

# **UNIT-IV(PPDS)**

## **CLASSIFICATION**



# Classification

Classification:

Classification is the process of recognizing, understanding, and grouping ideas and objects into present categories or “sub-populations.” Using pre-categorized training datasets, machine learning programs use a variety of algorithms to classify future datasets into categories.

or

In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.

Or

Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.

# Classification

## Classification Algorithms

- The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data.
- In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups.
- the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc.
- In classification algorithm, a discrete output function( $y$ ) is mapped to input variable( $x$ ).

$y=f(x)$ , where  $y$  = categorical output

- The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.

# Classification

## Classification Algorithms

➤ The algorithm which implements the classification on a dataset is known as a **classifier**.

➤ There are two types of Classifications:

1. **Binary Classifier**
2. **Multi-class Classifier**

### **Binary Classifier:**

If the classification problem has only **two possible outcomes**, then it is called as BinaryClassifier.

**Examples:** YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.

### **Multi-class Classifier:**

If a classification problem **has more than two outcomes**, then it is called as Multi-classClassifier.

**Example:** Classifications of types of crops, Classification of types of music.

# Classification

## Types of ML Classification Algorithms:

➤ Classification Algorithms can be further divided into the Mainly two category:

**1. Linear Models**

**2. Non-linear Models**

### Linear Models

Logistic Regression

Support Vector Machines

### Non-linear Models

K-Nearest Neighbours

Kernel SVM

Naïve Bayes

Decision Tree Classification

Random Forest Classification

# Classification

## Logistic Regression in Machine Learning

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique.
- It is **used for predicting the categorical dependent variable using a given set of independent variables.**
- Logistic regression predicts the output of a categorical dependent variable. Therefore the **outcome must be a categorical or discrete value**. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1.**
- Logistic Regression is much similar to the Linear Regression except that how they are used.
- Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems.**

# Classification

## Logistic Regression in Machine Learning

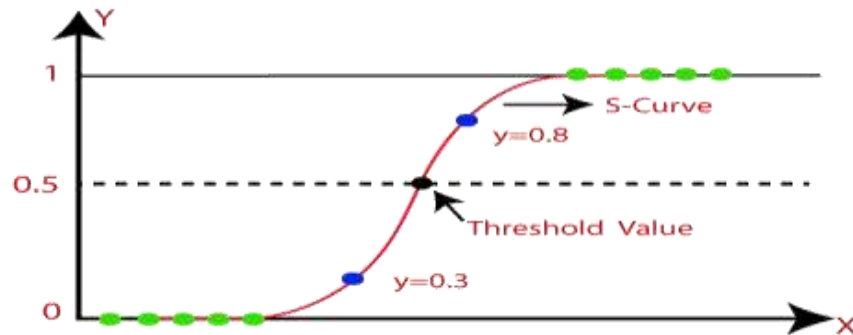
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped **logistic function**, which predicts two maximum values (0 or 1).
- The curve from the logistic function indicates the **likelihood** of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
- Logistic Regression is a significant machine learning algorithm because it has the **ability to provide probabilities and classify new data using continuous and discrete datasets**.
- Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.



# Classification

## Logistic Function (Sigmoid Function):

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It **maps any real value into another value within a range of 0 and 1.**
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- The below image is showing the logistic function:



# Classification

## Logistic Function (Sigmoid Function):

➤ In logistic regression, we use the concept of the **threshold value**, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

## Assumptions for Logistic Regression:

1. The **dependent variable must be categorical** in nature.
2. The **independent variable should not have multi-collinearity**.



# Classification

## Logistic Regression Equation:

- The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:
- We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_nx_n$$

- In Logistic Regression  $y$  can be between 0 and 1 only, so for this let's divide the above equation by  $(1-y)$ :

$$\frac{y}{1-y} ; 0 \text{ for } y=0, \text{ and infinity for } y=1$$



# Classification

## Logistic Regression Equation:

➤ But we need range between  $-\infty$  to  $+\infty$ , then take logarithm of the equation it will become:

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

➤ The above equation is the **final equation for Logistic Regression**.

# Classification

## Type of Logistic Regression:

➤ On the basis of the **categories**, Logistic Regression can be classified into **three** types:

1. Binomial
2. Multinomial
3. Ordinal

### **Binomial:**

In binomial Logistic regression, there can be only two possible types of the dependent variables.

Examples:

0 or 1,  
Pass or Fail, etc.



# Classification

## Type of Logistic Regression:

### **Multinomial:**

In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable.

Examples:

"cat", "dogs", or "sheep"

### **Ordinal:**

In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables.

Examples:

"low", "Medium", or "High".

# Classification



## Steps in Logistic Regression:

➤ To implement the Logistic Regression using Python the following steps are used:

1. Data Pre-processing step
2. Fitting Logistic Regression to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.



# Classification

## Example:

- There is a **dataset** given which **contains the information of various users** obtained from the social networking sites. There is a **car making company that has recently launched a new SUV car**. So the company wanted to check how many users from the dataset, wants to purchase the car.
- For this problem, we will build a **Machine Learning model using the Logistic regression algorithm**. The dataset is shown in the below image. In this problem, we will predict the **purchased variable (Dependent Variable)** by using **age and salary (Independent variables)**.



# Classification

Example:

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	15000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

# Classification

## Data Pre-processing step:

➤ In this step, we will pre-process/prepare the data so that we can use it in our code efficiently.

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
#importing datasets
```

```
data_set= pd.read_csv('user_data.csv')
```

By executing the above lines of code, we will get the dataset as the output.

# Classification

data_set - DataFrame					
Index	User ID	Gender	Age	EstimatedSalary	Purchased
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

Format

Resize

☒ Background color

☒ Column min/max

Save and Close

Close

# Classification

## Data Pre-processing step:

➤ we will extract the dependent and independent variables from the given dataset.

#Extracting Independent

`x= data_set.iloc[:, [2,3]].values`

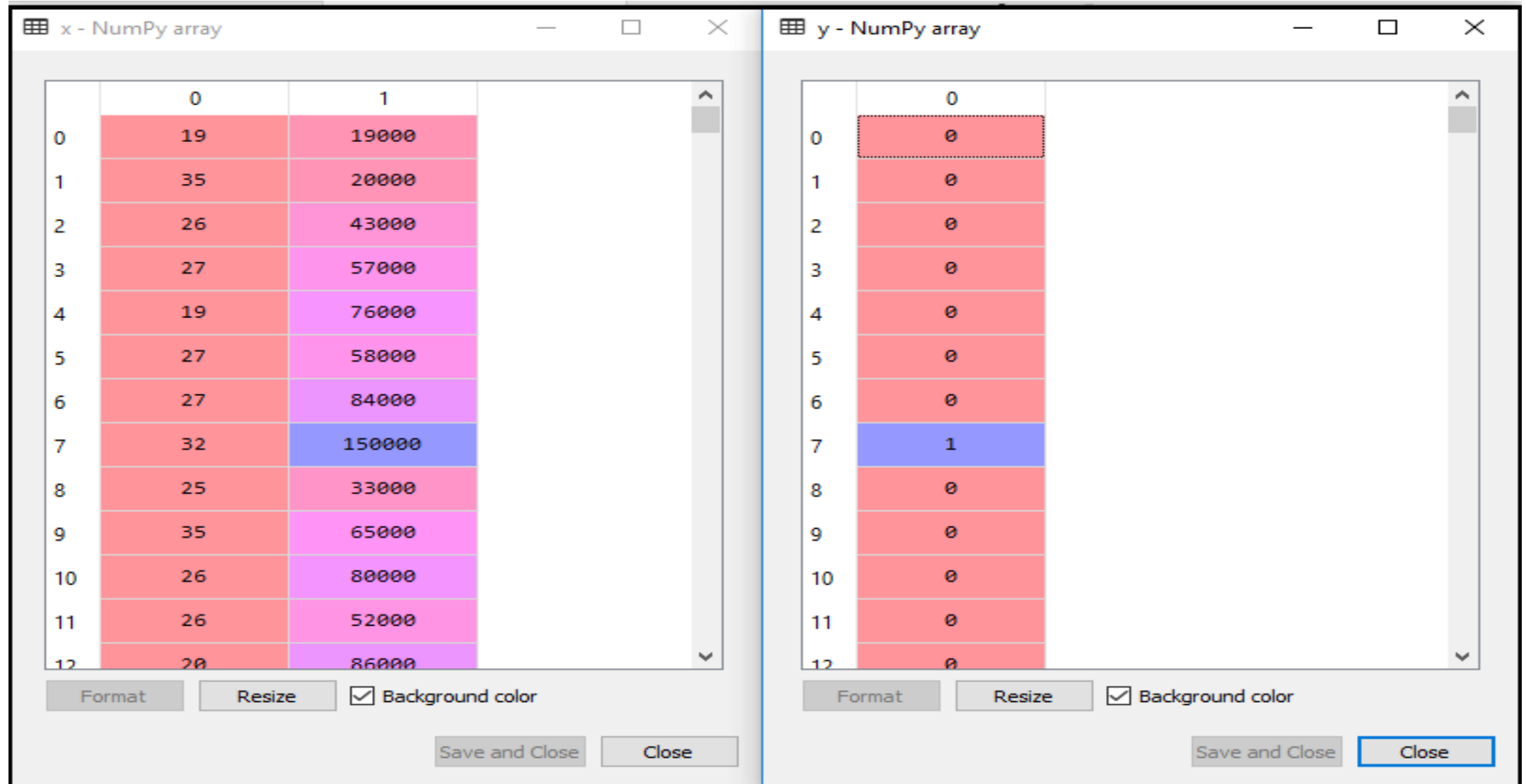
#Extracting dependent Variable

`y= data_set.iloc[:, 4].values`

➤ In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4.

➤ The output will be:

# Classification



# Classification

➤ we will split the dataset into a training set and test set.

from sklearn.model\_selection **import** train\_test\_split

x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)

The output for this is given below

```
print(x_train)
```

```
[[ 44  39000]
 [ 32 120000]
 [ 38  50000]
 [ 32 135000]
 [ 52  21000]
 [ 53 104000]
 [ 39  42000]
 [ 38  61000]
 [ 36  50000]
 [ 36  63000]
 [ 35  25000]
 [ 35  50000]
 [ 42  73000]
 [ 47  49000]
 [ 59  29000]
 [ 49  65000]
 [ 45 131000]
 [ 31  89000]
 [ 46  82000]
 [ 47  51000]
 [ 35  45000]
```

```
print(y_train)
```

```
[0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1
 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1
 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0
 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0
 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0
 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0]
```

# Classification

For testing set:

```
print(x_test)
```

```
[[ 30 87000]
 [ 38 50000]
 [ 35 75000]
 [ 30 79000]
 [ 35 50000]
 [ 27 20000]
 [ 31 15000]
 [ 36 144000]
 [ 18 68000]
 [ 47 43000]
 [ 30 49000]
 [ 28 55000]
 [ 37 55000]
 [ 39 77000]
 [ 20 86000]
 [ 32 117000]
 [ 37 77000]
 [ 19 85000]
 [ 55 130000]
 [ 35 22000]
 [ 35 47000]
 [ 47 144000]
```

```
print(y_test)
```

```
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
```

# Classification


➤ In logistic regression, we will do **feature scaling** because we want accurate **result of predictions**. Here we will only scale the independent variable because dependent variable have only 0 and 1 values.

```
from sklearn.preprocessing import StandardScaler  
st_x= StandardScaler()  
x_train= st_x.fit_transform(x_train)  
x_test= st_x.transform(x_test)
```

The scaled output is given below:



# Classification



x\_test - NumPy array

	0	1
0	-0.804802	0.504964
1	-0.0125441	-0.567782
2	-0.309641	0.157046
3	-0.804802	0.273019
4	-0.309641	-0.567782
5	-1.1019	-1.43758
6	-0.70577	-1.58254
7	-0.210609	2.15757
8	-1.99319	-0.0459058
9	0.878746	-0.770734
10	-0.804802	-0.596776
11	-1.00287	-0.422817
12	-0.111576	-0.422817

Format

Resize

☒ Background color

Save and Close

Close

x\_train - NumPy array

	0	1
0	0.581649	-0.886707
1	-0.606738	1.46174
2	-0.0125441	-0.567782
3	-0.606738	1.89663
4	1.37391	-1.40858
5	1.47294	0.997847
6	0.0864882	-0.799728
7	-0.0125441	-0.248858
8	-0.210609	-0.567782
9	-0.210609	-0.190872
10	-0.309641	-1.29261
11	-0.309641	-0.567782
12	0.383585	0.0990599

Format

Resize

☒ Background color

Save and Close

Close

# Classification

## 2. Fitting Logistic Regression to the Training set:

- We have well prepared our dataset, and now we will **train the dataset using the training set**. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.
- After importing the class, we will create a classifier object and use it to fit the model to the logistic regression.

#Fitting Logistic Regression to the training set

```
from sklearn.linear_model import LogisticRegression  
classifier= LogisticRegression(random_state=0)  
classifier.fit(x_train, y_train)
```

# Classification

## 2. Fitting Logistic Regression to the Training set:

### Output:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
                    intercept_scaling=1, l1_ratio=None, max_iter=100,  
                    multi_class='warn', n_jobs=None, penalty='l2',  
                    random_state=0, solver='warn', tol=0.0001, verbose=0,  
                    warm_start=False)
```

Hence our model is well fitted to the training set.

## 3. Predicting the Test Result

Our model is well trained on the training set, so we will now predict the result by using test set data.

#Predicting the test set result

```
y_pred= classifier.predict(x_test)
```

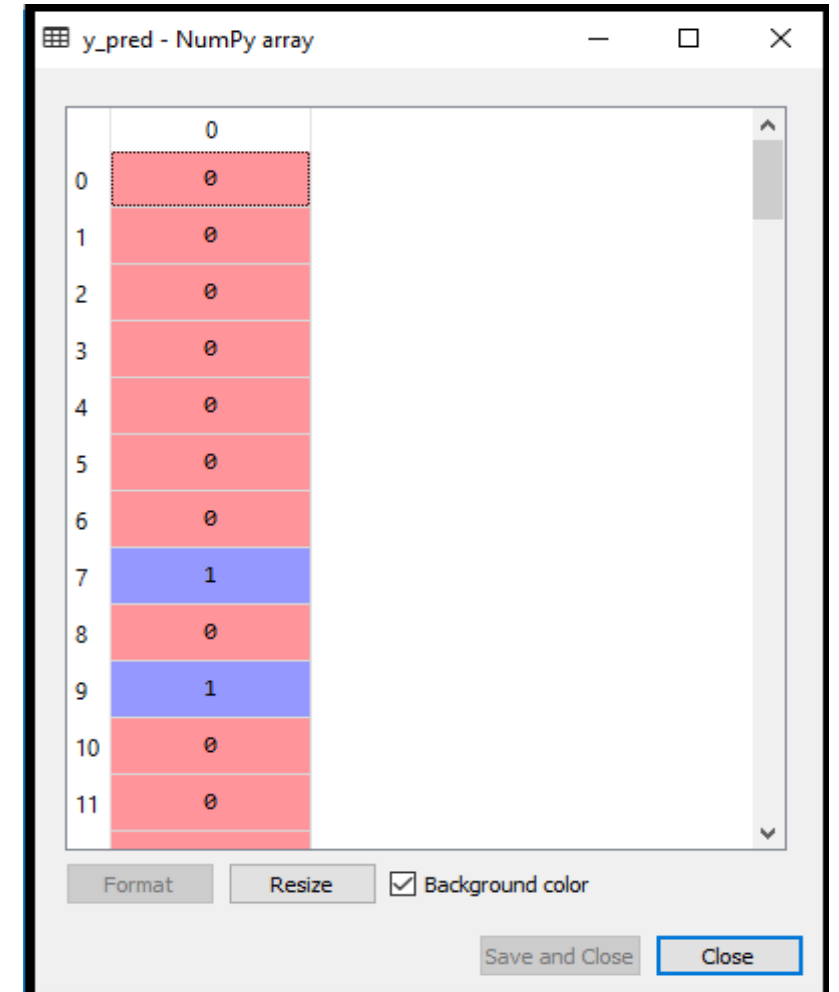
# Classification

## 3. Predicting the Test Result

### Output:

- a new vector (`y_pred`) will be created under the variable explorer option. It can be seen as:

The output image shows the corresponding predicted users who want to purchase or not purchased the car.



# Classification

## 4. Test Accuracy of the result

➤ Now we will create the **confusion matrix** here to check the accuracy of the classification.

➤ To create it, we need to import the **confusion\_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **y\_true**( the actual values) and **y\_pred** (the targeted value return by the classifier).

#Creating the Confusion matrix

```
from sklearn.metrics import confusion_matrix  
cm= confusion_matrix()
```

# Classification

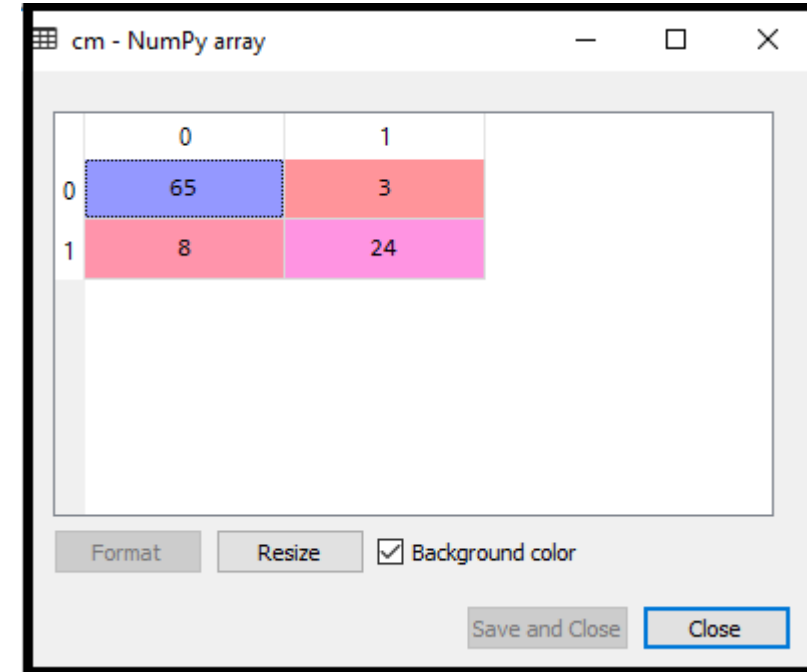
## 4. Test Accuracy of the result

### Output:

➤ a new confusion matrix will be created.

We can find the accuracy of the predicted result by interpreting the confusion matrix.

By above output, we can interpret that  $65+24= 89$  (Correct Output) and  $8+3= 11$ (Incorrect Output).



# Classification

## 5. Visualizing the training set result

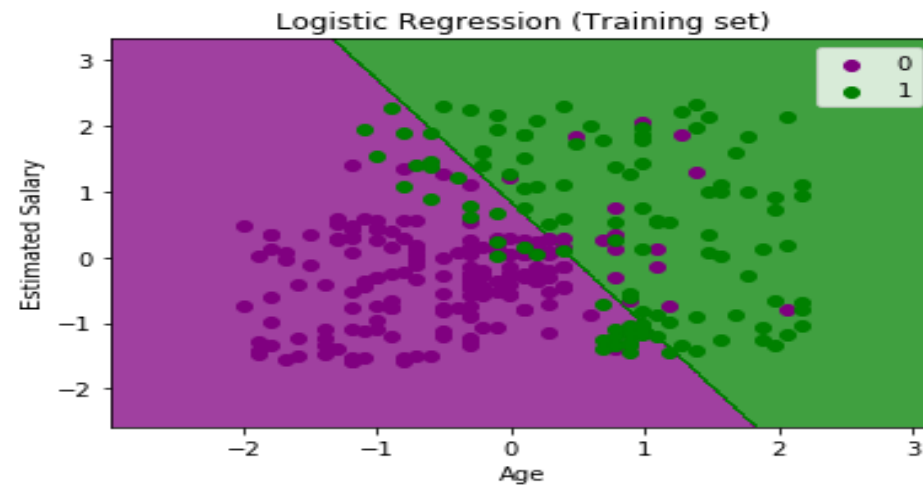
➤ Finally, we will **visualize the training set result**. To visualize the result, we will use **ListedColormap** class of matplotlib library.

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple', 'green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
               c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

# Classification

## 5. Visualizing the training set result

Output:





# Classification

## 5. Visualizing the training set result

The graph can be explained in the below points:

- In the above graph, we can see that there are some **Green points** within the green region and **Purple points** within the purple region.
- All these data points are the observation points from the training set, which shows the result for purchased variables.
- This graph is made by using two independent variables i.e., **Age on the x-axis** and **Estimated salary on the y-axis**.
- The **purple point observations** are for which purchased (dependent variable) is probably 0, i.e., users who did not purchase the SUV car.
- The **green point observations** are for which purchased (dependent variable) is probably 1 means user who purchased the SUV car.

# Classification

## 5. Visualizing the training set result

- We can also estimate from the graph that the **users who are younger with low salary, did not purchase the car, whereas older users with high estimated salary purchased the car.**
- But there are some purple points in the green region (Buying the car) and some green points in the purple region (Not buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.

# Classification

## 5. Visualizing the test set result:

- Our model is well trained using the training dataset. Now, we will visualize the result for new observations (Test set).
- The code for the test set will remain same as above except that here we will use **x\_test** and **y\_test** instead of **x\_train** and **y\_train**.

# Classification

## 5. Visualizing the test set result:

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple', 'green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Logistic Regression (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

# Classification

## 5. Visualizing the test set result:

➤ The graph shows the test set result. As we can see, the graph is divided into two regions (Purple and Green). And Green observations are in the green region, and Purple observations are in the purple region. So we can say it is a good prediction and model. Some of the green and purple data points are in different regions, which can be ignored as we have already calculated this error using the confusion matrix (11 Incorrect output).

➤ Hence our model is pretty good and ready to make new predictions for this classification problem.



# Classification

## Advantages:

- Logistic regression is specifically meant for classification, it is useful in understanding how a set of independent variables affect the outcome of the dependent variable.

## Disadvantages:

- The main disadvantage of the logistic regression algorithm is that it only works when the predicted variable is binary, it assumes that the data is free of missing values and assumes that the predictors are independent of each other.

## Use Cases:

Identifying risk factors for diseases

Word classification

Weather Prediction

Voting Applications

# K-Nearest Neighbor(KNN) Algorithm

## K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suitable category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification. But mostly it is used for the Classification problems.

# K-Nearest Neighbor(KNN) Algorithm

## K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-NN is a **non-parametric algorithm**, which means it does **not make any assumption on underlying data**.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.



# K-Nearest Neighbor(KNN) Algorithm

## K-Nearest Neighbor(KNN) Algorithm for Machine Learning

### Example:

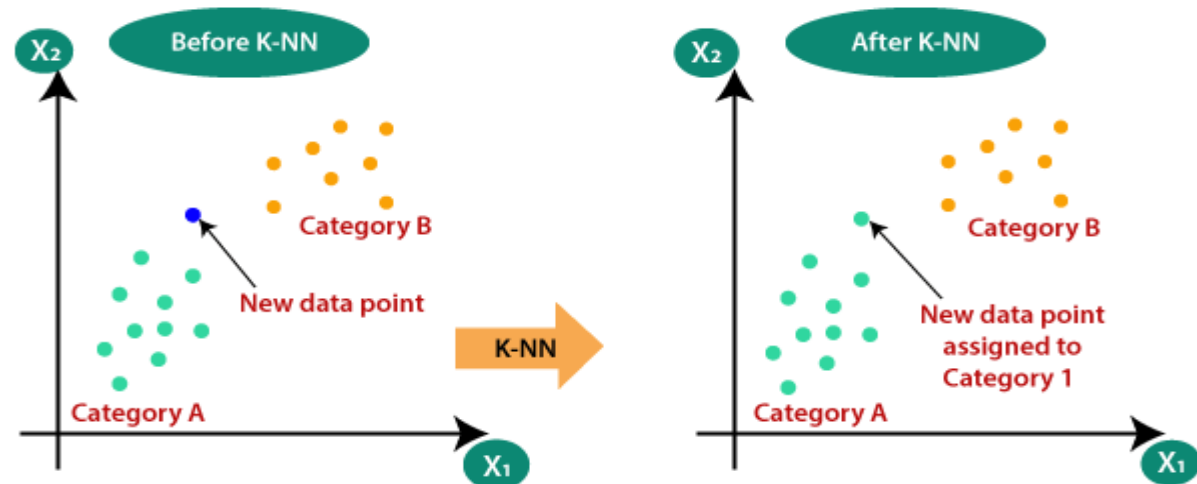
Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cat and dog images and based on the most similar features it will put it in either cat or dog category.



# K-Nearest Neighbor(KNN) Algorithm

## Why do we need a K-NN Algorithm?

- Suppose there are two categories, i.e., Category A and Category B, and we have a new data point  $x_1$ , so this data point will lie in which of these categories.
- To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily **identify the category or class of a particular dataset**. Consider the below diagram:



# K-Nearest Neighbor(KNN) Algorithm

How does K-NN work?

➤ The K-NN working can be explained on the basis of the below algorithm:

**Step-1:** Select the number K of the neighbors

**Step-2:** Calculate the Euclidean distance of **K number of neighbors**

**Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.

**Step-4:** Among these k neighbors, count the number of the data points in each category.

**Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.

**Step-6:** Our model is ready.

# K-Nearest Neighbor(KNN) Algorithm

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them.
- **The most preferred value for K is 5.**
- A very low value for K such as  $K=1$  or  $K=2$ , can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

# K-Nearest Neighbor(KNN) Algorithm

## Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

## Disadvantages of KNN Algorithm:

- Always needs to **determine the value of K** which may be complex some time.
- The **computation cost is high** because of calculating the distance between the data points for all the training samples.

# K-Nearest Neighbor(KNN) Algorithm

## Python implementation of the KNN algorithm

➤ Python's **Scikit-Learn library** can be used to implement the KNN algorithm

Example:

➤ There is a Car manufacturer company that has manufactured a new SUV car. The company wants to give the ads to the users who are interested in buying that SUV. So for this problem, we have a dataset that contains multiple user's information through the social network.

➤ The dataset contains lots of information but the **Estimated Salary** and **Age** we will consider for the independent variable and the **Purchased variable** is for the dependent variable. Below is the dataset:

# K-Nearest Neighbor(KNN) Algorithm

## Python implementation of the KNN algorithm

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0
15697686	Male	29	80000	0
15733883	Male	47	25000	1
15617482	Male	45	26000	1
15704583	Male	46	28000	1
15621083	Female	48	29000	1
15649487	Male	45	22000	1
15736760	Female	47	49000	1

# K-Nearest Neighbor(KNN) Algorithm

## **Steps to implement the K-NN algorithm:**

1. Data Pre-processing step
2. Fitting the K-NN algorithm to the Training set
3. Predicting the test result
4. Test accuracy of the result(Creation of Confusion matrix)
5. Visualizing the test set result.



# K-Nearest Neighbor(KNN) Algorithm

## 1. Data Pre-Processing Step:

➤ In this step, we will pre-process/prepare the data so that we can use it in our code efficiently.

```
import numpy as nm
```

```
import matplotlib.pyplot as mtp
```

```
import pandas as pd
```

```
#importing datasets
```

```
data_set= pd.read_csv('suv_data.csv')
```

Index	User ID	Gender	Age	EstimatedSalary	Purchased
92	15809823	Male	26	15000	0
150	15679651	Female	26	15000	0
43	15792008	Male	30	15000	0
155	15610140	Female	31	15000	0
32	15573452	Female	21	16000	0
180	15685576	Male	26	16000	0
79	15655123	Female	26	17000	0
40	15764419	Female	27	17000	0
128	15722758	Male	30	17000	0
58	15642885	Male	22	18000	0
29	15669656	Male	31	18000	0
13	15704987	Male	32	18000	0
74	15592877	Male	32	18000	0
0	15624510	Male	19	19000	0

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

# K-Nearest Neighbor(KNN) Algorithm

## Data Pre-Processing Step:

➤ The next step is to **split our dataset into its attributes and labels**. i.e, extract the **dependent and independent variables** from the given dataset.

#Extracting Independent and dependent Variable

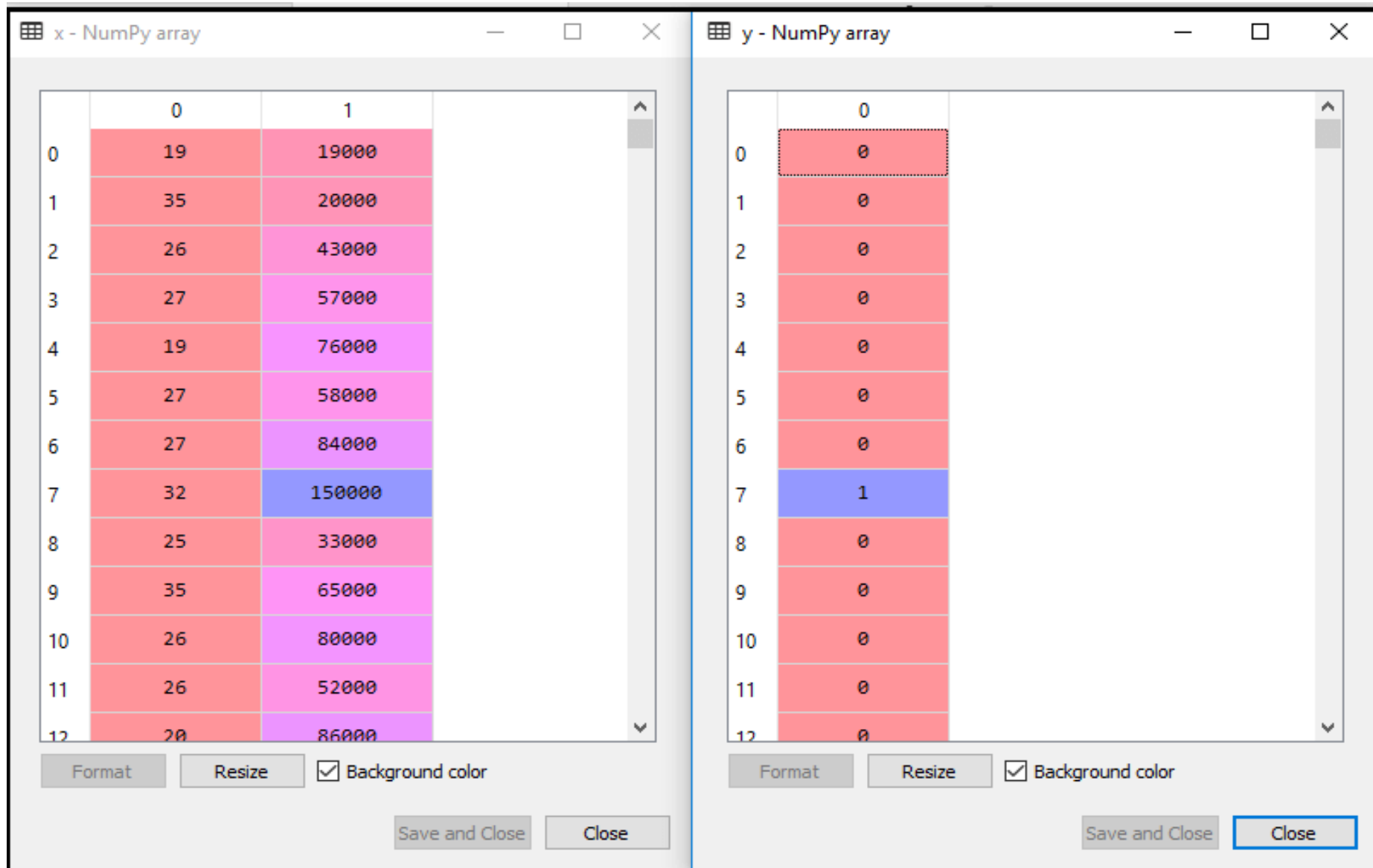
```
x= data_set.iloc[:, [2,3]].values
```

```
y= data_set.iloc[:, 4].values
```

➤ In the above code, we have taken [2, 3] for x because our independent variables are age and salary, which are at index 2, 3. And we have taken 4 for y variable because our dependent variable is at index 4.

```
Print(x)
```

```
Print(y)
```



# K-Nearest Neighbor(KNN) Algorithm

## Data Pre-Processing Step:

### *Train Test Split*

➤ To avoid **over-fitting**, we will **divide our dataset into training and test splits**, which gives us a better idea as to how our algorithm performed during the testing phase. This way our algorithm is tested on un-seen data, as it would be in a production application.

➤ To create training and test splits, execute the following script:

```
# Splitting the dataset into training and test set.  
from sklearn.model_selection import train_test_split  
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

➤ The above script **splits the dataset into 75% train data and 25% test data.**

# K-Nearest Neighbor(KNN) Algorithm

## Data Pre-Processing Step: *Training data*

```
print(x_train)
```

```
[[ 44 39000]
 [ 32 120000]
 [ 38 50000]
 [ 32 135000]
 [ 52 21000]
 [ 53 104000]
 [ 39 42000]
 [ 38 61000]
 [ 36 50000]
 [ 36 63000]
 [ 35 25000]
 [ 35 50000]
 [ 42 73000]
 [ 47 49000]
 [ 59 29000]
 [ 49 65000]
 [ 45 131000]
 [ 31 89000]
 [ 46 82000]
 [ 47 51000]
 [ 35 45000]]
```

```
print(y_train)
```

```
[0 1 0 1 1 1 0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 1 1 0 1 0 1 0 0 1
 0 0 1 0 0 0 0 0 1 1 1 1 0 0 0 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 0 1
 1 1 0 0 1 1 0 0 1 1 0 1 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 0 1 1 1 1 1 0 1 1 0
 1 0 0 0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1 1 0 0
 0 0 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0
 0 0 0 0 1 1 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 0]
```

# K-Nearest Neighbor(KNN) Algorithm

## Data Pre-Processing Step:

### *Test data*

```
print(x_test)
```

```
[[ 30 87000]
 [ 38 50000]
 [ 35 75000]
 [ 30 79000]
 [ 35 50000]
 [ 27 20000]
 [ 31 15000]
 [ 36 144000]
 [ 18 68000]
 [ 47 43000]
 [ 30 49000]
 [ 28 55000]
 [ 37 55000]
 [ 39 77000]
 [ 20 86000]
 [ 32 117000]
 [ 37 77000]
 [ 19 85000]
 [ 55 130000]
 [ 35 22000]
 [ 35 47000]
 [ 47 144000]
```

```
print(y_test)
```

```
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1
 0 0 0 0 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 0 0 1 0 1 1 1]
```

# K-Nearest Neighbor(KNN) Algorithm

## Data Pre-Processing Step:

### *Feature Scaling*

➤ Before making any actual predictions, it is always a good practice to scale the features so that all of them can be uniformly evaluated.

#feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
st_x= StandardScaler()
```

```
x_train= st_x.fit_transform(x_train)
```

```
x_test= st_x.transform(x_test)
```

➤ From the output image, we can see that our data is successfully scaled.



# K-Nearest Neighbor(KNN) Algorithm

## Data Pre-Processing Step: *Feature Scaling*

```
print(x_train)
```

```
[ [ 0.58164944 -0.88670699]
  [-0.60673761  1.46173768]
  [-0.01254409 -0.5677824 ]
  [-0.60673761  1.89663484]
  [ 1.37390747 -1.40858358]
  [ 1.47293972  0.99784738]
  [ 0.08648817 -0.79972756]
  [-0.01254409 -0.24885782]
  [-0.21060859 -0.5677824 ]
  [-0.21060859 -0.19087153]
  [-0.30964085 -1.29261101]
  [-0.30964085 -0.5677824 ]
  [ 0.38358493  0.09905991]
  [ 0.8787462  -0.59677555]
  [ 2.06713324 -1.17663843]
  [ 1.07681071 -0.13288524]
  [ 0.68068169  1.78066227]
  [-0.70576986  0.56295021]
  - - - - -
```

```
print(x_test)
```

```
[ [-0.80480212  0.50496393]
  [-0.01254409 -0.5677824 ]
  [-0.30964085  0.1570462 ]
  [-0.80480212  0.27301877]
  [-0.30964085 -0.5677824 ]
  [-1.10189888 -1.43757673]
  [-0.70576986 -1.58254245]
  [-0.21060859  2.15757314]
  [-1.99318916 -0.04590581]
  [ 0.8787462  -0.77073441]
  [-0.80480212 -0.59677555]
  [-1.00286662 -0.42281668]
  [-0.11157634 -0.42281668]
  [ 0.08648817  0.21503249]
  [-1.79512465  0.47597078]
  [-0.60673761  1.37475825]
  [-0.11157634  0.21503249]
  [-1.89415691  0.44697764]
  [ 1.67100423  1.75166912]
  [-0.30964085 -1.37959044]
  [-0.30964085 -0.65476184]
  [ 0.8787462  2.15757314]
  [ 0.28455268 -0.53878926]
  [ 0.8787462  1.02684052]
```

# K-Nearest Neighbor(KNN) Algorithm

## 2. Fitting K-NN classifier to the Training data:

- Now we will fit the K-NN classifier to the training data.
- To do this we will **import the KNeighborsClassifier class of Sklearn Neighbors** library.
- After importing the class, we will create the **Classifier** object of the class. The Parameter of this class will be
  - n\_neighbors:** To define the required neighbors of the algorithm. Usually, it takes 5.
  - metric='minkowski':** This is the default parameter and it decides the distance between the points.
  - p=2:** It is equivalent to the standard Euclidean metric.
- And then we will fit the classifier to the training data.

# K-Nearest Neighbor(KNN) Algorithm

## Fitting K-NN classifier to the Training data:

```
#Fitting K-NN classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2 )
classifier.fit(x_train, y_train)
```

## **Output:**

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

# K-Nearest Neighbor(KNN) Algorithm

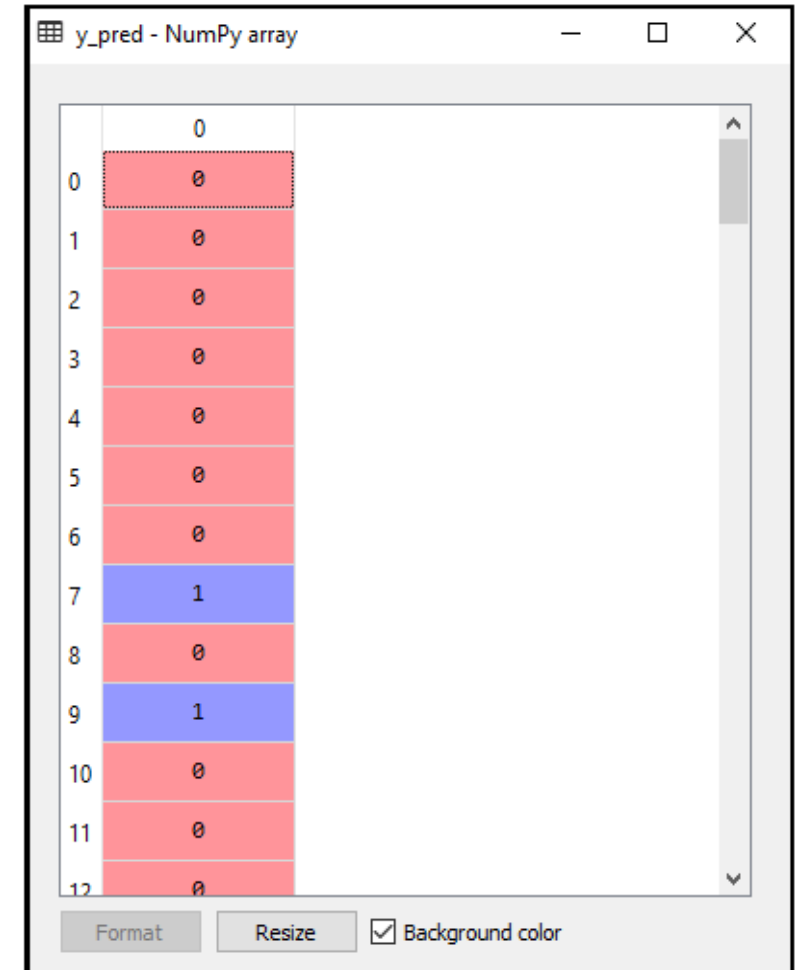
## 3. Predicting the Test Result:

To predict the test set result, we will create a **y\_pred** vector .

#Predicting the test set result

```
y_pred= classifier.predict(x_test)
```

**Output:**



# K-Nearest Neighbor(KNN) Algorithm

## 4. Test Accuracy:

- Now we will create the Confusion Matrix for our K-NN model to see the accuracy of the classifier.

#Creating the Confusion matrix

```
from sklearn.metrics import confusion_matrix  
cm= confusion_matrix(y_test, y_pred)  
print(cm)
```

## Output:

In the image, we can see there are  
 $64+29= 93$  correct predictions and  
 $3+4= 7$  incorrect predictions



	0	1
0	64	4
1	3	29

# K-Nearest Neighbor(KNN) Algorithm

## 5. Visualizing the Training set result:

➤ Now, we will visualize the training set result for K-NN model

#Visualizing the training set result

```
from matplotlib.colors import ListedColormap
```

```
x_set, y_set = x_train, y_train
```

```
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),  
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
```

```
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
```

```
alpha = 0.75, cmap = ListedColormap(('red', 'green' )))
```

```
mtp.xlim(x1.min(), x1.max())
```

```
mtp.ylim(x2.min(), x2.max())
```

```
for i, j in enumerate(nm.unique(y_set)):
```

```
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
```

```
               c = ListedColormap(('red', 'green'))(i), label = j)
```

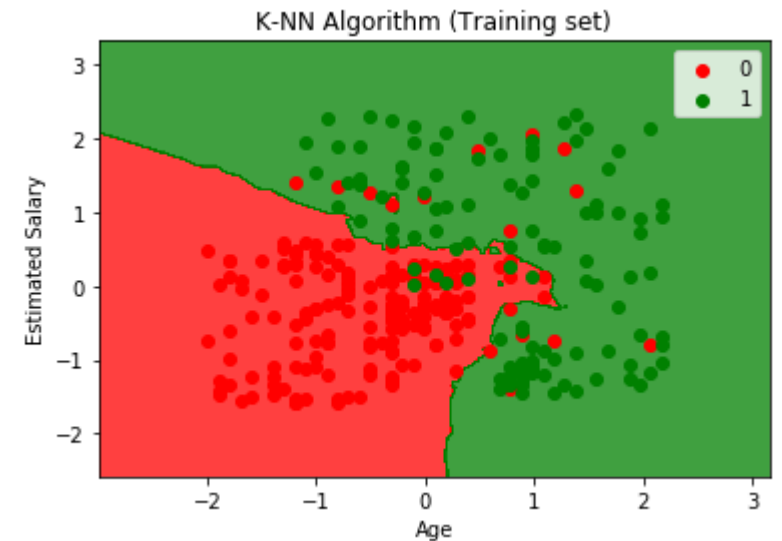
```
mtp.title('K-NN Algorithm (Training set)')
```

```
mtp.xlabel('Age')
```

```
mtp.ylabel('Estimated Salary')
```

```
mtp.legend()
```

```
mtp.show()
```



# K-Nearest Neighbor(KNN) Algorithm

## Visualizing the Training set result:

- As we can see the graph is showing the red point and green points. The green points are for Purchased(1) and Red Points for not Purchased(0) variable.
- The graph is showing an irregular boundary instead of showing any straight line or any curve because it is a K-NN algorithm, i.e., finding the nearest neighbor.
- The graph has classified users in the correct categories as most of the users who didn't buy the SUV are in the red region and users who bought the SUV are in the green region.
- The graph is showing good result but still, there are some green points in the red region and red points in the green region. But this is no big issue as by doing this model is prevented from overfitting issues.
- Hence our model is well trained.

# K-Nearest Neighbor(KNN) Algorithm

## Visualizing the Test set result:

➤ After the training of the model, we will now test the result by putting a new dataset, i.e., Test dataset. Code remains the same except some minor changes: such as **x\_train** and **y\_train** will be replaced by **x\_test** and **y\_test**.

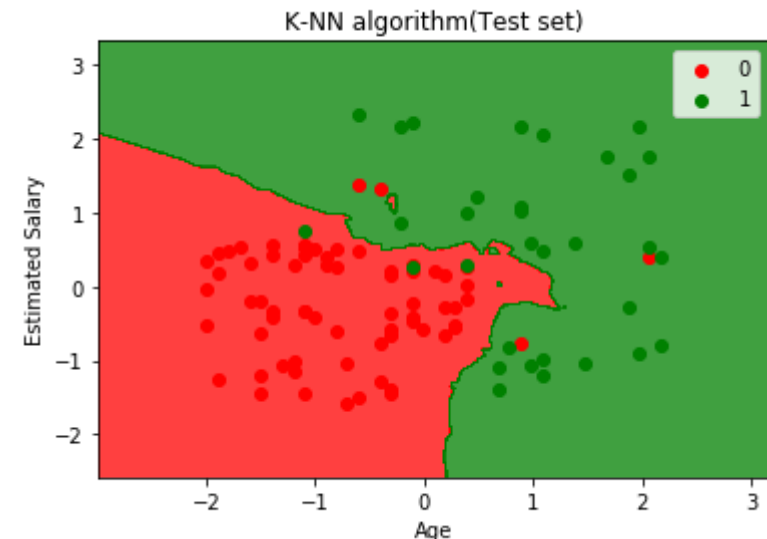


# K-Nearest Neighbor(KNN) Algorithm

## Visualizing the Test set result:

```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step = 0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

## Output:



# K-Nearest Neighbor(KNN) Algorithm

- As we can see in the graph, the predicted output is well good as most of the red points are in the red region and most of the green points are in the green region.
- However, there are few green points in the red region and a few red points in the green region. So these are the incorrect observations that we have observed in the confusion matrix(7 Incorrect output).

# Naïve Bayes Classifier Algorithm

## Naïve Bayes Classifier Algorithm

- Naïve Bayes algorithm is a supervised learning algorithm, which is based on **Bayes theorem** and used for solving classification problems.
- It is mainly **used in *text classification* that includes a high-dimensional training dataset.**
- Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions.
- **It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.**
- Some popular examples of Naïve Bayes Algorithm are **spam filtration, Sentimental analysis, and classifying articles.**

# Naïve Bayes Classifier Algorithm

## Why is it called Naïve Bayes?

- The Naïve Bayes algorithm is comprised of **two words** Naïve and Bayes, which can be described as:
  - **Naïve:**

It is called Naïve because it **assumes that the occurrence of a certain feature is independent of the occurrence of other features**
  - **Bayes:**

It is called Bayes because it depends on the principle of Baye's Theorem.

# Naïve Bayes Classifier Algorithm

## Bayes' Theorem:

➤ Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the **probability of a hypothesis with prior knowledge**. It depends on **the conditional probability**.

➤ The formula for Bayes' theorem is given as: 
$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

**P(A | B) is Posterior probability:** Probability of hypothesis A on the observed event B.

**P(B | A) is Likelihood probability:** Probability of the evidence given that the probability of a hypothesis is true.

**P(A) is Prior Probability:** Probability of hypothesis before observing the evidence.

**P(B) is Marginal Probability:** Probability of Evidence.

# Naïve Bayes Classifier Algorithm

## Working of Naïve Bayes' Classifier:

### Example:

- Suppose we have a dataset of **weather conditions** and corresponding target variable "**Play**". So using this dataset we need to decide that whether we should play or not on a particular day according to the weather conditions.
- **Problem:** If the weather is sunny, then the Player should play or not?
- So to solve this problem, we need to follow the below steps:
  1. Convert the given dataset into frequency tables.
  2. Generate Likelihood table by finding the probabilities of given features.
  3. Now, use Bayes theorem to calculate the posterior probability.

# Naïve Bayes Classifier Algorithm

## Working of Naïve Bayes' Classifier:

### Solution:

- To solve this, first consider the below dataset:
- **Frequency table for the Weather Conditions:**

Weather	Yes	No
Overcast	5	0
Rainy	2	2
Sunny	3	2
Total	10	5

0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

# Naïve Bayes Classifier Algorithm

## Working of Naïve Bayes' Classifier:

### ➤ Likelihood table weather condition:

Weather	No	Yes	
Overcast	0	5	$5/14 = 0.35$
Rainy	2	2	$4/14 = 0.29$
Sunny	2	3	$5/14 = 0.35$
All	$4/14 = 0.29$	$10/14 = 0.71$	

### ➤ Applying Bayes'theorem:

$$P(\text{Yes} | \text{Sunny}) = P(\text{Sunny} | \text{Yes}) * P(\text{Yes}) / P(\text{Sunny})$$

$$P(\text{Sunny} | \text{Yes}) = 3/10 = 0.3$$

$$P(\text{Sunny}) = 0.35$$

$$P(\text{Yes}) = 0.71$$

$$\text{So } P(\text{Yes} | \text{Sunny}) = 0.3 * 0.71 / 0.35 = \mathbf{0.60}$$



# Naïve Bayes Classifier Algorithm

## Working of Naïve Bayes' Classifier:

$$P(\text{No} | \text{Sunny}) = P(\text{Sunny} | \text{No}) * P(\text{No}) / P(\text{Sunny})$$

$$P(\text{Sunny} | \text{NO}) = 2/4 = 0.5$$

$$P(\text{No}) = 0.29$$

$$P(\text{Sunny}) = 0.35$$

$$\text{So } P(\text{No} | \text{Sunny}) = 0.5 * 0.29 / 0.35 = \mathbf{0.41}$$

- So as we can see from the above calculation that
- **$P(\text{Yes} | \text{Sunny}) > P(\text{No} | \text{Sunny})$**
- **Hence on a Sunny day, Player can play the game.**

# Naïve Bayes Classifier Algorithm

## Advantages of Naïve Bayes Classifier:

1. Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
2. It can be used for Binary as well as Multi-class Classifications.
3. It performs well in Multi-class predictions as compared to the other Algorithms.
4. It is the most popular choice for **text classification problems**.

## Disadvantages of Naïve Bayes Classifier:

1. Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

# Naïve Bayes Classifier Algorithm

## Applications of Naïve Bayes Classifier:

1. It is used for **Credit Scoring**.
2. It is used in **medical data classification**.
3. It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
4. It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

# Decision Tree Classification Algorithm

## 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.

Decision Tree is the **hierarchical tree structured algorithm that is used for derived a meaningful output from a variety of inputs.**

The output fetched from this kind of hierarchical arrangement is considered a valuable contribution for producing analytical results for essential business decision-making.

➤ 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.**

# Decision Tree Classification Algorithm

## Decision Tree Classification Algorithm

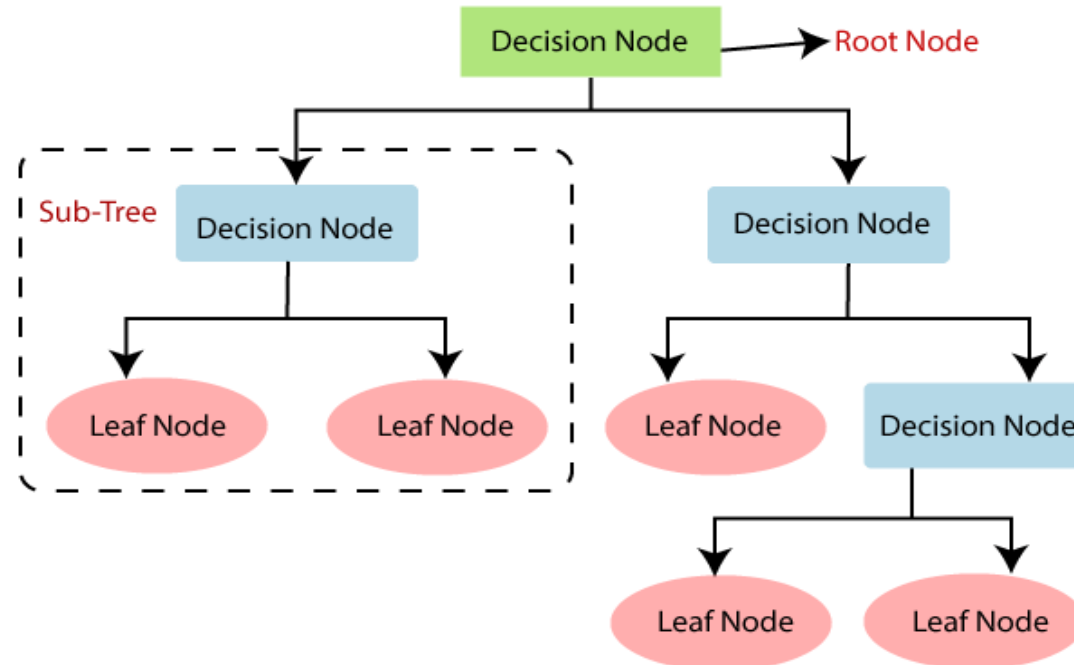
- In a Decision tree, there are **two nodes**, which are the **Decision Node and Leaf Node**. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed 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.**

# Decision Tree Classification Algorithm

## Decision Tree Classification 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:



# Decision Tree Classification Algorithm

## 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.

# Decision Tree Classification Algorithm

## Decision Tree Terminologies

### **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.



# Decision Tree Classification Algorithm

## 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.

# Decision Tree Classification Algorithm

## How does the Decision Tree algorithm Work?

➤ 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:

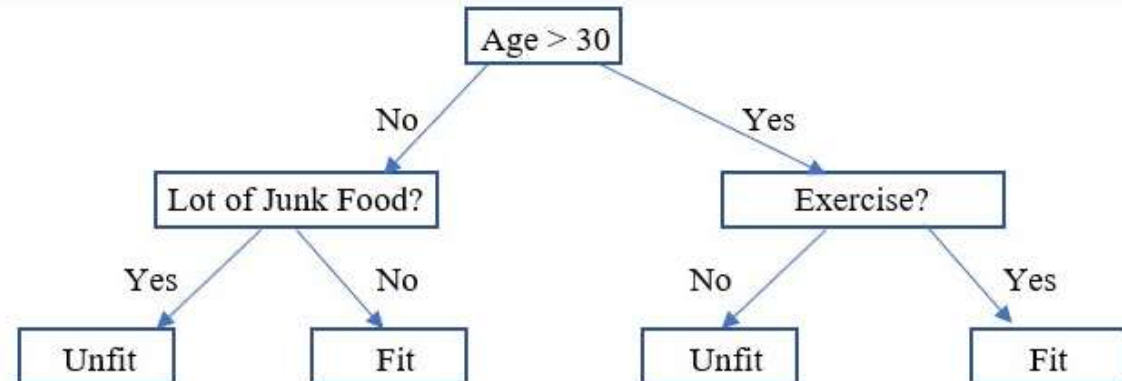
Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

# Decision Tree Classification Algorithm

Example:

Age	A lot of Junk food	Exercise	Fit/Unfit
45	No	No	Unfit
40	No	Yes	Fit
25	Yes	No	Unfit
29	No	Yes	Fit
23	Yes	Yes	Unfit

➤ decision tree for the table mentioned with data



# Decision Tree Classification Algorithm

## 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:

1. **Information Gain**
2. **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

# Decision Tree Classification Algorithm

## Attribute Selection Measures

- It can be calculated using the below formula:

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

### **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**

# Decision Tree Classification Algorithm

## Attribute Selection Measures

### 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:

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

# Decision Tree Classification Algorithm

## 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:
  1. **Cost Complexity Pruning**
  2. **Reduced Error Pruning.**



# Decision Tree Classification Algorithm

## Advantages of the Decision Tree

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

## Disadvantages of the Decision Tree

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

# Random Forest Algorithm

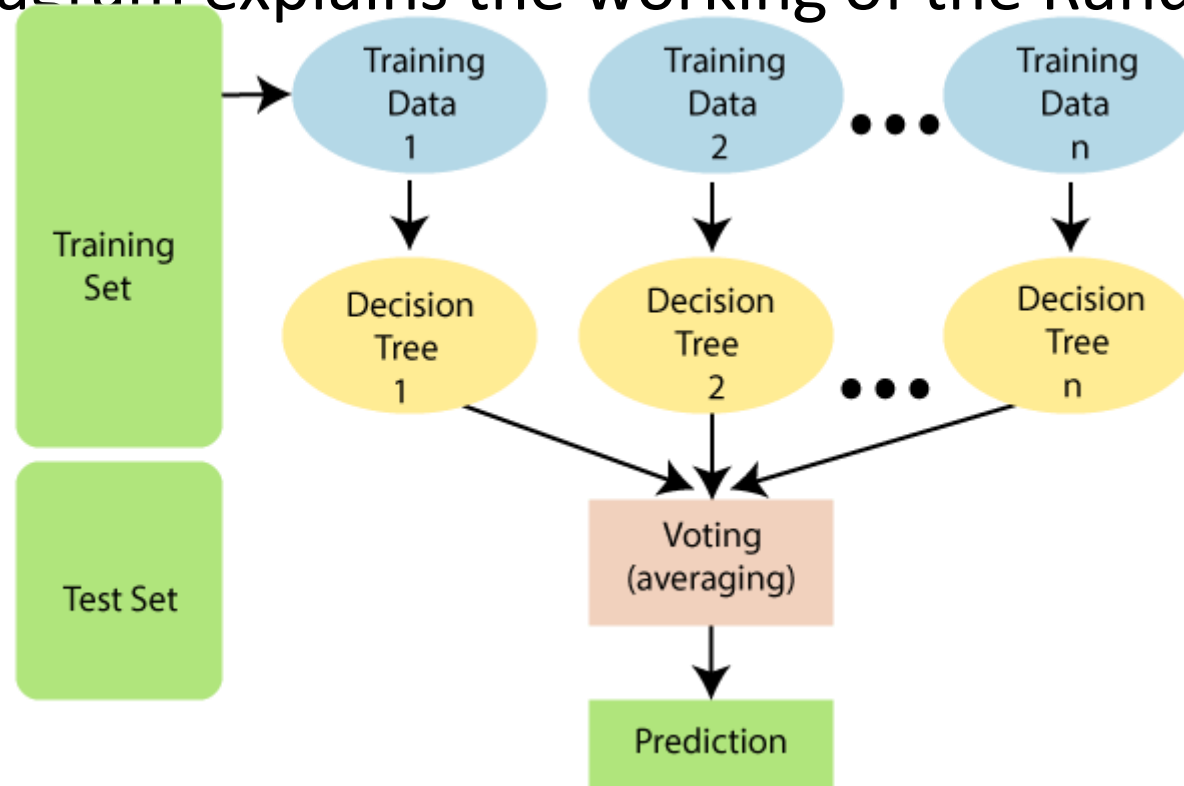
## Random Forest Algorithm

- 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.

# Random Forest Algorithm

## Random Forest Algorithm

- 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:



# Random Forest 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.

# Random Forest Algorithm

## Why use Random Forest?

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

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

# Random Forest Algorithm

## 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).

# Random Forest Algorithm

## How does Random Forest algorithm work?

### **Step-3:**

Choose the number  $N$  for decision trees that you want to build.

### **Step-4:** Repeat Step 1 & 2.

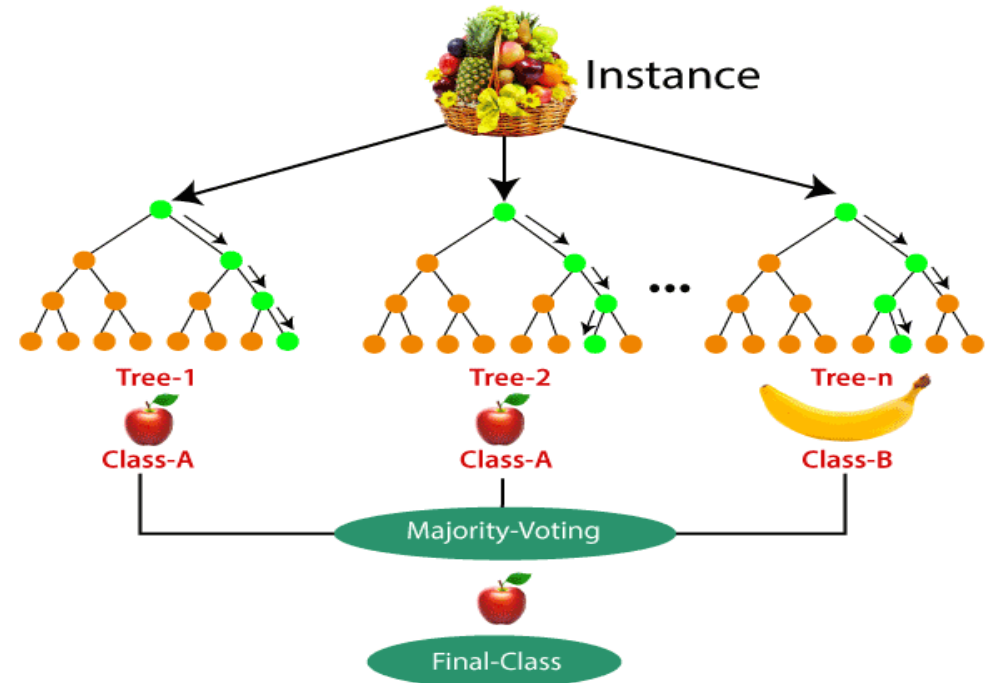
### **Step-5:**

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.

# Random Forest Algorithm

## 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:





# Random Forest Algorithm

## Applications of Random Forest

➤ There are mainly four sectors where Random forest mostly used:

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

**Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.

**Land Use:** We can identify the areas of similar land use by this algorithm.

**Marketing:** Marketing trends can be identified using this algorithm.

# Random Forest Algorithm

## **Advantages of Random Forest**

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

## **Disadvantages of Random Forest**

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