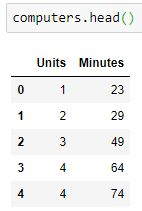
**Machine Learning -2**

**Data:**

Consider a computer service center where the technicians repair computers by replacing damaged parts.



The data (shown) represents the number of units (Units) being replaced in a computer and the corresponding time taken to repair the computer in minutes (Minutes).

# Regression

To predict the value of time taken to repair a computer based on the number of units being replaced, we could build a regression model using regression analysis.

Regression analysis is a statistical process for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables. When the focus is on the relationship between a dependent variable and one or more independent variables (or '**predictors**'). (Source: Wikipedia-Regression Analysis)

Regression analysis focuses on building a model that can be used to predict the value of the dependent variable based on the predictor variables.

The regression model is represented using a mathematical model of form y= f(X), where y is the dependent variable and X is the set of predictor variables (x1, x2, …, xn).

In general, f(X) can take a linear form or a non-linear form.

Few of the linear models are as shown below.

* f(X) = ß0 +ß1x1 +ß2x2+ ... +ßnxn+ ∈
* f(X) = ß0 +ß1x12+ß2x1+ ∈
* f(X) = ß0 +ß1x1p1 +ß2x2p2+ ... +ßnxnpn+ ∈

Where,   ß0, ß1, ß2, ..., ßnare said to be the regression coefficients and ∈ accounts for the error in prediction. The regression coefficients and the error in prediction are real numbers.

If a linear regression model involves only one predictor variable and one dependent variable, it is called a **Simple Linear Regression model**.

E.g. f(X) = ß0 +ß1x1+ ∈

If a linear regression model involves multiple predictor variables and one dependent variable, it is called a **Multiple Linear Regression model**.

E.g. f(X) = ß0 +ß1x1 +ß2x2+ ... +ßnxn+ ∈

**Non-linear models**

A regression model takes the linear form if the derivative of the model does not depend upon any of its regression coefficients. And when the model’s derivative does depend upon its coefficients, then it is termed a non-linear model.  An example of a non-linear model is shown below.

E.g. f(X)= ß0+ß12X

Since the scatter plot between time taken and units replaced suggested a linear association, let us build a linear regression model to quantify the nature of this relationship.

# Simple Linear Regression

In our example, we shall use the number of units being replaced to predict the time taken to repair a computer. Therefore, the time taken to repair a computer can be considered as the dependent variable while the number of units being replaced can be considered as the predictor variable. This is a case of simple linear regression since we have one predictor and one dependent variable.

Therefore the regression model to predict the value of time taken to repair a computer could be expressed as,

Time taken to repair a computer = ß0+ (ß1\* Units being replaced) + ε

# Creating Regression Models

We have understood that we can create a regression model using simple linear regression, to predict the time taken to repair a computer based on the units to be replaced. Let us speculate a few linear regression models that may help us predict the value of time taken to repair a computer.

**Model 0**: Time taken to repair a computer = 97.21 (i.e. mean)

**Model 1**: Time taken to repair a computer = 10 + (12 \* No. of Units being replaced)

**Model 2**: Time taken to repair a computer = 6 + (18 \* No. of Units being replaced)

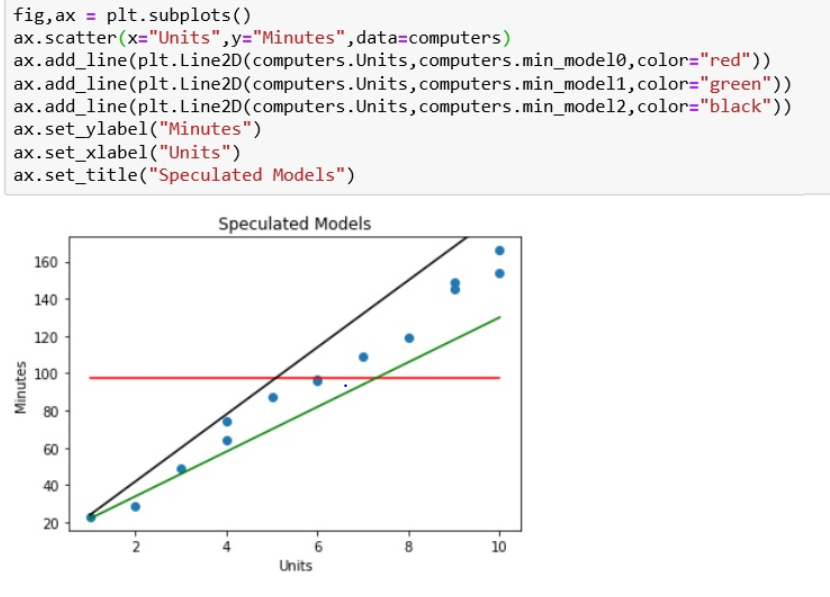
In Model 0, it is assumed that the time taken to repair a computer is constant. Hence, the expected time taken to repair a computer is predicted using the mean.

In Model 1, ß0= 10,can be assumed as the startup time required to understand the repair work to be done on a given computer. The other coefficient, ß1 = 12, is assumed to represent the time required to replace a single unit of computer. The Model 1, thus takes into account the initial time being spent, to understand the repair work to be done along with the time that every single component shall consume for replacement.

Similarly, in Model 2, ß0and ß1, are chosen to be 6 and 18 respectively.

# Visualizing the Speculated Regression Models

Plots of Model 0 (red line), Model 1 (green line) and Model 2 (black line) along with the given data are as shown below.

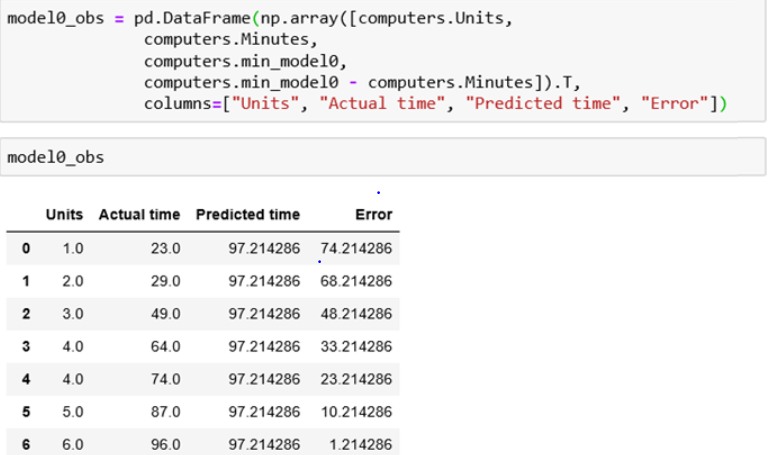


We can observe from the above plot that Model 1 and Model 2 seem to predict the value of time taken to repair a computer better than Model 0.

Let us now use the speculated models (Model 0, Model 1 and Model 2) to predict the value of time taken to repair a computer.

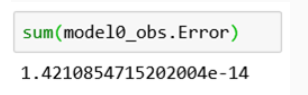
# Analyzing the Speculated Models

The following code snippet shows the units replaced, the observed time taken, expected time taken (based on the model) and the difference between predicted and observed values for Model 0.



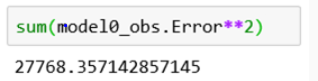
Here, we can observe that the predicted values are significantly away from the observed values. This difference between predicted value and observed value is termed as an error in prediction.

The sum of individual errors is as shown below.

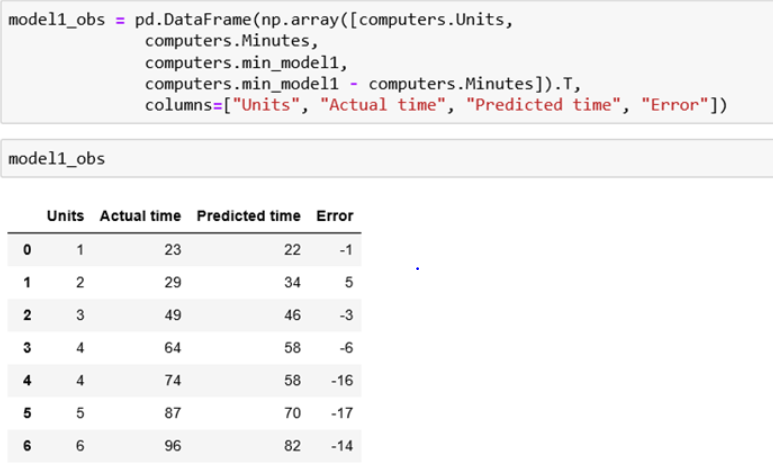


Recall that the total deviation about the mean is always close to 0. Here we can observe that the **sum of individual errors** is approximately 0.

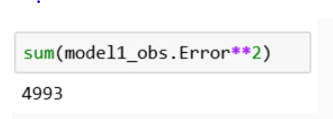
Therefore, to identify the total error of the model, we square the individual errors and sum them as shown below. We square these errors in order to avoid cancellation. Hence, the total sum of squared errors for Model 0 is 27768.36 (approximately), as shown below.



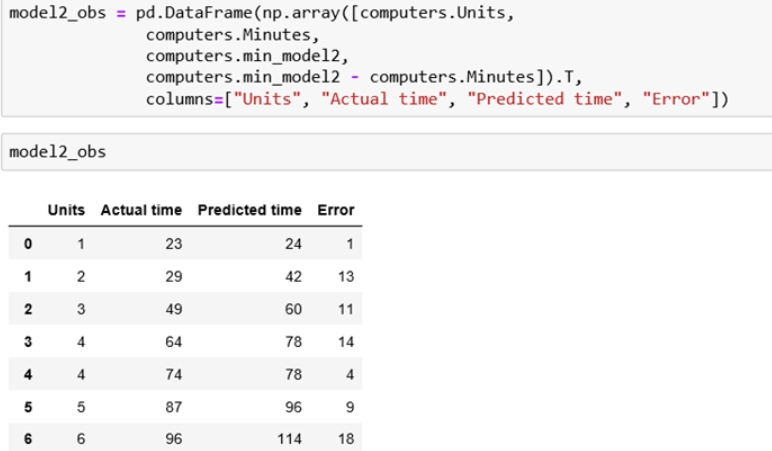
The following code snippet shows the units replaced, the observed time taken, expected time taken (based on the model) and the difference between predicted and observed values for Model 1.

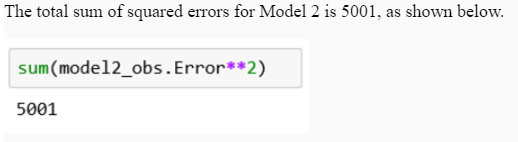


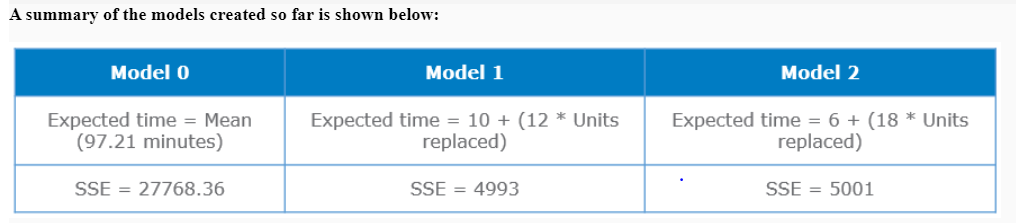
Here, we can observe that the predicted value is quite close to the observed value. This further strengthens the argument that Model 1 is a better predictor in comparison to Model 0. The total sum of squared errors for Model 1 is 4993, as shown below.



Similarly, the following code snippet shows the units replaced, the observed time taken, expected time taken (based on the model) and the difference between predicted and observed values for Model 2.



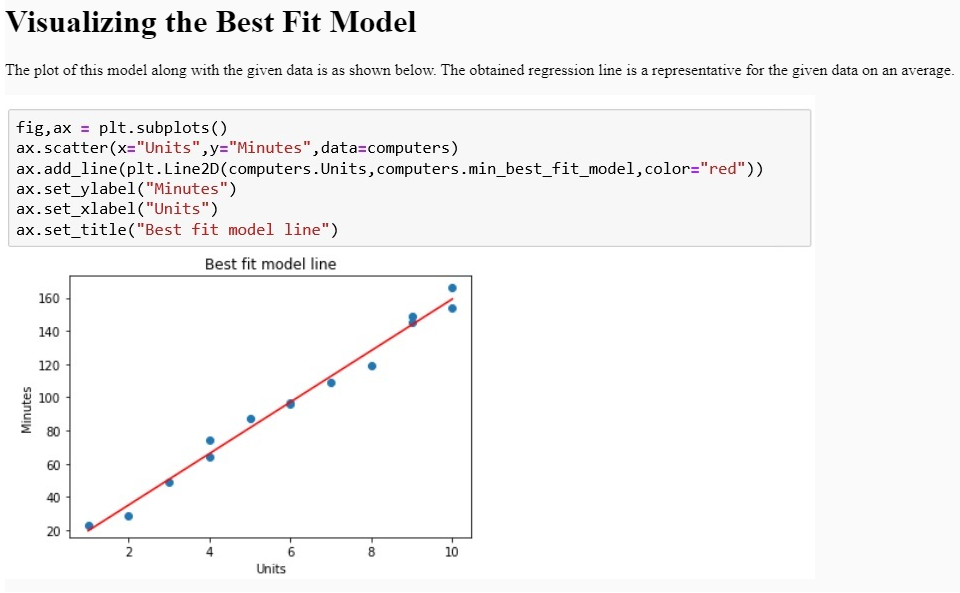


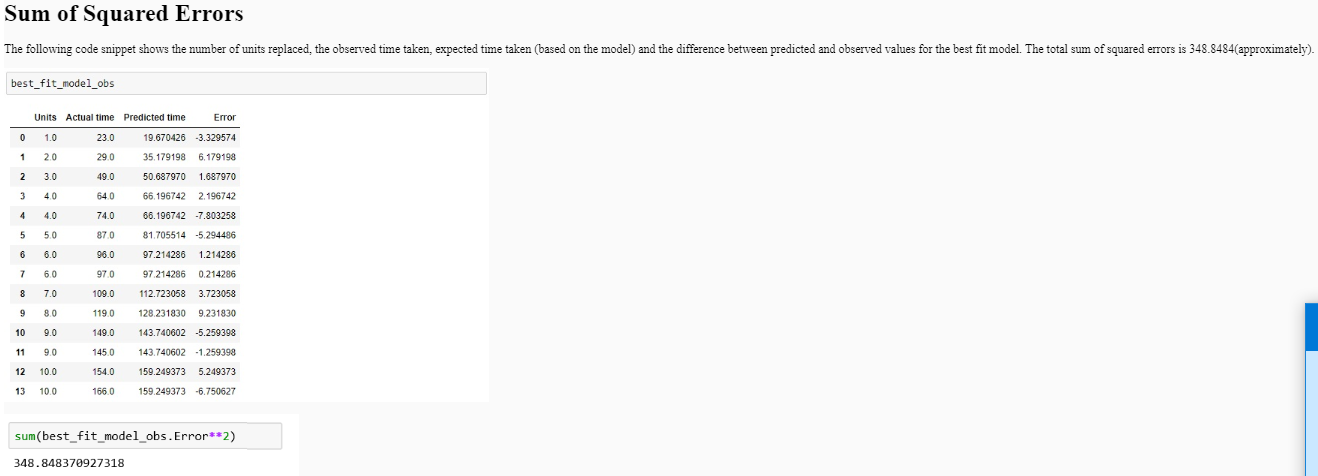


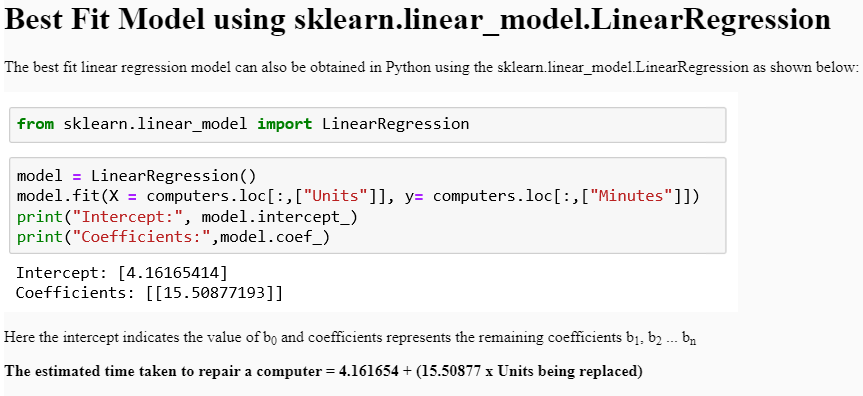
We can observe that the total sum of squared errors for Model 1 and Model 2 are close to each other and also significantly lower than the sum of squared errors of Model 0. With this in consideration, we can say that for the given sample dataset, Model 1 and Model 2 are better than Model 0 at predicting the expected value of time taken to repair a computer, based on the number of parts being replaced.

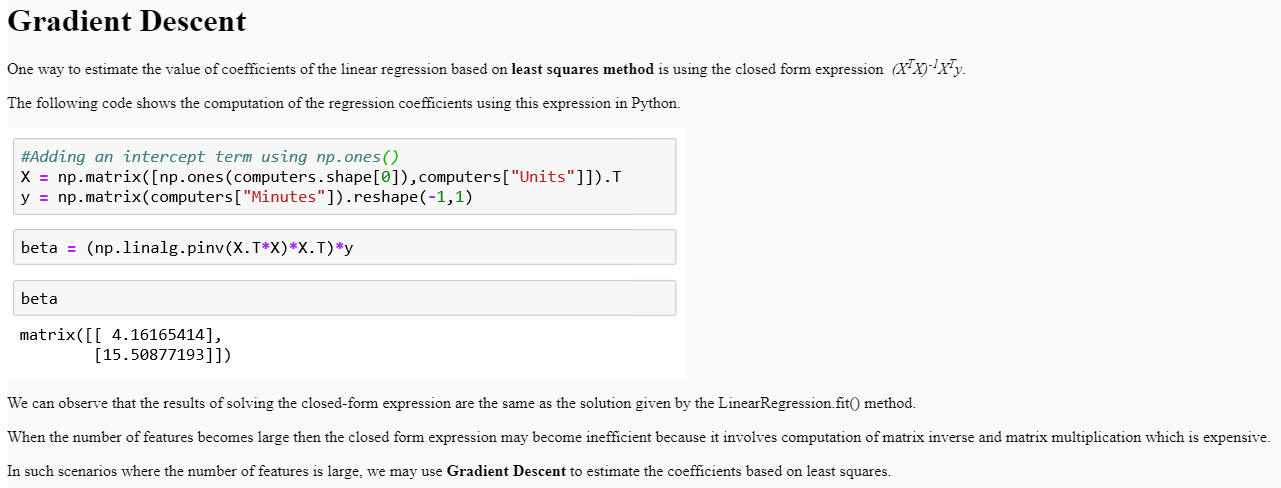
Here, we speculated 3 models and determined the expected values and the total sum of squared errors. The goal of linear regression is to create a model that predicts the value accurately and consequently has the lowest sum of squared errors (also known as least squares). Such a model is called as the **best fit model**.



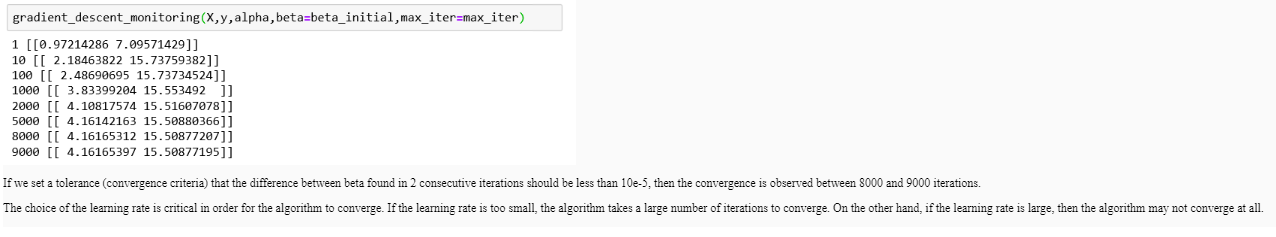


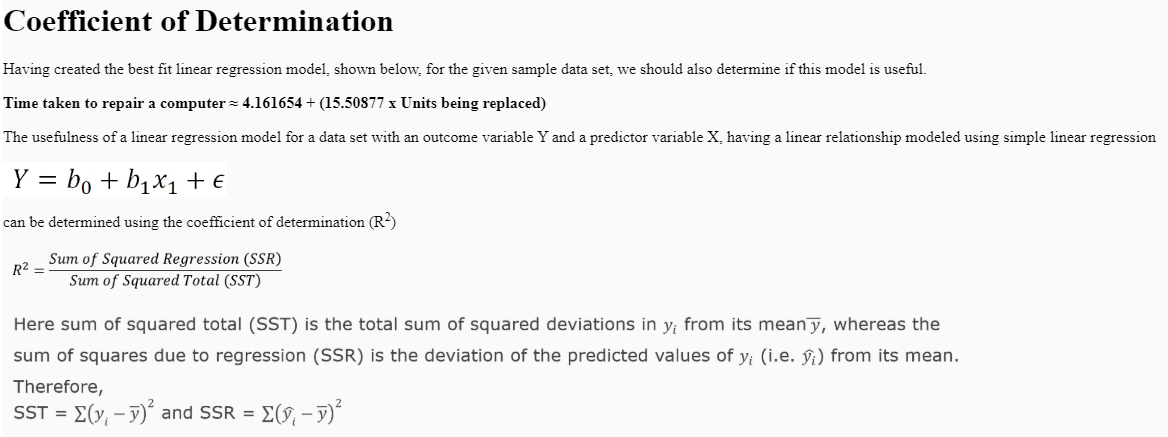


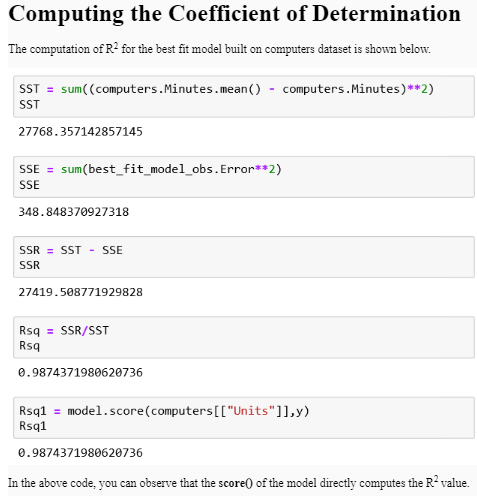












An R2 value of 0.9874372 indicates that 98.74 % of the variability observed in the dependent variable (time taken to repair computer) can be explained by variability in the predictor variable (units being replaced) in this sample data set.

R2can take values in the range 0 and 1. The higher the value of R2, the more useful is the model.

**Multiple Linear Regression**

Recall that in a simple linear regression model, the dependent variable Y is related to a single predictor variable X. This simple linear regression model having the dependent variable Y can be described using



However, there may be more than one predictor variables available. For example:

1. The volume of a tree trunk might be dependent on its height and girth.
2. The price of a house might be dependent on the number of bedrooms, the built-up area of the plot, the age of the house etc.
3. The height of a child might be dependent on age, weight, heights of the parents etc.

In order to predict the dependent variable based on multiple predictors, we use a **Multiple Linear Regression model** described as



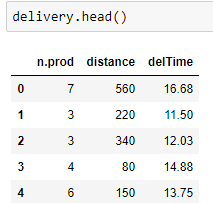
The regression coefficients *ßj*, where j = 1 to n

# Multiple Linear Regression for the delivery dataset

Let us consider the delivery dataset to understand multiple linear regression. The delivery dataset provides information about the time taken to bottle a soft drink using the vending machines.

The delivery dataset is a data frame with 25 observations on the following 3 variables.

* **n.prod**: Number of Products stocked in a vending machine.
* **distance**: Distance traversed by the driver mechanism of the vending machine to dispense a soft drink.
* **delTime**: Time required to deliver(bottle) the soft drink.

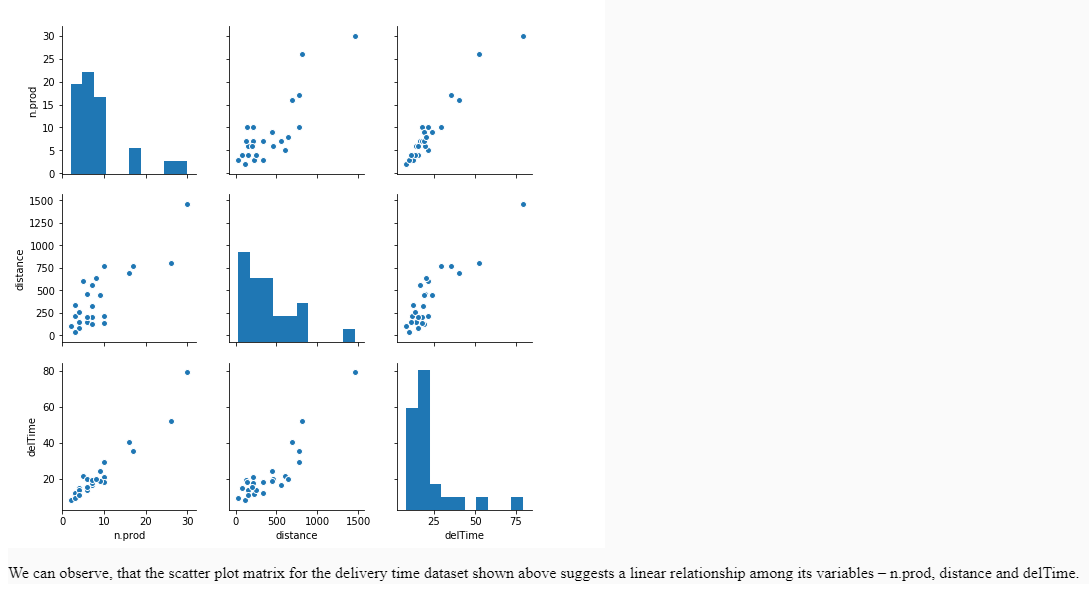


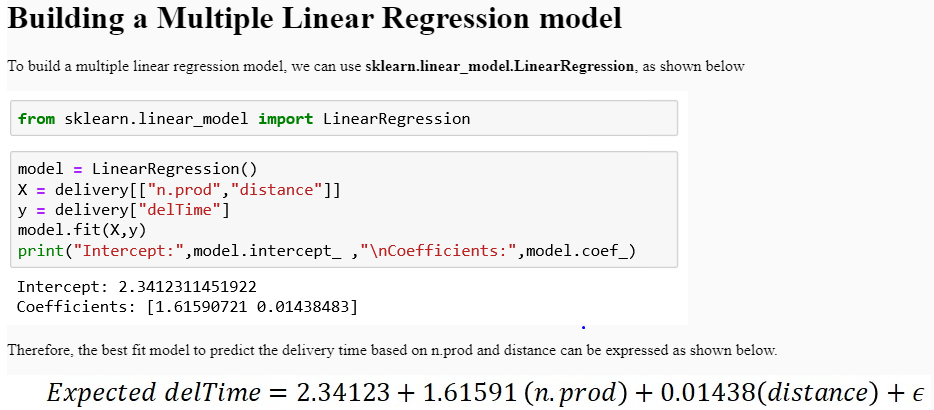


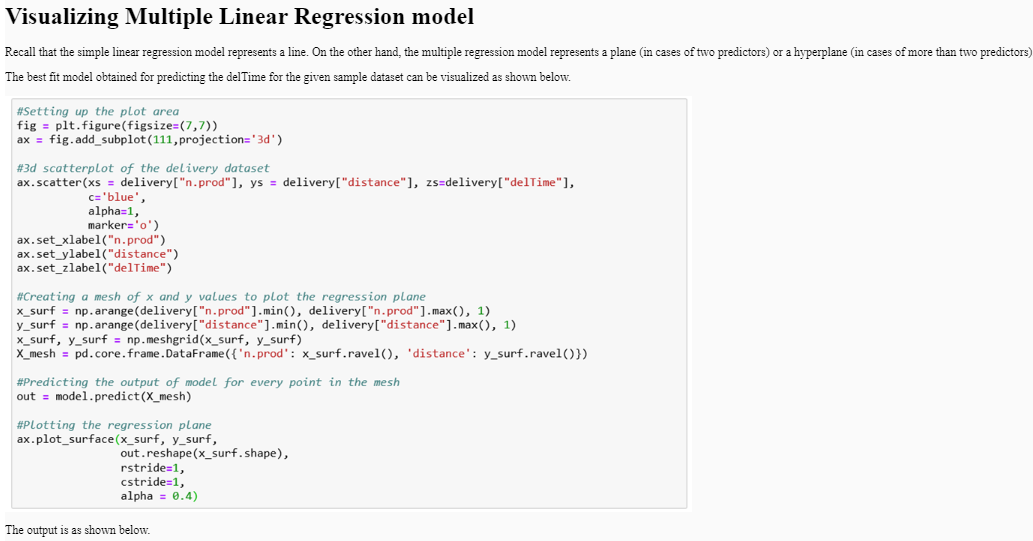
# Visualizing the delivery dataset

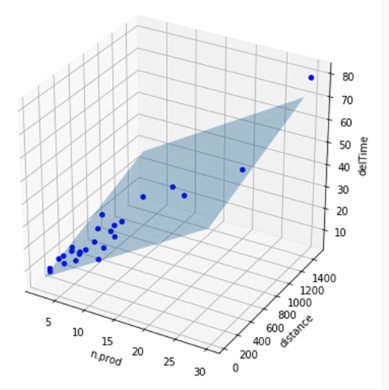
A **scatter plot** can be used to get an insight on the nature of the relationship between the variables. Using a scatter plot, we can visually determine whether there exists a linear association between the variables.

The scatter plot matrix of the variables - n.prod, distance and delTime is shown below.









# MultiCollinearity

In a multiple regression model where two or more predictor variables are involved, it is possible that one predictor can be linearly predicted from the others, with a substantial degree of accuracy. In such a situation, the predictors are said to be **highly correlated**. In statistics, this phenomenon is called multicollinearity, or in other words collinearity (Definition from Wikipedia: Multicollinearity).

Due to collinearity, the coefficient estimates of the multiple regression may change erratically in response to small changes in the model or the data. But, for a linear regression model to be valid it is essential that the predictors of the model be linearly independent of each other.

In case of the delivery time dataset, seen previously, the obtained best fit model



shall be valid only if the predictor variables (n.prod and distance) are linearly independent of each other.

The variables are suggested to be linearly dependent if the correlation values are close to 1.

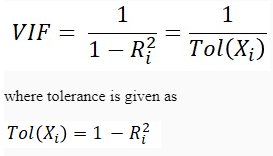
Hence, to determine the strength of linear relationship between the variables, correlation can be used.

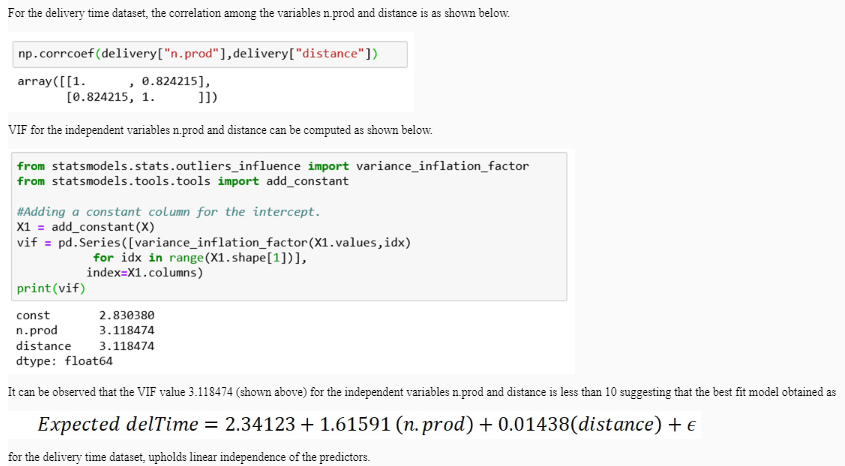
# Variance Inflation Factor

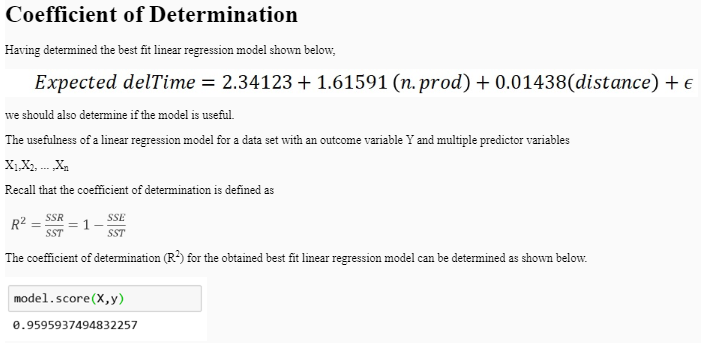
In addition to correlation, we shall use a measure called variance inflation factor(VIF) to determine if the predictor variables are independent of each other.

Variance inflation factor provides an extent of how much the variance of the estimated regression coefficients are inflated as compared to when the predictor variables are not related.

VIF is given as,







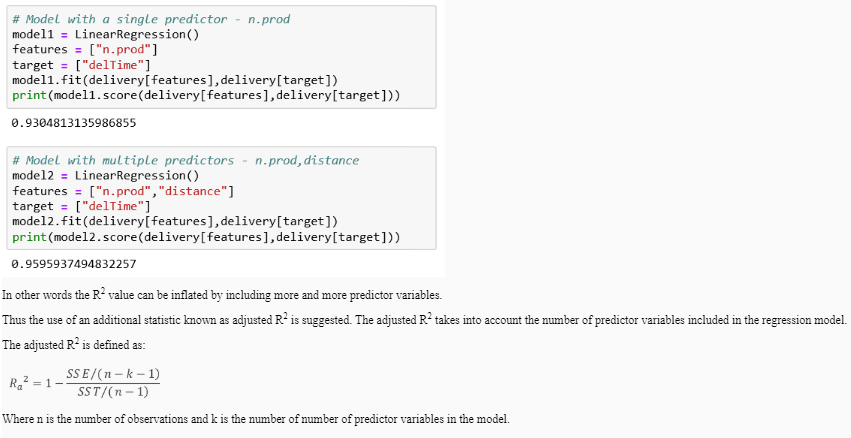
It can be observed that the R2value for the obtained best fit linear regression model is approximately 0.9595937. The R2value here indicates that approximately 95.95 % of the variability observed in the delTime of the vending machine can be explained by variability in the predictor variables, n.prod and distance.

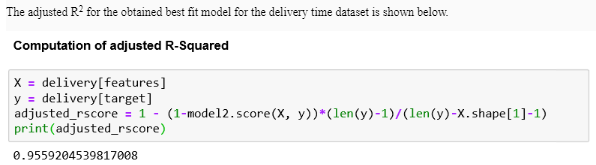
Recall from simple linear regression that R2can take values in the range 0 and 1. The higher the value of R2 (close to 1), the more useful is the model.

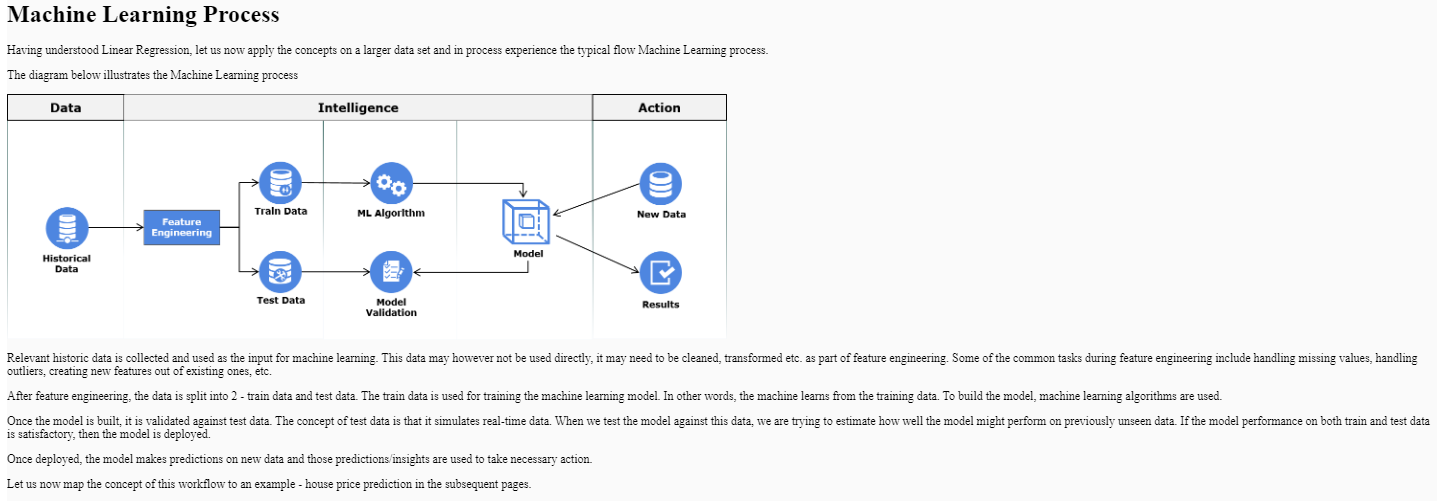
# Adjusted R-squared

Using the least squares method we try to establish a best fit linear regression model with minimum error. For a linear regression model every additional predictor variable tends to minimize the error of the model. As a result the R2 value will never decrease for any number of additional predictor variables being included in the model.

The below code illustrates the increase in the R2for an additional predictor being included in the model.





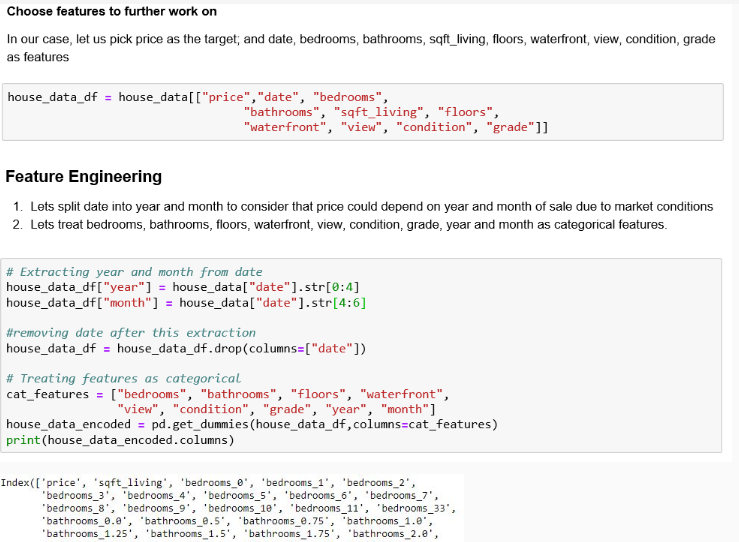


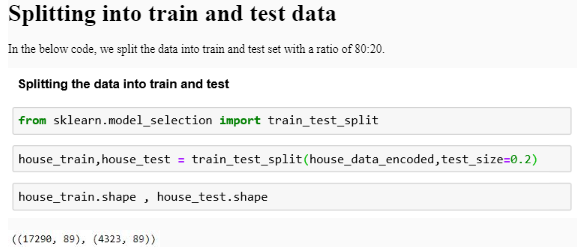
We shall use the King County House price dataset for this exercise. The data represents the details of houses sold in King County, USA in 2014 and 2015.

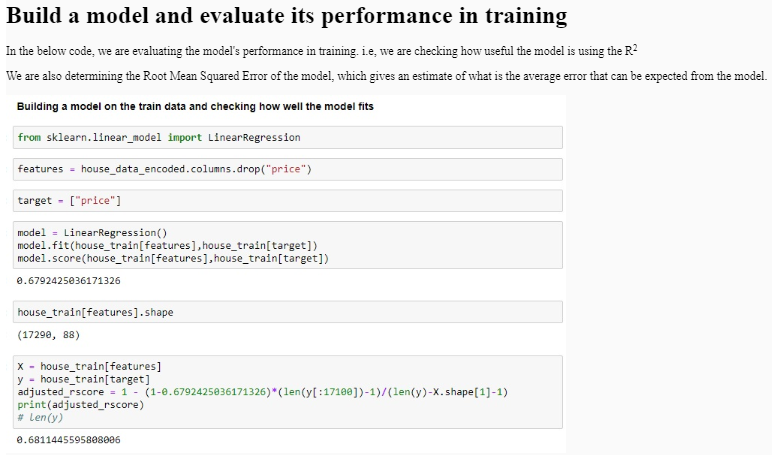
# Feature Engineering

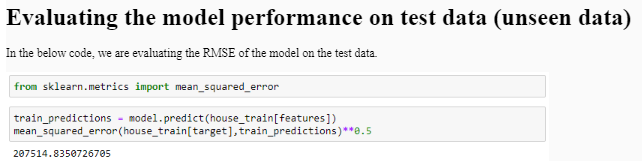
In the below code, we are:

* Selecting features to work on.
* Creating 2 new features - year and month from the date
* Converting features









During the evaluation of a model on train and test data, we could encounter some of the below situations

# Model performance on train and test data is poor (High RMSE)

Such models are typically referred to as **underfit** models because the model is not able to explain the variation in the data reasonably. In such situations, we need to go back and re-look at the quality and veracity of the data and also analyze the features selected to build the model. We may need to invest more time and effort in engineering features.

# Model performance on train data is good (Low RMSE, High R-squared) but on test data is poor (High RMSE)

Such models are typically referred to as **overfit** models, i.e. they have been fit perfectly for the train data but are not generalized enough. In such situations, we may need to:

* Gather more data instances - It is difficult to overfit larger size of data.
* Reduce the complexity of the model - Evaluate which features are important and use only them to build the model.
* Tune **hyperparameters** of the algorithm - This will be discussed later.

# Classification

Recall that a machine makes 2 kinds of predictions - a **category** (e.g. human/non-human in case of the self-driven cars) and a **quantity**(e.g. success score in case of the marketing campaign).

When predicting a quantity, the models built are referred to as regression models. Simple linear regression and multiple linear regression analysis techniques were the two examples of the same that we just discussed.

Whereas, when predicting a category, the models built are referred to as classification models. The category of a new instance must be predicted based on the existing data. This can be done by finding patterns in the available data and then making a prediction based on the observed patterns.

The categories of the data instances are referred to as class labels or classes. The attribute that holds these categorical values is called the class label attribute. The class of a new instance can be predicted by studying the patterns in the previously processed data. The attributes that are used in order to predict the value of the class label attribute are called the predictor attributes. The previously processed dataset is also referred to as historical dataset since the instances in it are already labeled.

Finally, the entire task of prediction of classes of new instances is termed as **classification**.

For the purpose of classification one needs to move from historical data to a general model, known as the **classifier**. This classifier can be arrived at by analyzing patterns in the historical data, so as to classify new instances.

In this course, we shall discuss a few of the classification techniques/algorithms namely **Logistic Regression, Decision Trees, k- Nearest Neighbors (kNN) and Support Vector Machines (SVM)**, **Neural Networks**.

# Logistic Regression example - Coronary heart disease

Consider the coronary heart disease dataset which lists the age in years (AGE) and the presence/absence of evidence of significant coronary heart disease (CHD) for 100 subjects.

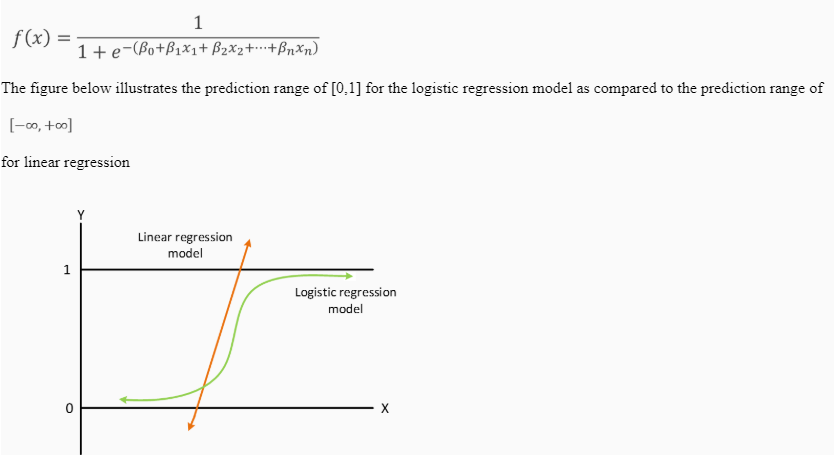
The variable chd = 0 indicates the absence of coronary heart disease, whereas chd=1 indicates the presence of coronary heart disease.

the dependent variable (chd) is bound to the values 0 and 1.

# Logistic Regression vs Linear Regression

Recall that in linear regression models, the dependent variable is continuous quantitative in nature. However, in the case of the coronary heart disease dataset, the dependent variable is categorical in nature. In such situations, we use logistic regression to model the relationship.

The logistic regression model is given as

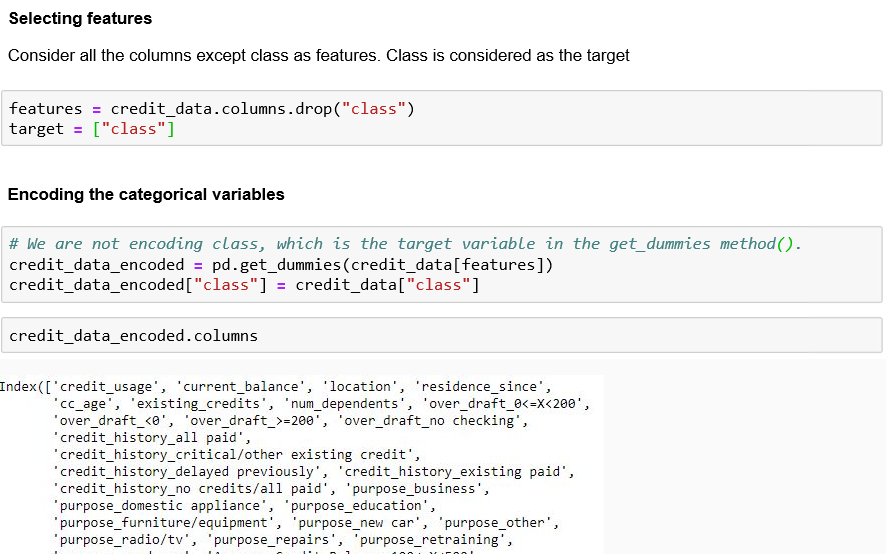


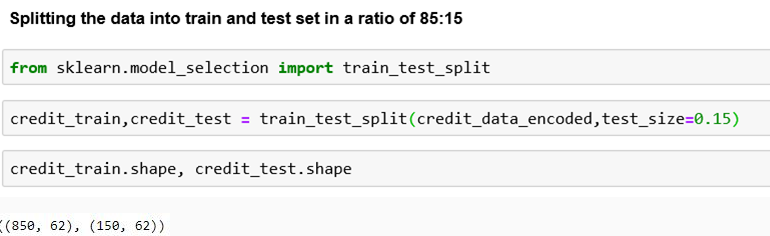


# Logistic Regression on credit risk data

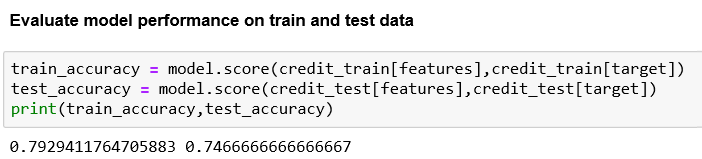
Having understood the basic concept behind logistic regression, we shall now use logistic regression to classify if a loan applicant is a good/bad credit risk. We shall use the credit risk data set for this.

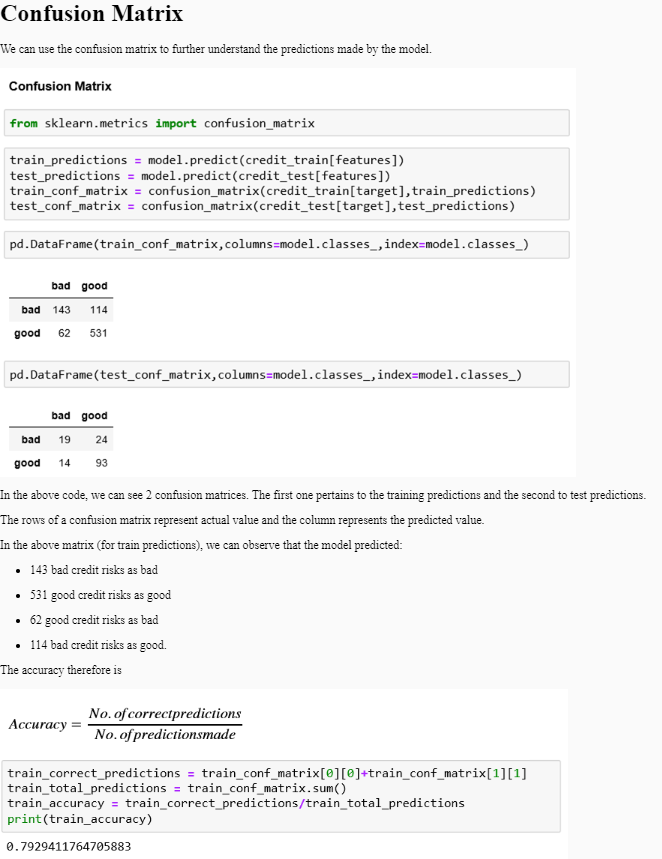
In the below code, we are importing the data and understanding its structure.





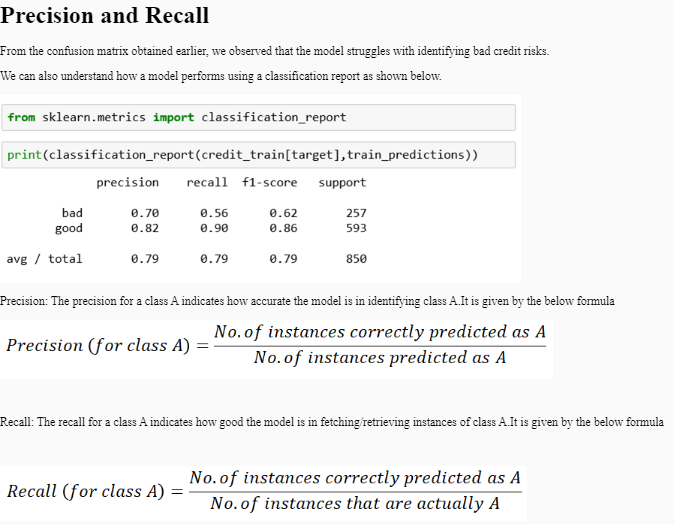






Using the confusion matrix to understand the predictions made is essential whenever there is a cost associated with wrong predictions. The confusion matrix can help us evaluate the cost-benefit analysis of a model.

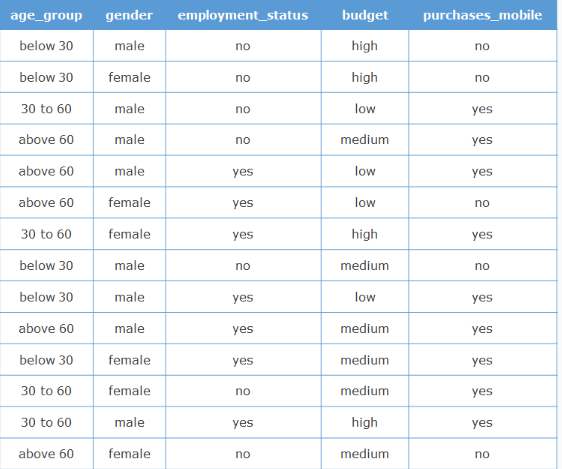
For example, if we consider that there is a huge loss of capital whenever a bad credit risk is predicted as good because the applicant is likely to be a defaulter and the lender will receive neither the principal amount or interest. In such cases, we will need to choose a model that is better at correctly identifying bad credit risks.

While it is desirable to have a model with both high precision and recall, it often is a trade-off.

To evaluate a model with a single metric based on precision and recall, we can use the F-Score. This metric is the harmonic mean of precision and recall and can indicate how good the model is in classifying instances of a particular class. It is defined as follows

# Classification using Decision Trees

Let us now look at Decision Trees which is another algorithm that can be used to build classifiers. To understand the principle behind decision trees, let us consider the below customer data set which comprises details of previous visitors to an online mobile portal

Even though this is a small data set, it could be a tedious task to do classification just by looking at the data. Hence, to simplify the process, let us start by answering a few questions.

1. What is the probability that a male visitor purchases a mobile?

    Ans: 6/8 = 0.75

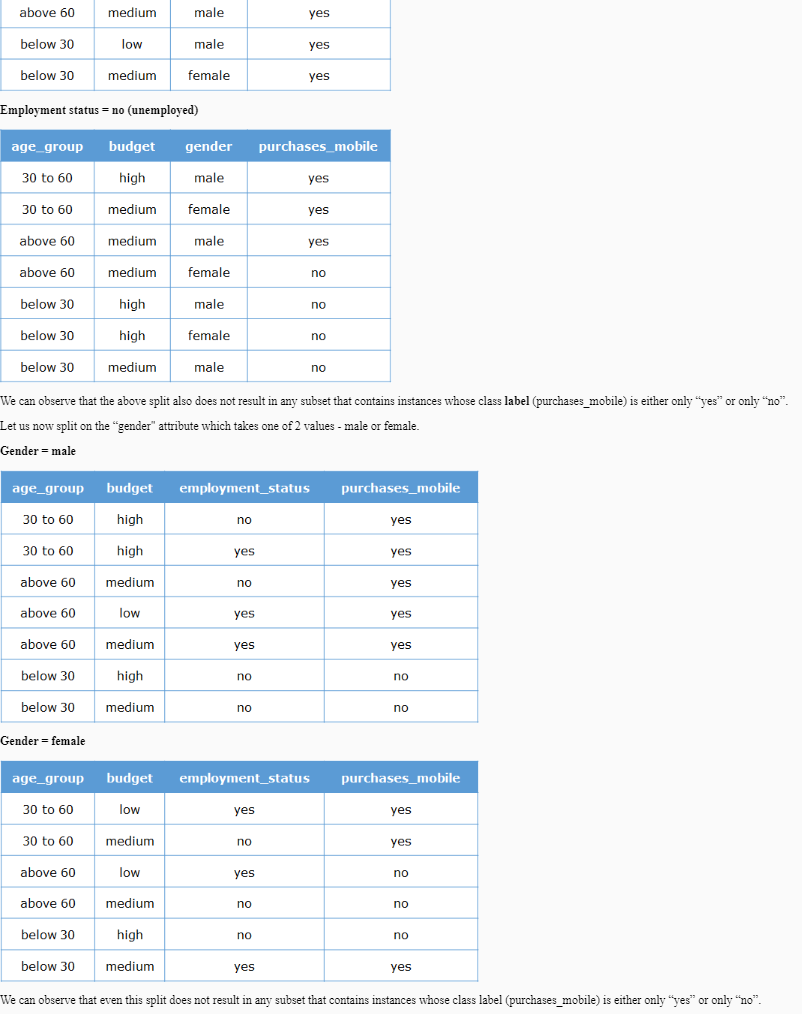
2. What is the probability that a visitor with a medium budget purchases a mobile?

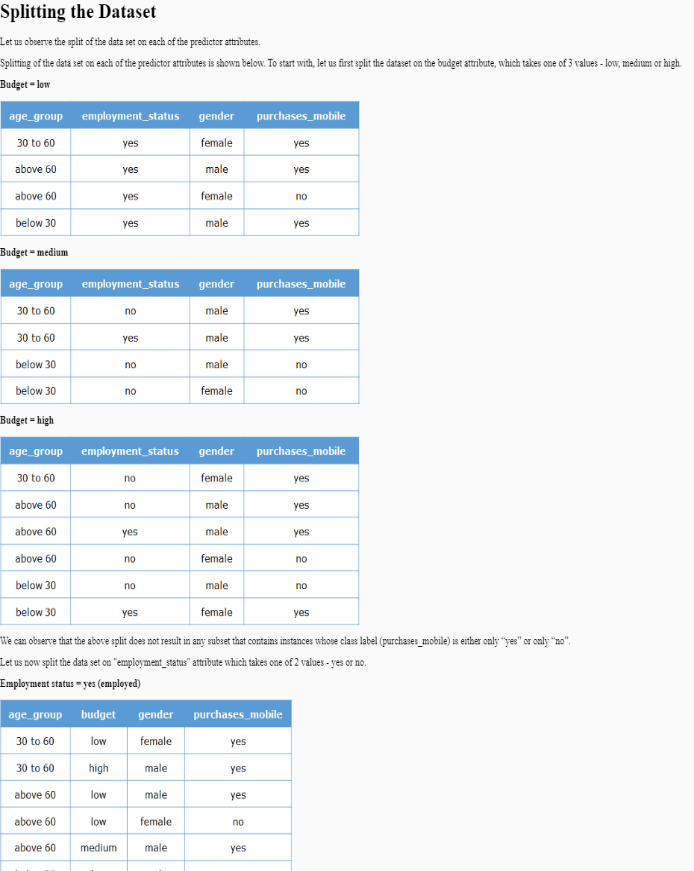
    Ans: 2/4 = 0.50

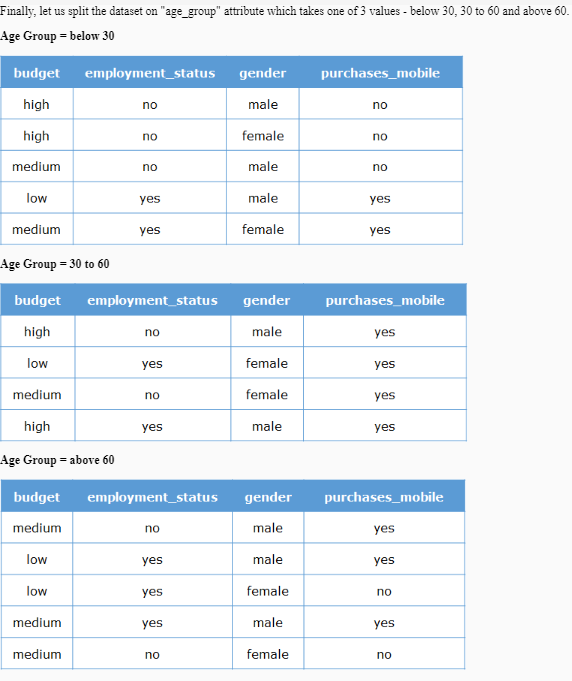
3. What is the probability that a visitor in the age group 30 to 60 purchases a mobile?

    Ans: 4/4 = 1

Here, we can observe that we have split the historical data on a particular predictor attribute into subsets of data in order to answer these questions. The information that we derive from these splits can help us predict the outcome better.

The goal while splitting is to check whether any of the resulting subsets constitute all instances belonging to the same class label.



We can observe that the above split results in one subset that contains instances belonging to a single class label; in other words, all the visitors in the “30 to 60” age group have always purchased a mobile, that is, the class label is “yes” for all of them.

A subset which contains instances belonging to only one class label is called a **pure (homogeneous)** subset. The predictor attribute on which the dataset is split to obtain the maximum number of pure subsets is called the **best attribute**.

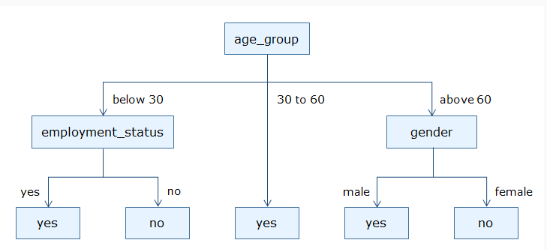
At this point, we can observe that the subsets formed for the “below 30” and “above 60” age group categories are still impure. Hence, the next goal is to keep splitting the impure subsets, based on the corresponding best attributes, until a pure subset is arrived at.

The end result of the splitting process for the sample data results in the following splits:

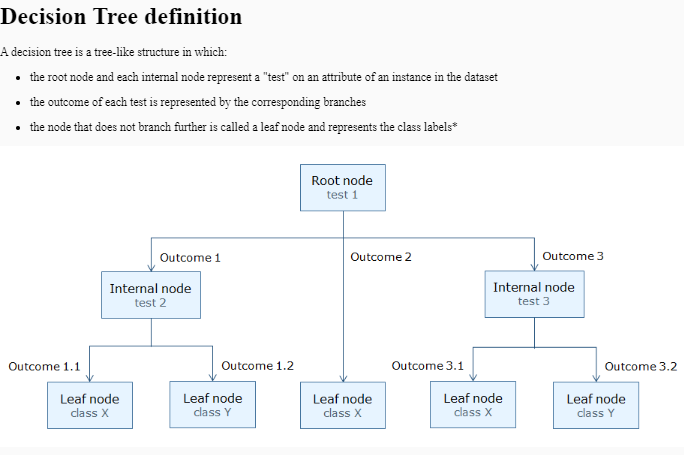
1. The data is first split on age\_group to provide 3 subsets where age\_group takes 'below 30', '30 to 60', and 'above 60' values respectively. all instances in the subset where age\_group = '30 to 60' have purchases\_mobile = 'yes'.
2. The subset where age\_group = 'below 30' is further split on the basis of employment\_status resulting in 2 subsets where employment\_status takes values 'yes' and 'no' respectively. For the subset where employment\_status = 'yes', the value purchases\_mobile of all instances is seen to be 'yes'. Also, for the subset where employment\_status = 'no', the value purchases\_mobile of all instances is seen to be 'no'.
3. The subset where age\_group = 'above 60' is further split on the basis of gender resulting in 2 subsets where gender takes values 'male' and 'female' respectively. For the subset where gender = 'male', the value purchases\_mobile of all instances is seen to be 'yes'. Also, for the subset where gender = 'female', the value purchases\_mobile of all instances is seen to be 'no'.

We can observe that the splitting of data seizes whenever a pure subset is obtained.

We can also represent the diagram showing the end result of splitting such that it contains only the names of the predictors based on which we split the data and the resultant class labels in the obtained pure subsets.

This diagram represents various decision rules that we can traverse in order to predict the value of purchases\_mobile. For example, we predict that the value of purchases\_mobile is likely to be "yes" when the age\_group is "below 30" and employment\_status is "yes".

This diagram also looks like an inverted tree. Also, since a decision is taken on how to split the data at each node of this tree, it is called a **Decision Tree**.



# Decision Tree Algorithm

The basic algorithm to induct (create) a decision tree is shown below. It takes three parameters:

1. Instances – the set of instances for which class label is already known
2. Target\_Attribute – the class label attribute
3. Attributes\_List – the list of predictor attributes
4. Algorithm (Instances, Target\_Attribute, Attributes\_List)
5. If all Instances are of the same class, return the single-node tree, with the label of that class.
6. If Attributes\_List is empty, then
7. return the single-node tree, with the label as the most common value of the target attribute among the Instances.
8. Else Begin
9. A ← The Attribute that best classifies Instances from the Attributes\_List.
10. For each possible value, v of A,
11. Add a new tree branch below the current node, corresponding to v.
12. Let Instances (v) be the subset of examples that have the value v for A.
13. Below this new branch add the subtree, Algorithm (Instances, Target\_Attribute, Attributes\_List)
14. End

We can observe that the above algorithm does not state how to select the best attribute. The selection of the **best attribute** is performed based on various attribute selection measures.

# Attribute selection measures

Attribute selection measures compare different predictor attributes and rank them for the purpose of model construction. They convey, how much more a predictor attribute is relevant compared to the other predictor attributes. Decision tree algorithms make use of the ranked list given by the attribute selection measures and then select the best attribute to split the data set.

Three of the most commonly used attribute selection measures to induct a decision tree are:

* Information gain
* Gain ratio
* Gini Index

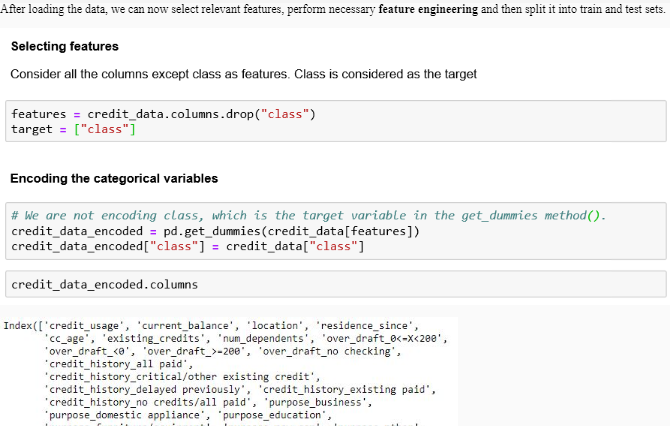
The above measures are incorporated by various algorithms to select the best attribute.

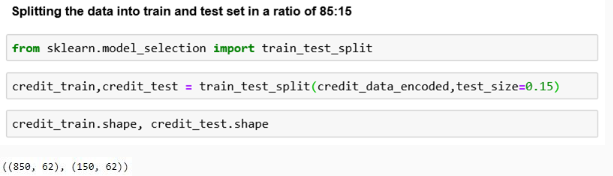
In this course, we shall build trees using the classes in **sklearn.tree** package. The attribute selection measures used here are either gini index or information gain.

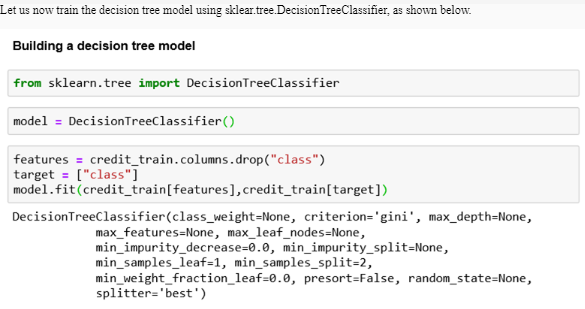
# Decision tree on credit risk data set

Having understood the basic principle behind decision trees, let us now use decision trees to classify if a loan applicant is a good/bad credit risk using the credit risk data set.

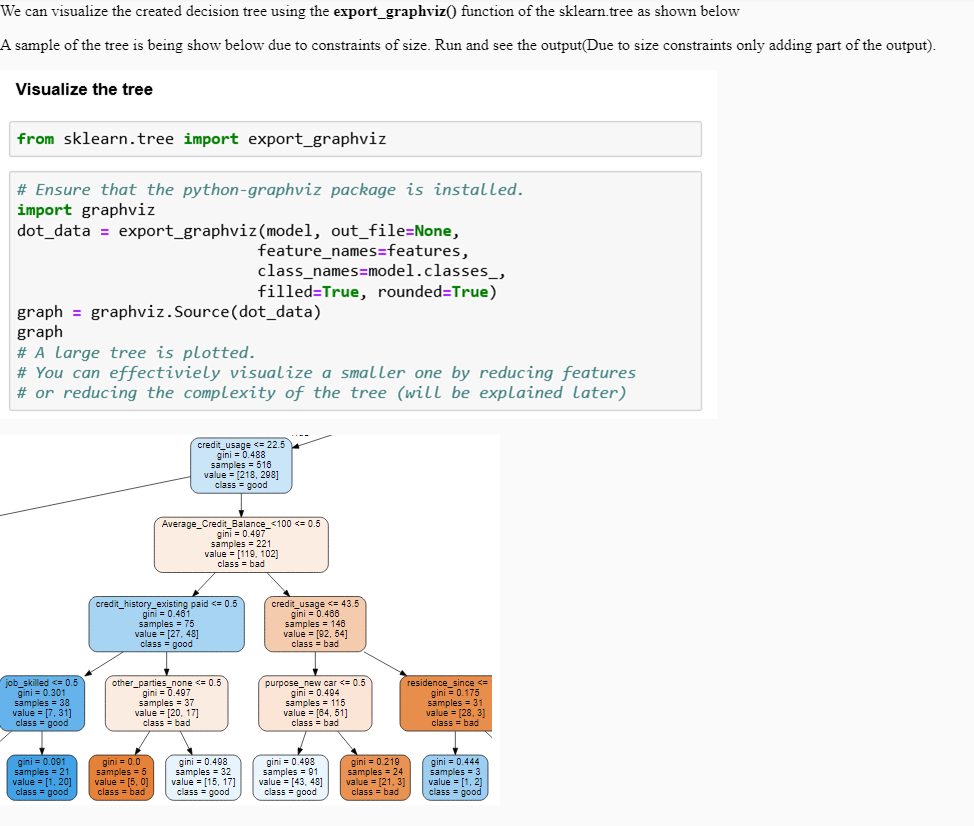
In the below code, we are loading the data from a CSV file







You can observe that the way we create a decision tree model is similar to how we create a logistic regression model. This is an advantage of using the sklearn package. It provides a simple interface for building machine learning models

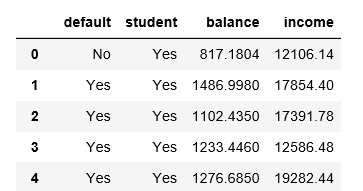




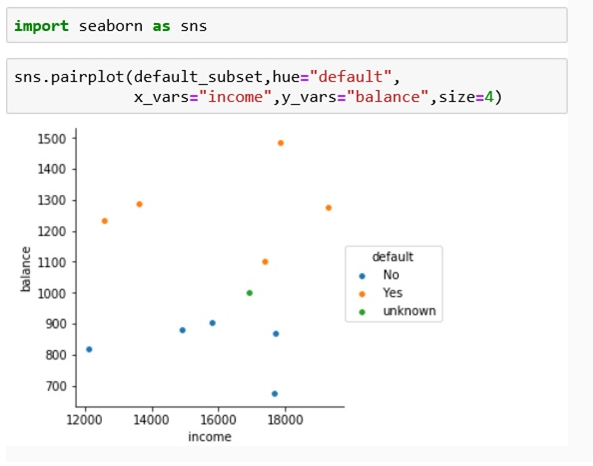
# Classification using k-NN

k-Nearest Neighbors (kNN) is a non-parametric learning algorithm used for classification.

This algorithm tries to determine the class label of a new data point (instance/tuple) by comparing it with existing data points (instances/tuples), that are closest to it. Here it compares the new data point with the k closest data points to make the predictions

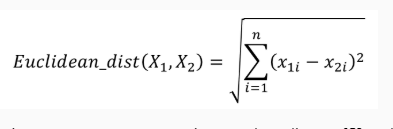
Let us consider the default dataset, to understand how kNN helps classify tuples.

The default dataset contains 10,000 observations for 4 variables, as shown below.We can observe that the variables “balance” and “income” are continuous numeric variables whereas the variables “default” and “student” are categorical variables.

The scatter plot of income vs balance is shown below indicates defaulters and non-defaulters. You can observe that the customer with a balance of 1000 and income of 16,900 is currently not classified as either defaulter/non-defaulter.

# Euclidean Distance

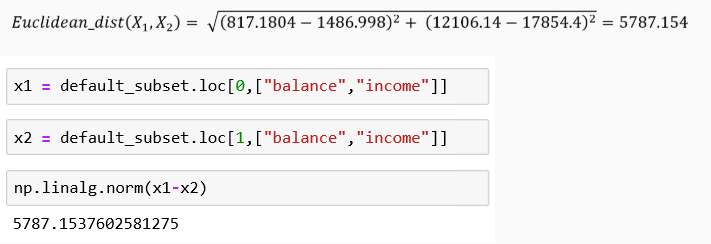
The kNN algorithm identifies the nearest neighbors based on Euclidean distance\* - a commonly used distance metric. The Euclidean distance between two (tuples) - X1 (x11, x12, x13, .. x1n) and X2 (x21, x22, x23, .. x2n) can be computed as

where, x11, x12, x13, .. x1nare the numeric attributes of X1and x21, x22, x23, .. x2nare the numeric attributes of X2.

In the given scenario, each instance has two numeric attributes – balance and income. Thus, each instance here is represented as a point in a 2-dimensional space.

Let us now compute the Euclidean distance between a few tuples in the given scenario.

Let X1 be the first tuple (balance=817.1804, income=12106.14, defaulter=No) and X2 be the second tuple (balance=1486.998, income=17854.4, defaulter=Yes). The Euclidean distance between X1 and X2 can be computed as



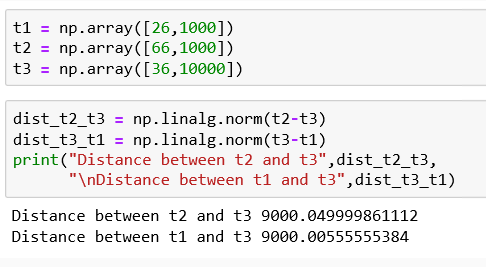
Note : Distance metrics such as Hamming distance can be used when the data tuple comprises of categorical attributes.

One of the caveats of Euclidean distance is that attributes having larger ranges, contribute more value to the Euclidean distance.

For example, if the numeric attributes are age and loan amount, then it can be observed that the typical age in years for humans may range from 0 to 100 years, however, the loan amount in dollars may range from 0 dollars to several thousand dollars.

Let us illustrate this with a simple example. Consider 3 data tuples T1(age=26, loanAmt=1000), T2(age=66, loanAmt=1000) and T3(age=36, loanAmt=10000).

The Euclidean distance of the tuple T3 from T1 and T2 can be computed as

We can observe that the difference in the values of age for T1 and T3 (i.e. 26-36=10) is significantly less than the difference in the values of age for T2 and T3 (i.e. 66-36=30). However, the Euclidean distance between the tuples T1 and T3 is same as that of the Euclidean distance between the tuples T2 and T3, suggesting that the parameter loanAmt has a high influence on the Euclidean distance as compared to the age.

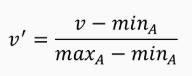
To avoid this situation, all the numeric attributes of the tuples can be **normalized** before they are used for computing the Euclidean distance. Normalization is essential to take into consideration, the different measurement scales of the attributes. Normalization is not just applied to kNN, it is often considered a good practice to normalize the data before running any machine learning algorithm.

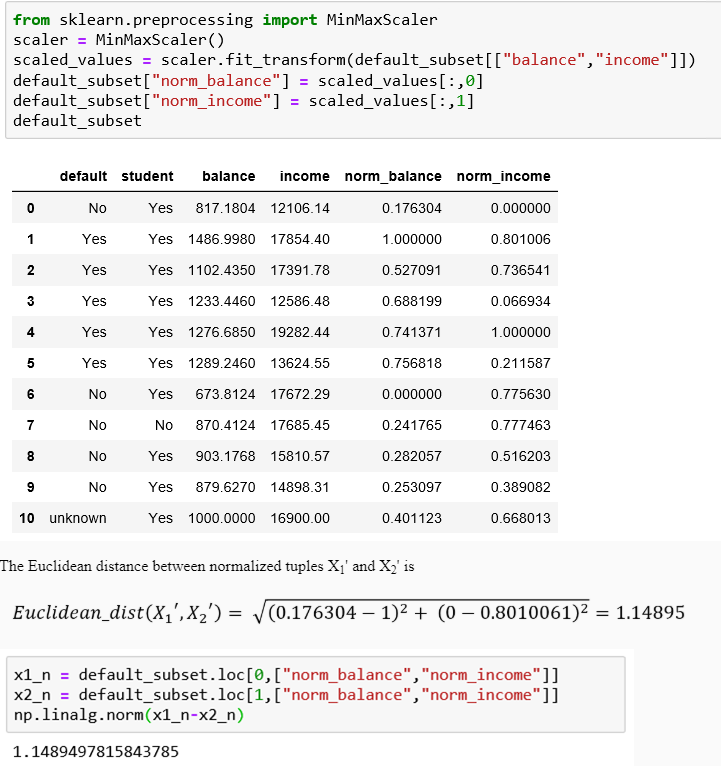
# Normalizing the Data

There are several normalization techniques that can be applied. These methods are available in sklearn.preprocessing.

Let us now normalize the balance and income in the default (subset) and find the euclidean distance between instances X1 (balance=817.1804, income=12106.14, defaulter=No) and X2(age=1486.998, income=17854.4, defaulter=Yes).

To normalize the data, we shall us the min-max normalization. This normalization transforms the value v of a numeric attribute A to a value v', where the



We shall first normalize the values of the attributes of X1 and X2. Let us use the min-max normalization technique to normalize the age and loan amount data.

The below code demonstrates the use of MinMaxScaler in sklearn.preprocessing to normalize the data.

Recall that the kNN algorithm identifies the k nearest neighbors of a new tuple and assigns it a class label based on the class labels of the nearest neighbors.

Let us consider the value of k as 1, which means that, the kNN algorithm assigns a class label based on its nearest neighbor. The kNN algorithm would therefore classify the tuple X11 as default= Yes based on its nearest neighbor X3 since the nearest neighbor (X3) has a class label of default=Yes.

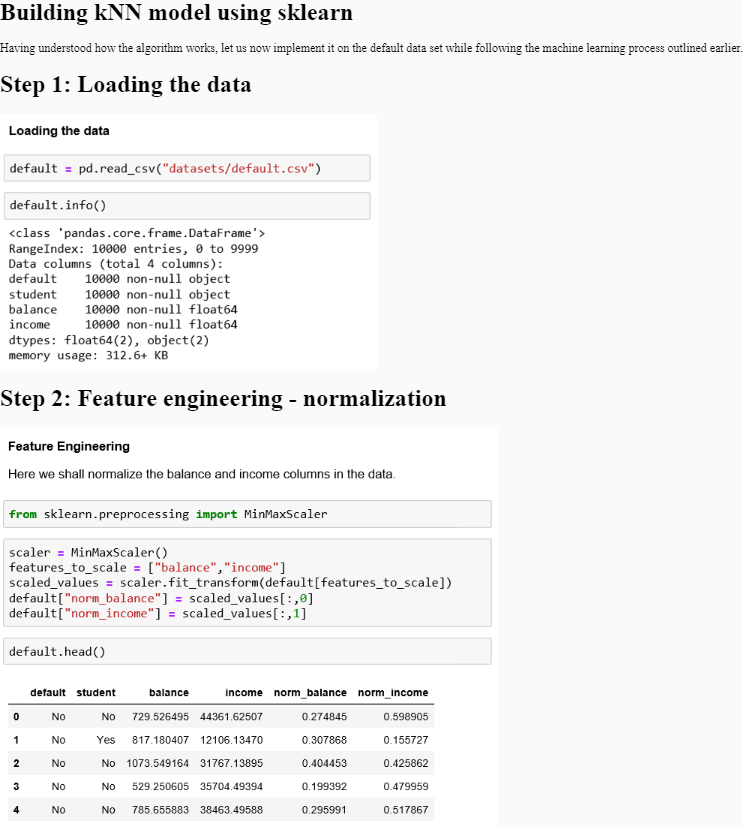
For all values of k greater than 1, a data tuple is classified by a majority vote of its neighbors, with the tuple being assigned the class most common among its k nearest neighbors.

If the value of k was selected as 3, the 3-nearest-neighbors of the data tuple X11 (balance=1000, income=16900) are identified. In this case they are X3 (balance=1102.435, income=17391.78, default=Yes), X9(balance=903.1768, income=15810.57, default=No) and X8(balance=870.4124, income=17685.45, default=No) based on the Euclidean distances computed earlier.

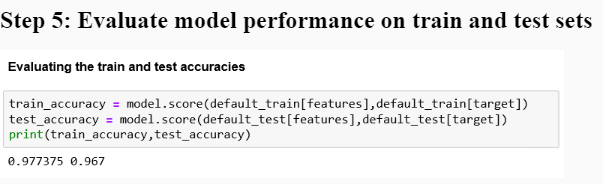
The new tuple would be classified as defaulter/non-defaulter based on the majority vote of the neighbors X3, X9and X8. In this case, the majority vote indicates that the new tuple (X11) would be a non-defaulter

For k=2, X3 (balance=1102.435, income=17391.78, default=Yes) and X9(balance=903.1768, income=15810.57, default=No) are the 2 closest neighbors of X11 based on the Euclidean distances. In this case, we observe a tie in the votes (i.e. 1 vote for defaulter and 1 vote for non-defaulter).

It is therefore difficult to classify the data tuple X11 as defaulter or a non-defaulter because the vote count results in a tie. Typically for **binary-class** cases (like defaulter= Y/N), an odd value of k is preferred to avoid such tie situations. For **multi-class** cases (more than 2 classes), the ties can be broken by assigning a class at random or by assigning the class that occurs most frequently.

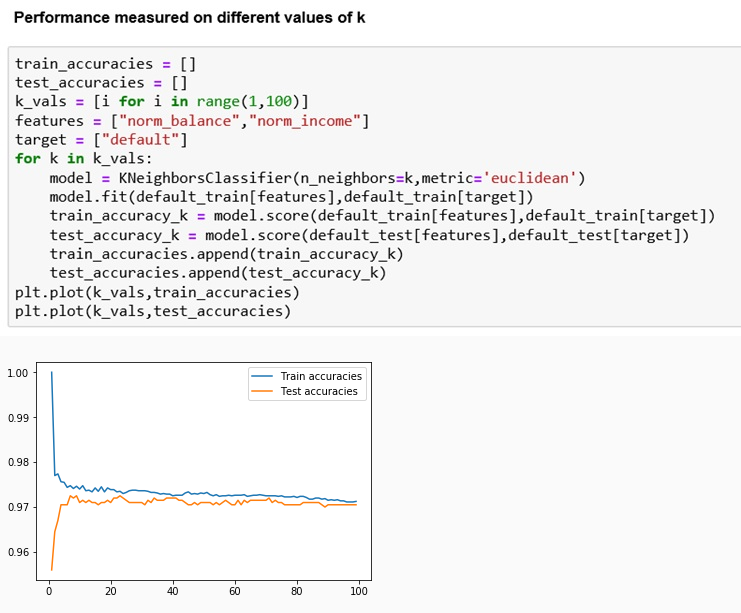




In our example, we chose the value of k=3 while building the model. However, the choice of k affects the performance of the model. Typically, lower values of k lead to overfitting considering that the model is very specific to the nearest data point. On the flip side, a very the maximum value of k, k = number of instances, gives a model that predicts the class with maximum probability and is an underfit model.

So, for the kNN algorithm, k is a hyperparameter that needs to be tuned.

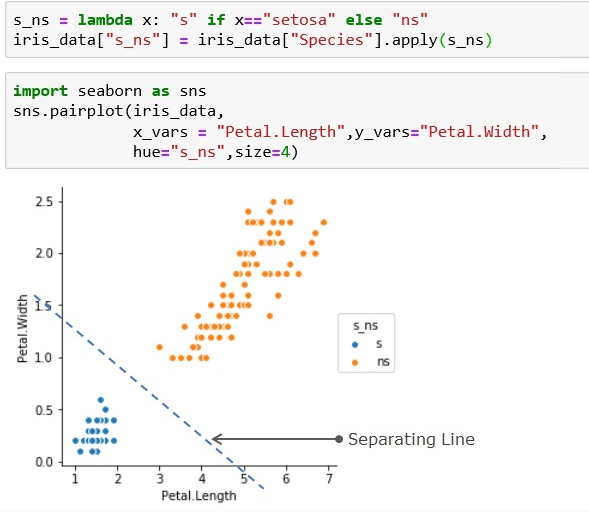
The below code demonstrates the train and test accuracy for different values of k on the default dataset



# Support Vector Machines

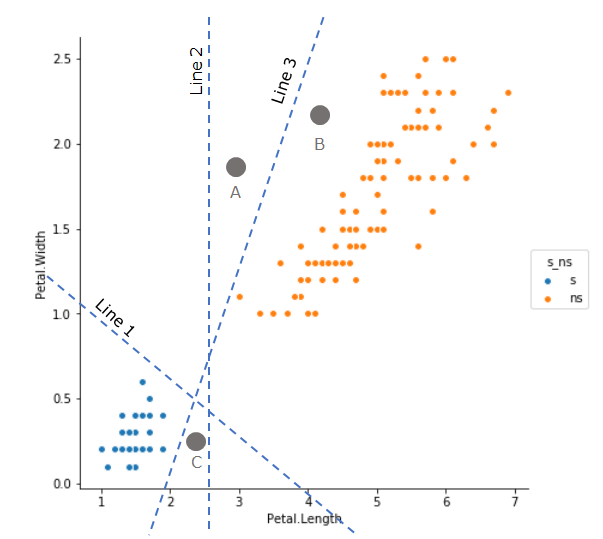
Consider the iris dataset which provides measurements of sepal length, sepal width, petal length, and petal width for 50 flowers from each of 3 species (setosa, versicolor, and virginica) of iris.

If we were to classify an iris flower as setosa or non-setosa based on Petal Width and Petal Length, we could use a separating line to do the same as shown below.

Note: the separating line here has been drawn manually for illustration purpose

Classifying data by using separating hyperplanes is the premise of Support Vector Machines.

Support Vector Machines (SVM) is another classification algorithm that classifies data into one category or the other by using hyperplanes based on the training data.

Consider the plot below which illustrates a few lines that can separate data into setosa or non-setosa.

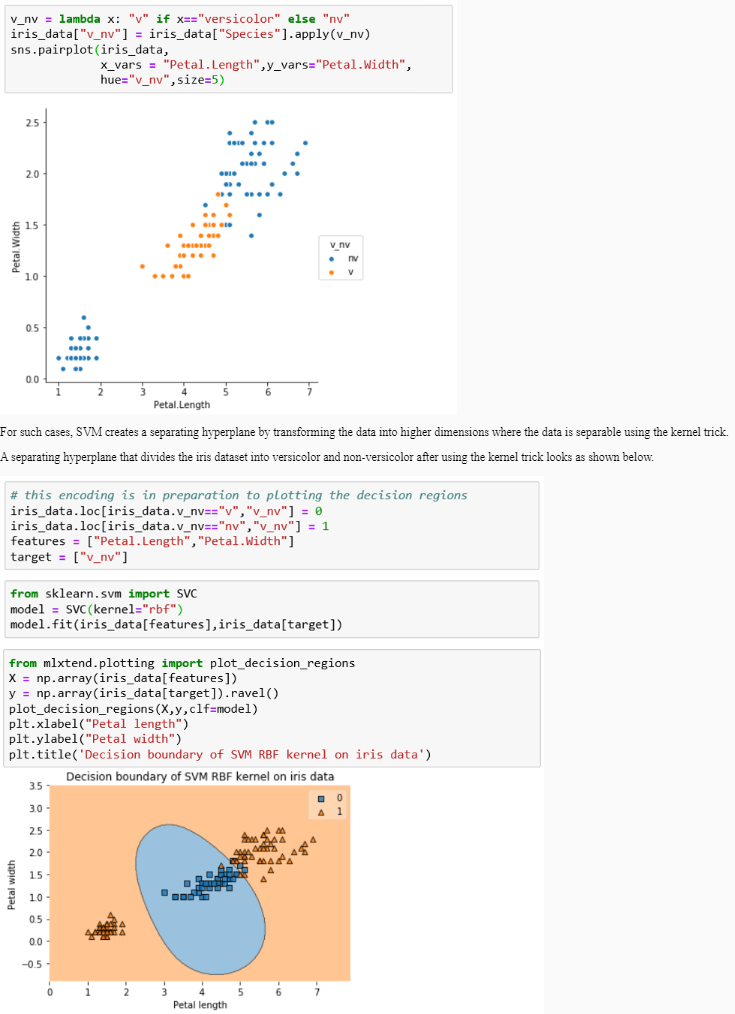
From the above plot we can observe that all three lines – Line 1, 2 and 3 separate the given sample data into different classes. Points on the left of these lines are marked setosa and the ones on the right are marked non-setosa.

Let us now consider three new data points (A, B and C) where the class labels are unknown. We can observe that A is classified as setosa by Line 3 and non-setosa by Line 1 and 2. The following observations can be made from the above plot:

1. There can be many lines that separate the known data correctly
2. These lines may agree/disagree with each other when newer data points are presented
3. Point A lies very close to Line 2 and 3. It is possible that the lines would have misclassified point A.

Based on the above observations, the goal of SVM can be redefined to find an **optimal hyperplane** that separates the data into classes.

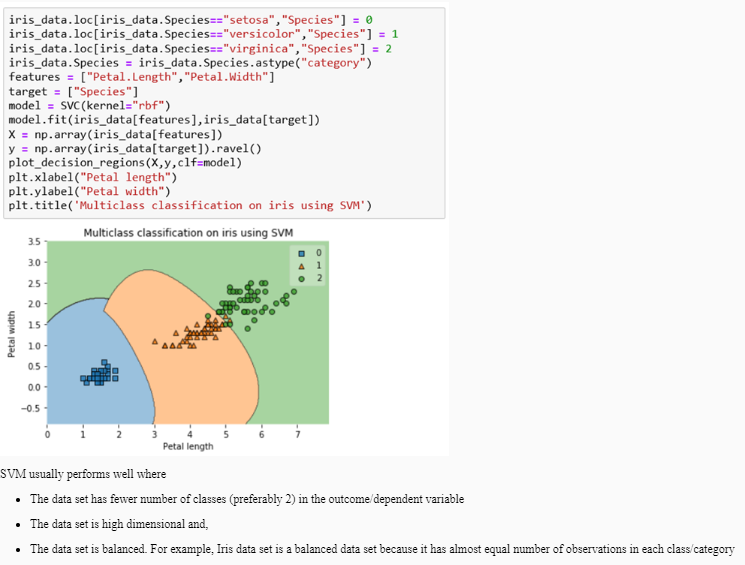
Consider the below plot where we want to find the separating hyperplane to classify data as versicolor and non-versicolor. Here we can observe that the data is not linearly separable and hence we cannot separate it using a line.



Further, even though SVM is considered as a binary classifier, it can be used to for multi-class classification as well. This is achieved in one of 2 ways:

1. One-vs-One classification: It compares every class against every remaining class, building n \* (n-1) / 2 models, where n is the number of classes. So if there are 3 classes (in our case) then (3\*2)/2=3 models are used.
2. One-vs-All classification: It compares every class with the remaining classes thereby building a model for every class. The class with the highest probability is the chosen one. So if there are 3 classes (in our case) then 3 models are used.

The below plot shows the classification of Iris data set using SVM for multiple classes – setosa, versicolor and virginica.



# Real life examples of SVM

**Predicting diabetes**

The number of people with diabetes has risen from 108 million in 1980 to 422 million in 2014. In majority of the cases, diabetics can be predicted before it affects the person. ‘BMC Medical Informatics and Decision Making’ used data captured by National Health and Nutrition Examination Survey (NHANES) to develop and validate SVM models to predict diabetes based on family history, age, race and ethnicity, weight, height, waist circumference, body mass index (BMI), and hypertension.

**Detection of Analyzer faults in Petrochemical industry**

Petrochemical industries incur a huge loss because of abnormal situations such as process disturbance and equipment failures. One important group of abnormal situations in process industries are faults in on-line product analyzers. SVM is used to predict the abnormalities so that it can be attended before the breakdown.

**Text Categorization to find interesting information on WWW**

With the overload of online information, it is of paramount importance to categorize the online stories/information into categories so that they could be discovered by the reader in a convenient manner. SVM is used to create text classifiers that help to achieve the aforementioned.

# Tuning model hyper parameters using cross-validation

So far we have built various machine learning models using algorithms such as Regression, k-Nearest Neighbors, Decision Trees and SVM.

However, while building some of these models, we encountered situation such as overfitting or underfitting. We also briefly discussed that we can control the models using their hyper parameters.

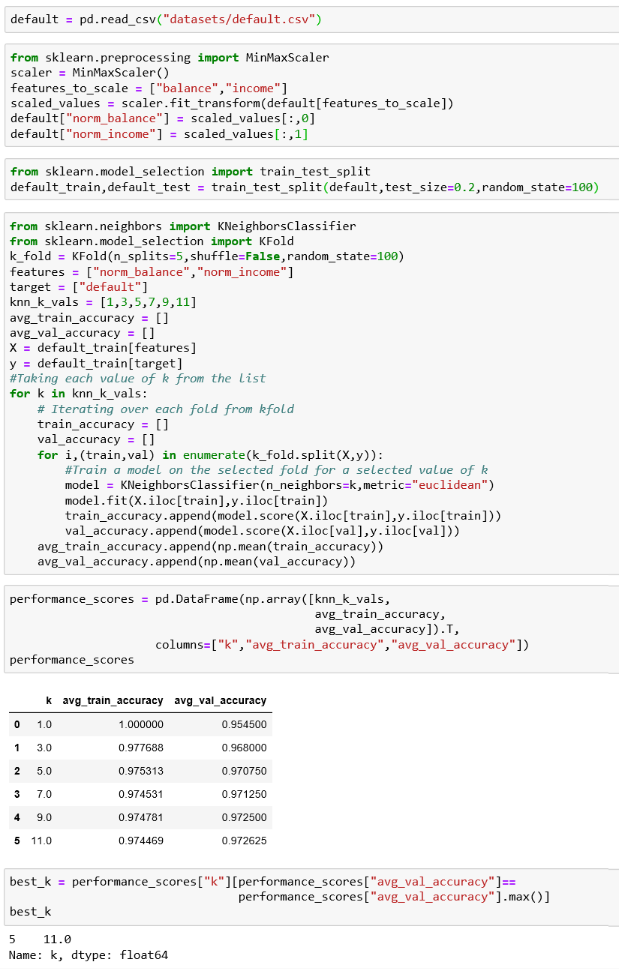
In this section, we shall look at cross validation as one of the approaches to tuning the hyperparameters of the model.

While tuning models, we typically further divide the training data into subsets - train set, validation set. The idea is that the model is trained with different hyperparameters on the train set and its performance is measured on the validation set. The validation set acts as a simulation of test data. The actual test data is however not used while model building or tuning.

Considering the fact that setting aside a separate validation set might further reduce the data from which the machine can learn, we use a popular approach called cross-validation.

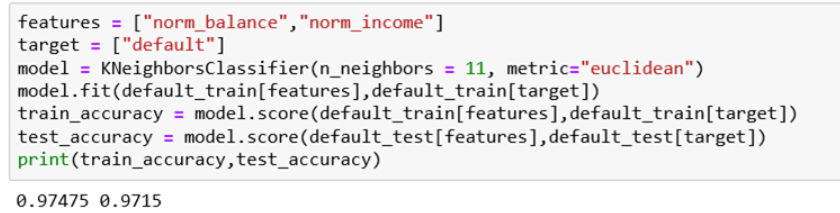
One of the widely used cross-validation technique is k-fold cross-validation where the sample data is divided into k equal size subsamples. Among these subsamples, k-1 subsamples are selected at random for training and the remaining subsample is used for validation. This procedure is repeated k times such that each subsample is used exactly once as a validation data. The results are aggregated to get the final prediction.

The below code demonstrates the usage of 5 fold cross validation to determine the best k value for a kNN model built on the default data set.



Here we stopped at the value of k = 11 and looks like it has the best average crossvalidation performance. It is, however, possible that a k value beyond 11 could give a better score, we may need to repeat the cross-validation process with higher values of k to determine this. However, we are not doing it in this example.

Having determined the best value of k using 5 fold cross-validation, we now use that value to train a model on the entire training data and now check the performance on train and test data as shown below



# Ensemble methods

Ensemble learning is a technique where instead of using a single learning algorithm for prediction, we use multiple learning algorithms in order to improve prediction accuracy.  For example, instead of using a single decision tree, we can use multiple decision tree models that are combined in a way to fit the training data.

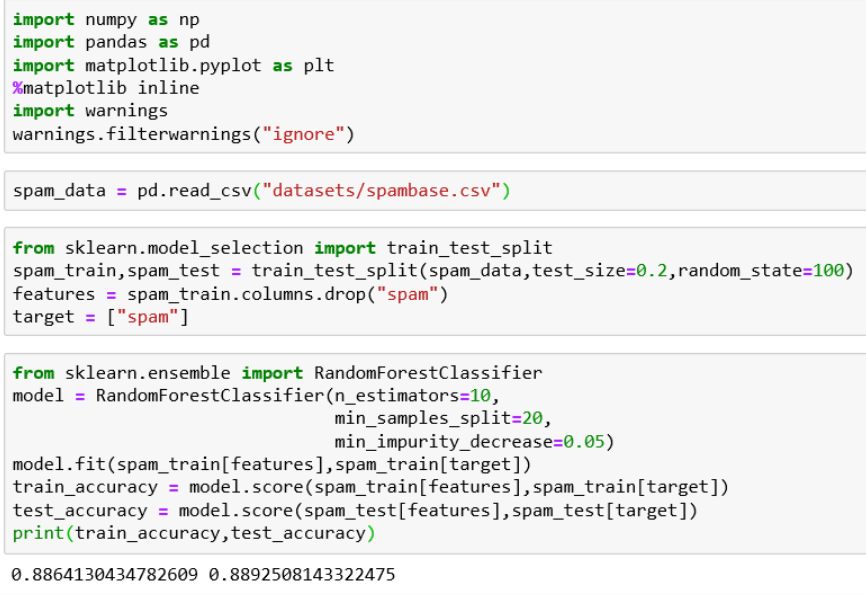
Two commonly used ensemble methods are **Bagging** and **Boosting**.

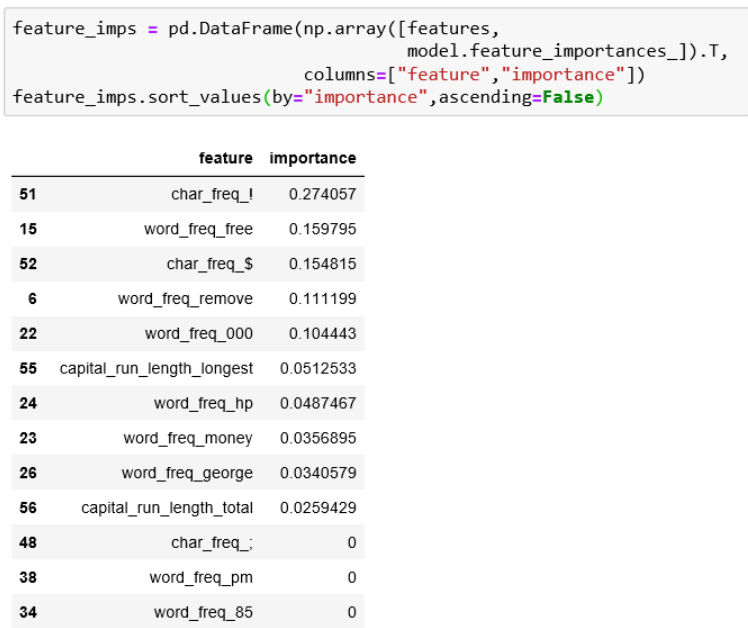
# Bagging

In Bagging, multiple models are trained using the same algorithm with a subset of training data selected (with replacement) to train each model. Once multiple models are trained in this manner they are aggregated using voting or simple aggregation methods such as averaging.

**Random Forest** which uses a random feature selection is a popular bagging model based on Decision tree algorithm. It learns a decision tree by taking sample of a subset of available features and a subset of training data at each decision node of the decision tree. Every tree thus learned which forms a forest, vote for the prediction.

In the below code we are training a random forest model with 10 decision trees.



One of the interesting features of a random forest model because it builds multiple trees is that it can evaluate which features are important. The below code demonstrates how to determine what features were important for the random forest model.

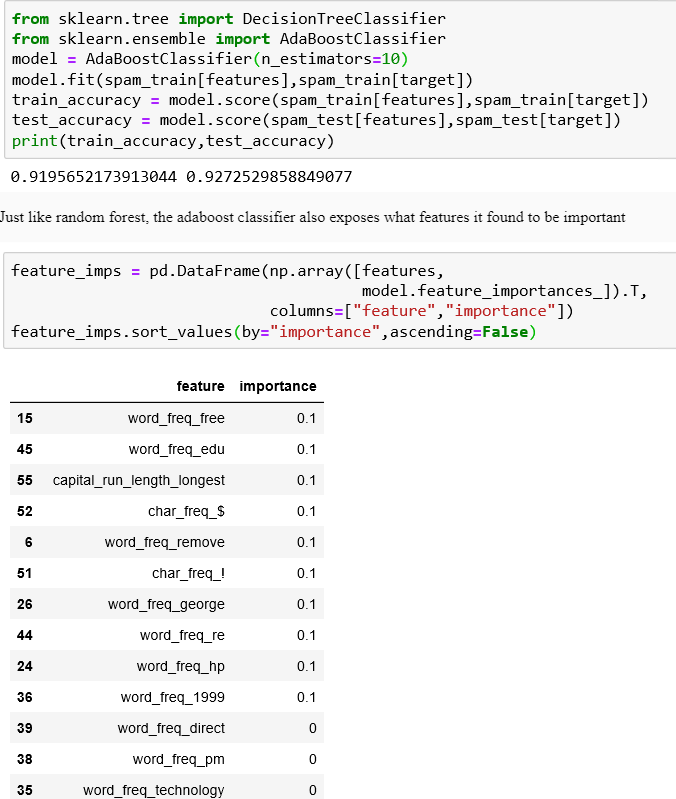
In the above code we are using the model.feature\_importances\_ value to determine the importance of each feature in the random forest model.

Among the 50+ features in the data, the random forest model found just 10 features to be useful.

# Boosting

Boosting techniques emphasize training data that were wrongly predicted in previous iterations. A model is considered weak if it has a substantial error rate. The key idea is to make the model stronger. The model is trained such that initially each training data has equal weight but after each iteration, the data that is wrongly classified will have its weight increased. This will force the model to focus on those hard data and make the model stronger. AdaBoost is one of the well known boosting techniques.

The below code shows how you can use **Adaboost** in sklearn.



# Clustering

So far we have seen various algorithms that help in supervised learning. Let us now briefly discuss unsupervised learning with an example of clustering.

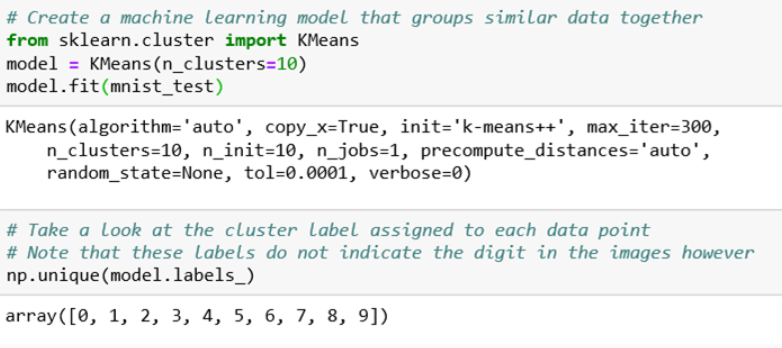
Recall that in unsupervised learning, we have historical data which contains no labels. Let us use an example of handwritten digits. The handwritten digits dataset contains pixel intensities of digits captured as 28x28 images. In this example, these images do not have any labels associated with them.

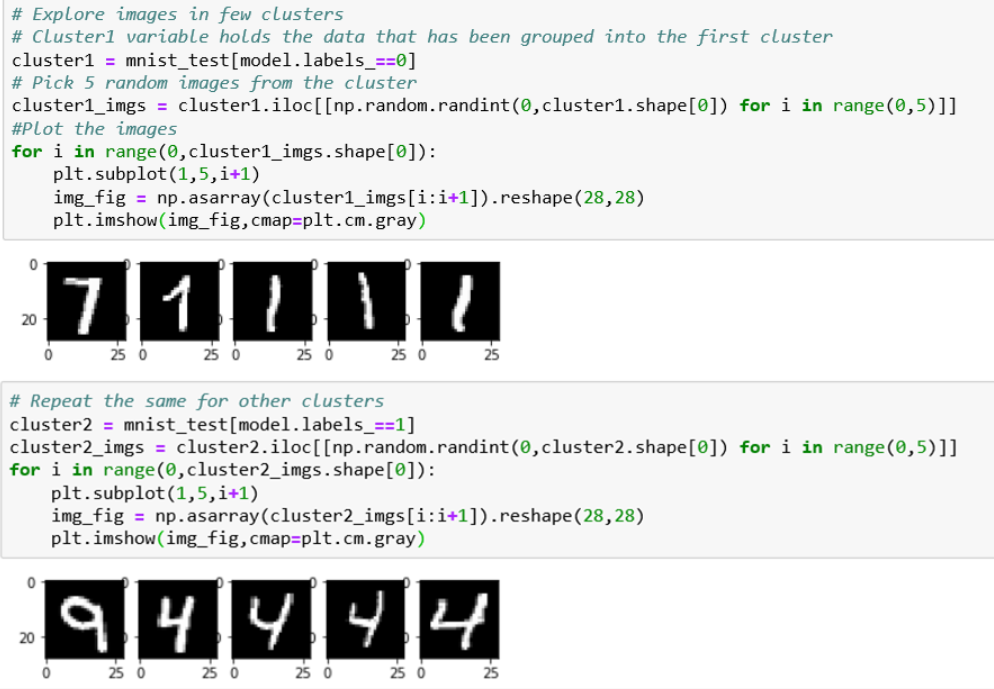
The below code helps you understand the data. What you can see here is the plot of the first data point in the set. in this case, it happens to be an image of the digit 2.

Let us now employ clustering to group similar-looking data points (digits). The aim of clustering is to bring similar records together and make sure that the members of different clusters are significantly different from each other.

There are several algorithms using which clustering can be performed. One of the widely used clustering algorithm is the K-means algorithm. The K-means algorithm works on numerical data and uses Euclidean distance to identify which data points are close to one another.

The following in the following code, we are running the K-means algorithm.





# Topical Clustering

Documents such as white papers, research papers, reports etc. are published in great quantities and there is a need to cluster and visualize them based on topics.

# Information Retrieval

Clustering can be used to group web sites or documents based on the contents, if there is a hit on a website/document then other websites/documents in that cluster are also likely to be relevant.

# Anomaly Detection

The aim of anamoly detection is to find objects that are significantly different from the rest. This is useful in detecting fraudulent transactions in finance and banking, intrusion detection, etc.