

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

-----

**Group B**

**Assignment No : 1**

-----

**Title of the Assignment:**Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

- 1. Pre-process the dataset.
- 2. Identify outliers.
- 3. Check the correlation.
- 4. Implement linear regression and random forest regression models.
- 5. Evaluate the models and compare their respective scores like R2, RMSE, etc.

**Dataset Description:**The project is about on world's largest taxi company Uber inc. In this project, we're looking to predict the fare for their future transactional cases. Uber delivers service to lakhs of customers daily. Now it becomes really important to manage their data properly to come up with new business ideas to get best results. Eventually, it becomes really important to estimate the fare prices accurately.

**Link for Dataset:**<https://www.kaggle.com/datasets/yasserh/uber-fares-dataset>

**Objective of the Assignment:**

Students should be able to preprocess dataset and identify outliers, to check correlation and implement linear regression and random forest regression models. Evaluate them with respective scores like R2, RMSE etc.

**Prerequisite:**

- 1. Basic knowledge of Python
- 2. Concept of preprocessing data
- 3. Basic knowledge of Data Science and Big Data Analytics.

**Contents of the Theory:**

1. Data Preprocessing
2. Linear regression
3. Random forest regression models
4. Box Plot
5. Outliers
6. Haversine
7. Mathplotlib
8. Mean Squared Error

**Data Preprocessing:**

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put it in a formatted way. So for this, we use data preprocessing task.

**Why do we need Data Preprocessing?**

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

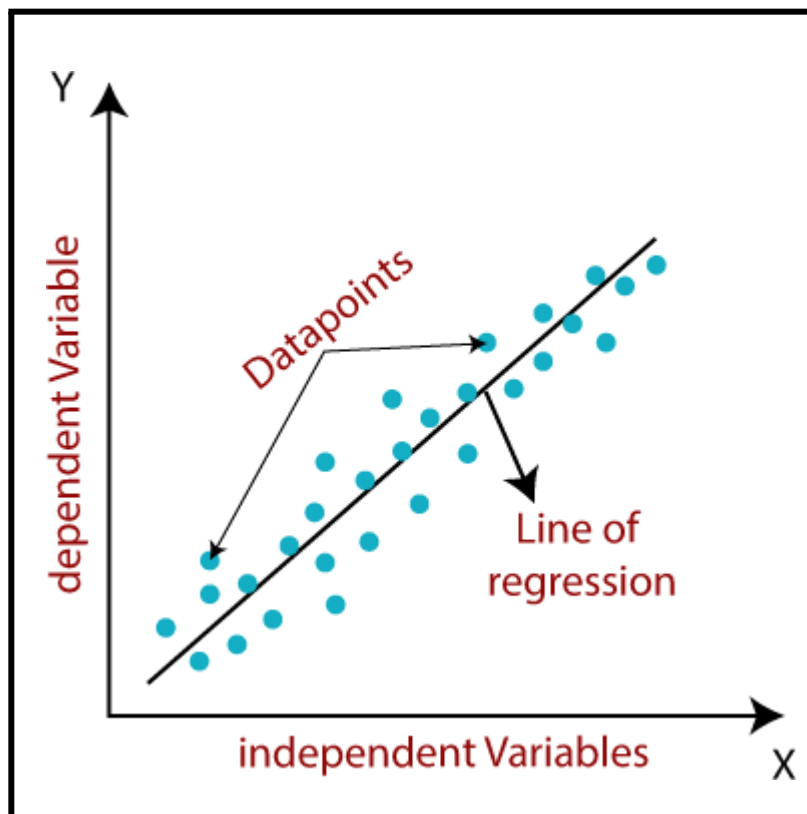
- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling

**Linear Regression:**

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as **sales, salary, age, product price**, etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (x) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it shows how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:

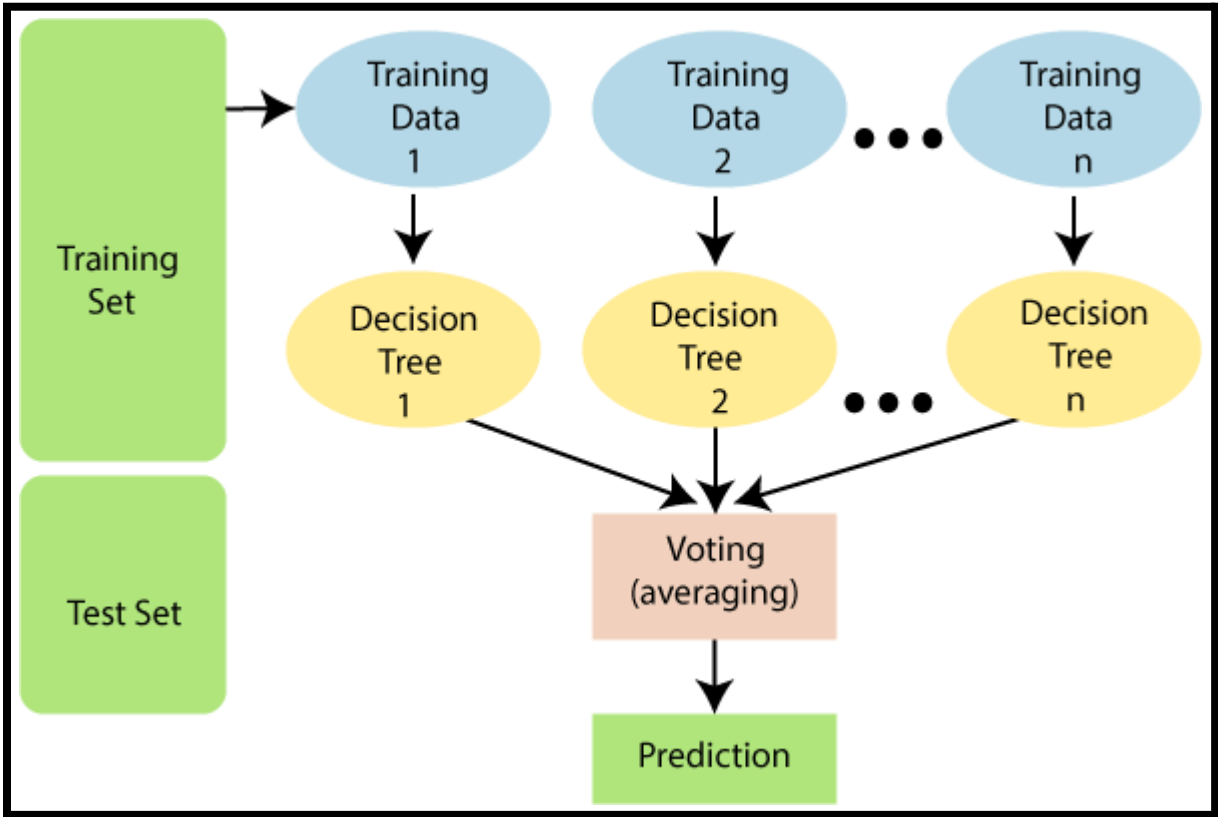
**Random Forest Regression Models:**

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.



**Boxplot:**

Boxplots are a measure of how well data is distributed across a data set. This divides the data set into three quartiles. This graph represents the minimum, maximum, average, first quartile, and the third quartile in the data set. Boxplot is also useful in comparing the distribution of data in a data set by drawing a boxplot for each of them.

R provides a `boxplot()` function to create a boxplot. There is the following syntax of `boxplot()` function:

```
boxplot(x, data, notch, varwidth, names, main)
```

Here,

S.No	Parameter	Description
------	-----------	-------------

1.	x	It is a vector or a formula.
2.	data	It is the data frame.
3.	notch	It is a logical value set as true to draw a notch.
4.	varwidth	It is also a logical value set as true to draw the width of the boxsame as the sample size.
5.	names	It is the group of labels that will be printed under each boxplot.
6.	main	It is used to give a title to the graph.

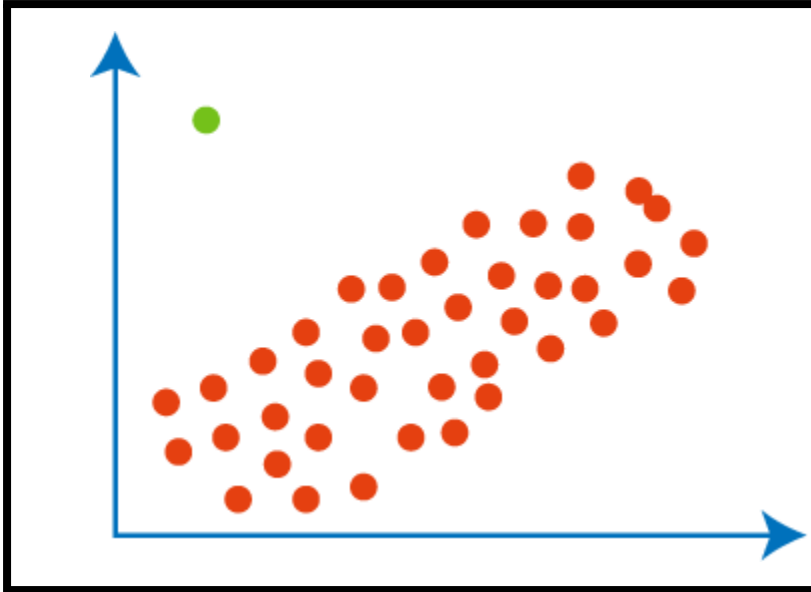
**Outliers:**

As the name suggests, "outliers" refer to the data points that exist outside of what is to be expected. The major thing about the outliers is what you do with them. If you are going to analyze any task to analyze data sets, you will always have some assumptions based on how this data is generated. If you find some data points that are likely to contain some form of error, then these are definitely outliers, and depending on the context, you want to overcome those errors. The data mining process involves the analysis and prediction of data that the data holds. In 1969, Grubbs introduced the standard definition of outliers.



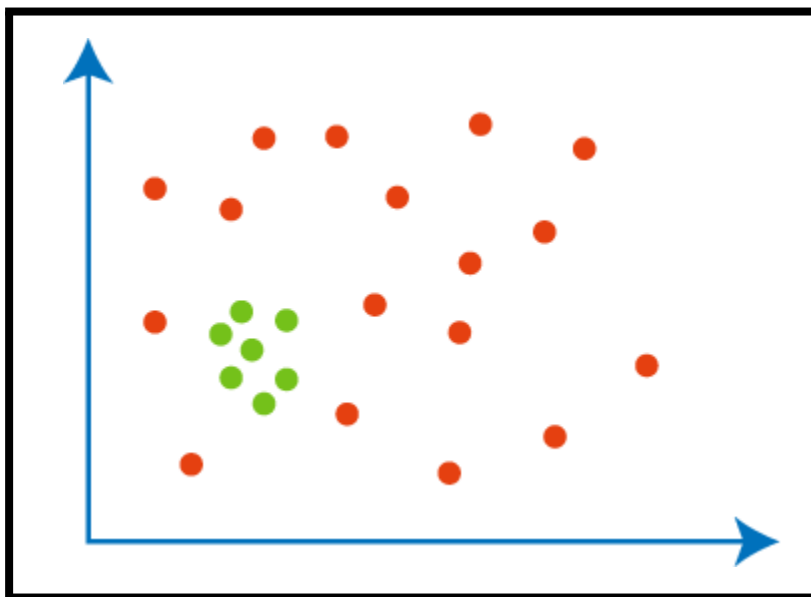
**Global Outliers**

Global outliers are also called point outliers. Global outliers are taken as the simplest form of outliers. When data points deviate from all the rest of the data points in a given data set, it is known as the global outlier. In most cases, all the outlier detection procedures are targeted to determine the global outliers. The green data point is the global outlier.



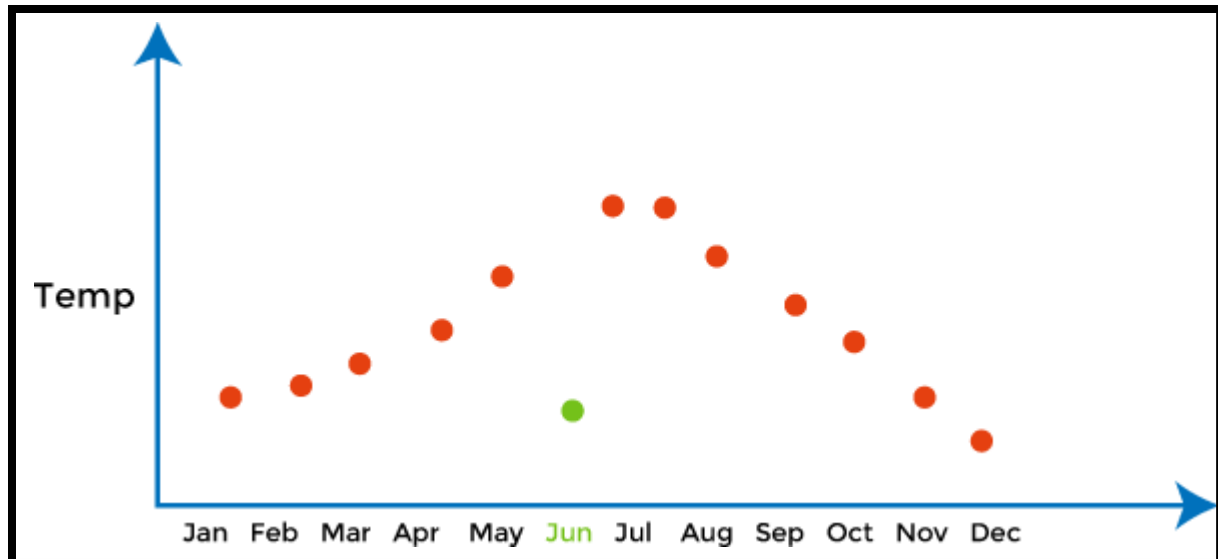
### Collective Outliers

In a given set of data, when a group of data points deviates from the rest of the data set is called collective outliers. Here, the particular set of data objects may not be outliers, but when you consider the data objects as a whole, they may behave as outliers. To identify the types of different outliers, you need to go through background information about the relationship between the behavior of outliers shown by different data objects. For example, in an Intrusion Detection System, the DOS package from one system to another is taken as normal behavior. Therefore, if this happens with the various computer simultaneously, it is considered abnormal behavior, and as a whole, they are called collective outliers. The green data points as a whole represent the collective outlier.



## Contextual Outliers

As the name suggests, "Contextual" means this outlier introduced within a context. For example, in the speech recognition technique, the single background noise. Contextual outliers are also known as Conditional outliers. These types of outliers happen if a data object deviates from the other data points because of any specific condition in a given data set. As we know, there are two types of attributes of objects of data: contextual attributes and behavioral attributes. Contextual outlier analysis enables the users to examine outliers in different contexts and conditions, which can be useful in various applications. For example, A temperature reading of 45 degrees Celsius may behave as an outlier in a rainy season. Still, it will behave like a normal data point in the context of a summer season. In the given diagram, a green dot representing the low-temperature value in June is a contextual outlier since the same value in December is not an outlier.



## Haversine:

The Haversine formula calculates the shortest distance between two points on a sphere using their latitudes and longitudes measured along the surface. It is important for use in navigation.

## Matplotlib:

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

**Mean Squared Error;**

The **Mean Squared Error (MSE)** or **Mean Squared Deviation (MSD)** of an estimator measures the average of error squares i.e. the average squared difference between the estimated values and true value. It is a risk function, corresponding to the expected value of the squared error loss. It is always non-negative and values close to zero are better. The MSE is the second moment of the error (about the origin) and thus incorporates both the variance of the estimator and its bias.

Code :- <https://www.kaggle.com/code/proxzima/uber-fare-price-prediction>

**Conclusion:**

In this way we have explored Concept correlation and implement linear regression and random forest regression models.

**Assignment Questions:**

1. What is data preprocessing?
2. Define Outliers?
3. What is Linear Regression?
4. What is Random Forest Algorithm?
5. Explain: pandas, numpy?



Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

-----

Group B

Assignment No : 2

-----

**Title of the Assignment:**Classify the email using the binary classification method. Email Spam detection has two states:  
a) Normal State - Not Spam,  
b) Abnormal State - Spam.  
Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance.

**Dataset Description:**The csv file contains 5172 rows, each row for each email. There are 3002 columns. The first column indicates Email name. The name has been set with numbers and not recipients' name to protect privacy. The last column has the labels for prediction : 1 for spam, 0 for not spam. The remaining 3000 columns are the 3000 most common words in all the emails, after excluding the non-alphabetical characters/words. For each row, the count of each word(column) in that email(row) is stored in the respective cells. Thus, information regarding all 5172 emails are stored in a compact dataframe rather than as separate text files.

**Link:**<https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv>

**Objective of the Assignment:**

Students should be able to classify email using the binary Classification and implement email spam detection technique by using K-Nearest Neighbors and Support Vector Machine algorithm.

**Prerequisite:**

- 1. Basic knowledge of Python

## 2. Concept of K-Nearest Neighbors and Support Vector Machine for classification.

### Contents of the Theory:

1. Data Preprocessing
2. Binary Classification
3. K-Nearest Neighbours
4. Support Vector Machine
5. Train, Test and Split Procedure

### Data Preprocessing:

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put it in a formatted way. So for this, we use data preprocessing task.

### Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling

Code :- <https://www.kaggle.com/code/mfaisalqureshi/email-spam-detection-98-accuracy/notebook>

Binary classification is a supervised machine learning task where the goal is to assign each data point to one of two classes or categories. Classification into one of two classes is a common machine learning problem. You might want to predict whether or not a customer is likely to make a purchase, whether or not a credit card transaction was fraudulent, whether deep space signals show evidence of a new planet, or a medical test evidence of a disease.

These are all binary classification problems.

Two common algorithms used for binary classification are K-Nearest Neighbors (K-NN) and Support Vector Machine (SVM). Here's an overview of these algorithms and the train-test split procedure:

### 1. K-Nearest Neighbors (K-NN):

- K-NN is a simple classification algorithm that makes predictions based on the majority class of the k-nearest data points to a given test point. To use K-NN for binary classification, you need labeled data with two classes (e.g., positive and negative).
- K-NN is a lazy learner because it doesn't build a specific model during the training phase; it memorizes the training data.
- The choice of 'k' is a hyperparameter that can significantly affect the model's performance. A small 'k' may make the model sensitive to noise, while a large 'k' might lead to oversmoothing. K-NN can be computationally expensive for large datasets, as it requires calculating distances to all data points.
- Feature scaling is essential because K-NN relies on distance calculations. Features with different scales can bias the results. K-NN works well for simple and small to medium-sized datasets but may not be suitable for high-dimensional data or datasets with imbalanced class distributions.
- It's important to choose an appropriate distance metric based on the nature of your data and the problem you're trying to solve. K-NN is a versatile algorithm, and while it has its limitations, it can be a useful tool for various classification tasks, especially when the dataset is well-preprocessed, and the choice of 'k' is well-tuned.
- The steps for using K-NN for binary classification are as follows:
  - a. Collect and preprocess your data.
  - b. Split the data into a training set and a test set.
  - c. Choose a value for 'k' (the number of nearest neighbors to consider).
  - d. Train the K-NN model on the training data.

- e. Use the trained model to make predictions on the test data.
- f. Evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

## 2. Support Vector Machine (SVM):

- SVM is a powerful classification algorithm that finds a hyperplane that best separates the two classes in a way that maximizes the margin between them.
- SVM is effective in cases where the data is not linearly separable by transforming it into a higher-dimensional space. The choice of kernel and its hyperparameters can have a significant impact on the SVM's performance. SVM is capable of finding a decision boundary that maximizes the margin, making it robust against overfitting.
- SVM can handle both balanced and imbalanced datasets. SVM can be computationally intensive, especially for large datasets. Techniques like stochastic gradient descent and linear SVMs may be used for efficiency. Support vectors are essential for defining the decision boundary, so they play a crucial role in the SVM's performance. Support Vector Machines are a versatile algorithm that works well in a wide range of applications, including image classification, text classification, and anomaly detection, among others.
- Steps for using SVM for binary classification:
  - a. Collect and preprocess your data.
  - b. Split the data into a training set and a test set.
  - c. Choose a suitable kernel (e.g., linear, polynomial, or radial basis function) and kernel parameters.
  - d. Train the SVM model on the training data.
  - e. Use the trained model to make predictions on the test data.
  - f. Evaluate the model's performance, typically using metrics like accuracy, precision, recall, and F1-score.

## 3. Train-Test Split Procedure:

- The train-test split is a critical step in machine learning to assess the model's performance. It involves dividing the dataset into two parts: a training set and a test set.
- The training set is used to train the machine learning model, while the test set is used to evaluate the model's performance and generalization.
- The steps for the train-test split procedure are as follows:

- a. Randomly shuffle the dataset to ensure that the data is evenly distributed between the training and test sets.
- b. Split the dataset into two parts, typically with a ratio like 70-30 or 80-20, where the training set gets the larger portion.
- c. Ensure that the two sets are mutually exclusive, meaning that no data point appears in both sets.
- d. Use the training set to train the machine learning model.
- e. Use the test set to evaluate the model's performance by making predictions and comparing them to the actual labels.
- f. Calculate various evaluation metrics to assess the model's accuracy and generalization, such as accuracy, precision, recall, and F1-score.

The train-test split procedure helps you estimate how well your model is likely to perform on new, unseen data and avoid overfitting, where the model performs well on the training data but poorly on new data.

### **Conclusion**

Hence in this way we have studied the Concept of K-Nearest Neighbors and Support Vector Machine for classification.

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group B

Assignment No : 3

**Title of the Assignment:**Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months

**Dataset Description:**The case study is from an open-source dataset from Kaggle. The dataset contains 10,000 sample points with 14 distinct features such as CustomerId, CreditScore, Geography, Gender, Age, Tenure, Balance, etc.

**Link for Dataset:**<https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling>

Perform the following steps:

1. Read the dataset.
2. Distinguish the feature and target set and divide the data set into training and test sets.
3. Normalize the train and test data.
4. Initialize and build the model. Identify the points of improvement and implement the same.
5. Print the accuracy score and confusion matrix (5 points).

Objective of the Assignment:

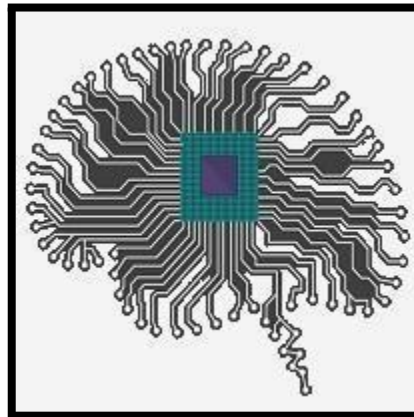
Students should be able to distinguish the feature and target set and divide the data set into training and test sets and normalize them and students should build the model on the basis of that.

Prerequisite:

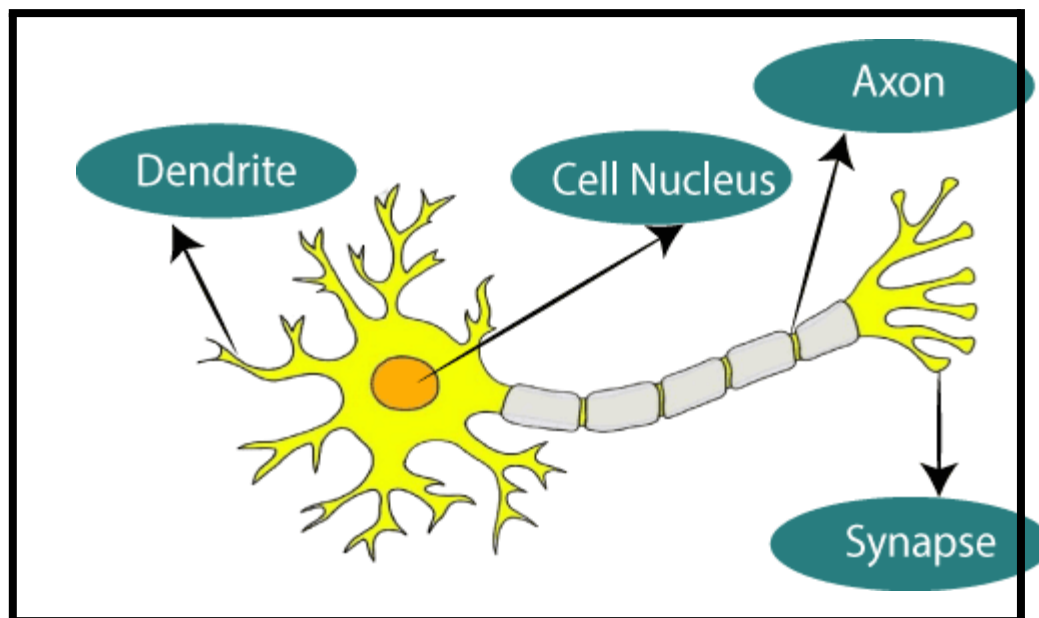
1. Basic knowledge of Python
2. Concept of Confusion Matrix

1. Artificial Neural Network
2. Keras
3. tensorflow
4. Normalization
5. Confusion Matrix

**Artificial Neural Network:**

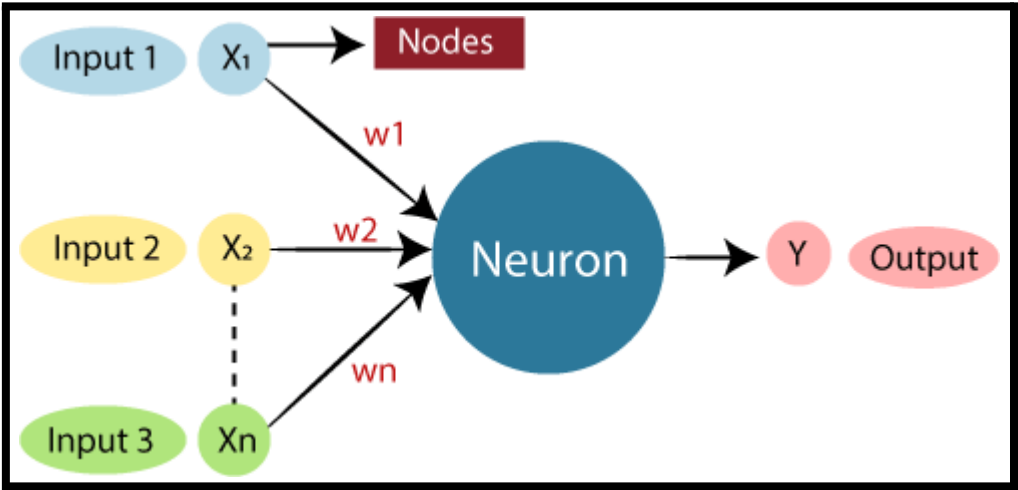


The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.



The given figure illustrates the typical diagram of Biological Neural Network.

The typical Artificial Neural Network looks something like the given figure.



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

Relationship between Biological neural network and artificial neural network:

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells.



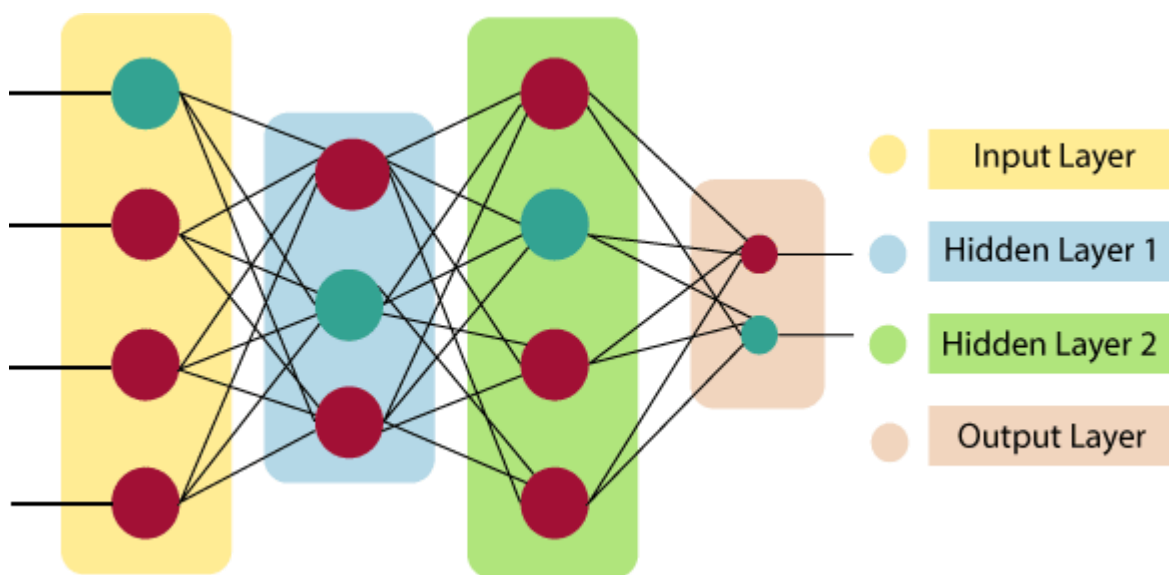
There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed, and we can extract more than one piece of this data when necessary from our memory parallelly. We can say that the human brain is made up of incredibly amazing parallel processors.

We can understand the artificial neural network with an example, consider an example of a digital logic gate that takes an input and gives an output. "OR" gate, which takes two inputs. If one or both the inputs are "On," then we get "On" in output. If both the inputs are "Off," then we get "Off" in output. Here the output depends upon input. Our brain does not perform the same task. The outputs to inputs relationship keep changing because of the neurons in our brain, which are "learning."

### The architecture of an artificial neural network:

To understand the concept of the architecture of an artificial neural network, we have to understand what a neural network consists of. In order to define a neural network that consists of a large number of artificial neurons, which are termed units arranged in a sequence of layers. Let us look at various types of layers available in an artificial neural network. □

Artificial Neural Network primarily consists of three layers:



#### Input Layer:

As the name suggests, it accepts inputs in several different formats provided by the programmer.

#### Hidden Layer:

The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.

### Output Layer:

The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^n W_i * X_i + b$$

It determines weighted total is passed as an input to an activation function to produce the output. Activation functions choose whether a node should fire or not. Only those who are fired make it to the output layer. There are distinctive activation functions available that can be applied upon the sort of task we are performing.

### Keras:

Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

It cannot handle low-level computations, so it makes use of the **Backend** library to resolve it. The backend library acts as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

Initially, it had over 4800 contributors during its launch, which now has gone up to 250,000 developers. It has a 2X growth ever since every year it has grown. Big companies like Microsoft, Google, NVIDIA, and Amazon have actively contributed to the development of Keras. It has an amazing industry interaction, and it is used in the development of popular firms like Netflix, Uber, Google, Expedia, etc.

**TensorFlow:**

TensorFlow is a Google product, which is one of the most famous deep learning tools widely used in the research area of machine learning and deep neural network. It came into the market on 9<sup>th</sup> November 2015 under the Apache License 2.0. It is built in such a way that it can easily run on multiple CPUs and GPUs as well as on mobile operating systems. It consists of various wrappers in distinct languages such as Java, C++, or Python.

**Normalization:**

Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

Mathematically, we can calculate normalization with the below formula:  $X_n = (X - X_{\text{minimum}}) / (X_{\text{maximum}} - X_{\text{minimum}})$

$$X_{\text{minimum}}) / (X_{\text{maximum}} - X_{\text{minimum}})$$

Where,

- $X_n$  = Value of Normalization
- $X_{\text{maximum}}$  = Maximum value of a feature
- $X_{\text{minimum}}$  = Minimum value of a feature

**Example:**Let's assume we have a model dataset having maximum and minimum values of feature as mentioned above. To normalize the machine learning model, values are shifted and rescaled so their range can vary between 0 and 1. This technique is also known as Min-Max scaling. In this scaling technique, we will change the feature values as follows:

**Case1-**If the value of X is minimum, the value of Numerator will be 0; hence Normalization will also be 0.

$X_n = (X - X_{\text{minimum}}) / (X_{\text{maximum}} - X_{\text{minimum}})$  ----- formula

Put  $X = X_{\text{minimum}}$  in above formula, we get;

$X_n = X_{\text{minimum}} - X_{\text{minimum}} / (X_{\text{maximum}} - X_{\text{minimum}})$   
 $X_n = 0$

**Case2-**If the value of X is maximum, then the value of the numerator is equal to the denominator; hence Normalization will be 1.

$X_n = (X - X_{\text{minimum}}) / (X_{\text{maximum}} - X_{\text{minimum}})$  Put  $X$

$= X_{\text{maximum}}$  in above formula, we get;

$X_n = X_{\text{maximum}} - X_{\text{minimum}} / (X_{\text{maximum}} - X_{\text{minimum}})$   
 $X_n = 1$

**Case3-**On the other hand, if the value of X is neither maximum nor minimum, then values of normalization will also be between 0 and 1.

Hence, Normalization can be defined as a scaling method where values are shifted and rescaled to maintain their ranges between 0 and 1, or in other words; it can be referred to as Min-Max scaling technique.

### Normalization techniques in Machine Learning

Although there are so many feature normalization techniques in Machine Learning, few of them are most frequently used. These are as follows:

- **Min-Max Scaling:** This technique is also referred to as scaling. As we have already discussed above,

the Min-Max scaling method helps the dataset to shift and rescale the values of their attributes, so they end up ranging between 0 and 1.

- **Standardization scaling:**

Standardization scaling is also known as **Z-score** normalization, in which values are centered around the mean with a unit standard deviation, which means the attribute becomes zero and the resultant distribution has a unit standard deviation. Mathematically, we can calculate the standardization by subtracting the feature value from the mean and dividing it by standard deviation.

Hence, standardization can be expressed as follows:

$$X' = \frac{X - \mu}{\sigma}$$

Here,  $\mu$  represents the mean of feature value, and  $\sigma$  represents the standard deviation of feature values.

However, unlike Min-Max scaling technique, feature values are not restricted to a specific range in the standardization technique.

This technique is helpful for various machine learning algorithms that use distance measures such as **KNN, K-means clustering, and Principal component analysis**, etc. Further, it is also important that the model is built on assumptions and data is normally distributed.

#### When to use Normalization or Standardization?

Which is suitable for our machine learning model, Normalization or Standardization? This is probably a big confusion among all data scientists as well as machine learning engineers. Although both terms have the almost same meaning choice of using normalization or standardization will depend on your problem and the algorithm you are using in models.

1. Normalization is a transformation technique that helps to improve the performance as well as the accuracy of your model better. Normalization of a machine learning model is useful when you don't know feature distribution exactly. In other words, the feature distribution of data does not follow a **Gaussian** (bell curve) distribution. Normalization must

have an abounding range, so if you have outliers in data, they will be affected by Normalization.

Further, it is also useful for data having variable scaling techniques such as **KNN, artificial neural networks**. Hence, you can't use assumptions for the distribution of data.

2. Standardization in the machine learning model is useful when you are exactly aware of the feature distribution of data or, in other words, your data follows a Gaussian distribution. However, this does not have to be necessarily true. Unlike Normalization, Standardization does not necessarily have a bounding range, so if you have outliers in your data, they will not be affected by Standardization.

Further, it is also useful when data has variable dimensions and techniques such as **linear regression, logistic regression, and linear discriminant analysis**.

**Example:** Let's understand an experiment where we have a dataset having two attributes, i.e., age and salary. Where the age ranges from 0 to 80 years old, and the income varies from 0 to 75,000 dollars or more. Income is assumed to be 1,000 times that of age. As a result, the ranges of these two attributes are much different from one another.

Because of its bigger value, the attributed income will organically influence the conclusion more when we undertake further analysis, such as multivariate linear regression. However, this does not necessarily imply that it is a better predictor. As a result, we normalize the data so that all of the variables are in the same range.

Further, it is also helpful for the prediction of credit risk scores where normalization is applied to all numeric data except the class column. It uses the **tanh transformation** technique, which converts all numeric features into values of range between 0 to 1.

**Confusion Matrix:**

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an **error matrix**. Some features of Confusion matrix are given below:

- For the 2 prediction classes of classifiers, the matrix is of 2\*2 table, for 3 classes, it is 3\*3 table, and so on.
- The matrix is divided into two dimensions, that are **predicted values** and **actual values** along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations.
- It looks like the below table:

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

The above table has the following cases.

- **True Negative:**Model has given prediction No, and the real or actual value was also No.
- **True Positive:**The model has predicted yes, and the actual value was also true.
- **False Negative:**The model has predicted no, but the actual value was Yes, it is also called as **Type-II error**.
- **False Positive:**The model has predicted Yes, but the actual value was No. It is also called a **Type-I error**.

Need for Confusion Matrix in Machine learning

- It evaluates the performance of the classification models, when they make predictions on test data, and tells how good our classification model is.
- It not only tells the error made by the classifiers but also the type of errors such as it is either type- I or type-II error.
- With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

**Example:** We can understand the confusion matrix using an example.

Suppose we are trying to create a model that can predict the result for the disease that is either a person has that disease or not. So, the confusion matrix for this is given as:

n = 100	Actual: No	Actual: Yes	
Predicted: No	TN: 65	FP: 3	68
Predicted: Yes	FN: 8	TP: 24	32
	73	27	

From the above example, we can conclude that:



- The table is given for the two-class classifier, which has two predictions "Yes" and

"NO." Here, Yes defines that patient has the disease, and No defines that patient doesnot has that disease.

- The classifier has made a total of **100 predictions**. Out of 100 predictions, **89 are true predictions**, and **11 are incorrect predictions**.
- The model has given prediction "yes" for 32 times, and "No" for 68 times. Whereas the actual "Yes" was 27, and actual "No" was 73 times.

#### Calculations using Confusion Matrix:

We can perform various calculations for the model, such as the model's accuracy, using this matrix. These calculations are given below:

- **Classification Accuracy:** It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be

Department of Computer Engineering and predictions made by Course Laboratory Practice of  
predictions made by the classifiers. The formula is given below:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+FN+TN}$$

- **Misclassification rate:** It is also termed as Error rate, and it defines how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect

$$\text{Error rate} = \frac{FP+FN}{TP+FP+FN+TN}$$

predictions to all number of the predictions made by the classifier. The formula is given below:

- **Precision:** It can be defined as the number of correct outputs provided by the model or out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

$$\text{Precision} = \frac{TP}{TP+FP}$$

- **Recall:** It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\text{Recall} = \frac{TP}{TP+FN}$$

- **F-measure:** If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\text{F-measure} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

Other important terms used in Confusion Matrix:

- **Null Error rate:** It defines how often our model would be incorrect if it always predicted the



- **ROC Curve:** The ROC is a graph displaying a classifier's performance for all possible thresholds. The graph is plotted between the true positive rate (on the Y-axis) and the false Positive rate (on the x-axis).

Code :- <https://www.kaggle.com/code/jaysadguru00/starter-bank-customer-churn-modeling-6dbfe05e-a>

**Conclusion:**

In this way we build a neural network-based classifier and determine whether they will leave or not in next 6 months

- 1) What is Normalization?
- 2) What is Standardization?
- 3) Explain Confusion Matrix ?
- 4) Define the following: Classification Accuracy, Misclassification Rate, Precision.
- 5) One Example of Confusion Matrix?



Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

### Group B ASSIGNMENT NO: 4

**Title:** Implement Gradient Descent Algorithm to find the local minima of a function. For example, find the local minima of the function  $y=(x+3)^2$  starting from the point  $x=2$ .

**Objective:** Students should learn to implement Gradient Descent Algorithm.

**Prerequisite:**

1. Basic of Python Language
2. Concept of Gradient Descent Algorithm

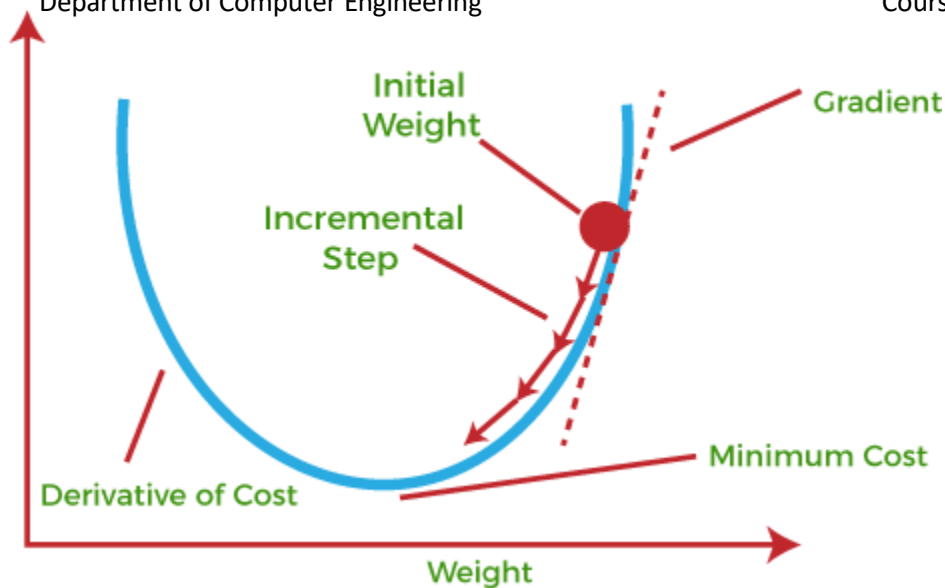
**Theory:**

Gradient descent was initially discovered by "Augustin-Louis Cauchy" in mid of 18th century. Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the local minimum of that function.

Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the local maximum of that function.



This entire procedure is known as Gradient Ascent, which is also known as steepest descent. The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.

### Types of Gradient Descent

Based on the error in various training models, the Gradient Descent learning algorithm can be divided into **Batch gradient descent, stochastic gradient descent, and mini-batch gradient descent**. Let's understand these different types of gradient descent:

#### 1. Batch Gradient Descent:

Batch gradient descent (BGD) is used to find the error for each point in the training set and update the model after evaluating all training examples. This procedure is known as the training epoch. In simple words, it is a greedy approach where we have to sum over all examples for each update.

#### Advantages of Batch gradient descent:

- It produces less noise in comparison to other gradient descent.
- It produces stable gradient descent convergence.
- It is Computationally efficient as all resources are used for all training samples.

#### 2. Stochastic gradient descent

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration. Or in other words, it processes a training epoch for each example within a dataset and updates each training example's parameters one at a time. As it requires only one training example at a time, hence it is easier to store in allocated memory. However, it shows some computational efficiency losses in comparison to batch gradient systems as it shows frequent updates that require more detail and speed. Further, due to frequent updates, it is also treated as a noisy gradient.

However, sometimes it can be helpful in finding the global minimum and also escaping the local minimum.

**Advantages of Stochastic gradient descent:**

In Stochastic gradient descent (SGD), learning happens on every example, and it consists of a few advantages over other gradient descent.

- It is easier to allocate in desired memory.
- It is relatively fast to compute than batch gradient descent.
- It is more efficient for large datasets.

**3. MiniBatch Gradient Descent:**

Mini Batch gradient descent is the combination of both batch gradient descent and stochastic gradient descent. It divides the training datasets into small batch sizes then performs the updates on those batches separately. Splitting training datasets into smaller batches make a balance to maintain the computational efficiency of batch gradient descent and speed of stochastic gradient descent.

Hence, we can achieve a special type of gradient descent with higher computational efficiency and less noisy gradient descent.

**Advantages of Mini Batch gradient descent:**

- It is easier to fit in allocated memory.
- It is computationally efficient.
- It produces stable gradient descent convergence. Here is

the basic algorithm for Gradient Descent:

**Gradient Descent Algorithm Input:**

- Initial parameters  $\theta$  (often initialized randomly)
- Learning rate  $\alpha$  (controls the step size)
- Cost function  $J(\theta)$  (a function that you want to minimize)
- Convergence threshold  $\epsilon$  (a small value that determines when to stop)

**Output:**

- Optimized parameters  $\theta$  that minimize the cost function

**Procedure:**

1. Initialize parameters: Set  $\theta$  to an initial value, often randomly. Initialize a variable to keep track of the previous cost,  $J_{\text{prev}}$ , with a large value (e.g.,  $+\infty$ ).

2. Loop until convergence: a. Compute the gradient of the cost function with respect to the parameters:  $\nabla J(\theta)$ . b. Update the parameters  $\theta$ :  $\theta = \theta - \alpha * \nabla J(\theta)$  This update is performed for each parameter  $\theta_i$ :  
 $\theta_i = \theta_i - \alpha * \partial J(\theta) / \partial \theta_i$
3. Calculate the new cost:  $J_{\text{new}} = J(\theta)$ .
4. Check for convergence: If  $|J_{\text{new}} - J_{\text{prev}}| < \epsilon$ , where  $\epsilon$  is the convergence threshold, stop the algorithm. The algorithm has converged, and  $\theta$  is the optimized parameter set. Otherwise, set  $J_{\text{prev}} = J_{\text{new}}$  and go back to step 2.

**Hyperparameters:**

- Learning Rate ( $\alpha$ ): The learning rate controls the step size in each update. Choosing an appropriate learning rate is crucial for the convergence and stability of the algorithm. It may require some experimentation.
- Convergence Threshold ( $\epsilon$ ): The convergence threshold determines when to stop the algorithm. Setting it too low may result in longer training times, while setting it too high may result in suboptimal solutions.

**Additional Considerations:**

- Gradient Descent can be sensitive to the choice of the learning rate, and various techniques like learning rate schedules or adaptive learning rates (e.g., Adam, RMSprop) can be used to improve convergence.
- Mini-batch Gradient Descent and Stochastic Gradient Descent are variants of this algorithm that use subsets of the training data in each iteration, which can be more computationally efficient.
- Regularization terms can be added to the cost function to prevent overfitting.
- Make sure the cost function is differentiable with respect to the parameters  $\theta$  for this algorithm to work.

Gradient Descent is a foundational optimization algorithm and serves as the basis for many advanced optimization techniques used in machine learning and deep learning.

**Conclusion:**

Hence by this way we have successfully implemented GDA.

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group B

Assignment No : 5

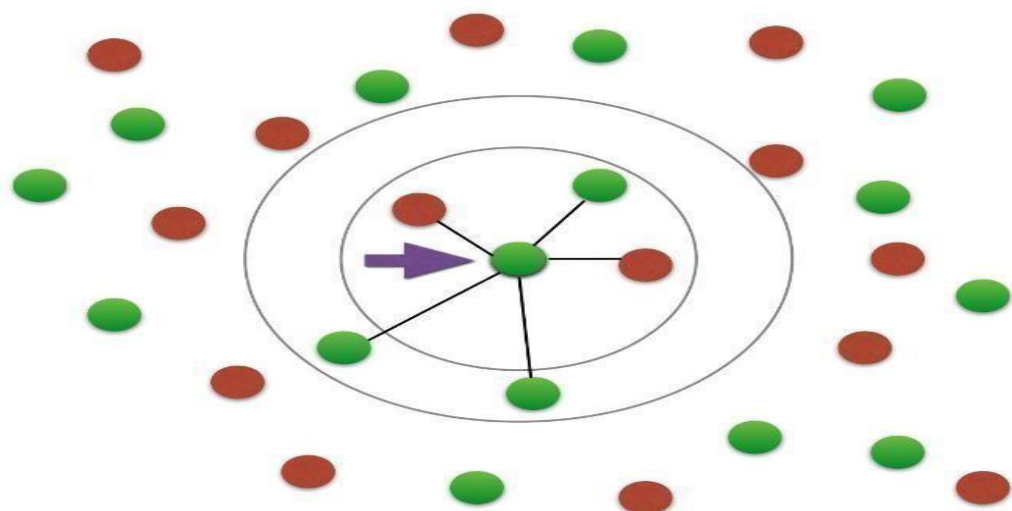
**Title of the Assignment:** Implement K-Nearest Neighbors algorithm on diabetes.csv dataset.  
Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

**Dataset Description:** We will try to build a machine learning model to accurately predict whether or not the patients in the dataset have diabetes or not?  
The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

**Link for Dataset:** [Diabetes predication system with KNN algorithm | Kaggle](#)

**Objective of the Assignment:**  
Students should be able to preprocess dataset and identify outliers, to check correlation and implement KNN algorithm and random forest classification models. Evaluate them with respective scores like confusion\_matrix, accuracy\_score, mean\_squared\_error, r2\_score, roc\_auc\_score, roc\_curve etc.

- Prerequisite:**
- 1. Basic knowledge of Python
  - 2. Concept of Confusion Matrix
  - 3. Concept of roc\_auc curve.
  - 4. Concept of Random Forest and KNN algorithms



k-Nearest-Neighbors (k-NN) is a supervised machine learning model. Supervised learning is when a model learns from data that is already labeled. A supervised learning model takes in a set of input objects and output values. The model then trains on that data to learn how to map the inputs to the desired output so it can learn to make predictions on unseen data.

k-NN models work by taking a data point and looking at the `_k_` closest labeled data points. The data point is then assigned the label of the majority of the `_k_` closest points.

For example, if `k = 5`, and 3 of points are `_green_` and 2 are `_red_`, then the data point in question would be labeled `_green_`, since `_green_` is the majority (as shown in the above graph).

Scikit-learn is a machine learning library for Python. In this tutorial, we will build a k-NN model using Scikit-learn to predict whether or not a patient has diabetes.

Reading in the training data

For our k-NN model, the first step is to read in the data we will use as input. For this example, we are using the diabetes dataset. To start, we will use Pandas to read in the data. I will not go into detail on Pandas, but it is a library you should become familiar with if you're looking to dive further into data science and machine learning.

```
import pandas as pd#read in the data using pandas
df = pd.read_csv(_data/diabetes_data.csv')#check data has been read in properly
df.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age	diabetes
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Next, let's see how much data we have. We will call the `_shape_` function on our dataframe to see how many rows and columns there are in our data. The rows indicate the number of patients and the columns indicate the number of features (age, weight, etc.) in the dataset for each patient.

1

```
#check number of rows and columns in dataset
df.shape
```

Op → (768,9)

We can see that we have 768 rows of data (potential diabetes patients) and 9 columns (8 input features and 1 target output).

Split up the dataset into inputs and targets

Now let's split up our dataset into inputs (X) and our target (y). Our input will be every column except `_diabetes'` because `_diabetes'` is what we will be attempting to predict. Therefore, `_diabetes'` will be our target.

We will use pandas `_drop'` function to drop the column `_diabetes'` from our dataframe and store it in the variable `_X'`. This will be our input.

```
#create a dataframe with all training data except the target column
X = df.drop(columns=[_diabetes'])#check that the target variable has been removed
X.head()
```

	pregnancies	glucose	diastolic	triceps	insulin	bmi	dpf	age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

We will insert the `_diabetes'` column of our dataset into our target variable (y).

```
#separate target values
y = df[_diabetes'].values#view target values
y[0:5]
```

```
array([1, 0, 1, 0, 1])
```

Split the dataset into train and test data

Now we will split the dataset into training data and testing data. The training data is the data that the model will learn from. The testing data is the data we will use to see how well the model performs on unseen data.

Scikit-learn has a function we can use called `_train_test_split` that makes it easy for us to split our dataset into training and testing data.

```
from sklearn.model_selection import train_test_split#split dataset into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1, stratify=y)
```

`_train_test_split` takes in 5 parameters. The first two parameters are the input and target data we split up earlier. Next, we will set `_test_size` to 0.2. This means that 20% of all the data will be used for testing, which leaves 80% of the data as training data for the model to learn from. Setting `_random_state` to 1 ensures that we get the same split each time so we can reproduce our results.

Setting `_stratify` to `y` makes our training split represent the proportion of each value in the `y` variable. For example, in our dataset, if 25% of patients have diabetes and 75% don't have diabetes, setting

`_stratify` to `y` will ensure that the random split has 25% of patients with diabetes and 75% of patients without diabetes.

Building and training the model

Next, we have to build the model. Here is the code:

```
from sklearn.neighbors import KNeighborsClassifier# Create KNN classifier
knn = KNeighborsClassifier(n_neighbors = 3)# Fit the classifier to the data
knn.fit(X_train,y_train)
```

First, we will create a new k-NN classifier and set `_n_neighbors` to 3. To recap, this means that if at least 2 out of the 3 nearest points to a new data point are patients without diabetes, then the new data point will be labeled as `_no diabetes`, and vice versa. In other words, a new data point is labeled with by majority from the 3 nearest points.

We have set `_n_neighbors` to 3 as a starting point. We will go into more detail below on how to better select a value for `_n_neighbors` so that the model can improve its performance.



Next, we need to train the model. In order to train our new model, we will use the `_fit` function and pass in our training data as parameters to fit our model to the training data.

### Testing the model

Once the model is trained, we can use the `_predict` function on our model to make predictions on our test data. As seen when inspecting `_y` earlier, 0 indicates that the patient does not have diabetes and 1 indicates that the patient does have diabetes. To save space, we will only show print the first 5 predictions of our test set.

```
#show first 5 model predictions on the test data  
knn.predict(X_test)[0:5]
```

```
array([0, 0, 0, 0, 1])
```

We can see that the model predicted `_no diabetes` for the first 4 patients in the test set and `_has diabetes` for the 5th patient.

Now let's see how accurate our model is on the full test set. To do this, we will use the `_score` function and pass in our test input and target data to see how well our model predictions match up to the actual results.

```
#check accuracy of our model on the test data  
knn.score(X_test, y_test)
```

```
0.66883116883116878
```

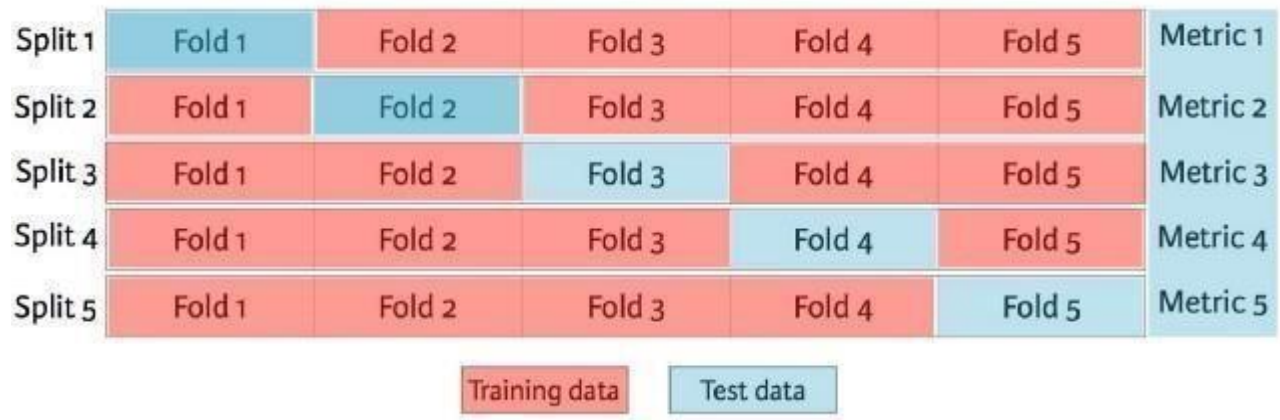
Our model has an accuracy of approximately 66.88%. It's a good start, but we will see how we can increase model performance below.

Congrats! You have now built an amazing k-NN model!

### k-Fold Cross-Validation

Cross-validation is when the dataset is randomly split up into `_k` groups. One of the groups is used as the test set and the rest are used as the training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group has been used as the test set.

For example, for 5-fold cross validation, the dataset would be split into 5 groups, and the model would be trained and tested 5 separate times so each group would get a chance to be the test set. This can be seen in the graph below.



5-fold cross validation (image credit)

The train-test-split method we used in earlier is called `_holdout`. Cross-validation is better than using the holdout method because the holdout method score is dependent on how the data is split into train and test sets. Cross-validation gives the model an opportunity to test on multiple splits so we can get a better idea on how the model will perform on unseen data.

In order to train and test our model using cross-validation, we will use the `_cross_val_score` function with a cross-validation value of 5. `_cross_val_score` takes in our k-NN model and our data as parameters. Then it splits our data into 5 groups and fits and scores our data 5 separate times, recording the accuracy score in an array each time. We will save the accuracy scores in the `_cv_scores` variable.

To find the average of the 5 scores, we will use numpy's mean function, passing in `_cv_score`.

```
Numpy is a useful math library in Python.
from sklearn.model_selection import cross_val_score
import numpy as np#create a new KNN model
knn_cv = KNeighborsClassifier(n_neighbors=3)#train model with cv of 5
cv_scores = cross_val_score(knn_cv, X, y, cv=5)#print each cv score (accuracy) and average them
print(cv_scores)
print(_cv_scores mean:{}.format(np.mean(cv_scores)))

[ 0.68181818  0.69480519  0.75324675  0.75163399  0.68627451]
_cv_scores mean:0.7135557253204311
```

Bharati Vidyapeeth's College Of Engineering Lavale Pune.

Using cross-validation, our mean score is about 71.36%. This is a more accurate representation of how our model will perform on unseen data than our earlier testing using the holdout method.

## Hypertuning model parameters using GridSearchCV

When built our initial k-NN model, we set the parameter `_n_neighbors` to 3 as a starting point with no real logic behind that choice.

Hypertuning parameters is when you go through a process to find the optimal parameters for your model to improve accuracy. In our case, we will use GridSearchCV to find the optimal value for `_n_neighbors`.

GridSearchCV works by training our model multiple times on a range of parameters that we specify. That way, we can test our model with each parameter and figure out the optimal values to get the best accuracy results.

For our model, we will specify a range of values for `_n_neighbors` in order to see which value works best for our model. To do this, we will create a dictionary, setting `_n_neighbors` as the key and using numpy to create an array of values from 1 to 24.

Our new model using grid search will take in a new k-NN classifier, our `param_grid` and a

cross- validation value of 5 in order to find the optimal value for `_n_neighbors`.

```
from sklearn.model_selection import GridSearchCV#create new a knn model
knn2 = KNeighborsClassifier()#create a dictionary of all values we want to test for n_neighbors
param_grid = {_n_neighbors': np.arange(1, 25)}#use gridsearch to test all values for n_neighbors
knn_gscv = GridSearchCV(knn2, param_grid, cv=5)#fit model to data
knn_gscv.fit(X, y)
```

After training, we can check which of our values for `_n_neighbors` that we tested performed the best. To do this, we will call `_best_params_` on our model.

```
#check top performing n_neighbors value
knn_gscv.best_params_
```

```
{'_n_neighbors': 14}
```

We can see that 14 is the optimal value for `_n_neighbors`. We can use the `_best_score_` function to check the accuracy of our model when `_n_neighbors` is 14. `_best_score_` outputs the mean accuracy of the scores obtained through cross-validation.

```
#check mean score for the top performing value of n_neighbors
knn_gscv.best_score_
```

```
0.7578125
```

By using grid search to find the optimal parameter for our model, we have improved our model accuracy by over 4%!

Code :- <https://www.kaggle.com/code/shrutechlearn/step-by-step-diabetes-classification-knn-detailed>

**Conclusion:**

In this way we build a a neural network-based classifier that can determine whether they willleave or not in the next 6 months

Write-up	Correctness of Program	Documentation of Program	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
4	4	4	4	4	20	

Expected Date of Completion:..... Actual Date of Completion:.....

Group B

Assignment No : 6

**Title of the Assignment:** Implement K-Means clustering/ hierarchical clustering on sales\_data\_sample.csv dataset. Determine the number of clusters using the elbow method.

**Dataset Description:** The data includes the following features:

- 1. Customer ID
- 2. Customer Gender
- 3. Customer Age
- 4. Annual Income of the customer (in Thousand Dollars)
- 5. Spending score of the customer (based on customer behavior and spending nature)

**Objective of the Assignment:**

Students should able to understand how to use unsupervised learning to segment different-different clusters or groups and used to them to train your model to predict future things.

**Prerequisite:**

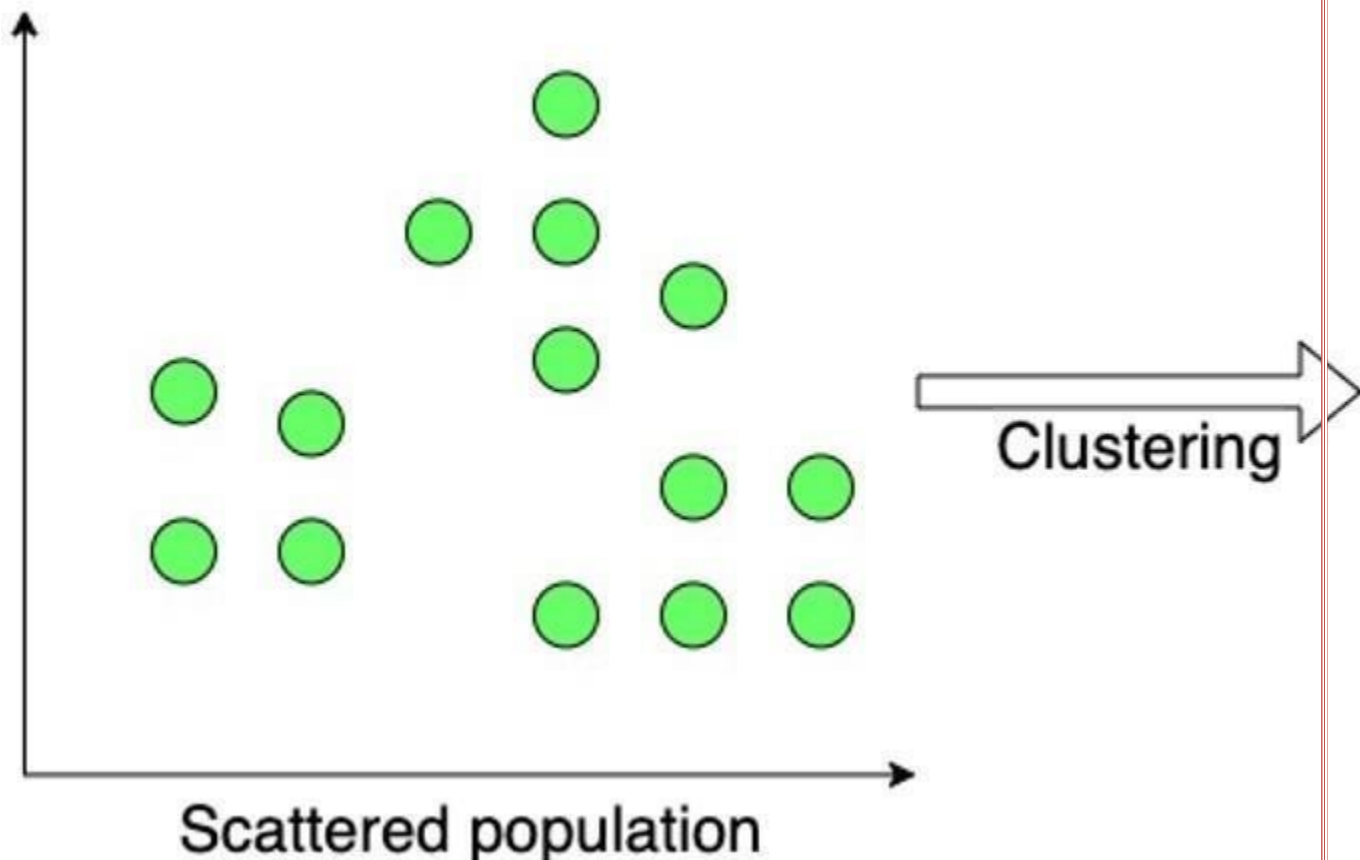
- 1. Knowledge of Python
- 2. Unsupervised learning
- 3. Clustering
- 4. Elbow method

Clustering algorithms try to find natural clusters in data, the various aspects of how the algorithms to cluster data can be tuned and modified. Clustering is based on the principle that items within the same cluster must be similar to each other.

The data is grouped in such a way that related elements are close to each other.

Bharati Vidyapeeth’s College Of Engineering Lavale Pune.

# Unsupervised Learning - C



Diverse and different types of data are subdivided into smaller groups.

## Uses of Clustering

Marketing:

In the field of marketing, clustering can be used to identify various customer groups with existing customer data. Based on that, customers can be provided with discounts, offers, promo codes etc.

Real Estate:

Clustering can be used to understand and divide various property locations based on value and importance. Clustering algorithms can process through the data and identify various groups of property on the basis of probable price.

BookStore and Library management:

Libraries and Bookstores can use Clustering to better manage the book database. With proper book ordering, better operations can be implemented.

Document Analysis:

Often, we need to group together various research texts and documents according to similarity. And in such cases, we don't have any labels. Manually labelling large amounts of data is also not possible. Using clustering, the algorithm can process the text and group it into different themes.

These are some of the interesting use cases of clustering.

### K-Means Clustering

K-Means clustering is an unsupervised machine learning algorithm that divides the given data into the given number of clusters. Here, the  $K$  is the given number of predefined clusters, that need to be created.

It is a centroid based algorithm in which each cluster is associated with a centroid. The main idea is to reduce the distance between the data points and their respective cluster centroid.

The algorithm takes raw unlabelled data as an input and divides the dataset into clusters and the process is repeated until the best clusters are found.

K-Means is very easy and simple to implement. It is highly scalable, can be applied to both small and large datasets. There is, however, a problem with choosing the number of clusters or  $K$ . Also, with the increase in dimensions, stability decreases. But, overall K Means is a simple and robust algorithm that makes clustering very easy.

```
#Importing the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

The necessary libraries are imported.

```
#Reading the excel file
data=pd.read_excel("Mall_Customers.xlsx")
```

The data is read. I will share a link to the entire code and excel data at the end of the article.

The data has 200 entries, that is data from 200 customers.

```
data.head()
```

So let us have a look at the data.

	CustomerID	Gender	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

data.corr()

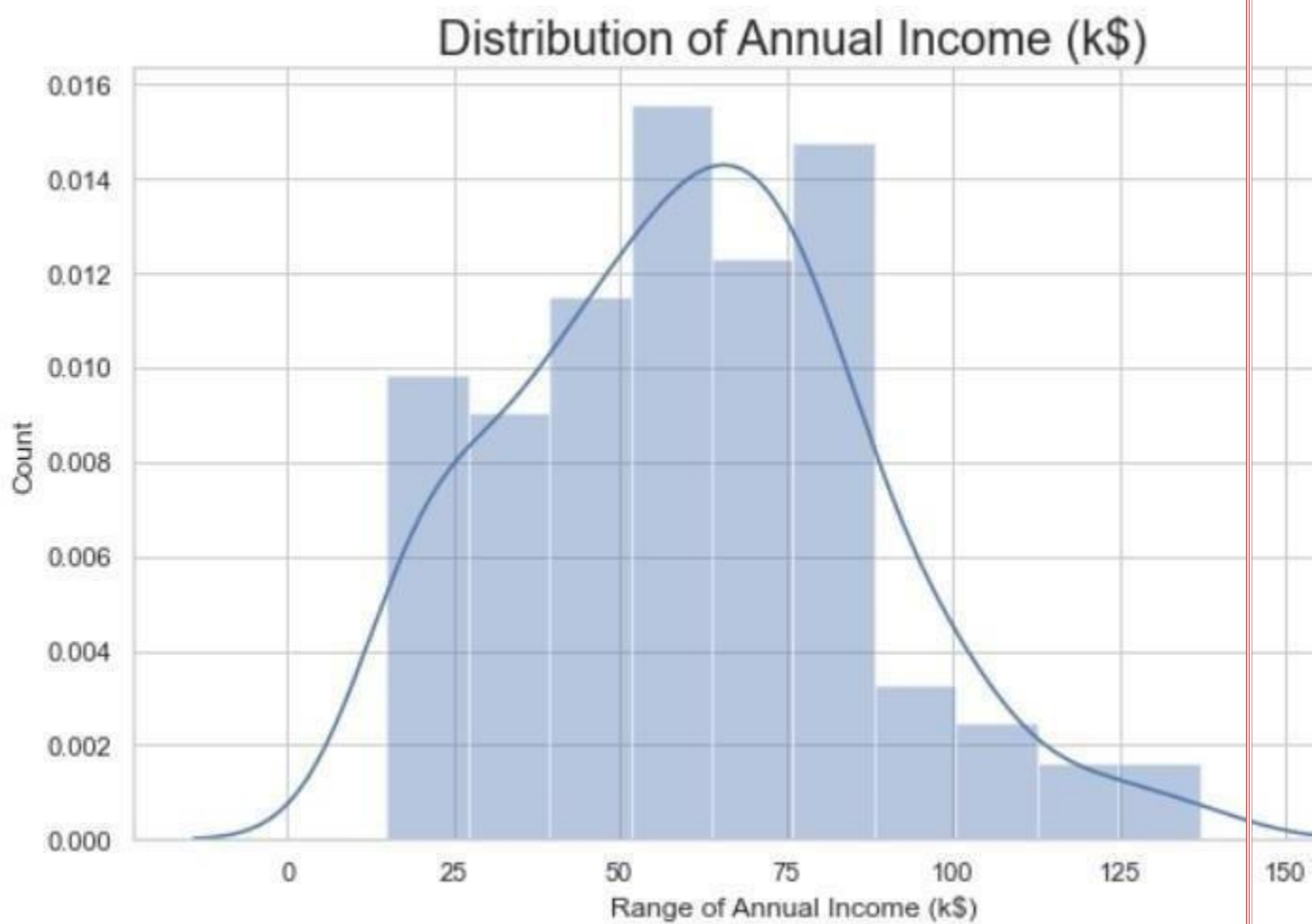
	CustomerID	Age	Annual Income (k\$)	Spending Score (1-100)
CustomerID	1.000000	-0.026763	0.977548	0.013835
Age	-0.026763	1.000000	-0.012398	-0.327227
Annual Income (k\$)	0.977548	-0.012398	1.000000	0.009903
Spending Score (1-100)	0.013835	-0.327227	0.009903	1.000000

The data seems to be interesting. Let us look at the data distribution.

Annual Income Distribution:

```
#Distribution of Annual Income
plt.figure(figsize=(10, 6))
sns.set(style = 'whitegrid')
sns.distplot(data['Annual Income (k$)'])
plt.title('Distribution of Annual Income (k$)', fontsize = 20)
plt.xlabel('Range of Annual Income (k$)')
plt.ylabel('Count')
```

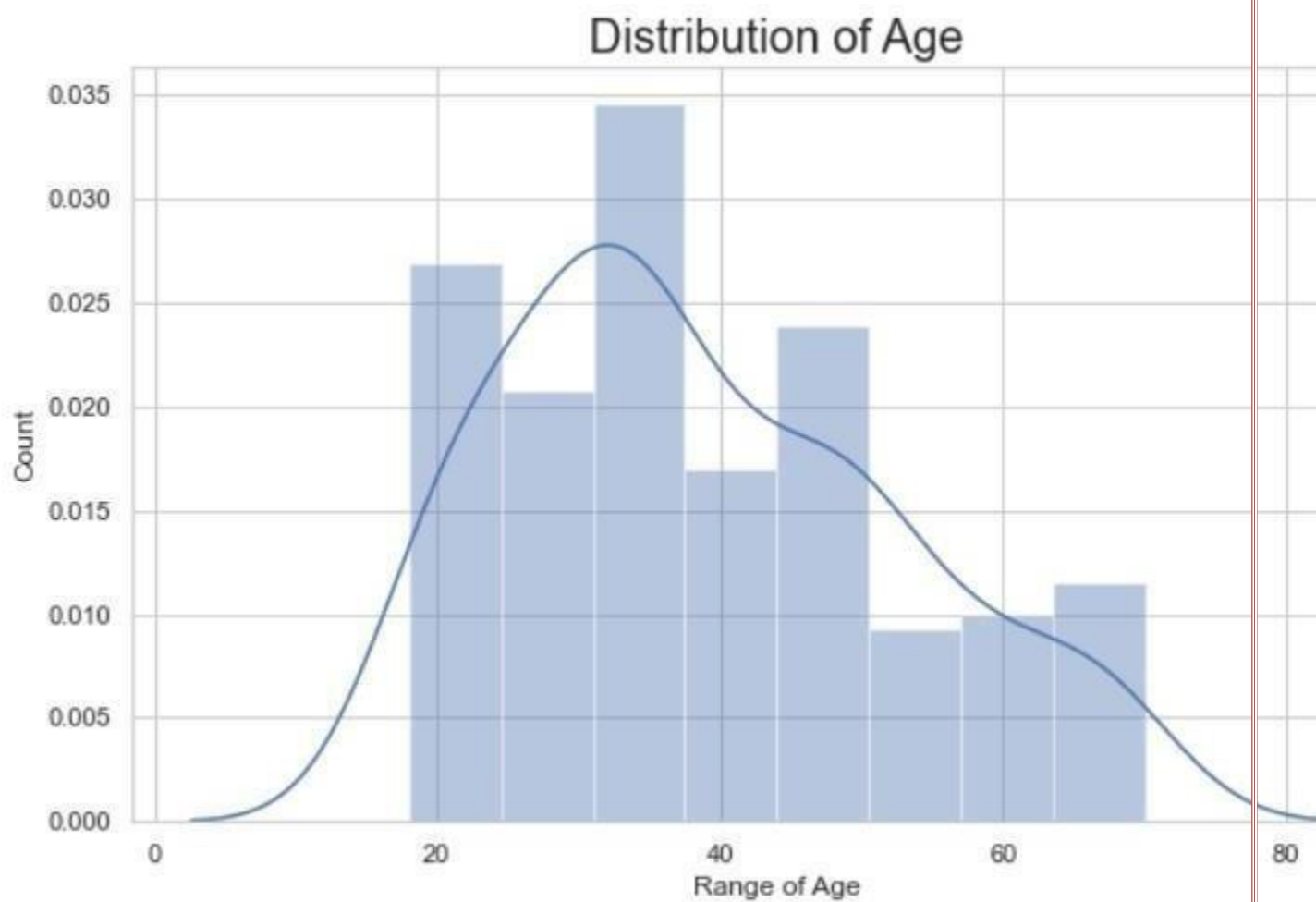




Most of the annual income falls between 50K to 85K.

**Age Distribution:**

```
#Distribution of age
plt.figure(figsize=(10, 6))
sns.set(style = 'whitegrid')
sns.distplot(data['Age'])
plt.title('Distribution of Age', fontsize = 20)
plt.xlabel('Range of Age')
plt.ylabel('Count')
```



There are customers of a wide variety of ages.

Spending Score Distribution:

Earn Rewards by Writing and Sharing Data Science Knowledge



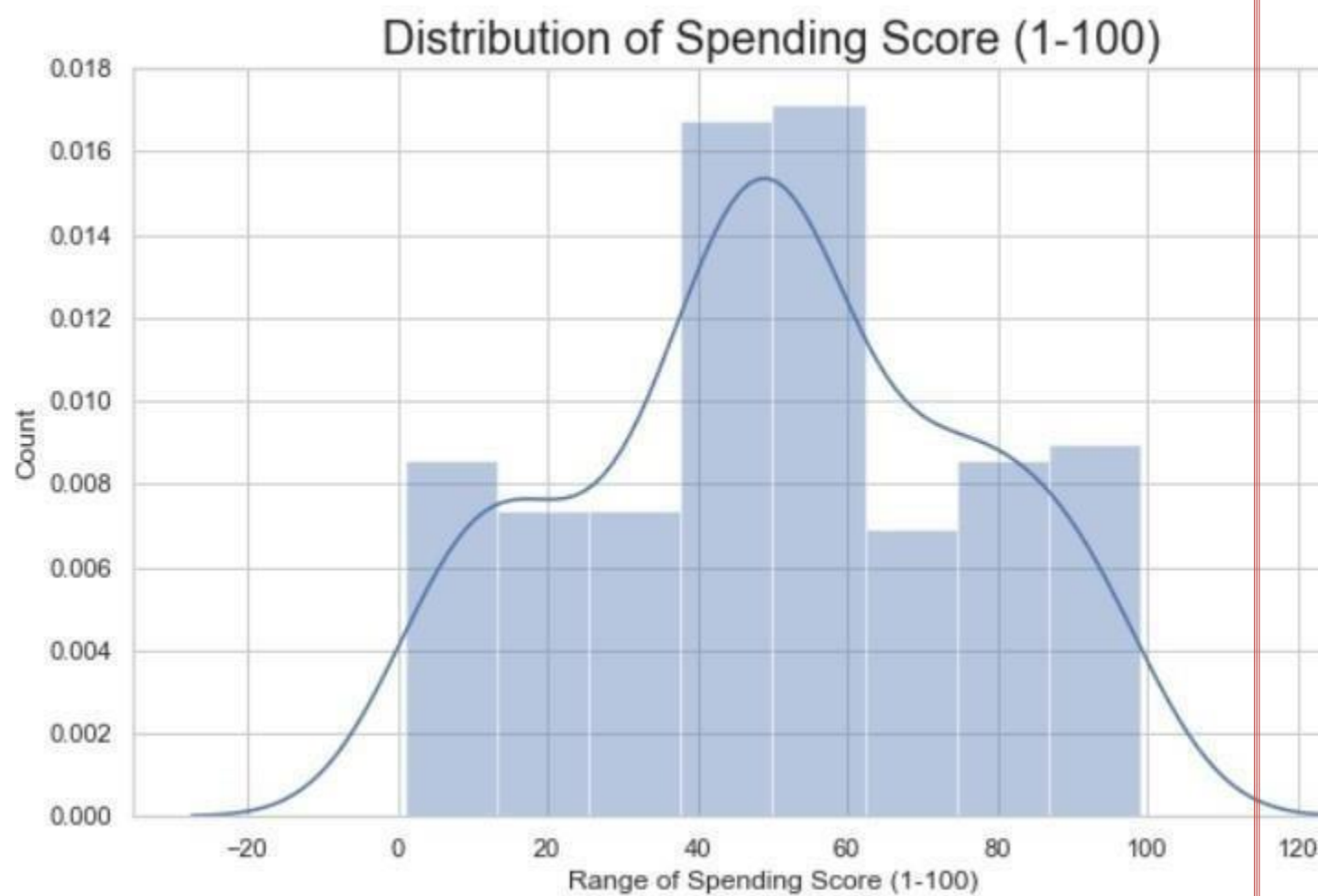
Learn | Write | Earn

Assured INR 2000 (\$26) for every published article![Register Now](#)

#Distribution of spending score

```
plt.figure(figsize=(10, 6))
sns.set(style = 'whitegrid')
sns.distplot(data['Spending Score (1-100)'])
plt.title('Distribution of Spending Score (1-100)', fontsize =
20) plt.xlabel('Range of Spending Score (1-100)')
plt.ylabel('Count')
```

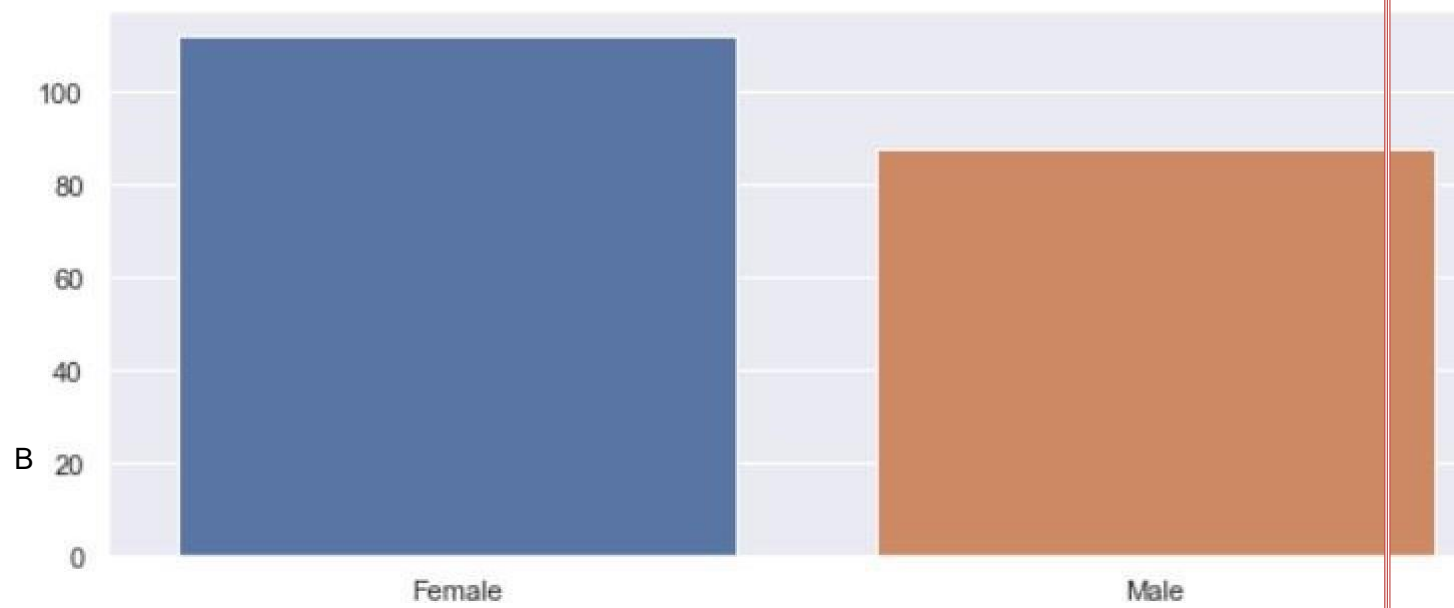
DYPATIL VIDYAPEETH'S College Of Engineering Lavale Pune.



The maximum spending score is in the range of 40 to 60.

**Gender Analysis:**

```
genders = data.Gender.value_counts()
sns.set_style("darkgrid")
plt.figure(figsize=(10,4))
sns.barplot(x=genders.index, y=genders.values)
plt.show()
```



More female customers than male.

I have made more visualizations. Do have a look at the GitHub link at the end to understand the data analysis and overall data exploration.

Clustering based on 2 features

First, we work with two features only, annual income and spending score.

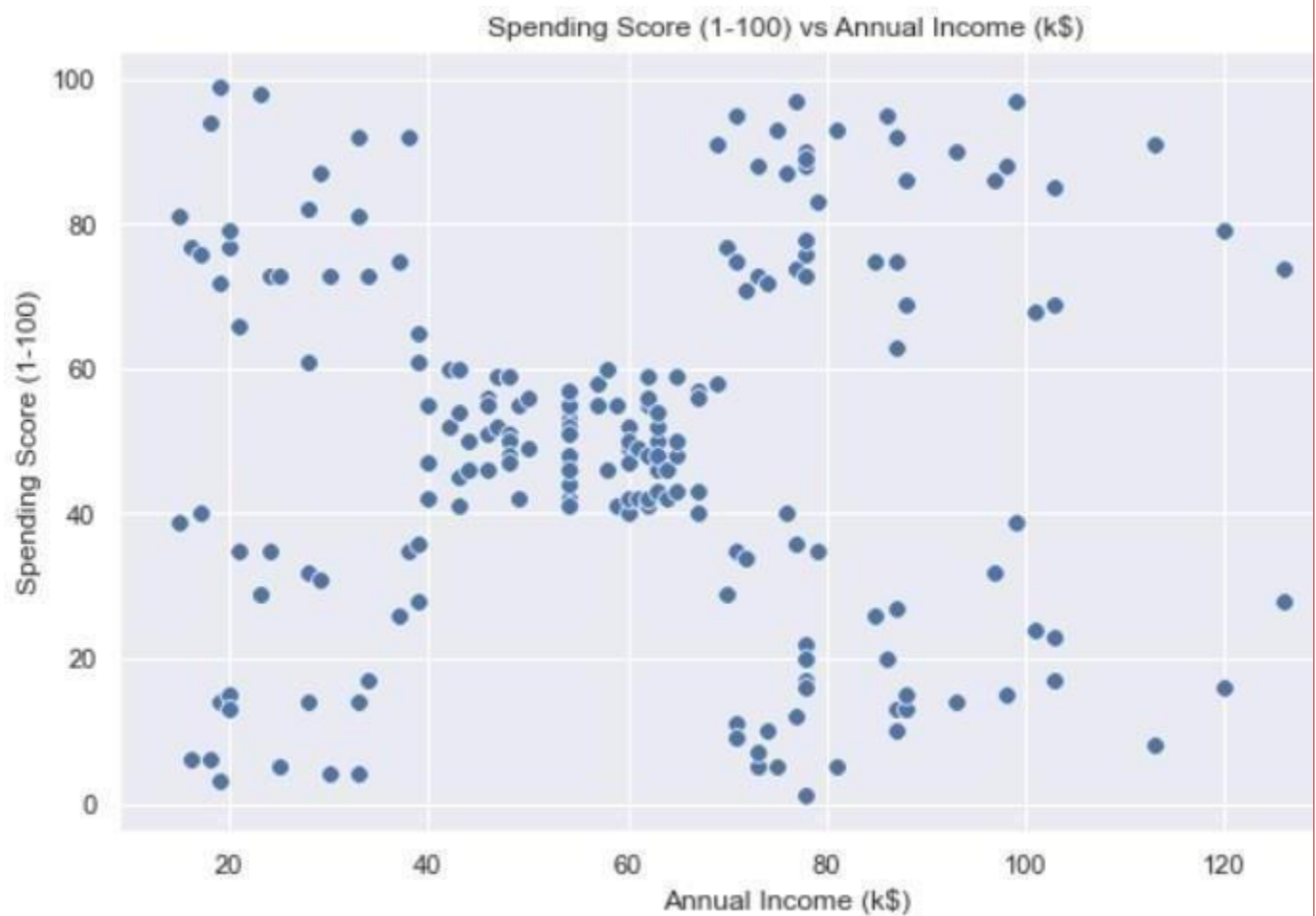
```
#We take just the Annual Income and Spending score
df1=data[["CustomerID","Gender","Age","Annual Income (k$)","Spending Score (1-100)"]]
X=df1[["Annual Income (k$)","Spending Score (1-100)"]]

#The input data
X.head()
```

	Annual Income (k\$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40

```
#Scatterplot of the input data
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'Annual Income (k$)',y = 'Spending Score (1-100)', data = X ,s = 60)
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Spending Score (1-100) vs Annual Income (k$)')
plt.show()
```

The data does seem to hold some patterns.

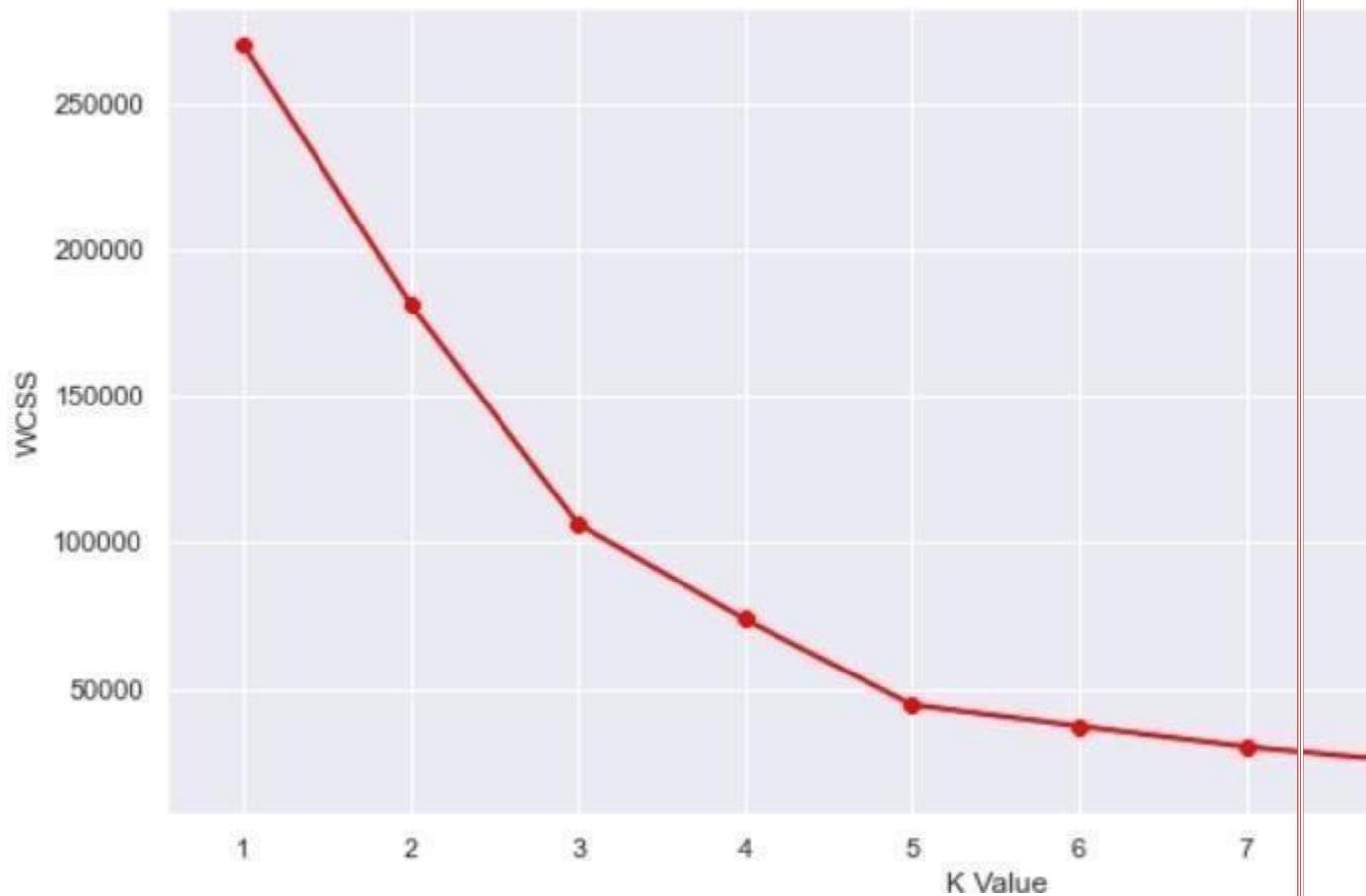


```
#Importing KMeans from sklearn
from sklearn.cluster import
KMeans
```

Now we calculate the Within Cluster Sum of Squared Errors (WSS) for different values of k. Next, we choose the k for which WSS first starts to diminish. This value of K gives us the best number of clusters to make from the raw data.

```
wcss=[]
for i in range(1,11):
    km=KMeans(n_clusters=i)
    km.fit(X)
    wcss.append(km.inertia_)
#The elbow curve
plt.figure(figsize=(12,6))
plt.plot(range(1,11),wcss)
plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1
)) plt.ylabel("WCSS")
plt.show()
```

The plot:



This is known as the elbow graph, the x-axis being the number of clusters, the number of clusters is taken at the elbow joint point. This point is the point where making clusters is most relevant as here the value of WCSS suddenly stops decreasing. Here in the graph, after 5 the drop is minimal, so we take 5 to be the number of clusters.

```
#Taking 5 clusters
km1=KMeans(n_clusters=5)
#Fitting the input data
km1.fit(X)
#predicting the labels of the input data
y=km1.predict(X)
#adding the labels to a column named label
df1["label"] = y
#The new dataframe with the clustering done
df1.head()
```

The labels added to the data.

Bharati Vidyapeeth's College Of Engineering Lavale Pune.

	CustomerID	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	label
0	1	Male	19	15	39	4
1	2	Male	21	15	81	2
2	3	Female	20	16	6	4
3	4	Female	23	16	77	2
4	5	Female	31	17	40	4

#Scatterplot of the clusters

```
plt.figure(figsize=(10,6))
sns.scatterplot(x = 'Annual Income (k$)',y = 'Spending Score (1-100)',hue="label",
                palette=['green','orange','brown','dodgerblue','red'],
                legend='full',data = df1 ,s = 60 )
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.title('Spending Score (1-100) vs Annual Income (k$)')
plt.show()
```



We can clearly see that 5 different clusters have been formed from the data. The red cluster is the customers with the least income and least spending score, similarly, the blue cluster is the customers with the most income and most spending score.

**k-Means Clustering on the basis of 3D data**

Now, we shall be working on 3 types of data. Apart from the spending score and



annual income of customers, we shall also take in the age of the customers.

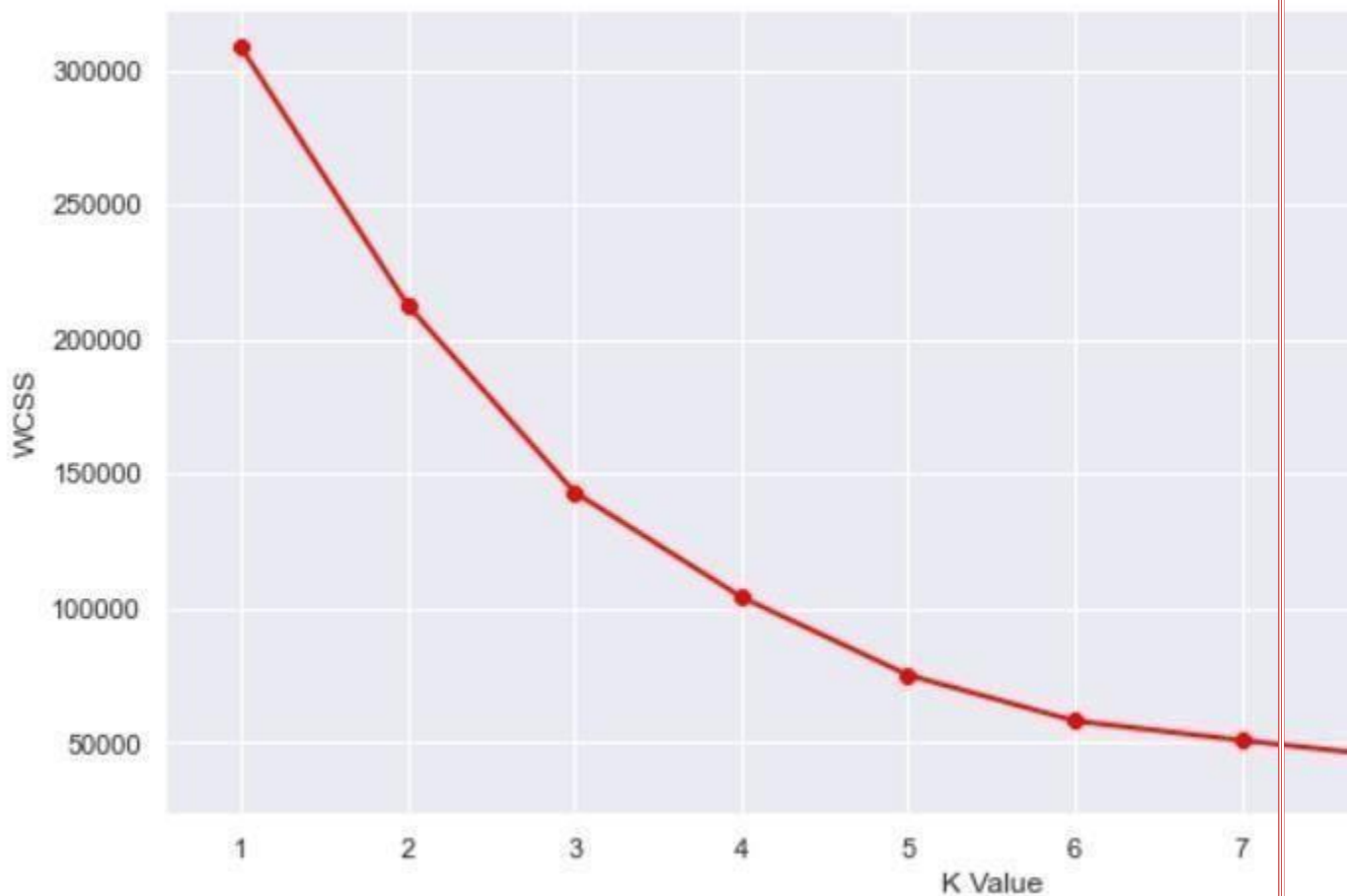
```
#Taking the features
X2=df2[["Age","Annual Income (k$)","Spending Score (1-100)"]]

#Now we calculate the Within Cluster Sum of Squared Errors (WSS) for different values
of k.

wcss = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(X2)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(12,6))
plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")
plt.xlabel("K Value")
plt.xticks(np.arange(1,11,1
)) plt.ylabel("WCSS")
plt.show()
```

The WCSS curve.



Here can assume that K=5 will be a good value.

```
#We choose the k for which WSS starts to
diminish km2 = KMeans(n_clusters=5)
y2 = km2.fit_predict(X2)
df2["label"] = y2

#The data with
labels df2.head()
```



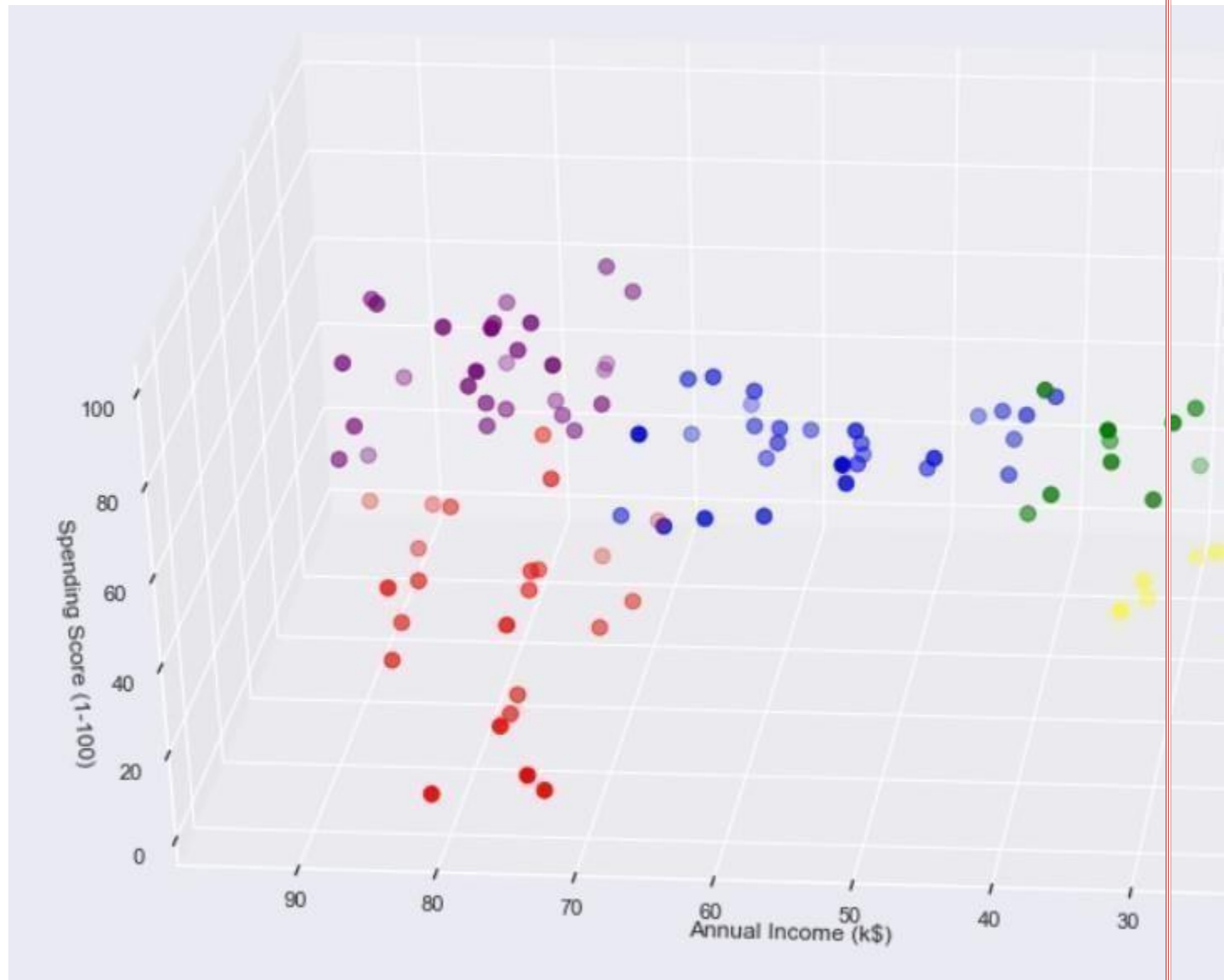
The data:

	CustomerID	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	label
0	1	Male	19	15	39	5
1	2	Male	21	15	81	3
2	3	Female	20	16	6	4
3	4	Female	23	16	77	3
4	5	Female	31	17	40	5

Now we plot it.

```
#3D Plot as we did the clustering on the basis of 3 input
features fig = plt.figure(figsize=(20,10))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df2.Age[df2.label == 0], df2["Annual Income (k$)"][df2.label == 0],
df2["Spending Score (1-100)"][df2.label == 0], c='purple', s=60)
ax.scatter(df2.Age[df2.label == 1], df2["Annual Income (k$)"][df2.label == 1],
df2["Spending Score (1-100)"][df2.label == 1], c='red', s=60)
ax.scatter(df2.Age[df2.label == 2], df2["Annual Income (k$)"][df2.label == 2],
df2["Spending Score (1-100)"][df2.label == 2], c='blue', s=60)
ax.scatter(df2.Age[df2.label == 3], df2["Annual Income (k$)"][df2.label == 3],
df2["Spending Score (1-100)"][df2.label == 3], c='green', s=60)
ax.scatter(df2.Age[df2.label == 4], df2["Annual Income (k$)"][df2.label == 4],
df2["Spending Score (1-100)"][df2.label == 4], c='yellow', s=60)
ax.view_init(35, 185)
plt.xlabel("Age")
plt.ylabel("Annual Income (k$)")
ax.set_zlabel('Spending Score (1-100)') plt.show()
```

The output:



What we get is a 3D plot. Now, if we want to know the customer IDs, we can do that too.

```

cust1=df2[df2["label"]==1]
print('Number of customer in 1st group=',
len(cust1)) print('They are -',
cust1["CustomerID"].values) print("-----")
cust2=df2[df2["label"]==2]
print('Number of customer in 2nd group=',
len(cust2)) print('They are -',
cust2["CustomerID"].values) print("-----")
cust3=df2[df2["label"]==0]
print('Number of customer in 3rd group=',
len(cust3)) print('They are -',
cust3["CustomerID"].values) print("
-----")

")
cust4=df2[df2["label"]==3]
print('Number of customer in 4th group=',
len(cust4)) print('They are -',
cust4["CustomerID"].values) print("-----")
cust5=df2[df2["label"]==4]
print('Number of customer in 5th group=', len(cust5))

```



```
print("-----")
```

The output we get:

Number of customer in 1st group= 24

They are - [129 131 135 137 139 141 145 147 149 151 153 155 157 159 161 163 165 167 169 171 173 175 177 179]

---

Number of the customer in 2nd group= 29

They are - [ 47 51 55 56 57 60 67 72 77 78 80 82 84 86 90 93 94 97 99 102 105 108 113 118 119 120 122 123 127]

---

Number of the customer in 3rd group= 28

They are - [124 126 128 130 132 134 136 138 140 142 144 146 148 150 152 154 156 158

160 162 164 166 168 170 172 174 176 178]

---

Number of the customer in 4th group= 22

They are - [ 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 46]

-----

Number of customer in 5th group= 12

They are - [ 3 7 9 11 13 15 23 25 31 33 35 37]

---

So, we used K-Means clustering to understand customer data. K-Means is a good clustering algorithm. Almost all the clusters have similar density. It is also fast and efficient in terms of computational cost.

**Problem Statement:** - Build a machine learning model that predicts the type of people who survived the Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class, etc.).

Objective: Students should learn to build a machine learning model.

Theory:

Here's a step-by-step guide on how to approach this problem using Python and some popular libraries:

**1. Data Collection and Understanding:**

- Start by obtaining the Titanic dataset, which contains passenger information and survival labels. You can find datasets on websites like Kaggle.

**2. Data Pre-processing:**

- Clean the data by handling missing values, outliers, and redundant features.
- Perform feature engineering to create relevant features or transform existing ones.
- Encode categorical variables into numerical format using techniques like one-hot encoding.

**3. Data Splitting:**

- Split your dataset into a training set and a test set. This allows you to evaluate your model's performance on unseen data.

**4. Select a Machine Learning Algorithm:**

- Choose a classification algorithm suitable for this problem. Common choices include Decision Trees, Random Forests, Logistic Regression, Support Vector Machines, or Gradient Boosting.

**5. Model Training:**

- Fit your chosen algorithm to the training data. The model learns patterns from the data.

**6. Model Evaluation:**

- Evaluate your model's performance using metrics like accuracy, precision, recall, F1-score, and the ROC-AUC score. Cross-validation can help in assessing how well the model generalizes to new data.

**7. Hyperparameter Tuning:**

- Experiment with different hyperparameters to optimize your model's performance. Techniques like grid search or random search can be helpful.

**8. Model Interpretation:**

- Understand the feature importance or coefficients of your model to interpret how different features affect survival.

**9. Prediction:**

- Use your trained model to make predictions on new, unseen data or the test set.

**10. Post-processing:**

- You may need to further process the model's output, such as setting a threshold for classification.

**Importing the Libraries**

```
# linear algebra
import numpy as np

# data processing
import pandas as pd

# data visualization
import seaborn as sns
%matplotlib inline
from matplotlib import pyplot as plt
from matplotlib import style

# Algorithms
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.naive_bayes import GaussianNB
```

**Getting the Data**

```
test_df = pd.read_csv("test.csv")
train_df = pd.read_csv("train.csv")
```

**Data Exploration/Analysis**

```
train_df.info()
```

Bharati Vidyapeeth's College Of Engineering Lavale Pune.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId      891 non-null int64
Survived         891 non-null int64
Pclass           891 non-null int64
Name             891 non-null object
Sex              891 non-null object
Age              714 non-null float64
SibSp            891 non-null int64
Parch           891 non-null int64
Ticket           891 non-null object
Fare             891 non-null float64
Cabin            204 non-null object
Embarked         889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

**The training-set has 891 examples and 11 features + the target variable (survived).** 2 of the features are floats, 5 are integers and 5 are objects. Below I have listed the features with a short description:

```
survival:      Survival
PassengerId:  Unique Id of a passenger.
pclass:       Ticket class
sex:         Sex
Age:         Age in years
sibsp:       # of siblings / spouses aboard the Titanic
parch:       # of parents / children aboard the Titanic
ticket:      Ticket number
fare:        Passenger fare
cabin:       Cabin number
embarked:    Port of Embarkation
train_df.describe()
```



	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Above we can see that **38% out of the training-set survived the Titanic**. We can also see that the passenger ages range from 0.4 to 80. Ontop of that we can already detect some features, that contain missing

values, like the „Age“ feature.  
train\_df.head(8)

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S

From the table above, we can note a few things. First of all, that we **need to convert a lot of features into numeric** ones later on, so that the machine learning algorithms can process them. Furthermore, we can see that the **features have widely different ranges**, that we will need to





Let’s take a more detailed look at what data is actually missing:

```
total = train_df.isnull().sum().sort_values(ascending=False)
percent_1 = train_df.isnull().sum()/train_df.isnull().count()*100
percent_2 = (round(percent_1, 1)).sort_values(ascending=False)
missing_data = pd.concat([total, percent_2], axis=1, keys=['Total',
'%'])
missing_data.head(5)
```

	Total	%
Cabin	687	77.1
Age	177	19.9
Embarked	2	0.2
Fare	0	0.0
Ticket	0	0.0

The Embarked feature has only 2 missing values, which can easily be filled. It will be much more tricky, to deal with the „Age“ feature, which has 177 missing values. The „Cabin“ feature needs further investigation, but it looks like that we might want to drop it from the dataset, since 77 % of it are missing.

```
train_df.columns.values
array(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'], dtype=object)
```

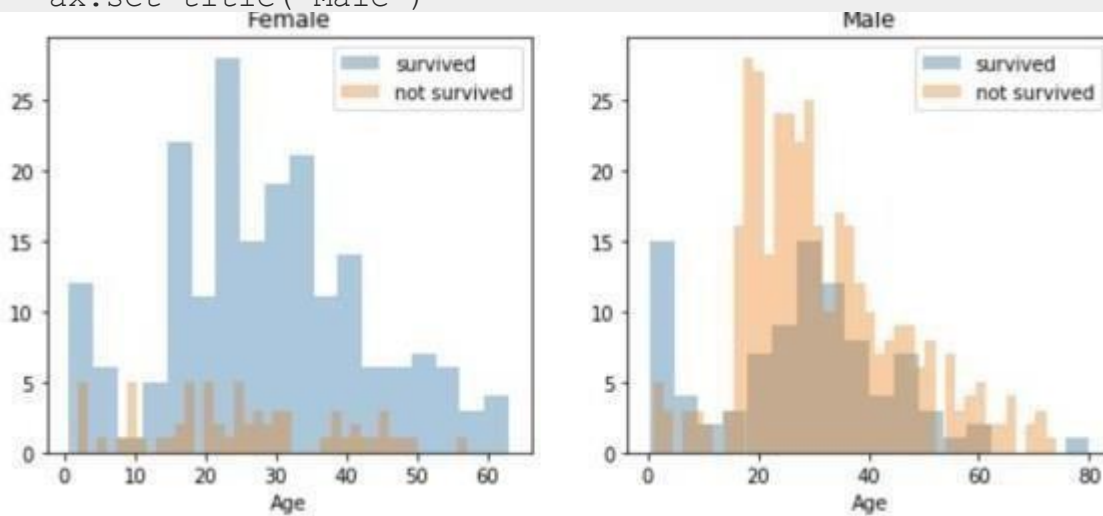
Above you can see the 11 features + the target variable (survived). **What features could contribute to a high survival rate ?**

To me it would make sense if everything except „PassengerId“, „Ticket“ and „Name“ would be correlated with a high survival rate.

1. Age and Sex:

```
survived = 'survived'
not_survived = 'not survived'
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
women = train_df[train_df['Sex']=='female']
men = train_df[train_df['Sex']=='male']
ax = sns.distplot(women[women['Survived']==1].Age.dropna(), bins=18,
label = survived, ax = axes[0], kde =False)
ax = sns.distplot(women[women['Survived']==0].Age.dropna(), bins=40,
label = not_survived, ax = axes[0], kde =False)
ax.legend()
ax.set_title('Female')
ax = sns.distplot(men[men['Survived']==1].Age.dropna(), bins=18, label
= survived, ax = axes[1], kde = False)
```

```
ax = sns.distplot(men[men['Survived']==0].Age.dropna(), bins=40, label=
not_survived, ax = axes[1], kde = False)
ax.legend()
= ax.set title('Male')
```



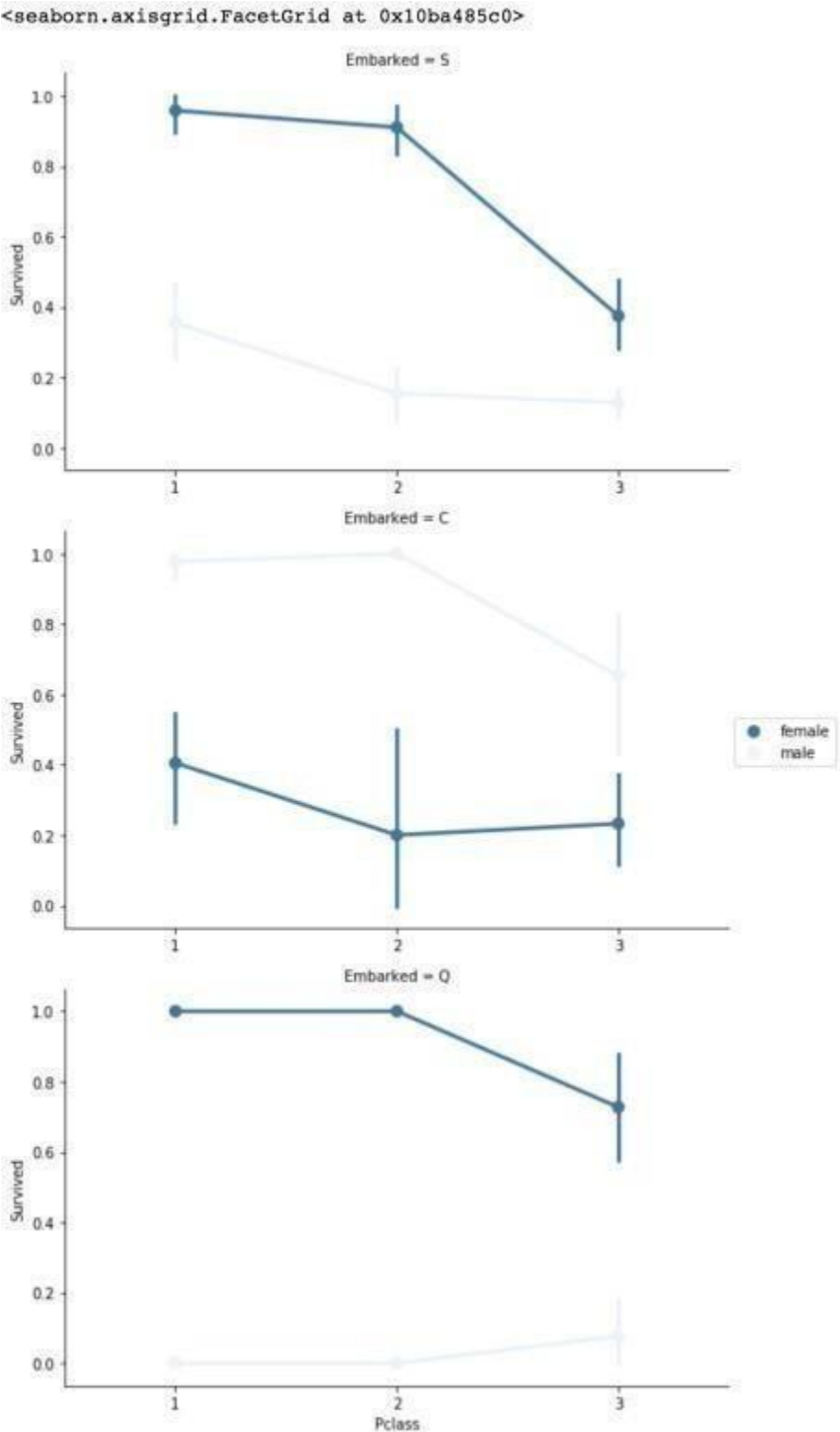
You can see that men have a high probability of survival when they are between 18 and 30 years old, which is also a little bit true for women but not fully. For women the survival chances are higher between 14 and 40.

For men the probability of survival is very low between the age of 5 and 18, but that isn't true for women. Another thing to note is that infants also have a little bit higher probability of survival.

Since there seem to be **certain ages, which have increased odds of survival** and because I want every feature to be roughly on the same scale, I will create age groups later on.

### 3. Embarked, Pclass and Sex:

```
FacetGrid = sns.FacetGrid(train_df, row='Embarked', size=4.5,
aspect=1.6)
FacetGrid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex',
palette=None, order=None, hue_order=None)
FacetGrid.add_legend()
```



Embarked seems to be correlated with survival, depending on the gender.

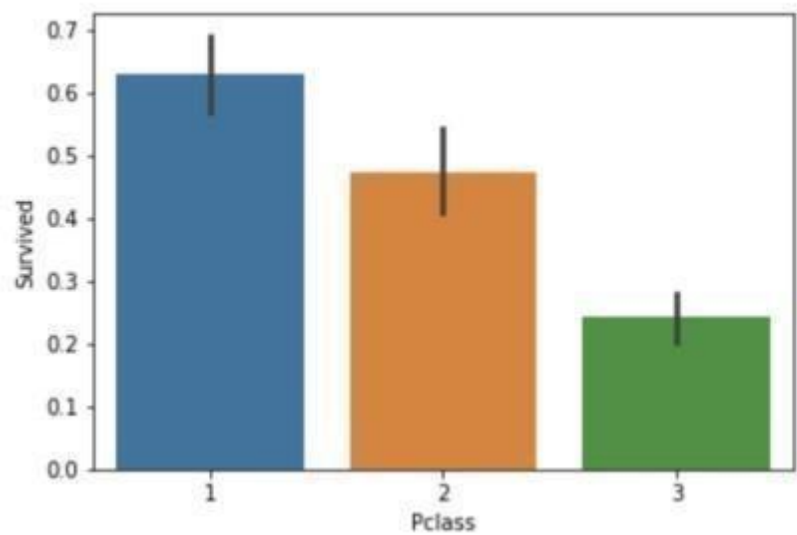
Women on port Q and on port S have a higher chance of survival. The inverse is true, if they are at port C. Men have a high survival probability if they are on port C, but a low probability if they are on port Q or S.

Pclass also seems to be correlated with survival. We will generate another plot of it below.

**4. Pclass:**

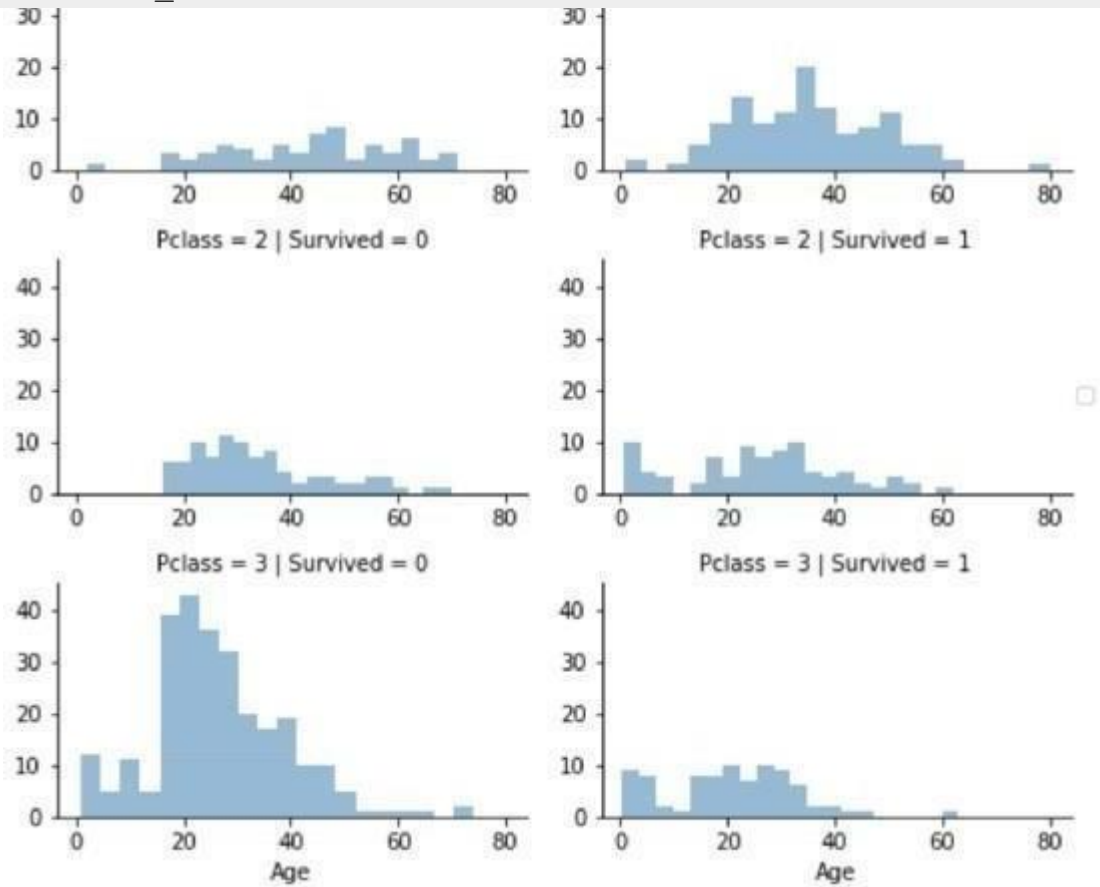
```
sns.barplot(x='Pclass', y='Survived', data=train_df)
```

<matplotlib.axes.\_subplots.AxesSubplot at 0x10d1dc7b8>



Here we see clearly, that Pclass is contributing to a persons chance of survival, especially if this person is in class 1. We will create another pclassplot below.

```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', size=2.2,
                    aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```



The plot above confirms our assumption about pclass 1, but we can also spot a high probability that a person in pclass 3 will not survive.

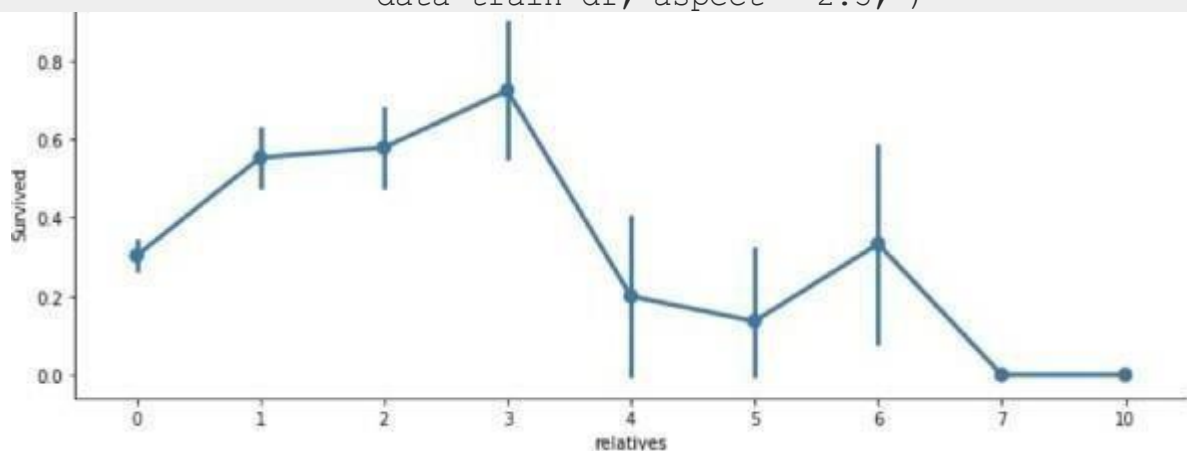
## 5. SibSp and Parch:

SibSp and Parch would make more sense as a combined feature, that shows the total number of relatives, a person has on the Titanic. I will create it below and also a feature that shows if someone is not alone.

```
data = [train_df, test_df]
for dataset in data:
    dataset['relatives'] = dataset['SibSp'] + dataset['Parch']
    dataset.loc[dataset['relatives'] > 0, 'not_alone'] = 0
    dataset.loc[dataset['relatives'] == 0, 'not_alone'] = 1
    dataset['not_alone'] =
dataset['not_alone'].astype(int)
train_df['not_alone'].value_counts()
```

```
1    537
0    354
Name: not_alone, dtype: int64
```

```
axes = sns.factorplot('relatives', 'Survived',
                      data=train_df, aspect = 2.5, )
```



Here we can see that you had a high probability of survival with 1 to 3 relatives, but a lower one if you had less than 1 or more than 3 (except for some cases with 6 relatives).

## Data Preprocessing

First, I will drop „PassengerId“ from the train set, because it does not contribute to a person's survival probability. I will not drop it from the test set, since it is required there for the submission.

```
train_df = train_df.drop(['PassengerId'], axis=1)
```

Missing Data:

## Cabin:

As a reminder, we have to deal with Cabin (687), Embarked (2) and Age (177). First I thought, we have to delete the „Cabin“ variable but then I

found something interesting. A cabin number looks like „C123“ and the **letter refers to the deck**. Therefore we’re going to extract these and create a new feature, that contains a person's deck. Afterwards we will convert the feature into a numeric variable. The missing values will be converted to zero. In the picture below you can see the actual decks of the Titanic, ranging from A to G.

```
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [train_df, test_df]

for dataset in data:
    dataset['Cabin'] = dataset['Cabin'].fillna("U0")
    dataset['Deck'] = dataset['Cabin'].map(lambda x: re.compile("([a-zA-Z]+)").search(x).group())
    dataset['Deck'] = dataset['Deck'].map(deck)
    dataset['Deck'] = dataset['Deck'].fillna(0)
    dataset['Deck'] = dataset['Deck'].astype(int) # we can now drop the cabin feature
train_df = train_df.drop(['Cabin'], axis=1)
test_df = test_df.drop(['Cabin'], axis=1)
```

### Age:

Now we can tackle the issue with the age features missing values. I will create an array that contains random numbers, which are computed based on the mean age value in regards to the standard deviation and is\_null.

```
data = [train_df, test_df]

for dataset in data:
    mean = train_df["Age"].mean()
    std = test_df["Age"].std()
    is_null = dataset["Age"].isnull().sum()
    # compute random numbers between the mean, std and is_null
    rand_age = np.random.randint(mean - std, mean + std, size = is_null)
    # fill NaN values in Age column with random values generated
    age_slice = dataset["Age"].copy()
    age_slice[np.isnan(age_slice)] = rand_age
    dataset["Age"] = age_slice
    dataset["Age"] = dataset["Age"].astype(int)
train_df["Age"].isnull().sum()
```

0

Bharati Vidyapeeth's College Of Engineering Lavale Pune.

3

### Embarked:

Since the Embarked feature has only 2 missing values, we will just fill these with the most common one.

```
train_df['Embarked'].describe()
```



```

count      889
unique      3
top         S
freq       644
Name: Embarked, dtype: object
common_value = 'S'
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].fillna(common_value)

```

Converting Features:

train\_df.info()

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
Survived      891 non-null int64
Pclass        891 non-null int64
Name          891 non-null object
Sex           891 non-null object
Age           891 non-null int64
SibSp         891 non-null int64
Parch         891 non-null int64
Ticket        891 non-null object
Fare          891 non-null float64
Embarked      891 non-null object
relatives     891 non-null int64
not_alone     891 non-null int64
Deck          891 non-null int64
dtypes: float64(1), int64(8), object(4)
memory usage: 90.6+ KB

```

Above you can see that „Fare“ is a float and we have to deal with 4 categorical features: Name, Sex, Ticket and Embarked. Lets investigate and transform one after another.

### Fare:

Converting —Fare— from float to int64, using the —astype()— function pandas provides:

```

data = [train_df, test_df]

for dataset in data:
    dataset['Fare'] = dataset['Fare'].fillna(0)
    dataset['Fare'] = dataset['Fare'].astype(int)

```

### Name:

We will use the Name feature to extract the Titles from the Name, so that we can build a new feature out of that.

```

data = [train_df, test_df]
titles = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}

for dataset in data:
    # extract titles

```

```
dataset['Title'] = dataset.Name.str.extract(' ([A-Za-z]+)\.',
expand=False)
# replace titles with a more common title or as Rare
dataset['Title'] = dataset['Title'].replace(['Lady',
'Countess','Capt', 'Col', 'Don', 'Dr',\
'Major', 'Rev', 'Sir',
'Jonkheer', 'Dona'], 'Rare')
dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
# convert titles into numbers
dataset['Title'] = dataset['Title'].map(titles)
# filling NaN with 0, to get safe
dataset['Title'] = dataset['Title'].fillna(0)
train_df =
train_df.drop(['Name'], axis=1)
test_df = test_df.drop(['Name'], axis=1)
```

### Sex:

Convert „Sex“ feature into numeric.

```
genders = {"male": 0, "female": 1}
data = [train_df, test_df]

for dataset in data:
    dataset['Sex'] = dataset['Sex'].map(genders)
```

### Ticket:

```
train_df['Ticket'].describe()
count      891
unique      681
top         1601
freq         7
Name: Ticket, dtype: object
```

Since the Ticket attribute has 681 unique tickets, it will be a bit tricky to convert them into useful categories. So we will drop it from the dataset.

```
train_df = train_df.drop(['Ticket'], axis=1)
test_df = test_df.drop(['Ticket'], axis=1)
```

### Embarked:

Convert „Embarked“ feature into numeric.

```
ports = {"S": 0, "C": 1, "Q": 2}
data = [train_df, test_df]

for dataset in data:
    dataset['Embarked'] = dataset['Embarked'].map(ports)
```

### Creating Categories:

We will now create categories within the following features:

**Age:**

Now we need to convert the „age“ feature. First we will convert it from float into integer. Then we will create the new „AgeGroup“ variable, by categorizing every age into a group. Note that it is important to place attention on how you form these groups, since you don't want for example that 80% of your data falls into group 1.

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age'] = dataset['Age'].astype(int)
    dataset.loc[ dataset['Age'] <= 11, 'Age'] = 0
    dataset.loc[(dataset['Age'] > 11) & (dataset['Age'] <= 18), 'Age']
= 1
    dataset.loc[(dataset['Age'] > 18) & (dataset['Age'] <= 22), 'Age']
= 2
    dataset.loc[(dataset['Age'] > 22) & (dataset['Age'] <= 27), 'Age']
= 3
    dataset.loc[(dataset['Age'] > 27) & (dataset['Age'] <= 33), 'Age']
= 4
    dataset.loc[(dataset['Age'] > 33) & (dataset['Age'] <= 40), 'Age']
= 5
    dataset.loc[(dataset['Age'] > 40) & (dataset['Age'] <= 66), 'Age']
= 6
    dataset.loc[ dataset['Age'] > 66, 'Age'] = 6

# let's see how it's distributed
train_df['Age'].value_counts()
4      165
6      158
5      147
3      129
2      124
1      100
0       68
Name: Age, dtype: int64
```

**Fare:**

For the „Fare“ feature, we need to do the same as with the „Age“ feature. But it isn't that easy, because if we cut the range of the fare values into a few equally big categories, 80% of the values would fall into the first category. Fortunately, we can use sklearn —qcut() function, that we can use to see, how we can form the categories.

```
train_df.head(10)
```

Bharati Vidyapeeth's College Of Engineering Lavale Pune.

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Title
0	0	3	0	2	1	0	7	0	1	0	8	1
1	1	1	1	5	1	0	71	1	1	0	3	3
2	1	3	1	3	0	0	7	0	0	1	8	2
3	1	1	1	5	1	0	53	0	1	0	3	3
4	0	3	0	5	0	0	8	0	0	1	8	1
5	0	3	0	4	0	0	8	2	0	1	8	1
6	0	1	0	6	0	0	51	0	0	1	5	1
7	0	3	0	0	3	1	21	0	4	0	8	4
8	1	3	1	3	0	2	11	0	2	0	8	3
9	1	2	1	1	1	0	30	1	1	0	8	3

```
data = [train_df, test_df]

for dataset in data:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <=
14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31),
'Fare'] = 2
    dataset.loc[(dataset['Fare'] > 31) & (dataset['Fare'] <= 99),
'Fare'] = 3
    dataset.loc[(dataset['Fare'] > 99) & (dataset['Fare'] <= 250),
'Fare'] = 4
    dataset.loc[ dataset['Fare'] > 250, 'Fare'] = 5
    dataset['Fare'] = dataset['Fare'].astype(int)
```

Creating new Features

I will add two new features to the dataset, that I compute out of otherfeatures.

1. Age times Class

```
data = [train_df, test_df]
for dataset in data:
    dataset['Age Class']= dataset['Age']* dataset['Pclass']
```

2. Fare per Person

```
for dataset in data:
    dataset['Fare_Per_Person'] =
dataset['Fare']/(dataset['relatives']+1)
    dataset['Fare_Per_Person'] =
dataset['Fare_Per_Person'].astype(int)# Let's take a last look at the
training set, before we start training the models.
train df.head(10)
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	relatives	not_alone	Deck	Title	Age_Class	Fare_Per_Pers
0	0	3	0	2	1	0	0	0	1	0	8	1	6	0
1	1	1	1	5	1	0	3	1	1	0	3	3	5	1
2	1	3	1	3	0	0	0	0	0	1	8	2	9	0
3	1	1	1	5	1	0	3	0	1	0	3	3	5	1
4	0	3	0	5	0	0	1	0	0	1	8	1	15	1
5	0	3	0	4	0	0	1	2	0	1	8	1	12	1
6	0	1	0	6	0	0	3	0	0	1	5	1	6	3
7	0	3	0	0	3	1	2	0	4	0	8	4	0	0
8	1	3	1	3	0	2	1	0	2	0	8	3	9	0
9	1	2	1	1	1	0	2	1	1	0	8	3	2	1
10	1	3	1	0	1	1	2	0	2	0	7	2	0	0

Building Machine Learning Models

Now we will train several Machine Learning models and compare their results. Note that because the dataset does not provide labels for their testing-set, we need to use the predictions on the training set to compare the algorithms with each other. Later on, we will use cross validation.

```
X_train = train_df.drop("Survived", axis=1)
Y_train = train_df["Survived"]
X_test = test_df.drop("PassengerId", axis=1).copy()
```

Stochastic Gradient Descent (SGD):

```
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)

sgd.score(X_train, Y_train)

acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
```

Random Forest:

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
```

Logistic Regression:

```
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)

Y_pred = logreg.predict(X_test)

acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
```

**K Nearest Neighbor:**

```
# KNN knn = KNeighborsClassifier(n_neighbors = 3) knn.fit(X_train,
Y_train) Y_pred = knn.predict(X_test) acc_knn =
round(knn.score(X_train, Y_train) * 100, 2)
```

**Gaussian Naive Bayes:**

```
gaussian = GaussianNB() gaussian.fit(X_train, Y_train) Y_pred =
gaussian.predict(X_test) acc_gaussian = round(gaussian.score(X_train,
Y_train) * 100, 2)
```

**Perceptron:**

```
perceptron = Perceptron(max_iter=5)
perceptron.fit(X_train, Y_train)

Y_pred = perceptron.predict(X_test)

acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
```

**Linear Support Vector Machine:**

```
linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)

Y_pred = linear_svc.predict(X_test)

acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
```

**Decision Tree**

```
decision_tree = DecisionTreeClassifier() decision_tree.fit(X_train,
Y_train) Y_pred = decision_tree.predict(X_test) acc_decision_tree =
round(decision_tree.score(X_train, Y_train) * 100, 2)
```

**Which is the best Model ?**

```
results = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent',
              'Decision Tree'],
    'Score': [acc_linear_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_decision_tree]})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df.head(9)
```

Bharati Vidyapeeth's College Of Engineering Lavale Pune.

	Model
Score	
92.82	Random Forest
92.82	Decision Tree
87.32	KNN
81.14	Logistic Regression
80.81	Support Vector Machines
80.70	Perceptron
77.10	Naive Bayes
76.99	Stochastic Gradient Decent

As we can see, the Random Forest classifier goes on the first place. Butfirst, let us check, how random-forest performs, when we use cross validation.

**K-Fold Cross Validation:**

K-Fold Cross Validation randomly splits the training data into **K subsetscalled folds**. Let’s image we would split our data into 4 folds ( $K = 4$ ). Ourrandom forest model would be trained and evaluated 4 times, using a different fold for evaluation everytime, while it would be trained on the remaining 3 folds.

The image below shows the process, using 4 folds ( $K = 4$ ). Every row represents one training + evaluation process. In the first row, the model get’s trained on the first, second and third subset and evaluated on the fourth. In the second row, the model get’s trained on the second, third andfourth subset and evaluated on the first. K-Fold Cross Validation repeats this process till every fold acted once as an evaluation fold.

Bharati Vidyapeeth’s College Of Engineering Lavale Pune.

Training	Training	Training	Evaluation
1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

The result of our K-Fold Cross Validation example would be an array that contains 4 different scores. We then need to compute the mean and the standard deviation for these scores.

The code below performs K-Fold Cross Validation on our random forest model, using 10 folds (K = 10). Therefore it outputs an array with 10 different scores.

```
from sklearn.model_selection import cross_val_score
rf = RandomForestClassifier(n_estimators=100)
scores = cross_val_score(rf, X_train, Y_train, cv=10, scoring =
"accuracy")
print("Scores:", scores)
print("Mean:", scores.mean())
print("Standard Deviation:", scores.std())
```

```
Scores: [ 0.76666667  0.82222222  0.7752809   0.82022472  0.85393258  0.86516854
 0.83146067  0.76404494  0.85393258  0.85227273]
Mean: 0.820520655998
Standard Deviation: 0.036733365466
```

This looks much more realistic than before. Our model has an average accuracy of 82% with a standard deviation of 4 %. The standard deviation shows us how precise the estimates are.

This means in our case that the accuracy of our model can differ + — 4%.

I think the accuracy is still really good and since random forest is an easy-to-use model, we will try to increase its performance even further in the following section.

## Random Forest

What is Random Forest ?

Random Forest is a supervised learning algorithm. Like you can already see from its name, it creates a forest and makes it somehow random. The „forest“ it builds, is an ensemble of Decision Trees, most of the time trained with the —bagging method. The general idea of the bagging method is that a combination of learning models increases the overall result.

To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

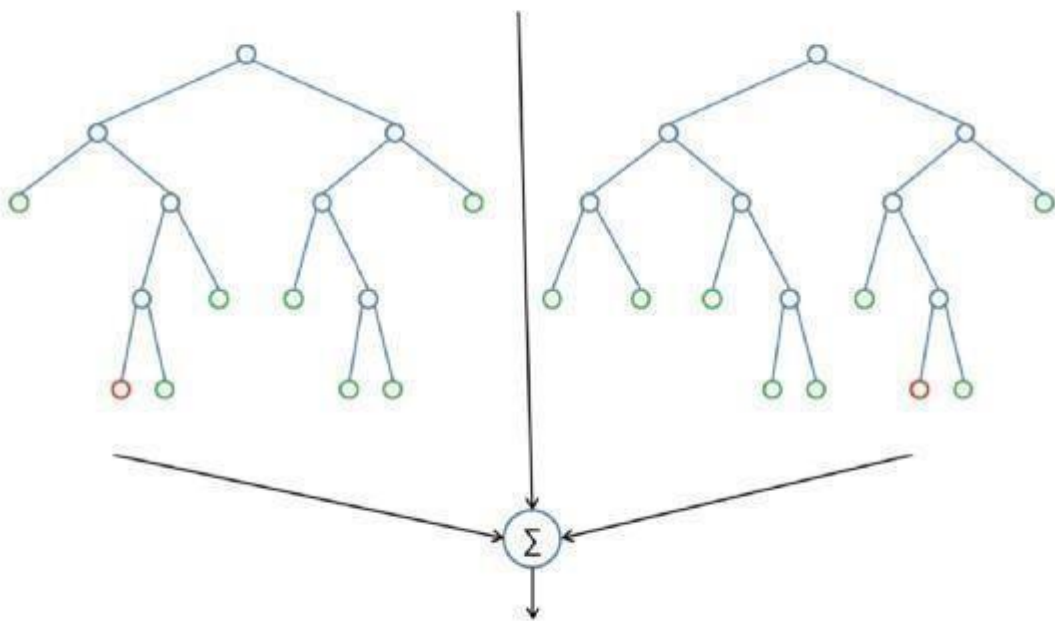


One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of current machine learning systems. With a few exceptions a random-forest classifier has all the hyperparameters of a decision-tree classifier and also all the hyperparameters of a bagging classifier, to control the ensemble itself.

The random-forest algorithm brings extra randomness into the model, when it is growing the trees. Instead of searching for the best feature while splitting a node, it searches for the best feature among a random subset of features. This process creates a wide diversity, which generally results in a better model. Therefore when you are growing a tree in random forest, only a random subset of the features is considered for splitting a node. You can even make trees more random, by using random thresholds on top of it, for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Below you can see how a random forest would look like with two trees:

Figure 1: A random forest with two trees. The output of the ensemble is the sum of the outputs of the individual trees.

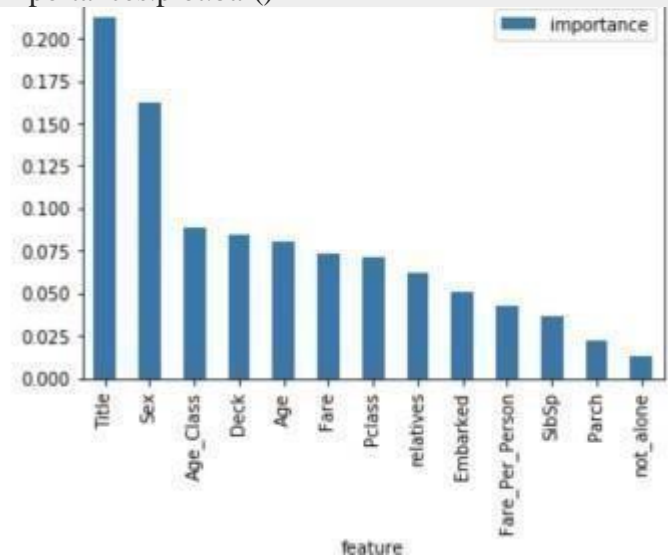


feature, reduce impurity on average (across all trees in the forest). It computes this score automaticall for each feature after training and scales the results so that the sum of all importances is equal to 1. We will acces this below:

```
importances =
pd.DataFrame({'feature':X_train.columns,'importance':np.round(random_f
orest.feature_importances_,3)})
importances =
importances.sort_values('importance',ascending=False).set_index('featu
re')importances.head(15)
```

	importance
feature	
Title	0.212
Sex	0.162
Age_Class	0.089
Deck	0.084
Age	0.080
Fare	0.073
Pclass	0.071
relatives	0.062
Embarked	0.051
Fare_Per_Person	0.043
SibSp	0.036
Parch	0.022
not_alone	0.013

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a157c1e10>
importances.plot.bar()
```



not\_alone and Parch doesn't play a significant role in our random forest classifiers prediction process. Because of that I will drop them from the dataset and train the classifier again. We could also remove more or less

features, but this would need a more detailed investigation of the feature effect on our model. But I think it's just fine to remove only Alone and Parch.

```
train_df = train_df.drop("not_alone", axis=1)
test_df = test_df.drop("not_alone", axis=1)

train_df = train_df.drop("Parch", axis=1)
test_df = test_df.drop("Parch", axis=1)
```

### Training random forest again:

```
# Random Forest

random_forest = RandomForestClassifier(n_estimators=100, oob_score =
True)
random_forest.fit(X_train, Y_train)
Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)

acc_random_forest = round(random_forest.score(X_train, Y_train) * 100,
2)
print(round(acc_random_forest,2), "%")
```

92.82%

Our random forest model predicts as good as it did before. A general rule is that, **the more features you have, the more likely your model will suffer from overfitting** and vice versa. But I think our data looks fine for now and hasn't too much features.

There is also another way to evaluate a random-forest classifier, which is probably much more accurate than the score we used before. What I am talking about is the **out-of-bag samples** to estimate the generalization accuracy. I will not go into details here about how it works. Just note that out-of-bag estimate is as accurate as using a test set of the same size as the training set. Therefore, using the out-of-bag error estimate removes the need for a set aside test set.

oob score: 81.82 %

```
print("oob score:", round(random_forest.oob_score_, 4)*100, "%")
```

4

Now we can start tuning the hyperparameters of random forest.

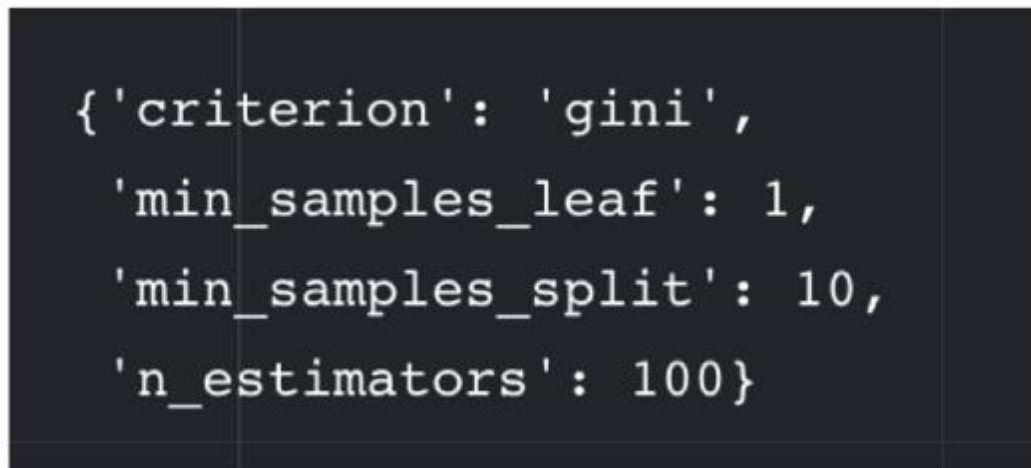
## Hyperparameter Tuning

Below you can see the code of the hyperparameter tuning for the parameters criterion, min\_samples\_leaf, min\_samples\_split and n\_estimators.

I put this code into a markdown cell and not into a code cell, because it takes a long time to run it.

Directly underneath it, I put a screenshot of the gridsearch's output.

```
param_grid = { "criterion" : ["gini", "entropy"], "min_samples_leaf" :  
[1, 5, 10, 25, 50, 70], "min_samples_split" : [2, 4, 10, 12, 16, 18,  
25, 35], "n_estimators": [100, 400, 700, 1000, 1500]}  
from  
sklearn.model_selection import GridSearchCV, cross_val_score  
rf =  
RandomForestClassifier(n_estimators=100, max_features='auto',  
oob_score=True, random_state=1, n_jobs=-1)  
clf =  
GridSearchCV(estimator=rf, param_grid=param_grid, n_jobs=-  
1)  
clf.fit(X_train, Y_train)  
clf.best_params
```



```
{'criterion': 'gini',  
 'min_samples_leaf': 1,  
 'min_samples_split': 10,  
 'n_estimators': 100}
```

### Test new Parameters:

```
# Random Forest  
random_forest = RandomForestClassifier(criterion = "gini",  
                                     min_samples_leaf = 1,  
                                     min_samples_split = 10,  
                                     n_estimators=100,  
                                     max_features='auto',  
                                     oob_score=True,  
                                     random_state=1,  
                                     n_jobs=-1)  
  
random_forest.fit(X_train, Y_train)  
Y_prediction = random_forest.predict(X_test)  
  
random_forest.score(X_train, Y_train)  
  
print("oob score:", round(random_forest.oob_score , 4)*100, "%")
```

oob score: 83.05 %

Now that we have a proper model, we can start evaluating it's performace in a more accurate way. Previously we only used accuracy and the oob score, which is just another form of accuracy. The problem is just, that it'smore complicated to evaluate a classification model than a regression model. We will talk about this in the following section.

### Further Evaluation

Confusion Matrix:

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
predictions = cross_val_predict(random_forest, X_train, Y_train, cv=3)
confusion_matrix(Y_train, predictions)
```

`array([[488, 61],  
 [ 95, 247]])`  
The first row is about the not-survived-predictions: **493 passengers were correctly classified as not survived** (called true negatives) and **56 where wrongly classified as not survived** (false positives).

The second row is about the survived-predictions: **93 passengers wherewrongly classified as survived** (false negatives) and **249 where correctly classified as survived** (true positives).

A confusion matrix gives you a lot of information about how well yourmodel does, but theres a way to get even more, like computing the classifiers precision.

Precision and Recall:

```
from sklearn.metrics import precision_score, recall_score

print("Precision:", precision_score(Y_train, predictions))
print("Recall:", recall_score(Y_train, predictions))
```

Precision: 0.801948051948

Recall: 0.722222222222

Our model predicts 81% of the time, a passengers survival correctly (precision). The recall tells us that it predicted the survival of 73 % of the people who actually survived.

### F-Score

You can combine precision and recall into one score, which is called the F-score. The F-score is computed with the harmonic mean of precision and recall. Note that it assigns much more weight to low values. As a result of that, the classifier will only get a high F-score, if both recall and precision are high.

```
from sklearn.metrics import f1_score
f1_score(Y_train, predictions)
0.7599999999999999
```

There we have it, a 77 % F-score. The score is not that high, because we have a recall of 73%. But unfortunately the F-score is not perfect, because it favors classifiers that have a similar precision and recall. This is a problem, because you sometimes want a high precision and sometimes a high recall. The thing is that an increasing precision, sometimes results in a decreasing recall and vice versa (depending on the threshold). This is called the precision/recall tradeoff. We will discuss this in the following section.

### Precision Recall Curve

For each person the Random Forest algorithm has to classify, it computes a probability based on a function and it classifies the person as survived (when the score is bigger than the threshold) or as not survived (when the score is smaller than the threshold). That's why the threshold plays an important part.

We will plot the precision and recall with the threshold using matplotlib:

```
from sklearn.metrics import precision_recall_curve

# getting the probabilities of our predictions
y_scores = random_forest.predict_proba(X_train)
y_scores = y_scores[:,1]

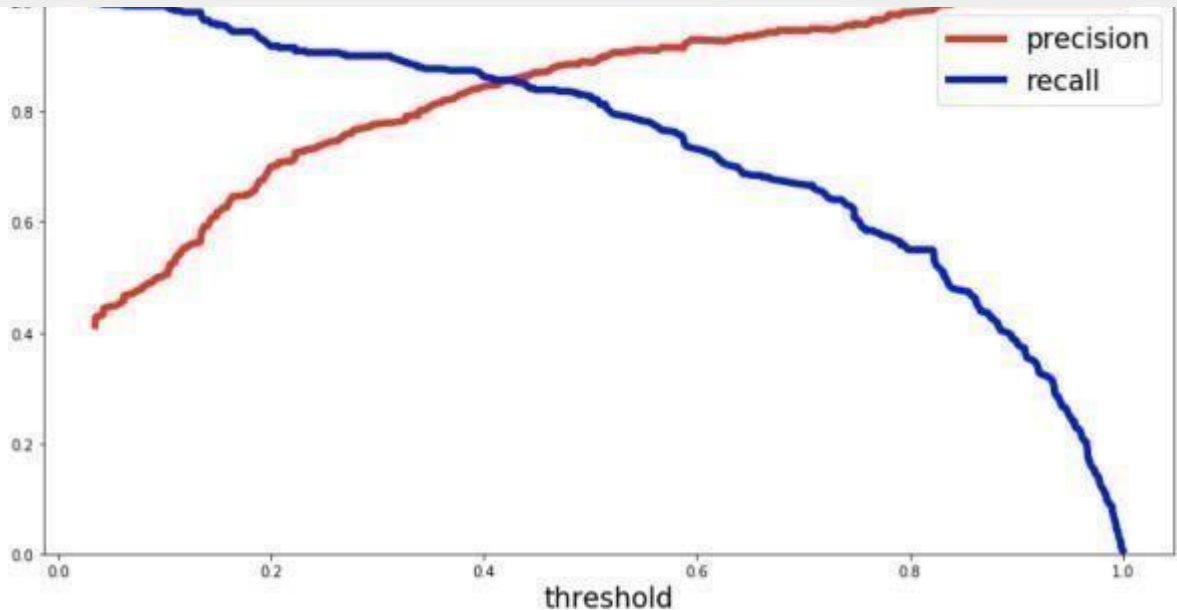
precision, recall, threshold = precision_recall_curve(Y_train,
```

```

y_scores)def plot_precision_and_recall(precision, recall, threshold):
    plt.plot(threshold, precision[:-1], "r-", label="precision",
linewidth=5)
    plt.plot(threshold, recall[:-1], "b", label="recall", linewidth=5)
    plt.xlabel("threshold", fontsize=19)
    plt.legend(loc="upper right", fontsize=19)
    plt.ylim([0, 1])

plt.figure(figsize=(14, 7))
plot_precision_and_recall(precision, recall, threshold)
plt.show()

```



Above you can clearly see that the recall is falling rapidly at a precision of around 85%. Because of that you may want to select the precision/recall tradeoff before that — maybe at around 75 %.

You are now able to choose a threshold, that gives you the best precision/recall tradeoff for your current machine learning problem. If you want for example a precision of 80%, you can easily look at the plots and see that you would need a threshold of around 0.4. Then you could train a model with exactly that threshold and would get the desired accuracy.

Another way is to plot the precision and recall against each other:

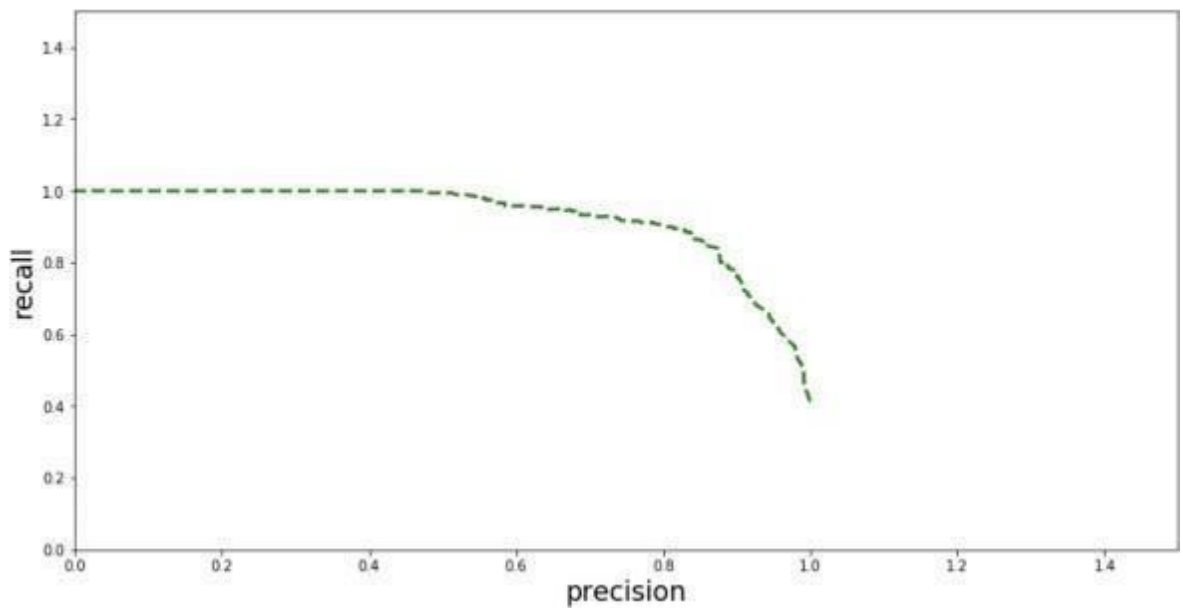
```

def plot_precision_vs_recall(precision, recall):
    plt.plot(recall, precision, "g--", linewidth=2.5)
    plt.ylabel("recall", fontsize=19)
    plt.xlabel("precision", fontsize=19)
    plt.axis([0, 1.5, 0, 1.5])

plt.figure(figsize=(14, 7))
plot_precision_vs_recall(precision, recall)
plt.show()

```



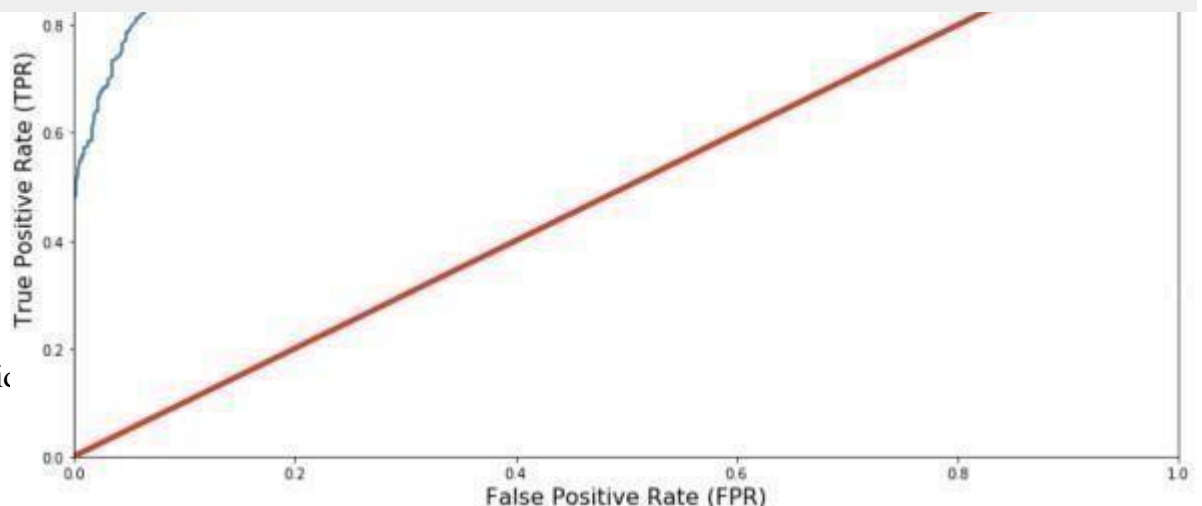


### ROC AUC Curve

Another way to evaluate and compare your binary classifier is provided by the ROC AUC Curve. This curve plots the true positive rate (also called recall) against the false positive rate (ratio of incorrectly classified negative instances), instead of plotting the precision versus the recall.

```
from sklearn.metrics import roc_curve
# compute true positive rate and false positive rate
false_positive_rate, true_positive_rate, thresholds =
roc_curve(Y_train, y_scores) # plotting them against each other
def plot_roc_curve(false_positive_rate, true_positive_rate,
label=None):
    plt.plot(false_positive_rate, true_positive_rate, linewidth=2,
label=label)
    plt.plot([0, 1], [0, 1], 'r', linewidth=4)
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate (FPR)', fontsize=16)
    plt.ylabel('True Positive Rate (TPR)', fontsize=16)

plt.figure(figsize=(14, 7))
plot_roc_curve(false_positive_rate, true_positive_rate)
plt.show()
```



Bharati Vic

The red line in the middle represents a purely random classifier (e.g. a coin flip) and therefore your classifier should be as far away from it as possible. Our Random Forest model seems to do a good job.

Of course we also have a tradeoff here, because the classifier produces more false positives, the higher the true positive rate is.

### ROC AUC Score

The ROC AUC Score is the corresponding score to the ROC AUC Curve. It is simply computed by measuring the area under the curve, which is called AUC.

A classifier that is 100% correct, would have a ROC AUC Score of 1 and a completely random classifier would have a score of 0.5.

```
from sklearn.metrics import roc_auc_score
r_a_score = roc_auc_score(Y_train, y_scores)
print("ROC-AUC-Score:", r_a_score)
```

ROC\_AUC\_SCORE: 0.945067587