SRI VENKATESWARA INTERNSHIP PROGRAM FOR RESEARCH IN ACADEMICS (SRI-VIPRA)

Project Report of 2022: SVP-2231

"Machine Learning Methods and its Applications"

IQAC
Sri Venkateswara College
University of Delhi
Dhaula Kuan
New Delhi - 110021

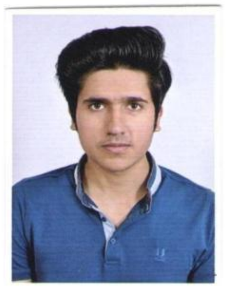
Name of the Mentor: Mr. Anirban Chatterjee **Name of Department:** Mathematics **Designation:** Assistant Professor	

SRI VIPRA PROJECT 2022

Title: Machine Learning Methods and its Applications

List of students under SRI VIPRA Project-

S.No.	**Name of student**	**Course**	**Photo**
1.	Uditi Bajaj (Roll no. 1720124)	B.Sc.(H) Mathematics (Sem-5)	
2.	Rishika Panwar (Roll No. 1720085)	B.Sc.(H) Mathematics (Sem-5)	
3.	Abhya Ahuja (Roll No. 1720022)	B.Sc.(H) Mathematics (Sem-5)	

4.	Hitaishi (Roll No. 1720126)	B.Sc.(H) Mathematics (Sem-5)	
5.	Pranavi K (Roll No. 1720075)	B.Sc.(H) Mathematics (Sem-5)	
6.	Harsh Yadav (Roll no. 1920018)	B.Sc. (H) Statistics (Sem 5)	
7.	Shubham Snehil (Roll No. 1920052)	B.Sc.(H) Statistics (Sem-5)	
8.	Aryaman Bawa (Roll No. 0520036)	B.A.(H) Economics (Sem-5)	

Signature of Mentor

Certificate

This is to certify that the aforementioned students from Sri Venkateswara College have participated in the summer project SVP-2231 titled “Machine Learning Methods and its Applications”. The participants carried out research project work under my guidance and supervision from July 1, 2022, to September 20, 2022. The teaching and work carried out were held in a mixed format, both online and offline.

(Signature of Mentor)

Mr. Anirban Chatterjee
Assistant Professor
Department of Mathematics
Sri Venkateswara College
University of Delhi

Acknowledgement

We would like to extend our heartfelt gratitude to respected Mr. Anirban Chatterjee for showing his faith in us and believing that we were capable enough to be a part of SRI-VIPRA. Machine Learning and Data Science have always been something that fascinated us and getting this opportunity to study some of its concepts and how they can be applied to an industrial-level problem has piqued our curiosity to continue learning and researching more on these topics. During this period, sir taught us the basics of statistical concepts and then gradually advanced to a higher level of analysis while providing a deeper understanding of each concept. We are extremely thankful to him for guiding us in every way possible.

Also, we would like to thank the authorities of Sri Venkateswara College to have conducted this internship program as it gave us a lot of exposure.

CONTENTS

Abstract

This project is everything about exploring the most demanding topic of today's technological nature, i.e., machine learning and its types. We are here mainly focusing on the major types of machine learning and their applications. Along with its myriad applications, we will come across the various machine learning systems and deeply analyze the mathematical and statistical models like overfitting & underfitting, linear regression, logical regression, and many more. It also examines the gradient descent algorithm that is most important in minimizing the cost function to get the best model with high efficiency. Apart from these, this project also focuses on SVM, KNN, and Naive Bayes Classifier in detail and deeply its functions incorporating the required mathematical and statistical models.

The aim of this project is first to deep dive into the theoretical aspects of descriptive statistics, overfitting & underfitting, gradient descent, linear regression, logistic regression, performance metrics, support vector machine (SVM), k-nearest neighbor (KNN), and Naive Bayes classifier and then incorporate them into its application. Our primary focus is to do exploratory data analysis (EDA) and predict results by building mathematical and statistical models. We have done EDA on the basketball and healthcare data sets. We use the kernel function in SVM to manipulate data. And, the Naive Bayes Classifier to predict whether or not the patient has diabetes. And also uses, the KNN Classifier on a medical data set and makes the statistical model predict the condition of the patient depending on the received test results. We utilize the Jupyter Notebook to document our python code that aims at creating such models. We also hope to achieve a sufficient level of accuracy for our model to work correctly.

Towards the end of this project, we also observe the drawbacks of our model and move on to further discuss the bright future that data science and artificial intelligence hold while also spreading their applications to a wide variety of fields, including healthcare, sports, etc.

Descriptive Statistics

A measure of central tendency is a single value that tries to represent a dataset by identifying the central position within that set of data. Measures of central tendency are also called measures of central location. The mean (often called the average) is the most popular measure of central tendency. Other central tendencies are the median and mode.
The mean, median and mode are all valid measures of central tendency. Some situations make one more appropriate than the others. In this summary, we will describe these measures and advise when they should be invoked.

<u>Mean</u>

The mean is equal to the sum of all the values in the data divided by the cardinality of the dataset. So, if we have n values in a data set and they have values $x_1, x_2, x_3 \dots x_n$, the sample mean x¯is:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$$

This can be written more concisely with the summation notation,

$$\bar{x} = \frac{(\sum_1^n x_i)}{n}$$

As this mean is derived from a dataset of n points, we refer to it as the sample mean. The mean of the entire population is denoted by μ.

Important features of the mean:

- It minimizes error in the prediction of any one value in your data set.
- Its calculation involves every point in the dataset.
- It the only measure of central tendency where the sum of deviations of each value from the mean is always zero.
- It is generally used when the data is continuous

Drawback(s):

Means are susceptible to outliers which might cause some misrepresentation of the data. For example: If we measure the heights of 5 people and their heights (in meters) are 1.67, 1.62, 1.65, 1.64, 6. The average height here would be 2.52, which is not a good representation of the general trend in heights even within this small sample. Also, what’s disconcerting is that we may have giants walking among us.

Median
The median denotes the value lying at the midpoint of a frequency distribution of observed values or quantities, so that there is an equal probability of a data point being greater than or less than it. It is also commonly referred to as the 50th percentile. To find the median of the distribution it must first be sorted, and then the value in the 'middle' (in case of odd number of points) or the average of the two terms in the 'middle' (in case of even number of points) is selected as the median of the distribution.

Mode
The mode is the most commonly recurring value in the dataset. It is generally considered when dealing with categorical variables, where the most common category must be ascertained.

Skewed Distributions

Often, we check if the data is distributed normally as many statistical tests require this underlying assumption. When the data is normally distributed, the usual measure for central tendency that is considered is the mean even though in a normal distribution the mean and median are the same. This is done because the mean is more representative of the data since all the data points are used in its calculation.
However, if the data is skewed in either direction, the mean is pulled towards the direction in which the data is skewed. In these situations, the median is usually considered as the measure for central tendency. The greater the skewness, the greater is the distance between the mean and the median.

Linear Regression

In statistics, we use regression to model a target (dependent) variable based on some explanatory (independent) variables. We use this technique to establish a cause-and-effect relationship between variables.
Simple Linear Regression involves one independent variable and a linear relationship between the independent and dependent variables.
The equation used for the same is:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

where $y = dependent\ variable, x = independent\ variable, \beta_0 = intercept, \beta_1 = slope\ and\ \varepsilon = error$

In simple linear regression, we aim to find the best value for β_0 and β_1.
In order to achieve this, we find the line of best fit. Any line would be termed as best if the error between the predicted values of y and actual values of y is the minimum.
This now becomes a minimization problem with the error function as

$$j = \frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} - y_i)^2$$

Where, $\hat{y_i}$= predicted value and y_i= actual value
The function J is called the cost function or mean squared error (MSE) function.
Here, we square the error term because of these few reasons:

1. When we consider $(\hat{y_i} - y_i)$, some error values might get eliminated because of opposite signs and the overall error be reduced but that wouldn't indicate a good line of fit.
2. One might go ahead with $|\hat{y_i} - y_i|$ this function in order to make sure that error terms don't result in nullification but here, this function might not be differentiable everywhere. So, we square the term since graphs of even functions are easier to derive.
3. Also, $|\hat{y_i} - y_i|$ is more robust to outliers when compared with the squared term.

We also follow some assumptions for SLR:

1. y is independent of the error term
2. The variance of residuals is same for all values of x
3. The mean value of residuals is zero

Assumption 2 and 3 suggests that the error term is normally distributed. Here is an explanation of why the mean should be zero.
Let linear regression model be $y_i = \beta_1 + \beta_2 X2_i + \beta_3 X3_i + \ldots + \beta_k X_i + u_i$ (a data point in K dimensions)
Assume E $(U_i | X2_i, X3_i, \ldots, Xk_i)$ =W (W is a constant, in standard model W=0)
Conditional expectations of the equation for y_i can be expressed as
a. E $(y_i | X2_i, X3_i, \ldots, Xk_i) = \beta_1 + \beta_2 X2_i + \beta_3 X3_i + \ldots + \beta_k X_i + W$

b. => $(\beta_1+W) + \beta_2 X2_i + \beta_3 X3_i+\ldots+ \beta_k X_i$
c. => $\alpha + \beta_2 X2_i + \beta_3 X3_i+\ldots+ \beta_k X_i$ where $\alpha=(\beta_1+W)$

Given the training data, the Xs are treated as constant while the β_S are the variables

If the assumption 3 is not fulfilled, we cannot solve the equation for β_1 !
Now, the question arises how do we minimize the square function?
For this, we use gradient descent algorithm

Gradient Descent:

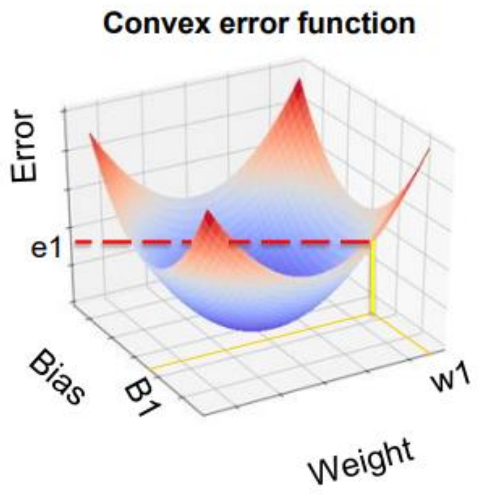
Gradient Descent is an optimization algorithm. An optimization algorithm helps us to **minimize (or maximize) an objective function (Error function) E(x) such as Sum of Squared Errors (SSE)** which is basically a mathematical function dependent on the model's learnable parameters which are used in computing the target
value(Y) from the set of predictors(X) used in the model. The optimizer algorithms try to estimate the values of wi and b in the cost function, C**= ½ ((wi. xi + b) – y)** which when used, will give minimum or maximum C. In ML we look for minimum value instead of the maximum value.

Gradient in *Gradient descent* refers to a vector that is tangent of a function and points in the direction of greatest increase of this function. Gradient is zero at a local maximum or minimum because there is no single direction of increase. In mathematics, gradient is defined as a partial derivative for every input variable of a function.
Here, we use the concept of Negative gradient which is a vector pointing at the greatest decrease of a function. Therefore, we can minimize a function by iteratively moving a little bit in the direction of negative gradient.

One must wonder why we use gradient descent instead of manually differentiating the cost function?
This is because, in problems wherein there are more than two weights/slopes that need to be adjusted, it is not manually possible to find the right combination of weights using brute force.

Graphical representation of Gradient Descent:

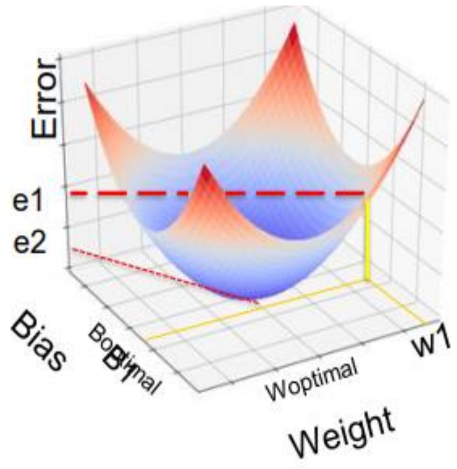
1. A random combination of bias B1 and input weights W1 (more than one is not possible to visualize)

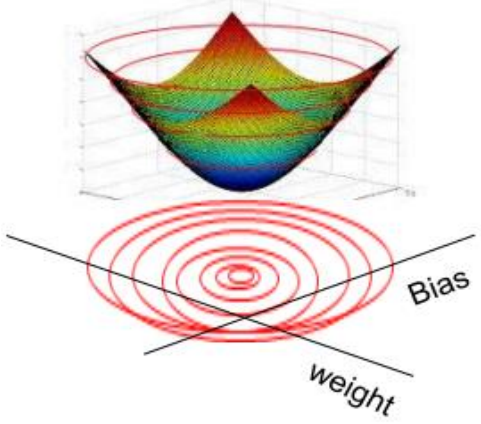


2. Each combination of W1 and B1 is one particular linear model in a neuron. That model is associated with proportionate error e1 (red dashed line).

3. Objective is to drive e1 towards 0. For which we need to find the optimal weight (Woptimal) and bias (Boptimal)

4. The algorithm uses gradient descent algorithm to change bias and weight from starting values of B1 and W1 towards the Boptimal, Woptimal.
This movement from starting weight and bias to optimal weight and bias may not happen in one shot. It is likely to happen in multiple iterations. The step size can be influenced using a parameter called ***Learning Rate***. It decides the size of the steps i.e., the amount by which the parameters are updated.
Too small a learning step will slow down the entire process while too large may lead to an infinite loop.





5. Every ring on the error function represents a combination of coefficients (m1 and m2 in the image) which result in same quantum of error i.e., SSE

6. Let us convert that to a 2d contour plot. In the contour plot, every ring represents one quantum of error.

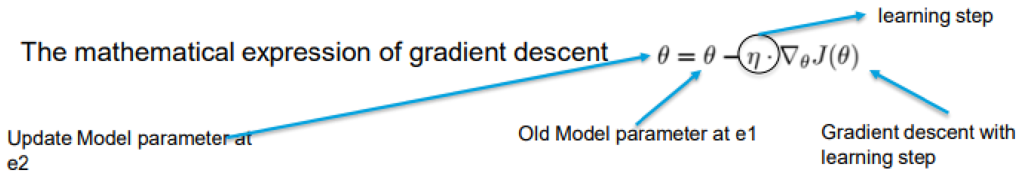
7. The innermost ring / bull's eye is the combination of the coefficients that gives the least SSE

If we look closely at the contour plot, we would find out the following points:

1. Outermost circle is highest error while innermost is the least error circle
2. A circle represents a combination of parameters which result in the same error. Moving on a circle will not reduce error.
3. Objective is to start from anywhere but reach the innermost circle.

Steps to evaluate Gradient Descent:

1. First evaluate dy(error)/d(weight) to find the direction of highest increase in error given a unit change in weight (Blue arrow). Partial derivative w.r.t. to weight
2. Next find dy(error) /d(bias) to find the direction of highest increase in error given a unit change in bias (green arrow). Partial derivative w.r.t. to bias
3. Partial derivatives give the gradient in the given axis and gradient is a vector
4. Add the two vectors to get the direction of gradient (black arrow) i.e., direction of max increase in error
5. We want to decrease error, so find the negative of the gradient i.e., opposite to black arrow (Orange arrow). The arrow tip is a new value of bias and weight.
6. Recalculate the error at this combination an iterate to step 1 till movement in any direction only increases the error


The mathematical expression of gradient descent
$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$
learning step
Update Model parameter at e2
Old Model parameter at e1
Gradient descent with learning step

Test statistic: R^2

It is called the coefficient of determination and explains variation in y around its mean. One can think of it as a measure of variability between dependent and independent variables (how much variance can the model explain about the dependent variable). A high value suggests that there is less difference between predicted and actual values.
We use this formula to calculate R^2:

$$\begin{aligned} R^2 &= 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}, \\ &= 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}. \end{aligned}$$

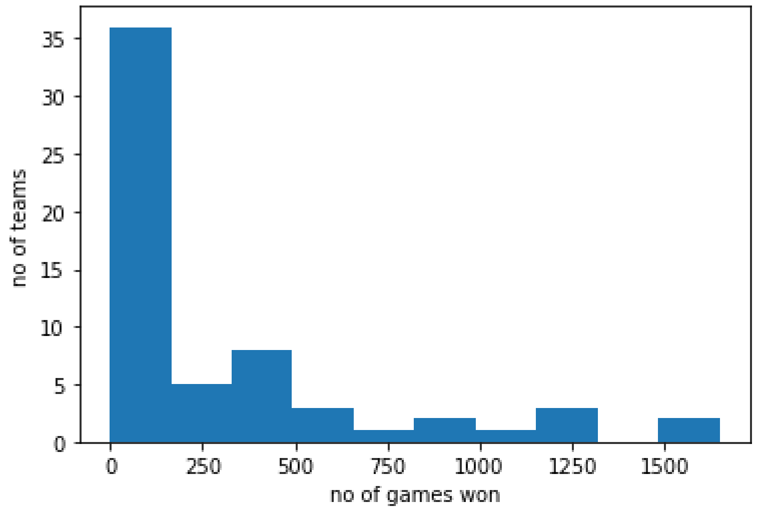
Exploratory Data Analysis on Basketball Teams Dataset

Basketball EDA task

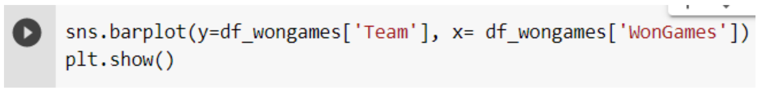
The given dataset about different basketball teams had 12 columns including Team Tournament, Score, PlayedGame, WonGames, DrawnGames, LostGames, BasketScored, BasketGiven, TournamentChampion, Runner-up, TeamLaunch, HighestPositionHeld.

There were 61 entries in the dataset and there were no null entries in the dataset.

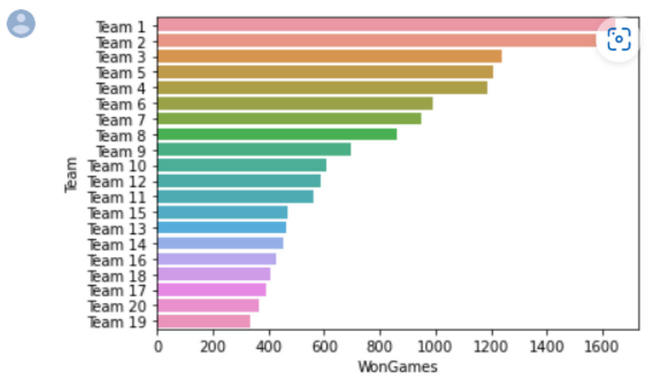
We started with analyzing the number of games won and saw that the data is right skewed with most of the teams winning 0-200 matches and few winning more than 250 matches.

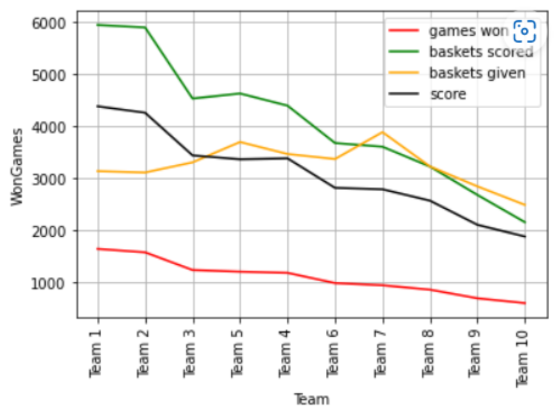


We used seaborn library to visualize the next part of the dataset. To find the top 20 teams with highest score and the greatest number of matches won, we used the following code snippet:



```
sns.barplot(y=df_wongames['Team'], x= df_wongames['WonGames'])
plt.show()
```

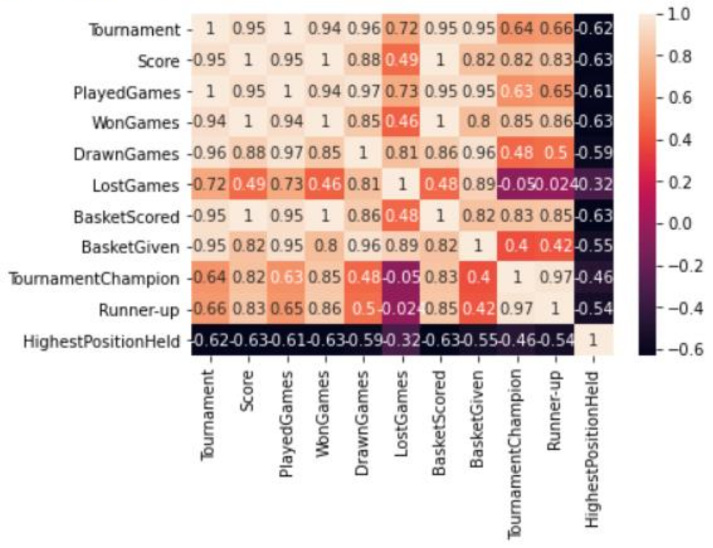




We then used a heatmap to find correlations among the various variables and found that there is a very strong correlation between games won, baskets scored, and score

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

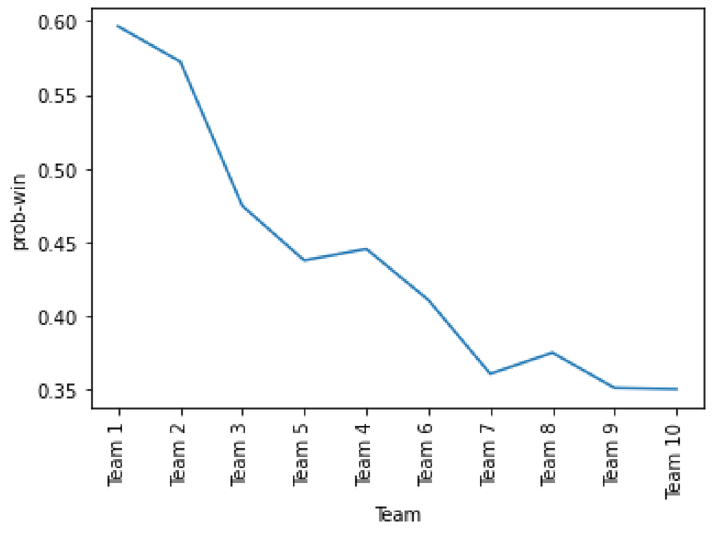
```
<AxesSubplot:>
```



We also added a new column called the winning probability and losing probability along with winning percentage by using the following code:

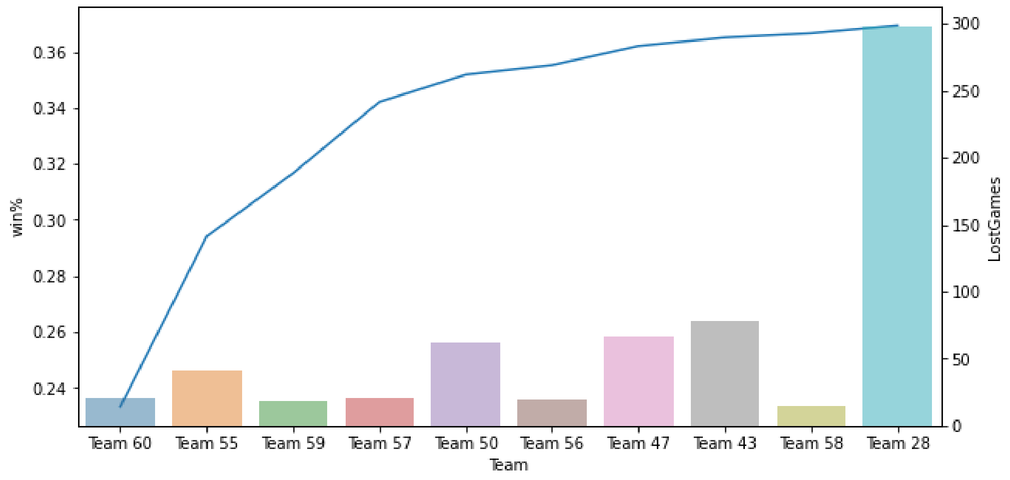
```
x=0
df['prob'] =0
for x in range (61):
      df['prob']. iloc[x]= df['WonGames']. iloc[x]/ df['PlayedGames'].
iloc[x]
       df['lost']. iloc[x]= df['LostGames']. iloc[x]/df['PlayedGames'].
iloc[x]
       df['win%']. iloc[x]= (df['WonGames']. iloc[x]+ 0.5*
df['DrawnGames']. iloc[x])/ df['PlayedGames']. iloc[x]
```

Here is the scatter plot obtained

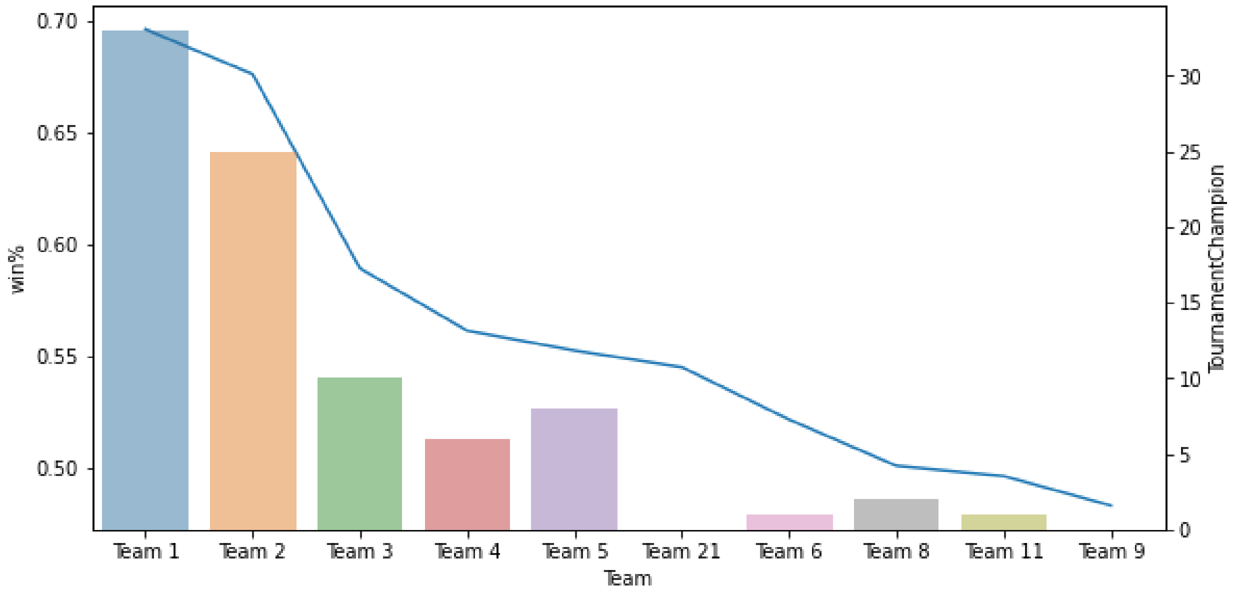


We used lineplot to find the teams with lowest performance:

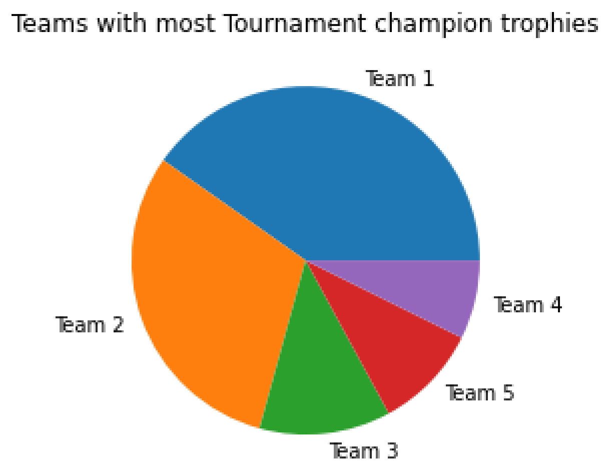
```
ax1 = sns.set_style (style=None, rc=None)

fig, ax1 = plt. subplots (figsize= (10,5))

sns. lineplot ('Team', 'win%', data = df. nsmallest (10, 'win%'))
ax2 = ax1.twinx()
sns. barplot (data = df. nsmallest (10, 'win%'), x='Team', y='LostGames',
alpha=0.5, ax=ax2)
```



Similarly, the highest winning% teams are:



We then used a pie chart to find the teams with most tournament champion trophies



All this EDA helped us reach to the conclusion that we can approach any of the top 5 teams, namely TEAM 1,2,3,4,5 for the next matches with high possibility of winning.

Logistic Regression

The classification problem is when the values we now want to predict take on a small number of discrete values. For this report, we will focus on the binary classification problem in which y can take on only two values, 0 and 1. (Most of what we show here will also generalize to the multiple-class case.)
Given $x^{(i)}$ (parameterized by θ), the corresponding $y^{(i)}$ is also called the label for the training example.
Intuitively, we want the hypothesis $h_\theta(x)$, to be in the range (0,1) when we know that $y \in \{0, 1\}$.
We choose the hypothesis to be,
$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$, where $g(z) = \frac{1}{1+e^{-z}}$, is called the logistic function or the sigmoid function.
$g(z)\to 0$ when $z\to -\infty$ and $g(z)\to 1$ when $z\to\infty$

A useful property of the derivative of the sigmoid function is that,

$$g'(z) = \frac{d}{dz}\frac{1}{1+e^{-z}}$$

$$= \frac{1}{(1+e^{-z})^2}e^{-z}$$

$$= \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right) = g(z)(1-g(z))$$

Learning in Logistic Regression

Given the logistic regression model, we can fit θ for it under a set of probabilistic assumptions via maximum likelihood.
Assuming,

$P(y = 1|x;\ \theta) = h_\theta(x)$

$P(y = 0|x; \theta) = 1 - h_\theta(x)$

A succinct way to formulize the above is,

$p(y = 0|x; \theta) = h_\theta(x)^y \left(1 - h_\theta(x)\right)^{1-y}$

If we have m independently generated training examples, the likelihood L(θ) of the parameters may be written as,

$L(\theta) = \ \ \Pi_i^m\ p(y^{(i)}|x^{(i)};\ \theta)$

$$= \Pi_i^m\ h_\theta\left(x^{(i)}\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y}$$

Taking the log of likelihood would make it easier to maximize. However, convention dictates that the objective function should be minimized so we change the sign. So,

$$l(\theta) = -\log L(\theta)$$

$$= \sum_i^m y^{(i)} \log h\left(x^{(i)}\right) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

This function is also known as the cross-entropy loss. We may minimize this loss using gradient descent, which allows us to minimize the function with the steepest descent. Taking one training example (*x*, *y*) and calculating the gradients for the gradient descent update rule.

$$\frac{\partial}{\partial \theta_j} l(\theta) = -(y\frac{1}{g(\theta^T x)} - (1-y)\frac{1}{1-g(\theta^T x)})\frac{\partial}{\partial \theta_j} g(\theta^T x)$$

$$= -\left(y\left(1-g(\theta^T x)\right) + (1-y)\left(g(\theta^T x)\right)\right) x_j$$

$$= -\left(y - h_\theta\,(x)\right) x_j$$

Therefore, our gradient update rule is

$$\theta_j := \theta_j - \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

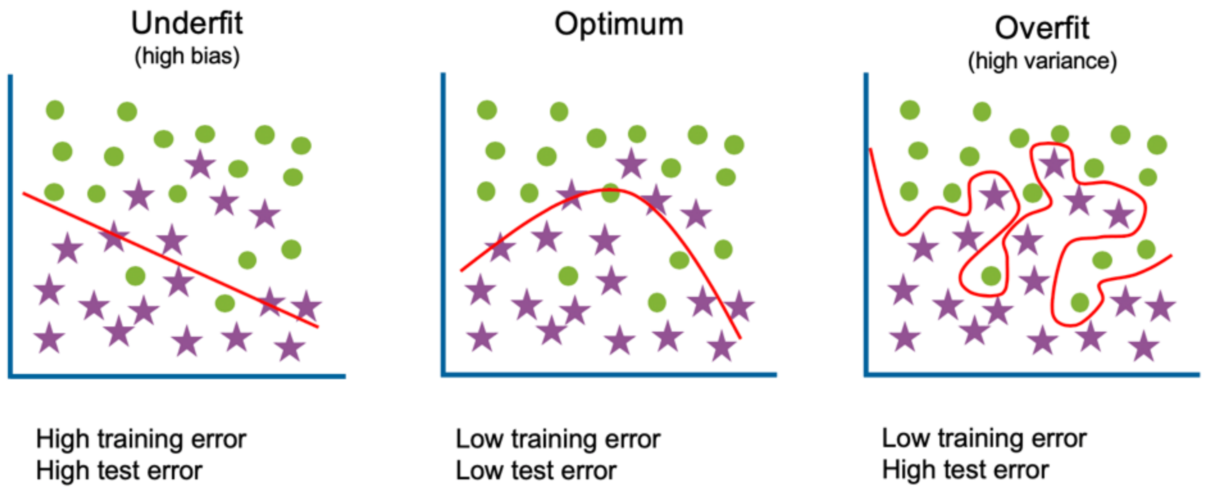
Underfitting and Overfitting

What is Overfitting?
Overfitting occurs when a statistical model fits exactly against training data. When this happens, the algorithm becomes inaccurate against testing data. Generalisation, which allows a model to predict and classify data, becomes difficult.
When a machine learning algorithm is constructed, they leverage a sample dataset to train the model. However, if model is complex or trains too long on sample data, it starts to learn the noise in the dataset. When model learns the noise, it fits too closely to the training set, the model is overfitted and unable to generalise new data well. In such cases, the model can’t generalise well to new data and not perform classification or prediction tasks that it was meant for.

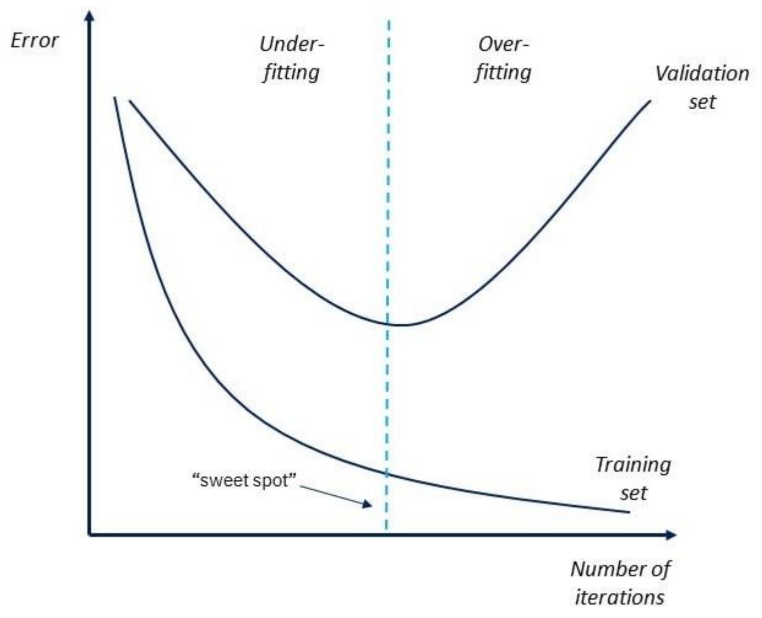
High variance and low error rates indicate overfitting. To prevent this, part of the training set is set aside as a test set to check overfitting. If training data has low error rate but test data has a high corresponding value, there could be overfitting.

Overfitting vs Underfitting

If training process is ended early or less relevant variables are eliminated in an attempt to prevent overfitting, the model may be underfitted. Underfitting occurs when the model is not trained long enough or variables are not significant enough to determine a meaningful relationship between the variables.



In either case, the model can’t establish the dominant trend within the training set and generalise poorly. An underfitted model experiences high bias and low variance in the predictions. This is the bias variance tradeoff. When fitting a model, the goal is to find the right balance between underfitting and overfitting to establish a dominant trend to be applied to real datasets.

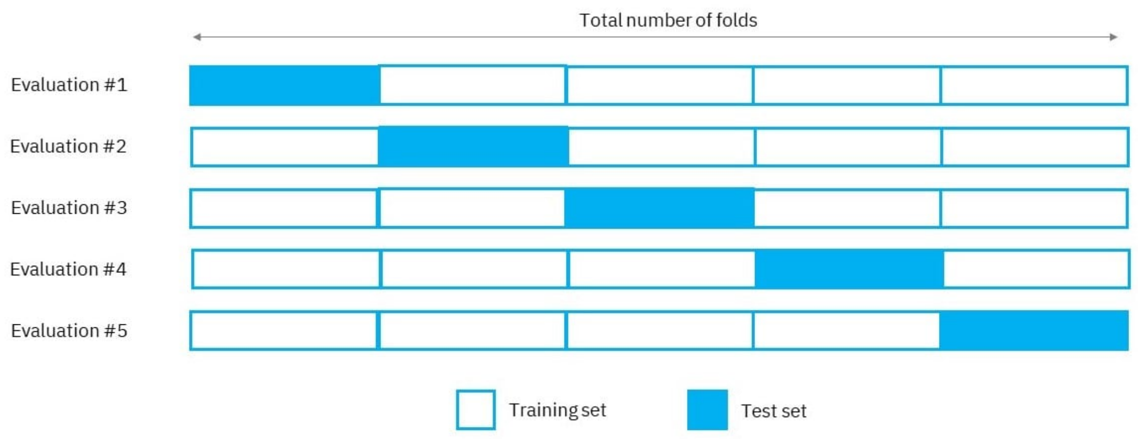


Detecting Overfitting

To understand the accuracy of models, it is important to test for fitness. K-fold cross-validation is one of the most popular model accuracy tests.

In k-folds cross-validation, the dataset is split into k equal subsets, called folds. One of the folds will act as the test set, called the validation set while the remaining models train the model. The process is repeated until each fold is used as the validation set. After each evaluation, a score is retained and once all evaluations are done, the scores are averaged to calculate the performance of the model.

Suppose we divide the dataset into 5 subgroups. The process can be visualized like this:



Performance Metrics for Classification

Evaluating the performance of a Machine learning model is one of the important steps while building an effective ML model. For this purpose, various metrics are used, known as

PREDICTED	**Positives (1)**	**TP**	**FP**

performance or evaluation metrics. The metrics chosen to evaluate the ML model are very important as they influence how the performance of the algorithms are measured and compared.

Types of Metrics:

1. Confusion Matrix
2. Accuracy
3. Precision
4. Recall
5. F-Score

1. Confusion Matrix:

Confusion matrices are used to evaluate classification problems where the output can be of two or more classes.
For example, let's say we are solving a classification problem where we are predicting whether a person is having cancer or not.

Assigning values:

1 – Person has cancer
0 – Person doesn't have cancer

The confusion matrix is a table with two dimensions (Actual and Predicted) and sets of classes in both dimensions.

Columns – Actual classifications
Rows - Predicted classifications

Negatives (0)	**FN**	**TN**

ACTUAL

Positives (1) **Negatives (0)**

Terminology:

1. True Positive (TP): Actual class of the data point is 1 (True) and the predicted is also 1 (True).

Eg: The person has cancer (1) and the model classifies their case as cancer (1).

2. True Negative (TN): Actual class of the data point is 0 (False) and the predicted is also 0 (False).

Eg: The person doesn’t have cancer (0) and the model classifies their case as not cancer (0).

3. False Positive (TP): Actual class of the data point is 0 (False) and the predicted is 1 (True).

Eg: The person doesn’t have cancer (0) and the model classifies their case as cancer (1).

4. False Negative (FN): Actual class of the data point is 1 (True) and the predicted is 0 (False).

Eg: The person has cancer (1) and the model classifies their case as not cancer (0).

In an ideal scenario, the model gives 0 False Positives and 0 False Negatives but most models aren’t that accurate. Depending on the type of model, you can decide if minimizing the number of False Positives is more important than minimizing the number of False Negatives.

2. Accuracy:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ number\ of\ predictions}$$

Accuracy is a simple metric to use but is only suitable for cases where the target variable classes are approximately balanced.

When To Use Accuracy?

Eg: 60% classes in fruit images data are apples and 40% are oranges.
A model that predicts whether a new image is an apple or orange 97% of the time is a very good measure.

When Not to Use Accuracy?

Eg: In a model that detects cancer in patients, out of the 100 people, let 5 of them have cancer. If the model is bad and classifies every case as not having cancer, then the 95 non-cancer patients are diagnosed accurately whereas the 5 cancerous patients are diagnosed inaccurately which can be very dangerous. Even though the model is bad at predicting cancer, its accuracy is still 95%.

3. Precision:

Precision is used to overcome the limitations of Accuracy by determining the proportion of positive predictions that were actually correct.

$$\boldsymbol{Precision = \frac{TP}{(TP + FP)}}$$

Eg: In the cancer model, it determines the proportion of patients who had cancer and were also diagnosed as having cancer.

4. Recall or Sensitivity:

Though similar to Precision, it is used to determine the proportion of positive predictions that were incorrect.

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

Thus, maximizing Precision minimizes FP errors and maximizing Recall minimizes FN errors.

5. F – Scores:

F – Scores are used in cases where we need both Precision and Recall. Since simply taking the arithmetic mean of the two metrics will give a highly biased answer in some situations, we calculate their harmonic mean instead which is more balanced.

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$

Support Vector Machines

What is a Support Vector Machine?

The objective of SVM algorithm is to find a hyperplane in an N- dimensional space, N is the number of features, to classify the data.

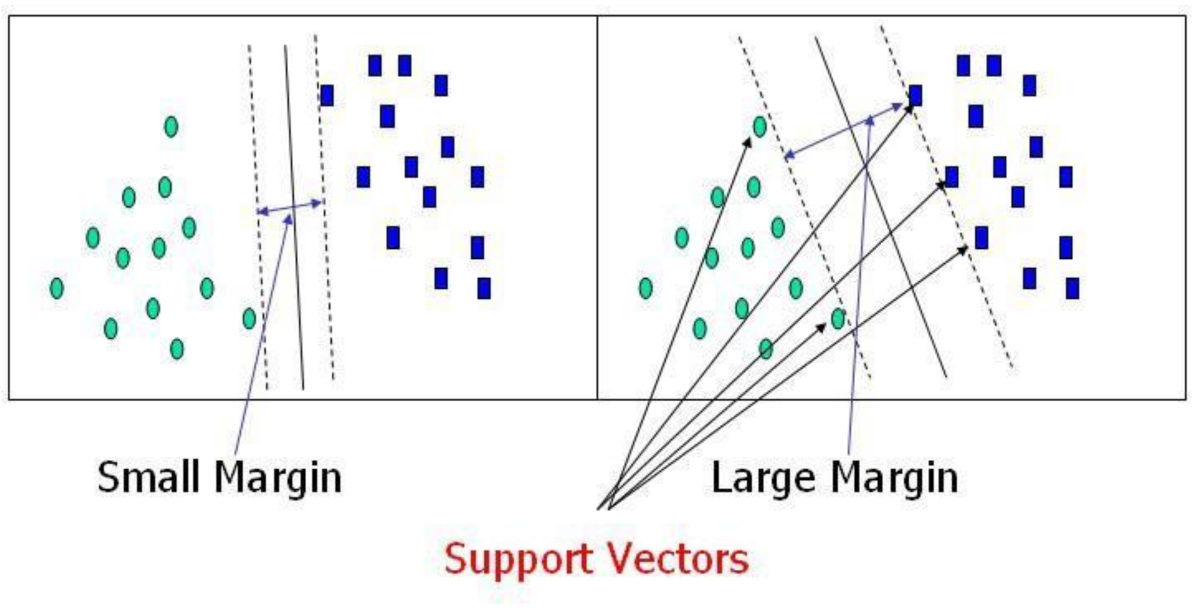
The idea behind SVM is:

1. We start with data in a relatively low dimension, let's start with data in 1-Dimension.

2. We move the data into a higher dimension, say 2-Dimensional.

3. Now we find a Support Vector Classifier that separates the data into two groups.

We select the hyperplane with the highest margin in order to classify new data points with the greatest accuracy. There are many possible hyperplanes, but we select the one with the highest margin.



What is a support vector?

Orientation and position of the hyperplane are influenced by support vectors. The margin of the classifier is maximized with these support vectors. The position of the hyperplane will change if support vectors are removed, the SVM's are built around these points.

But now the next question arises as to how we transform the data?

SVM's use something called the kernel functions to systematically find support vector classifiers in higher dimensions.

Kernel Functions:

The term "Kernel" refers to the mathematical functions used in Support Vector Machines for manipulating data. As a result, Kernel Functions generally transform the training data so that non-linear decision surfaces can be transformed into linear equations in more dimensions.

Radial Kernel/Radial Basis Function Kernel:

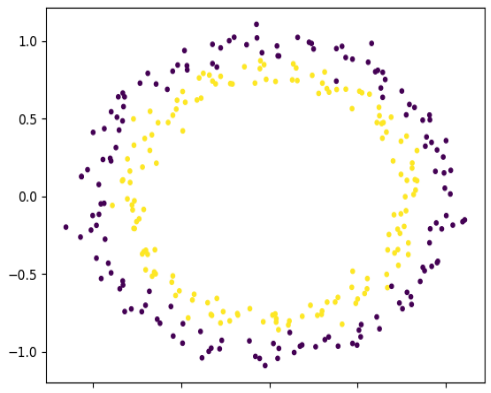
Equation for radial basis kernel:

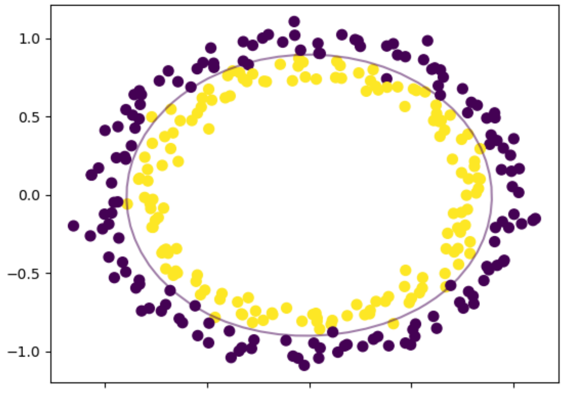
f (X1, X2) = exp(-gamma * ||X1 - X2||^2)

The value of gamma is determined by cross validation.

Radial kernels are used to find support vector classifiers in infinite dimensions. Our classification of new observations is heavily influenced by the closest observations.

Rather than actually transforming the points, kernel functions only calculate their relations as if they were already in the higher dimension.
A kernel trick enables SVM to calculate relationships in infinite dimensions using Radial kernels, reducing the computation required for SVM.





Naive Bayes

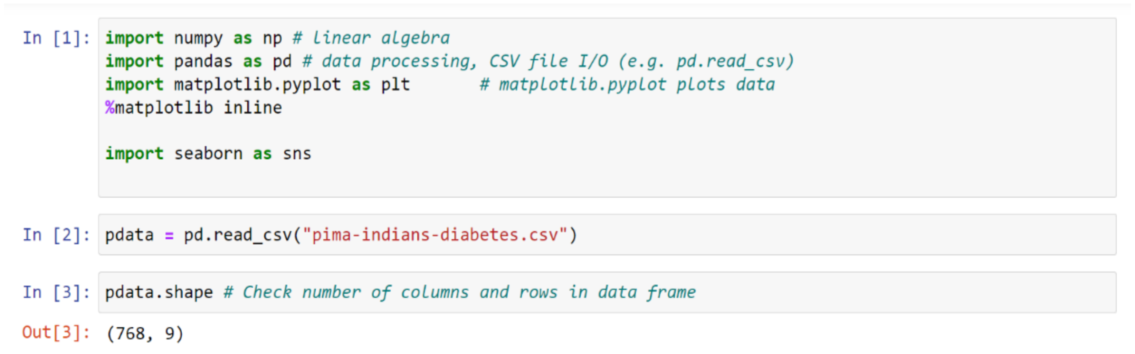
Introduction:

Naive Bayes is one of the most efficient and effective inductive supervised learning algorithms for machine learning and data mining. It's based on applying Bayes' theorem with the "Naive" assumption of conditional independence between every pair of features given the value of the class variable. Its competitive performance in classification is surprising, because the conditional independence assumption on which it is based, is rarely true in real-world applications.

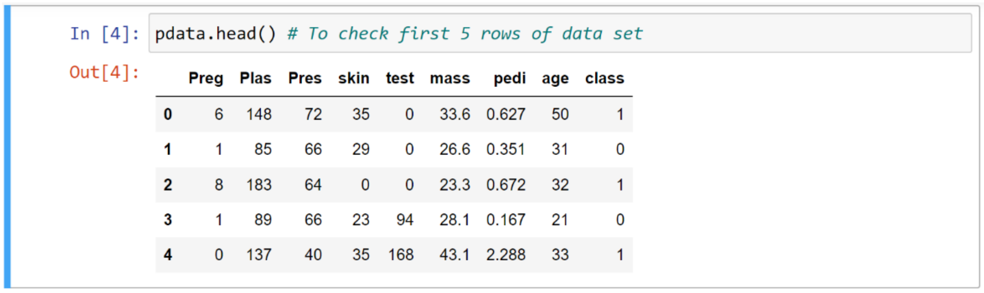
Example: Consider a "Pima Indians Diabetes" data set.

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

The first five rows of the data set.



```
In [1]: import numpy as np # linear algebra
        import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
        import matplotlib.pyplot as plt       # matplotlib.pyplot plots data
        %matplotlib inline

        import seaborn as sns

In [2]: pdata = pd.read_csv("pima-indians-diabetes.csv")

In [3]: pdata.shape # Check number of columns and rows in data frame

Out[3]: (768, 9)
```



```
In [4]: pdata.head() # To check first 5 rows of data set
```

Out[4]:

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
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

Assumption:

The fundamental Naive assumption is that each feature makes an:

- independent, and
- equal

contribution to the outcome.

Bayes' Theorem:

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$\square(\square|\square) = \frac{\square(\square|\square)\square(\square)}{\square(\square)}$$

where A and E are events and P(E) ≠ 0.

- Basically, we are trying to find probability of event A, given that the event E is true. Event E is also termed as evidence.
- P(A) is the priori of A (the prior probability, i.e., Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event E).
- P(A|E) is a posteriori probability of E, i.e., probability of event after evidence is seen.

Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$\square(\square|\square) = \frac{\square(\square|\square)\square(\square)}{\square(\square)} \quad ...(1)$$

where, Y is class variable and X is an independent feature vector (of size n) where:

$Y = y$ and $X = (x_1, \ldots, x_n)$

In our data set feature vector and corresponding class variable are:

X = (Preg, Plas, Pres, skin, test, mass, pedi, age)

Y = class

So basically, P(Y|X) here means, the probability of "having diabetes", class (0 or 1), given that the following constraints

- Preg: Number of times pregnant
- Plas: Plasma glucose concentration 2 hours in an oral glucose tolerance test
- Pres: Diastolic blood pressure (mm Hg)
- skin: Triceps skin fold thickness (mm)
- test: 2-Hour serum insulin (mu U/ml)
- mass: Body mass index (weight in kg/ (height in m)^2)
- pedi: Diabetes pedigree function
- age: Age (years)

Naive Assumption:

Now, it's time to put a Naive assumption to the Bayes' theorem, which is, independence among the features. So now, we split evidence into the independent parts.

Now, if any two events A and B are independent, then

P(AB) = P(A)P(B) i.e., P(A|B) = P(A) or P(B|A) = P(B)

Now, equation (1) can also be written as:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Using the Naive conditional independence assumption that

$$P(x_i | y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i | y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Since, $P(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

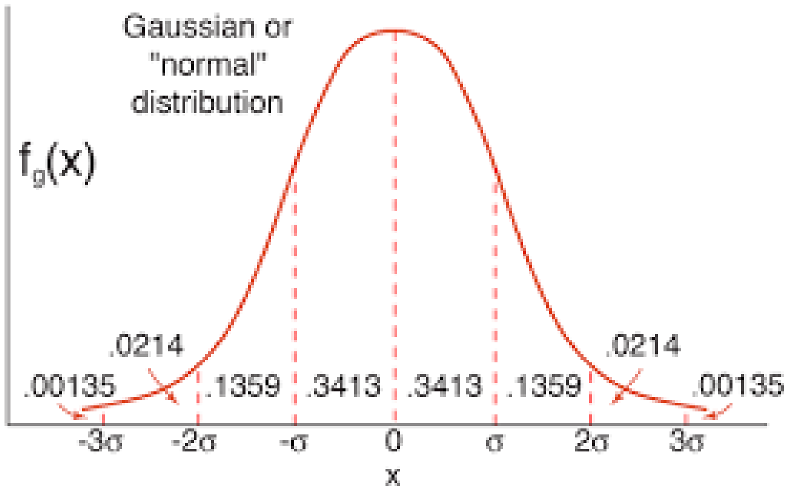
$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \Downarrow \hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different Naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

Gaussian Naive Bayes:

In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a Gaussian distribution. A Gaussian distribution is also called Normal distribution. Plotting the distribution gives a bell-shaped curve which is symmetric about the mean of the feature values.



The likelihood of the features is assumed to be Gaussian; hence, conditional probability is given by:

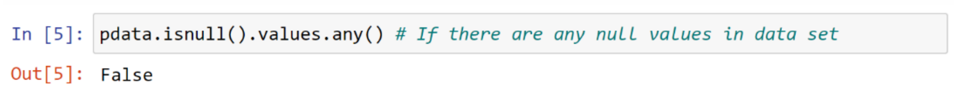
$$\square(\square_{\square} \mid \square) = \frac{1}{\sqrt{2\square\square_{\square}^{2}}} \square\square\square\left(-\frac{(\square_{\square} - \square_{\square})^{2}}{2\square_{\square}^{2}}\right)$$

The parameters $\sigma_{\square}$ and $\mu_{\square}$ are estimated using maximum likelihood.

Implementation of Gaussian Naive Bayes:

Now, let’s predict whether or not a patient has diabetes using Gaussian Naive Bayes.

- First, let’s see if there any null value in data set.



```
In [5]: pdata.isnull().values.any() # If there are any null values in data set
Out[5]: False
```

- Let’s check the correlation between features and class variables.

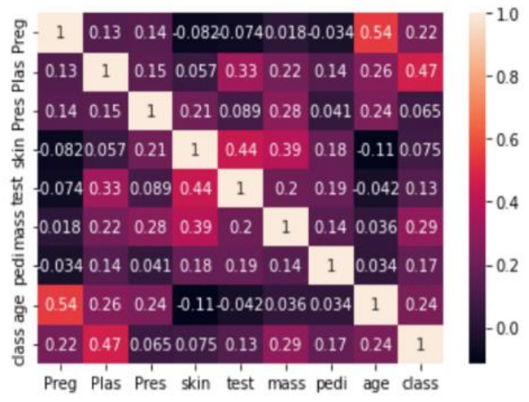
```
In [7]: pdata.corr() # It will show correlation matrix
```

Out[7]:

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
Preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Plas	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
Pres	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
test	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
mass	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
pedi	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
class	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

- Heatmap

```
In [8]: #Correlation in graphical representation
        sns.heatmap(pdata.corr(), annot = True);
```



- Calculating diabetes ratio of True/False from outcome variable:

True outcome is represented as 1 and False outcome as 0.

True cases% = $\frac{□□□□ □□□□}{□□□□ □□□□+□□□□□ □□□□}$

False cases% = $\frac{□□□□□ □□□□}{□□□□ □□□□+□□□□□ □□□□}$

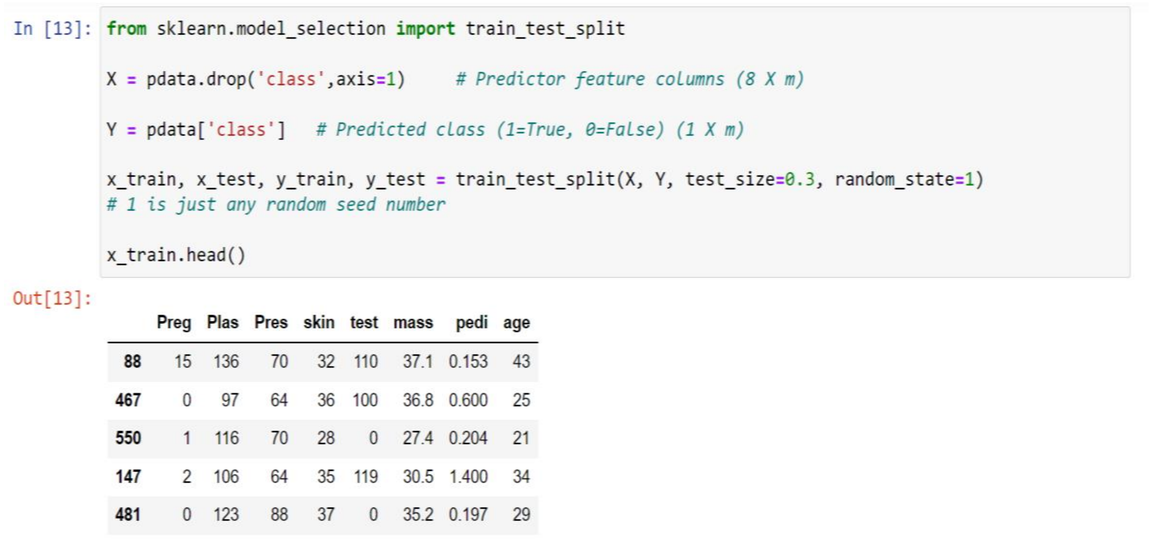
```
In [12]: n_true = len(pdata.loc[pdata['class'] == True])
         n_false = len(pdata.loc[pdata['class'] == False])
         print("Number of true cases: {0} ({1:2.2f}%)".format(n_true, (n_true / (n_true + n_false)) * 100 ))
         print("Number of false cases: {0} ({1:2.2f}%)".format(n_false, (n_false / (n_true + n_false)) * 100))

         Number of true cases: 268 (34.90%)
         Number of false cases: 500 (65.10%)
```

So, we have 34.90% people in current data set who have diabetes and rest of 65.10% doesn't have diabetes.

It’s a good distribution True/False cases of diabetes in data.

- Splitting the data

we will use 70% of data for training and 30% for testing.



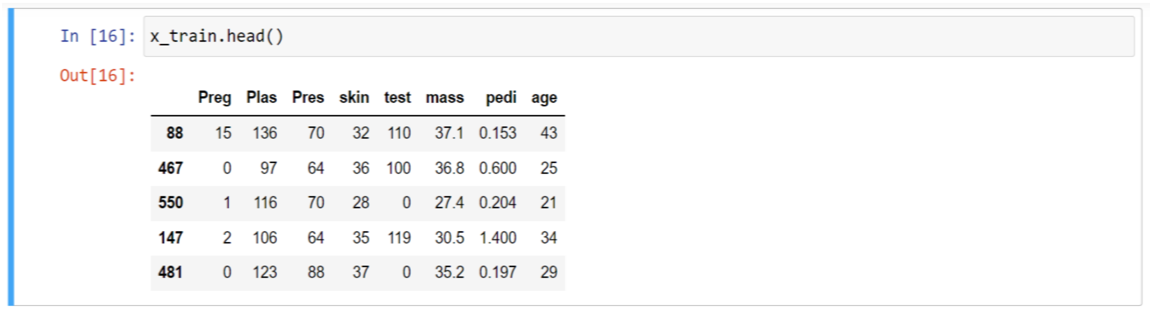
```
In [13]: from sklearn.model_selection import train_test_split

         X = pdata.drop('class',axis=1)     # Predictor feature columns (8 X m)

         Y = pdata['class']   # Predicted class (1=True, 0=False) (1 X m)

         x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=1)
         # 1 is just any random seed number

         x_train.head()
```

Out[13]:

	Preg	Plas	Pres	skin	test	mass	pedi	age
88	15	136	70	32	110	37.1	0.153	43
467	0	97	64	36	100	36.8	0.600	25
550	1	116	70	28	0	27.4	0.204	21
147	2	106	64	35	119	30.5	1.400	34
481	0	123	88	37	0	35.2	0.197	29

- Checking hidden missing values

As we checked missing values earlier but haven't gotten any. But there can be lots of entries with 0 values. We must need to take care of those as well.



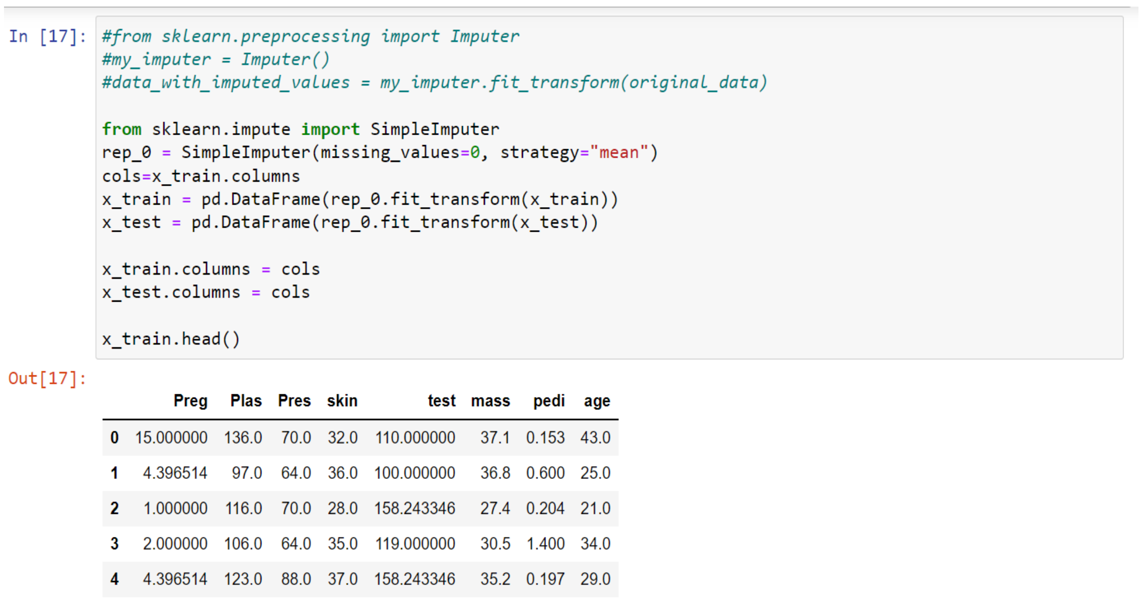
```
In [16]: x_train.head()
```

Out[16]:

	Preg	Plas	Pres	skin	test	mass	pedi	age
88	15	136	70	32	110	37.1	0.153	43
467	0	97	64	36	100	36.8	0.600	25
550	1	116	70	28	0	27.4	0.204	21
147	2	106	64	35	119	30.5	1.400	34
481	0	123	88	37	0	35.2	0.197	29

We can see lots of 0 entries above.

- Replacing 0s with serial mean

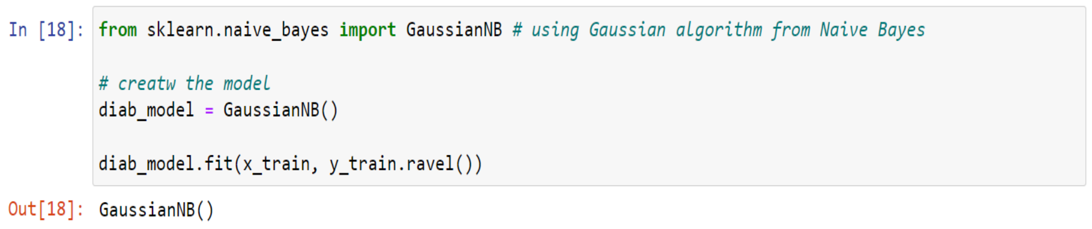


```
In [17]: #from sklearn.preprocessing import Imputer
         #my_imputer = Imputer()
         #data_with_imputed_values = my_imputer.fit_transform(original_data)

         from sklearn.impute import SimpleImputer
         rep_0 = SimpleImputer(missing_values=0, strategy="mean")
         cols=x_train.columns
         x_train = pd.DataFrame(rep_0.fit_transform(x_train))
         x_test = pd.DataFrame(rep_0.fit_transform(x_test))

         x_train.columns = cols
         x_test.columns = cols

         x_train.head()
```

Out[17]:

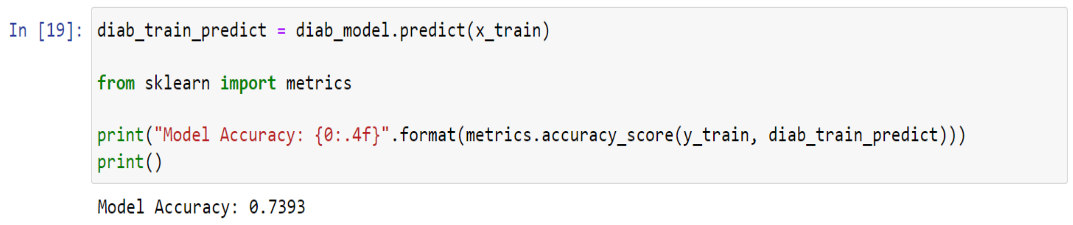
	Preg	Plas	Pres	skin	test	mass	pedi	age
0	15.000000	136.0	70.0	32.0	110.000000	37.1	0.153	43.0
1	4.396514	97.0	64.0	36.0	100.000000	36.8	0.600	25.0
2	1.000000	116.0	70.0	28.0	158.243346	27.4	0.204	21.0
3	2.000000	106.0	64.0	35.0	119.000000	30.5	1.400	34.0
4	4.396514	123.0	88.0	37.0	158.243346	35.2	0.197	29.0

- Training Naïve Bayes Algorithm



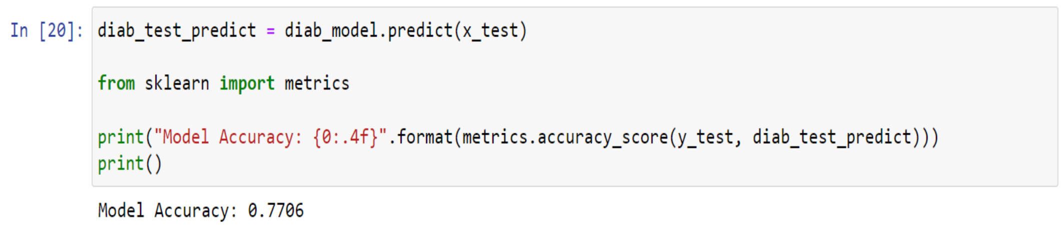
```
In [18]: from sklearn.naive_bayes import GaussianNB # using Gaussian algorithm from Naive Bayes

         # creatw the model
         diab_model = GaussianNB()

         diab_model.fit(x_train, y_train.ravel())

Out[18]: GaussianNB()
```

- Performance of our model with training data



```
In [19]: diab_train_predict = diab_model.predict(x_train)

         from sklearn import metrics

         print("Model Accuracy: {0:.4f}".format(metrics.accuracy_score(y_train, diab_train_predict)))
         print()

         Model Accuracy: 0.7393
```

- Performance of our model with testing data

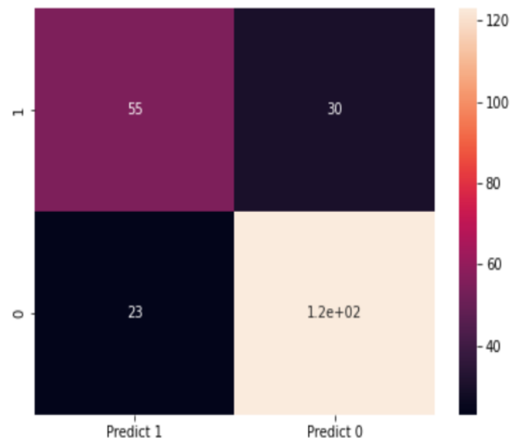


```
In [20]: diab_test_predict = diab_model.predict(x_test)

         from sklearn import metrics

         print("Model Accuracy: {0:.4f}".format(metrics.accuracy_score(y_test, diab_test_predict)))
         print()

         Model Accuracy: 0.7706
```

- Confusion matrix and classification report

```
In [21]: print("Confusion Matrix")
         cm=metrics.confusion_matrix(y_test, diab_test_predict, labels=[1, 0])

         df_cm = pd.DataFrame(cm, index = [i for i in ["1","0"]],
                           columns = [i for i in ["Predict 1","Predict 0"]])
         plt.figure(figsize = (7,5))
         sns.heatmap(df_cm, annot=True);
```

```
Confusion Matrix
```



```
In [22]: print("Classification Report")
         print(metrics.classification_report(y_test, diab_test_predict, labels=[1, 0]))
```

```
Classification Report
              precision    recall  f1-score   support

           1       0.71      0.65      0.67        85
           0       0.80      0.84      0.82       146

    accuracy                           0.77       231
   macro avg       0.75      0.74      0.75       231
weighted avg       0.77      0.77      0.77       231
```

We can see our true positive numbers with value 1 is of precision and it was below 70%.

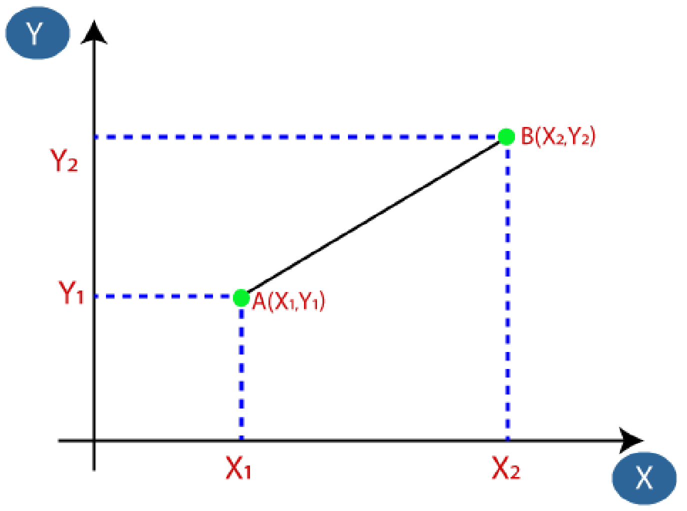
Drawbacks:

- Naive Bayes assumes that all predictors (or features) are independent, rarely happening in real life. This limits the applicability of this algorithm in real-world use cases.
- This algorithm faces the ‘zero-frequency problem’ where it assigns zero probability to a categorical variable whose category in the test data set wasn’t available in the training dataset. It would be best if you used a smoothing technique to overcome this issue.
- Its estimations can be wrong in some cases, so we shouldn’t take its probability outputs very seriously.

KNN Algorithm

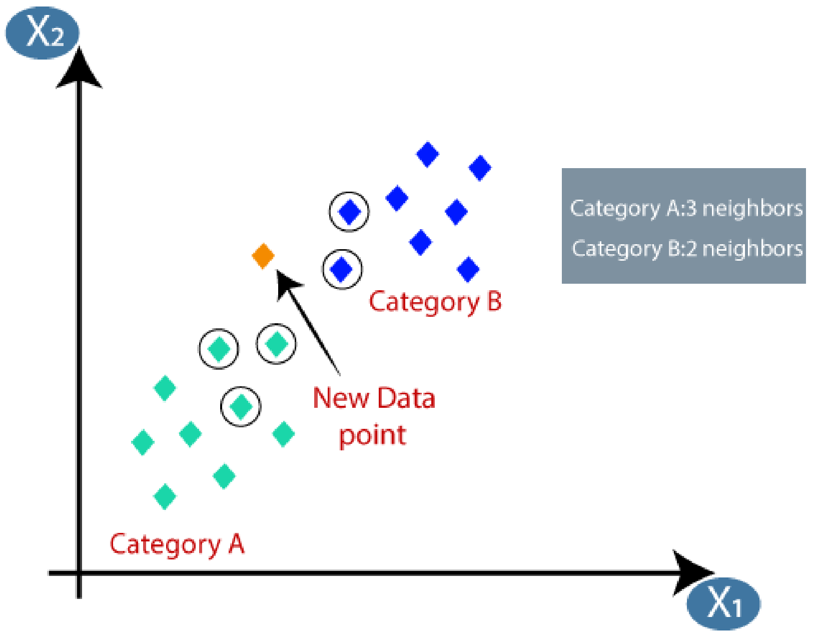
What is KNN Algorithm?

- K-Nearest Neighbors is one of the simplest Machine Learning algorithms based on Supervised Learning.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm can be used for Regression as well as for Classification but primarily it is used for Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- **Example -** Given below is the pictorial representation of how the KNN classifier works.
- Firstly, we will choose the number of neighbors, so we will choose **k=5**.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:

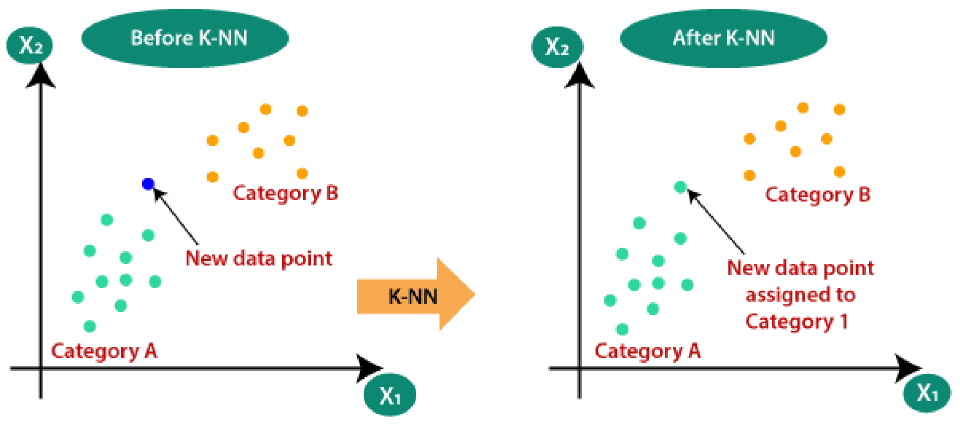


- Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- By calculating the Euclidean distance, we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B.



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.



How does KNN work?

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

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of neighbors is maximum.
- **Step-6:** Our model is ready.

Python implementation of the KNN algorithm

Domain: Medical
Context: Medical research university X is undergoing deep research on patients with certain conditions. University has an internal AI team. Due to confidentiality the patient's details and conditions are masked by the client by providing different datasets to the AI team for developing an AIML model which can predict the condition of the patient depending on the received test results.
Data Description: The data consists of biomechanics features of the patients according to their current conditions. Each patient is represented in the data set by six biomechanics attributes derived from the shape and orientation of the condition to their body part.

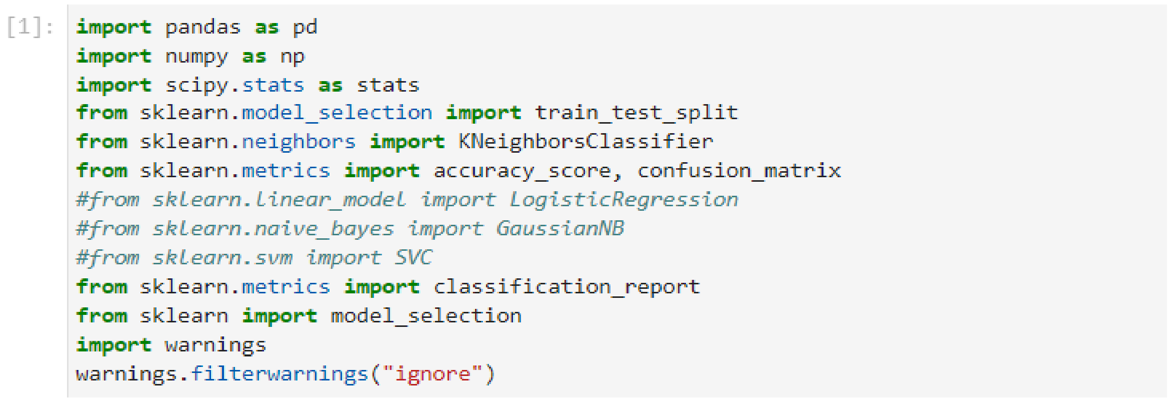
Project Objective: To Demonstrate the ability to fetch, process, and leverage data to generate useful predictions by training Supervised Learning algorithms.

Steps to implement the K-NN algorithm:

- Data Pre-processing step
- Fitting the K-NN algorithm to the Training set
- Predicting the test result
- Test accuracy of the result (Creation of Confusion matrix)

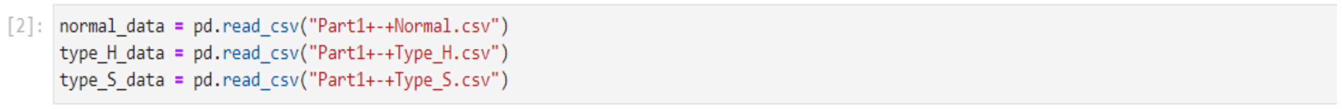
Data Preprocessing and EDA

1. Importing the necessary libraries

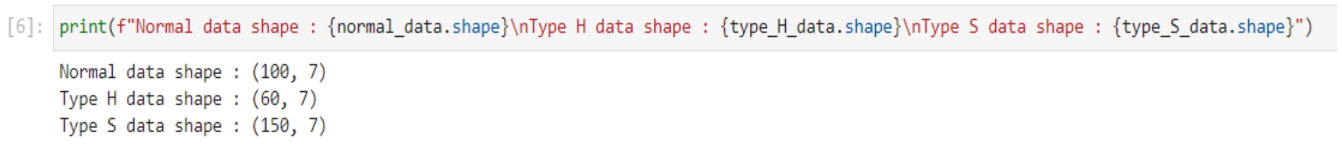


```
[1]: import pandas as pd
     import numpy as np
     import scipy.stats as stats
     from sklearn.model_selection import train_test_split
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, confusion_matrix
     #from sklearn.linear_model import LogisticRegression
     #from sklearn.naive_bayes import GaussianNB
     #from sklearn.svm import SVC
     from sklearn.metrics import classification_report
     from sklearn import model_selection
     import warnings
     warnings.filterwarnings("ignore")
```

2. Loading all three datasets

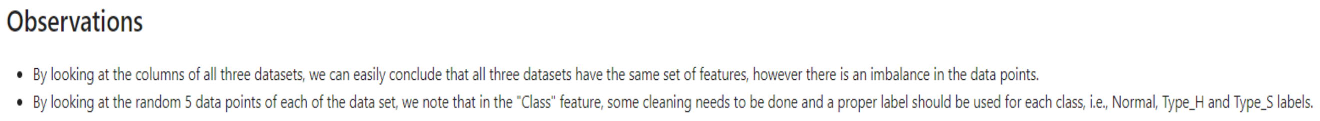


```
[2]: normal_data = pd.read_csv("Part1+-+Normal.csv")
     type_H_data = pd.read_csv("Part1+-+Type_H.csv")
     type_S_data = pd.read_csv("Part1+-+Type_S.csv")
```



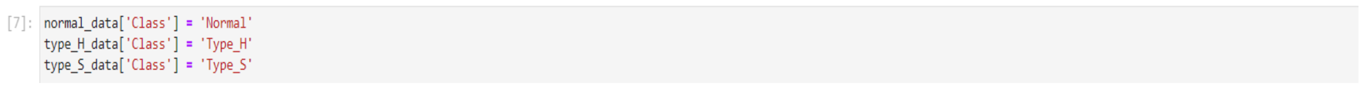
```
[6]: print(f"Normal data shape : {normal_data.shape}\nType H data shape : {type_H_data.shape}\nType S data shape : {type_S_data.shape}")

     Normal data shape : (100, 7)
     Type H data shape : (60, 7)
     Type S data shape : (150, 7)
```

3. After observing the datasets, modifications are needed in the "Class" feature in each dataset to ensure proper labeling of the three classes.



Observations

- By looking at the columns of all three datasets, we can easily conclude that all three datasets have the same set of features, however there is an imbalance in the data points.
- By looking at the random 5 data points of each of the data set, we note that in the "Class" feature, some cleaning needs to be done and a proper label should be used for each class, i.e., Normal, Type_H and Type_S labels.



```
[7]: normal_data['Class'] = 'Normal'
     type_H_data['Class'] = 'Type_H'
     type_S_data['Class'] = 'Type_S'
```

```
[8]: df = normal_data.append([type_H_data, type_S_data])
```

```
[31]: df.sample(5)
```

[31]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree	Class
138	74.854480	13.909084	62.693259	60.945396	115.208701	33.172255	2
9	36.686353	5.010884	41.948751	31.675469	118.000000	0.664437	1
46	65.013773	9.838262	57.735837	55.175511	94.738525	49.696955	2
45	50.912440	23.015169	47.000000	27.897271	117.422259	-2.526702	1
21	53.911054	12.939318	39.000000	40.971736	118.193035	5.074353	0

```
[10]: df.shape
```

```
[10]: (310, 7)
```

```
[11]: df.isna().sum()
```

```
[11]: P_incidence    0
      P_tilt         0
      L_angle        0
      S_slope        0
      P_radius       0
      S_Degree       0
      Class          0
      dtype: int64
```

```
[12]: df['Class']=df['Class'].astype('category')
```

There are no missing values in our dataset

```
[13]: df.describe()
```

[13]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree
count	310.000000	310.000000	310.000000	310.000000	310.000000	310.000000
mean	60.496653	17.542822	51.930930	42.953831	117.920655	26.296694
std	17.236520	10.008330	18.554064	13.423102	13.317377	37.559027
min	26.147921	-6.554948	14.000000	13.366931	70.082575	-11.058179
25%	46.430294	10.667069	37.000000	33.347122	110.709196	1.603727
50%	58.691038	16.357689	49.562398	42.404912	118.268178	11.767934
75%	72.877696	22.120395	63.000000	52.695888	125.467674	41.287352
max	129.834041	49.431864	125.742385	121.429566	163.071041	418.543082

Observation

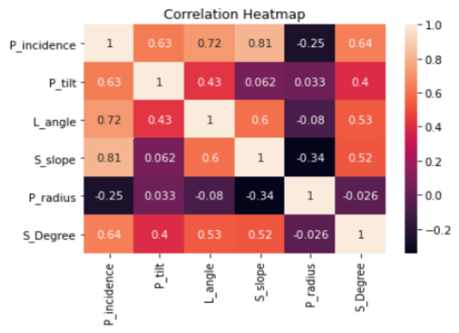
- mean and median for all features except S_Degree are nearly equal which means normal distribution might be present.

```
[14]: df.corr()
```

[14]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree
P_incidence	1.000000	0.629199	0.717282	0.814960	-0.247467	0.638743
P_tilt	0.629199	1.000000	0.432764	0.062345	0.032668	0.397862
L_angle	0.717282	0.432764	1.000000	0.598387	-0.080344	0.533667
S_slope	0.814960	0.062345	0.598387	1.000000	-0.342128	0.523557
P_radius	-0.247467	0.032668	-0.080344	-0.342128	1.000000	-0.026065
S_Degree	0.638743	0.397862	0.533667	0.523557	-0.026065	1.000000

```
[16]: sns.heatmap(df.corr(), annot = True)
      plt.title('Correlation Heatmap')
      plt.figure(figsize=(20,20))
      plt.show()
```

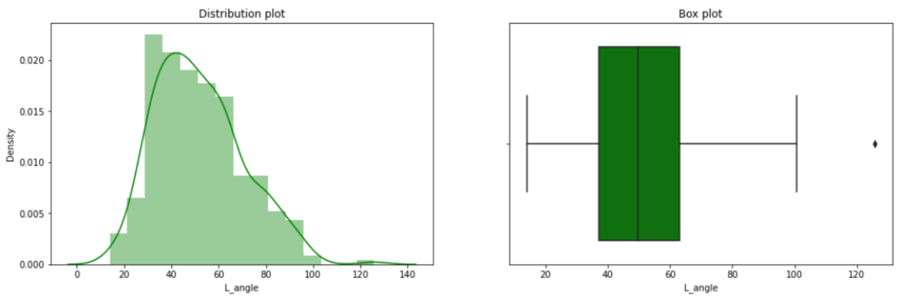


Observation

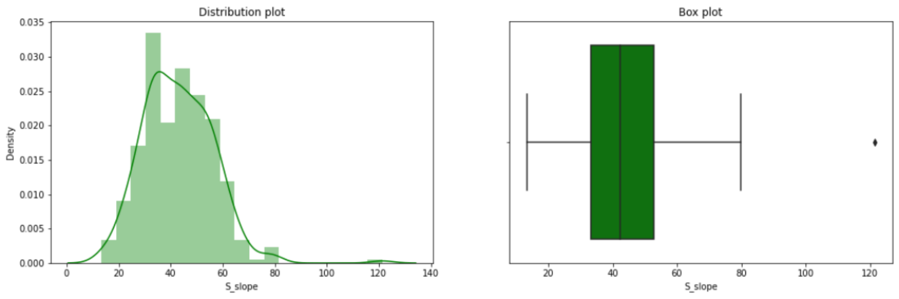
There is a significant correlation between P_incidence and S_slope, and some moderate correlation between P_incidence and L_angle.

Univariate, Bivariate, and Multivariate Analysis

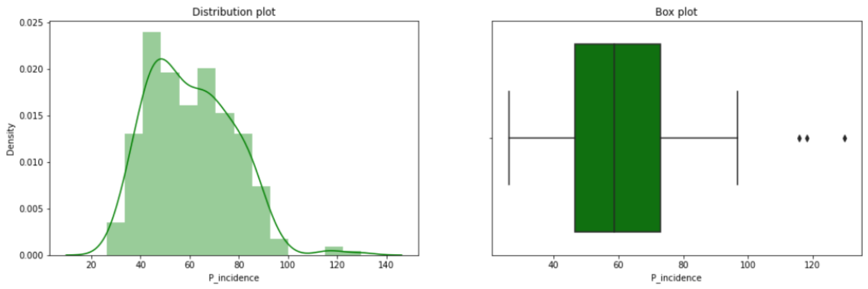
```
[19]: for col in ["P_incidence","P_tilt","L_angle","S_slope","P_radius","S_Degree"]:
          f, axes = plt.subplots(1, 2, figsize=(17,5))
          sns.boxplot(x = col, data=df,  orient='h' , ax=axes[1],color='Green')
          sns.distplot(df[col],  ax=axes[0],color='Green')
          axes[0].set_title('Distribution plot')
          axes[1].set_title('Box plot')
          plt.show()

          #checking count of outliers.
          q25,q75=np.percentile(df[col],25),np.percentile(df[col],75)
          IQR=q75-q25
          lower,upper=q25-(IQR*1.5),q75+(IQR*1.5)
          Outliers=[i for i in df[col] if i < lower or i > upper]
          print(f"Total Number of outliers in {col}: {len(Outliers)}")
```



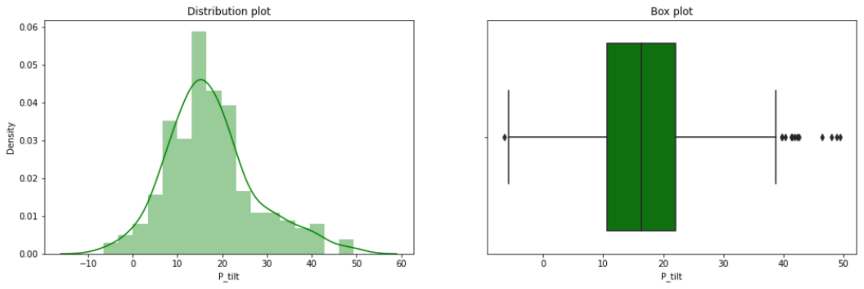
```
Total Number of outliers in L_angle: 1
```



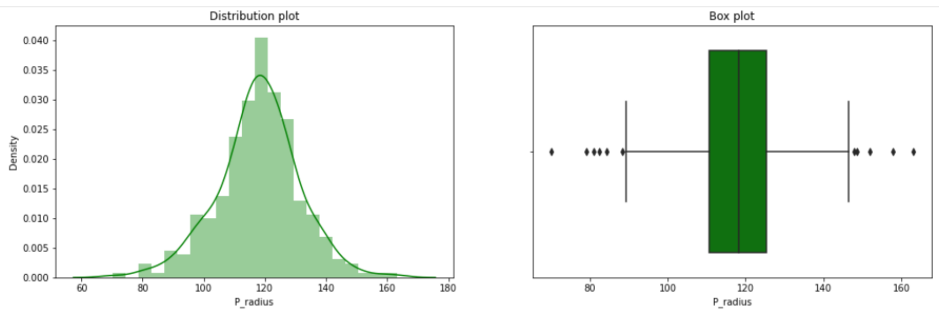
```
Total Number of outliers in S_slope: 1
```



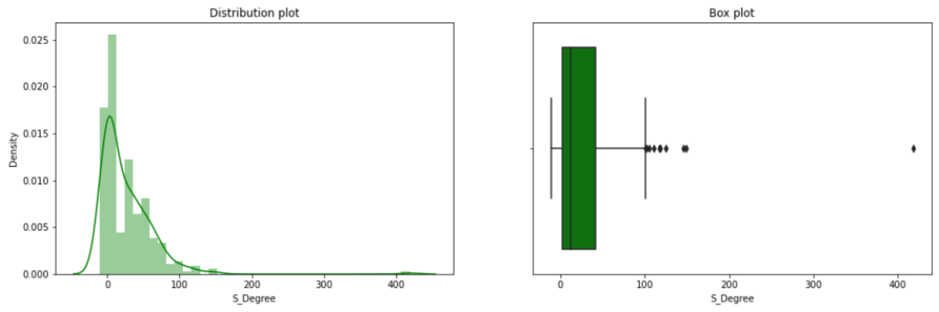
```
Total Number of outliers in P_incidence: 3
```



```
Total Number of outliers in P_tilt: 13
```



```
Total Number of outliers in P_radius: 11
```



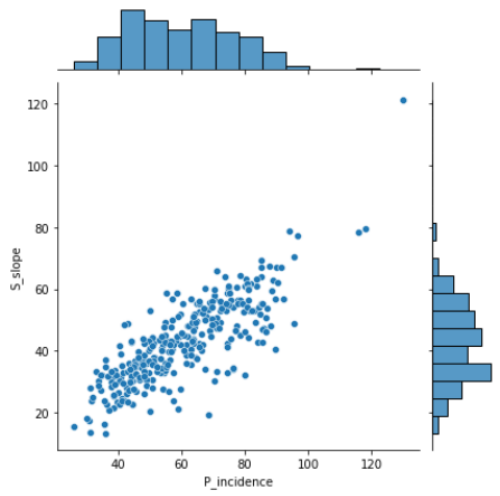
```
Total Number of outliers in S_Degree: 10
```

Observation

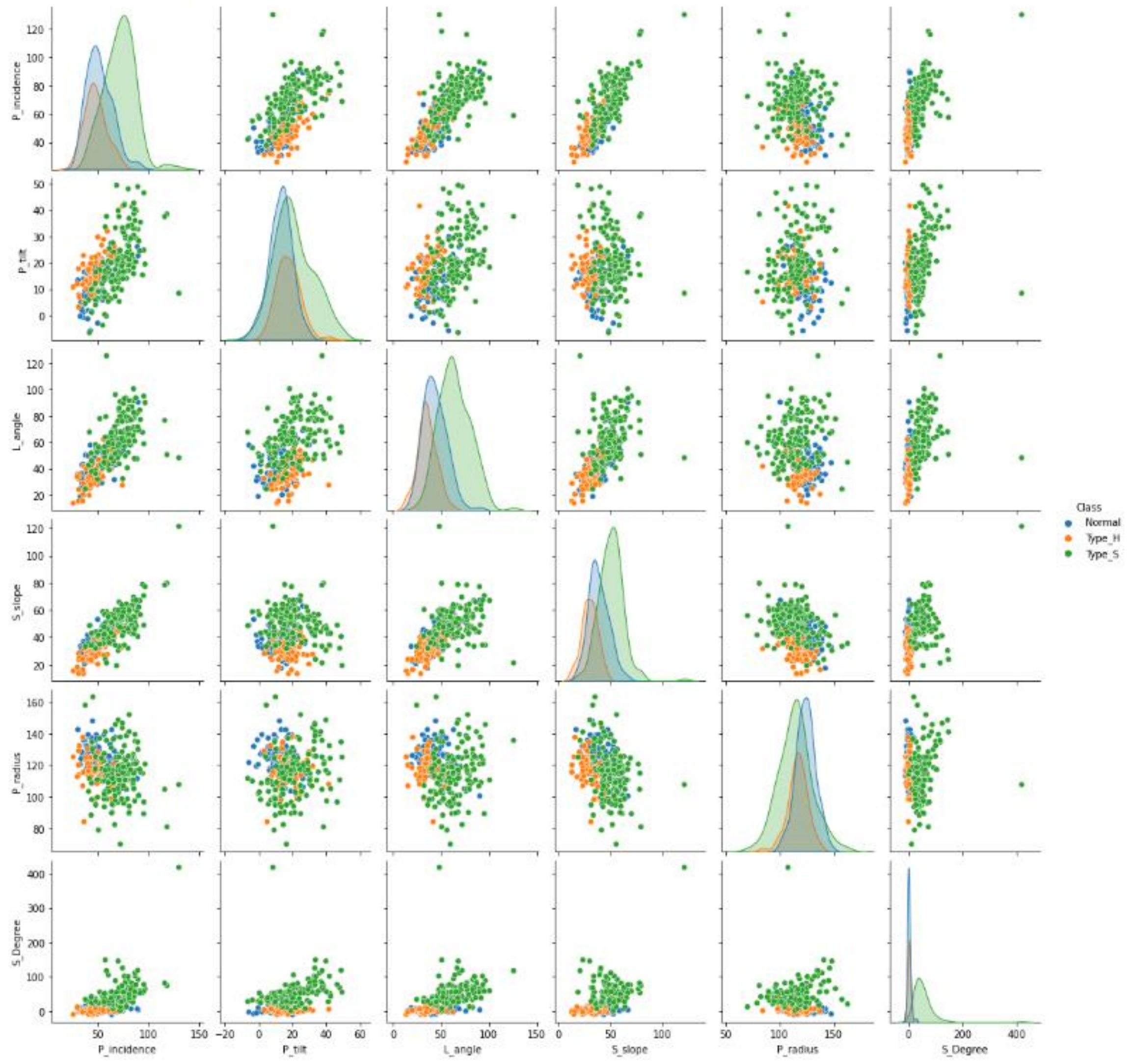
- The above plot displays a scatterplot with two histograms at the margins of the graph. If you observe the scatterplot, there seems to be a positive relationship between the columns 'P_incidence' and 'S_Slope', because if the values of one variable increase so do the other.
- The strength of the relationship appears to be strong. The marginal histograms are both right-skewed as most values are concentrated around the left side of the distribution while the right side of the distribution is longer.
- Outliers are the data points that lie far away from the rest of the data values, in the graph we can see a few outliers in the scatterplot as well as the histograms.

```
[18]: sns.jointplot(x="P_incidence", y = "S_slope", data=df)
```

```
[18]: <seaborn.axisgrid.JointGrid at 0x1b9d2027130>
```



```
<seaborn.axisgrid.PairGrid at 0x1b9ube3ae80>
```

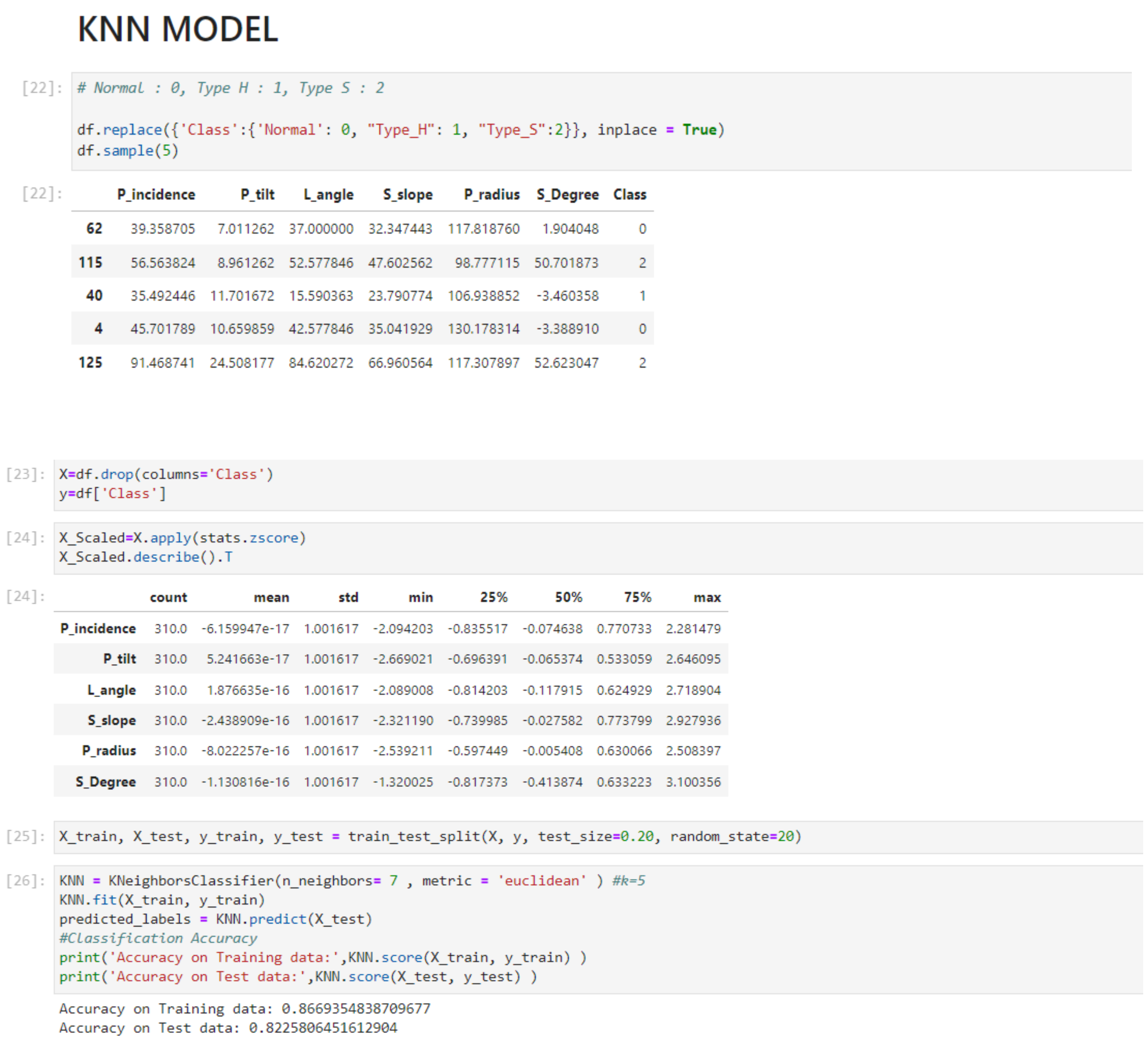


Removing the outliers

```
[21]: for c in col:
          #getting upper lower quartile values
          q25,q75=np.percentile(df[c],25),np.percentile(df[c],75)
          IQR=q75-q25
          Threshold=IQR*1.5
          lower,upper=q25-Threshold,q75+Threshold
          #taking mean of a column without considering outliers
          df_include = df.loc[(df[c] >= lower) & (df[c] <= upper)]
          mean=int(df_include[c].mean())
          #imputing outliers with mean
          df[c]=np.where(df[c]>upper,mean,df[c])
          df[c]=np.where(df[c]<lower,mean,df[c])
          Outliers=[i for i in df[c] if i < lower or i > upper]
      print("Success")
```

```
Success
```

KNN BASE MODEL



KNN MODEL

```
[22]: # Normal : 0, Type H : 1, Type S : 2

      df.replace({'Class':{'Normal': 0, "Type_H": 1, "Type_S":2}}, inplace = True)
      df.sample(5)
```

[22]:

	P_incidence	P_tilt	L_angle	S_slope	P_radius	S_Degree	Class
62	39.358705	7.011262	37.000000	32.347443	117.818760	1.904048	0
115	56.563824	8.961262	52.577846	47.602562	98.777115	50.701873	2
40	35.492446	11.701672	15.590363	23.790774	106.938852	-3.460358	1
4	45.701789	10.659859	42.577846	35.041929	130.178314	-3.388910	0
125	91.468741	24.508177	84.620272	66.960564	117.307897	52.623047	2

```
[23]: X=df.drop(columns='Class')
      y=df['Class']
```

```
[24]: X_Scaled=X.apply(stats.zscore)
      X_Scaled.describe().T
```

[24]:

	count	mean	std	min	25%	50%	75%	max
P_incidence	310.0	-6.159947e-17	1.001617	-2.094203	-0.835517	-0.074638	0.770733	2.281479
P_tilt	310.0	5.241663e-17	1.001617	-2.669021	-0.696391	-0.065374	0.533059	2.646095
L_angle	310.0	1.876635e-16	1.001617	-2.089008	-0.814203	-0.117915	0.624929	2.718904
S_slope	310.0	-2.438909e-16	1.001617	-2.321190	-0.739985	-0.027582	0.773799	2.927936
P_radius	310.0	-8.022257e-16	1.001617	-2.539211	-0.597449	-0.005408	0.630066	2.508397
S_Degree	310.0	-1.130816e-16	1.001617	-1.320025	-0.817373	-0.413874	0.633223	3.100356

```
[25]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=20)
```

```
[26]: KNN = KNeighborsClassifier(n_neighbors= 7 , metric = 'euclidean' ) #k=5
      KNN.fit(X_train, y_train)
      predicted_labels = KNN.predict(X_test)
      #Classification Accuracy
      print('Accuracy on Training data:',KNN.score(X_train, y_train) )
      print('Accuracy on Test data:',KNN.score(X_test, y_test) )

      Accuracy on Training data: 0.8669354838709677
      Accuracy on Test data: 0.8225806451612904
```

Classification matrix

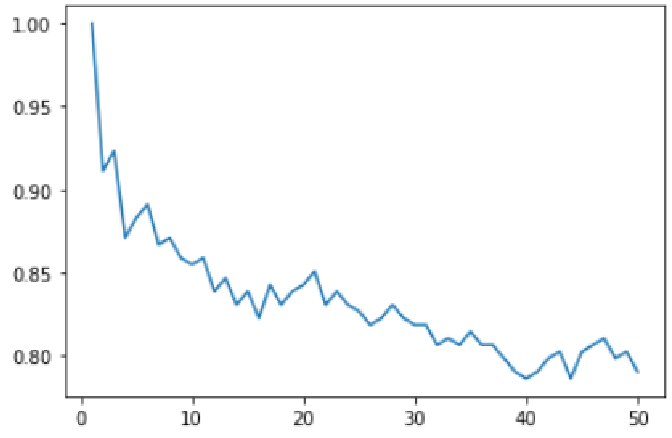
```
[27]: print("classification  Matrix:\n\n",classification_report(y_test,predicted_labels))
```

```
classification  Matrix:

              precision    recall  f1-score   support

           0       0.80      0.73      0.76        22
           1       0.50      0.50      0.50        10
           2       0.94      1.00      0.97        30

    accuracy                           0.82        62
   macro avg       0.75      0.74      0.74        62
weighted avg       0.82      0.82      0.82        62
```

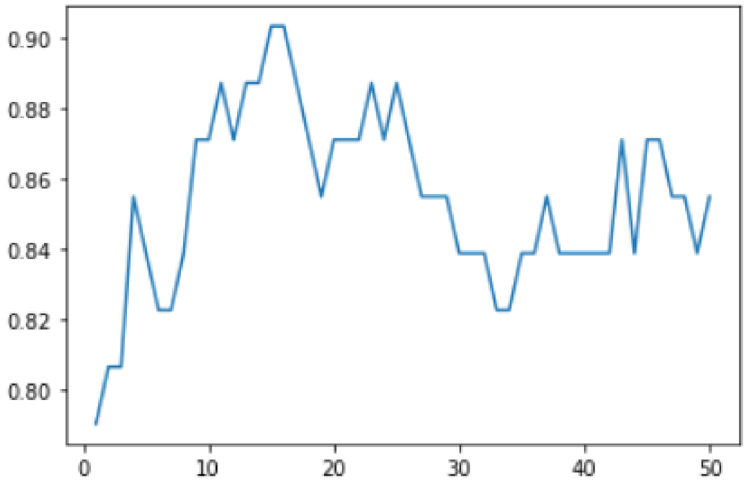
We can clearly tell the difference in precision for different classes. Our KNN Classifier with k = 7 classifies the Normal class with 80% accuracy and 94% accuracy for Type_S, however precision for Type H is very low. Hence, we need to use a better value of k.

Finding the best Value of K

```
[28]: # TRAINING DATA
      train_score=[]
      test_score=[]
      for k in range(1,51):
          KNN = KNeighborsClassifier(n_neighbors= k , metric = 'euclidean' )
          KNN.fit(X_train, y_train)
          train_score.append(KNN.score(X_train, y_train))
          test_score.append(KNN.score(X_test, y_test))
      plt.plot(range(1,51),train_score)
      plt.show()
```

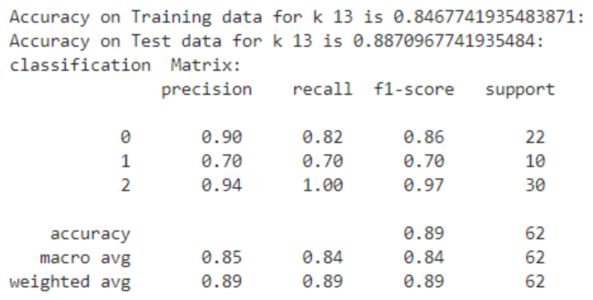


```
[29]: #TEST DATA
      plt.plot(range(1,51),test_score)
      plt.show()
```

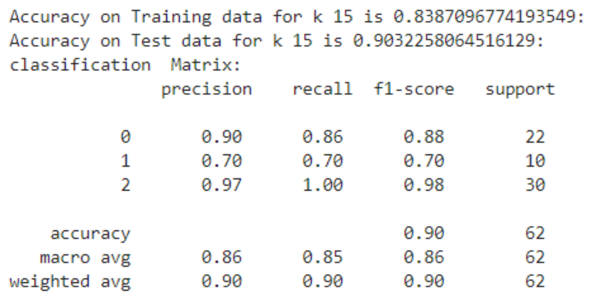


Accuracy decreases for an increase in k for training data, however, accuracy is maximum for 13 < k < 20. Let's try checking the accuracy when k = 13, 15,17,1**9**

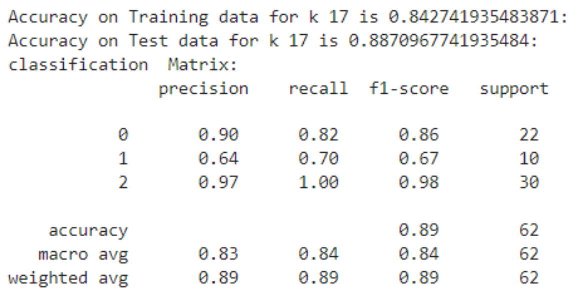
```
[30]: k = [13,15, 17, 19]
      for i in k:
          KNN = KNeighborsClassifier(n_neighbors=i, metric = 'euclidean')
          KNN.fit(X_train, y_train)
          predicted_labels = KNN.predict(X_test)
          print('Accuracy on Training data for k {} is {}:'.format(i,KNN.score(X_train, y_train)))
          print('Accuracy on Test data for k {} is {}:'.format(i,KNN.score(X_test, y_test)))
          print("classification  Matrix:\n",classification_report(y_test,predicted_labels))
```



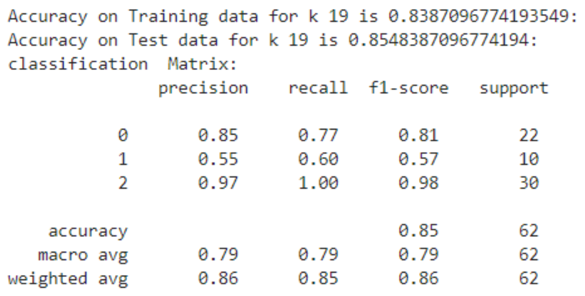
```
Accuracy on Training data for k 13 is 0.8467741935483871:
Accuracy on Test data for k 13 is 0.8870967741935484:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.90      0.82      0.86        22
           1       0.70      0.70      0.70        10
           2       0.94      1.00      0.97        30

    accuracy                           0.89        62
   macro avg       0.85      0.84      0.84        62
weighted avg       0.89      0.89      0.89        62
```



```
Accuracy on Training data for k 15 is 0.8387096774193549:
Accuracy on Test data for k 15 is 0.9032258064516129:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.90      0.86      0.88        22
           1       0.70      0.70      0.70        10
           2       0.97      1.00      0.98        30

    accuracy                           0.90        62
   macro avg       0.86      0.85      0.86        62
weighted avg       0.90      0.90      0.90        62
```



```
Accuracy on Training data for k 17 is 0.842741935483871:
Accuracy on Test data for k 17 is 0.8870967741935484:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.90      0.82      0.86        22
           1       0.64      0.70      0.67        10
           2       0.97      1.00      0.98        30

    accuracy                           0.89        62
   macro avg       0.83      0.84      0.84        62
weighted avg       0.89      0.89      0.89        62
```



```
Accuracy on Training data for k 19 is 0.8387096774193549:
Accuracy on Test data for k 19 is 0.8548387096774194:
classification  Matrix:
               precision    recall  f1-score   support

           0       0.85      0.77      0.81        22
           1       0.55      0.60      0.57        10
           2       0.97      1.00      0.98        30

    accuracy                           0.85        62
   macro avg       0.79      0.79      0.79        62
weighted avg       0.86      0.85      0.86        62
```

Observation: K = 15 produces the best results for our given dataset.

REFERENCES

- https://www.analyticsvidhya.com/blog/2021/05/knn-the-distance-based-machine-learning-algorithm/
- https://machinelearningmastery.com/radius-neighbors-classifier-algorithm-with-python/#:~:text=Radius%20Neighbors%20Classifier%20is%20a,than%20the%20k%2Dclosest%20neighbors
- https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
- https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html
- https://en.wikipedia.org/wiki/Radial_basis_function_kernel
- https://www.kaggle.com/datasets/mathchi/diabetes-data-set?resource=download
- https://www.aaai.org/Papers/FLAIRS/2004/Flairs04-097.pdf
- https://en.wikipedia.org/wiki/Naive_Bayes_classifier
- http://gerardnico.com/wiki/data_mining/Naive_bayes
- http://scikit-learn.org/stable/modules/Naive_bayes.html
- https://www.ibm.com/cloud/learn/overfitting
- https://www.javatpoint.com/performance-metrics-in-machine-learning
- https://medium.com/@MohammedS/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b