

# Fundamentals of Data science and Machine Learning

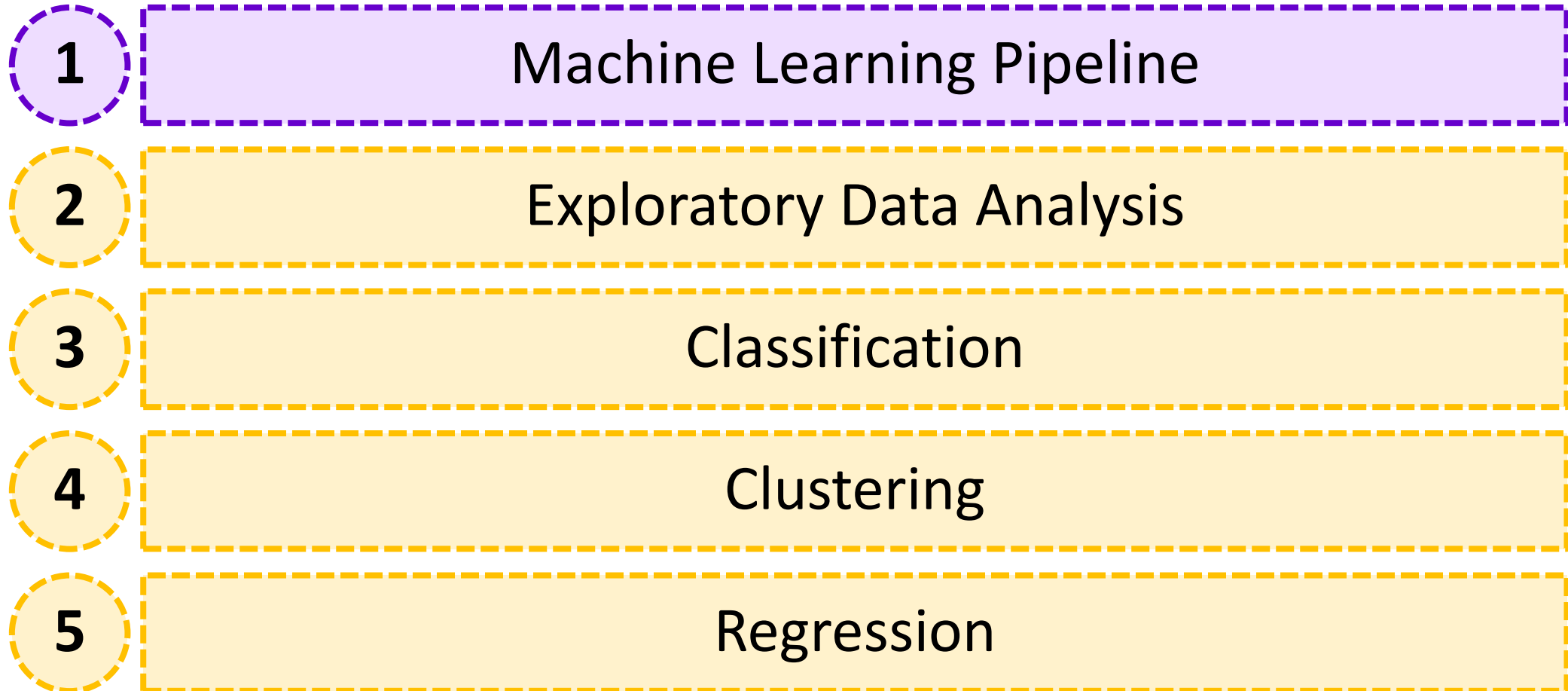
Concepts, Techniques and Tools to Build Intelligent Systems

## Module 05

### Dive Into Machine Learning

**Ali Samanipour**

May. 2023



# Introduction To Machine Learning

**“it gives computers the **ability to learn without being explicitly programmed.**”**

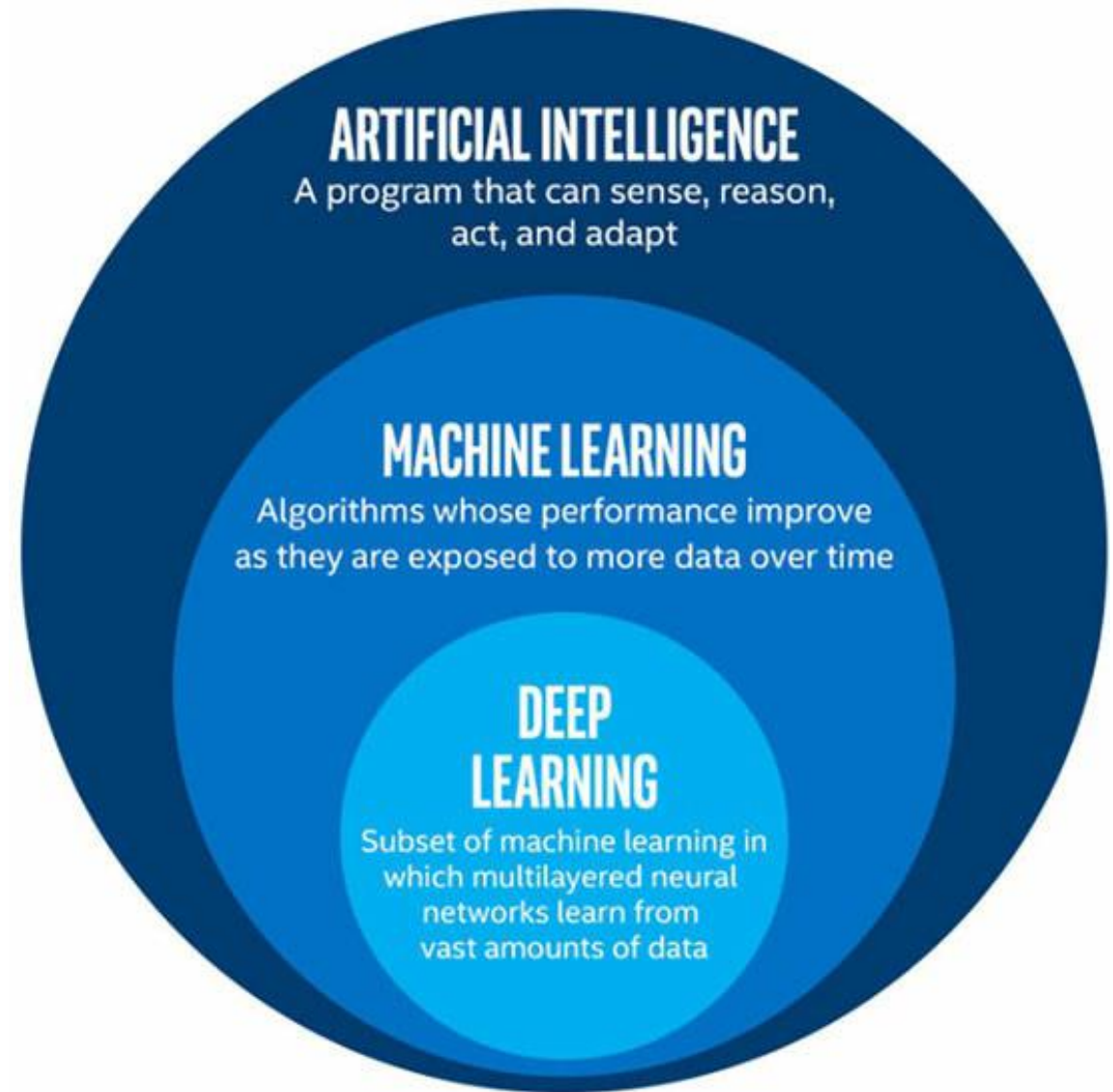
*Arthur Samuel (1959)*

**“A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its **performance on  $T$ , as measured by  $P$ , improves with experience  $E$ .**”**

*Tom Mitchell(1997)*

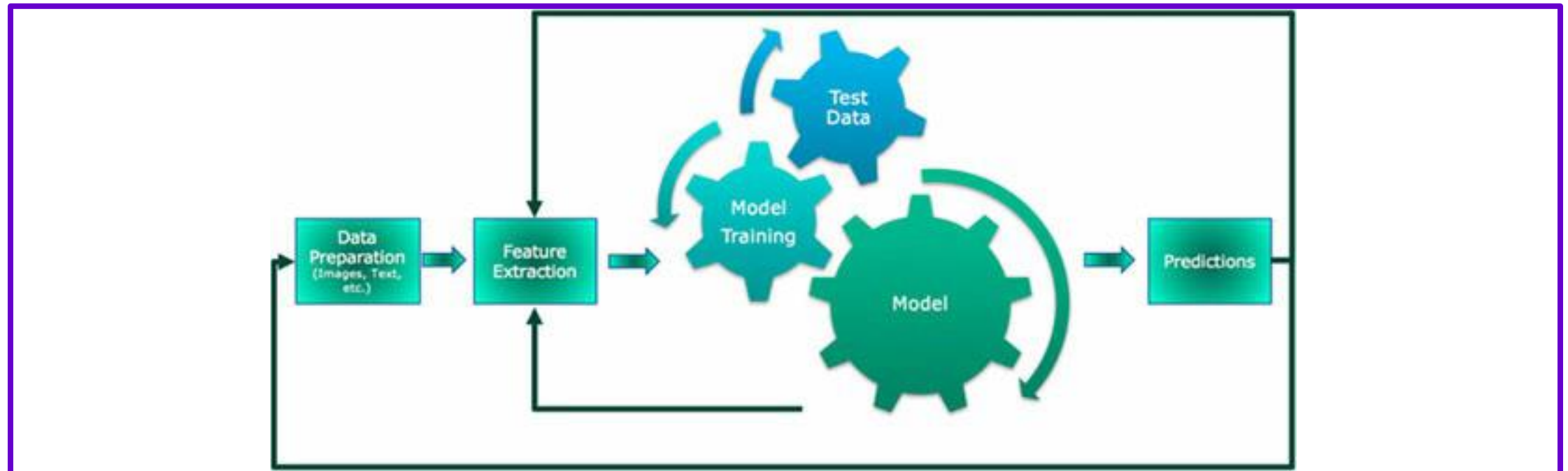
# What is Machine Learning

Machine Learning is the field of study where **algorithms and statistical models automatically learn** and improve from the data with underlying patterns and inference, **without explicit instruction programmed.**



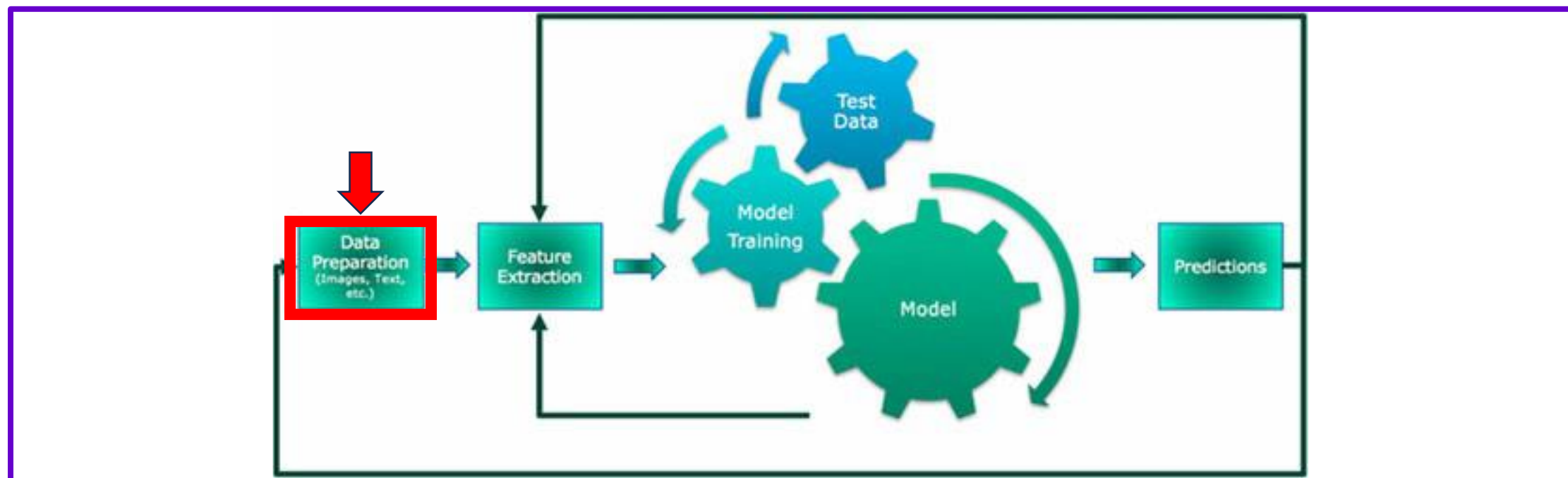
# Machine Learning Pipeline

**Machine Learning pipeline** is an iterative model where we have to go back-and-forth to come up with an optimal model for any kind of Machine Learning model.



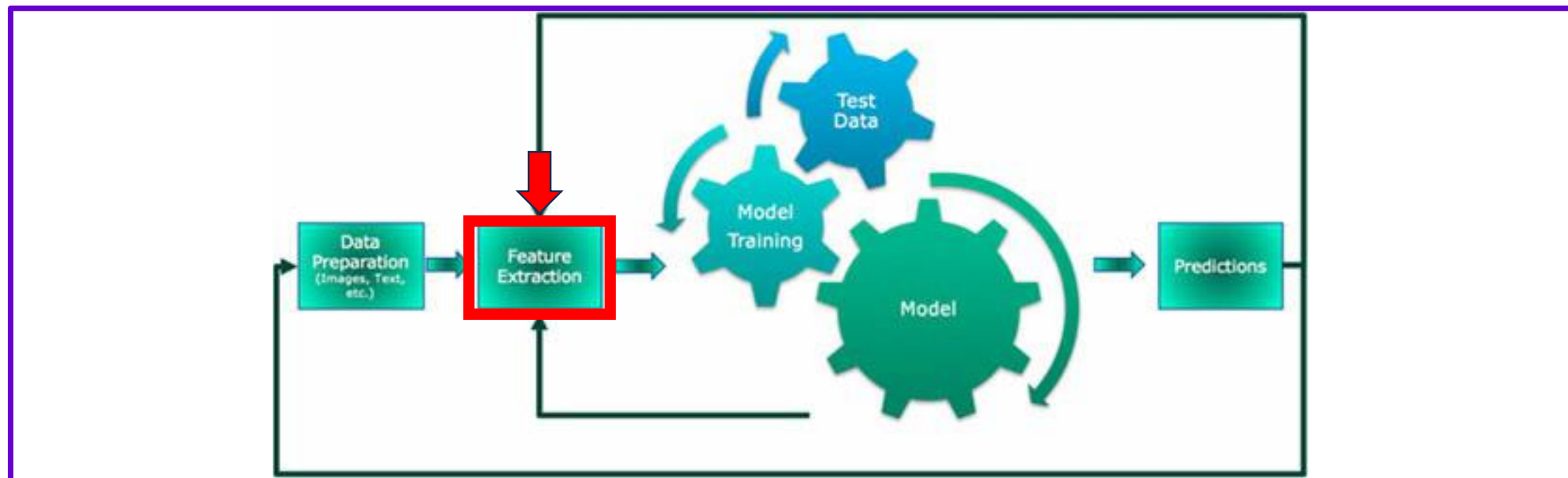
# Machine Learning Pipeline: Data Preparation

Data Preparation is the process by which we can **make the data ready for training**, testing, and validating the ML model.



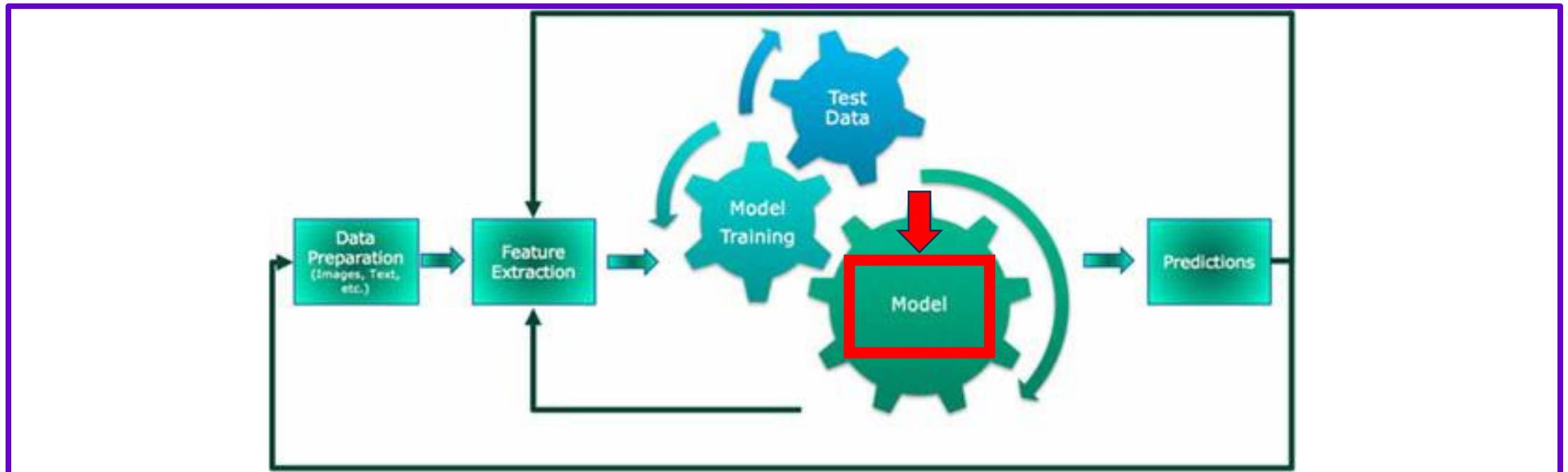
# Machine Learning Pipeline: Feature Extraction

Feature extraction is the process by which we **select, modify, or mutate the right features** for training the model



# Machine Learning Pipeline: Model

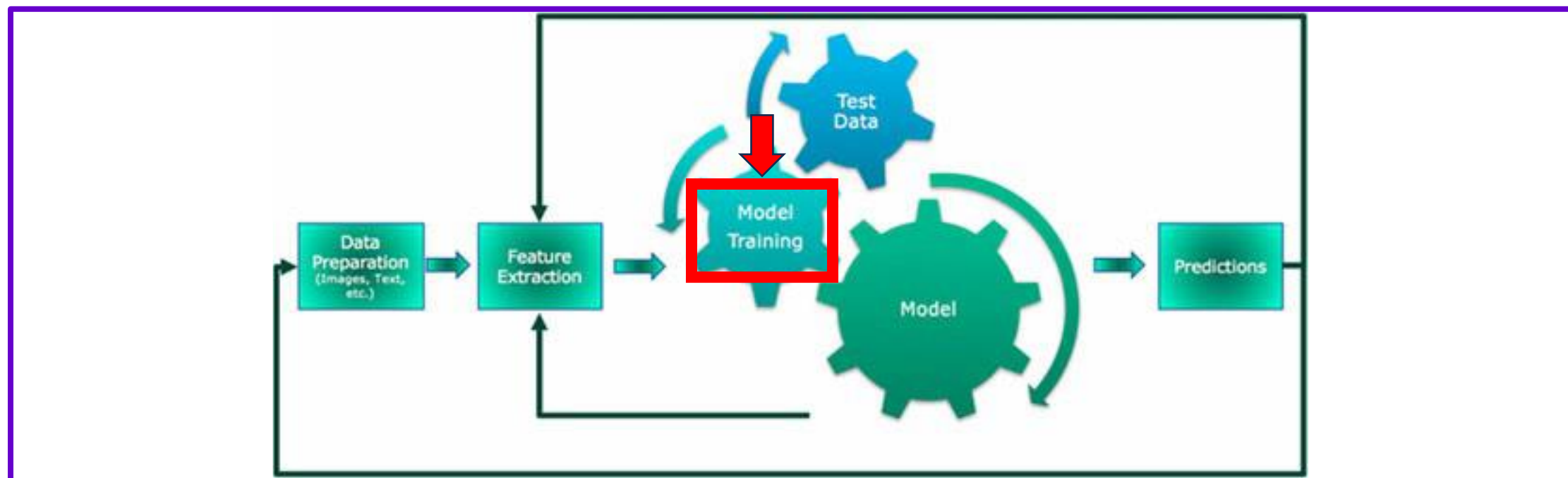
Machine Learning model is a **mathematical representation learned/formed from the pattern of the data**, which is given as an input for training the model





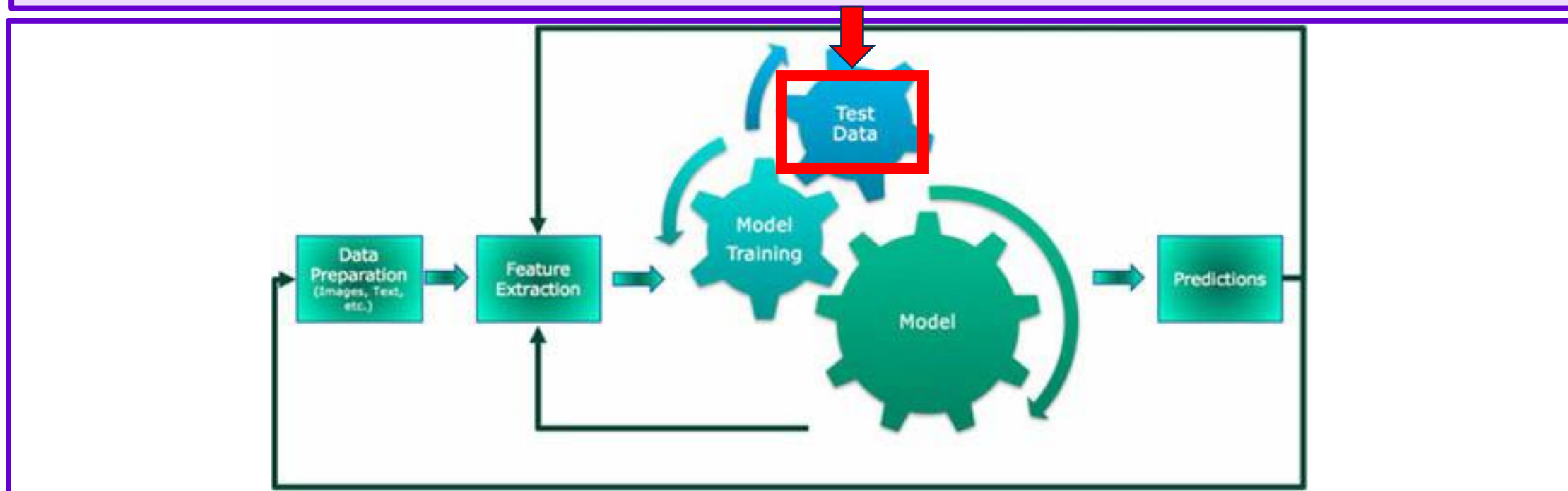
# Machine Learning Pipeline: Model Training

**Model Training is a process to learn the underlying pattern from the data** with the given output feature (for supervised learning)



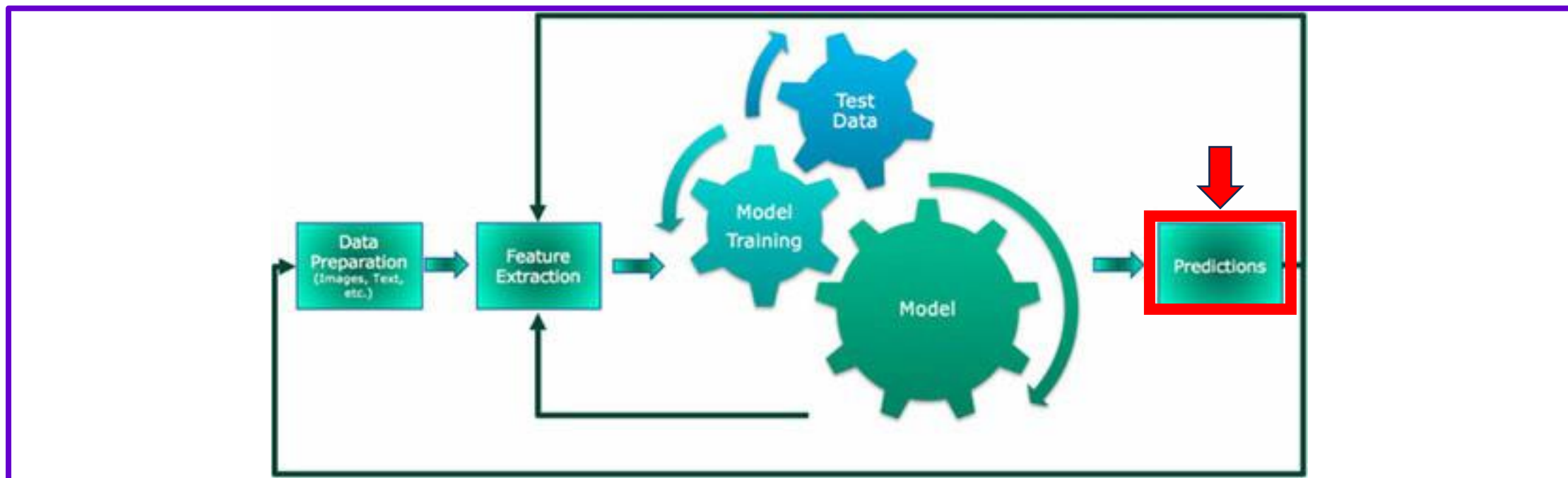
# Machine Learning Pipeline: Test Data

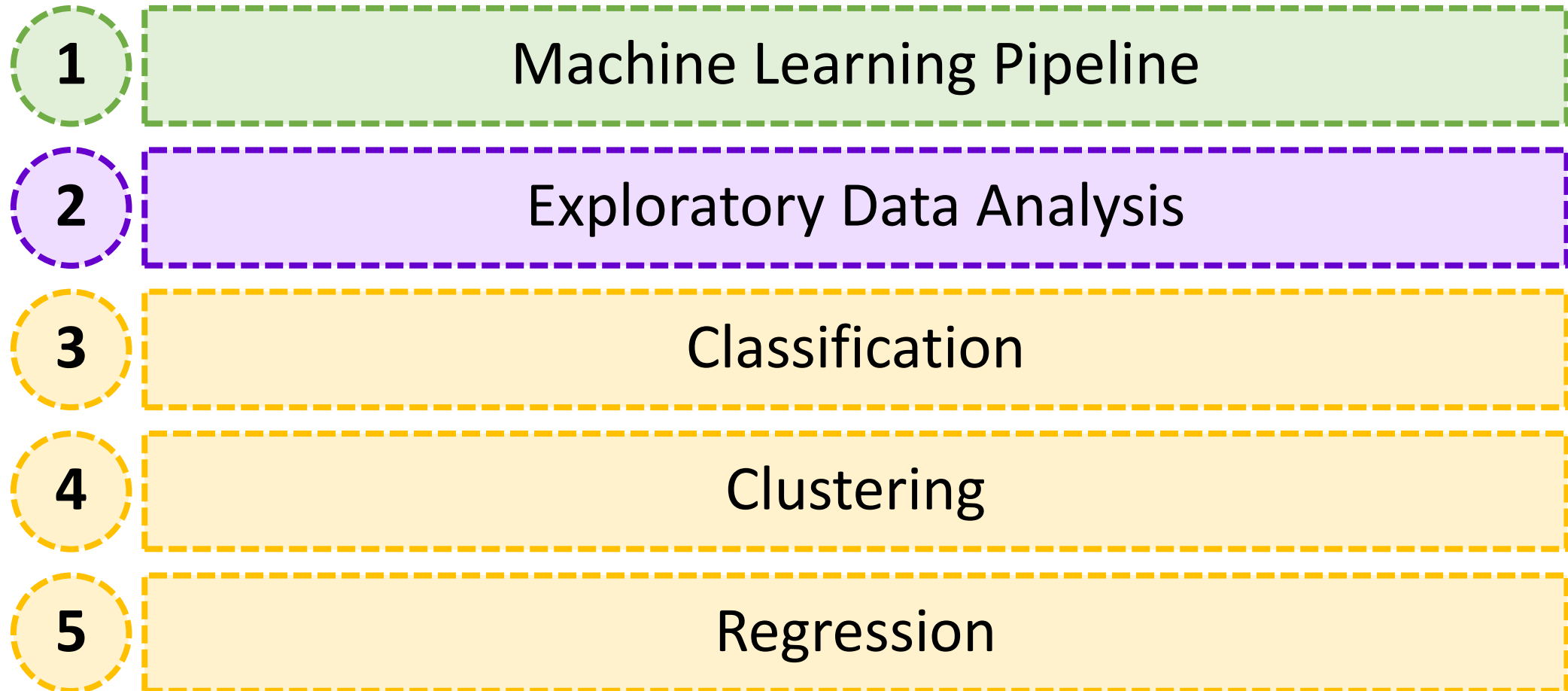
This is a subset of the entire dataset. This data is not given as an input to the model training phase. **This data will be used to find the accuracy of the model** for the unseen data.



# Machine Learning Pipeline: Prediction/Evaluation

This is the step where we can see the quality of the model using different metrics such as Accuracy, Recall, Precision, etc.





# Lets Start Our Project (The Dataset)

Iris dataset contains data for 3 species namely Iris Setosa (a.k.a. setosa), Iris versicolor (a.k.a. versicolor), Iris virginica (a.k.a. virginica) and each species contain 50 records.



Iris Versicolor



Iris Setosa



Iris Virginica

**Samples**  
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

**Features**  
(attributes, measurements, dimensions)

**Class labels**  
(targets)

**Petal**

**Sepal**

A detailed diagram of an Iris flower with yellow arrows indicating measurements. One arrow points to the length of a petal, and another points to the width of a sepal.

# Exploratory Data Analysis

## (Load Dataset using Scikit-learn)

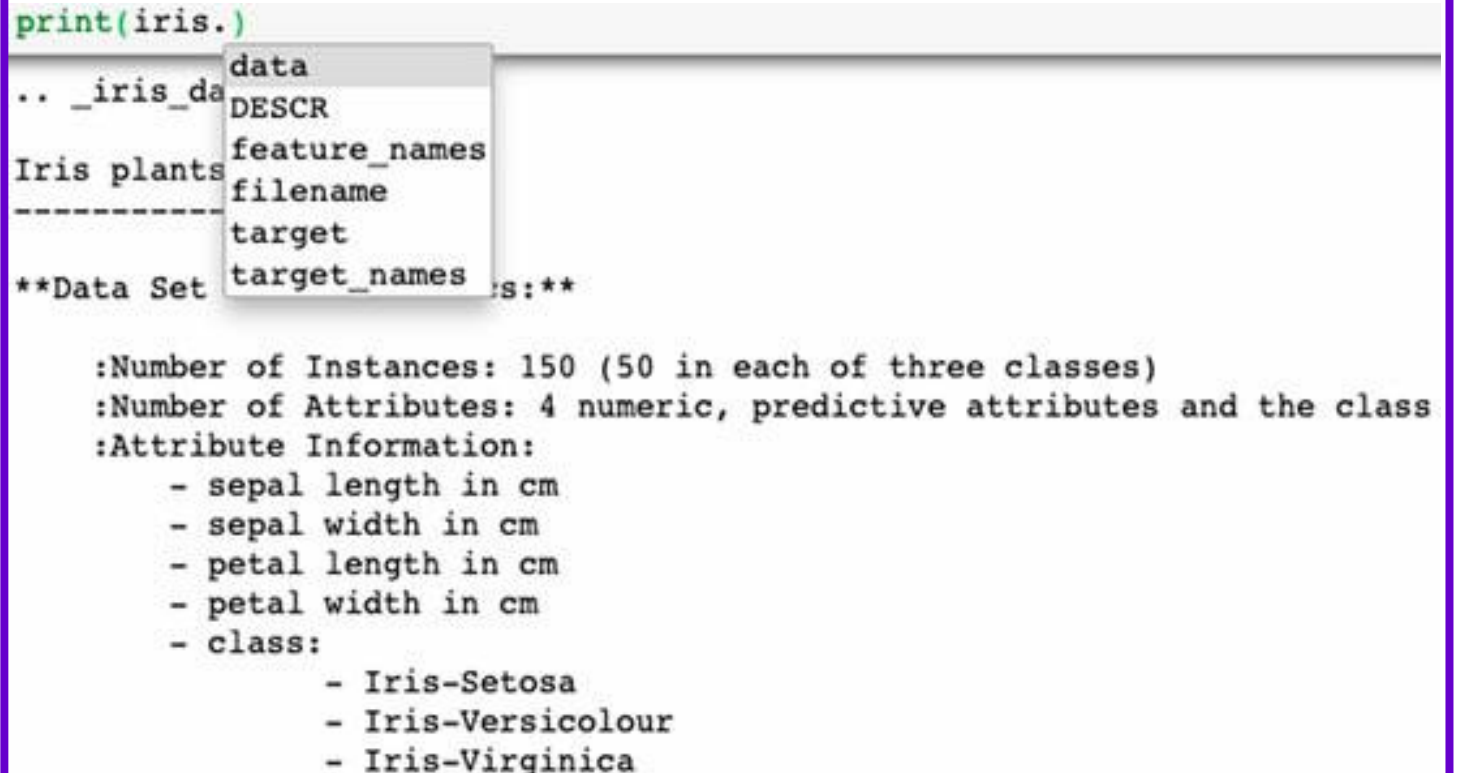
For the first scenario, we will load the dataset from sklearn and see how to make the dataset useful for later phases. This step comes under the data obtaining for O.S.E.M.N.

```
import pandas as pd
from sklearn import datasets
iris = datasets.load_iris()
```

## Content of Iris variable

From all the different information, we will only use the “data,” “feature\_names,” “target,” “target\_names.”

```
print(iris.)
```



```
.. _iris_data
Iris plants
-----
**Data Set Description**

: Number of Instances: 150 (50 in each of three classes)
: Number of Attributes: 4 numeric, predictive attributes and the class
: Attribute Information:
  - sepal length in cm
  - sepal width in cm
  - petal length in cm
  - petal width in cm
  - class:
    - Iris-Setosa
    - Iris-Versicolour
    - Iris-Virginica
```



## Basic Data table of Iris Dataset

We will see how we can combine all the data and form a table with input and output features.

```
iris_data = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_data["target"] = iris.target
iris_data.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0



# Renaming Columns

The name of the columns is not as per standards, and it is very important to maintain the naming standard for ease of coding and readability

```
iris_data.rename(columns={'sepal length (cm)':'sepal_length',  
                          'sepal width (cm)':'sepal_width',  
                          'petal length (cm)':'petal_length',  
                          'petal width (cm)':'petal_width'},  
                 inplace=True)
```

# Adding Species name

We have another problem, i.e., for the target column, we are using integers to represent different categories. Having the name of the species will help in the data visualization.

```
iris_data["target_names"] = iris.target  
iris_data["target_names"].replace({0: 'setosa',  
                                   1: 'versicolor',  
                                   2: 'virginica'}, inplace=True)
```

## Basic Data table of Iris Dataset after some Scrubbing

As we have seen, we have mostly covered the basic data preparation for the next step, which is data visualization.

```
iris_data.head()
```

	sepal_length	sepal_width	petal_length	petal_width	target	target_names
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

## Exploratory Data Analysis (Load Dataset using Seaborn)

Seaborn is a data visualization library that we had seen earlier, and it has a limited set of datasets

```
import seaborn as sns
iris = sns.load_dataset('iris')
iris.head()
```

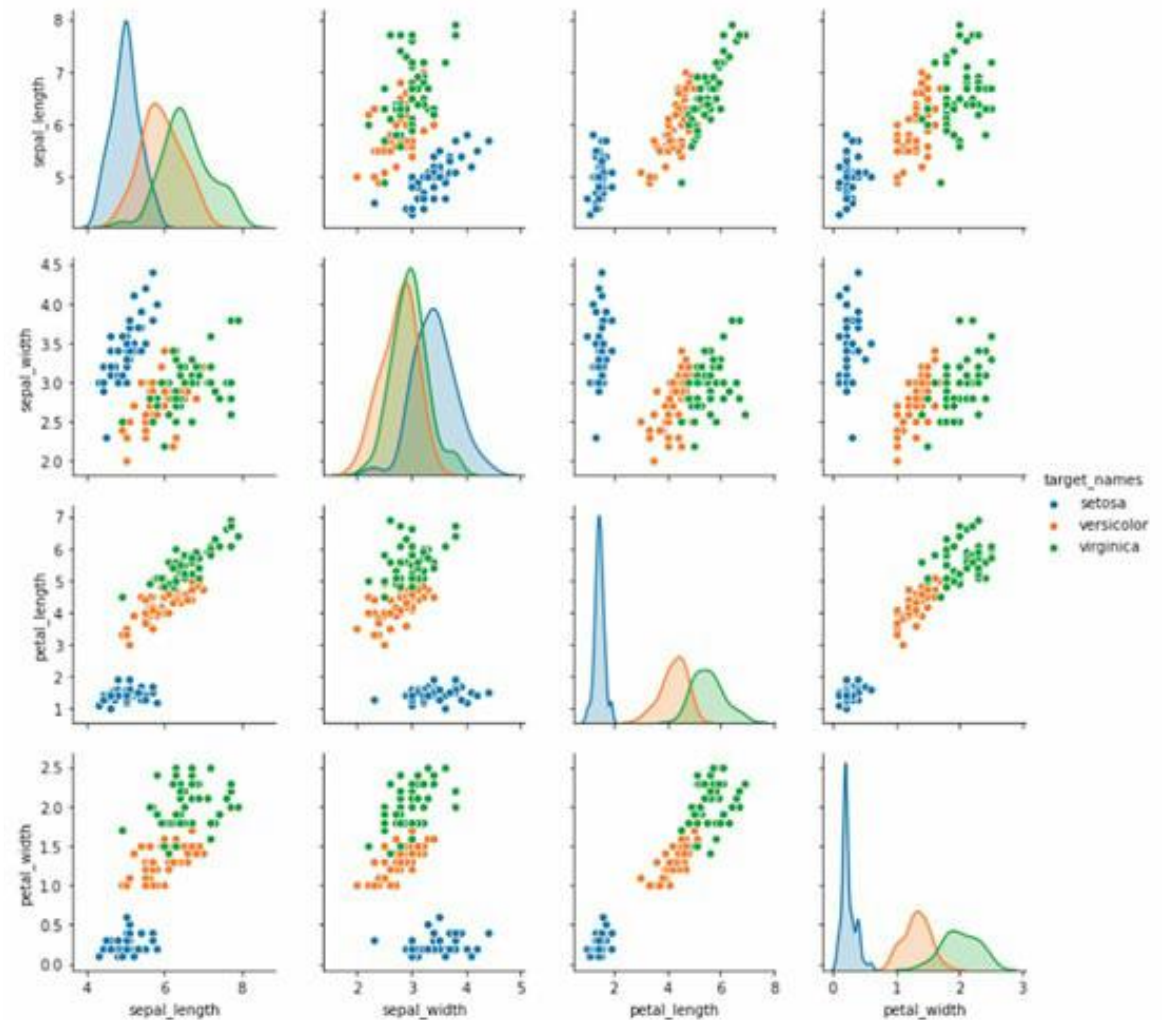
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

## Data Analysis (Pair Plot)

Pair plot enables us to **visualize the distribution** of the single variable **and relation with all the variables**.

```
sns.pairplot(iris_data[['sepal_length', 'sepal_width', 'target_names',  
                        'petal_length', 'petal_width']], hue='target_names')
```

<seaborn.axisgrid.PairGrid at 0x1a23907a58>

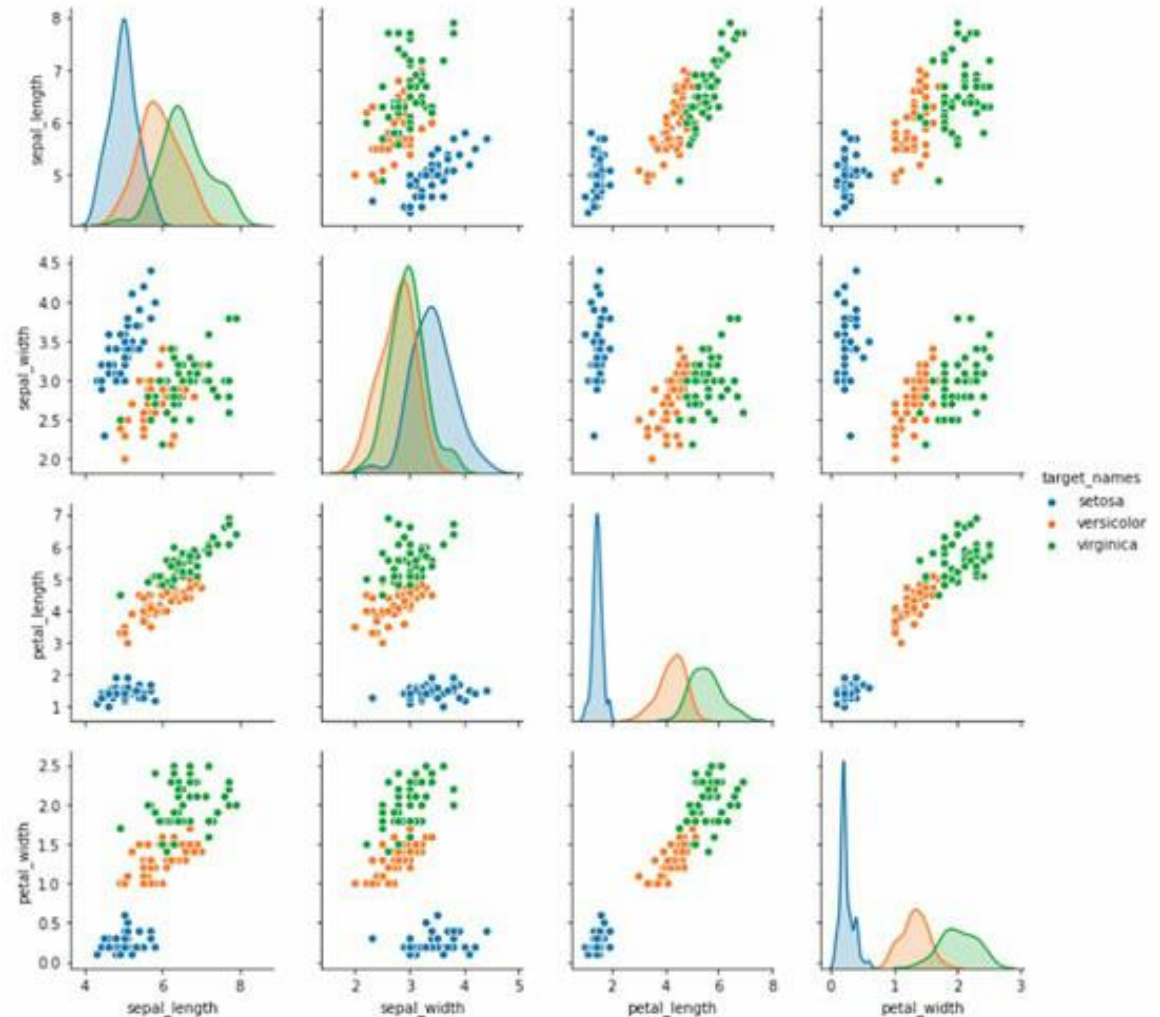


## Pair Plot: Observations

the feature  
“sepal\_width” the  
**over**lap of the  
distribution of 3  
different types of the  
iris is very high. This is  
not a sign of good  
feature which will  
**class**ify different  
species.

```
sns.pairplot(iris_data[['sepal_length', 'sepal_width', 'target_names',  
                        'petal_length', 'petal_width']], hue='target_names')
```

<seaborn.axisgrid.PairGrid at 0x1a23907a58>



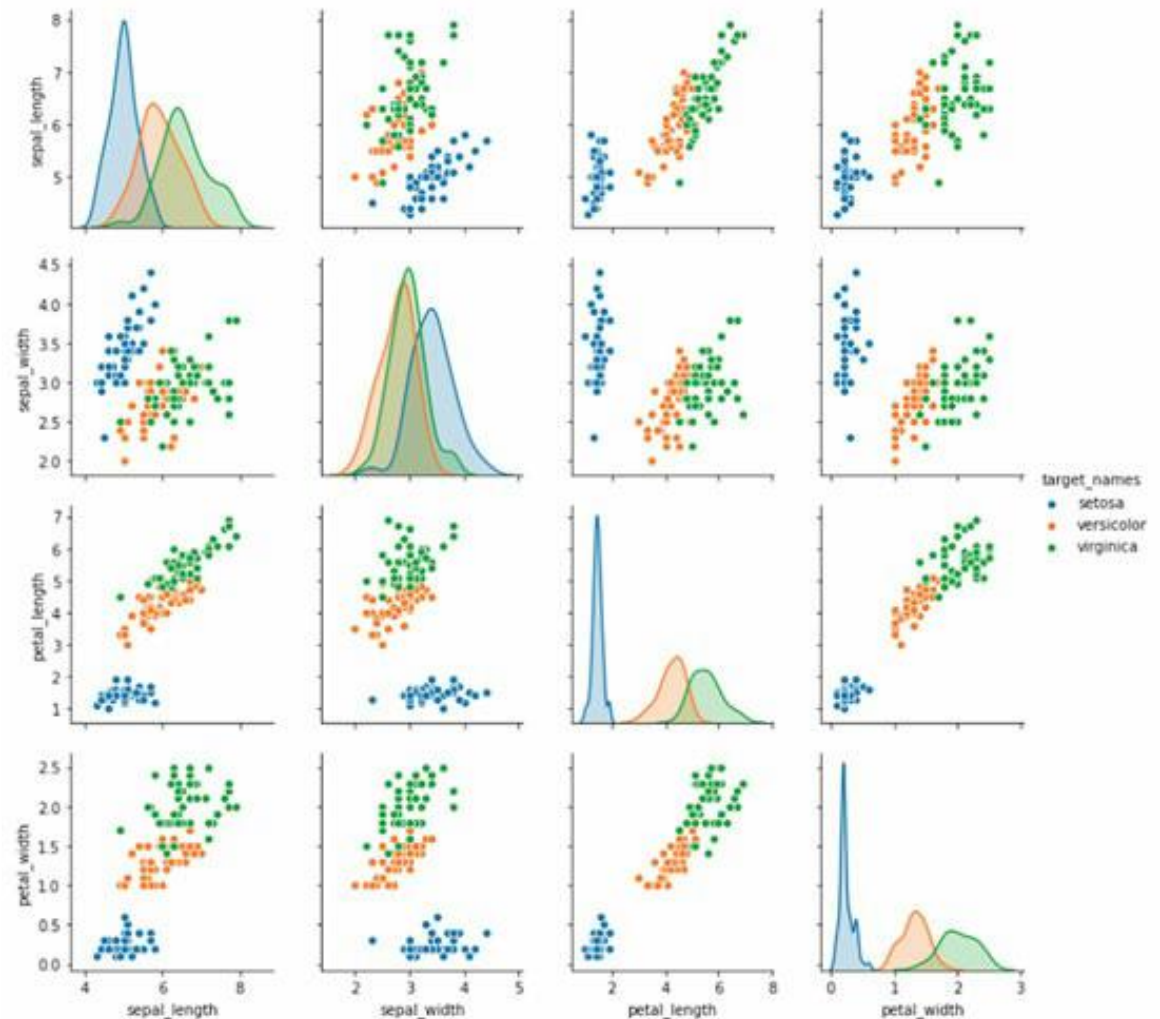


## Pair Plot: Observations

Talking about other features, we can see that “**versicolor**” and “**virginica**” are very close to each other, and **with the feature “sepal\_width” the overlap is high as expected**

```
sns.pairplot(iris_data[['sepal_length', 'sepal_width', 'target_names',  
                        'petal_length', 'petal_width']], hue='target_names')
```

<seaborn.axisgrid.PairGrid at 0x1a23907a58>



# Our Hypothesis!

**Our Hypothesis:** “sepal\_length” vs “sepal\_width” is the worst feature descriptor for identifying the species “versicolor” and “virginica.”

**How we can know if two feature is correlate?**

**We can validate the correlation of two feature by finding the correlation coefficient**



# Correlation

Correlation is a **technique by which we can find out how strong the features are related.**

(methods to find the correlation : pearson, kendall, spearman)

```
iris_data.corr(method='pearson')
```

	sepal_length	sepal_width	petal_length	petal_width	target
sepal_length	1.000000	-0.117570	0.871754	0.817941	0.782561
sepal_width	-0.117570	1.000000	-0.428440	-0.366126	-0.426658
petal_length	0.871754	-0.428440	1.000000	0.962865	0.949035
petal_width	0.817941	-0.366126	0.962865	1.000000	0.956547
target	0.782561	-0.426658	0.949035	0.956547	1.000000

# Correlation Calculation

The correlation coefficient values range from -1 to 1.

Roughly, we can say when the correlation is 1 then two features/variables are linearly dependable, i.e. if at least one of the vectors in the set can be defined as a linear combination of the others.

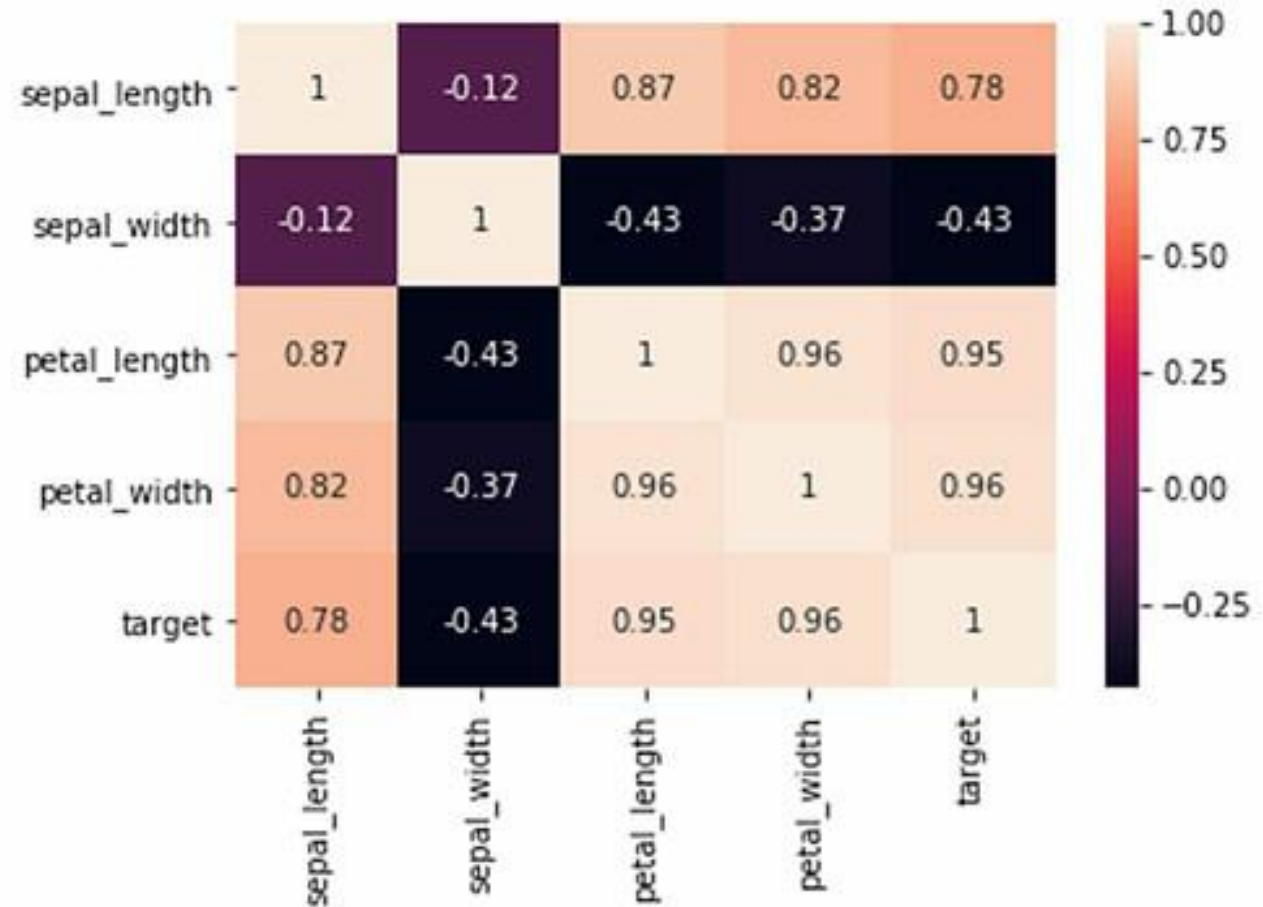
$$r_{x,y} = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \cdot \sum (y - \bar{y})^2}}$$

## Heatmap of Correlation

From above correlation table, we can plot a heatmap. Heatmap will visually give us the same information.

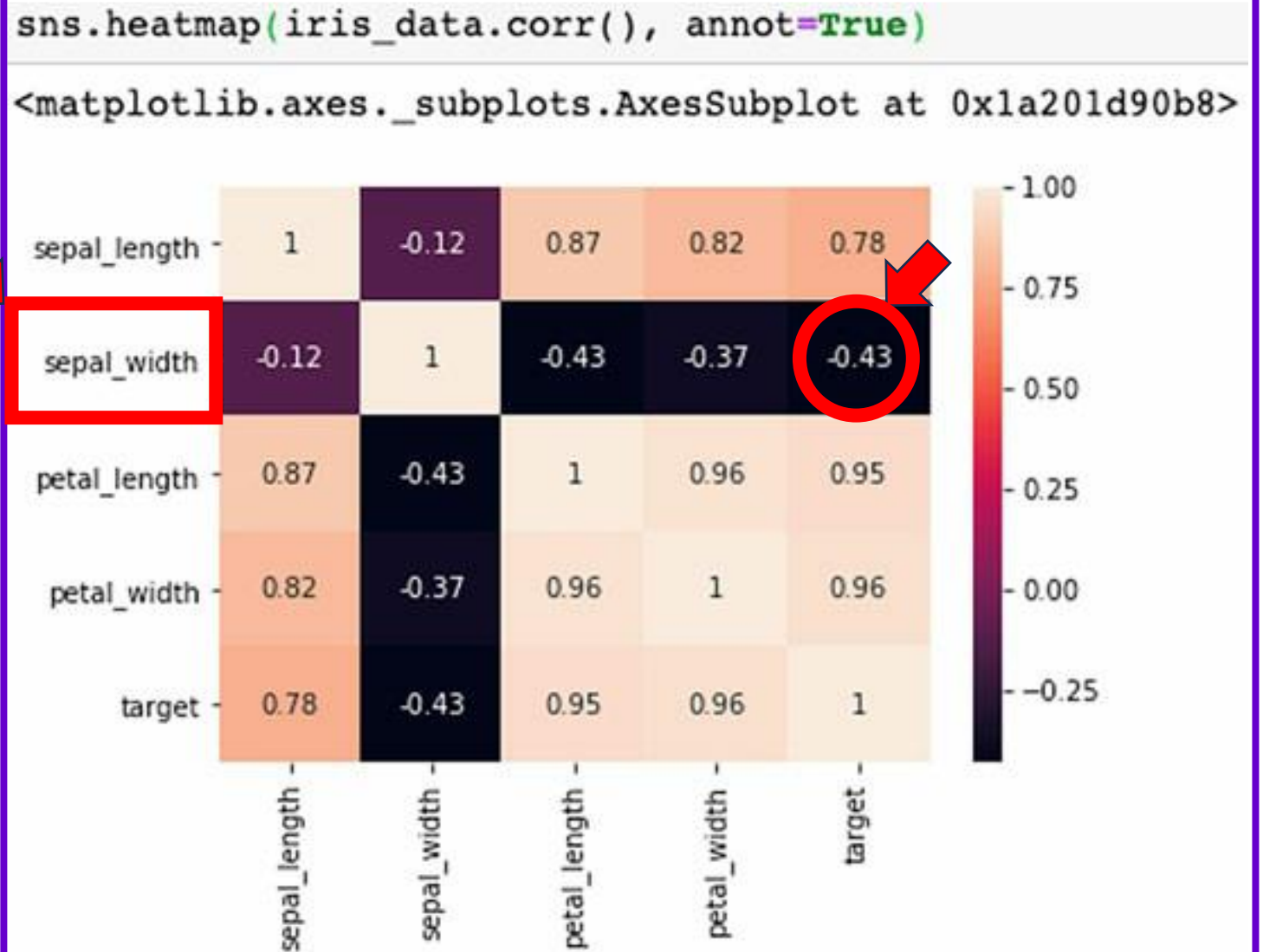
```
sns.heatmap(iris_data.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a201d90b8>
```



Analysis based on  
Heatmap of  
Correlation

we can conclude  
that removing  
“sepal\_width” can  
result in a better-  
correlated dataset  
with “target” as the  
output column.



# Linear Models

we are removing the near to zero correlated feature because we will fit the data to a linear model

```
iris_data.drop(columns=['sepal_width'], inplace=True)  
iris_data.head()
```

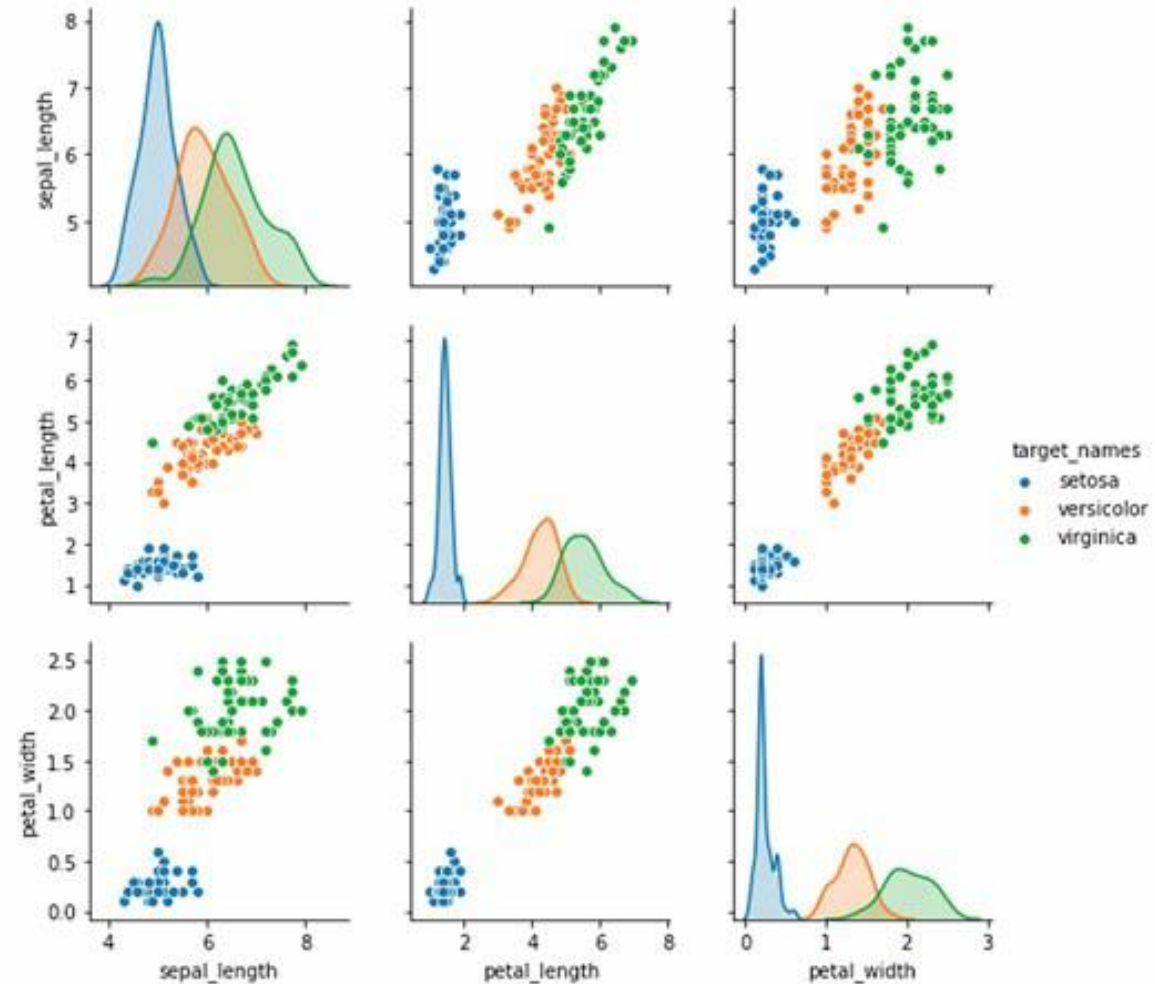
	sepal_length	petal_length	petal_width	target	target_names
0	5.1	1.4	0.2	0	setosa
1	4.9	1.4	0.2	0	setosa
2	4.7	1.3	0.2	0	setosa
3	4.6	1.5	0.2	0	setosa
4	5.0	1.4	0.2	0	setosa

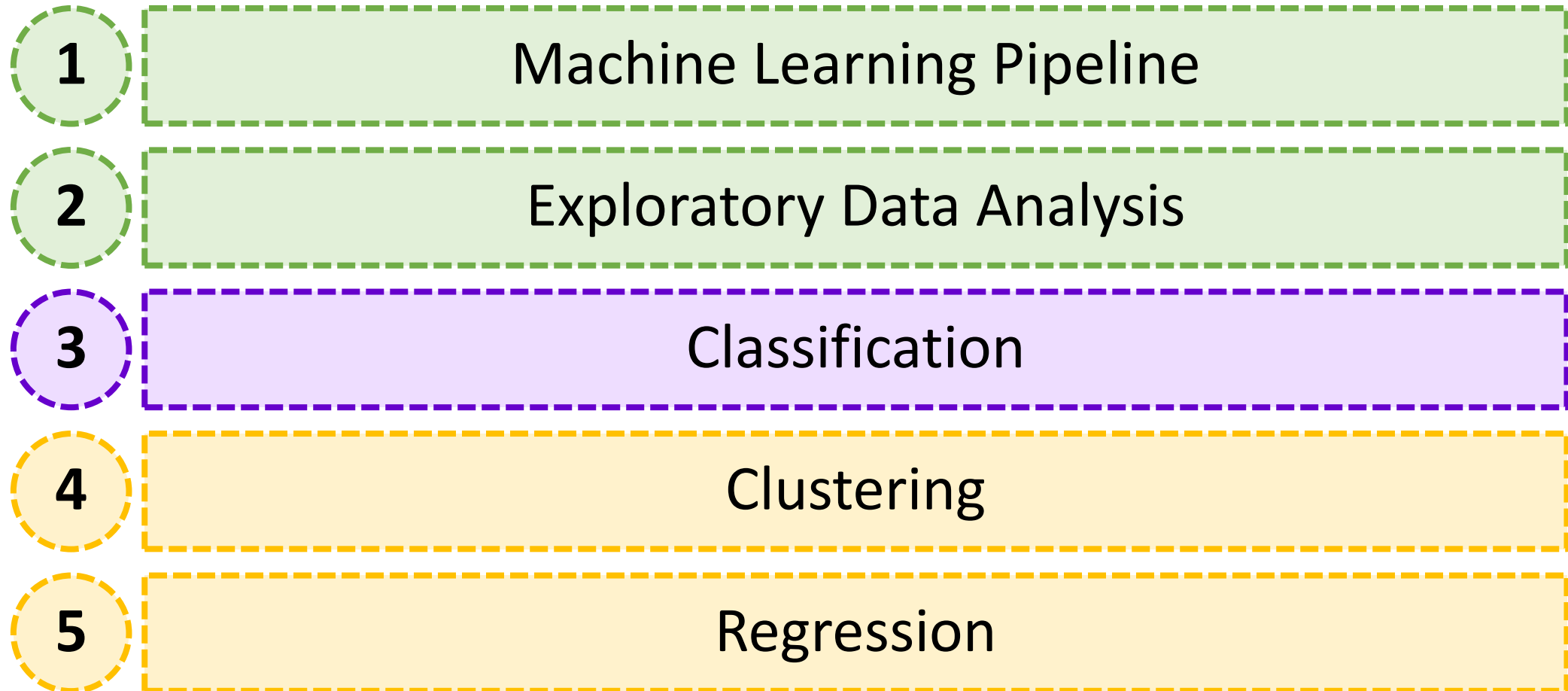
## Pair Plot with the updated data

We should see the pair plot again to see the changes, and later we will confirm our hypothesis.

```
sns.pairplot(iris_data[['sepal_length', 'target_names',\n                        'petal_length', 'petal_width']], hue="target_names")
```

<seaborn.axisgrid.PairGrid at 0x1a240133c8>





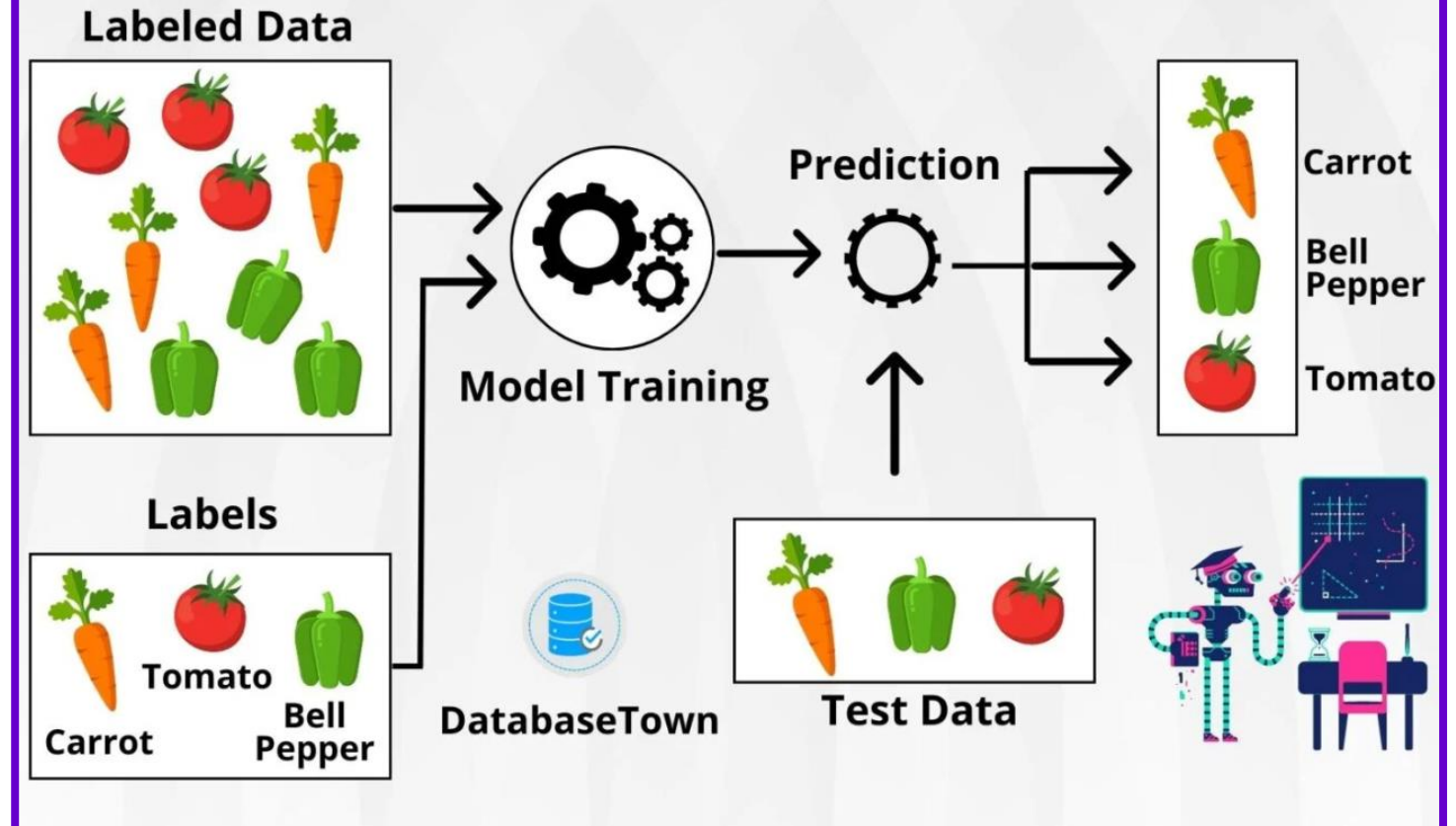


# Supervised Learning: Classification

Also known as statistical classification, which **identifies and predicts different class/groups using predictive learning.**

## SUPERVISED LEARNING

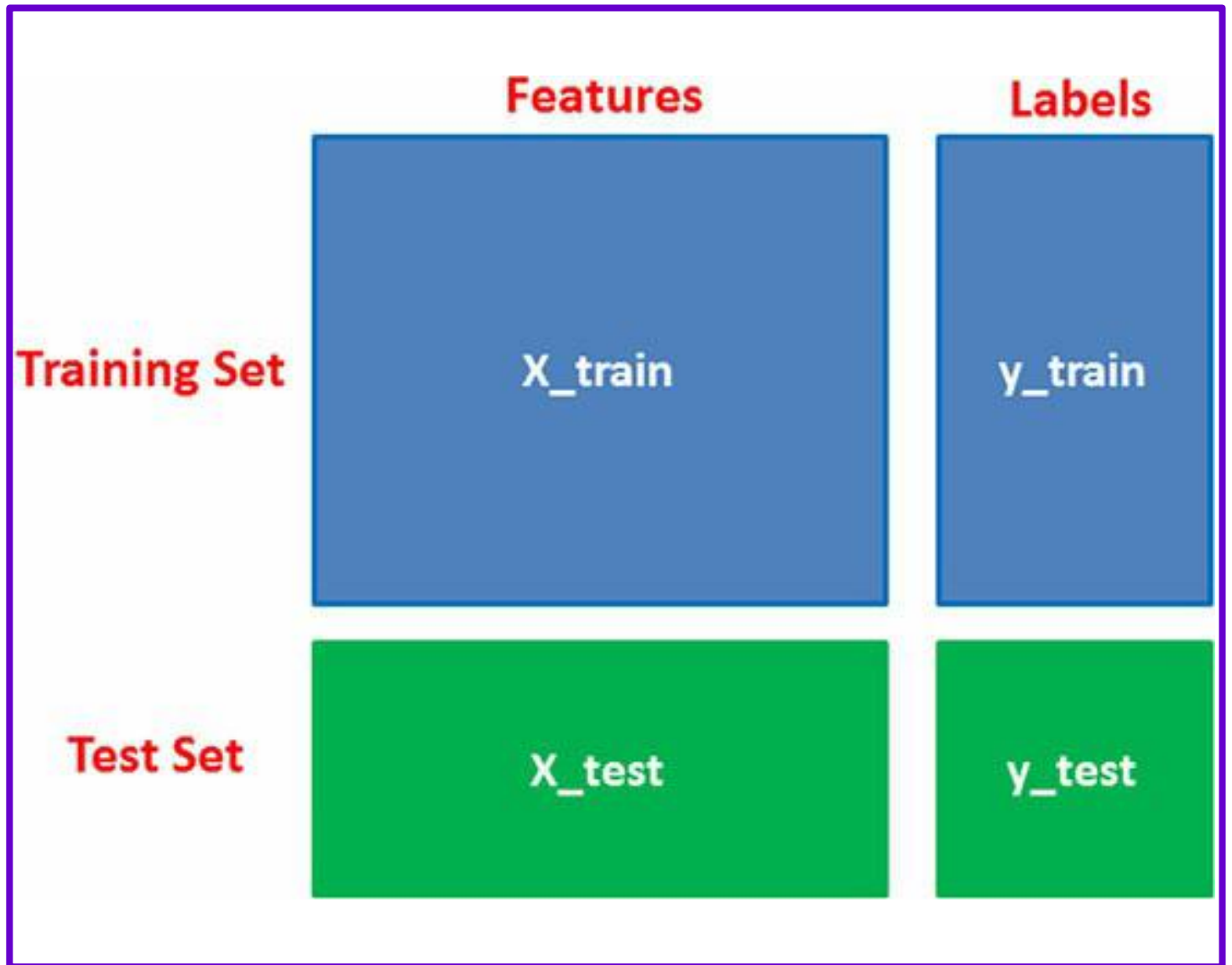
Supervised machine learning is a branch of artificial intelligence that focuses on training models to make predictions or decisions based on labeled training data.





## Basic Data Splitting for ML

Input for training classification model is a pair of input output features, and during the phase of testing, it only takes input features to predict.



# Splitting the Dataset

As we can see, we have split the dataset into 3:1 train to test ratio. For any kind of classification problem, this is the technique used to split the dataset

```
from sklearn.model_selection import train_test_split

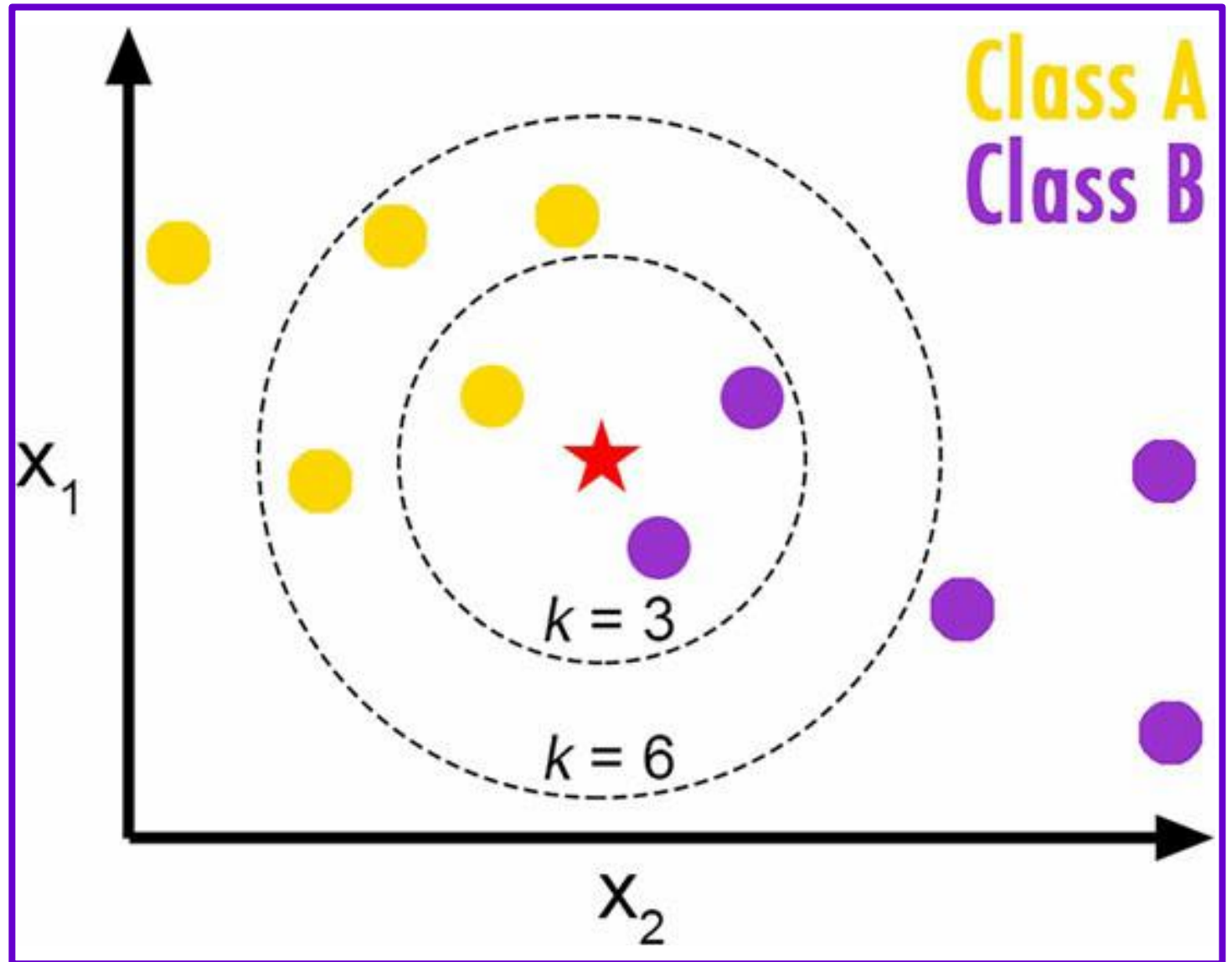
X_train, X_test, y_train, y_test = train_test_split( \
    iris_data[['sepal_length', 'petal_length', 'petal_width']], \
    iris_data[['target']], test_size=0.33, random_state=1)
print(iris_data.shape)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

(150, 5)

((100, 3), (50, 3), (100, 1), (50, 1))
```

## Choose ML Algorithms (K - nearest Neighbors K-nn)

From the image, we can see when  $K = 3$  (the Violet circle, Class B), the class it predicts for the unknown data point is Class B but, when we increase the  $K = 6$  (the Yellow circle), then the class of the unknown datapoint is Class A



# Model Training

As this is a classification model, we need to train them with input-output pair.

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=1)
model.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

# Evaluation (The Confusion Matrix)

A confusion matrix is also referred to as an error matrix, which **gives us the total number of correct vs. incorrect result** in the form of a matrix.

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) <b>Type II Error</b>	<b>Sensitivity</b> $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) <b>Type I Error</b>	True Negative (TN)	<b>Specificity</b> $\frac{TN}{(TN + FP)}$
		<b>Precision</b> $\frac{TP}{(TP + FP)}$	<b>Negative Predictive Value</b> $\frac{TN}{(TN + FN)}$	<b>Accuracy</b> $\frac{TP + TN}{(TP + TN + FP + FN)}$

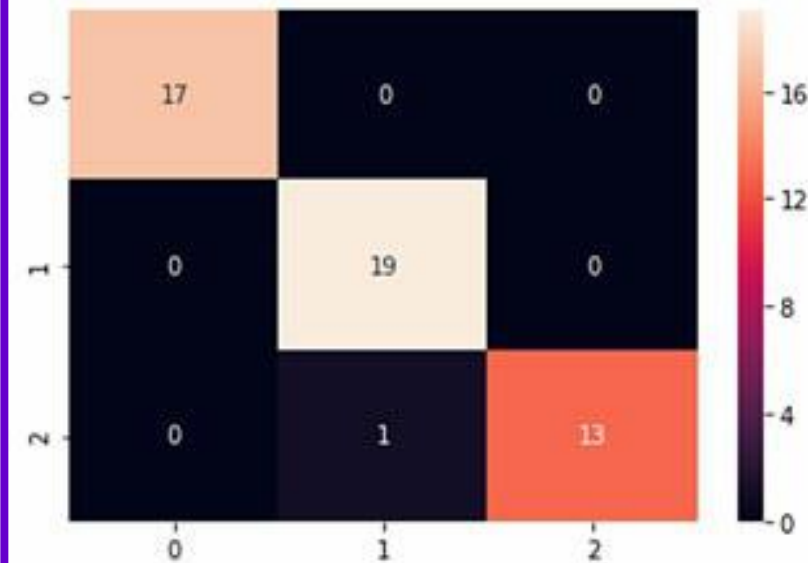
# Evaluation (The Confusion Matrix)

The diagonal represents true positives, i.e., it has correctly predicted all the classes which it was intended to. There is one misclassification for “class 2”, where it should have been predicted “class 2” instead it predicted “class 1”.

```
from sklearn.metrics import confusion_matrix  
  
sns.heatmap(confusion_matrix(y_test, model.predict(X_test)), annot=True)  
print(y_test['target'].value_counts())
```

```
1    19  
0    17  
2    14
```

Name: target, dtype: int64





# Model Tuning

Tuning is a process of improving the model's accuracy by tuning the hyperparameters, in this case; the hyperparameters are no. of clusters and no. of iterations.

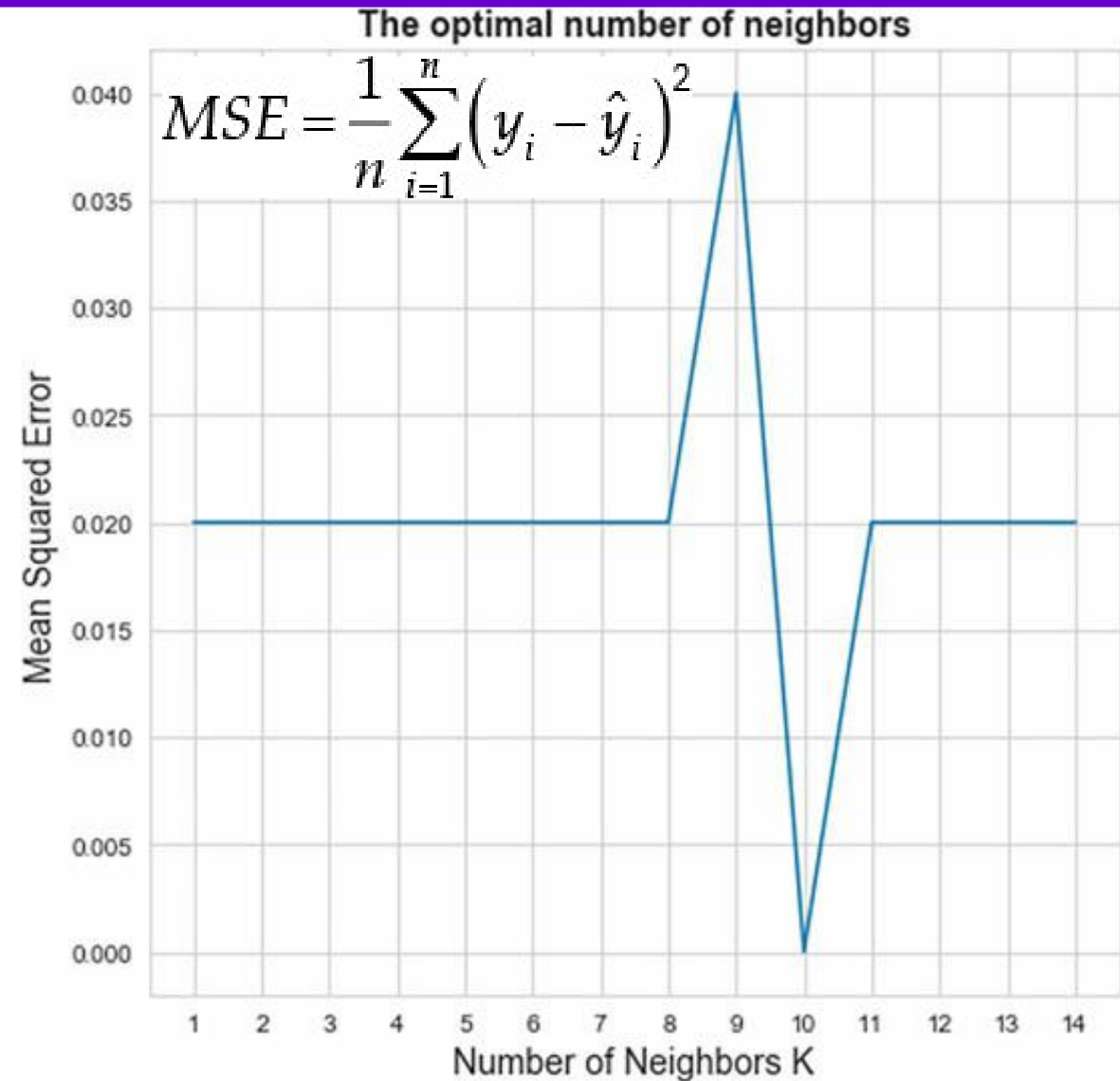
```
import matplotlib.pyplot as plt
from sklearn import metrics

# try K=1 through K=25 and record testing accuracy
k_range = range(1, 15)
mse = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    mse.append(metrics.mean_squared_error(y_test, knn.predict(X_test)))

plt.figure()
plt.figure(figsize=(7,7))
plt.title('The optimal number of neighbors', fontsize=14, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=14)
plt.ylabel('Mean Squared Error', fontsize=14)
sns.set_style("whitegrid")
plt.xticks(np.arange(min(k_range), max(k_range)+1, 1.0))
plt.plot(k_range, mse)
plt.show()
```

## k-NN vs. Error plot

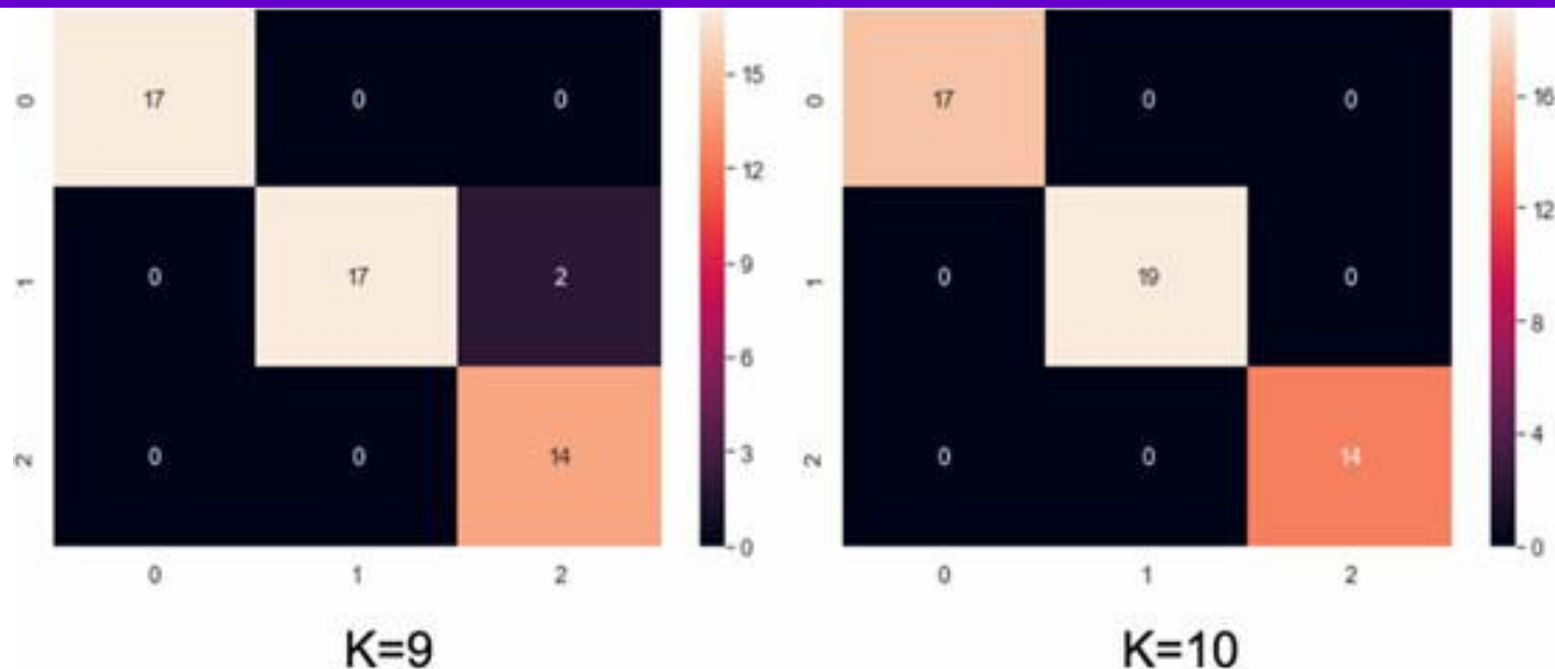
We can see that error is highest when we choose the value of  $K = 9$ , and it is lowest when  $K = 10$ . Lower the MSE better the model it is.





# Comparison of best and the worst model

The confusion matrix shows that when  $K = 10$  the MSE is Zero, which means all the classes are successfully classified, and there is no misclassification



- 1 Machine Learning Pipeline
- 2 Exploratory Data Analysis
- 3 Classification
- 4 Clustering
- 5 Regression

# Clustering

Clustering is the method by which the similar data points in n-dimensional space is **divided into different groups**.

Data points in each group will be similar to each other, and it will be different from other groups.

# Clustering

We need to remember that clustering is an unsupervised type of machine learning, so the input to the training algorithm will only be the input features.(remove targets columns)

```
data = iris_data[['sepal_length', 'petal_length', 'petal_width']]  
data.head()
```

	sepal_length	petal_length	petal_width
0	5.1	1.4	0.2
1	4.9	1.4	0.2
2	4.7	1.3	0.2
3	4.6	1.5	0.2
4	5.0	1.4	0.2

# K-means

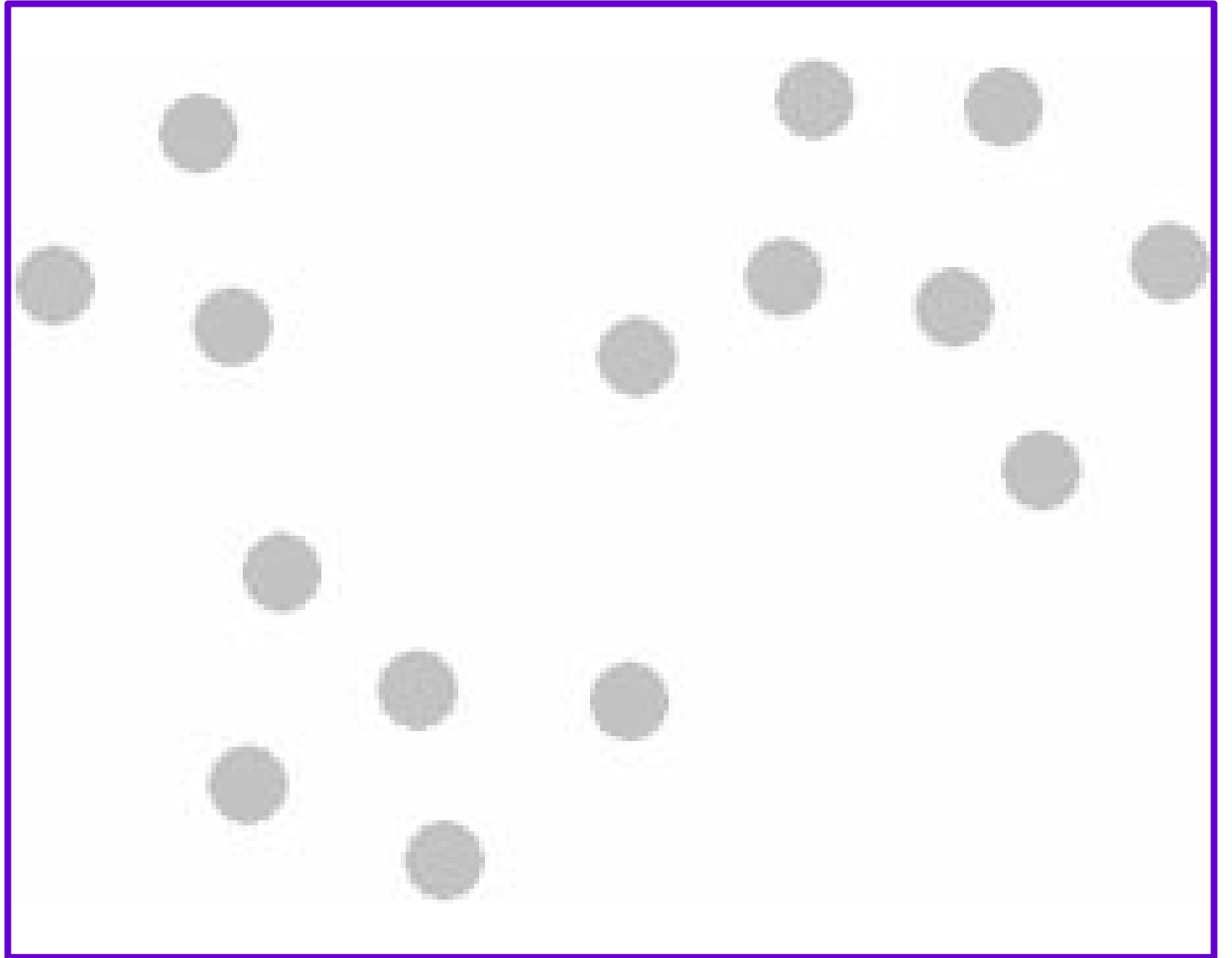
**K-means clustering** is an unsupervised machine learning algorithm to group unlabeled data into multiple classes.

$$\sum_{i=1}^n \sum_{j=1}^k (\|x_i - c_j\|)^2 = 1$$

Where,  $\|x_i - c_j\|$  is the Euclidean distance between the datapoint  $x_i$  and the centroid  $c_j$  which is looped overall  $n$ th points in the  $i$ th cluster for all  $k$  clusters.

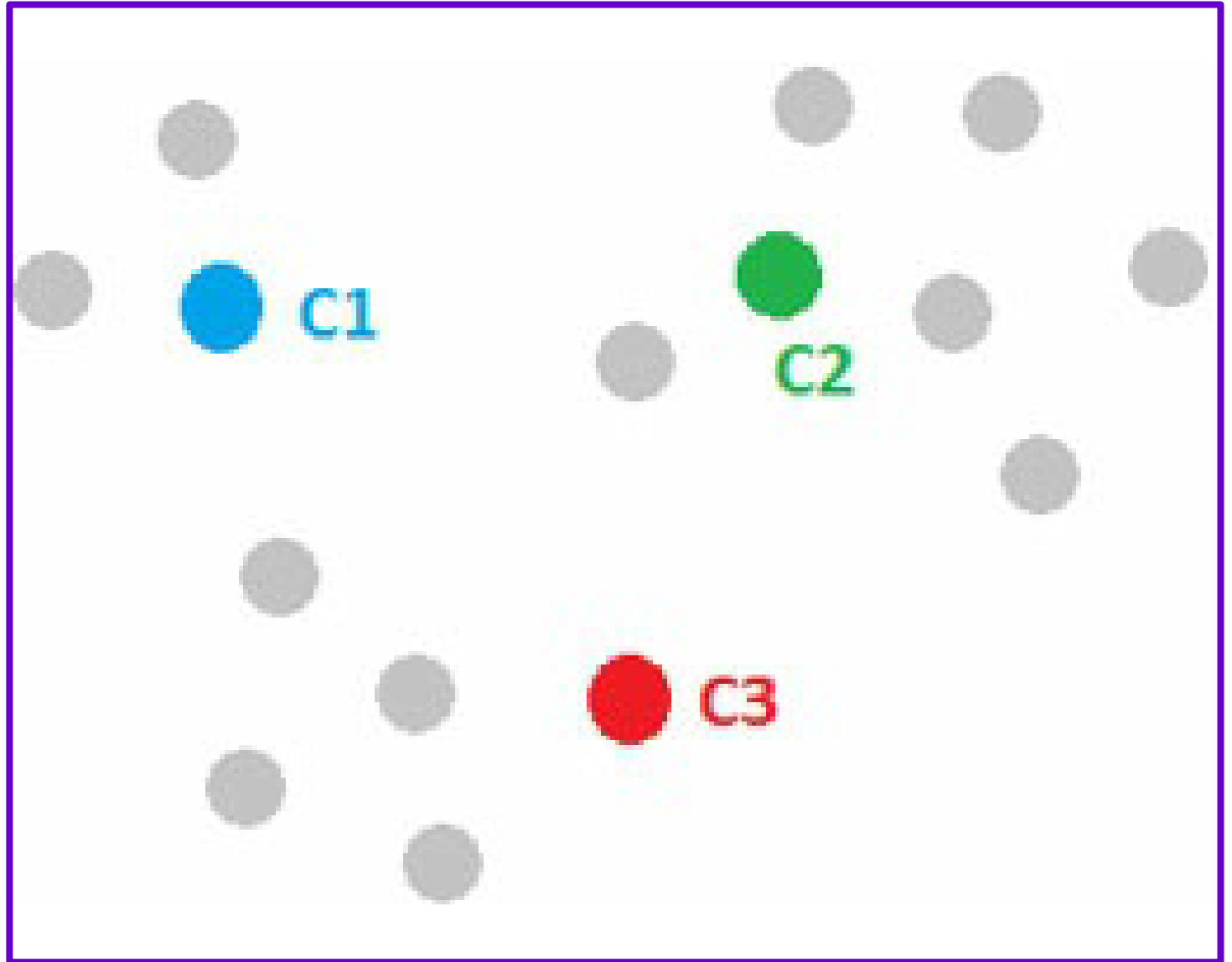
## K-means (How Does It Work-1)

Plot all the data points in n-dimensional feature space and decide the value of K i.e., the number of clusters



## K-means (How Does It Work-2)

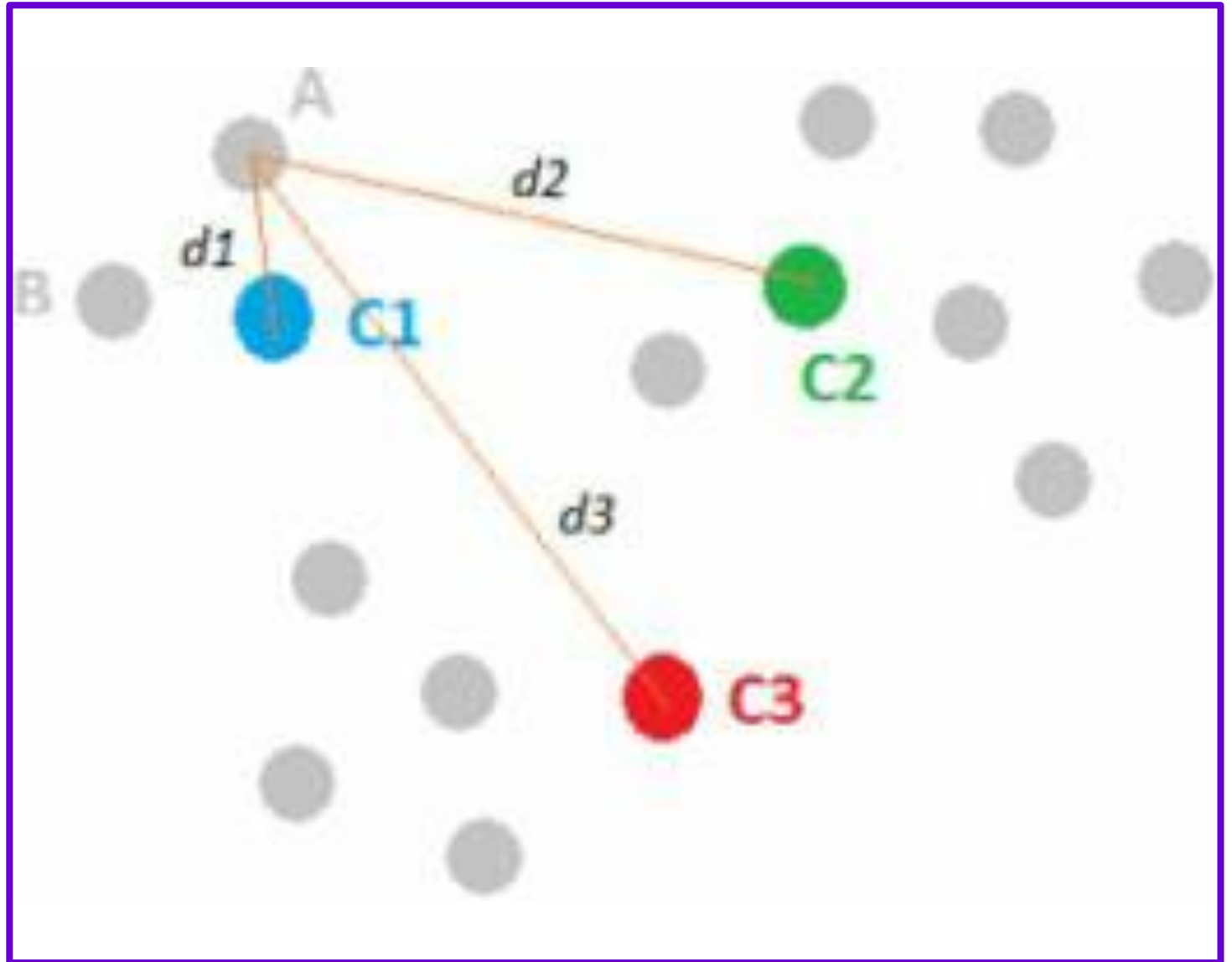
Now randomly plot the value of K points on the graph that will represent each centroid of the cluster.





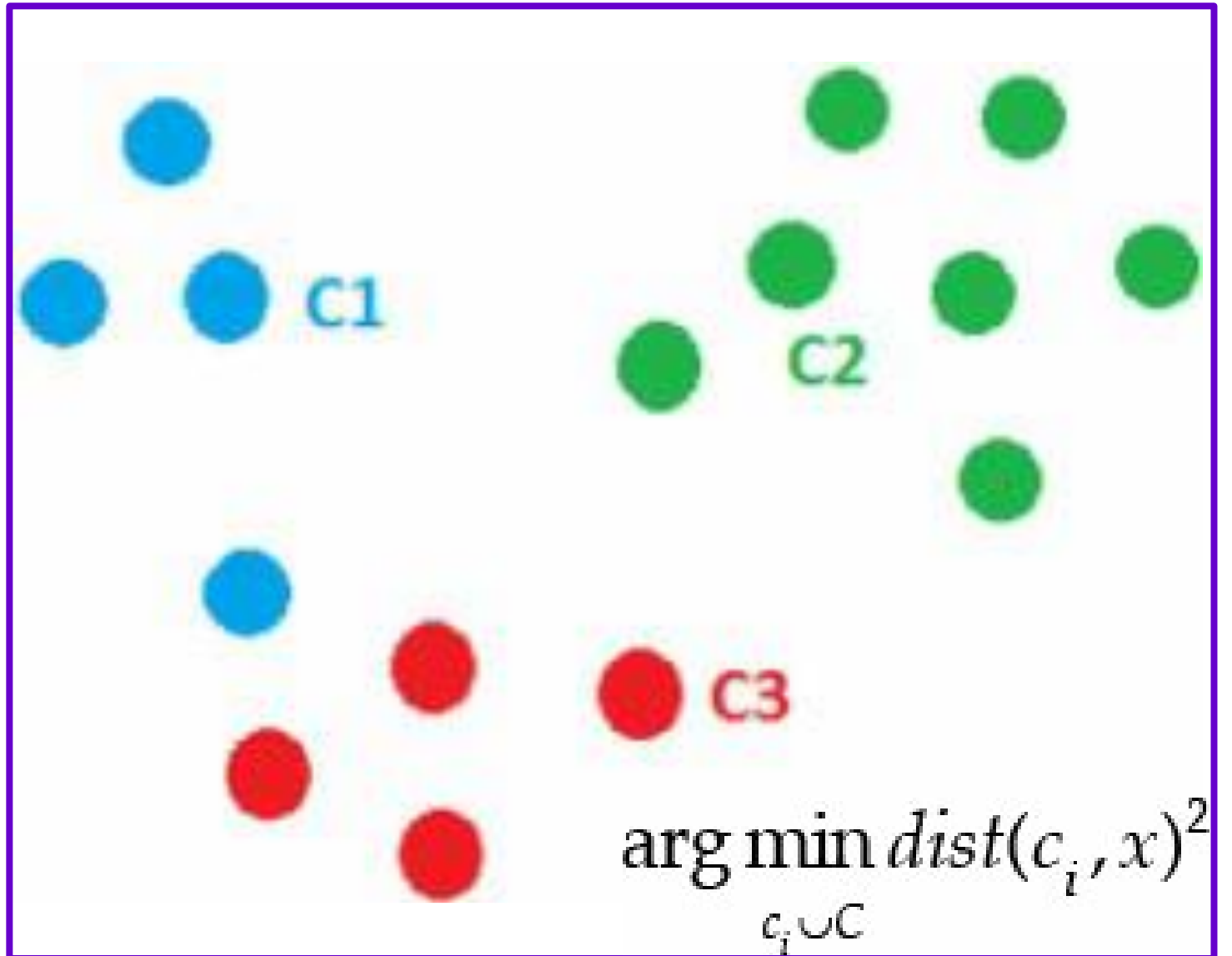
## K-means (How Does It Work-3)

Calculate all the Euclidean distance of the datapoint from the centroids.



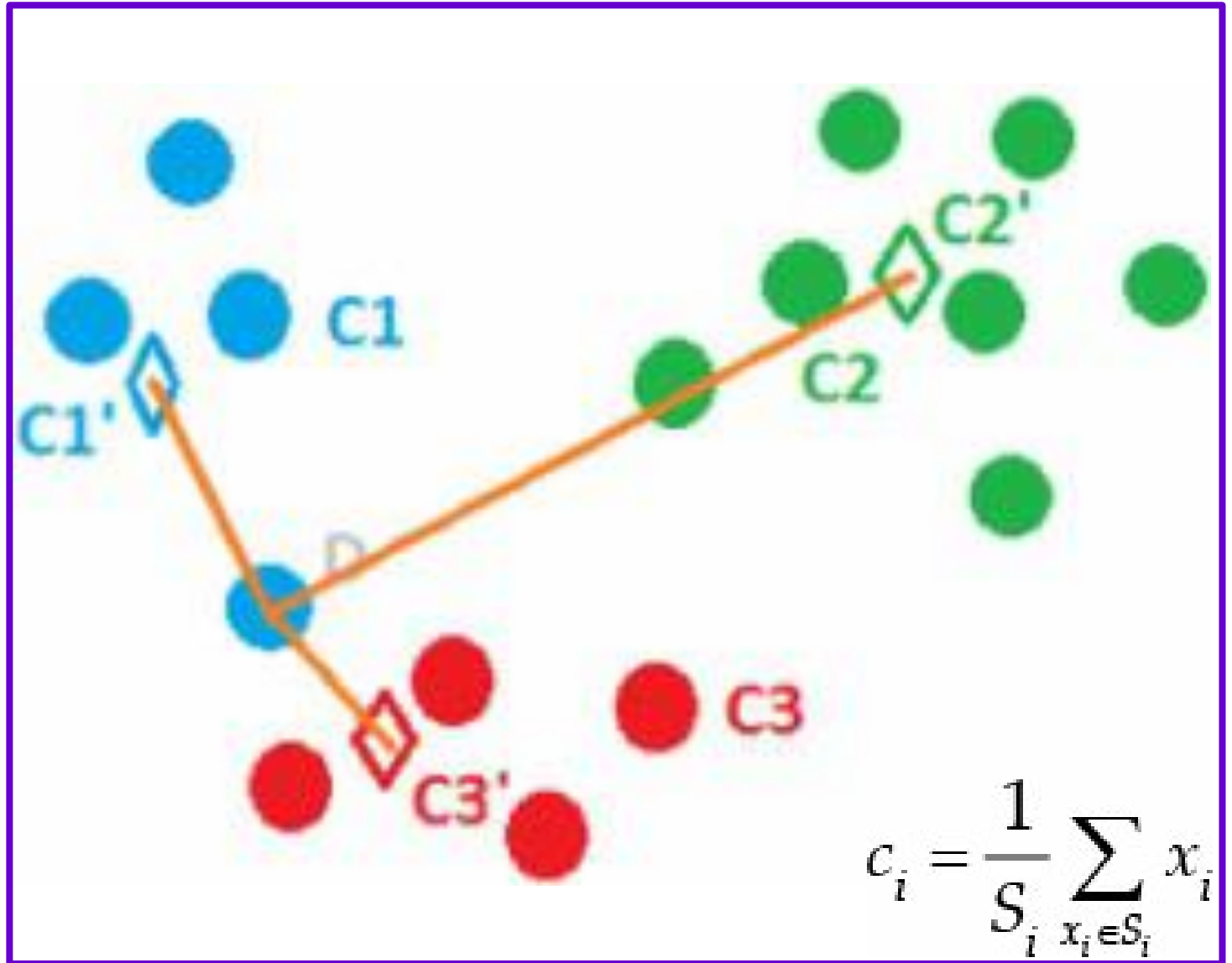
## K-means (How Does It Work-4)

Assign the least  
distant centroid as a  
class to the data  
points



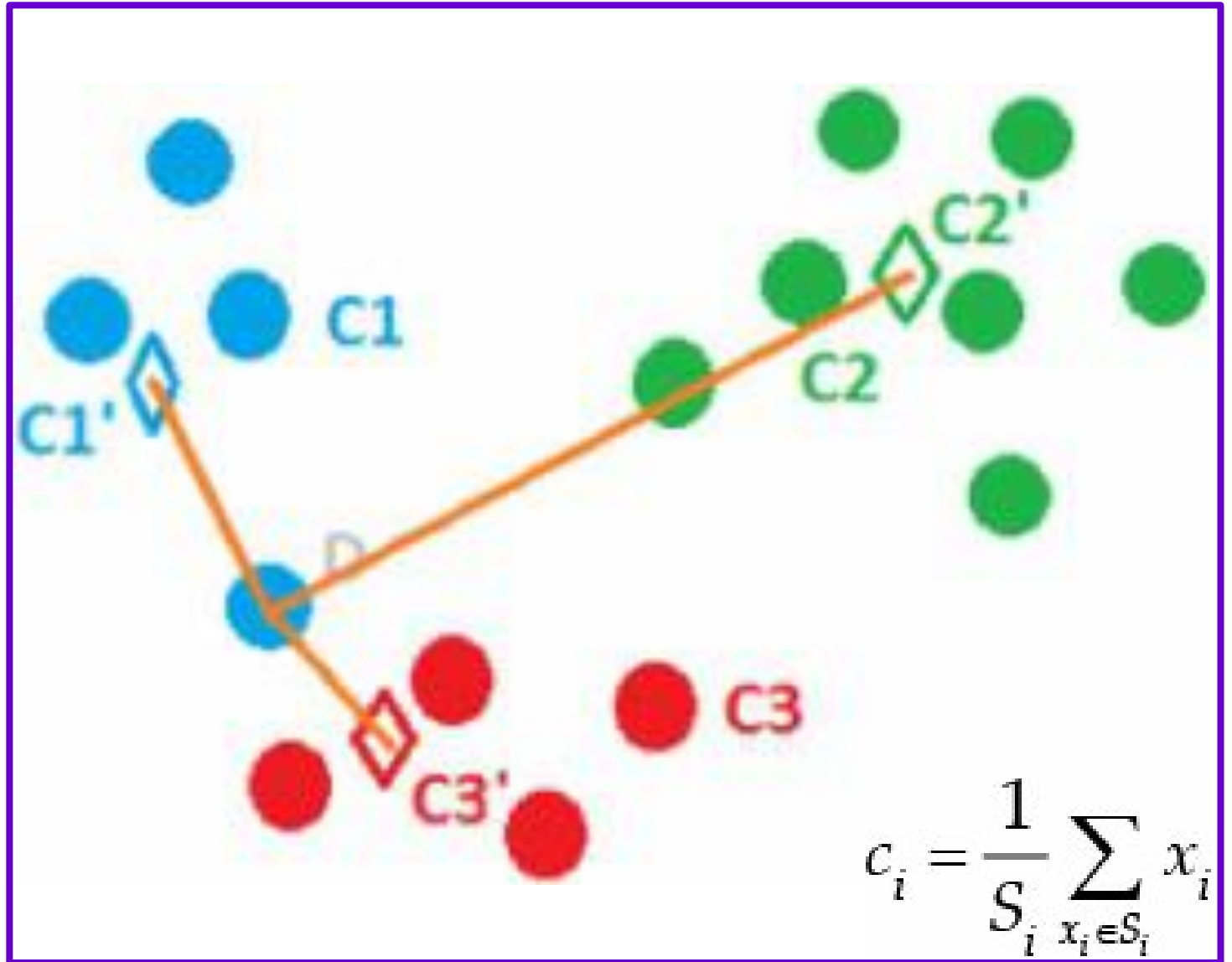
## K-means (How Does It Work-5)

Calculate the new value of centroid with the data points assigned from the previous step



## K-means (How Does It Work-6)

Repeat Step 3 to Step 5 until and unless the position of the centroid is not changing, or we can say until convergence



# Modeling K-means clustering with entire data

Let us start with all the features and train K-Means where  $K = 2$ . It is very important to know that we need to minimize the objective function.

```
from sklearn.cluster import KMeans

model = KMeans(n_clusters=2, max_iter=1000)
model.fit(data)
print(model, end="\n\n")
print("Centriods: \n" + \
      str(model.cluster_centers_))

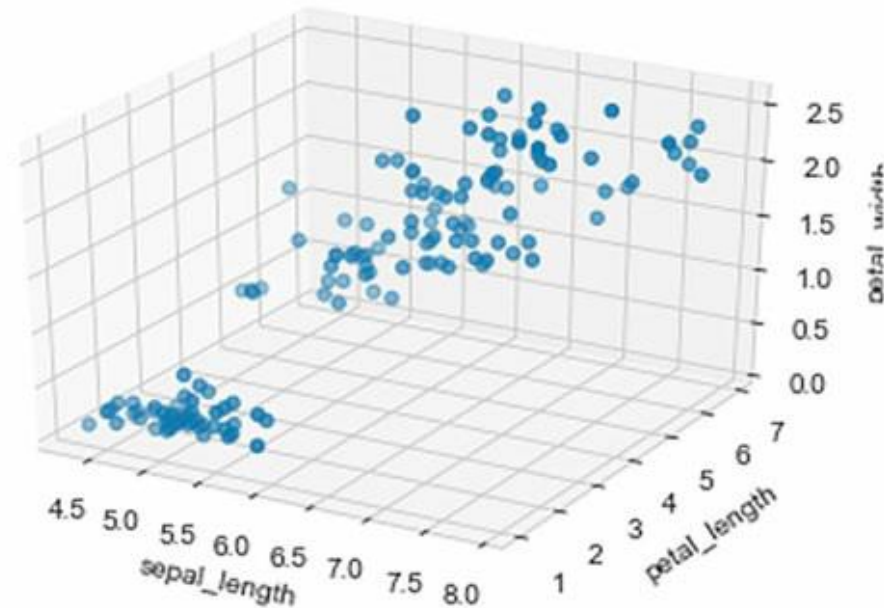
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=1000,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)

Centriods:
[[5.00555556 1.5962963  0.3037037 ]
 [6.31458333 4.97395833 1.703125  ]]
```

## 3D plot for three features with data points

We can see the visualizing 3D graph is tough, and understanding the spatial information is complex. Just for visualization, we will be using only two features so that it can be plotted on the 2-D graph.

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data["sepal_length"],
           data["petal_length"],
           data["petal_width"],)
ax.set_xlabel('sepal_length')
ax.set_ylabel('petal_length')
ax.set_zlabel('petal_width')
plt.show()
```





# K-means with two features

```
from sklearn.cluster import KMeans
```

```
model = KMeans(n_clusters=2, max_iter=1000)
```

```
pred = model.fit_predict(data.drop(columns=["sepal_length"]))
```

```
print(model, end="\n\n")
```

```
print("Centroids: \n" + \n      str(model.cluster_centers_))
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=1000,\n        n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',\n        random_state=None, tol=0.0001, verbose=0)
```

```
Centroids:
```

```
[[4.92525253 1.68181818]\n [1.49215686 0.2627451 ]]
```

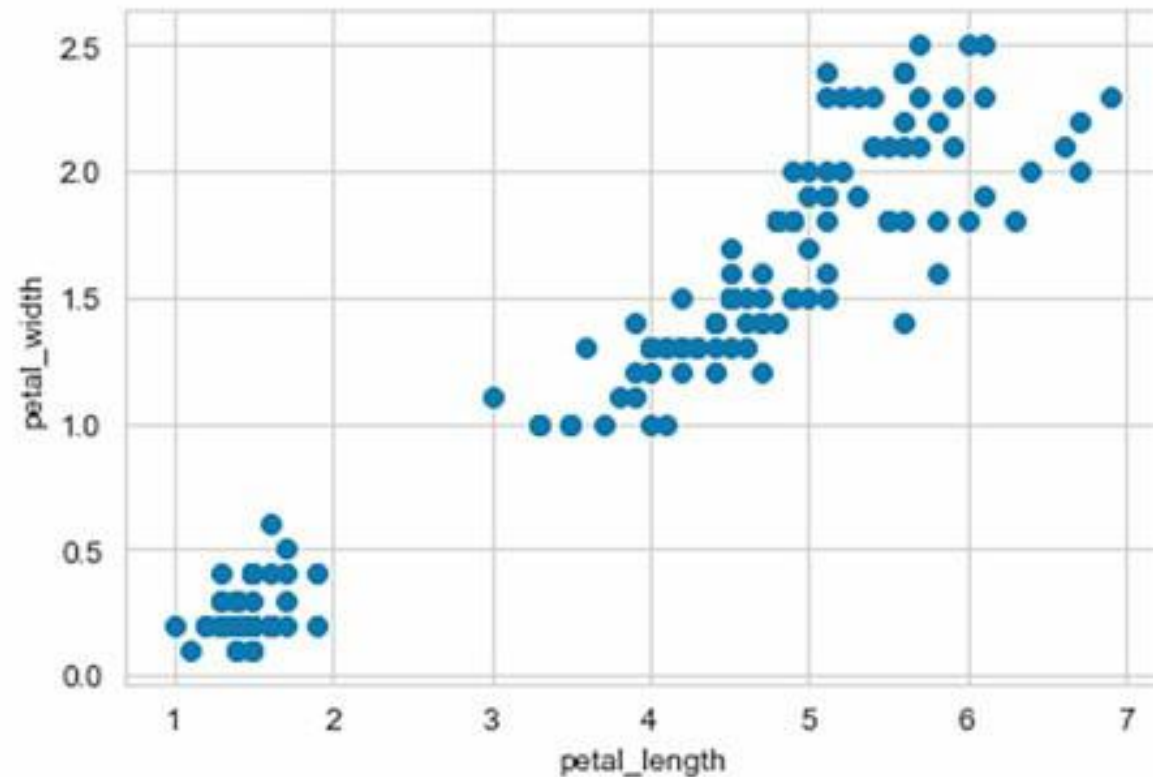


Visualizing two  
features data point  
with scatter plot

If we plot all the data points and manually try to find the cluster when we can see two groups, first group - at the lower left-hand corner, and the second group-at the top right corner.

```
import matplotlib.pyplot as plt

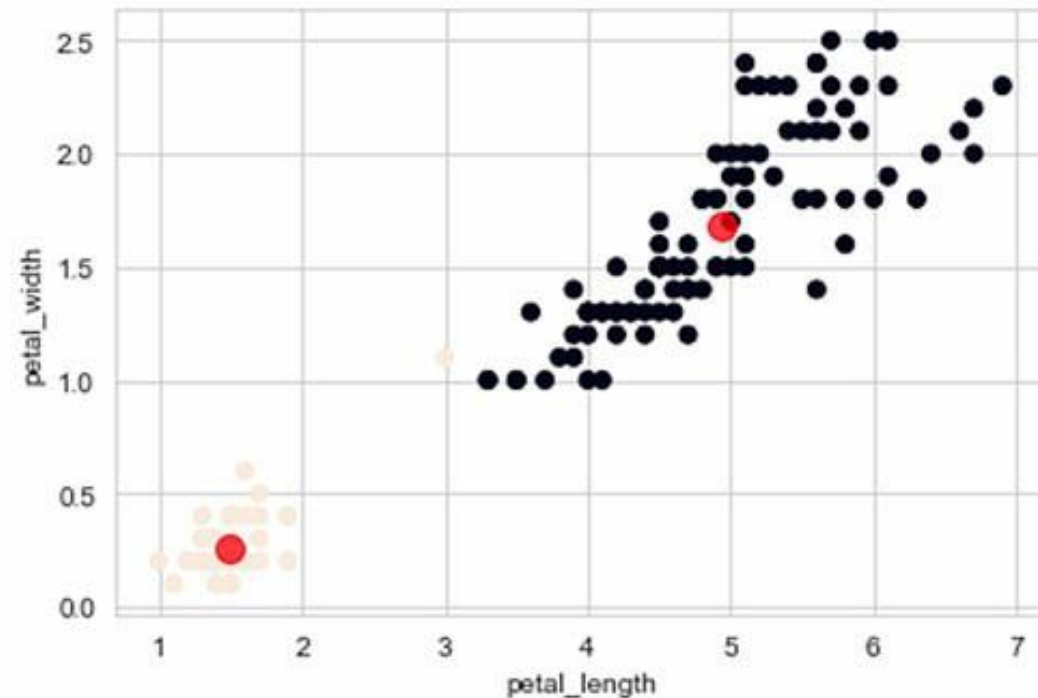
plt.scatter(x=data["petal_length"],
            y=data["petal_width"])
plt.xlabel("petal_length"); plt.ylabel("petal_width");
```



Visualizing the  
Centroids for two  
features and K=2

The red dots are the  
centroids of the  
clusters from the  
trained model with K  
= 2. We had made  
some hypothesis  
before plotting and, it  
matches the  
hypothesis.

```
import matplotlib.pyplot as plt
plt.scatter(x=data["petal_length"],
            y=data["petal_width"], c=pred)
centers = model.cluster_centers_
plt.xlabel("petal_length")
plt.ylabel("petal_width")
plt.scatter(centers[:, 0], centers[:, 1], c='Red',
            s=100, alpha=0.75);
```

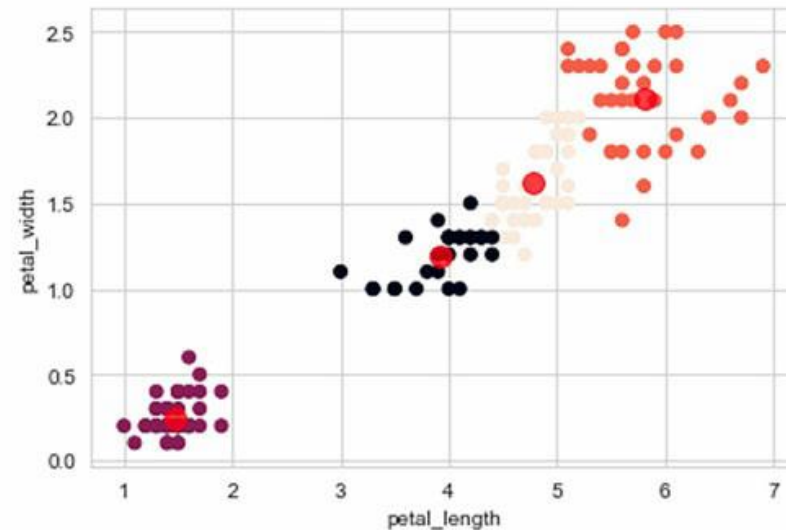


## Visualizing centroids when $K=4$

When we have increased the number of centroids to  $K = 4$ , then we get the clusters like the above image.

```
import matplotlib.pyplot as plt
plt.scatter(x=data["petal_length"],
            y=data["petal_width"], c=pred)
centers = model.cluster_centers_
plt.xlabel("petal_length")
plt.ylabel("petal_width")
print("Centroids: \n" + \
      str(model.cluster_centers_))
plt.scatter(centers[:, 0], centers[:, 1], c='Red',
            s=100, alpha=0.75);
```

```
Centroids:
[[3.92222222 1.1962963 ]
 [1.462      0.246      ]
 [5.80285714 2.11142857]
 [4.77894737 1.61578947]]
```



# Evaluation & Model Tuning

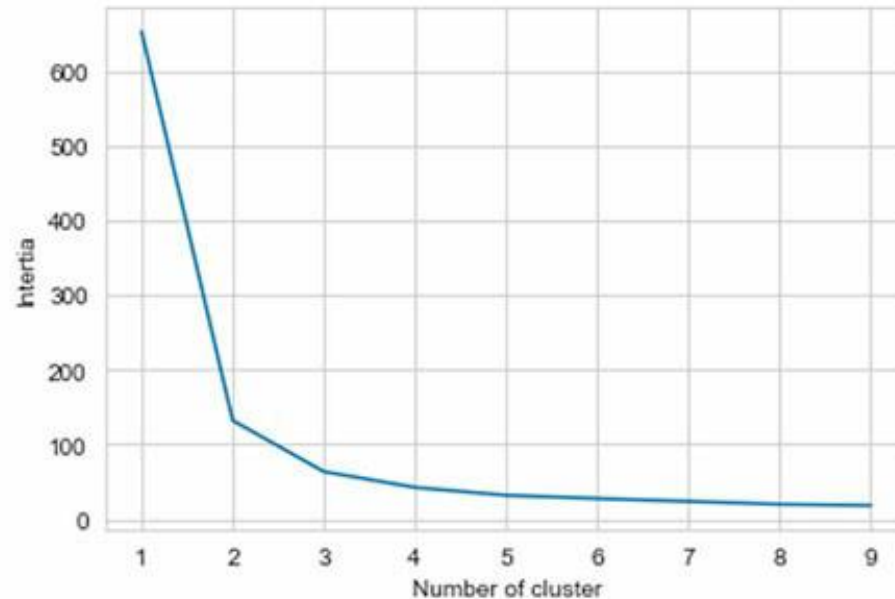
Here, we will play with the value to K and see how it changes the error of the overall cluster. Inertia or within-cluster sum-of-squares criterion

$$\sum_{i=0}^n \min_{\mu_j \in C} \left( \|x_i - c_j\|^2 \right)$$

# Elbow method using Inertia

The Elbow method is a process of selecting the optimal value of K for the data with the given clustering algorithm. The point with least error and the place where it follows an elbow pattern, that point can be considered as the optimal value of K.

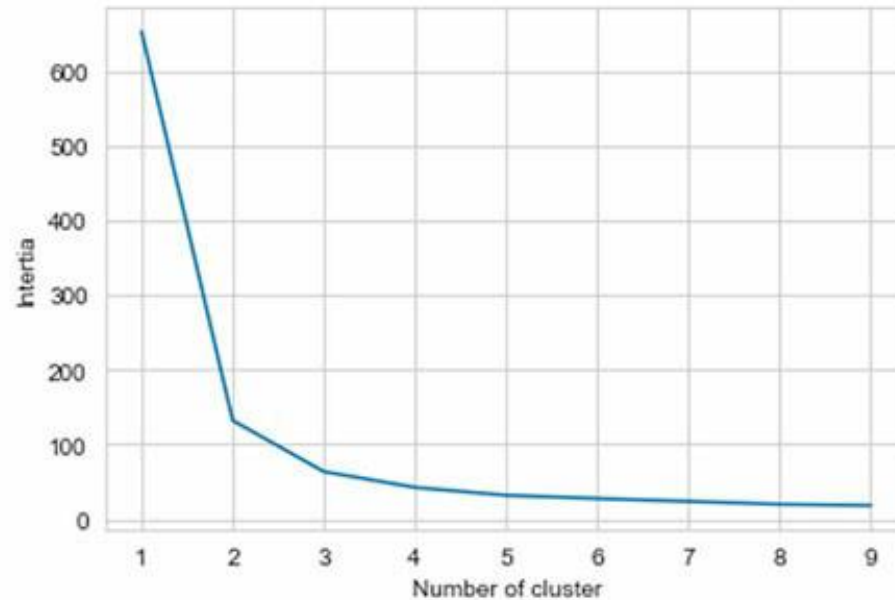
```
intertia = {}  
for k in range(1, 10):  
    kmeans = KMeans(n_clusters=k, max_iter=1000)  
    kmeans.fit(data)  
    # Inertia: Sum of distances of samples to  
    # their closest cluster center  
    intertia[k] = kmeans.inertia_  
plt.figure()  
plt.plot(list(intertia.keys()), list(intertia.values()))  
plt.xlabel("Number of cluster")  
plt.ylabel("Intertia")  
plt.show()
```



## Elbow Chart for 2 Features

We can confirm our hypothesis and say, that the elbow charts look similar, and it follows the same trend. Just that inertia values of all cluster(s) are slightly different

```
intertia = {}  
for k in range(1, 10):  
    kmeans = KMeans(n_clusters=k, max_iter=1000)  
    kmeans.fit(data)  
    # Inertia: Sum of distances of samples to  
    # their closest cluster center  
    intertia[k] = kmeans.inertia_  
plt.figure()  
plt.plot(list(intertia.keys()), list(intertia.values()))  
plt.xlabel("Number of cluster")  
plt.ylabel("Intertia")  
plt.show()
```





## Silhouette Score

Silhouette gives the consistency of the clusters. It is a measure of its cluster (cohesion) compared to other clusters(separation)

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |c_i| > 1$$

$$a(i) = \frac{1}{|c_i| - 1} \sum_{j \in c_i, i \neq j} d(i, j)$$

$$b(i) = \min_{k \neq i} \frac{1}{|c_k|} \sum_{j \in c_k} d(i, j)$$

$$s(i) = \begin{cases} 1 - a(i) / b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i) / a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$



## Silhouette Score

Value of  $s(i)$  will range from,  $-1 \leq s(i) \leq 1$ . And  $s(i) = 0$  when the number of clusters is 1. A higher value indicates higher cohesiveness, so the clustering is good.

$$s(i) = \begin{cases} 1 - a(i) / b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i) / a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

## Silhouette Score

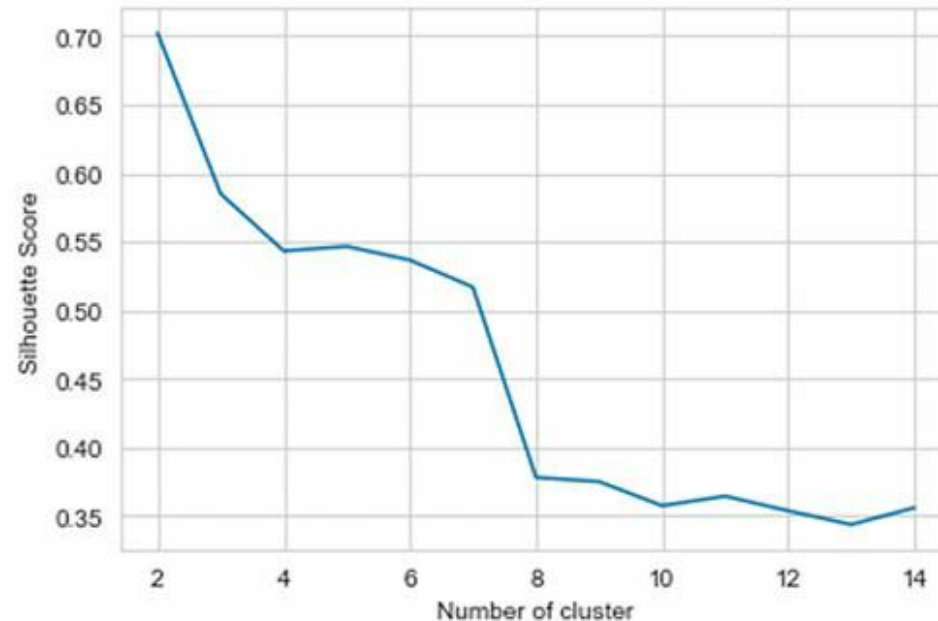
0 indicates overlapping clusters, whereas negative value means wrong clustering of the object because there exists another more similar cluster.

$$s(i) = \begin{cases} 1 - a(i) / b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i) / a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

Error vs. Number of clusters for three features

Here we are looking for higher values on the Silhouette Score for each value of K (Number of clusters).

```
score = {}  
for k in range(2, 15):  
    kmeans = KMeans(n_clusters=k, max_iter=1000)  
    kmeans.fit(data)  
    pred = kmeans.labels_  
    score[k] = silhouette_score(data, pred)  
plt.figure()  
plt.plot(list(score.keys()), list(score.values()))  
plt.xlabel("Number of cluster")  
plt.ylabel("Silhouette Score")  
plt.show()
```

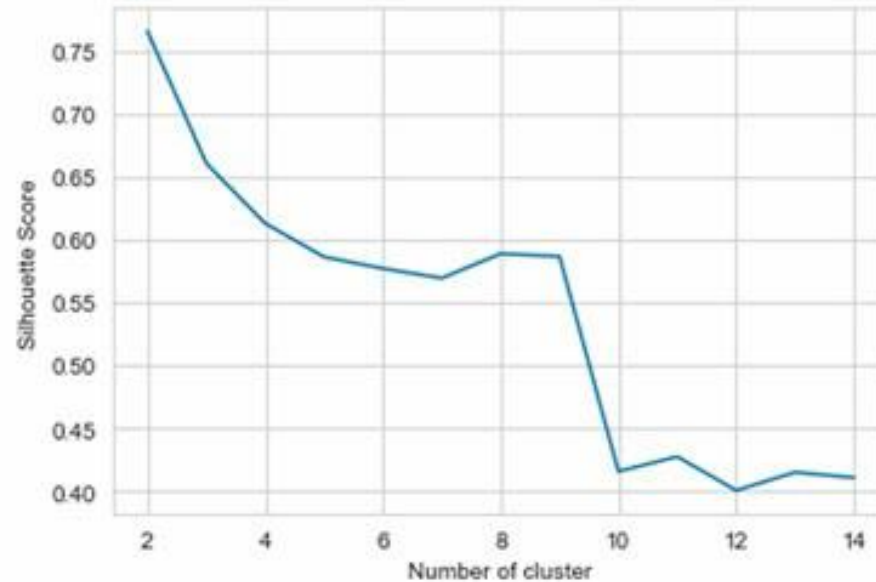


Error vs. Number of clusters for two features

Seeing the chart, we can conclude that the optimal value of K can either be 3 or 2.

```
from sklearn.metrics import silhouette_score
score = {}
for k in range(2, 15):
    kmeans = KMeans(n_clusters=k, max_iter=1000)
    kmeans.fit(data.drop(columns=["sepal_length"]))
    pred = kmeans.labels_
    score[k] = silhouette_score(data.drop(columns=["sepal_length"]),
                                pred)

plt.figure()
plt.plot(list(score.keys()), list(score.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Silhouette Score")
plt.show()
```



- 1 Machine Learning Pipeline
- 2 Exploratory Data Analysis
- 3 Classification
- 4 Clustering
- 5 Regression

# Regression

Regression is another type of Machine Learning problem, which helps us to predict continuous values.



## Regression (Data Preparation)

Here we will predict the “petal\_width” with the help of “petal\_length.”

```
data = iris_data[['petal_length', 'petal_width',  
                  'target', 'target_names']]  
data.head()
```

	petal_length	petal_width	target	target_names
0	1.4	0.2	0	setosa
1	1.4	0.2	0	setosa
2	1.3	0.2	0	setosa
3	1.5	0.2	0	setosa
4	1.4	0.2	0	setosa

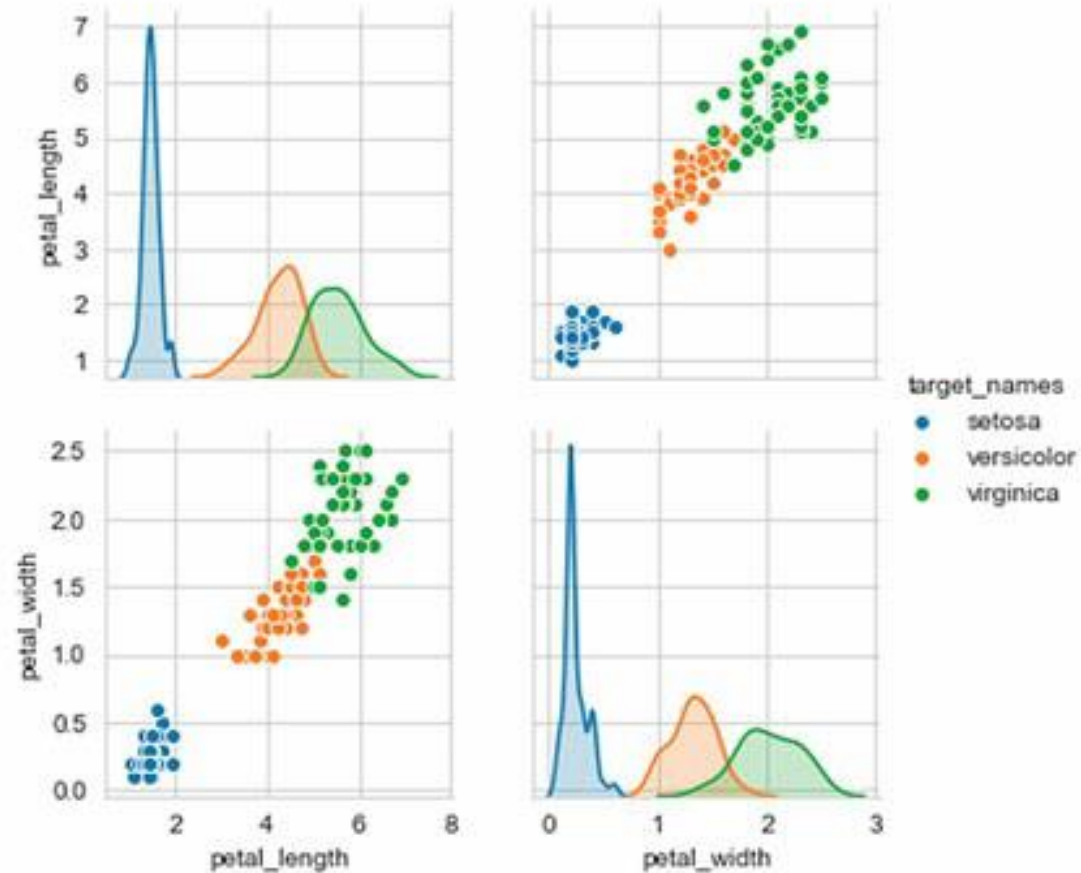


# Regression (Pair Plot for two selected features)

Let us see how both  
the features look in  
a pair plot.

```
sns.pairplot(data[['petal_length', 'petal_width', 'target_names']],  
             hue='target_names')
```

<seaborn.axisgrid.PairGrid at 0x1a2b85e4e0>



Choosing only one species

Because each species will have their characteristics so, all the features won't be logical and useful at the same time. Now, we will only keep 'versicolor' from the entire dataset

```
data = data[data['target_names']=='versicolor']  
data.head()
```

	petal_length	petal_width	target	target_names
50	4.7	1.4	1	versicolor
51	4.5	1.5	1	versicolor
52	4.9	1.5	1	versicolor
53	4.0	1.3	1	versicolor
54	4.6	1.5	1	versicolor

Dropping irrelevant columns for regression

With that, I need to remove/drop the “target\_names” and “targets” because there is no use for those columns for a regression problem

```
data.drop(columns=['target', 'target_names'],  
           inplace=True)  
print(data.shape)  
data.head()
```

(50, 2)

	petal_length	petal_width
50	4.7	1.4
51	4.5	1.5
52	4.9	1.5
53	4.0	1.3
54	4.6	1.5

# Linear Regression

**Linear regression** is a statistical method to model a linear relationship between two scalars

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \dots + \beta_n x_n$$

$\therefore$

$$\hat{y} = \beta_0 + \sum (\beta_i x_i)$$

# Linear Regression: Ordinary Least Squares (OLS)

To find the value of  $\beta_i$ ,  $i = 1 \dots n$  we can use Ordinary Least Squares (OLS). OLS is one of the methods to find unknown values

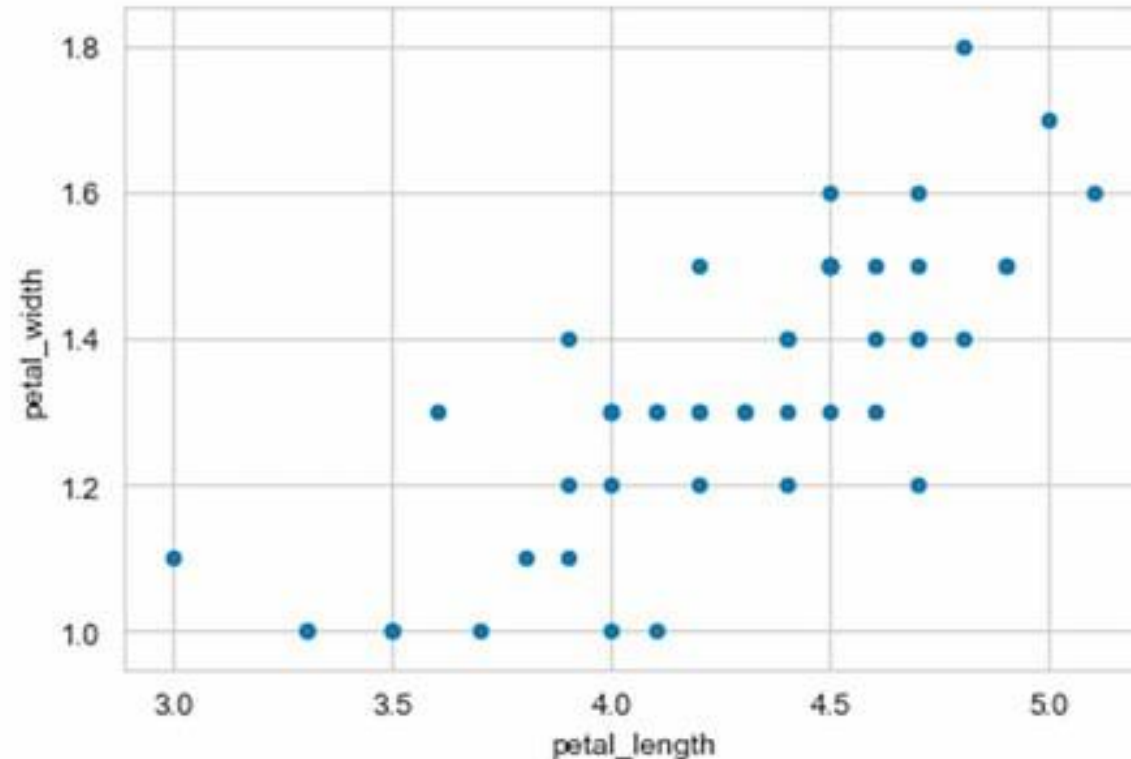
$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$
$$\Rightarrow \bar{y} - \left( \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} \right) \cdot \bar{x}$$
$$\beta_1 = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$

## Model Training

From the data pair plot, we have selected “petal\_length” and “petal\_width” for this work. If we carefully see the data, we see a linear pattern in the data. These kinds of data will perform best for the linear regression problems.

```
data.plot(x='petal_length', y='petal_width',  
          kind='scatter')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a2bf68048>
```



# Splitting data for training a regression model

Let us split the dataset into training and testing datasets

```
from sklearn.model_selection import train_test_split
X = data['petal_length'].values.reshape(-1,1)
y = data['petal_width'].values.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33, random_state=1)
print(data.shape)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

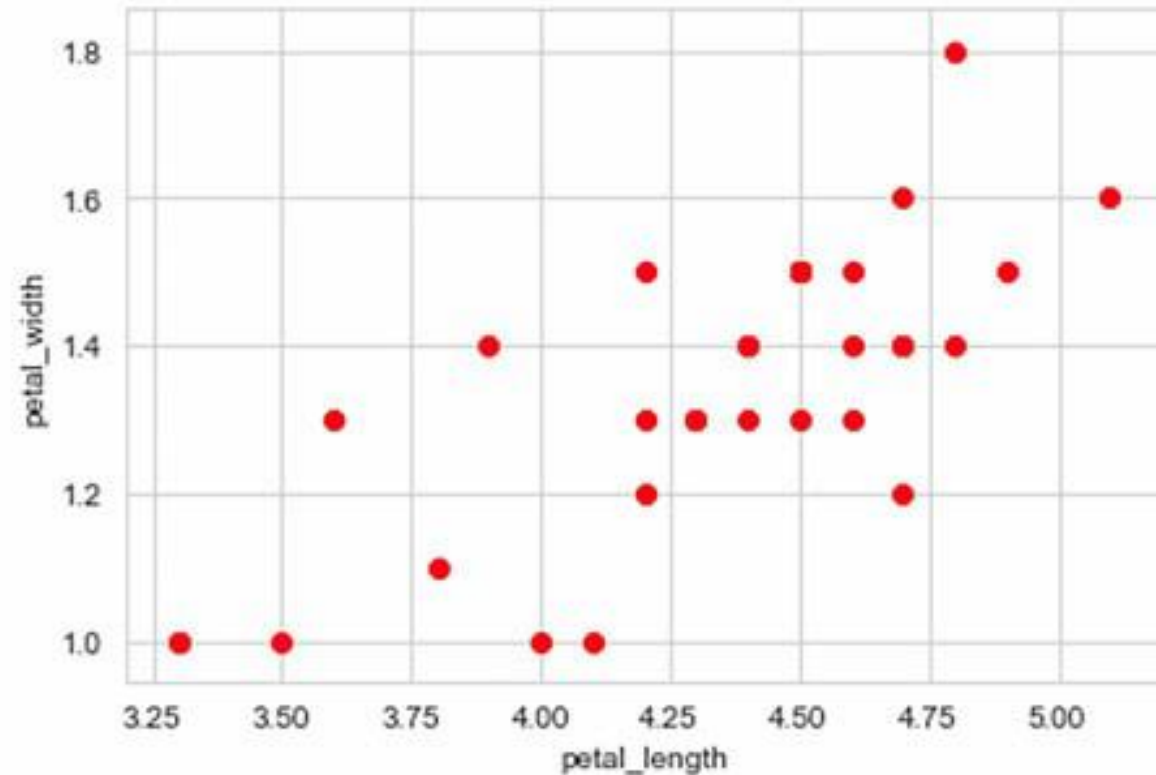
(50, 2)
((33, 1), (17, 1), (33, 1), (17, 1))
```



## Visualizing training data

If the distribution of test data and train data is different, then the model does not perform well for the test data

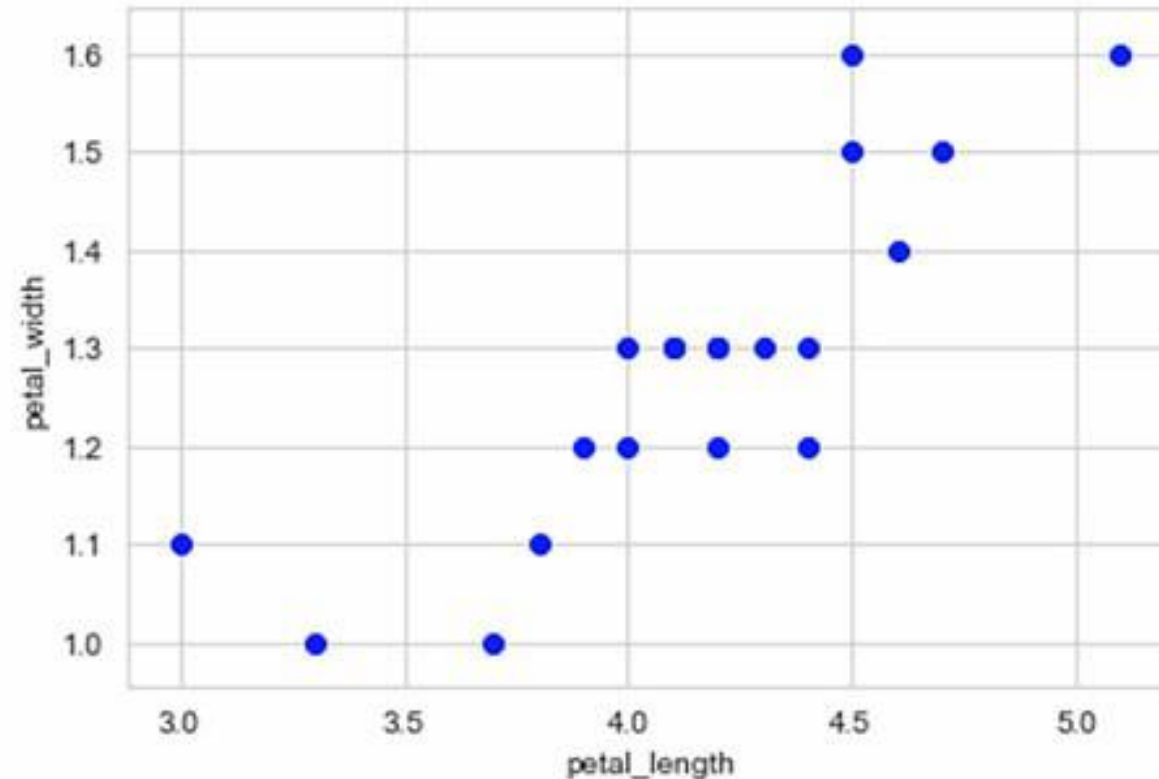
```
plt.scatter(x=X_train,y=y_train, c="Red")  
plt.xlabel("petal_length"); plt.ylabel("petal_width");
```



## Visualizing testing data

Seeing the test data and comparing it with the train data, we see a similar pattern. So, we can make a hypothesis that the model will work fine, given the linear nature of the data, test and train data follows the same linear pattern.

```
plt.scatter(x=X_test,y=y_test, c="Blue")  
plt.xlabel("petal_length"); plt.ylabel("petal_width");
```



# Modeling linear regression

Training the model is pretty straightforward using sklearn

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

# Values for the unknown variables

From the mathematical equation shown previously, we are hoping to find the intercept and the coefficient values

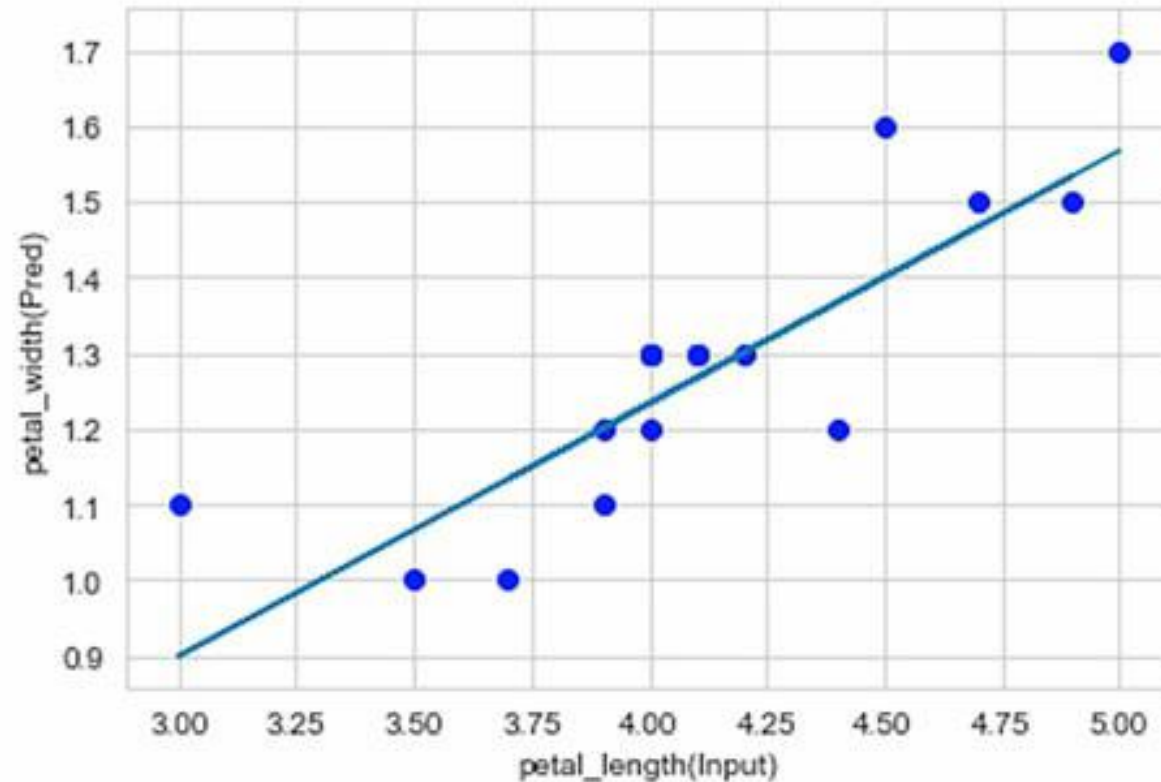
```
print("Intercept: " + str(model.intercept_))  
print("Co-ef: " + str(model.coef_))
```

```
Intercept: [-0.10228121]  
Co-ef: [[0.3338594]]
```

Plotting the predicted regression line

With the intercept and the coefficient values, we can plot a line. We can overlay the line on top of test data points and see where the predicted points lie

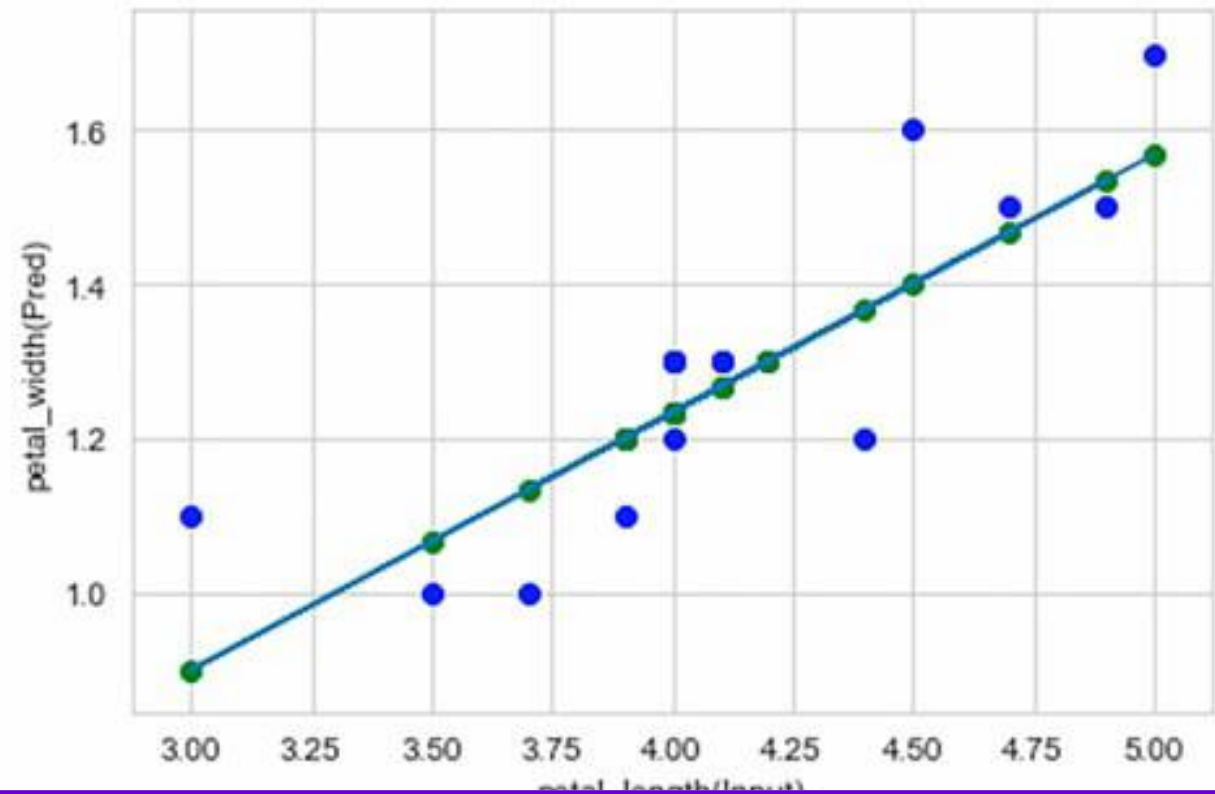
```
y_pred = model.predict(X_test)
plt.scatter(x=X_test,y=y_test, c='blue')
plt.plot(X_test, y_pred)
plt.xlabel("petal_length(Input)")
plt.ylabel("petal_width(Pred)");
```



Predicted values lie on the regression line

As discussed earlier that all the predicted points will lie on the line with the given intercept and coefficient. We will confirm that hypothesis by plotting the predicted data points.

```
y_pred = model.predict(X_test)\nplt.scatter(x=X_test,y=y_test, c='blue')\nplt.plot(X_test, y_pred)\nplt.scatter(x=X_test,y=y_pred, c='green',marker='o')\nplt.xlabel("petal_length(Input)")\nplt.ylabel("petal_width(Pred)");
```





K-means without any package

It is confirmed that all the data points are on the line, and the difference between the original value and predicted value gives us the error.

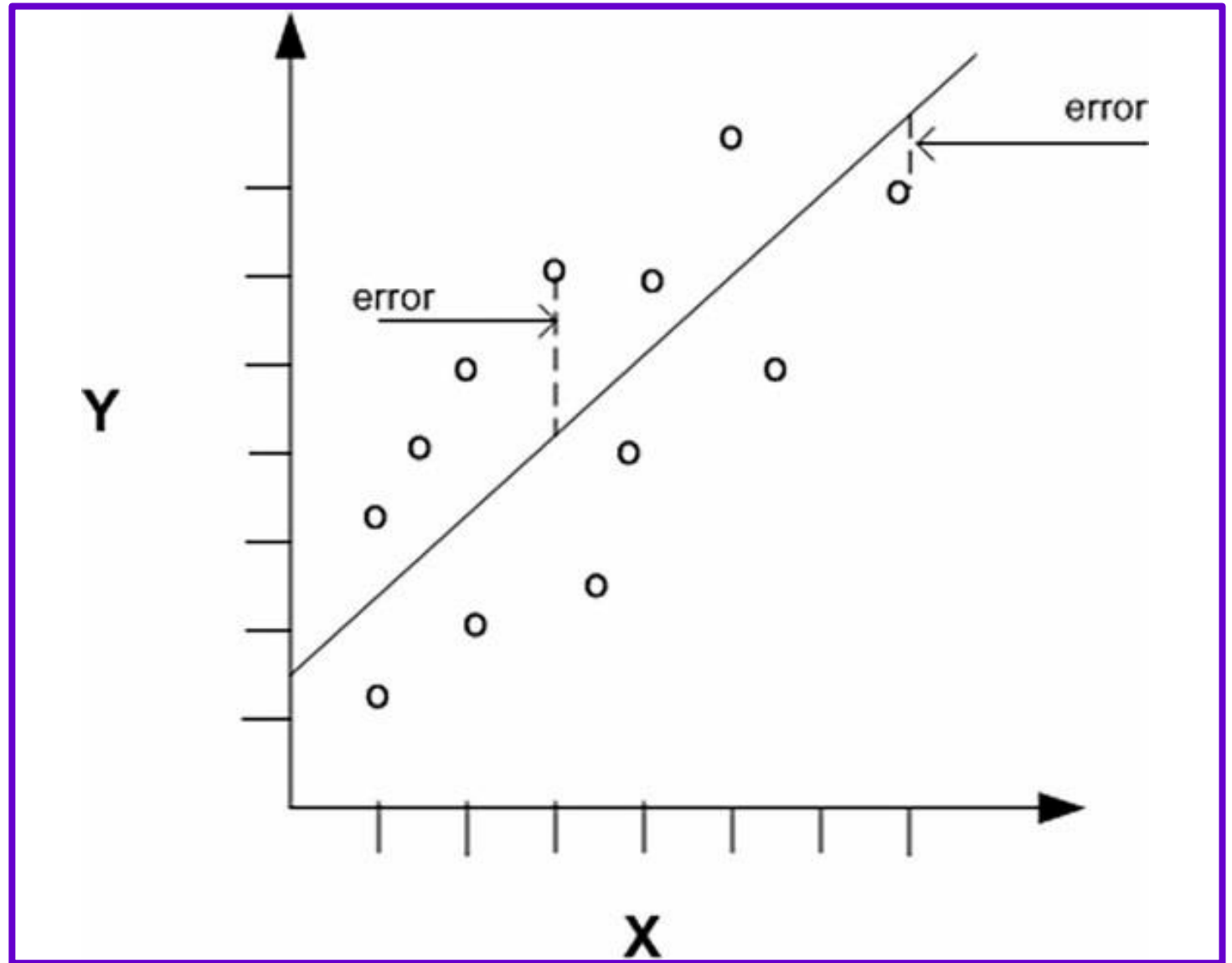
```
#mean of our inputs and outputs
x_mean = np.mean(X_train)
y_mean = np.mean(y_train)
n = len(X_train)
#using the formula to calculate the b1 and b0
numerator = 0
denominator = 0
for i in range(n):
    numerator += (X_train[i] - x_mean) * \
                 (y_train[i] - y_mean)
    denominator += (X_train[i] - x_mean) ** 2
b1 = numerator / denominator
b0 = y_mean - (b1 * x_mean)
print("Intercept: " + str(b0))
print("Co-ef: " + str(b1))
```

```
Intercept: [-0.10228121]
Co-ef: [0.3338594]
```



## Evaluation

We now need to evaluate the quality of the model. Lower the error better the model. As this is a regression problem, we will know the desired output. With the desired output and the predicted output, we can find the error in the prediction.



## Mean Absolute Error

As we know, the predicted values will lie on the line, so the error with the desired output is the perpendicular projection on the line

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

## Root Mean Squared Error

Similarly, it is the  
Root of MSE

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

## R2 Score or Coefficient of Determination

R2 score gives us a statistical measure of how good the predictions approximate the original values. The coefficient of determination value of 1 represents a perfect fit with the data points.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

$$SS_{res} = \sum_i (y_i - \hat{y}_i)^2$$

$$SS_{tot} = \sum_i (y_i - \bar{y}_i)^2$$

# Different error metrics for evaluation

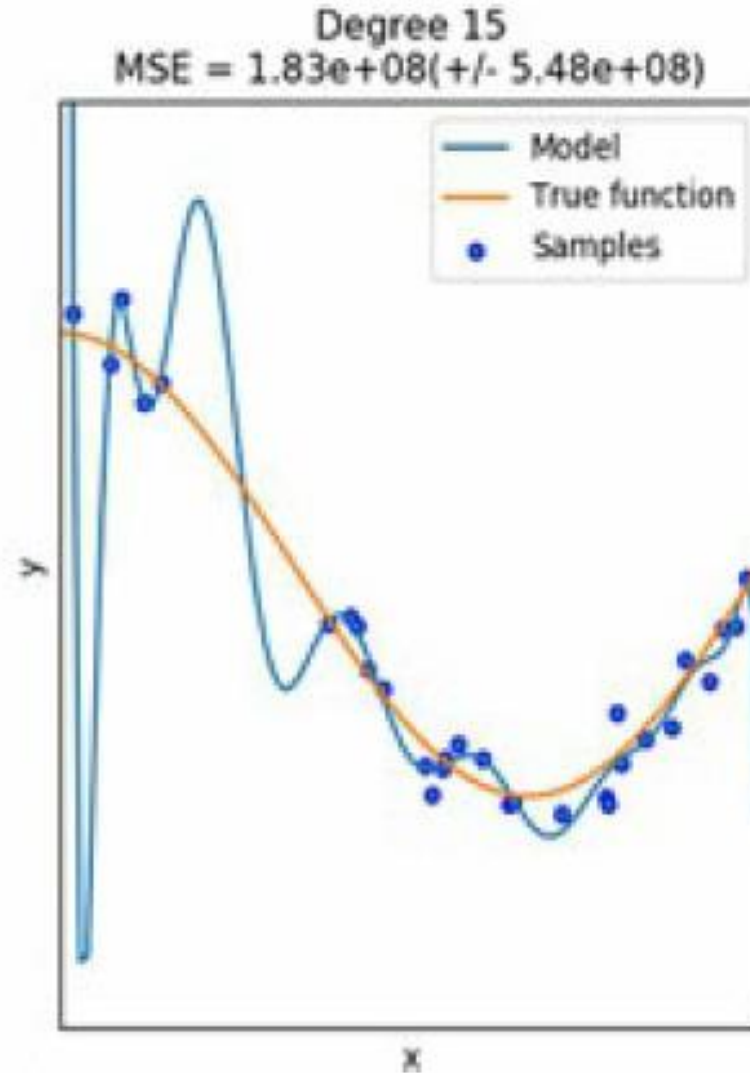
The error for the model is extremely low, and that's a good sign. But sometimes, if the error is too less, then there can be a problem of over-fitting.

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
import numpy as np
print('Mean Absolute error: %.2f' % mean_absolute_error(y_test, y_pred))
# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Root Mean squared error: %.2f" % np.sqrt(mean_squared_error(y_test, y_pred)))
# Explained variance score: 1 is perfect prediction
print('R Square Score: %.2f' % r2_score(y_test, y_pred))
```

```
Mean Absolute error: 0.08
Mean squared error: 0.01
Root Mean squared error: 0.10
R Square Score: 0.71
```

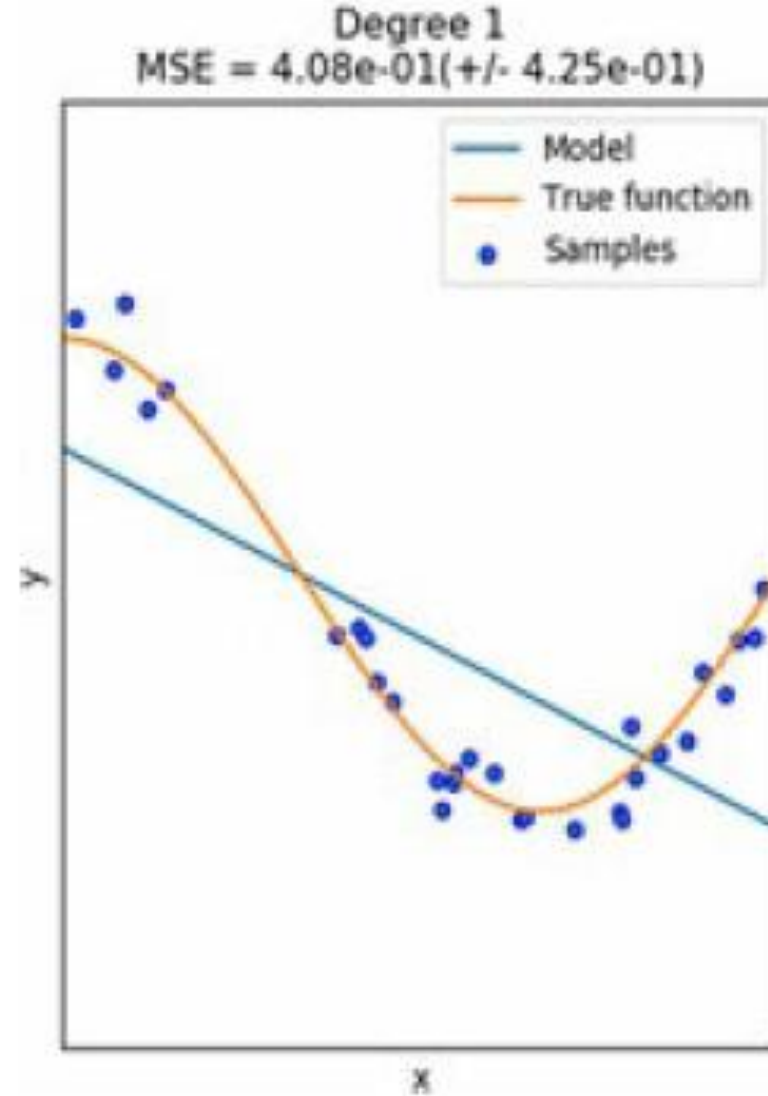
## Over-Fitting

When the model understands all the points, and it predicts accordingly, then it is said to over-fit the data.



## Over-Fitting

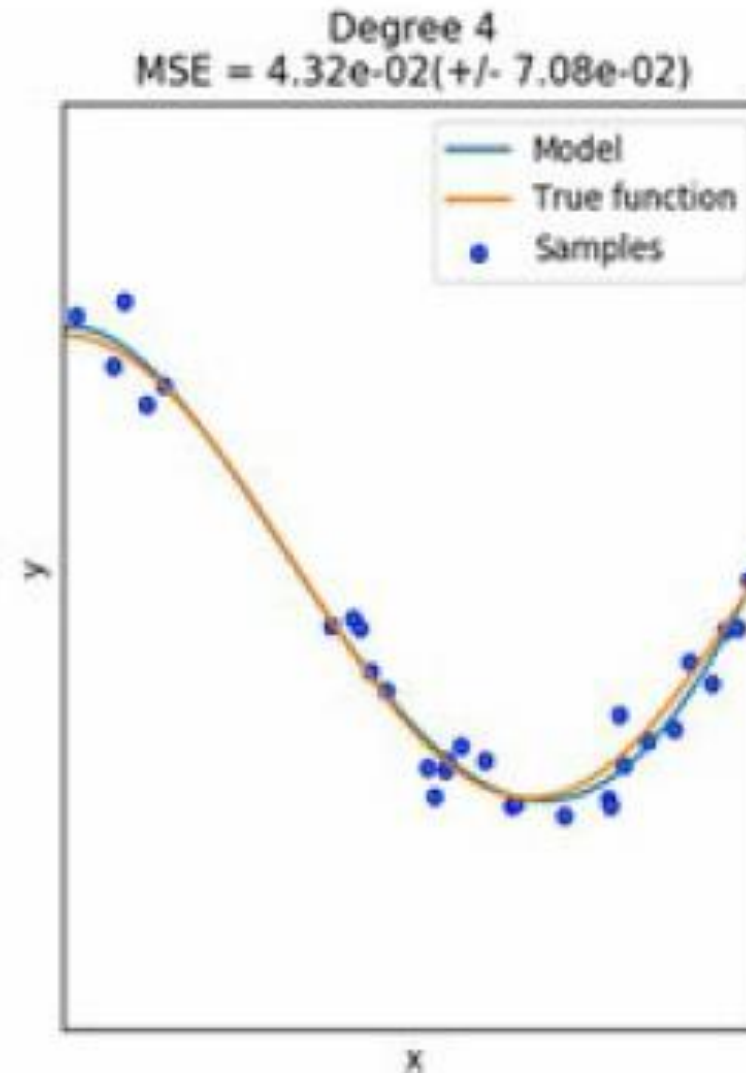
When the model does not understand the underlying pattern on the training dataset, then it is referred to as an under-fitting





## Fit Model

The optimal solution for this problem is a fit model where the model identifies the underlying data pattern. Still, it does not fit through all the training data points that model will be referred to as a fit model.



# Course References

- [1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2021.
- [2] T. Ghosh and S. K. B. Math, *Practical Mathematics for AI and Deep Learning: A Concise yet In-Depth Guide on Fundamentals of Computer Vision, NLP, Complex Deep Neural Networks and Machine Learning (English Edition)*. BPB Publications, 2022.
- [3] M. P. Deisenroth, A. A. Faisal, and C. S. Ong, *Mathematics for Machine Learning*. Cambridge University Press, 2020.
- [4] T. V. Geetha and S. Sendhilkumar, *Machine Learning: Concepts, Techniques and Applications*. CRC Press LLC, 2023.
- [5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2023.
- [6] O. Theobald, *Machine Learning for Absolute Beginners: A Plain English Introduction (Third Edition)*. Scatterplot Press, 2021.

# Accessing Course Resource



**[linkedin.com/in/Samanipour](https://www.linkedin.com/in/Samanipour)**



**[t.me/SamaniGroup](https://t.me/SamaniGroup)**



**[github.com/Samanipour](https://github.com/Samanipour)**