Classifier

Till now, we have discussed much about Kneighbors classifier, let's see if there are any drawbacks in our model.

LECTURE 18: DRAWBACKS OF NEIGHBORS CLASSIFIER

Quick-Revision: To make prediction on a new data point nearest neighbors classifier find nearest data point to the new data point and then make prediction based on label of that nearest data point.

It's fine if you have to read a sentence or a paragraph again and again for better understanding or just understanding.

Let us consider a student, *Mujtaba*, who plays video games all day and study 0 hours a day.

```
In [55]: study_hours = int(input('How many hours your student study per day: '))
         marks_previous = int(input('How much he scored in his previous grade give you answer in percentage: '))
         mp = marks_previous / 100

         class_performance = int(input('What is class performance of student out of 10: '))

         prediction = mdl.predict([[study_hours, mp, class_performance]])

         print('Predicted marks: ')
         print(prediction)

         How many hours your student study per day: 0
         How much he scored in his previous grade give you answer in percentage: 39
         What is class performance of student out of 10: 2
         Predicted marks:
         [64.]
```

Given data:

1. Study hours: 0.
2. Marks in previous grade: 39%.
3. Class performance out of 10: 2.

For *Mujtaba*, our model predicts that he will score 64% in annual exams

Drawbacks of KNeighborsClassifier (in context of our model) :

Highest label (score) in our dataset: 91.

Lowest label (score) in our dataset: 64.

Now, if we introduce a new student, *Alina*, as test entry and her study hours are 16, previous class score is 100 and class performance is 10, we might assume that the predicted score should be 100 in this case as well, but our model will predict 91 because it is the last entity/range till which our model is trained (look at the dataset). But we might expect that this student will score even higher.

1. **Study hours: 16**
2. **Previous grade: 100**
3. **Class performance: 10**

Model prediction: 91%

Thinking logically, this student, *Alina* will perform better based on her provided data.

In the same way, observe the above given image (data for *Mujtaba*), we gave input of 0 study hours, 39 percent previous score and 2/10 in class performance, here the predicted score of 64 might sound false to us and it is actually is weird for someone with 0 study hours to score 64%!

1. **Study hours: 0**
2. **Previous marks: 39**
3. **Class performance: 2**

Model prediction: 64

Thinking logically, this student, *Mujtaba*, will perform even lesser marks.

In order to overcome this drawback, we will introduce another algorithm to you, which is Linear Regression. We will train Linear Regression model on the same dataset, so that the difference will be clear and we will also get an idea about when to use linear regression model and when to use neighbor classifier model.

Lecture 19: K in Kneighbors Classifier

K in neighbor classifier deflect the number of neighbors on which we train our model. We will explore further about changing number of neighbors in coming lectures (Section 4). For now, we will be exploring another important machine learning model.

SECTION 3: LINEAR REGRESSION (ML ALGORITHM)

Lecture 20: What is Linear Regression and why use this

As discussed before, there are many algorithms of machine learning, and Linear Regression is one of the most used algorithm.

As the name says, it is some kind of linearity and it is a regression model. To revise the concept of regression and classification, read the section 2 lecture 10.

Quick Revision: When we know the number of classes that we want to predict, it is called Classification and when the predictions can be any number, there are no defined number of classes, it will be called Regression. Linear regression if a model for regression problems.

Linear Regression is the model that is designed to predict on continuous data (regression problems). **Neighbor classifier was having the defect that it could make predictions only from the labels that it is trained on** (*lecture 18*).

We hope you remember the defects from our nearest neighbor model. Now, to overcome those defects we are going to use linear regression model.

Here is brief idea of math behind linear regression models:

- When training, model tries to draw a straight line that is closest to all the data points. (Take distance of each data point to that line, take square of each distance and add all answers, that is mean squared error.)

- In simpler words, that line is closest to all the training data points.
- Based on that straight line, our model will make predictions for the coming unseen data.

Lecture 21: Making our Linear Regression model

Step 1 - Preparing data

As we already are familiar how we load our data in python, separate features and labels… now it will be very easy to code linear regression model.

```
# loading the dataset
data = pd.read_excel('D://Professor ahmad class.xlsx')

# specifying features and labels
features = data[['Study hours', 'Marks in previous grade', 'class performance
out of 10']]

label = data[['Marks scored']]
final_label = label * 100
```

Lines containing *'#'* in the beginning are not part of code, they are written for our instructions.

This code is same as we used before. If anything is not clear, you can navigate to *lecture 11 and 12*. This code is explained in those lectures step by step.

This is how our features look like:

```
In [32]: features = data[['Study hours', 'Marks in previous grade', 'class performance out of 10']]
         features.head()
```

Out[32]:

	Study hours	Marks in previous grade	class performance out of 10
0	3	0.72	7.2
1	1	0.69	6.3
2	5	0.87	8.6
3	2	0.75	7.9
4	3	0.78	7.9

```
In [ ]:
```

And this is how our labels look like:

```
In [43]: final_label.head()
```

Out[43]:

	Marks scored
0	75.0
1	72.0
2	84.0
3	78.0
4	76.0

```
In [ ]:
```

As we have our dataset loaded and made variables in which we have stored features and labels, let's train our linear regression model.

Step 2 - Importing linear regression

```
from sklearn.linear_model import LinearRegression
```

This line of code will import linear regression in our python environment

Step 3 - Initializing model

Once imported, firstly, we have to define a model, which we will be train later in this lecture.

```
model = LinearRegression()
```

Same way with which we were initializing nearest neighbor model, here we are initializing linear regression model.

Step 4 - Training our model

Using same way through which we trained our neighbors classifier model, here we will train our linear regression model.

Quick-revision: While training we pass features as first argument and labels as second argument

```
model.fit(features, final_label)
```

That's it! Our model is ready to use.

Lecture 22: Performance of our model

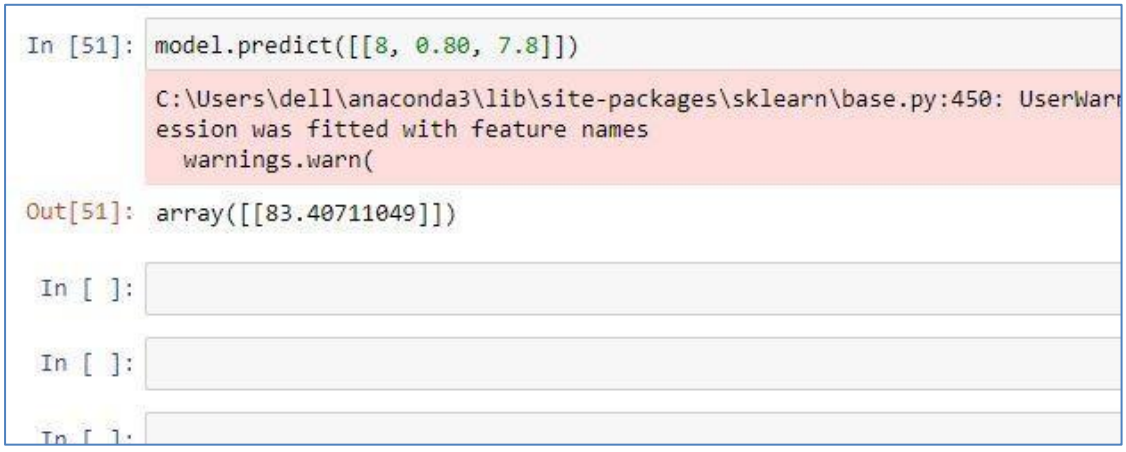
By the end of previous lecture, we have trained our linear regression model on the data provided by *Professor Ahmad* provided to us. Now, it's time to check if there are any improvement in predictions. If so, we will have a more efficient model which we will provide to *Professor Ahmad*.

To make a prediction, syntax is same as of neighbor classifier, which we discussed in *section 2*.

We use this line of code:

```
model.predict([[8, 0.80, 7.8]])
```

In the new data I have provided a student who study 8 hours a day, has scored 80% marks in previous exams, and class performance is 7.8 out of 10. You can change the values surely.



The predicted score (annual exam marks prediction) by our linear regression model is: **83.40.**

Next prediction will give further clarity.

We discussed a student, *Mujtaba*, in *lecture 18* who study 0 hours a day, have scored 39% in previous exams and class performance is 2.

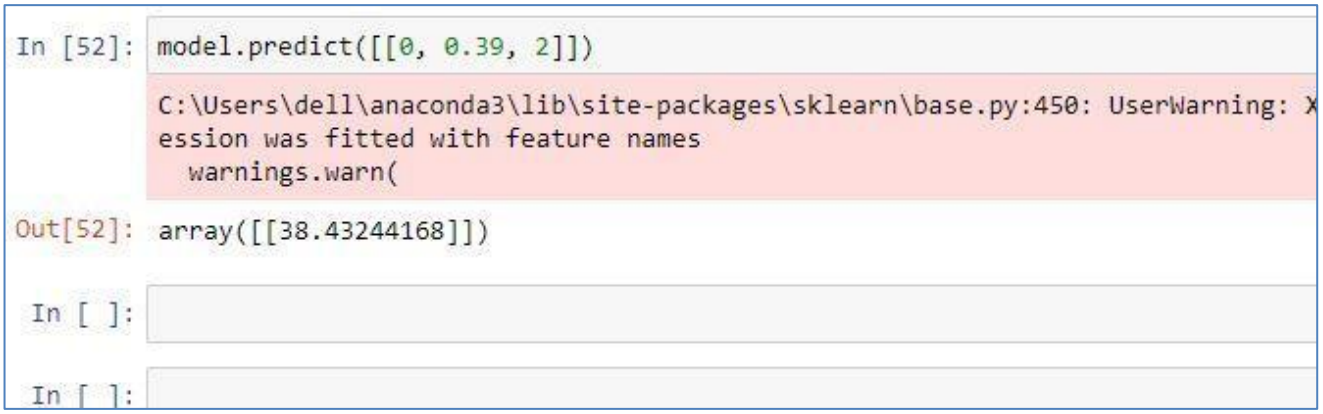
When we passed this data to neighbors classifier, our prediction was 64% and this is pretty unrealistic that student with this record will score 64% in annual exams. The reason that our model predicted 64% was:

- The data point with lowest marks in our training set was with marks 64%. The features of this least marks student were closest to the new student that is why our model (KNN model) predicted 64% score for new student: *Mujtaba*.

Now let's give same data to linear regression model see the predicted score.

```
model.predict([[0, 0.39, 2]])
```

Output:



Great improvement in our model!

Now, for the same student, our model is predicting that he, *Mujtaba*, will score 38.43 % in annual exams. (Based on the predictions made, this model sound more accurate than the nearest neighbor model)

Data of student:

1. Study hours: 0
2. Previous grade: 39%
3. Class performance: 2

Prediction made by Kneighbors classifier: 64%

Prediction made by Linear Regression model: 38.43%

Based the student data, prediction made by linear regression seems to be more accurate.

Furthermore, we can make our model more user friendly by the techniques we used while creating nearest neighbors model (taking inputs from user, and passing multiple lists in single statement).

Lecture 23: Line of best fit

To understand how predictions are made through linear regression model, let's first understand how to plot data points in 2d plane.

Assignment 2: Plotting data points

Take your notebook out and plot the data.

ASSIGNMENT 2: PLOTTING DATA POINTS

	Study hours	class performance out of 10
0	3	7.2
1	1	6.3
2	5	8.6
3	2	7.9
4	3	7.9
5	5	9.2
6	4	9.1
7	2	7.3
8	3	6.9
9	1	6.2
10	4	8.1
11	6	9.5

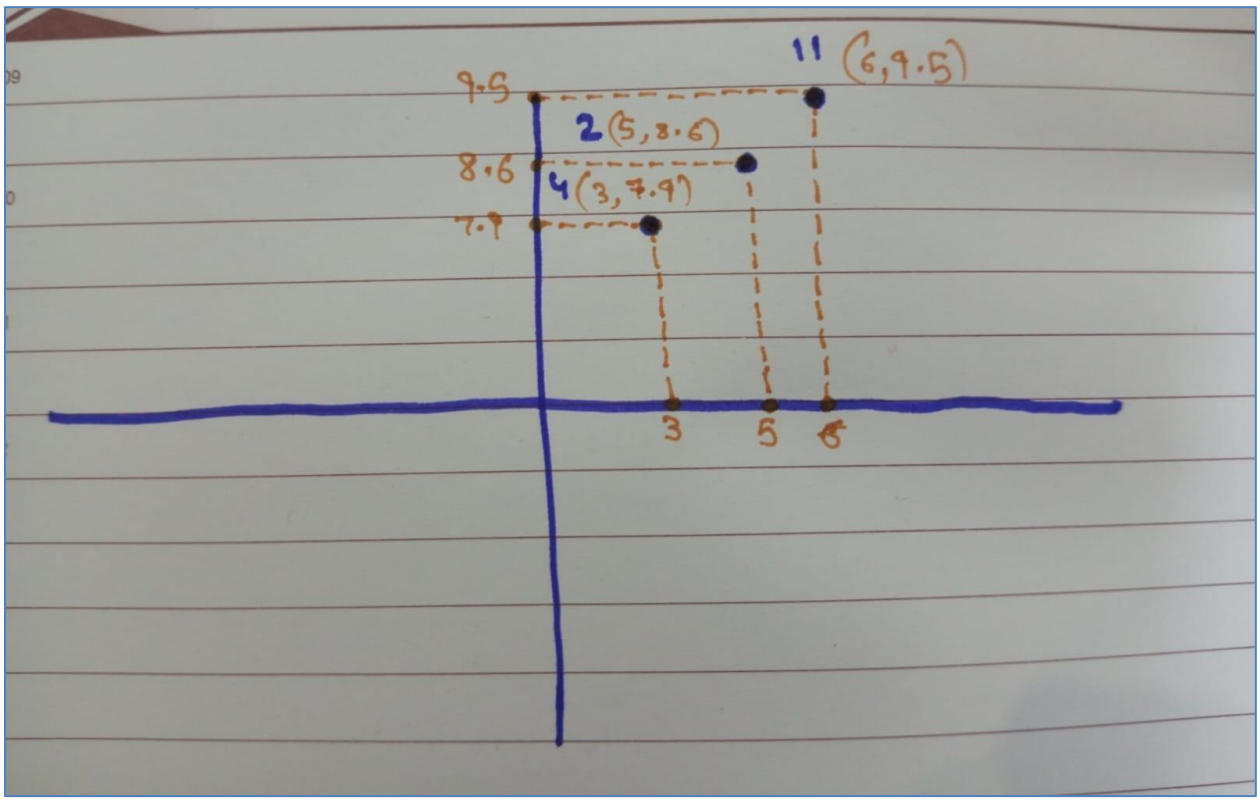
How to plot this? You may remember from your primary math book how we plotted x and y in a two-dimensional plane. We draw two lines cutting each other (x axis and y axis). Then we plot data points on those axes.

Now consider study hours as x and class performance as y.

For the first data point, we will plot (3, 7.2) in our x, y axis and so on. (see the image of data)

To make it clearer, I will plot few data points (2nd, 4th and 11th) from our dataset.

Assignment 2: Plotting data points



Orange dotted lines are just for instructions. Data points are marked with marker.

With each data point, I have mentioned coordinates and row number of data point that I plotted from dataset.

You have to mark each data point at right place. Furthermore, you don't have to draw those dotted lines if you don't want to, and you also don't have to mention coordinates. Just make a neat graph for your understanding.

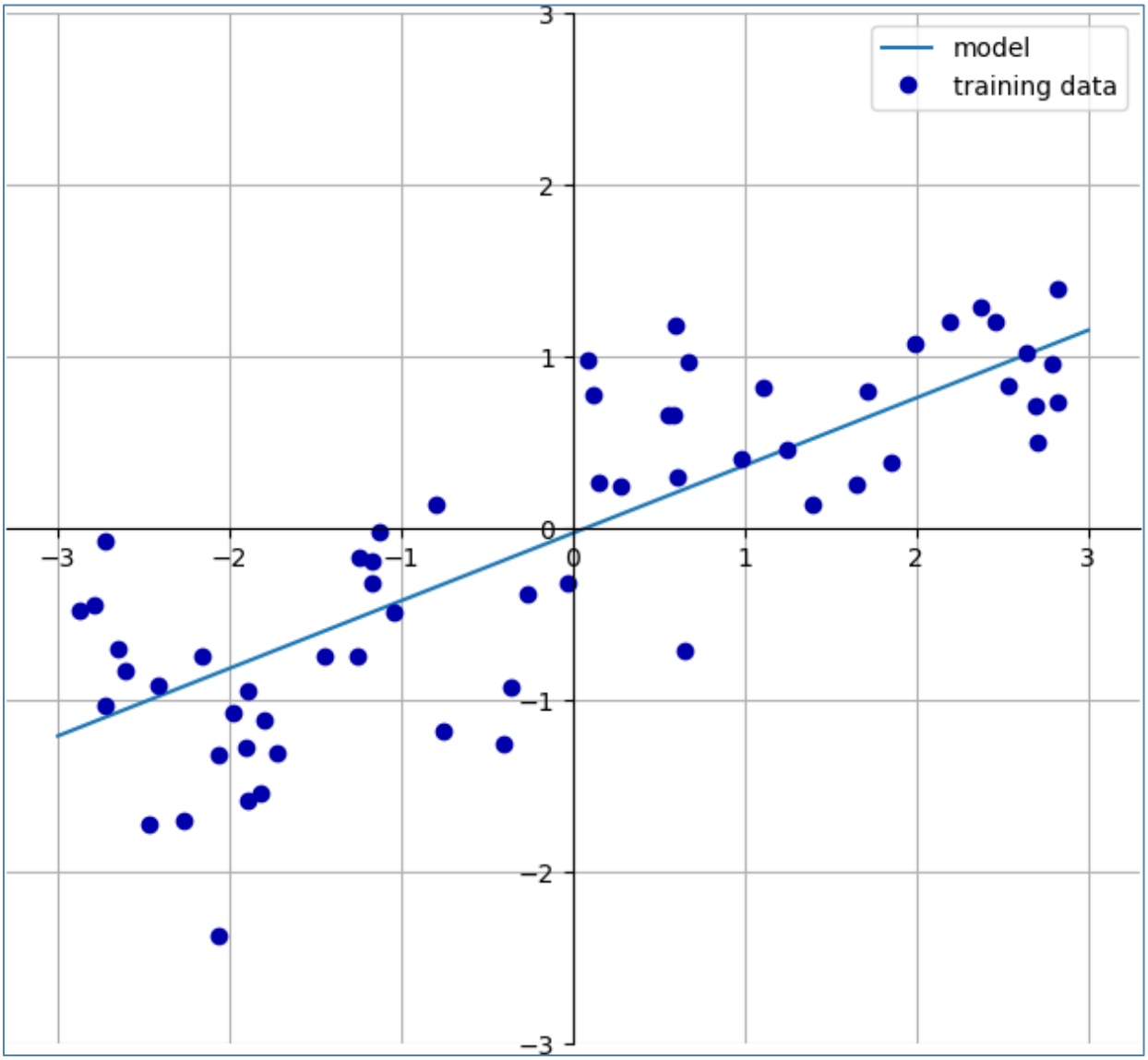
Question:
What is x and y from our dataset?

x is study hours in our dataset and y is class performance. You may have taken y as study hours and x as class performance. More suitable idea is first one.

Lecture 24: Line of best fit, part 2

Now, as you understand how to plot data,

Have a look at this graph (this is not graph of our dataset):



There are many data points scattered in this graph in blue color. To mark the data points, same logic has been used as used in the assignment.

Note that there is a blue straight line in the graph. This line is plotted in such a way that it is closest to all the data points. This is what our linear regression model does, it plots all the data points and then plot a line that is closest to all the points, and that line is known as line of best fit. Afterwards, predictions are made through this line of best fit.

SECTION 4: KNEIGHBORS CLASSIFIER (ML ALGORITHM)

Lecture 25: When to use nearest neighbors classifier

As we discussed, for *Professor Ahmad*, linear regression model was working more efficient than nearest neighbors. But that's not the case always. It depends on the dataset we are working with.

There are various cases in which nearest neighbors work more efficiently.

Why?

As discussed in *lecture 9*, there are two kinds of supervised machine learning problems and there are different algorithms for those. Those two kinds are:

1. **Classification problem.**
2. **Regression problem.**

Classification problems have its own algorithms, and regressions problems have its own algorithms.

Nearest neighbor classifier is used for classification problems and linear regression is used for regression problems. But what type of problem were we dealing with?

It was a regression problem (marks of students are continuous data). As it was a regression problem, that's why performance of linear regressions was considerably more accurate.

Linear regression is algorithm for Regression problems

Nearest neighbor is algorithm for classification problem

It's important to recognize the type of problem you are dealing with.

Lecture 26: Iris data, classification problem

Now we will be working on a classification problem. You can download that dataset here

Here: https://www.kaggle.com/datasets/vikrishnan/iris-dataset

Iris dataset is very famous dataset, on which we train classification models. It actually contains a classification problem, that's why will be using classification algorithm for it.

Here is the overview of dataset: There are total 5 columns in dataset which are petal length, petal width, and sepal length, sepal width and species.

```
In [27]: import pandas as pd

In [51]: df = pd.read_csv('D://datasets/iris.csv')

In [55]: df.head()
```

Out[55]:

	petal length	petal width	sepal length	sepal width	flower
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [ ]:
```

We loaded the dataset and prints its first 5 rows by using head() attribute.
Using this, we get a clear idea of how our dataset looks like.

Data description: A botanist did this research and measures these features (petal and sepal dimensions) of 3 categories of flowers which are **setosa, virginica and versicolor** (these three are labels). By using shape attribute, we see that there are 150 rows.

```
In [56]: df.shape
Out[56]: (150, 5)

In [9]:
```

That means there are 150 data points in our data and total 5 columns. For each of the data point, we have its type in the fifth column.

- Features columns are: Sepal length, sepal width, petal length, petal width.
- Labels column: flower. It basically contains specie of the flower.
- There are total 3 species in our dataset, we can say that there are 3 possible labels. These labels are: setosa, versicular and virginica.

Why is it a Classification problem? Because the number of labels are defined (only 3).

Now what we'll be doing is that we are going to train a classification model on this dataset and that classification model will predict the specie of flower, and that prediction will be based on sepal and petal dimensions.

Let's build our 1st classification model!

Importing nearest neighbors classifier:

```
from sklearn.neighbors import KNeighborsClassifier
```

After importing KNeighborsClassifier, we have to define our features and labels. Features are the data we give our model as input and label is the output we want. In this case, the features will be the petal length, petal width, sepal length, sepal width and the label will be 'flower' column which contain specie name of each of the flower.

You already know how we separate features and labels!

```
features = df[['petal length', 'petal width', 'sepal length', 'sepal width']]
label = df['flower']
```

In *section 2*, we used very similar code to separate features and labels.

The syntax for defining features and label is pretty simple. We are simply using square brackets to extract the wanted columns from our data set. The variable we used "features" and "label" is obviously changeable, you can use whatever you want.

By convention, mostly used variables are X for features and y for labels but here let's go along with features and labels as we defined them.

Training model

Firstly, we define the model.

```
model = KNeighborsClassifier()
```

Fit the data (features and labels).

```
model.fit(features, label)
```

Now we can easily make predictions by giving new data points. Let suppose you have a flower in your garden. The petal length of flower is **2cm**, petal width is **3cm**, sepal length is **1cm** and sepal width is **2cm**.

We will make our model predict the specie of our flower.

```
model.predict([[2, 3, 1, 2]])
```

OUTPUT: SETOSA

But do we know the science behind all this?

I hope you remember *lecture 15*, which was all about how nearest neighbors classifier make predictions. It is suggested to read that lecture again.

Quick-revision: KNeighborsClassifier classifies new data point same as nearest neighbor of that point, as the name suggest.

Changing Number of Neighbors

Reminder: When given unseen data, neighbors classifier finds nearest data point to the new data point and gives label of nearest data point to the that new data point.

Reread lecture 15 and lecture 16 for better understanding.

Again, imagine you are a good person, and I am a bad person. A new person who is close friends with you will be classified as a good person because you are good.

Now, instead of just one friend, we will check their 3 closest friends. If most of them are good, the person will be classified as good. If most are bad, they will be classified as bad.

Example 2:

Ali has 7 classmates in his study group. To decide if Ali is a good or bad person, we will check his 7 study group members. If most of them are good, Ali will be classified as good. If most are bad, Ali will be classified as bad person.

- In the upper two examples, we basically increased number of neighbors... based on which predictions are made.
- We can easily change number of neighbors in K neighbor classifier.

To change number of neighbors, we simply add an extra detail when initializing the model. Everything else remains same.

```
model = KNeighborsClassifier()
```

This is how we were initializing the model until now. Now, we can play around with the number of neighbors by initializing the model this way:

```
model = KNeighborsClassifier(n_neighbors = 3)
```

- We have defined `n_neighbors = 3`
- It means that we will look for 3 nearest neighbors to make prediction.
- Everything else (features, labels, training, etc.) will remain same.

SECTION 5: EVALUATING PERFORMANCE

Lecture 27: How well our model works?

In this section, we will discuss how to evaluate the performance of our models. Remember from previous section where we created a model that predicts the category of flower. We provided a data point to our model and it gave us a prediction.

But how do I know that the model is predicting correct or not?

This was one of the predictions we made:

```
In [9]: model.predict([[2, 3, 1, 2]])

C:\Users\dell\anaconda3\lib\site-packages\sklearn\base.py:450: User
Classifier was fitted with feature names
  warnings.warn(
C:\Users\dell\anaconda3\lib\site-packages\sklearn\neighbors\_classi
ions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typi
s behavior will change: the default value of `keepdims` will become
eliminated, and the value None will no longer be accepted. Set `kee
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)

Out[9]: array(['Iris-setosa'], dtype=object)

In [ ]:
```

Maybe flower wasn't setosa and our model has made a mistake... Let's explore the method, through which we will check accuracy of model.

Lecture 28: Splitting our data

Splitting the data into two chunks is most widely used method to evaluate the model performance. For example, in the iris dataset we were having 150 labeled data points.

We will divide whole data into two chunks:

- One containing 100 data points
- Other containing 50 data points.

We will train our model on 100 data points. After that we will make predictions on those 50 separated data points. Because that data was already labeled, so we know the original labels of those 50 data points.

Now we will just check whether the predicted label matches with original label or not. Hence, we can analyze how many times (from 50) the model has made correct prediction and how many times it was wrong.

WAIT, YOU SHOULD READ UPPER THREE PARAGRAPHS AGAIN

Training data

The data on which we train our model is called training data.

Test data

The data on which we check our model performance is called test data.

Normally we train model on 70% of our dataset and test on remaining 30%.

Let's apply this technique on iris dataset.

Lecture 29: Checking accuracy of model

To split our data into two parts, training data and test data, we have a module known as *train_test_split.*

We suppose that dataset is already loaded in python environment. Let's import required module

1 - Import module

```
from sklearn.model_selection import train_test_split
```

2 - Specifying features and labels

We have used this code many times to specify labels and the features.

```
features = df[['petal length', 'petal width', 'sepal length', 'sepal width']]
label = df['flower']
```

Quick-revision: Iris dataset was a flowers dataset with 4 features of a flower, those are: petal length, petal width, sepal length, sepal width. 5th column specifies the category of flower. There are only three possible categories: Setosa, Virginica, and Versicular. As the number of labels are specified, this is a classification problem.

```
In [27]: import pandas as pd

In [51]: df = pd.read_csv('D://datasets/iris.csv')

In [55]: df.head()

Out[55]:
```

	petal length	petal width	sepal length	sepal width	flower
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [ ]:
```

3 - Train-test-split

After specifyinh features and labels, let's split those.

```
X_train, X_test, y_train, y_test = train_test_split(features, label)
```

There are four variables created here.

1. X_train: contain training data points (training features only).
2. X_test: contain test data points (test features only).
3. y_train: contain training labels (training labels only).
4. y_test: contain test labels (test labels only).

We have to train our model on training data points (features) and training labels. Note that *X_train* contain training features and *y_train* contain training labels.

4 – Training the model

```
model = KNeighborsClassifier()
model.fit(X_train, y_train)
```

Notice that we have trained our model on training data points (not on whole data).

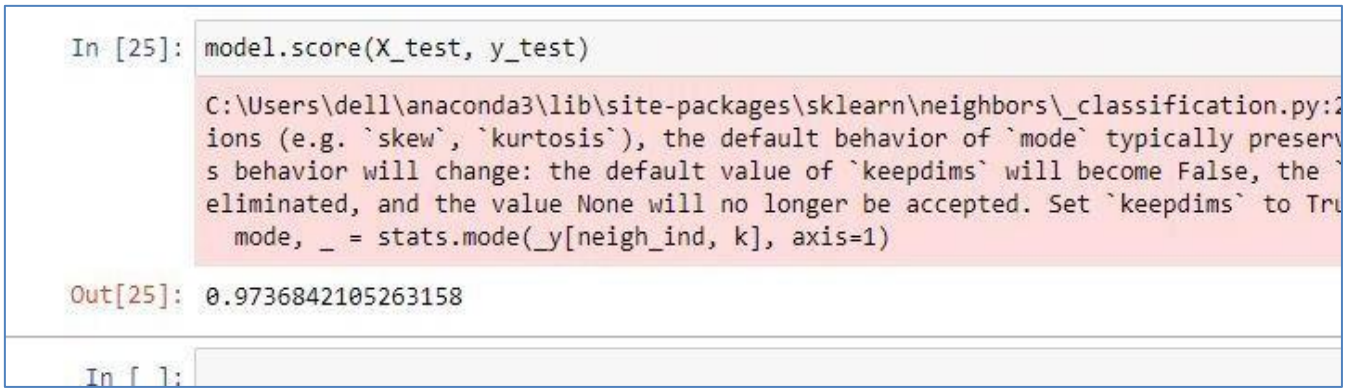
5 - Accuracy of our model

Now to check the accuracy of model, we pass test features and test labels. Note that *X_test* contain testing features and *y_test* contain test labels.

To check the accuracy, we use this line of code:

```
model.score(X_test, y_test)
```

Output:



Great! Our model has achieved an amazing accuracy of 97%. That's really ASTONISHING.

How this accuracy is measured?

Note that *X_test* contain all data points we reserved for test purpose and *y_test* contain labels for those data points.

To calculate the accuracy, we used *.score* method of model, which automatically do the stuff for us. But what is that?

When we use *.score()* method, it will make our model predict on EACH of the data point of X_test and then will check whether the predicted label matches with actual label or not (actual labels are stored in *y_test*). Hence, we will have number of correct predictions and number of incorrect predictions.

To calculate the accuracy, we use this formula:

Accuracy = correct predictions / Total number of predictions

Let say there were total 30 test data points and our model predicted 20 with correct label and 10 with incorrect. To calculate the accuracy of this model, put the values in upper formula:

Total prediction = 30

Correct predictions = 20

Accuracy = 20 / 30 = 0.66

Note that this score is out of 1, to get the percentage out of hundred, multiply this value by 100. So, we can say that the model accuracy is: *0.66 * 100 = 66%.*

We don't have to do this calculation manually; we can just use *.score* method as done in the image above. It will do all the calculations and show the accuracy of model.

NOTE: Testing model on the data on which model is trained is not logically correct. Consider you are a math teacher. You teach your students addition of two numbers. To check how well they understand the concept of addition, you will assign them few DIFFERENT questions of sum and they (your students) will answer those.

Same way, we train our model on different data, and test that model on different data.

Quiz 4: How to calculate model accuracy, MCQ

How do we analyze how well our model works?

1. Divide data into two chunks, train model on one chunk and test it on other chunk.
2. Test the model on data on which model was trained
3. There is no such technical way to do that

SECTION 6: LOGISTIC REGRESSION

Lecture 30: What is Logistic Regression

Despite its name, Logistic Regression is a classification algorithm and not a regression algorithm, and it should not be confused with Linear Regression. **Logistic regression is used for classification problems.**

Logistic regression model make prediction for every possible outcome. What does it mean?

Let say there are three possible outcomes for a dataset: Setosa, Versicular, and Virginica (from iris dataset). When a new datapoint is given and logistic regression model make prediction for that, it will output three values:

- Probability for Setosa.
- Probability for Versicular.
- Probability for Virginica.

Label with highest probability is considered as predicted value. For example, if probability for Setosa is highest, we will say that our model has predicted that the flower is Setosa.

Logistic regression model make prediction for every possible outcome. Outcome with highest probability is considered as predicted value by logistic regression model.

If in any case, there are only two possible labels, logistic regression will give probability for both the labels. One with highest probability will be considered as prediction.

Lecture 31: Working with diabetes dataset

Dataset we will be working with:

Now, we have selected another famous machine learning dataset, and that is diabetes dataset. This is how that dataset looks like:

	A	B	C	D	E	F	G	H	I	J
1	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	
2	6	148	72	35	0	33.6	0.627	50	1	
3	1	85	66	29	0	26.6	0.351	31	0	
4	8	183	64	0	0	23.3	0.672	32	1	
5	1	89	66	23	94	28.1	0.167	21	0	
6	0	137	40	35	168	43.1	2.288	33	1	
7	5	116	74	0	0	25.6	0.201	30	0	
8	3	78	50	32	88	31	0.248	26	1	
9	10	115	0	0	0	35.3	0.134	29	0	
10	2	197	70	45	543	30.5	0.158	53	1	
11	8	125	96	0	0	0	0.232	54	1	
12	4	110	92	0	0	37.6	0.191	30	0	
13	10	168	74	0	0	38	0.537	34	1	
14	10	139	80	0	0	27.1	1.441	57	0	
15	1	189	60	23	846	30.1	0.398	59	1	
16	5	166	72	19	175	25.8	0.587	51	1	
17	7	100	0	0	0	30	0.484	32	1	
18	0	118	84	47	230	45.8	0.551	31	1	
19	7	107	74	0	0	29.6	0.254	31	1	
20	1	103	30	38	83	43.3	0.183	33	0	

Each row of this dataset gives data about one patient. There are 8 features in this dataset which are:

1. Number of pregnancies.
2. Glucose level.
3. Blood pressure.
4. Skin thickness.
5. Insulin level
6. BMI (body mass index)
7. DiabetesPedgreeFunction (probability of diabetes in family).
8. Age.

The last column is the label, and is in binary format. Here 1 means that patient is suffering from diabetes and 0 means that patient is not suffering from diabetes.

Analyzing a costumer (12th row):

- She had 4 pregnancies
- Her glucose level is 110
- Blood pressure is 92
- Skin thickness and insulin level is 0
- BMI is 37.6
- Probability of diabetes in family is 0.191 (very low)
- And her age is 30

Based on this data, her label is 0, which means she isn't suffering from diabetes.

Now it’s your time. Try to analyze row 20 and write down your observation on a paper.

You can download whole dataset here

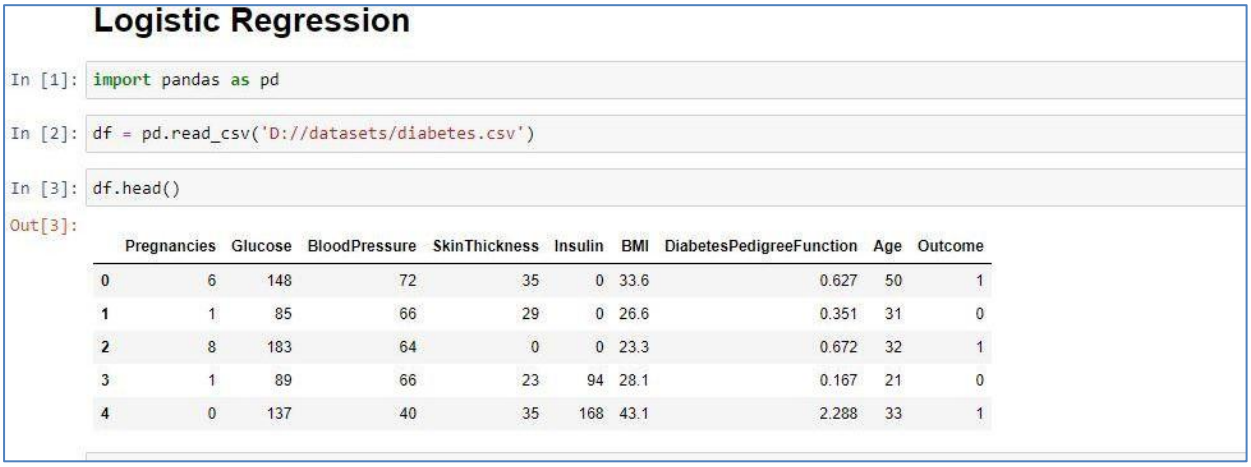
https://www.kaggle.com/datasets/mathchi/diabetes-data-set

Lecture 32: Making logistic regression model

We have to load our data and separate features and labels from dataset.

This is how we load our dataset:

```
import pandas as pd
df = pd.read_csv('D://datasets/diabetes.csv')
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Note that this dataset is not in excel format. It is in csv format. That's why we have used read_csv() instead of read_excel(), and .csv format in upper code.

.head() method of data frame is showing us first 5 rows of dataset.

To check the shape of data, we can use .shape method of data frame this way:

```
df.shape
```

Output will look like this:

```
In [7]: df.shape
Out[7]: (768, 9)
```

First element is number of rows in our dataset and second element is number of columns. Our data set contain information of approximately 770 patients, and 9 columns, which is much larger number than before.

As the dataset is loaded, it is time to specify features and labels from this dataset. It is clear what we want as prediction from this data, and that is the last column which is telling us whether the patient has diabetes or not.

Hence our label is last column *'Outcome'*.

And all other columns are features.

1. LABEL: *'Outcome'* column
2. FEATURES: All other columns are features.

To specify 8 columns as features, let's use another method (writing names of 8 columns is not feasible)

The new technique is:

```
features = df.drop('Outcome', axis=1)
labels = df[['Outcome']]
```

Instead of choosing 8 columns from 9 columns, I have deleted one column (label) from real data so that we are left with other 8 columns, and those are features. *Axis=1* is to make sure that we want to delete a column. (0 refers to rows and 1 refer to columns)

Labels are defined using the same method as before.

Features look like this:

```
In [14]: features.head()
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

And labels look like this:

```
In [15]: labels.head()
```

Out[15]:

	Outcome
0	1
1	0
2	1
3	0
4	1

Lecture 33: Making logistic regression part 2

To import logistic regression, we use this line of code:

```
from sklearn.linear_model import LogisticRegression
```

The way we train logistic regression model is same as other models, which we have discussed in previous sections.

Step - 1: Initialization

```
model = LogisticRegression()
```

Step - 2: Training/fitting

```
model.fit(features, labels)
```

Features and labels are the variables that we made in previous lecture. Features contain first 8 columns and labels contain last column of data that is to be predicted by model later.

We are all done! Let's make predictions now.

As discussed before, logistic regression returns the probability score.

To get that, we can use *predict_proba* method like this:

```
model.predict_proba([[6, 148, 72, 35, 0, 33.6, .627, 31]])
```

I have given a new data point containing information of a patient, and information is:

- 6 pregnancies
- 148 glucose level
- 72 blood pressure
- 35 skin thickness
- 0 insulin
- 33.6 body mass index
- 0.62 probability of diabetes in family
- 31 age

You can change the data values for sure.

Note that there are 8 pieces of information, in same arrangement, as features in dataset.

Output is:

```
In [40]: model.predict_proba([[6, 148, 72, 35, 0, 33.6, .627, 31]])
         C:\Users\dell\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not
         gression was fitted with feature names
           warnings.warn(
Out[40]: array([[0.32971802, 0.67028198]])
```

How many possible labels do we having? Just two, 0 and 1. As there are only two outputs available, hence we have 2 results in this prediction.

Each element in upper array gives probability of possible label.

- probability of 0: *0.32971802*
- probability of 1: *0.67028198*

As we see that probability of 1 is greater than 0, **hence our model thinks that this patient is suffering from diabetes (1 represent diabetes).**

Instead of these two results, if you only want to see the predicted label, you can use .predict method like this:

```
In [41]: model.predict([[6, 148, 72, 35, 0, 33.6, .627, 31]])

         C:\Users\dell\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not
         gression was fitted with feature names
           warnings.warn(

Out[41]: array([1], dtype=int64)
```

As discussed, our model thinks that this data point is suffering from diabetes. Similarly, you can make more predictions and see the probabilities

Using *.predict_proba* method we can also analyze confidence of our model on its prediction (higher the probability score is, more the model is confident).

Take a break reread lecture 30 at this point. Then again read this lecture (lecture 33).

Lecture 34: Accuracy of our model

Till now, we didn't split our data into training and test chunks, that's why we cannot check accuracy of model. To do so, we are going to train another model.

These lines of code load the dataset and specify features and labels.

```
import pandas as pd
df = pd.read_csv('D://datasets/diabetes.csv')
features = df.drop('Outcome', axis=1)
labels = df[['Outcome']]
```

These are the imports that we need:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

This line of code split the data into two chunks.

```
X_train, X_test, y_train, y_test = train_test_split(features, labels)
```

We will train our model on X_train and y_train. After training we will test our model on X_test and y_test. Review *section 5* for clarity (*Lecture 28, and Lecture 29*).

Initializing and training our model:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Our model is ready. Now, let's check how well it perform on test data. In other words, let's check accuracy of model. We don't have to worry about mathematical calculations; we can just use *.score* method like this:

```
model.score(X_test, y_test)
```

This line will give us the accuracy of our model.

Output of upper code:

```
In [11]: model.score(X_test, y_test)
Out[11]: 0.7708333333333334
```

Accuracy of our model is approximately 77% which is quite good. We can improve the accuracy by using some other machine learning model. Also, we can improve the accuracy by hyper parameter tuning.

But wait, what is hyper parameter tuning? There's a lot more for you to explore further.

Assignment 3: Create your own ML model

Task: train a k neighbors classifier model on diabetes dataset and check how well your model perform on test data.

SECTION 7: ALL IN ONE

Lecture 35: All codes here

Section 2

```
import pandas as pd
data = pd.read_excel('D://Professor ahmad class.xlsx')

#separating features and labels

features = data[['Study hours', 'Marks in previous grade', 'class performance
out of 10']]
label = data[['Marks scored']]
final_label = label * 100

#creating model

from sklearn.neighbors import KNeighborsClassifier
mdl = KNeighborsClassifier()
mdl.fit(features,final_label)

#making predictions

mdl.predict([[2, 0.60, 7]])


#making user friendly
```

```
study_hours = int(input('How many hours your student study per day: '))
marks_previous = int(input('How much he scored in his previous grade give you
answer in percentage: '))
mp = marks_previous / 100

class_performance = int(input('What is class performance of student out of
10: '))

prediction = mdl.predict([[study_hours, mp, class_performance]])

print('Predicted marks: ')
print(prediction)
```

Predictions at once:

```
data_of_students = [[2, 0.71, 6], [4, 0.67, 8], [8, 0.91, 8.6]]
mdl.predict(data_of_students)
```

Section 3

```
import pandas as pd
from sklearn.linear_model import LinearRegression

#loading the dataset

data = pd.read_excel('D://Professor ahmad class.xlsx')

#specifying features and labels

features = data[['Study hours', 'Marks in previous grade', 'class performance
out of 10']]

label = data[['Marks scored']]
final_label = label * 100

#creating model

model = LinearRegression()
model.fit(features, final_label)
```

Section 4

```
import pandas as pd
df = pd.read_csv('D://datasets/iris.csv')

#separataing features and labels

features = df[['petal length', 'petal width', 'sepal length', 'sepal width']]
label = df['flower']

#creating model

from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(features, label)

#making prediction

model.predict([[2, 3, 1, 2]])
```

To change number of neighbors:

```
model = KNeighborsClassifier(n_neighbors = 3)
```

Section 5

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
df = pd.read_csv('D://datasets/iris.csv')

#separating features and labels

features = df[['petal length', 'petal width', 'sepal length', 'sepal width']]
label = df['flower']
```

```
#splitting data and creating model

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, label)
model = KNeighborsClassifier()
model.fit(X_train, y_train)

#testing model

model.score(X_test, y_test)
```

Section 6

```
import pandas as pd
df = pd.read_csv('D://datasets/diabetes.csv')

#separating features and labels

features = df.drop('Outcome', axis=1)
labels = df[['Outcome']]

#making our model

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(features, labels)

#predictions of our model

model.predict_proba([[2, 148, 52, 35, 0, 33.6, .627, 31]])
model.predict([[2, 148, 52, 35, 0, 33.6, .627, 31]])
```

Splitting data to calculate accuracy

```
import pandas as pd
df = pd.read_csv('D://datasets/diabetes.csv')

#separating features and labels

features = df.drop('Outcome', axis=1)
labels = df[['Outcome']]
```

LECTURE 35: ALL CODES HERE

```
#splitting and training our model

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, labels)
model = LogisticRegression()
model.fit(X_train, y_train)

#checking accuracy

model.score(X_test, y_test)
```

CONGRATULATIONS!

What next

I highly you recommend diving into neural networks at this stage. Why?
Previously, we achieved 77% accuracy on diabetes dataset.
With neural networks, we've reached 92%, and there's still room to improve.