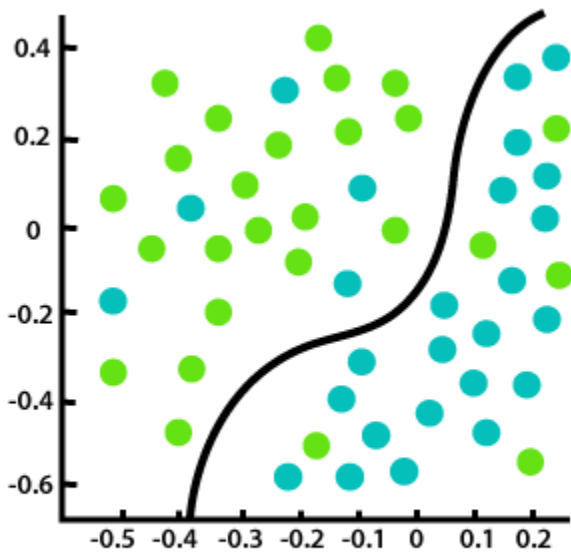


Regression vs. Classification in Machine Learning

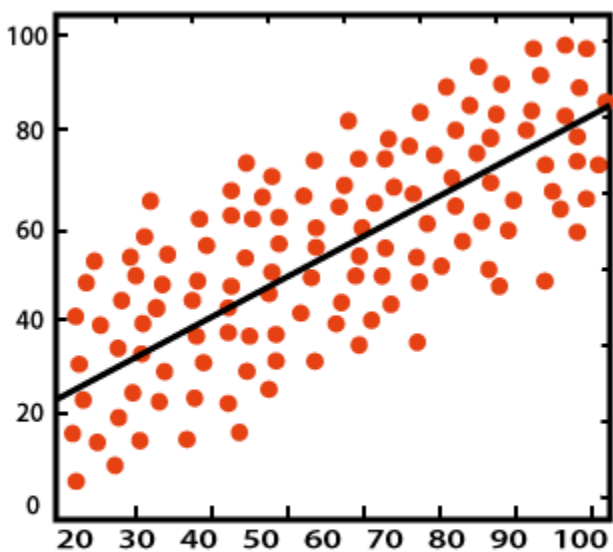
Regression and Classification algorithms are Supervised Learning algorithms. Both the algorithms are used for prediction in Machine learning and work with the labeled datasets. But the difference between both is how they are used for different machine learning problems.

The main difference between Regression and Classification algorithms that Regression algorithms are used to **predict the continuous** values such as price, salary, age, etc. and Classification algorithms are used to **predict/Classify the discrete values** such as Male or Female, True or False, Spam or Not Spam, etc.

Consider the below diagram:



Classification



Regression

Classification:

Classification is a process of finding a function which helps in dividing the dataset into classes based on different parameters. In Classification, a computer program is trained on the training dataset and based on that training, it categorizes the data into different classes.

The task of the classification algorithm is to find the mapping function to map the input(x) to the discrete output(y).

Example: The best example to understand the Classification problem is Email Spam Detection. The model is trained on the basis of millions of emails on different parameters, and whenever it receives a new email, it identifies whether the email is spam or not. If the email is spam, then it is moved to the Spam folder.

Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the following types:

- Logistic Regression
- K-Nearest Neighbours
- Support Vector Machines
- Kernel SVM
- Naïve Bayes
- Decision Tree Classification
- Random Forest Classification

Regression:

Regression is a process of finding the correlations between dependent and independent variables. It helps in predicting the continuous variables such as prediction of **Market Trends**, prediction of House prices, etc.

The task of the Regression algorithm is to find the mapping function to map the input variable(x) to the continuous output variable(y).

Example: Suppose we want to do weather forecasting, so for this, we will use the Regression algorithm. In weather prediction, the model is trained on the past data, and once the training is completed, it can easily predict the weather for future days.

Types of Regression Algorithm:

- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Regression

- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression

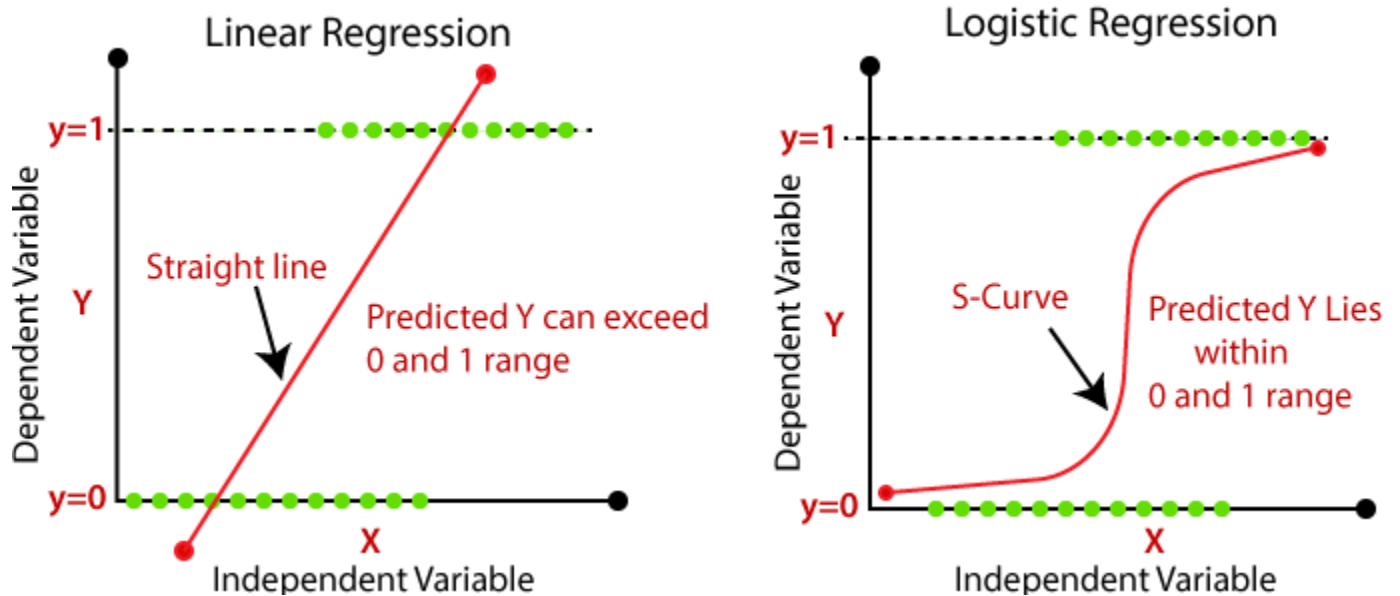
Difference between Regression and Classification

Regression Algorithm	Classification Algorithm
In Regression, the output variable must be of continuous nature or real value.	In Classification, the output variable must be a discrete value.
The task of the regression algorithm is to map the input value (x) with the continuous output variable(y).	The task of the classification algorithm is to map the input value(x) with the discrete output variable(y).
Regression Algorithms are used with continuous data.	Classification Algorithms are used with discrete data.
In Regression, we try to find the best fit line, which can predict the output more accurately.	In Classification, we try to find the decision boundary, which can divide the dataset into different classes.
Regression algorithms can be used to solve the regression problems such as Weather Prediction, House price prediction, etc.	Classification Algorithms can be used to solve classification problems such as Identification of spam emails, Speech Recognition, Identification of cancer cells, etc.
The regression Algorithm can be further divided into Linear and Non-linear Regression.	The Classification algorithms can be divided into Binary Classifier and Multi-class Classifier.

Linear Regression vs Logistic Regression

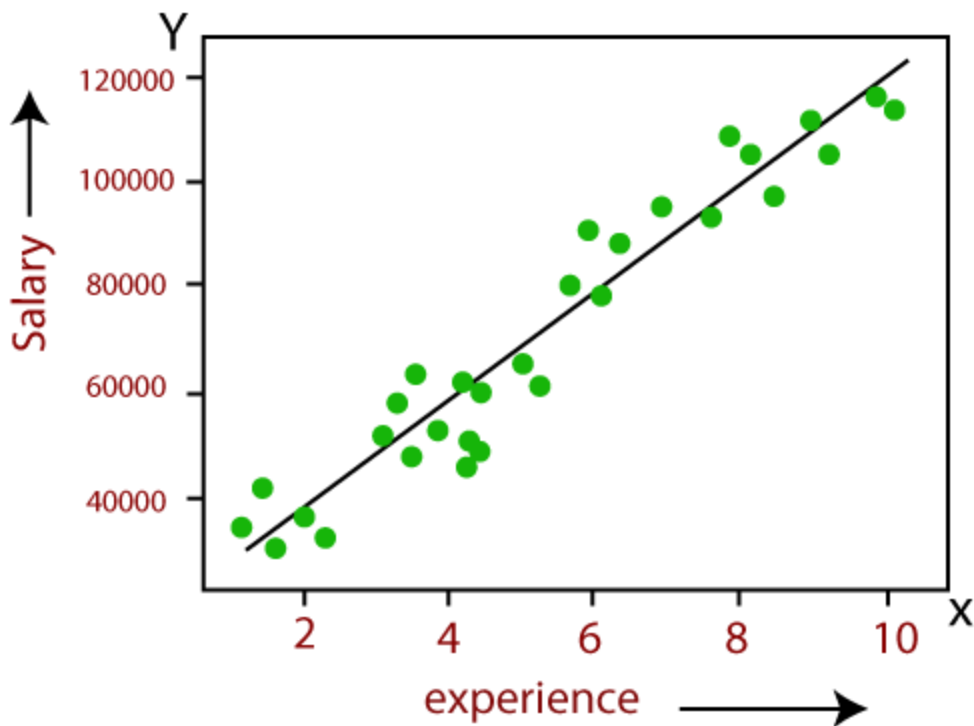
Linear Regression and Logistic Regression are the two famous Machine Learning Algorithms which come under supervised learning technique. Since both the algorithms are of supervised in nature hence these algorithms use labeled dataset to make the predictions. But the main difference between them is how they are being used. The Linear

Regression is used for solving Regression problems whereas Logistic Regression is used for solving the Classification problems. The description of both the algorithms is given below along with difference table.



Linear Regression:

- Linear Regression is one of the most simple Machine learning algorithm that comes under Supervised Learning technique and used for solving regression problems.
- It is used for predicting the continuous dependent variable with the help of independent variables.
- The goal of the Linear regression is to find the best fit line that can accurately predict the output for the continuous dependent variable.
- If single independent variable is used for prediction then it is called Simple Linear Regression and if there are more than two independent variables then such regression is called as Multiple Linear Regression.
- By finding the best fit line, algorithm establish the relationship between dependent variable and independent variable. And the relationship should be of linear nature.
- The output for Linear regression should only be the continuous values such as price, age, salary, etc. The relationship between the dependent variable and independent variable can be shown in below image:



In above image the dependent variable is on Y-axis (salary) and independent variable is on x-axis(experience). The regression line can be written as:

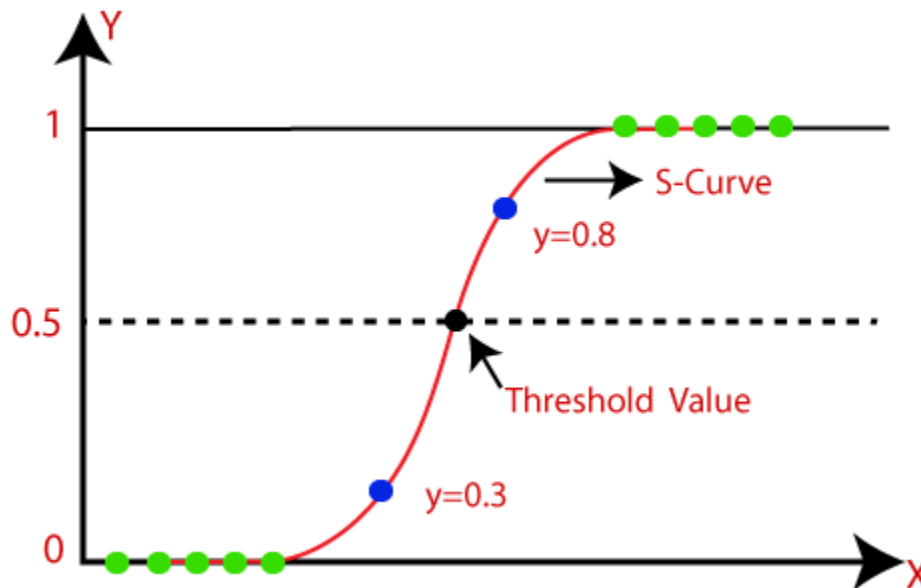
$$y = a_0 + a_1x + \varepsilon$$

Where, a_0 and a_1 are the coefficients and ε is the error term.

Logistic Regression:

- Logistic regression is one of the most popular Machine learning algorithm that comes under Supervised Learning techniques.
- It can be used for Classification as well as for Regression problems, but mainly used for Classification problems.
- Logistic regression is used to predict the categorical dependent variable with the help of independent variables.
- The output of Logistic Regression problem can be only between the 0 and 1.
- Logistic regression can be used where the probabilities between two classes is required. Such as whether it will rain today or not, either 0 or 1, true or false etc.

- Logistic regression is based on the concept of Maximum Likelihood estimation. According to this estimation, the observed data should be most probable.
- In logistic regression, we pass the weighted sum of inputs through an activation function that can map values in between 0 and 1. Such activation function is known as **sigmoid function** and the curve obtained is called as sigmoid curve or S-curve. Consider the below image:



- The equation for logistic regression is:

$$\log \left[\frac{y}{1-y} \right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

Difference between Linear Regression and Logistic Regression:

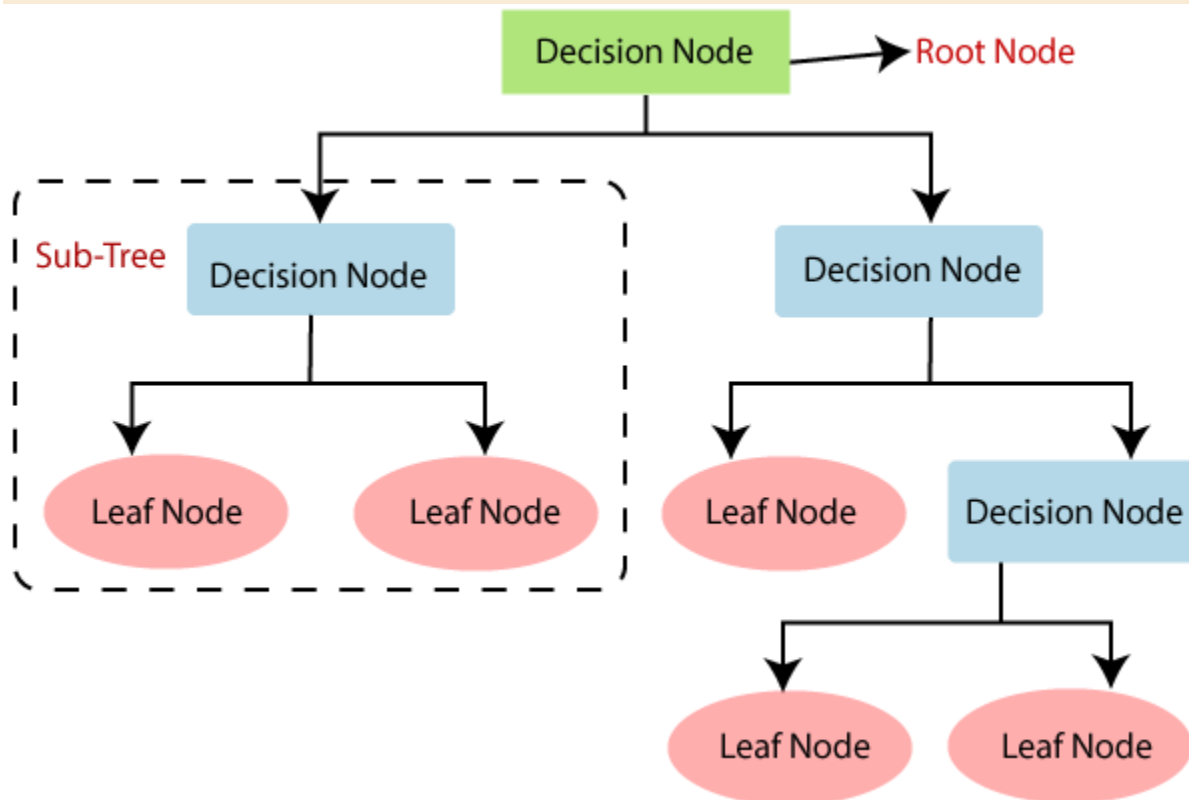
Linear Regression	Logistic Regression
Linear regression is used to predict the continuous dependent variable using a given set of independent variables.	Logistic Regression is used to predict the categorical dependent variable using a given set of independent variables.
Linear Regression is used for solving Regression problem.	Logistic regression is used for solving Classification problems.
In Linear regression, we predict the value of continuous variables.	In logistic Regression, we predict the values of categorical variables.
In linear regression, we find the best fit line, by which we can easily predict the output.	In Logistic Regression, we find the S-curve by which we can classify the samples.
Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for estimation of accuracy.
The output for Linear Regression must be a continuous value, such as price, age, etc.	The output of Logistic Regression must be a Categorical value such as 0 or 1, Yes or No, etc.
In Linear regression, it is required that relationship between dependent variable and independent variable must be linear.	In Logistic regression, it is not required to have the linear relationship between the dependent and independent variable.
In linear regression, there may be collinearity between the independent variables.	In logistic regression, there should not be collinearity between the independent variable.

Decision Tree Classification Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset**, **branches represent the decision rules** and **each leaf node represents the outcome**.

- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm**, which stands for **Classification and Regression Tree algorithm**.
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

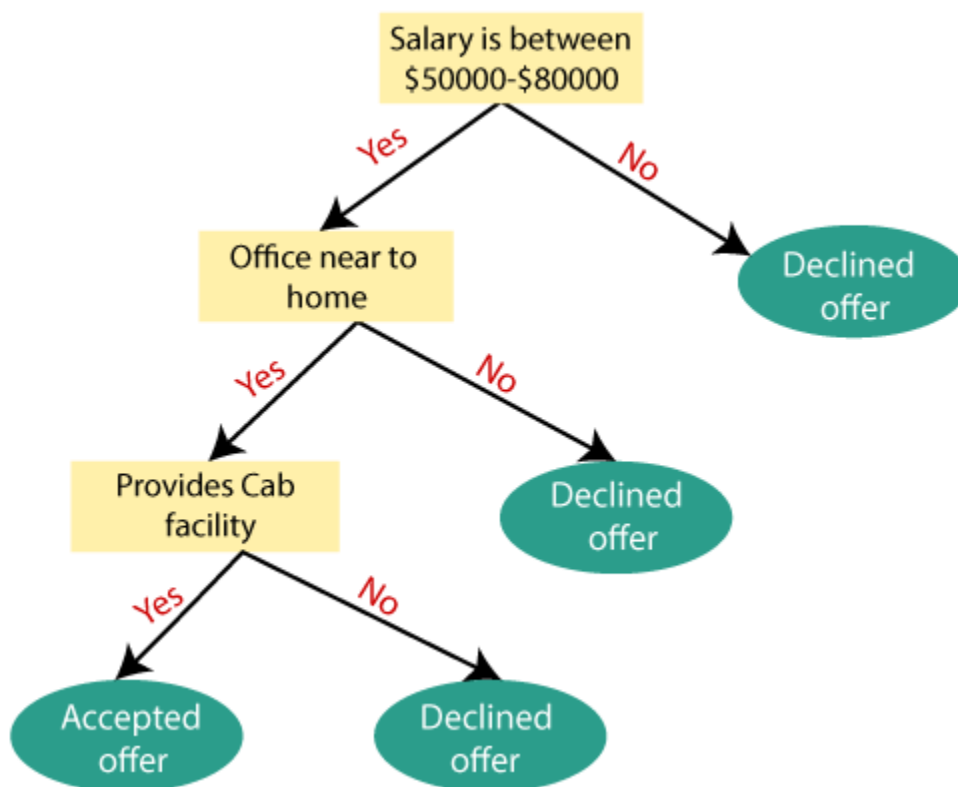
In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

- **Step-1:** Begin the tree with the root node, says S , which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM)**.
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.

- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM**. By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

$$1. \text{ Information Gain} = \text{Entropy}(S) - [(\text{Weighted Avg}) * \text{Entropy}(\text{each feature})]$$

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$\text{Entropy}(s) = -P(\text{yes}) \log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$$

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning.**

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm**.
- For more class labels, the computational complexity of the decision tree may increase.

Python Implementation of Decision Tree

Now we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as [KNN](#), [SVM](#), [LogisticRegression](#), etc.

Steps will also remain the same, which are given below:

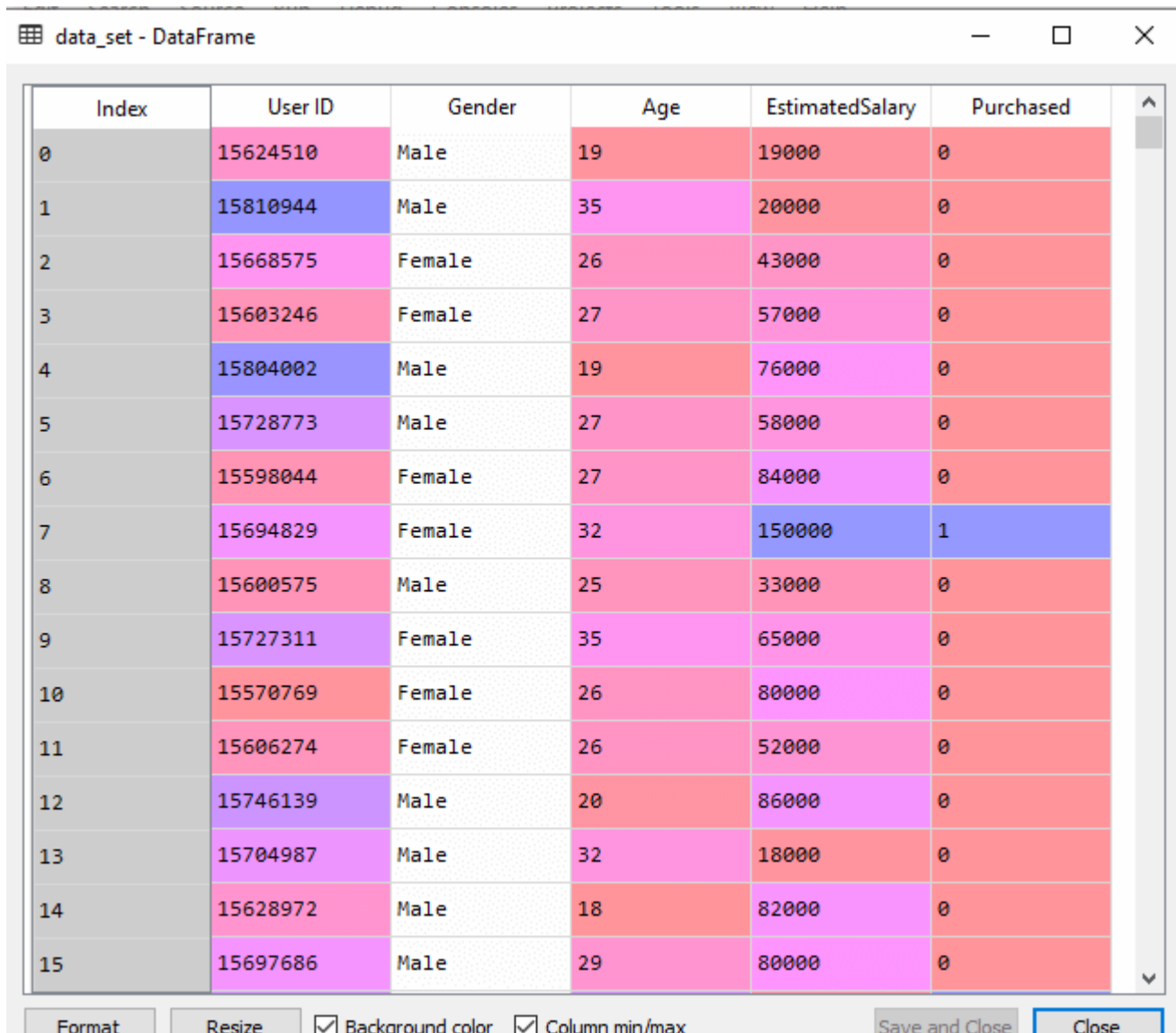
- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- **Visualizing the test set result.**

1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
1. # importing libraries
2. import numpy as nm
3. import matplotlib.pyplot as mtp
4. import pandas as pd
5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')
8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, [2,3]].values
11. y= data_set.iloc[:, 4].values
12.
13. # Splitting the dataset into training and test set.
14. from sklearn.model_selection import train_test_split
15. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
16.
17. #feature Scaling
18. from sklearn.preprocessing import StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)
21. x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

1. #Fitting Decision Tree classifier to the training set
2. From sklearn.tree **import** DecisionTreeClassifier
3. classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
4. classifier.fit(x_train, y_train)

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy'":** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random_state=0":** For generating the random states.

Below is the output for this:

```
Out[8]:
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False,
                        random_state=0, splitter='best')
```

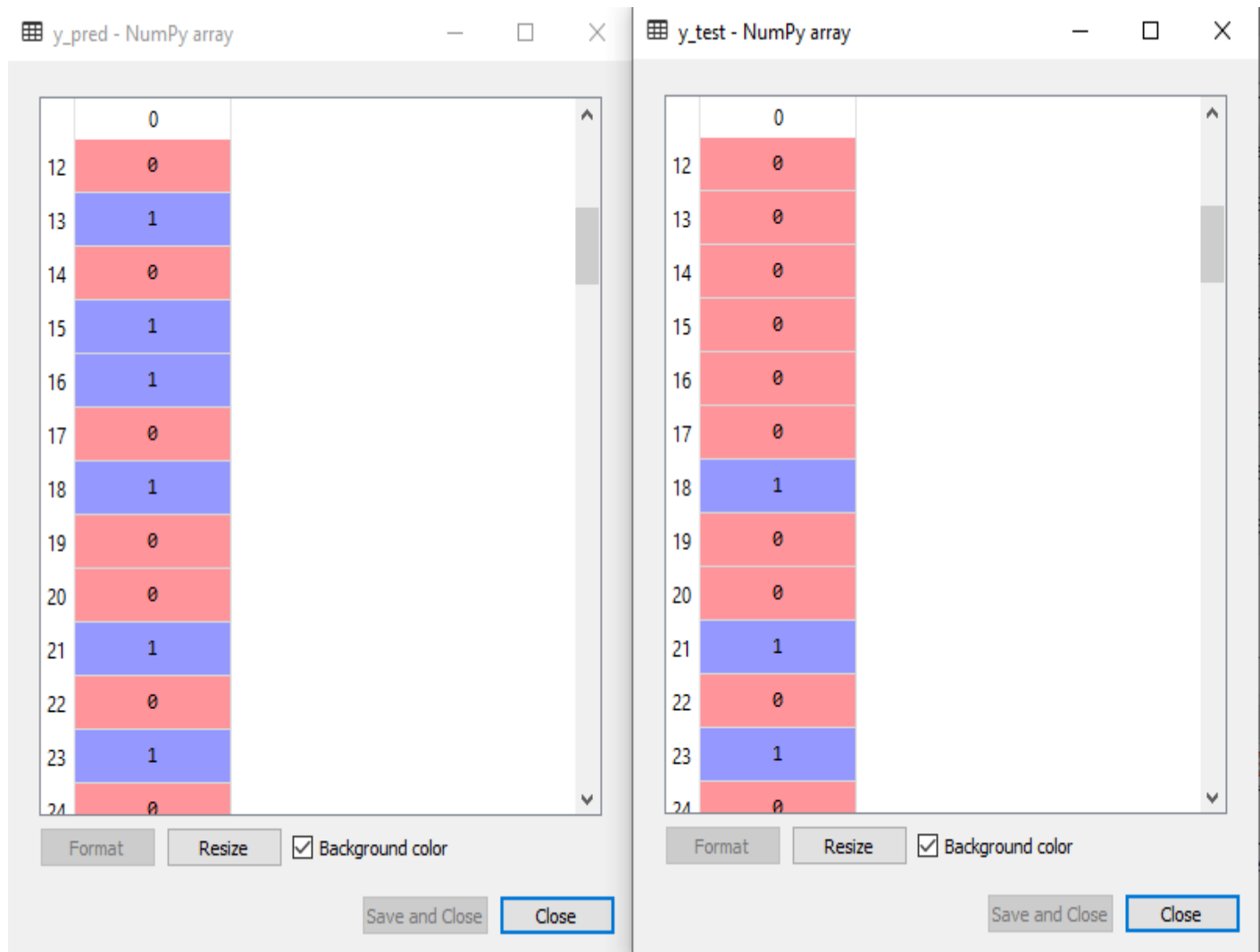
3. Predicting the test result

Now we will predict the test set result. We will create a new prediction vector **y_pred**. Below is the code for it:

1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)

Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

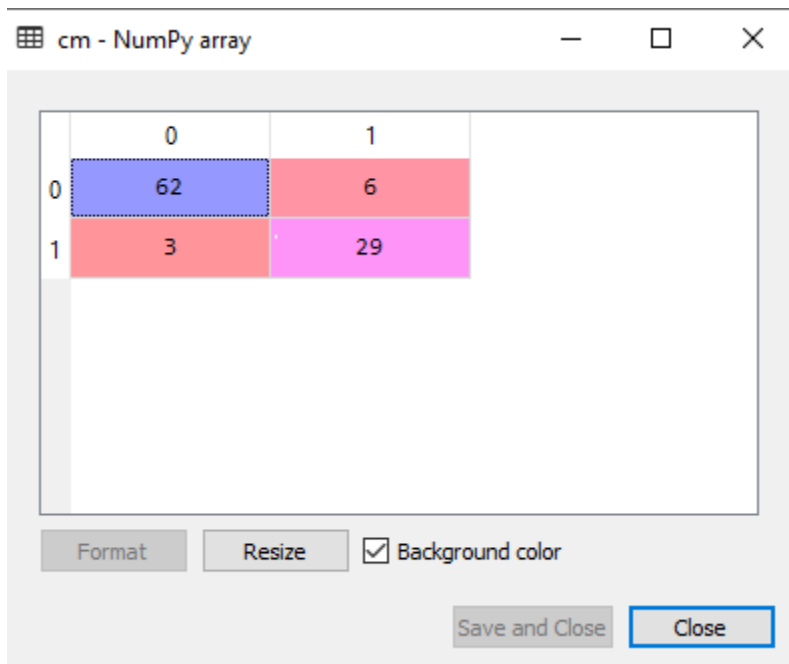


4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)

Output:



In the above output image, we can see the confusion matrix, which has **6+3= 9 incorrect predictions** and **62+29=91 correct predictions**. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.

5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in [Logistic Regression](#). Below is the code for it:

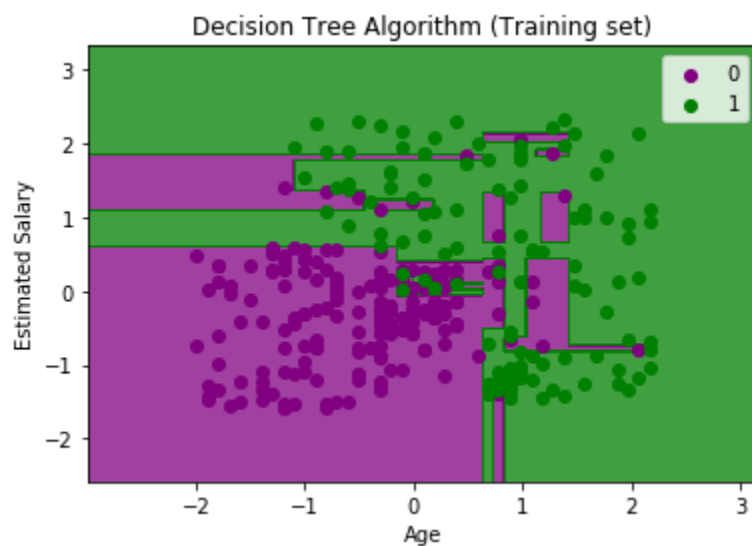
1. #Visulaizing the trianing set result
2. from matplotlib.colors **import** ListedColormap
3. x_set, y_set = x_train, y_train
4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(('purple','green')))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):

```

11. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.             c = ListedColormap(('purple', 'green'))(i), label = j)
13. mtp.title('Decision Tree Algorithm (Training set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

Output:



The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```

1. #Visulaizing the test set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_test, y_test

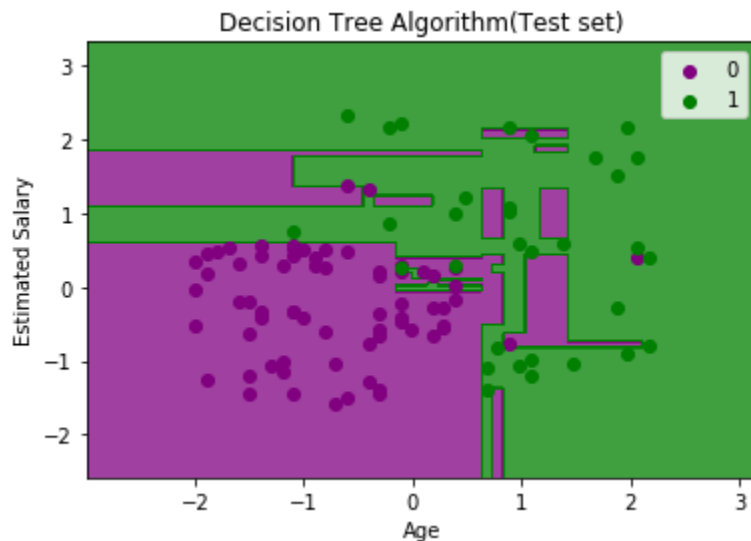
```

```

4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(['purple', 'green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.             c = ListedColormap(['purple', 'green'])(i), label = j)
13. mtp.title('Decision Tree Algorithm(Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

Output:



As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

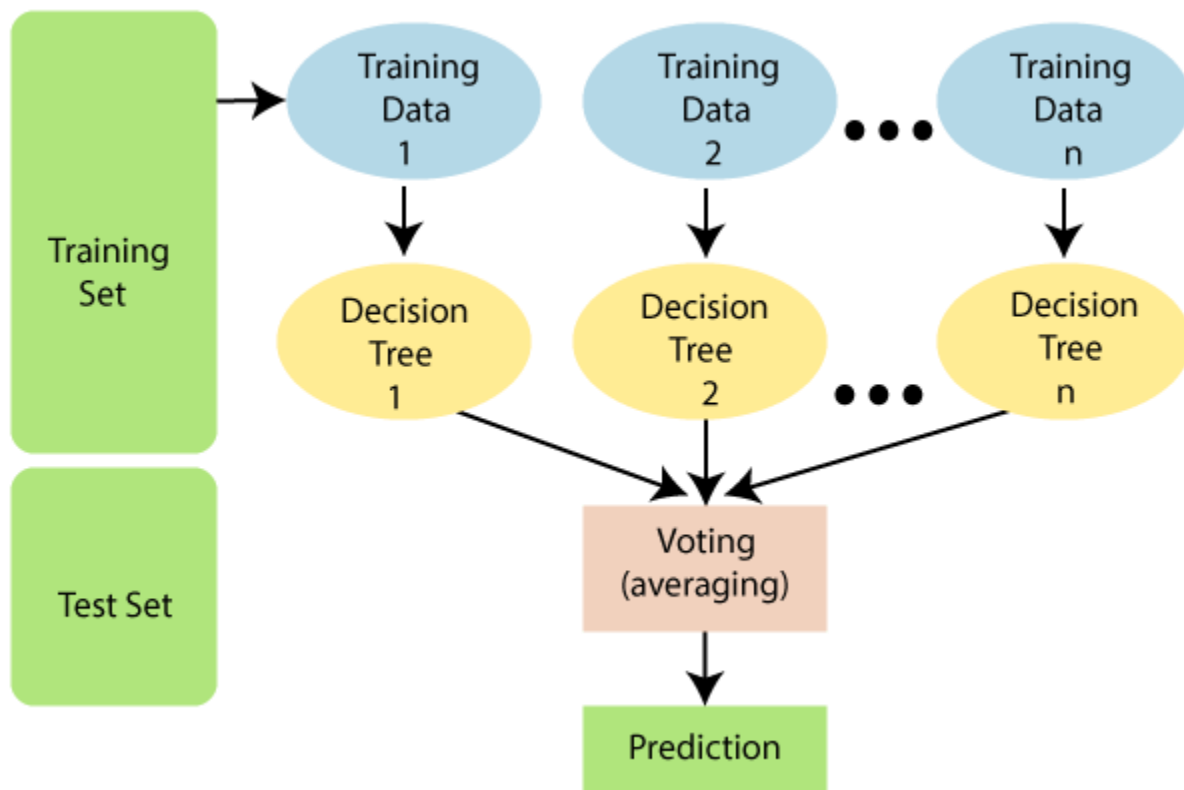
Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning**, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model*.

As the name suggests, "**Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.**" Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



Note: To better understand the Random Forest Algorithm, you should have knowledge of the Decision Tree Algorithm.

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

$\langle = "" \mid j = "" \rangle$

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

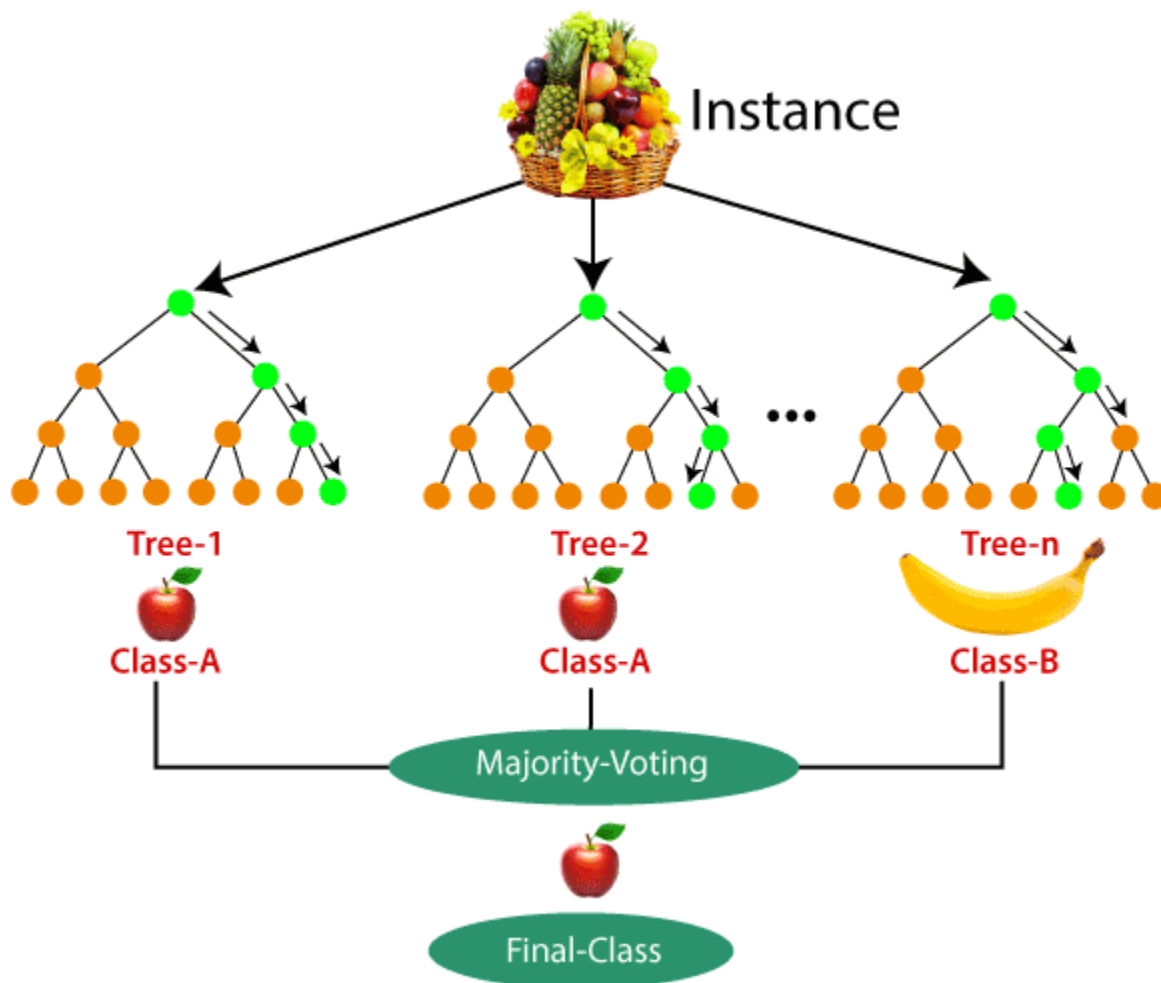
Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.

2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Python Implementation of Random Forest Algorithm

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user_data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as [Decision tree Classifier](#), [KNN](#), [SVM](#), [Logistic Regression](#), etc.

Implementation Steps are given below:

- Data Pre-processing step
- Fitting the Random forest algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)
- Visualizing the test set result.

1.Data Pre-Processing Step:

Below is the code for the pre-processing step:

1. # importing libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd
- 5.
6. #importing datasets
7. data_set= pd.read_csv('user_data.csv')
- 8.
9. #Extracting Independent and dependent Variable
10. x= data_set.iloc[:, [2,3]].values
11. y= data_set.iloc[:, 4].values
- 12.
13. # Splitting the dataset into training and test set.
14. from sklearn.model_selection **import** train_test_split
15. x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
- 16.
17. #feature Scaling
18. from sklearn.preprocessing **import** StandardScaler
19. st_x= StandardScaler()
20. x_train= st_x.fit_transform(x_train)
21. x_test= st_x.transform(x_test)

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0

Format Resize ☒ Background color ☐ Column min/max Save and Close Close

2. Fitting the Random Forest algorithm to the training set:

Now we will fit the Random forest algorithm to the training set. To fit it, we will import the **RandomForestClassifier** class from the **sklearn.ensemble** library. The code is given below:

1. #Fitting Decision Tree classifier to the training set
2. from sklearn.ensemble **import** RandomForestClassifier
3. classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
4. classifier.fit(x_train, y_train)

In the above code, the classifier object takes below parameters:

- **n_estimators**= The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
- **criterion**= It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

Output:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                        max_depth=None,                                max_features='auto',
max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10,
                        n_jobs=None, oob_score=False, random_state=None,
                        verbose=0, warm_start=False)
```

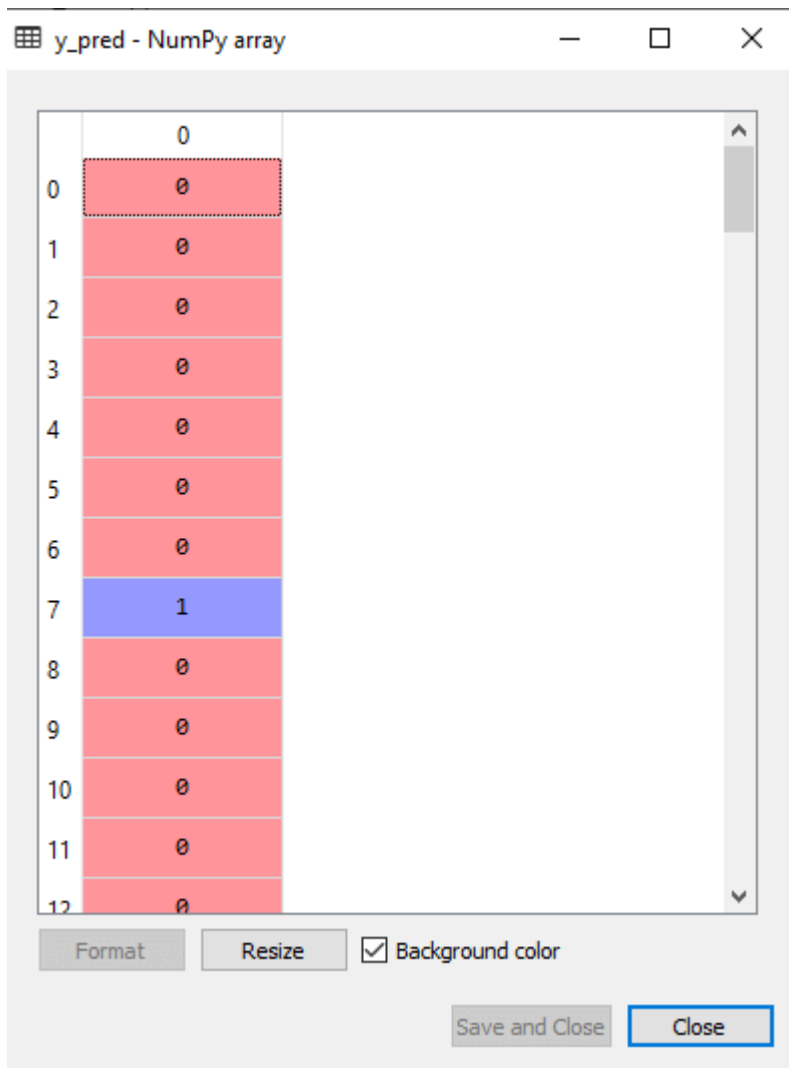
3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector y_pred. Below is the code for it:

1. #Predicting the test set result
2. y_pred= classifier.predict(x_test)

Output:

The prediction vector is given as:



By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier.

4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion_matrix
3. cm= confusion_matrix(y_test, y_pred)

Output:



As we can see in the above matrix, there are **4+4= 8 incorrect predictions** and **64+28= 92 correct predictions**.

5. Visualizing the training Set result

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the Random forest classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in [Logistic Regression](#). Below is the code for it:

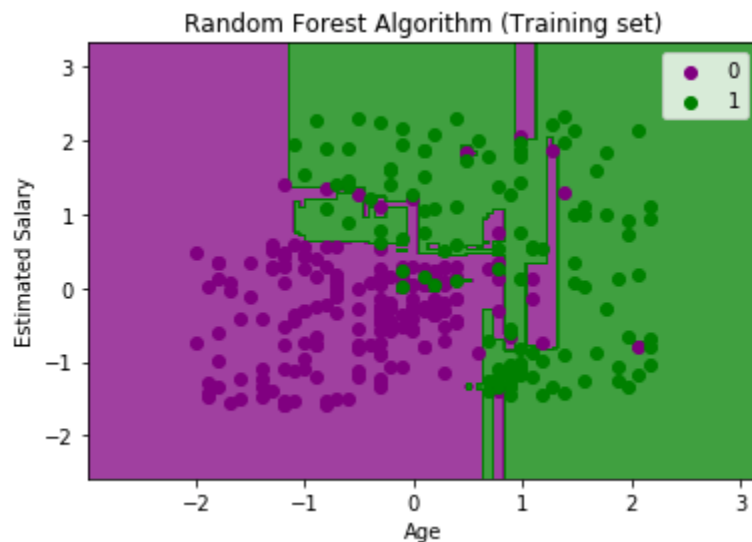
1. from matplotlib.colors **import** ListedColormap
2. x_set, y_set = x_train, y_train
3. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
4. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
5. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
6. alpha = 0.75, cmap = ListedColormap(('purple','green')))
7. mtp.xlim(x1.min(), x1.max())
8. mtp.ylim(x2.min(), x2.max())
9. **for** i, j in enumerate(nm.unique(y_set)):
10. mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
11. c = ListedColormap(('purple', 'green'))(i), label = j)

```

12. mtp.title('Random Forest Algorithm (Training set)')
13. mtp.xlabel('Age')
14. mtp.ylabel('Estimated Salary')
15. mtp.legend()
16. mtp.show()

```

Output:



The above image is the visualization result for the Random Forest classifier working with the training set result. It is very much similar to the Decision tree classifier. Each data point corresponds to each user of the user_data, and the purple and green regions are the prediction regions. The purple region is classified for the users who did not purchase the SUV car, and the green region is for the users who purchased the SUV.

So, in the Random Forest classifier, we have taken 10 trees that have predicted Yes or NO for the Purchased variable. The classifier took the majority of the predictions and provided the result.

6. Visualizing the test set result

Now we will visualize the test set result. Below is the code for it:

```

1. #Visulaizing the test set result
2. from matplotlib.colors import ListedColormap
3. x_set, y_set = x_test, y_test

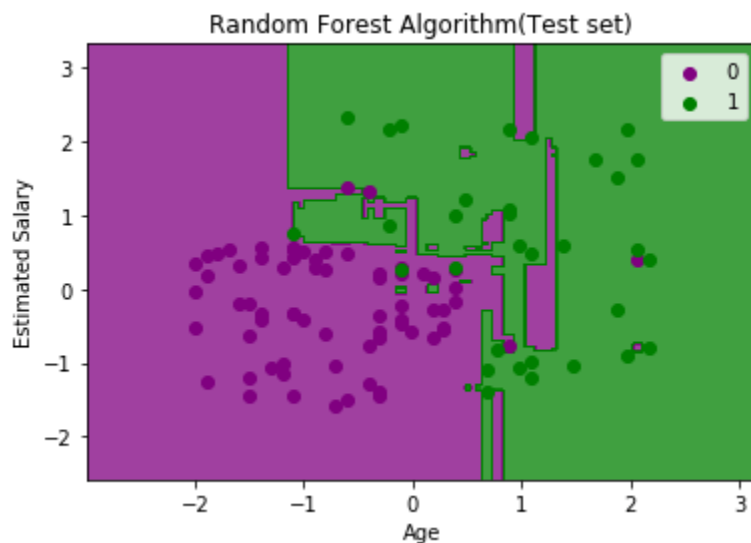
```

```

4. x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
5. nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
6. mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
7. alpha = 0.75, cmap = ListedColormap(['purple', 'green' )))
8. mtp.xlim(x1.min(), x1.max())
9. mtp.ylim(x2.min(), x2.max())
10. for i, j in enumerate(nm.unique(y_set)):
11.     mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
12.         c = ListedColormap(['purple', 'green'])(i), label = j)
13. mtp.title('Random Forest Algorithm(Test set)')
14. mtp.xlabel('Age')
15. mtp.ylabel('Estimated Salary')
16. mtp.legend()
17. mtp.show()

```

Output:



The above image is the visualization result for the test set. We can check that there is a minimum number of incorrect predictions (8) without the Overfitting issue. We will get different results by changing the number of trees in the classifier.

Clustering in Machine Learning

Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset. It can be defined as ***"A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group."***

It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behavior, etc., and divides them as per the presence and absence of those similar patterns.

It is an **unsupervised learning** method, hence no supervision is provided to the algorithm, and it deals with the unlabeled dataset.

After applying this clustering technique, each cluster or group is provided with a cluster-ID. ML system can use this id to simplify the processing of large and complex datasets.

The clustering technique is commonly used for **statistical data analysis**.

Note: *Clustering is somewhere similar to the **classification algorithm**, but the difference is the type of dataset that we are using. In classification, we work with the labeled data set, whereas in clustering, we work with the unlabelled dataset.*

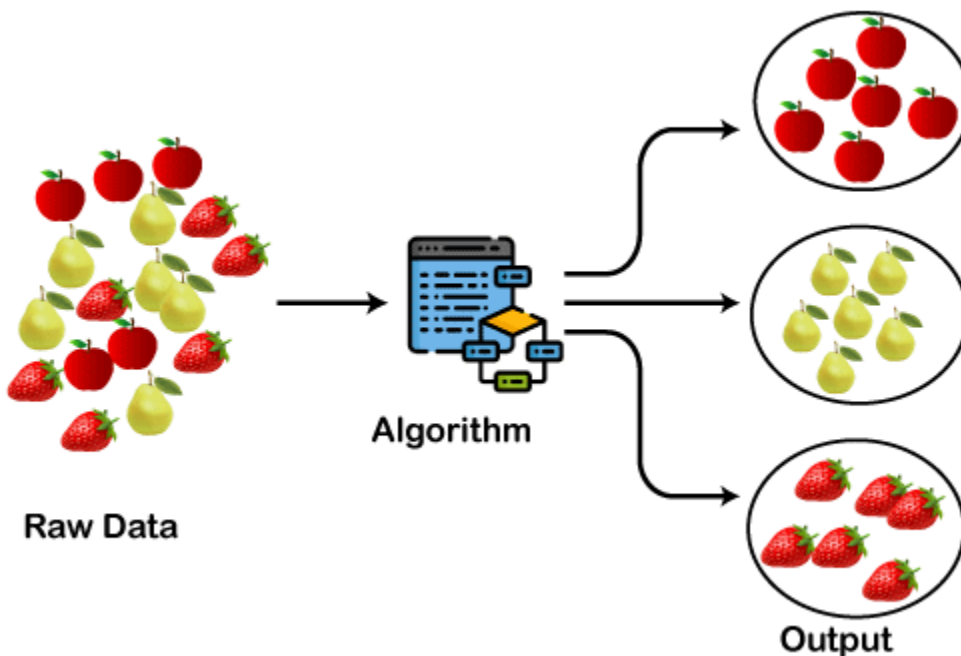
Example: Let's understand the clustering technique with the real-world example of Mall: When we visit any shopping mall, we can observe that the things with similar usage are grouped together. Such as the t-shirts are grouped in one section, and trousers are at other sections, similarly, at vegetable sections, apples, bananas, Mangoes, etc., are grouped in separate sections, so that we can easily find out the things. The clustering technique also works in the same way. Other examples of clustering are grouping documents according to the topic.

The clustering technique can be widely used in various tasks. Some most common uses of this technique are:

- Market Segmentation
- Statistical data analysis
- Social network analysis
- Image segmentation
- Anomaly detection, etc.

Apart from these general usages, it is used by the **Amazon** in its recommendation system to provide the recommendations as per the past search of products. **Netflix** also uses this technique to recommend the movies and web-series to its users as per the watch history.

The below diagram explains the working of the clustering algorithm. We can see the different fruits are divided into several groups with similar properties.



Types of Clustering Methods

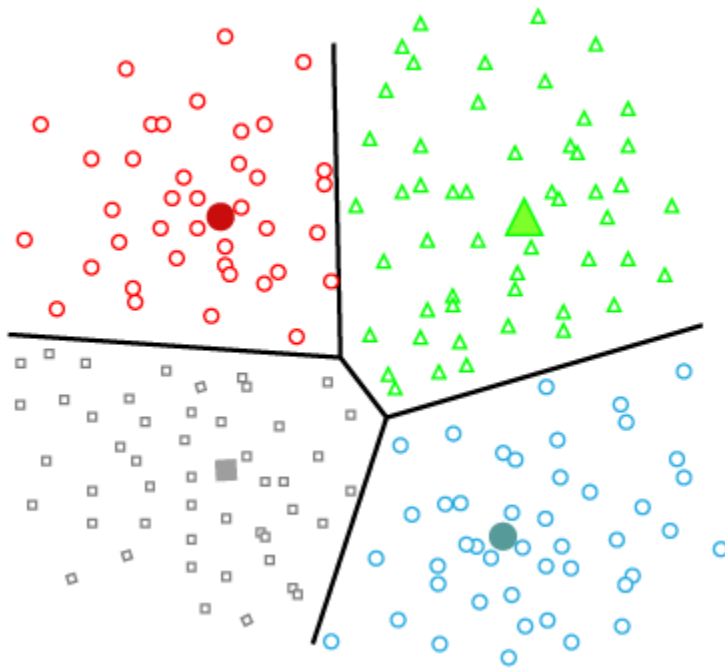
The clustering methods are broadly divided into **Hard clustering** (datapoint belongs to only one group) and **Soft Clustering** (data points can belong to another group also). But there are also other various approaches of Clustering exist. Below are the main clustering methods used in Machine learning:

1. **Partitioning Clustering**
2. **Density-Based Clustering**
3. **Distribution Model-Based Clustering**
4. **Hierarchical Clustering**
5. **Fuzzy Clustering**

Partitioning Clustering

It is a type of clustering that divides the data into non-hierarchical groups. It is also known as the **centroid-based method**. The most common example of partitioning clustering is the **K-Means Clustering algorithm**.

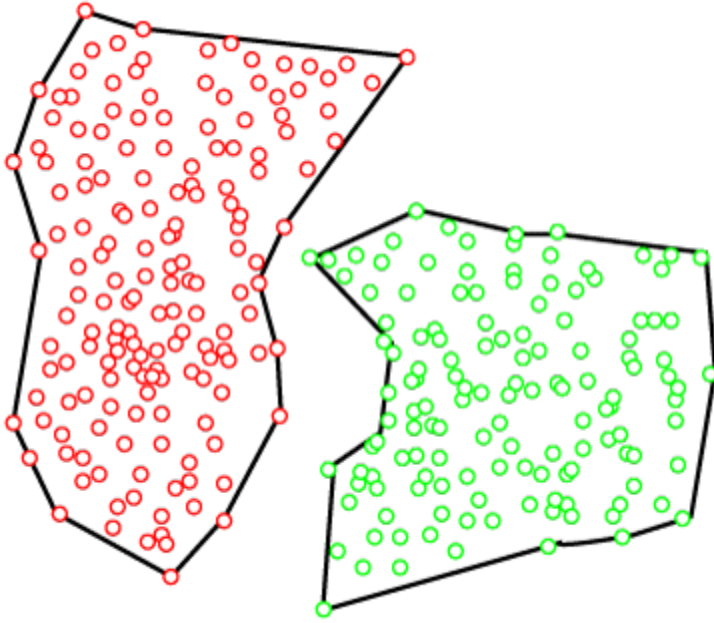
In this type, the dataset is divided into a set of k groups, where K is used to define the number of pre-defined groups. The cluster center is created in such a way that the distance between the data points of one cluster is minimum as compared to another cluster centroid.



Density-Based Clustering

The density-based clustering method connects the highly-dense areas into clusters, and the arbitrarily shaped distributions are formed as long as the dense region can be connected. This algorithm does it by identifying different clusters in the dataset and connects the areas of high densities into clusters. The dense areas in data space are divided from each other by sparser areas.

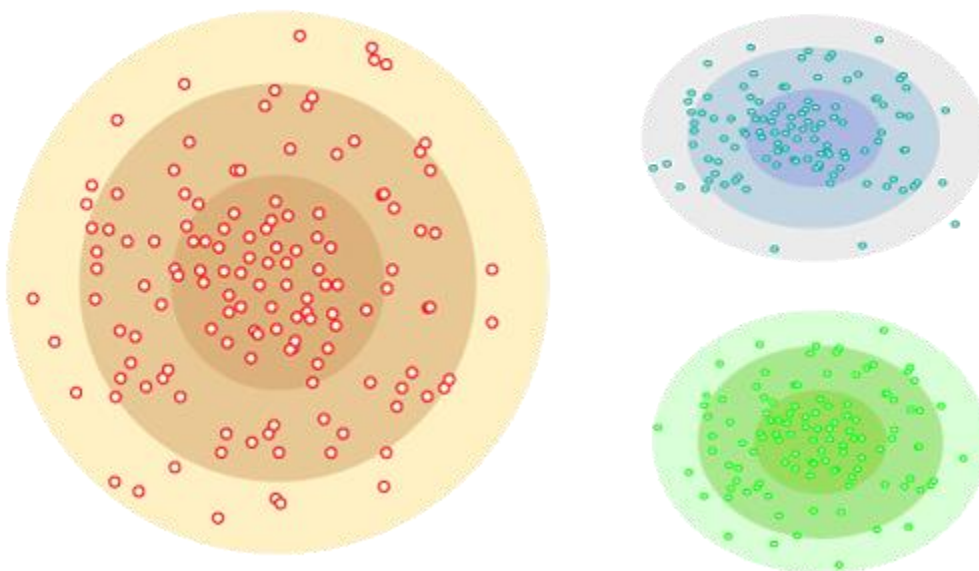
These algorithms can face difficulty in clustering the data points if the dataset has varying densities and high dimensions.



Distribution Model-Based Clustering

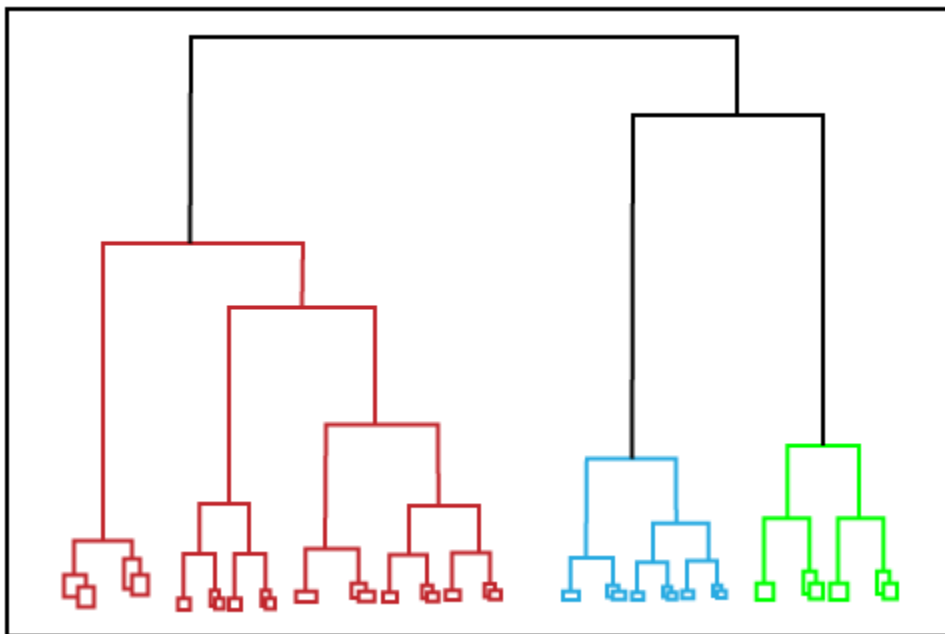
In the distribution model-based clustering method, the data is divided based on the probability of how a dataset belongs to a particular distribution. The grouping is done by assuming some distributions commonly **Gaussian Distribution**.

The example of this type is the **Expectation-Maximization Clustering algorithm** that uses Gaussian Mixture Models (GMM).



Hierarchical Clustering

Hierarchical clustering can be used as an alternative for the partitioned clustering as there is no requirement of pre-specifying the number of clusters to be created. In this technique, the dataset is divided into clusters to create a tree-like structure, which is also called a **dendrogram**. The observations or any number of clusters can be selected by cutting the tree at the correct level. The most common example of this method is the **Agglomerative Hierarchical algorithm**.



Fuzzy Clustering

Fuzzy clustering is a type of soft method in which a data object may belong to more than one group or cluster. Each dataset has a set of membership coefficients, which depend on the degree of membership to be in a cluster. **Fuzzy C-means algorithm** is the example of this type of clustering; it is sometimes also known as the Fuzzy k-means algorithm.

Clustering Algorithms

The Clustering algorithms can be divided based on their models that are explained above. There are different types of clustering algorithms published, but only a few are commonly used. The clustering algorithm is based on the kind of data that we are using. Such as, some algorithms need to guess the number of clusters in the given dataset, whereas some are required to find the minimum distance between the observation of the dataset.

Here we are discussing mainly popular Clustering algorithms that are widely used in machine learning:

1. **K-Means algorithm:** The k-means algorithm is one of the most popular clustering algorithms. It classifies the dataset by dividing the samples into different clusters of equal variances. The number of clusters must be specified in this algorithm. It is fast with fewer computations required, with the linear complexity of $O(n)$.
2. **Mean-shift algorithm:** Mean-shift algorithm tries to find the dense areas in the smooth density of data points. It is an example of a centroid-based model, that works on updating the candidates for centroid to be the center of the points within a given region.
3. **DBSCAN Algorithm:** It stands **for Density-Based Spatial Clustering of Applications with Noise**. It is an example of a density-based model similar to the mean-shift, but with some remarkable advantages. In this algorithm, the areas of high density are separated by the areas of low density. Because of this, the clusters can be found in any arbitrary shape.
4. **Expectation-Maximization Clustering using GMM:** This algorithm can be used as an alternative for the k-means algorithm or for those cases where K-means can be failed. In GMM, it is assumed that the data points are Gaussian distributed.
5. **Agglomerative Hierarchical algorithm:** The Agglomerative hierarchical algorithm performs the bottom-up hierarchical clustering. In this, each data point is treated as a single cluster at the outset and then successively merged. The cluster hierarchy can be represented as a tree-structure.
6. **Affinity Propagation:** It is different from other clustering algorithms as it does not require to specify the number of clusters. In this, each data point sends a message between the pair of data points until convergence. It has $O(N^2T)$ time complexity, which is the main drawback of this algorithm.

Applications of Clustering

Below are some commonly known applications of clustering technique in Machine Learning:

- **In Identification of Cancer Cells:** The clustering algorithms are widely used for the identification of cancerous cells. It divides the cancerous and non-cancerous data sets into different groups.

- **In Search Engines:** Search engines also work on the clustering technique. The search result appears based on the closest object to the search query. It does it by grouping similar data objects in one group that is far from the other dissimilar objects. The accurate result of a query depends on the quality of the clustering algorithm used.
- **Customer Segmentation:** It is used in market research to segment the customers based on their choice and preferences.
- **In Biology:** It is used in the biology stream to classify different species of plants and animals using the image recognition technique.
- **In Land Use:** The clustering technique is used in identifying the area of similar lands use in the GIS database. This can be very useful to find that for what purpose the particular land should be used, that means for which purpose it is more suitable.

Hierarchical Clustering in Machine Learning

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.

In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

The hierarchical clustering technique has two approaches:

1. **Agglomerative:** Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach**.

Why hierarchical clustering?

As we already have other **clustering** algorithms such as **K-Means Clustering**, then why we need hierarchical clustering? So, as we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters,

and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

In this topic, we will discuss the Agglomerative Hierarchical clustering algorithm.

Agglomerative Hierarchical clustering

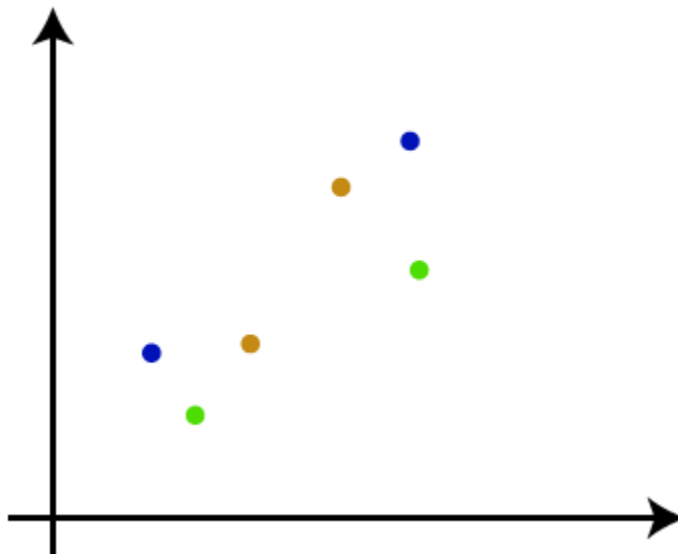
The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.

This hierarchy of clusters is represented in the form of the dendrogram.

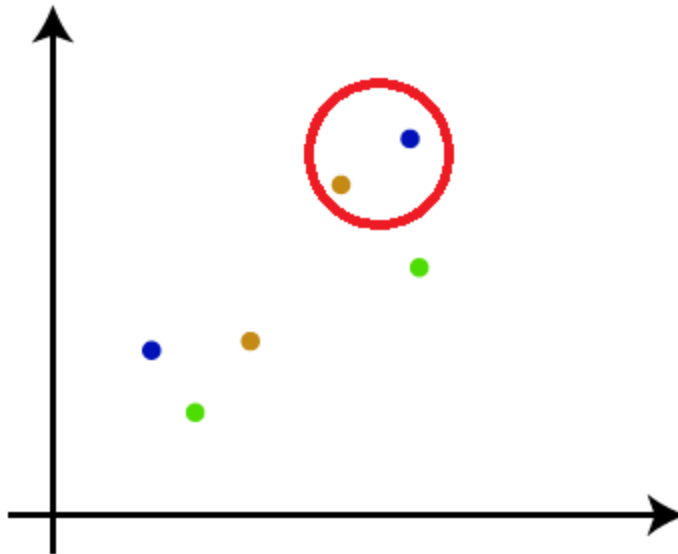
How the Agglomerative Hierarchical clustering Work?

The working of the AHC algorithm can be explained using the below steps:

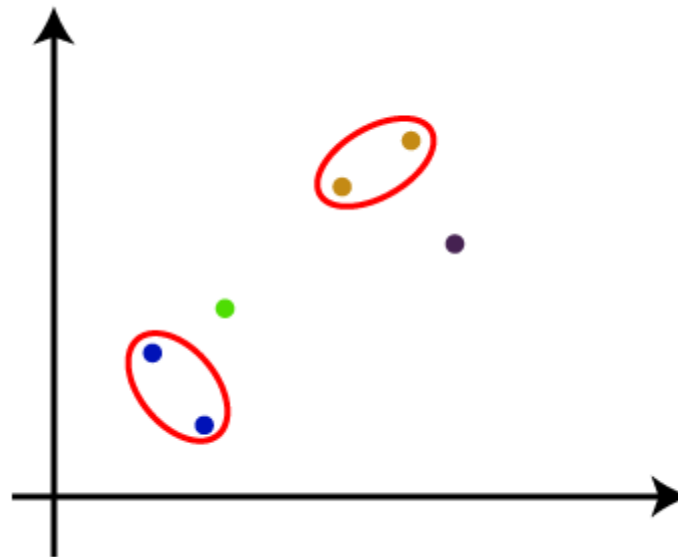
- **Step-1:** Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N .



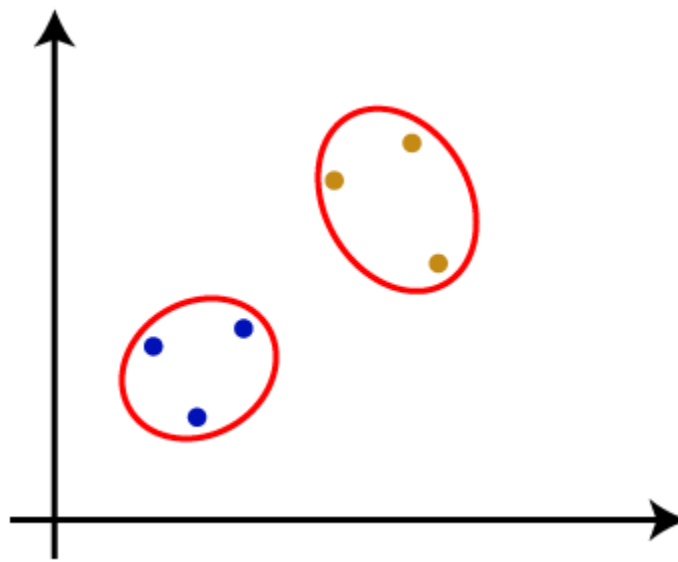
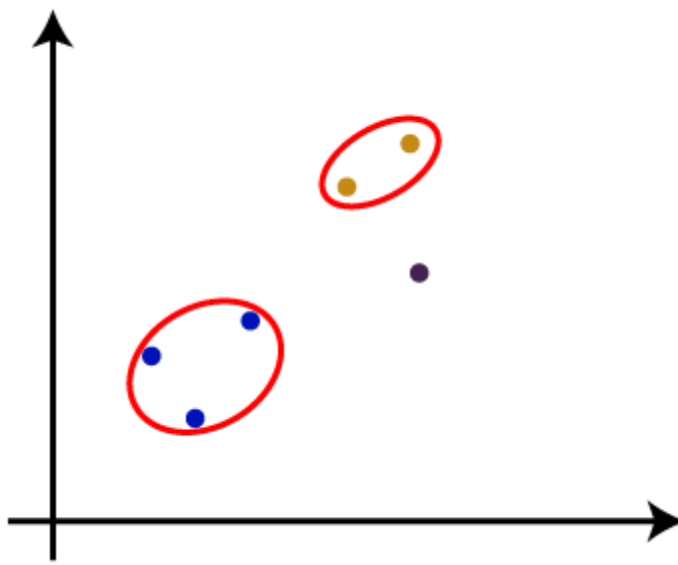
- **Step-2:** Take two closest data points or clusters and merge them to form one cluster. So, there will now be $N-1$ clusters.

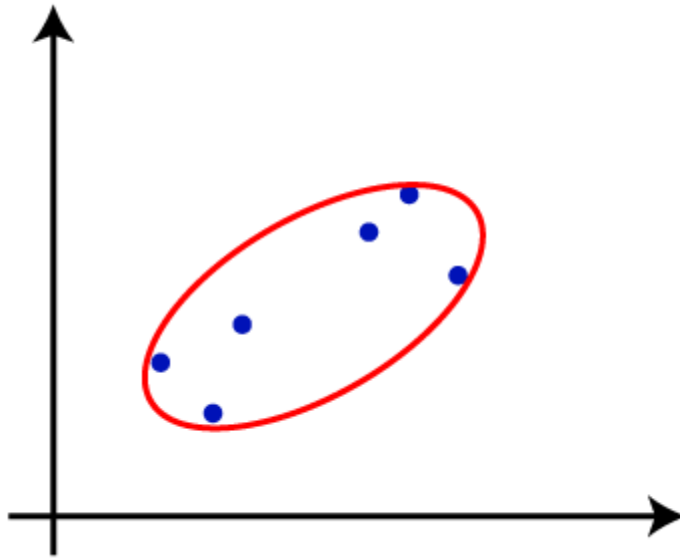


- **Step-3:** Again, take the two closest clusters and merge them together to form one cluster. There will be $N-2$ clusters.



- **Step-4:** Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:





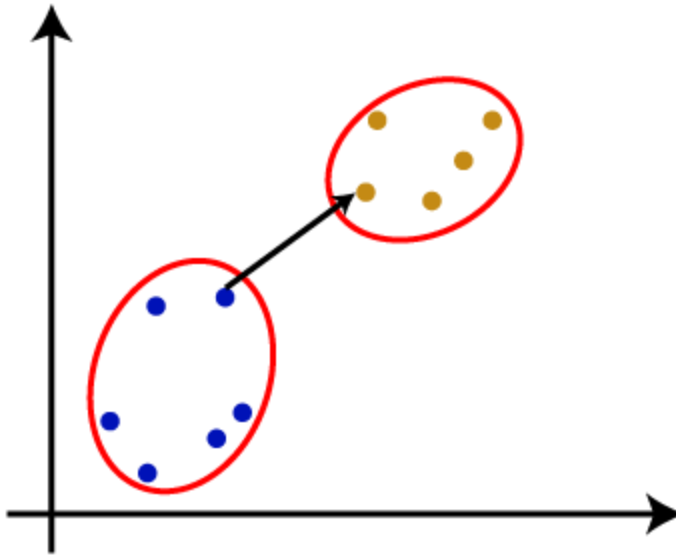
- **Step-5:** Once all the clusters are combined into one big cluster, develop the dendrogram to divide the clusters as per the problem.

***Note:** To better understand hierarchical clustering, it is advised to have a look on k-means clustering*

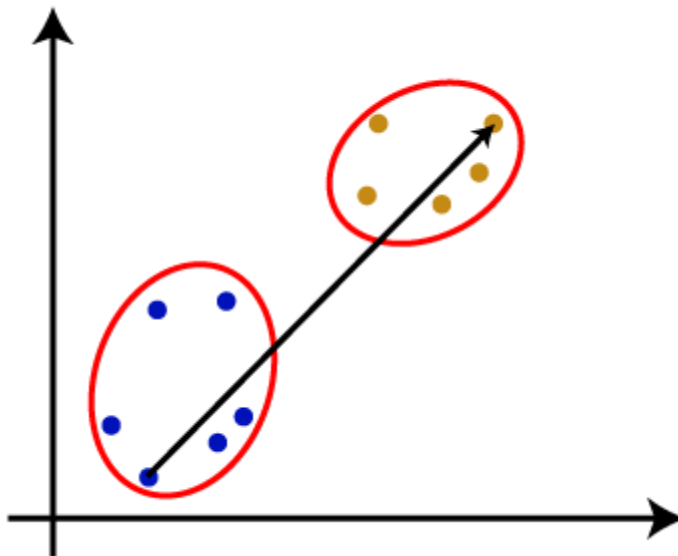
Measure for the distance between two clusters

As we have seen, the **closest distance** between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate the distance between two clusters, and these ways decide the rule for clustering. These measures are called **Linkage methods**. Some of the popular linkage methods are given below:

1. **Single Linkage:** It is the Shortest Distance between the closest points of the clusters.
Consider the below image:

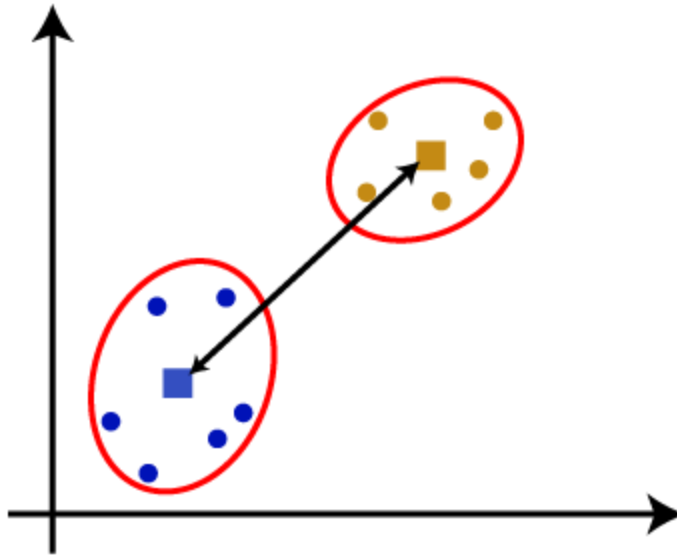


2. **Complete Linkage:** It is the farthest distance between the two points of two different clusters. It is one of the popular linkage methods as it forms tighter clusters than single-linkage.



3. **Average Linkage:** It is the linkage method in which the distance between each pair of datasets is added up and then divided by the total number of datasets to calculate the average distance between two clusters. It is also one of the most popular linkage methods.

4. **Centroid Linkage:** It is the linkage method in which the distance between the centroid of the clusters is calculated. Consider the below image:

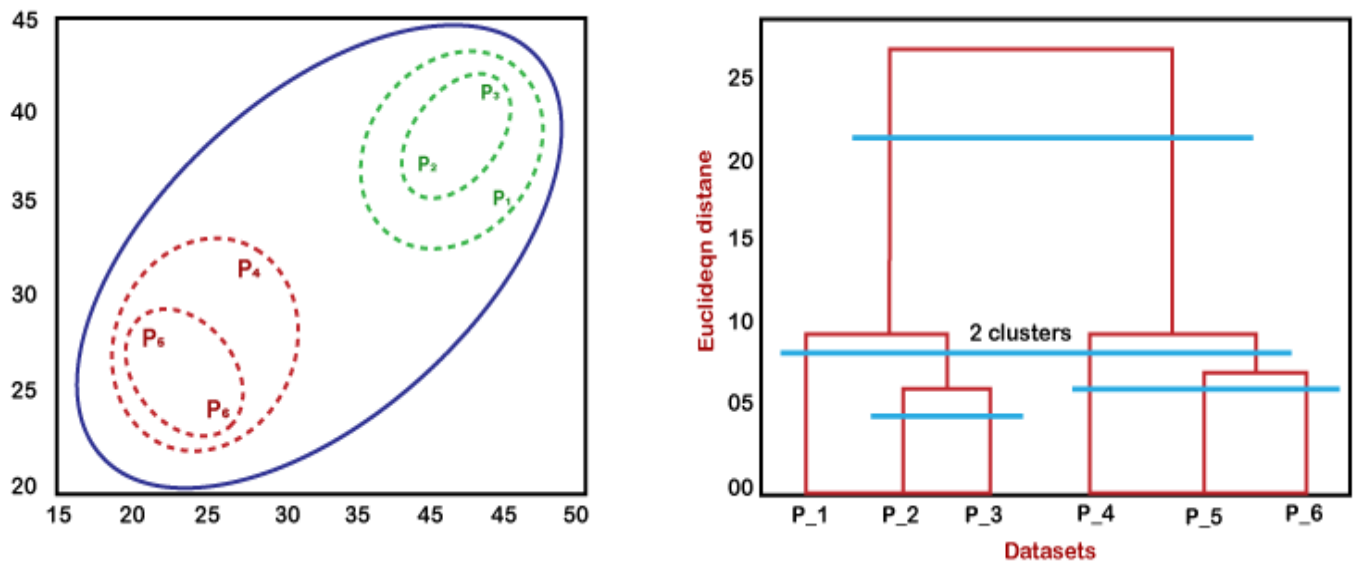


From the above-given approaches, we can apply any of them according to the type of problem or business requirement.

Working of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.

The working of the dendrogram can be explained using the below diagram:



In the above diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.

- As we have discussed above, firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The height is decided according to the Euclidean distance between the data points.
- In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
- Again, two new dendrograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
- At last, the final dendrogram is created that combines all the data points together.

We can cut the dendrogram tree structure at any level as per our requirement.

Python Implementation of Agglomerative Hierarchical Clustering

Now we will see the practical implementation of the agglomerative hierarchical clustering algorithm using Python. To implement this, we will use the same dataset problem that we

have used in the previous topic of K-means clustering so that we can compare both concepts easily.

The dataset is containing the information of customers that have visited a mall for shopping. So, the mall owner wants to find some patterns or some particular behavior of his customers using the dataset information.

Steps for implementation of AHC using Python:

The steps for implementation will be the same as the k-means clustering, except for some changes such as the method to find the number of clusters. Below are the steps:

1. **Data Pre-processing**
2. **Finding the optimal number of clusters using the Dendrogram**
3. **Training the hierarchical clustering model**
4. **Visualizing the clusters**

Data Pre-processing Steps:

In this step, we will import the libraries and datasets for our model.

○ **Importing the libraries**

1. # Importing the libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd

The above lines of code are used to import the libraries to perform specific tasks, such as **numpy** for the Mathematical operations, **matplotlib** for drawing the graphs or scatter plot, and **pandas** for importing the dataset.

○ **Importing the dataset**

1. # Importing the dataset
2. dataset = pd.read_csv('Mall_Customers_data.csv')

As discussed above, we have imported the same dataset of **Mall_Customers_data.csv**, as we did in k-means clustering. Consider the below output:

Index	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13

Format Resize ☒ Background color ☐ Column min/max Save and Close Close

- **Extracting the matrix of features**

Here we will extract only the matrix of features as we don't have any further information about the dependent variable. Code is given below:

1. `x = dataset.iloc[:, [3, 4]].values`

Here we have extracted only 3 and 4 columns as we will use a 2D plot to see the clusters. So, we are considering the Annual income and spending score as the matrix of features.

Step-2: Finding the optimal number of clusters using the Dendrogram

Now we will find the optimal number of clusters using the Dendrogram for our model. For this, we are going to use **scipy** library as it provides a function that will directly return the dendrogram for our code. Consider the below lines of code:

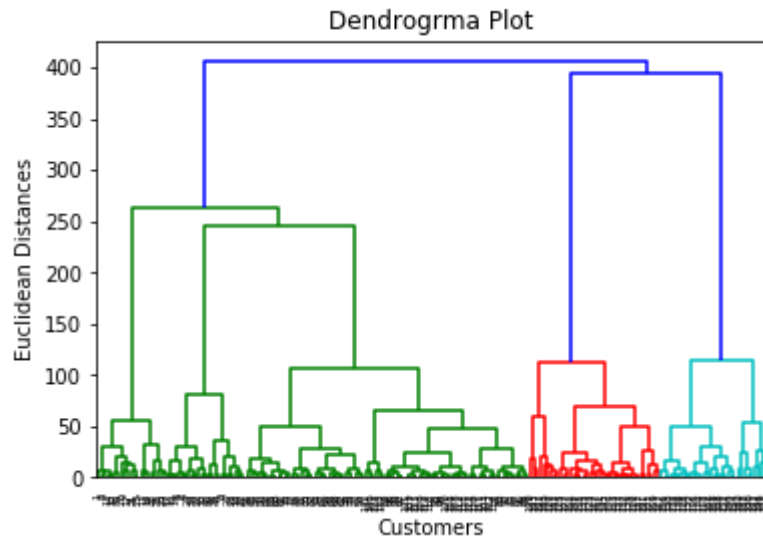
1. #Finding the optimal number of clusters using the dendrogram
2. **import** scipy.cluster.hierarchy as shc
3. dendro = shc.dendrogram(shc.linkage(x, method="ward"))
4. mtp.title("Dendrogram Plot")
5. mtp.ylabel("Euclidean Distances")
6. mtp.xlabel("Customers")
7. mtp.show()

In the above lines of code, we have imported the **hierarchy** module of scipy library. This module provides us a method **shc.dendrogram()**, which takes the **linkage()** as a parameter. The linkage function is used to define the distance between two clusters, so here we have passed the x(matrix of features), and method "**ward**," the popular method of linkage in hierarchical clustering.

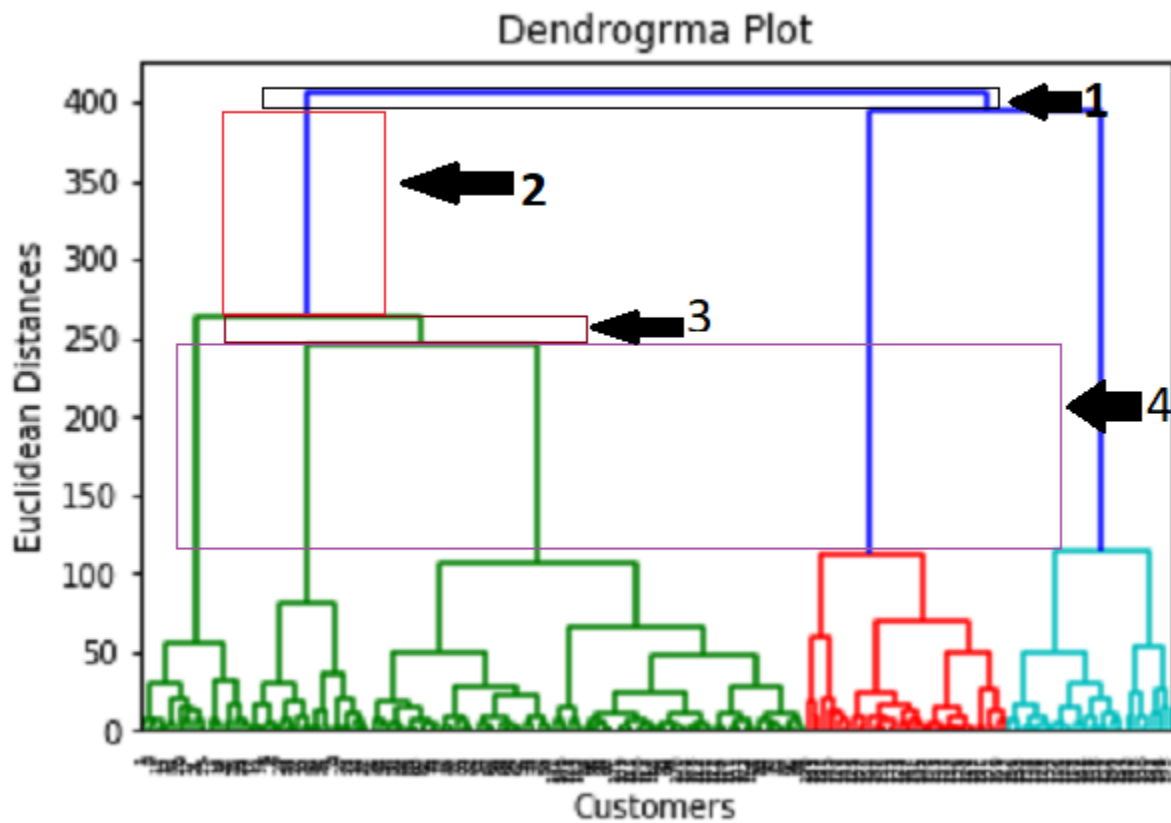
The remaining lines of code are to describe the labels for the dendrogram plot.

Output:

By executing the above lines of code, we will get the below output:



Using this Dendrogram, we will now determine the optimal number of clusters for our model. For this, we will find the **maximum vertical distance** that does not cut any horizontal bar. Consider the below diagram:



In the above diagram, we have shown the vertical distances that are not cutting their horizontal bars. As we can visualize, the 4th distance is looking the maximum, so according to this, **the number of clusters will be 5**(the vertical lines in this range). We can also take the 2nd number as it approximately equals the 4th distance, but we will consider the 5 clusters because the same we calculated in the K-means algorithm.

So, the optimal number of clusters will be 5, and we will train the model in the next step, using the same.

Step-3: Training the hierarchical clustering model

As we know the required optimal number of clusters, we can now train our model. The code is given below:

1. #training the hierarchical model on dataset
2. from sklearn.cluster **import** AgglomerativeClustering
3. hc= AgglomerativeClustering(n_clusters=**5**, affinity='euclidean', linkage='ward')
4. y_pred= hc.fit_predict(x)

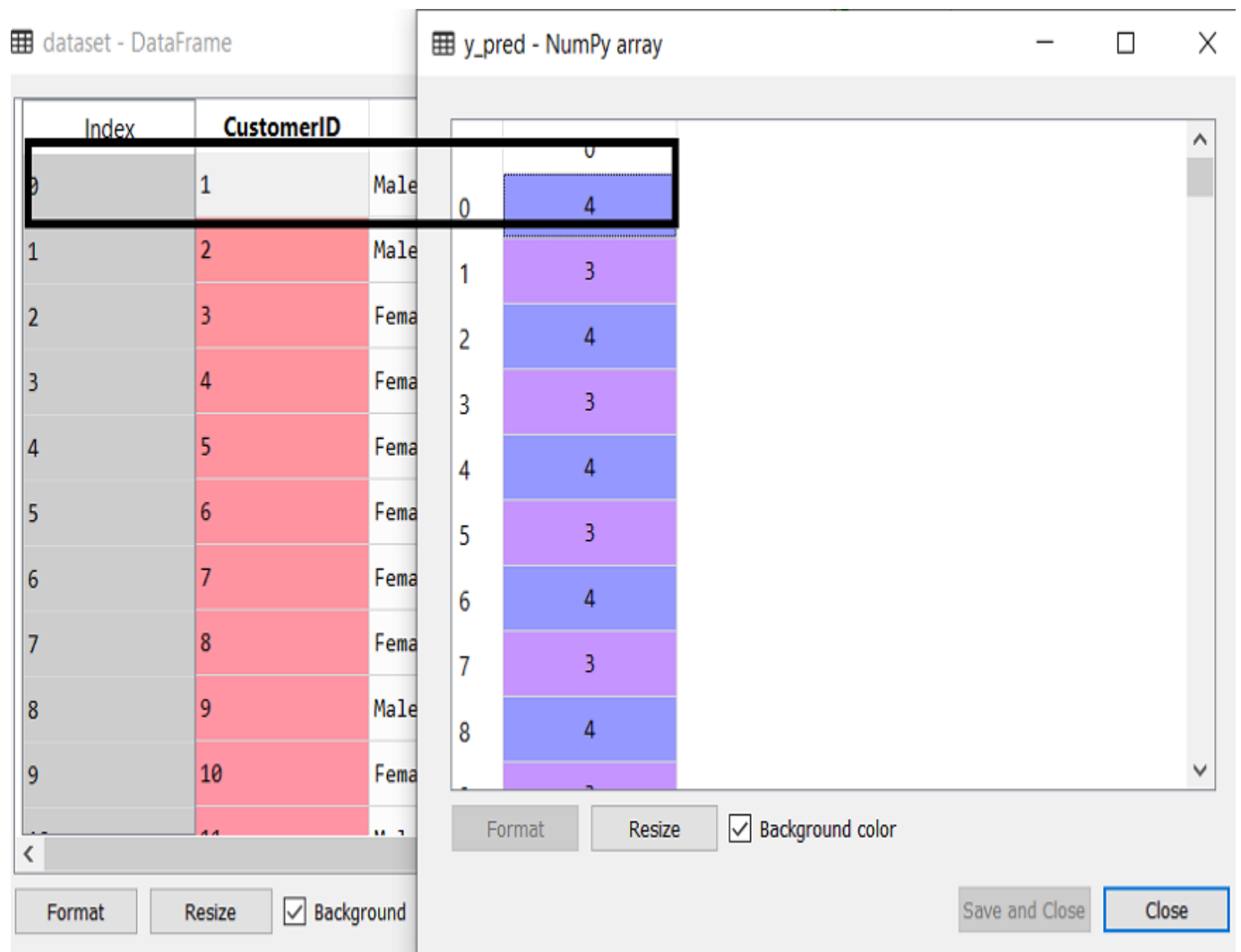
In the above code, we have imported the **AgglomerativeClustering** class of cluster module of scikit learn library.

Then we have created the object of this class named as **hc**. The AgglomerativeClustering class takes the following parameters:

- **n_clusters=5**: It defines the number of clusters, and we have taken here 5 because it is the optimal number of clusters.
- **affinity='euclidean'**: It is a metric used to compute the linkage.
- **linkage='ward'**: It defines the linkage criteria, here we have used the "ward" linkage. This method is the popular linkage method that we have already used for creating the Dendrogram. It reduces the variance in each cluster.

In the last line, we have created the dependent variable y_pred to fit or train the model. It does train not only the model but also returns the clusters to which each data point belongs.

After executing the above lines of code, if we go through the variable explorer option in our Sypder IDE, we can check the y_pred variable. We can compare the original dataset with the y_pred variable. Consider the below image:



As we can see in the above image, the **y_pred** shows the clusters value, which means the customer id 1 belongs to the 5th cluster (as indexing starts from 0, so 4 means 5th cluster), the customer id 2 belongs to 4th cluster, and so on.

Step-4: Visualizing the clusters

As we have trained our model successfully, now we can visualize the clusters corresponding to the dataset.

Here we will use the same lines of code as we did in k-means clustering, except one change. Here we will not plot the centroid that we did in k-means, because here we have used dendrogram to determine the optimal number of clusters. The code is given below:

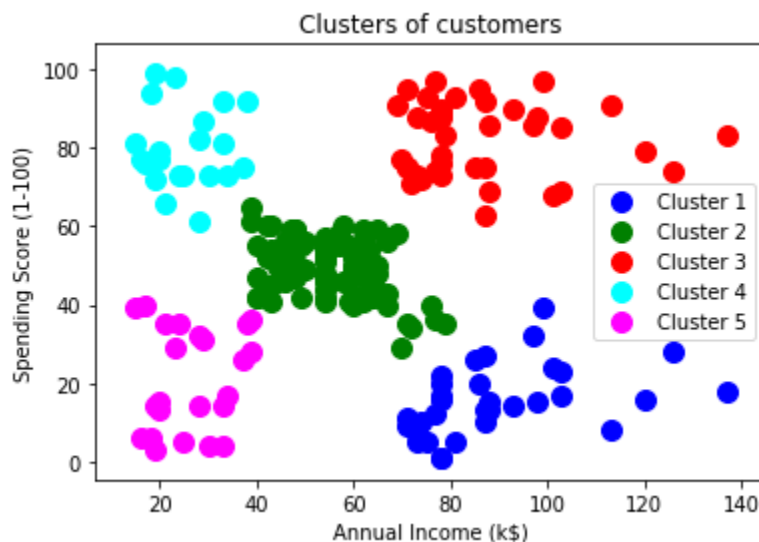
1. #visulaizing the clusters

```

2. mtp.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
3. mtp.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
4. mtp.scatter(x[y_pred == 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
5. mtp.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
6. mtp.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
7. mtp.title('Clusters of customers')
8. mtp.xlabel('Annual Income (k$)')
9. mtp.ylabel('Spending Score (1-100)')
10. mtp.legend()
11. mtp.show()

```

Output: By executing the above lines of code, we will get the below output:



K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an [Unsupervised Learning algorithm](#), which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that

need to be created in the process, as if $K=2$, there will be two clusters, and for $K=3$, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

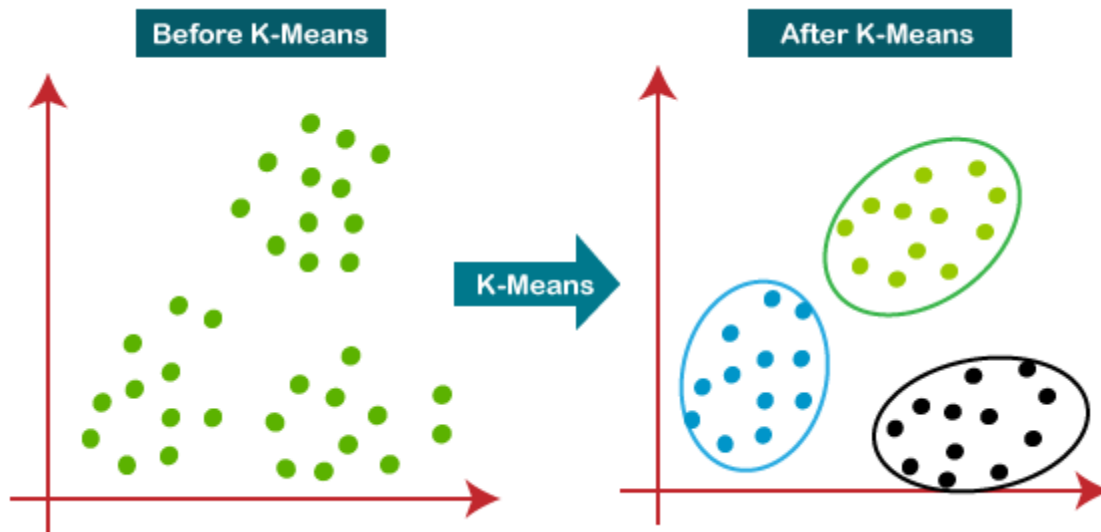
The algorithm takes the unlabeled dataset as input, divides the dataset into k -number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k -means [clustering](#) algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k -center. Those data points which are near to the particular k -center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K -means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

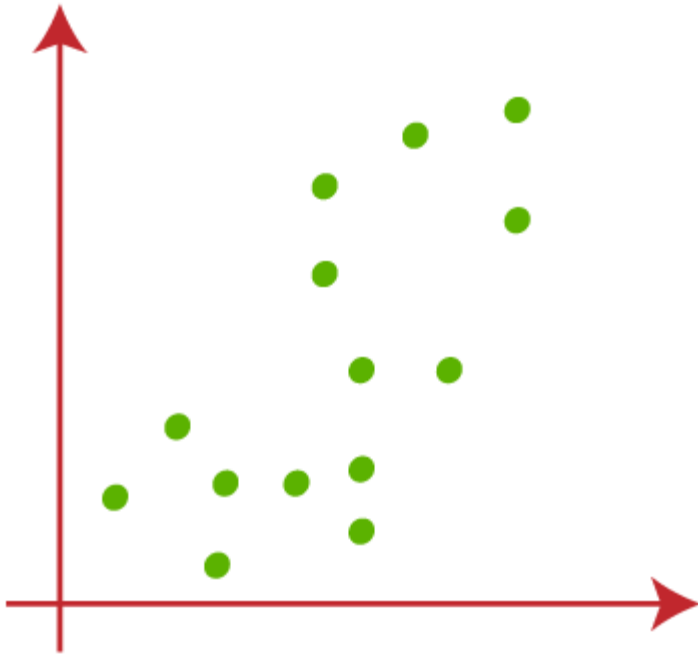
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

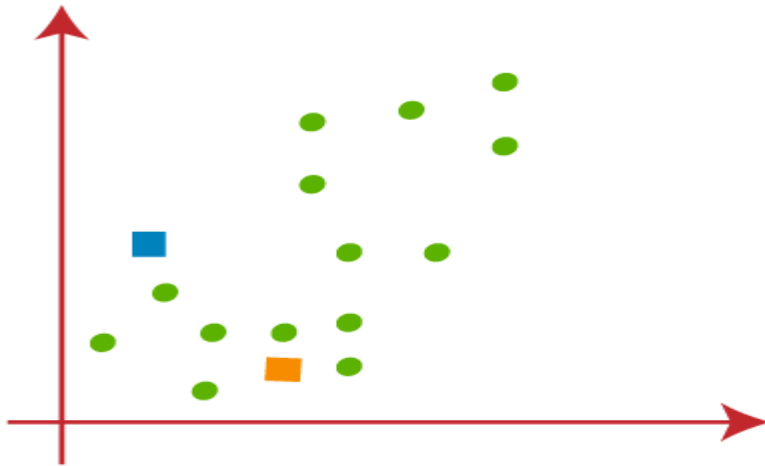
Let's understand the above steps by considering the visual plots:

Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:

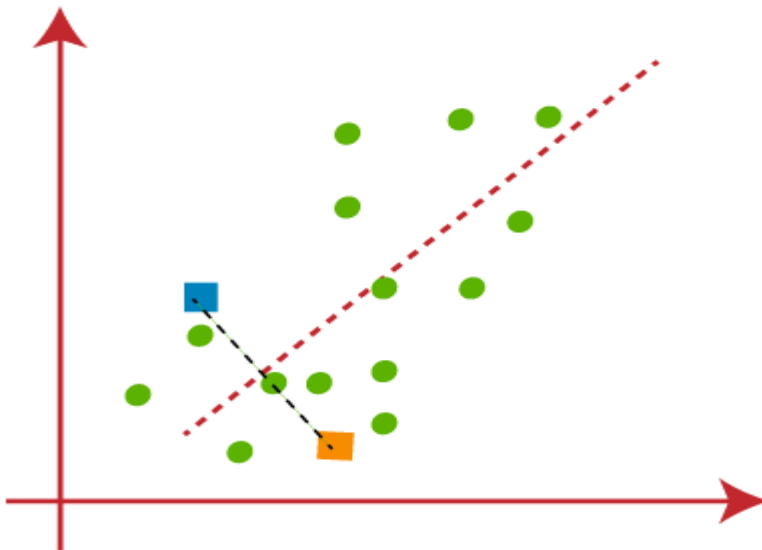


- Let's take number k of clusters, i.e., $K=2$, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the below

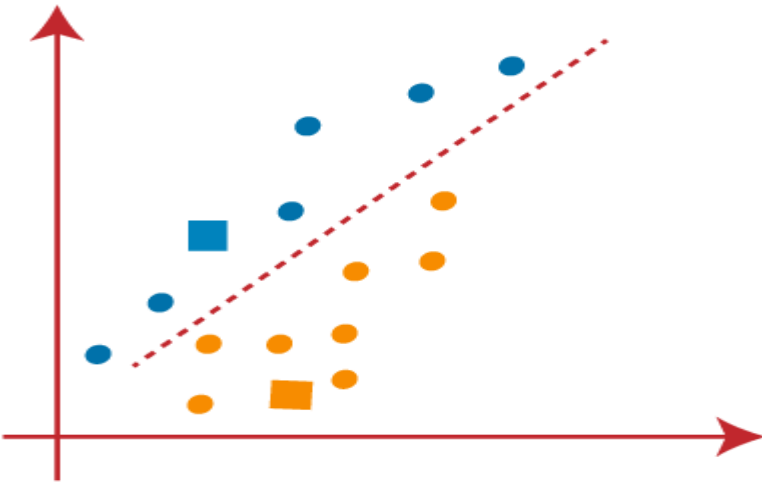
image:



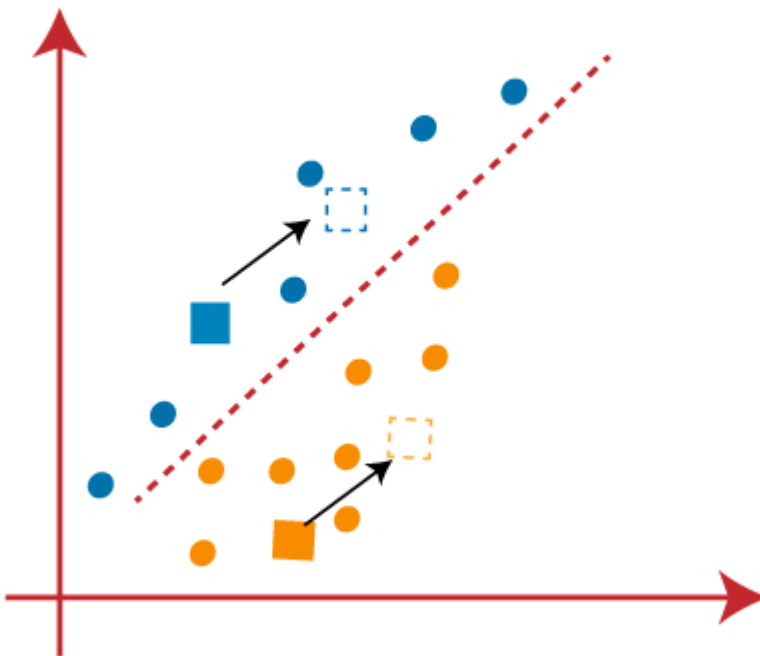
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



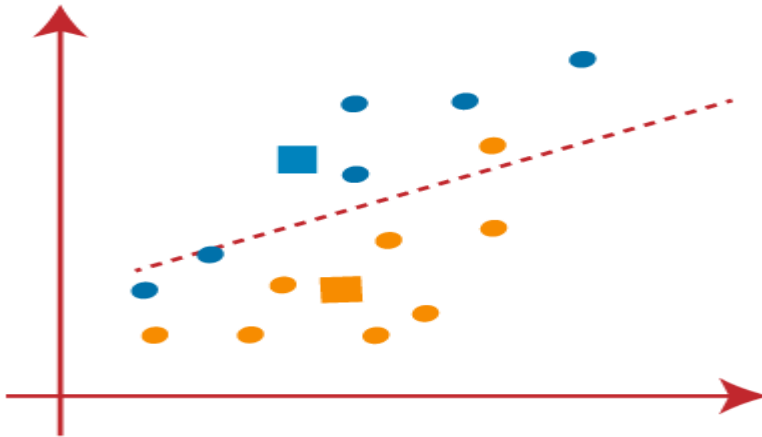
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



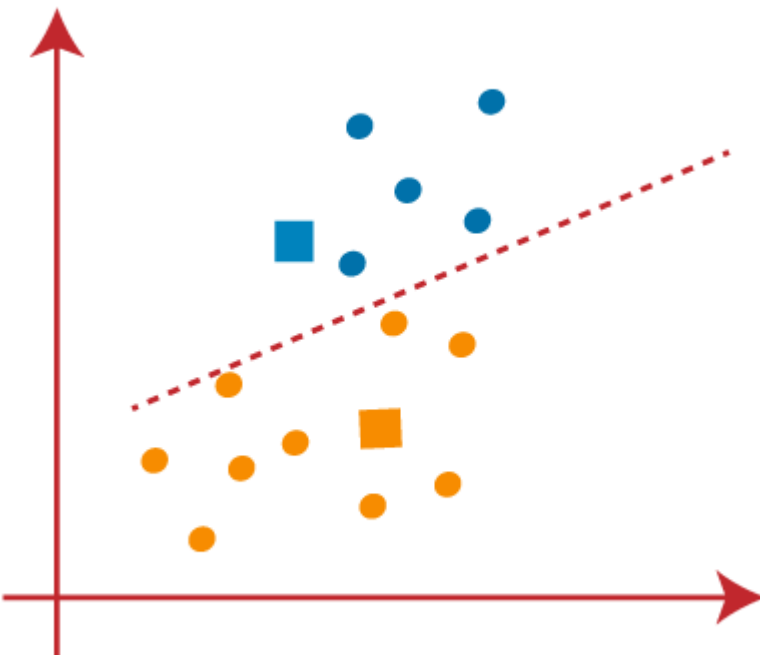
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

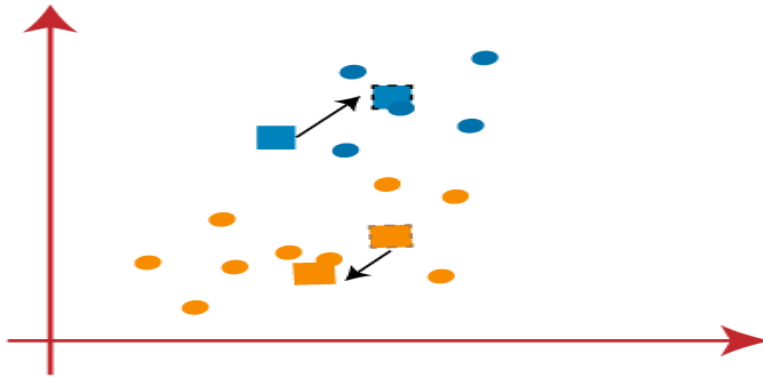


From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

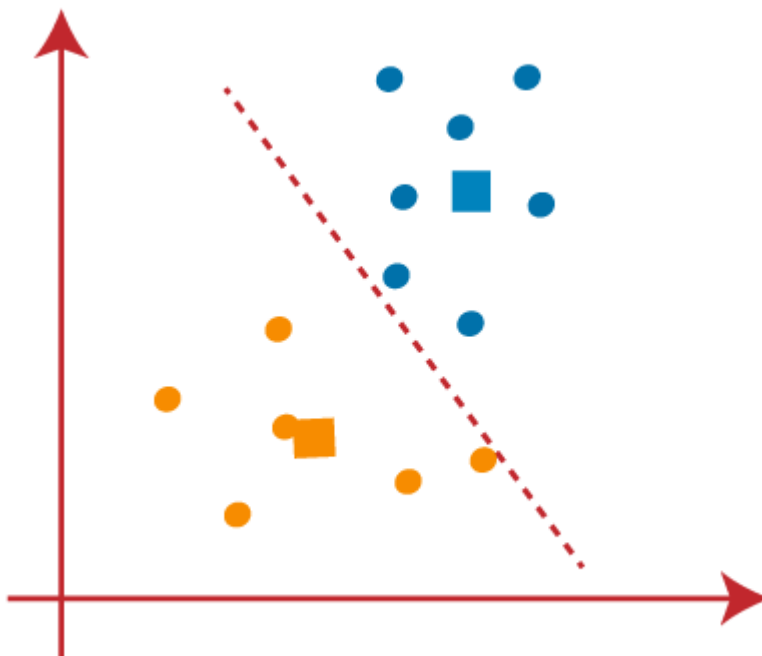


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

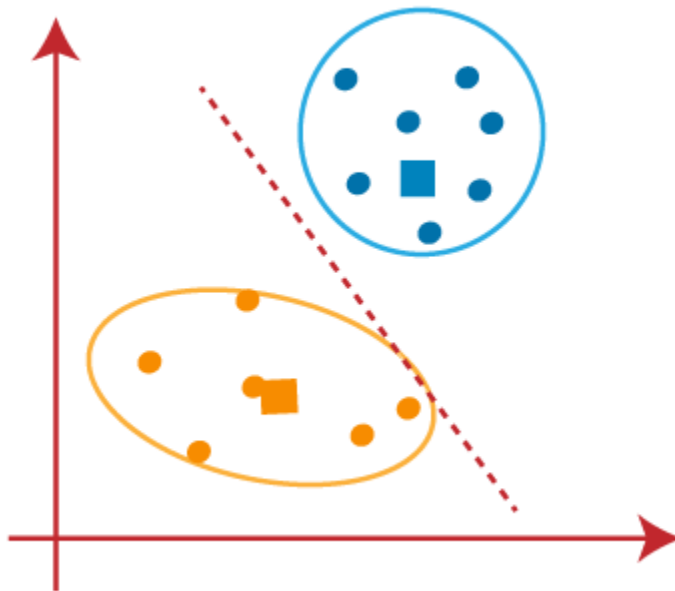
- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



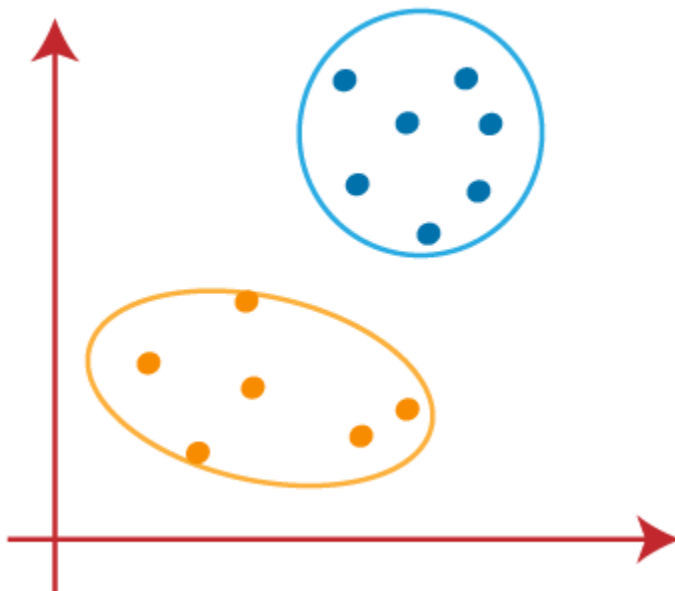
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS = \sum_{P_i \text{ in Cluster1}} \text{distance}(P_i, C_1)^2 + \sum_{P_i \text{ in Cluster2}} \text{distance}(P_i, C_2)^2 + \sum_{P_i \text{ in Cluster3}} \text{distance}(P_i, C_3)^2$$

In the above formula of WCSS,

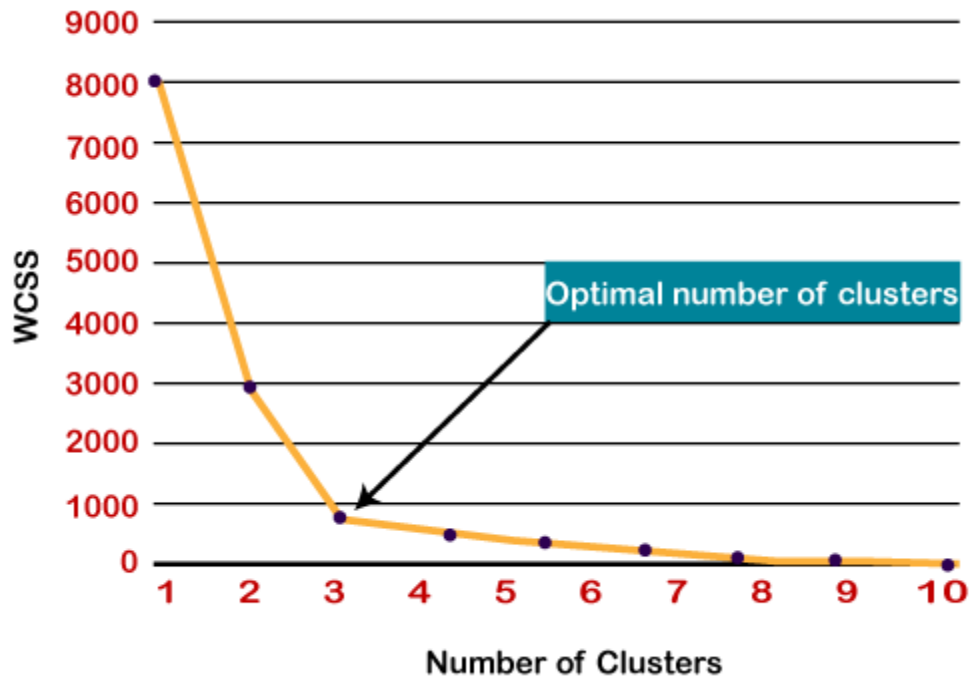
$\sum_{P_i \text{ in Cluster1}} \text{distance}(P_i, C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Note: We can choose the number of clusters equal to the given data points. If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot.

Python Implementation of K-means Clustering Algorithm

In the above section, we have discussed the K-means algorithm, now let's see how it can be implemented using [Python](#).

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id**, **Gender**, **Age**, **Annual Income (\$)**, and **Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**

- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

Step-1: Data pre-processing Step

The first step will be the data pre-processing, as we did in our earlier topics of Regression and Classification. But for the clustering problem, it will be different from other models. Let's discuss it:

- **Importing** **Libraries**

As we did in previous topics, firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

1. # importing libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd

In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

- **Importing the Dataset:**

Next, we will import the dataset that we need to use. So here, we are using the Mall_Customer_data.csv dataset. It can be imported using the below code:

1. # Importing the dataset
2. dataset = pd.read_csv('Mall_Customers_data.csv')

By executing the above lines of code, we will get our dataset in the Spyder IDE. The dataset looks like the below image:

Index	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

From the above dataset, we need to find some patterns in it.

- **Extracting Independent Variables**

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

1. `x = dataset.iloc[:, [3, 4]].values`

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10. Below is the code for it:

```
1. #finding optimal number of clusters using the elbow method
2. from sklearn.cluster import KMeans
3. wcss_list= [] #Initializing the list for the values of WCSS
4.
5. #Using for loop iterations from 1 to 10.
6. for i in range(1, 11):
7.     kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
8.     kmeans.fit(x)
9.     wcss_list.append(kmeans.inertia_)
10. mtp.plot(range(1, 11), wcss_list)
11. mtp.title('The Elbow Method Graph')
12. mtp.xlabel('Number of clusters(k)')
13. mtp.ylabel('wcss_list')
14. mtp.show()
```

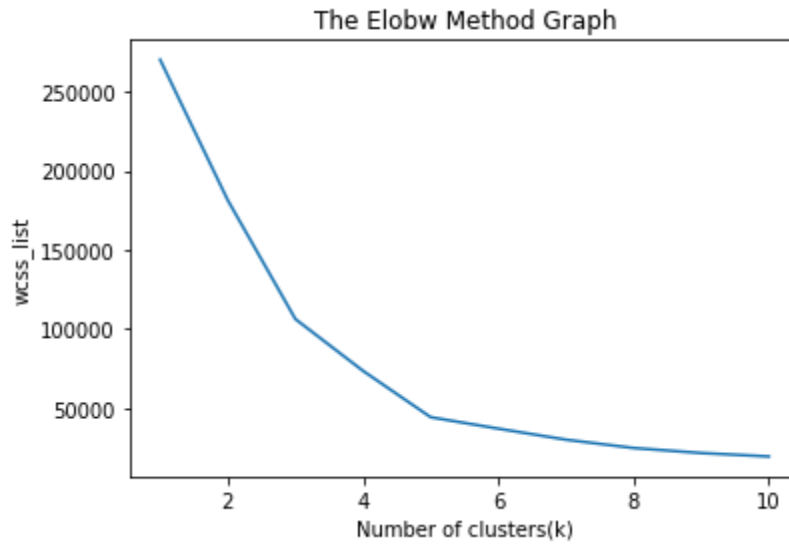
As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10th value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Output: After executing the above code, we will get the below output:



From the above plot, we can see the elbow point is at **5**. So the number of clusters here will be 5.

wcss_list - List (10 elements)

Index	Type	Size	Value
0	float64	1	269981.28
1	float64	1	181363.59595959596
2	float64	1	106348.37306211118
3	float64	1	73679.78903948834
4	float64	1	44448.45544793371
5	float64	1	37233.81451071001
6	float64	1	30259.65720728547
7	float64	1	25011.83934915659
8	float64	1	21850.165282585633
9	float64	1	19672.07284901432

Save and Close Close

Step- 3: Training the K-means algorithm on the training dataset

As we have got the number of clusters, so we can now train the model on the dataset.

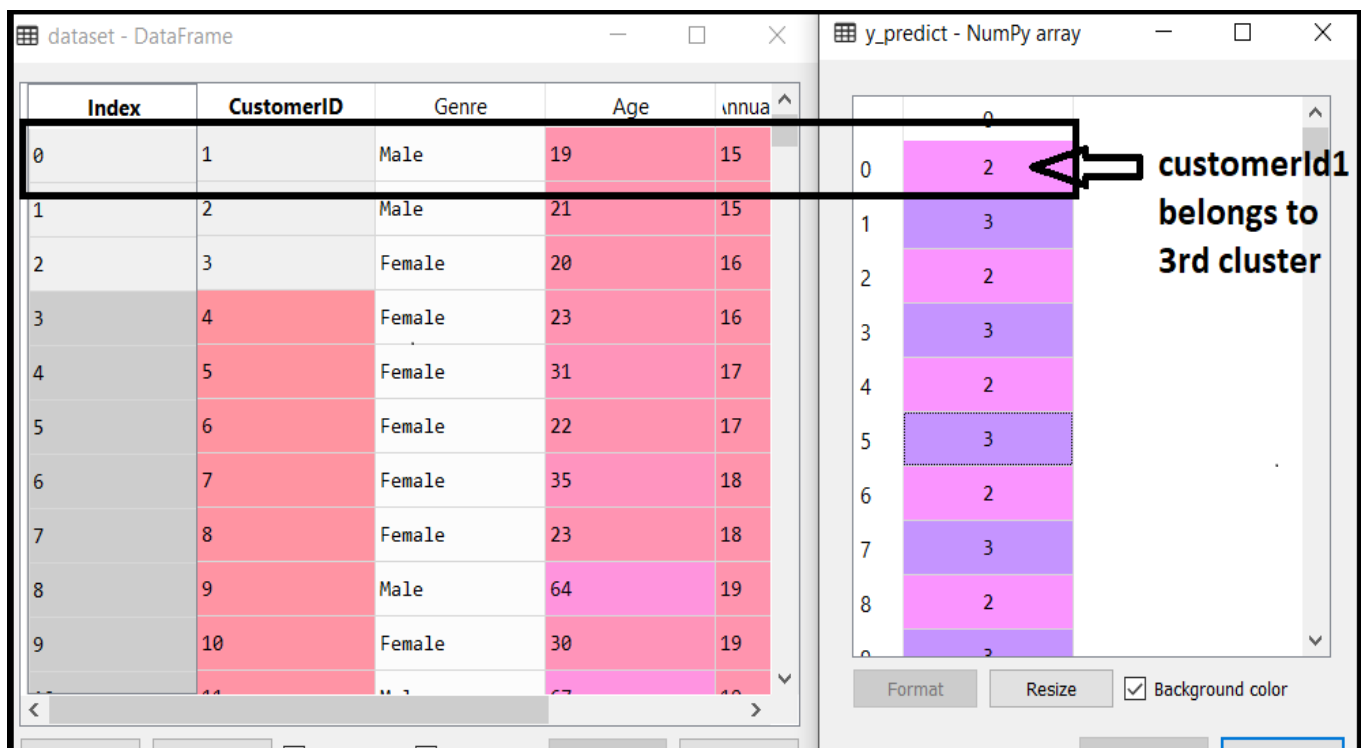
To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using `i`, we will use `5`, as we know there are 5 clusters that need to be formed. The code is given below:

1. #training the K-means model on a dataset
2. `kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)`
3. `y_predict= kmeans.fit_predict(x)`

The first line is the same as above for creating the object of KMeans class.

In the second line of code, we have created the dependent variable **y_predict** to train the model.

By executing the above lines of code, we will get the `y_predict` variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of `y_predict` with our original dataset. Consider the below image:



From the above image, we can now relate that the CustomerID 1 belongs to a cluster

3(as index starts from 0, hence 2 will be considered as 3), and 2 belongs to cluster 4, and so on.

Step-4: Visualizing the Clusters

The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

To visualize the clusters will use scatter plot using `mtp.scatter()` function of `matplotlib`.

1. `#visulaizing the clusters`
2. `mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') #for first cluster`
3. `mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #for second cluster`
4. `mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #for third cluster`
5. `mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #for fourth cluster`
6. `mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster`
7. `mtp.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], s = 300, c = 'yellow', label = 'Centroid')`
8. `mtp.title('Clusters of customers')`
9. `mtp.xlabel('Annual Income (k$)')`
10. `mtp.ylabel('Spending Score (1-100)')`
11. `mtp.legend()`
12. `mtp.show()`

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the `mtp.scatter`, i.e., `x[y_predict == 0, 0]` containing the x value for the showing the matrix of features values, and the `y_predict` is ranging from 0 to 1.

Output:



The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

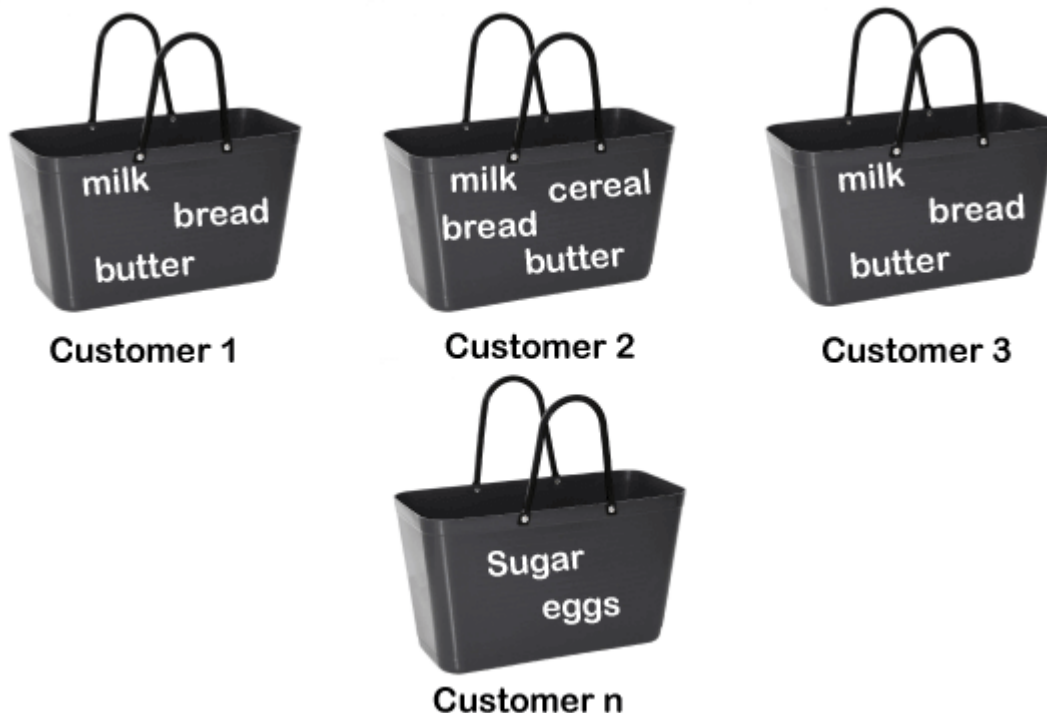
- **Cluster1** shows the customers with average salary and average spending so we can categorize these customers as
- Cluster2 shows the customer has a high income but low spending, so we can categorize them as **careful**.
- Cluster3 shows the low income and also low spending so they can be categorized as sensible.
- Cluster4 shows the customers with low income with very high spending so they can be categorized as **careless**.
- Cluster5 shows the customers with high income and high spending so they can be categorized as target, and these customers can be the most profitable customers for the mall owner.

Association Rule Learning

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of [machine learning](#), and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:



Association rule learning can be divided into three types of algorithms:

1. **Apriori**
2. **Eclat**
3. **F-P Growth Algorithm**

We will understand these algorithms in later chapters.

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.



Here the If element is called **antecedent**, and then statement is called as **Consequent**. These types of relationships where we can find out some association or relation between two items is known *as single cardinality*. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

- **Support**
- **Confidence**
- **Lift**

Let's understand each of them:

Support

Support is the frequency of A or how frequently an item appears in the dataset. It is defined as the fraction of the transaction T that contains the itemset X. If there are X datasets, then for transactions T, it can be written as:

$$\text{Supp}(X) = \frac{\text{Freq}(X)}{T}$$

Confidence

Confidence indicates how often the rule has been found to be true. Or how often the items X and Y occur together in the dataset when the occurrence of X is already given. It is the ratio of the transaction that contains X and Y to the number of records that contain X.

$$\text{Confidence} = \frac{\text{Freq}(X,Y)}{\text{Freq}(X)}$$

Lift

It is the strength of any rule, which can be defined as below formula:

$$\text{Lift} = \frac{\text{Supp}(X,Y)}{\text{Supp}(X) \times \text{Supp}(Y)}$$

It is the ratio of the observed support measure and expected support if X and Y are independent of each other. It has three possible values:

- If **Lift = 1**: The probability of occurrence of antecedent and consequent is independent of each other.
- **Lift > 1**: It determines the degree to which the two itemsets are dependent to each other.
- **Lift < 1**: It tells us that one item is a substitute for other items, which means one item has a negative effect on another.

Types of Association Rule Learning

Association rule learning can be divided into three algorithms:

Apriori Algorithm

This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.

It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

Eclat Algorithm

Eclat algorithm stands for **Equivalence Class Transformation**. This algorithm uses a depth-first search technique to find frequent itemsets in a transaction database. It performs faster execution than Apriori Algorithm.

F-P Growth Algorithm

The F-P growth algorithm stands for **Frequent Pattern**, and it is the improved version of the Apriori Algorithm. It represents the database in the form of a tree structure that is known as a frequent pattern or tree. The purpose of this frequent tree is to extract the most frequent patterns.

Applications of Association Rule Learning

It has various applications in machine learning and data mining. Below are some popular applications of association rule learning:

- **Market Basket Analysis:** It is one of the popular examples and applications of association rule mining. This technique is commonly used by big retailers to determine the association between items.
- **Medical Diagnosis:** With the help of association rules, patients can be cured easily, as it helps in identifying the probability of illness for a particular disease.
- **Protein Sequence:** The association rules help in determining the synthesis of artificial Proteins.
- It is also used for the **Catalog Design** and **Loss-leader Analysis** and many more other applications.