# Machine Learning

## Steps of Data Exploration and Preparation

Below are the steps involved to understand, clean and prepare your data for building your predictive model:

1. Variable Identification
2. Univariate Analysis
3. Bi-variate Analysis
4. Missing values treatment
5. Outlier treatment
6. Variable transformation
7. Variable creation

Finally, we will need to iterate over steps 4 – 7 multiple times before we come up with our refined model.

Let's now study each stage in detail:-

Variable Identification

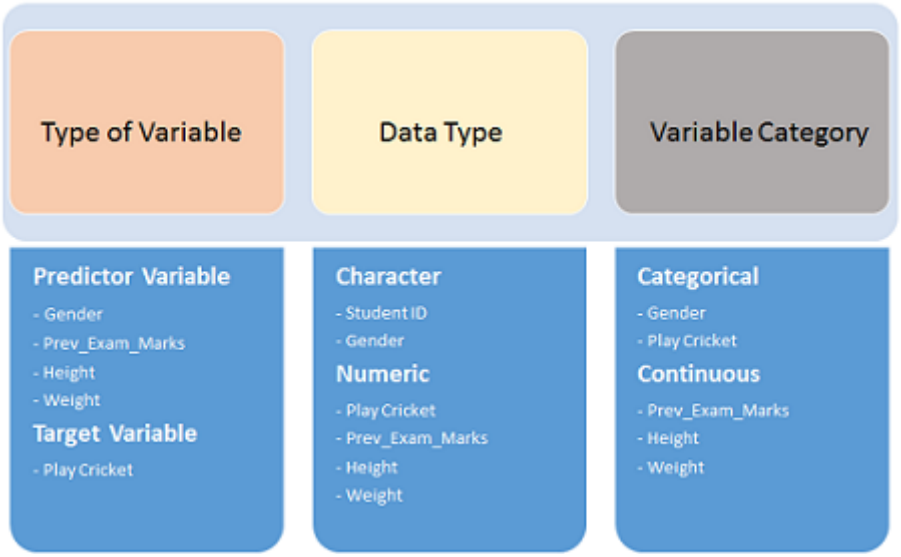
First, identify **Predictor** (Input) and **Target** (output) variables. Next, identify the data type and category of the variables.

Let's understand this step more clearly by taking an example.

Example:- Suppose, we want to predict, whether the students will play cricket or not (refer below data set). Here you need to identify predictor variables, target variable, data type of variables and category of variables.

| Student_ID | Gender | Prev_Exam_Marks | Height (cm) | Weight Caregory (kgs) | Play Cricket |
|---|---|---|---|---|---|
| S001 | M | 65 | 178 | 61 | 1 |
| S002 | F | 75 | 174 | 56 | 0 |
| S003 | M | 45 | 163 | 62 | 1 |
| S004 | M | 57 | 175 | 70 | 0 |
| S005 | F | 59 | 162 | 67 | 0 |

Below, the variables have been defined in different category:

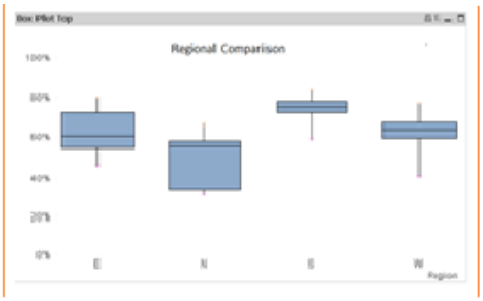


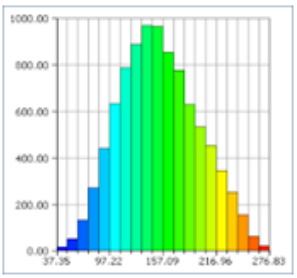

Univariate Analysis

At this stage, we explore variables one by one. Method to perform uni-variate analysis will depend on whether the variable type is categorical or continuous. Let's look at these methods and statistical measures for categorical and continuous variables individually:

**Continuous Variables:-** In case of continuous variables, we need to understand the central tendency and spread of the variable. These are measured using various statistical metrics visualization methods as shown below:

| Central Tendency | Measure of Dispersion | Visualization Methods |
|---|---|---|
| Mean | Range | Histogram |
| Median | Quartile | Box Plot |
| Mode | IQR | |
| Min | Variance | |
| Max | Standard Deviation | |
| | Skewness and Kurtosis | |



**Note:** Univariate analysis is also used to highlight missing and outlier values. In the upcoming part of this series, we will look at methods to handle missing and outlier values.

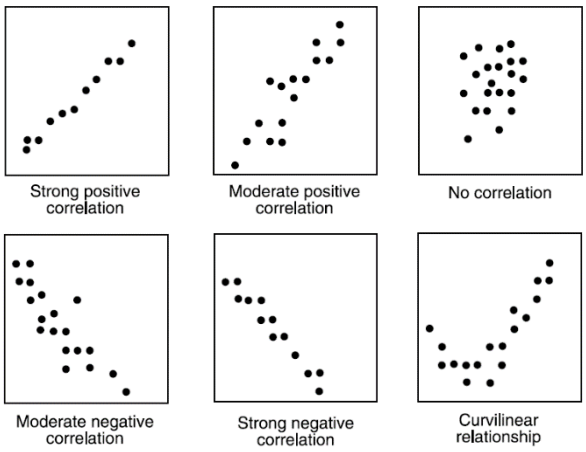
**Categorical Variables:** For categorical variables, we'll use frequency table to understand distribution of each category. We can also read as percentage of values under each category. It can be be measured using two metrics, **Count** and **Count%** against each category. Bar chart can be used as visualization.

Bi-variate Analysis

Bi-variate Analysis finds out the relationship between two variables. Here, we look for association and disassociation between variables at a pre-defined significance level. We can perform bi-variate analysis for any combination of categorical and continuous variables. The combination can be: Categorical & Categorical, Categorical & Continuous and Continuous & Continuous. Different methods are used to tackle these combinations during analysis process.

Let's understand the possible combinations in detail:

**Continuous & Continuous:** While doing bi-variate analysis between two continuous variables, we should look at scatter plot. It is a nifty way to find out the relationship between two variables. The pattern of scatter plot indicates the relationship between variables. The relationship can be linear or non-linear.




Scatter plot shows the relationship between two variable but does not indicates the strength of relationship amongst them. To find the strength of the relationship, we use Correlation. Correlation varies between -1 and +1.

- -1: perfect negative linear correlation
- +1:perfect positive linear correlation and
- 0: No correlation

Correlation can be derived using following formula:

**Correlation = Covariance(X,Y) / SQRT( Var(X)* Var(Y))**

Various tools have function or functionality to identify correlation between variables. In Excel, function CORREL() is used to return the correlation between two variables and SAS uses procedure PROC CORR to identify the correlation. These function returns Pearson Correlation value to identify the relationship between two variables:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **X** | 65 | 72 | 78 | 65 | 72 | 70 | 65 | 68 |
| **Y** | 72 | 69 | 79 | 69 | 84 | 75 | 60 | 73 |

| Metrics | Formula | Value |
|---|---|---|
| Co-Variance (X,Y) | =COVAR(E6:L6,E7:L7) | 18.77 |
| Variance (X) | =VAR.P(E6:L6) | 18.48 |
| Variance (Y) | =VAR.P(E7:L7) | 45.23 |
| Correlation | =G10/SQRT(G11*G12) | 0.65 |

In above example, we have good positive relationship(0.65) between two variables X and Y.

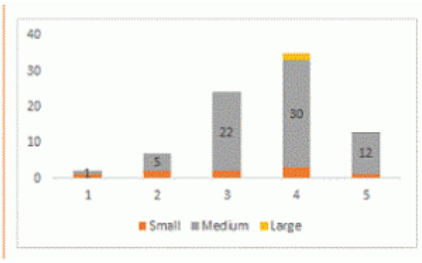
**Categorical & Categorical:** To find the relationship between two categorical variables, we can use following methods:

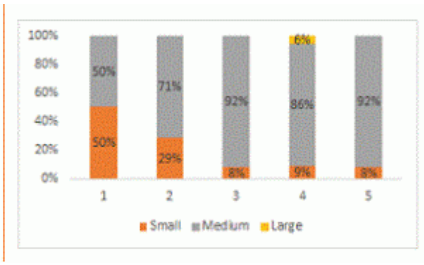
- **Two-way table:** We can start analyzing the relationship by creating a two-way table of count and count%. The rows represents the category of one variable and the columns represent the categories of the other variable. We show count or count% of observations available in each combination of row and column categories.
- **Stacked Column Chart:** This method is more of a visual form of Two-way table.

Product Category

| Frequency / Row Pct | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|
| Small | 1<br>11.11 | 2<br>22.22 | 2<br>22.22 | 3<br>33.33 | 1<br>11.11 | 9 |
| Medium | 1<br>1.43 | 5<br>7.14 | 22<br>31.43 | 30<br>42.86 | 12<br>17.14 | 70 |
| Large | 0<br>0.00 | 0<br>0.00 | 0<br>0.00 | 2<br>100.00 | 0<br>0.00 | 2 |
| Total | 2 | 7 | 24 | 35 | 13 | 81 |

Frequency Missing = 77




- **Chi-Square Test:** This test is used to derive the statistical significance of relationship between the variables. Also, it tests whether the evidence in the sample is strong enough to generalize that the relationship for a larger population as well. Chi-square is based on the difference between the expected and observed frequencies in one or more categories in the two-way table. It returns probability for the computed chi-square distribution with the degree of freedom.

Probability of 0: It indicates that both categorical variable are dependent

Probability of 1: It shows that both variables are independent.

Probability less than 0.05: It indicates that the relationship between the variables is significant at 95% confidence. The chi-square test statistic for a test of independence of two categorical variables is found by:

$X^2 = \sum (O - E)^2 / E$ where *O* represents the observed frequency. *E* is the expected frequency under the null hypothesis and computed by:

$$E = \frac{row\ total \times column\ total}{sample\ size}$$

From previous two-way table, the expected count for product category 1 to be of small size is 0.22. It is derived by taking the row total for Size (9) times the column total for Product category (2) then dividing by the sample size (81). This is procedure is conducted for each cell. Statistical Measures used to analyze the power of relationship are:

- Cramer's V for Nominal Categorical Variable
- Mantel-Haenszed Chi-Square for ordinal categorical variable.

Different data science language and tools have specific methods to perform chi-square test. In SAS, we can use **Chisq** as an option with **Proc freq** to perform this test.

**Categorical & Continuous:** While exploring relation between categorical and continuous variables, we can draw box plots for each level of categorical variables. If levels are small in number, it will not show the statistical significance. To look at the statistical significance we can perform Z-test, T-test or ANOVA.

- **Z-Test/ T-Test:-** Either test assess whether mean of two groups are statistically different from

$$z = \frac{\left|\bar{x}_1 - \bar{x}_2\right|}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

each other or not. If the probability of Z is small then the difference of two averages is more significant. The T-test is very similar to Z-test but it is used when number of observation for both categories is less than 30.

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{S^2\left(\frac{1}{N_1} + \frac{1}{N_2}\right)}}$$

$$S^2 = \frac{(N_1 - 1)S_1^2 + (N_2 - 1)S_2^2}{N_1 + N_2 - 2}$$

where:

- $\bar{X}_1, \bar{X}_2$: *Averages*
- $S_1^2, S_2^2$: *Variances*
- $N_1, N_2$: *Counts*
- $t$ : has t distribution with $N_1 + N_2 - 2$ degree of freedom

- **ANOVA:-** It assesses whether the average of more than two groups is statistically different.

**Example:** Suppose, we want to test the effect of five different exercises. For this, we recruit 20 men and assign one type of exercise to 4 men (5 groups). Their weights are recorded after a few weeks. We need to find out whether the effect of these exercises on them is significantly different or not. This can be done by comparing the weights of the 5 groups of 4 men each.

Till here, we have understood the first three stages of Data Exploration, Variable Identification, Uni-Variate and Bi-Variate analysis. We also looked at various statistical and visual methods to identify the relationship between variables.

Now, we will look at the methods of Missing values Treatment. More importantly, we will also look at why missing values occur in our data and why treating them is necessary.

2. Missing Value Treatment

Why missing values treatment is required?

Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction or classification.

| Name | Weight | Gender | Play Cricket/ Not |
|---|---|---|---|
| Mr. Amit | 58 | M | Y |
| Mr. Anil | 61 | M | Y |
| Miss Swati | 58 | F | N |
| Miss Richa | 55 | | Y |
| Mr. Steve | 55 | M | N |
| Miss Reena | 64 | F | Y |
| Miss Rashmi | 57 | | Y |
| Mr. Kunal | 57 | M | N |

| Gender | #Students | #Play Cricket | %Play Cricket |
|---|---|---|---|
| F | 2 | 1 | 50% |
| M | 4 | 2 | 50% |
| Missing | 2 | 2 | 100% |

| Name | Weight | Gender | Play Cricket/ Not |
|---|---|---|---|
| Mr. Amit | 58 | M | Y |
| Mr. Anil | 61 | M | Y |
| Miss Swati | 58 | F | N |
| Miss Richa | 55 | F | Y |
| Mr. Steve | 55 | M | N |
| Miss Reena | 64 | F | Y |
| Miss Rashmi | 57 | F | Y |
| Mr. Kunal | 57 | M | N |

| Gender | #Students | #Play Cricket | %Play Cricket |
|---|---|---|---|
| F | 4 | 3 | 75% |
| M | 4 | 2 | 50% |

Notice the missing values in the image shown above: In the left scenario, we have not treated missing values. The inference from this data set is that the chances of playing cricket by males is higher than females. On the other hand, if you look at the second table, which shows data after treatment of missing values (based on gender), we can see that females have higher chances of playing cricket compared to males.

Why my data has missing values?

We looked at the importance of treatment of missing values in a dataset. Now, let's identify the reasons for occurrence of these missing values. They may occur at two stages:

1. **Data Extraction**: It is possible that there are problems with extraction process. In such cases, we should double-check for correct data with data guardians. Some hashing procedures can also be used to make sure data extraction is correct. Errors at data extraction stage are typically easy to find and can be corrected easily as well.
2. **Data collection**: These errors occur at time of data collection and are harder to correct. They can be categorized in four types:
    - **Missing completely at random:** This is a case when the probability of missing variable is same for all observations. For example: respondents of data collection process decide that they will declare their earning after tossing a fair coin. If an head occurs, respondent declares his / her earnings & vice versa. Here each observation has equal chance of missing value.
    - **Missing at random:** This is a case when variable is missing at random and missing ratio varies for different values / level of other input variables. For example: We are collecting data for age and female has higher missing value compare to male.
    - **Missing that depends on unobserved predictors:** This is a case when the missing values are not random and are related to the unobserved input variable. For example: In a medical study, if a particular diagnostic causes discomfort, then there is higher chance of drop out from the study. This missing value is not at random unless we have included "discomfort" as an input variable for all patients.
    - **Missing that depends on the missing value itself:** This is a case when the probability of missing value is directly correlated with missing value itself. For example: People with higher or lower income are likely to provide non-response to their earning.

Which are the methods to treat missing values ?

1. **Deletion:** It is of two types: List Wise Deletion and Pair Wise Deletion.

- In list wise deletion, we delete observations where any of the variable is missing. Simplicity is one of the major advantage of this method, but this method reduces the power of model because it reduces the sample size.
- In pair wise deletion, we perform analysis with all cases in which the variables of interest are present. Advantage of this method is, it keeps as many cases available for analysis. One of the disadvantage of this method, it uses different sample size for different variables.

**List wise deletion**

| Gender | Manpower | Sales |
|---|---|---|
| M | 25 | 343 |
| F | . | 280 |
| M | 33 | 332 |
| M | . | 272 |
| F | 25 | . |
| M | 29 | 326 |
| | 26 | 259 |
| M | 32 | 297 |

**Pair wise deletion**

| Gender | Manpower | Sales |
|---|---|---|
| M | 25 | 343 |
| F | . | 280 |
| M | 33 | 332 |
| M | . | 272 |
| F | 25 | . |
| M | 29 | 326 |
| | 26 | 259 |
| M | 32 | 297 |

- Deletion methods are used when the nature of missing data is "**Missing completely at random**" else non random missing values can bias the model output.

2. **Mean/ Mode/ Median Imputation:** Imputation is a method to fill in the missing values with estimated ones. The objective is to employ known relationships that can be identified in the valid values of the data set to assist in estimating the missing values. Mean / Mode / Median imputation is one of the most frequently used methods. It consists of replacing the missing data for a given attribute by the mean or median (quantitative attribute) or mode (qualitative attribute) of all known values of that variable. It can be of two types:-
   - **Generalized Imputation:** In this case, we calculate the mean or median for all non missing values of that variable then replace missing value with mean or median. Like in above table, variable "**Manpower"** is missing so we take average of all non missing values of "**Manpower"** (**28.33**) and then replace missing value with it.
   - **Similar case Imputation:** In this case, we calculate average for gender "**Male"** (29.75) and "**Female**" (25) individually of non missing values then replace the missing value based on gender. For "**Male**", we will replace missing values of manpower with 29.75 and for "**Female**" with 25.

3. **Prediction Model**: Prediction model is one of the sophisticated method for handling missing data. Here, we create a predictive model to estimate values that will substitute the missing data. In this case, we divide our data set into two sets: One set with no missing values for the variable and another one with missing values. First data set become training data set of the model while second data set with missing values is test data set and variable with missing values is treated as target variable. Next, we create a model to predict target variable based on other attributes of the training data set and populate missing values of test data set.We can use regression, ANOVA, Logistic regression and various modeling technique to perform this. There are 2 drawbacks for this approach:
   1. The model estimated values are usually more well-behaved than the true values

2. If there are no relationships with attributes in the data set and the attribute with missing values, then the model will not be precise for estimating missing values.

4. **KNN Imputation:** In this method of imputation, the missing values of an attribute are imputed using the given number of attributes that are most similar to the attribute whose values are missing. The similarity of two attributes is determined using a distance function. It is also known to have certain advantage & disadvantages.
    - **Advantages:**
        - k-nearest neighbour can predict both qualitative & quantitative attributes
        - Creation of predictive model for each attribute with missing data is not required
        - Attributes with multiple missing values can be easily treated
        - Correlation structure of the data is taken into consideration
    - **Disadvantage:**
        - KNN algorithm is very time-consuming in analyzing large database. It searches through all the dataset looking for the most similar instances.
        - Choice of k-value is very critical. Higher value of k would include attributes which are significantly different from what we need whereas lower value of k implies missing out of significant attributes.

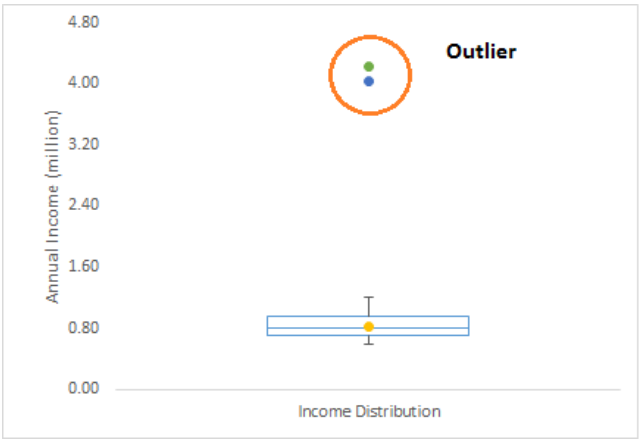
After dealing with missing values, the next task is to deal with outliers. Often, we tend to neglect outliers while building models. This is a discouraging practice. Outliers tend to make your data skewed and reduces accuracy. Let's learn more about outlier treatment.

3. Techniques of Outlier Detection and Treatment

What is an Outlier?

Outlier is a commonly used terminology by analysts and data scientists as it needs close attention else it can result in wildly wrong estimations. Simply speaking, Outlier is an observation that appears far away and diverges from an overall pattern in a sample.

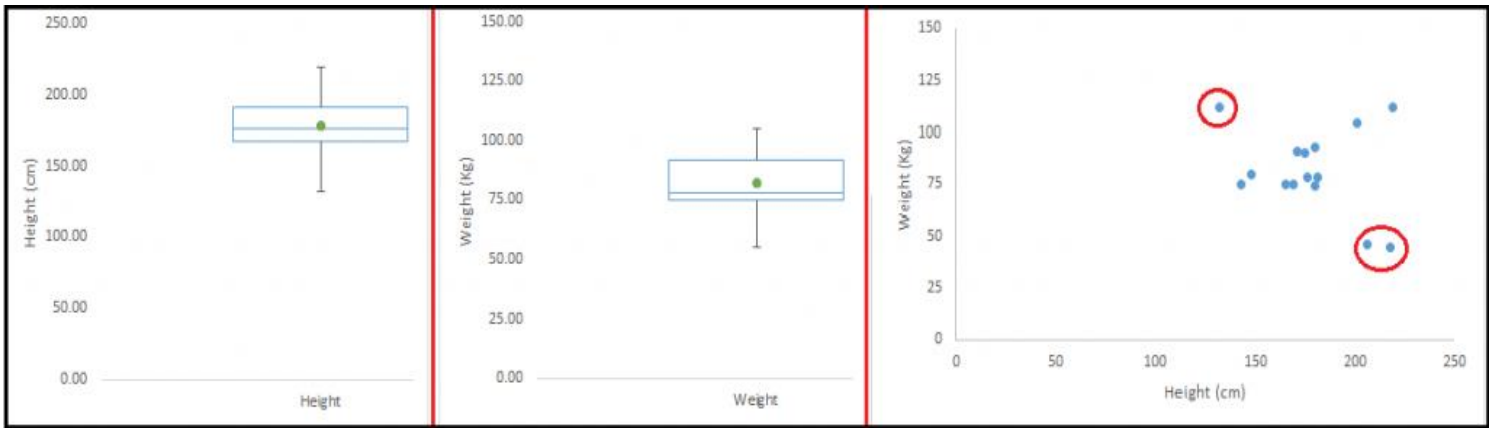
Let's take an example, we do customer profiling and find out that the average annual income of customers is $0.8 million. But, there are two customers having annual income of $4 and $4.2 million. These two customers annual income is much higher than rest of the population. These two observations will be seen as Outliers.



What are the types of Outliers?

Outlier can be of two types: **Univariate** and **Multivariate**. Above, we have discussed the example of univariate outlier. These outliers can be found when we look at distribution of a single variable. Multi-variate outliers are outliers in an n-dimensional space. In order to find them, you have to look at distributions in multi-dimensions.

Let us understand this with an example. Let us say we are understanding the relationship between height and weight. Below, we have univariate and bivariate distribution for Height, Weight. Take a look at the box plot. We do not have any outlier (above and below 1.5*IQR, most common method). Now look at the scatter plot. Here, we have two values below and one above the average in a specific segment of weight and height.




What causes Outliers?

Whenever we come across outliers, the ideal way to tackle them is to find out the reason of having these outliers. The method to deal with them would then depend on the reason of their occurrence. Causes of outliers can be classified in two broad categories:

1. **Artificial (Error) / Non-natural**
2. **Natural**.

Let's understand various types of outliers in more detail:

- **Data Entry Errors:-** Human errors such as errors caused during data collection, recording, or entry can cause outliers in data. For example: Annual income of a customer is $100,000. Accidentally, the data entry operator puts an additional zero in the figure. Now the income becomes $1,000,000 which is 10 times higher. Evidently, this will be the outlier value when compared with rest of the population.
- **Measurement Error:** It is the most common source of outliers. This is caused when the measurement instrument used turns out to be faulty. For example: There are 10 weighing machines. 9 of them are correct, 1 is faulty. Weight measured by people on the faulty machine will be higher / lower than the rest of people in the group. The weights measured on faulty machine can lead to outliers.
- **Experimental Error:** Another cause of outliers is experimental error. For example: In a 100m sprint of 7 runners, one runner missed out on concentrating on the 'Go' call which caused him

to start late. Hence, this caused the runner's run time to be more than other runners. His total run time can be an outlier.

- **Intentional Outlier*:*** This is commonly found in self-reported measures that involves sensitive data. For example: Teens would typically under report the amount of alcohol that they consume. Only a fraction of them would report actual value. Here actual values might look like outliers because rest of the teens are under reporting the consumption.
- **Data Processing Error:** Whenever we perform data mining, we extract data from multiple sources. It is possible that some manipulation or extraction errors may lead to outliers in the dataset.
- **Sampling error:** For instance, we have to measure the height of athletes. By mistake, we include a few basketball players in the sample. This inclusion is likely to cause outliers in the dataset.
- **Natural Outlier:** When an outlier is not artificial (due to error), it is a natural outlier. For instance: In my last assignment with one of the renowned insurance company, I noticed that the performance of top 50 financial advisors was far higher than rest of the population. Surprisingly, it was not due to any error. Hence, whenever we perform any data mining activity with advisors, we used to treat this segment separately.

What is the impact of Outliers on a dataset?

Outliers can drastically change the results of the data analysis and statistical modeling. There are numerous unfavourable impacts of outliers in the data set:

- It increases the error variance and reduces the power of statistical tests
- If the outliers are non-randomly distributed, they can decrease normality
- They can bias or influence estimates that may be of substantive interest
- They can also impact the basic assumption of Regression, ANOVA and other statistical model assumptions.

To understand the impact deeply, let's take an example to check what happens to a data set with and without outliers in the data set.

**Example:**

| Without Outlier | With Outlier |
|---|---|
| 4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7 | 4, 4, 5, 5, 5, 5, 6, 6, 6, 7, 7,300 |
| Mean = 5.45 | Mean = 30.00 |
| Median = 5.00 | Median = 5.50 |
| Mode = 5.00 | Mode = 5.00 |
| Standard Deviation = 1.04 | Standard Deviation = 85.03 |

As you can see, data set with outliers has significantly different mean and standard deviation. In the first scenario, we will say that average is 5.45. But with the outlier, average soars to 30. This would change the estimate completely.

How to detect Outliers?

Most commonly used method to detect outliers is visualization. We use various visualization methods, like **Box-plot**, **Histogram**, **Scatter Plot** (above, we have used box plot and scatter plot for visualization). Some analysts also various thumb rules to detect outliers. Some of them are:

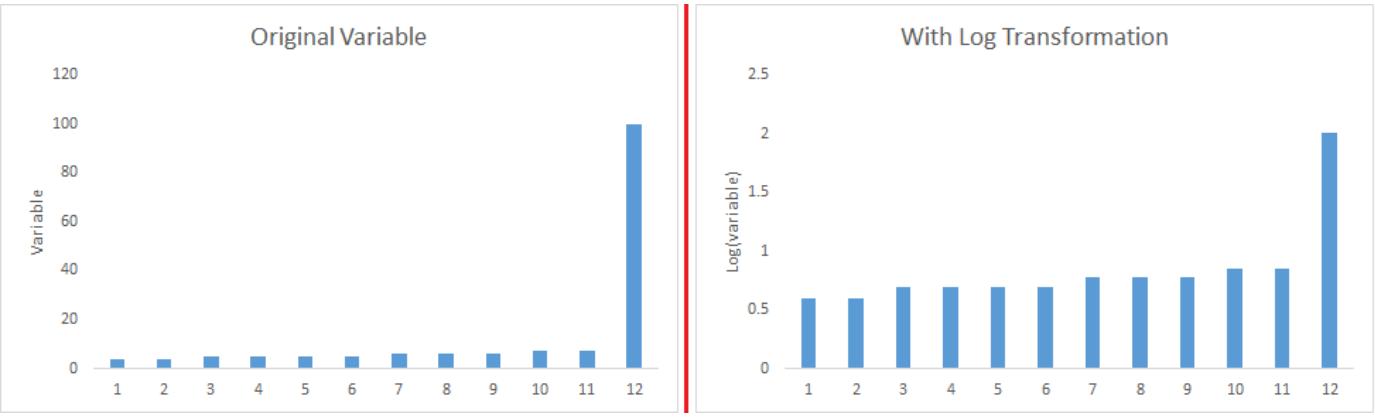
- Any value, which is beyond the range of -1.5 x IQR to 1.5 x IQR
- Use capping methods. Any value which out of range of 5th and 95th percentile can be considered as outlier
- Data points, three or more standard deviation away from mean are considered outlier
- Outlier detection is merely a special case of the examination of data for influential data points and it also depends on the business understanding
- Bivariate and multivariate outliers are typically measured using either an index of influence or leverage, or distance. Popular indices such as Mahalanobis' distance and Cook's *D* are frequently used to detect outliers.
- In SAS, we can use PROC Univariate, PROC SGPLOT. To identify outliers and influential observation, we also look at statistical measure like STUDENT, COOKD, RSTUDENT and others.

How to remove Outliers?

Most of the ways to deal with outliers are similar to the methods of missing values like deleting observations, transforming them, binning them, treat them as a separate group, imputing values and other statistical methods. Here, we will discuss the common techniques used to deal with outliers:

**Deleting observations:** We delete outlier values if it is due to data entry error, data processing error or outlier observations are very small in numbers. We can also use trimming at both ends to remove outliers.

**Transforming and binning values:** Transforming variables can also eliminate outliers. Natural log of a value reduces the variation caused by extreme values. Binning is also a form of variable transformation. Decision Tree algorithm allows to deal with outliers well due to binning of variable. We can also use the process of assigning weights to different observations.



**Imputing:** Like imputation of missing values, we can also impute outliers. We can use mean, median, mode imputation methods. Before imputing values, we should analyse if it is natural outlier or artificial. If it is artificial, we can go with imputing values. We can also use statistical model to predict values of outlier observation and after that we can impute it with predicted values.

**Treat separately:** If there are significant number of outliers, we should treat them separately in the statistical model. One of the approach is to treat both groups as two different groups and build individual model for both groups and then combine the output.

Till here, we have learnt about steps of data exploration, missing value treatment and techniques of outlier detection and treatment. These 3 stages will make your raw data better in terms of information availability and accuracy. Let's now proceed to the final stage of data exploration. It is Feature Engineering.

4. The Art of Feature Engineering

What is Feature Engineering?

Feature engineering is the science (and art) of extracting more information from existing data. You are not adding any new data here, but you are actually making the data you already have more useful.

For example, let's say you are trying to predict foot fall in a shopping mall based on dates. If you try and use the dates directly, you may not be able to extract meaningful insights from the data. This is because the foot fall is less affected by the day of the month than it is by the day of the week. Now this information about day of week is implicit in your data. You need to bring it out to make your model better.

This exercising of bringing out information from data in known as feature engineering.

What is the process of Feature Engineering ?

You perform feature engineering once you have completed the first 5 steps in data exploration – Variable Identification, Univariate, Bivariate Analysis, Missing Values Imputation and Outliers Treatment. Feature engineering itself can be divided in 2 steps:

- Variable transformation.
- Variable / Feature creation.

These two techniques are vital in data exploration and have a remarkable impact on the power of prediction. Let's understand each of this step in more details.

What is Variable Transformation?

In data modelling, transformation refers to the replacement of a variable by a function. For instance, replacing a variable x by the square / cube root or logarithm x is a transformation. In other words, transformation is a process that changes the distribution or relationship of a variable with others.

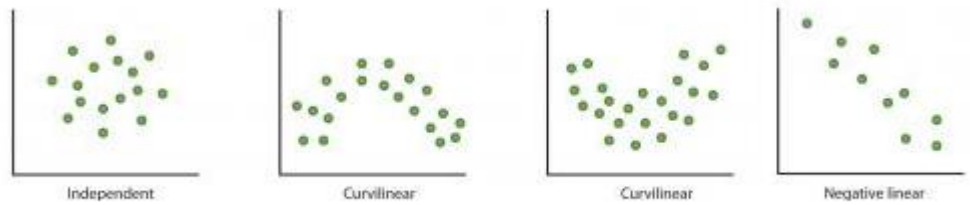
Let's look at the situations when variable transformation is useful.

When should we use Variable Transformation?

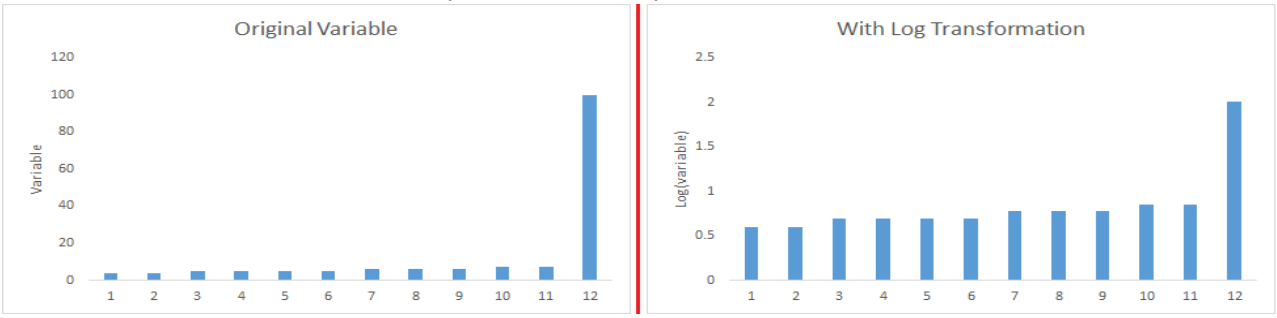
Below are the situations where variable transformation is a requisite:

- When we want to **change the scale** of a variable or standardize the values of a variable for better understanding. While this transformation is a must if you have data in different scales, this transformation does not change the shape of the variable distribution

- When we can **transform complex non-linear relationships into linear relationships**. Existence of a linear relationship between variables is easier to comprehend compared to a non-linear or curved relation. Transformation helps us to convert a non-linear relation into linear relation. Scatter plot can be used to find the relationship between two continuous variables. These transformations also improve the prediction. Log transformation is one of the commonly used transformation technique used in these situations.




- **Symmetric distribution is preferred over skewed distribution** as it is easier to interpret and generate inferences. Some modeling techniques requires normal distribution of variables. So, whenever we have a skewed distribution, we can use transformations which reduce skewness. For right skewed distribution, we take square / cube root or logarithm of variable and for left skewed, we take square / cube or exponential of variables.




- Variable Transformation is also done from an **implementation point of view** (Human involvement). Let's understand it more clearly. In one of my project on employee performance, I found that age has direct correlation with performance of the employee i.e. higher the age, better the performance. From an implementation stand point, launching age based progamme might present implementation challenge. However, categorizing the sales agents in three age group buckets of <30 years, 30-45 years and >45 and then formulating three different strategies for each group is a judicious approach. This categorization technique is known as Binning of Variables.

What are the common methods of Variable Transformation?

There are various methods used to transform variables. As discussed, some of them include square root, cube root, logarithmic, binning, reciprocal and many others. Let's look at these methods in detail by highlighting the pros and cons of these transformation methods.

- **Logarithm:** Log of a variable is a common transformation method used to change the shape of distribution of the variable on a distribution plot. It is generally used for reducing right skewness of variables. Though, It can't be applied to zero or negative values as well.

- **Square / Cube root:** The square and cube root of a variable has a sound effect on variable distribution. However, it is not as significant as logarithmic transformation. Cube root has its own advantage. It can be applied to negative values including zero. Square root can be applied to positive values including zero.

- **Binning:** It is used to categorize variables. It is performed on original values, percentile or frequency. Decision of categorization technique is based on business understanding. For example, we can categorize income in three categories, namely: High, Average and Low. We can also perform co-variate binning which depends on the value of more than one variables.

What is Feature / Variable Creation & its Benefits?

Feature / Variable creation is a process to generate a new variables / features based on existing variable(s). For example, say, we have date(dd-mm-yy) as an input variable in a data set. We can generate new variables like day, month, year, week, weekday that may have better relationship with target variable. This step is used to highlight the hidden relationship in a variable:

| Emp_Code | Gender | Date | New_Day | New_Month | New_Year |
|---|---|---|---|---|---|
| A001 | Male | 21-Sep-11 | 21 | 9 | 2011 |
| A002 | Female | 27-Feb-13 | 27 | 2 | 2013 |
| A003 | Female | 14-Nov-12 | 14 | 11 | 2012 |
| A004 | Male | 07-Apr-13 | 7 | 4 | 2013 |
| A005 | Female | 21-Jan-11 | 21 | 1 | 2011 |
| A006 | Male | 26-Apr-13 | 26 | 4 | 2013 |
| A007 | Male | 15-Mar-12 | 15 | 3 | 2012 |

There are various techniques to create new features. Let's look at the some of the commonly used methods:

- **Creating derived variables:** This refers to creating new variables from existing variable(s) using set of functions or different methods. Let's look at it through "**Titanic – Kaggle competition**". In this data set, variable age has missing values. To predict missing values, we used the salutation (Master, Mr, Miss, Mrs) of name as a new variable. How do we decide which variable to create? Honestly, this depends on business understanding of the analyst, his curiosity and the set of hypothesis he might have about the problem. Methods such as taking log of variables, binning variables and other methods of variable transformation can also be used to create new variables.

- **Creating dummy variables:** One of the most common application of dummy variable is to convert categorical variable into numerical variables. Dummy variables are also called Indicator Variables. It is useful to take categorical variable as a predictor in statistical models. Categorical variable can take values 0 and 1. Let's take a variable 'gender'. We can produce two variables, namely, "**Var_Male**" with values 1 (Male) and 0 (No male) and "**Var_Female**" with values 1 (Female) and 0 (No Female). We can also create dummy variables for more than two classes of a categorical variables with n or n-1 dummy variables.

| Emp_Code | Gender | Var_Male | Var_Female |
|---|---|---|---|
| A001 | Male | 1 | 0 |
| A002 | Female | 0 | 1 |
| A003 | Female | 0 | 1 |
| A004 | Male | 1 | 0 |
| A005 | Female | 0 | 1 |
| A006 | Male | 1 | 0 |
| A007 | Male | 1 | 0 |

# Machine Learning Algorithms

Broadly, there are 3 types of Machine Learning Algorithms

1. Supervised Learning

**How it works:** This algorithm consist of a target / outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.

2. Unsupervised Learning

**How it works:** In this algorithm, we do not have any target or outcome variable to predict / estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: Apriori algorithm, K-means.

3. Reinforcement Learning:

**How it works:** Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process

### List of Common Machine Learning Algorithms

Here is the list of commonly used machine learning algorithms. These algorithms can be applied to almost any data problem:

### 1. Linear Regression

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a *X + b.

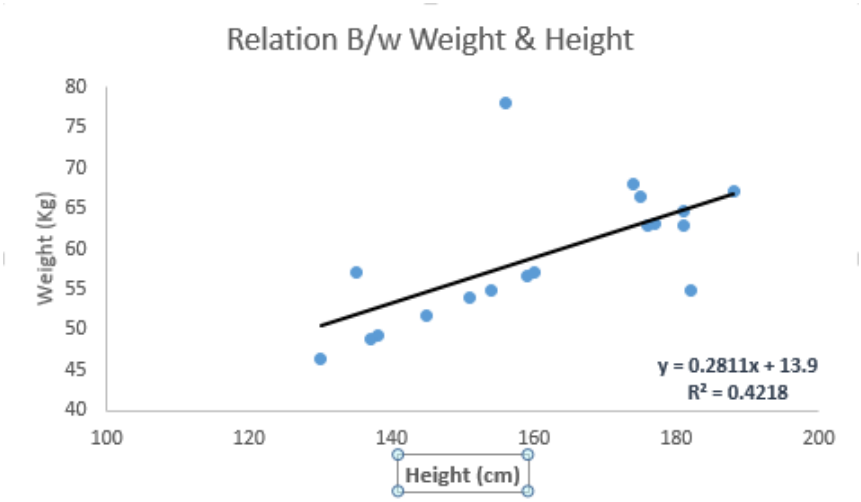
The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above.

In this equation:

- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.

Look at the below example. Here we have identified the best fit line having linear equation **y=0.2811x+13.9**. Now using this equation, we can find the weight, knowing the height of a person.




Linear Regression is of mainly two types: Simple Linear Regression and Multiple Linear Regression. Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression(as the name suggests) is characterized by multiple (more than 1) independent variables. While finding best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression.

**Python Code**

```
#Import Library

#Import other necessary libraries like pandas, numpy...

from sklearn import linear_model

#Load Train and Test datasets

#Identify feature and response variable(s) and values must be numeric and numpy arrays

x_train=input_variables_values_training_datasets

y_train=target_variables_values_training_datasets

x_test=input_variables_values_test_datasets

# Create linear regression object

linear = linear_model.LinearRegression()
```

```
# Train the model using the training sets and check score

linear.fit(x_train, y_train)

linear.score(x_train, y_train)

#Equation coefficient and Intercept

print('Coefficient: \n', linear.coef_)

print('Intercept: \n', linear.intercept_)

#Predict Output

predicted= linear.predict(x_test)
```

**R Code**

```
#Load Train and Test datasets

#Identify feature and response variable(s) and values must be numeric and numpy arrays

x_train <- input_variables_values_training_datasets

y_train <- target_variables_values_training_datasets

x_test <- input_variables_values_test_datasets

x <- cbind(x_train,y_train)

# Train the model using the training sets and check score

linear <- lm(y_train ~ ., data = x)

summary(linear)

#Predict Output

predicted= predict(linear,x_test)
```

**2. Logistic Regression**

Don't get confused by its name! It is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false ) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as **logit regression**. Since, it predicts the probability, its output values lies between 0 and 1 (as expected).

Again, let us try and understand this through a simple example.

Let's say your friend gives you a puzzle to solve. There are only 2 outcome scenarios – either you solve it or you don't. Now imagine, that you are being given wide range of puzzles / quizzes in an attempt to understand which subjects you are good at. The outcome to this study would be something like this – if you are given a trignometry based tenth grade problem, you are 70% likely to solve it. On the other hand, if it is grade fifth history question, the probability of getting an answer is only 30%. This is what Logistic Regression provides you.

Coming to the math, the log odds of the outcome is modeled as a linear combination of the predictor variables.

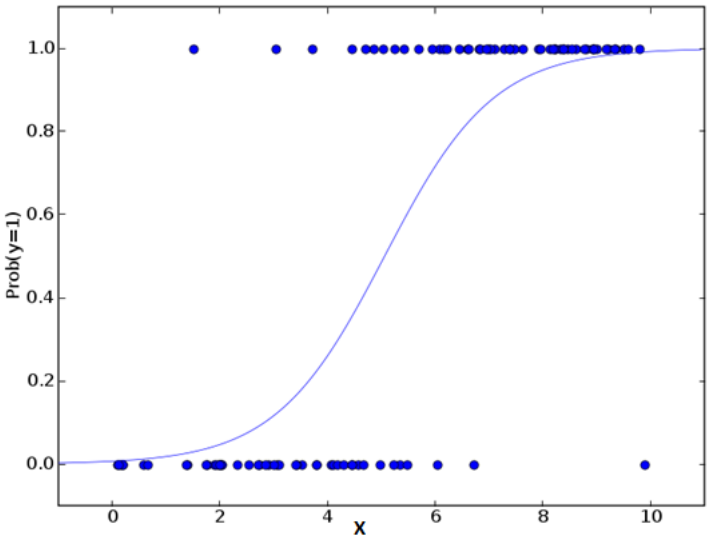
odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence

ln(odds) = ln(p/(1-p))

logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk

Above, p is the probability of presence of the characteristic of interest. It chooses parameters that maximize the likelihood of observing the sample values rather than that minimize the sum of squared errors (like in ordinary regression).

Now, you may ask, why take a log? For the sake of simplicity, let's just say that this is one of the best mathematical way to replicate a step function. I can go in more details, but that will beat the purpose of this article.



**Python Code**

```
#Import Library

from sklearn.linear_model import LogisticRegression

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dat
aset

# Create logistic regression object

model = LogisticRegression()

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Equation coefficient and Intercept

print('Coefficient: \n', model.coef_)

print('Intercept: \n', model.intercept_)

#Predict Output

predicted= model.predict(x_test)
```

**R Code**

```
x <- cbind(x_train,y_train)

# Train the model using the training sets and check score

logistic <- glm(y_train ~ ., data = x,family='binomial')

summary(logistic)

#Predict Output

predicted= predict(logistic,x_test)
```

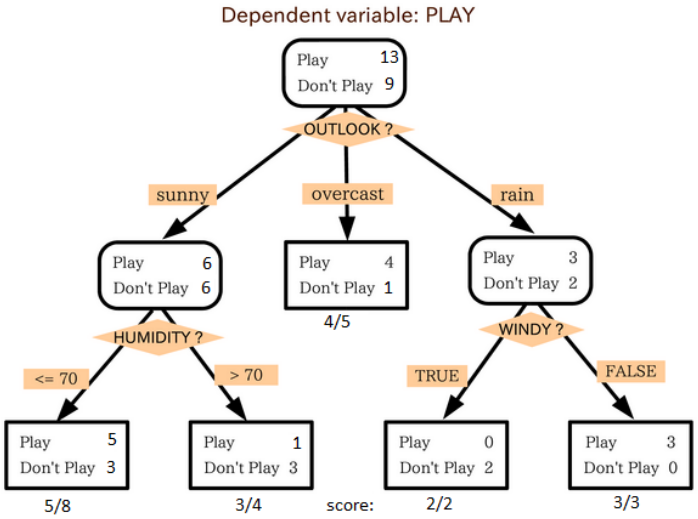
Furthermore..

There are many different steps that could be tried in order to improve the model:

- including interaction terms
- removing features
- regularization techniques
- using a non-linear model

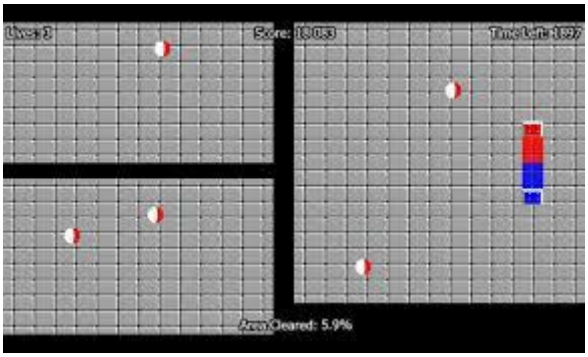
### 3. Decision Tree

This is one of my favorite algorithm and I use it quite frequently. It is a type of supervised learning algorithm that is mostly used for classification problems. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible. For more details, you can read: Decision Tree Simplified.




In the image above, you can see that population is classified into four different groups based on multiple attributes to identify 'if they will play or not'. To split the population into different heterogeneous groups, it uses various techniques like Gini, Information Gain, Chi-square, entropy.

The best way to understand how decision tree works, is to play Jezzball – a classic game from Microsoft (image below). Essentially, you have a room with moving walls and you need to create walls such that maximum area gets cleared off with out the balls.



So, every time you split the room with a wall, you are trying to create 2 different populations with in the same room. Decision trees work in very similar fashion by dividing a population in as different groups as possible.

***Python Code***

```
#Import Library

#Import other necessary libraries like pandas, numpy...

from sklearn import tree

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dat
aset

# Create tree object

model = tree.DecisionTreeClassifier(criterion='gini') # for classification, here you can change the algori
thm as gini or entropy (information gain) by default it is gini

# model = tree.DecisionTreeRegressor() for regression

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

**R Code**

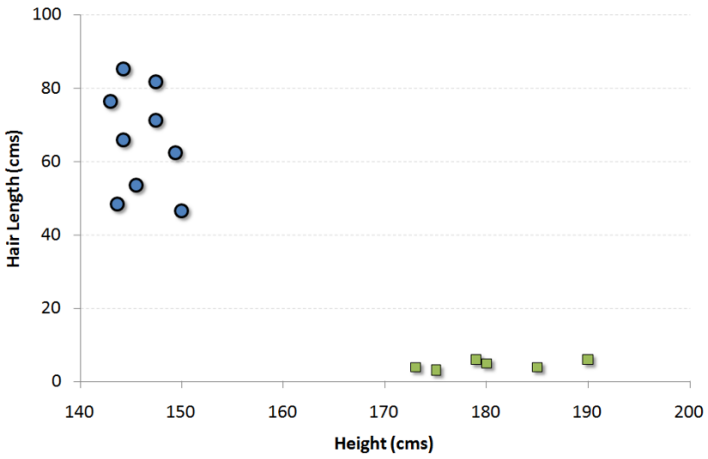
```
library(rpart)

x <- cbind(x_train,y_train)

# grow tree

fit <- rpart(y_train ~ ., data = x,method="class")

summary(fit)
```

```
#Predict Output

predicted= predict(fit,x_test)
```

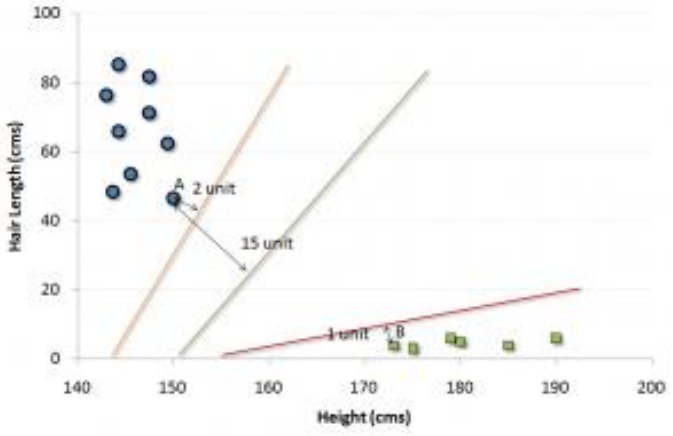
**4. SVM (Support Vector Machine)**

It is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

For example, if we only had two features like Height and Hair length of an individual, we'd first plot these two variables in two dimensional space where each point has two co-ordinates (these co-ordinates are known as **Support Vectors**)




Now, we will find some *line* that splits the data between the two differently classified groups of data. This will be the line such that the distances from the closest point in each of the two groups will be farthest away.



In the example shown above, the line which splits the data into two differently classified groups is the *black* line, since the two closest points are the farthest apart from the line. This line is our classifier. Then, depending on where the testing data lands on either side of the line, that's what class we can classify the new data as.

**Think of this algorithm as playing JezzBall in n-dimensional space. The tweaks in the game are:**

- You can draw lines / planes at any angles (rather than just horizontal or vertical as in classic game)
- The objective of the game is to segregate balls of different colors in different rooms.
- And the balls are not moving.

***Python Code***

```
#Import Library

from sklearn import svm

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dat
aset

# Create SVM classification object

model = svm.svc() # there is various option associated with it, this is simple for classification. You can
refer link, for mo# re detail.

# Train the model using the training sets and check score

model.fit(X, y)

model.score(X, y)

#Predict Output

predicted= model.predict(x_test)
```

**R Code**

```
library(e1071)

x <- cbind(x_train,y_train)

# Fitting model
```

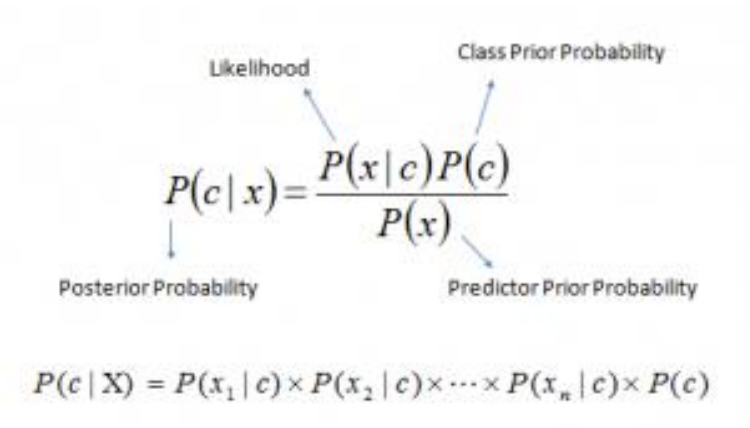
```
fit <-svm(y_train ~ ., data = x)

summary(fit)

#Predict Output

predicted= predict(fit,x_test)
```

**5. Naive Bayes**

It is a classification technique based on Bayes' theorem with an assumption of independence between predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a naive Bayes classifier would consider all of these properties to independently contribute to the probability that this fruit is an apple.

Naive Bayesian model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:




Here,

- *P*(*c*|*x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
- *P*(*x*) is the prior probability of *predictor*.

**Example:** Let's understand it using an example. Below I have a training data set of weather and corresponding target variable 'Play'. Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Step 1: Convert the data set to frequency table

Step 2: Create Likelihood table by finding the probabilities like Overcast probability = 0.29 and probability of playing is 0.64.

| Weather | Play |
|---|---|
| Sunny | No |
| Overcast | Yes |
| Rainy | Yes |
| Sunny | Yes |
| Sunny | Yes |
| Overcast | Yes |
| Rainy | No |
| Rainy | No |
| Sunny | Yes |
| Rainy | Yes |
| Sunny | No |
| Overcast | Yes |
| Overcast | Yes |
| Rainy | No |

| Frequency Table | | |
|---|---|---|
| Weather | No | Yes |
| Overcast | | 4 |
| Rainy | 3 | 2 |
| Sunny | 2 | 3 |
| Grand Total | 5 | 9 |

| Likelihood table | | | | |
|---|---|---|---|---|
| Weather | No | Yes | | |
| Overcast | | 4 | =4/14 | 0.29 |
| Rainy | 3 | 2 | =5/14 | 0.36 |
| Sunny | 2 | 3 | =5/14 | 0.36 |
| All | 5 | 9 | | |
| | =5/14 | =9/14 | | |
| | 0.36 | 0.64 | | |

Step 3: Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

**Problem:** Players will pay if weather is sunny, is this statement is correct?

We can solve it using above discussed method, so P(Yes | Sunny) = P( Sunny | Yes) * P(Yes) / P (Sunny)

Here we have P (Sunny |Yes) = 3/9 = 0.33, P(Sunny) = 5/14 = 0.36, P( Yes)= 9/14 = 0.64

Now, P (Yes | Sunny) = 0.33 * 0.64 / 0.36 = 0.60, which has higher probability.

Naive Bayes uses a similar method to predict the probability of different class based on various attributes. This algorithm is mostly used in text classification and with problems having multiple classes.

***Python Code***

```
#Import Library

from sklearn.naive_bayes import GaussianNB

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dat
aset

# Create SVM classification object model = GaussianNB() # there is other distribution for multinomial
classes like Bernoulli Naive Bayes, Refer link

# Train the model using the training sets and check score

model.fit(X, y)
```

```
#Predict Output

predicted= model.predict(x_test)
```

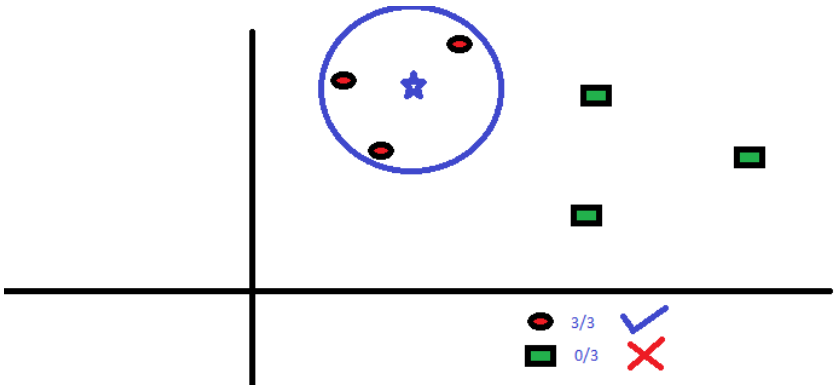
**R Code**

```
library(e1071)

x <- cbind(x_train,y_train)

# Fitting model

fit <-naiveBayes(y_train ~ ., data = x)

summary(fit)

#Predict Output

predicted= predict(fit,x_test)
```

6. kNN (k- Nearest Neighbors)

It can be used for both classification and regression problems. However, it is more widely used in classification problems in the industry. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors. The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.

These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables. If K = 1, then the case is simply assigned to the class of its nearest neighbor. At times, choosing K turns out to be a challenge while performing kNN modeling.



KNN can easily be mapped to our real lives. If you want to learn about a person, of whom you have no information, you might like to find out about his close friends and the circles he moves in and gain access to his/her information!

**Things to consider before selecting kNN:**

- KNN is computationally expensive
- Variables should be normalized else higher range variables can bias it
- Works on pre-processing stage more before going for kNN like outlier, noise removal

***Python Code***

```
#Import Library

from sklearn.neighbors import KNeighborsClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dat
aset

# Create KNeighbors classifier object model

KNeighborsClassifier(n_neighbors=6) # default value for n_neighbors is 5

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

**R Code**

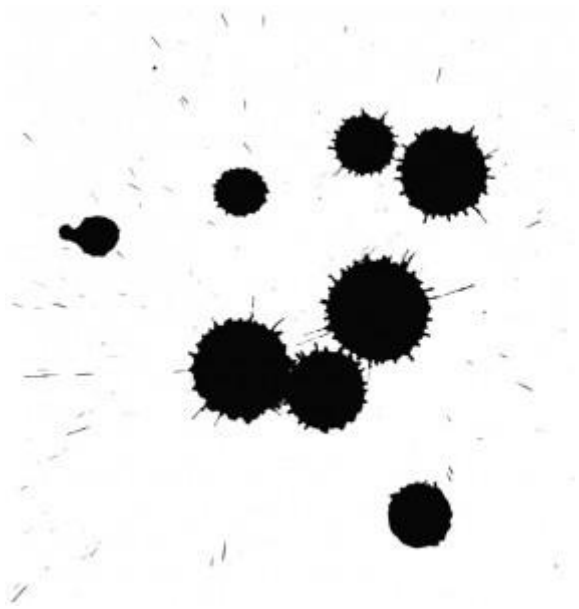
```
library(knn)

x <- cbind(x_train,y_train)

# Fitting model

fit <-knn(y_train ~ ., data = x,k=5)

summary(fit)

#Predict Output
```

```
predicted= predict(fit,x_test)
```

7**. K-Means**

It is a type of unsupervised algorithm which solves the clustering problem. Its procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters). Data points inside a cluster are homogeneous and heterogeneous to peer groups.

Remember figuring out shapes from ink blots? k means is somewhat similar this activity. You look at the shape and spread to decipher how many different clusters / population are present!



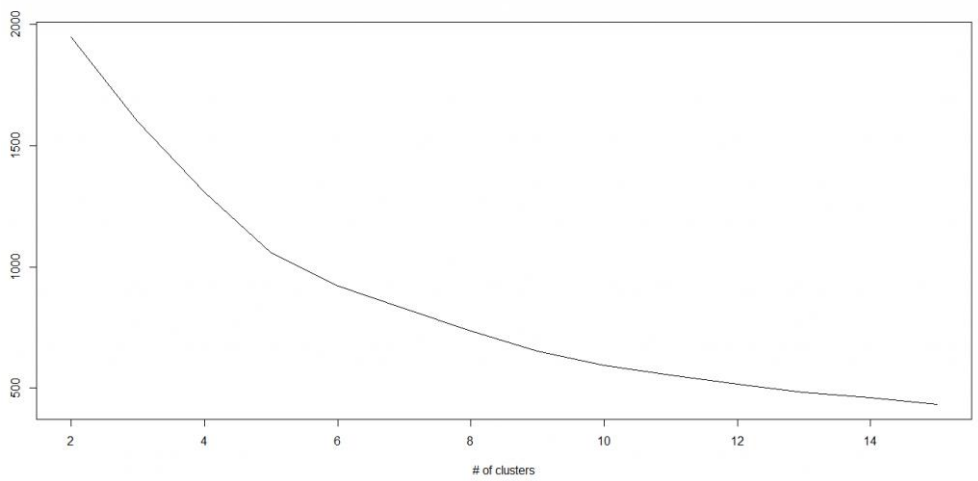
**How K-means forms cluster:**

1. K-means picks k number of points for each cluster known as centroids.
2. Each data point forms a cluster with the closest centroids i.e. k clusters.
3. Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
4. As we have new centroids, repeat step 2 and 3. Find the closest distance for each data point from new centroids and get associated with new k-clusters. Repeat this process until convergence occurs i.e. centroids does not change.

**How to determine value of K:**

In K-means, we have clusters and each cluster has its own centroid. Sum of square of difference between centroid and the data points within a cluster constitutes within sum of square value for that cluster. Also, when the sum of square values for all the clusters are added, it becomes total within sum of square value for the cluster solution.

We know that as the number of cluster increases, this value keeps on decreasing but if you plot the result you may see that the sum of squared distance decreases sharply up to some value of k, and then much more slowly after that. Here, we can find the optimum number of cluster.




***Python Code***

```
#Import Library

from sklearn.cluster import KMeans

#Assumed you have, X (attributes) for training data set and x_test(attributes) of test_dataset

# Create KNeighbors classifier object model

k_means = KMeans(n_clusters=3, random_state=0)

# Train the model using the training sets and check score

model.fit(X)

#Predict Output

predicted= model.predict(x_test)
```

**R Code**

```
library(cluster)

fit <- kmeans(X, 3) # 5 cluster solution
```

**8. Random Forest**

Random Forest is a trademark term for an ensemble of decision trees. In Random Forest, we've collection of decision trees (so known as "Forest"). To classify a new object based on attributes,

each tree gives a classification and we say the tree “votes” for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is planted & grown as follows:

1. If the number of cases in the training set is N, then sample of N cases is taken at random but *with replacement*. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

**Python**

```
#Import Library

from sklearn.ensemble import RandomForestClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dat
aset

# Create Random Forest object

model= RandomForestClassifier()

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

**R Code**

```
library(randomForest)

x <- cbind(x_train,y_train)

# Fitting model

fit <- randomForest(Species ~ ., x,ntree=500)

summary(fit)
```

```
#Predict Output

predicted= predict(fit,x_test)
```

**9. Dimensionality Reduction Algorithms**

In the last 4-5 years, there has been an exponential increase in data capturing at every possible stages. Corporates/ Government Agencies/ Research organisations are not only coming with new sources but also they are capturing data in great detail.

For example: E-commerce companies are capturing more details about customer like their demographics, web crawling history, what they like or dislike, purchase history, feedback and many others to give them personalized attention more than your nearest grocery shopkeeper.

As a data scientist, the data we are offered also consist of many features, this sounds good for building good robust model but there is a challenge. How'd you identify highly significant variable(s) out 1000 or 2000? In such cases, dimensionality reduction algorithm helps us along with various other algorithms like Decision Tree, Random Forest, PCA, Factor Analysis, Identify based on correlation matrix, missing value ratio and others.

***Python Code***

```
#Import Library

from sklearn import decomposition

#Assumed you have training and test data set as train and test

# Create PCA obeject pca= decomposition.PCA(n_components=k) #default value of k =min(n_sample
, n_features)

# For Factor analysis

#fa= decomposition.FactorAnalysis()

# Reduced the dimension of training dataset using PCA

train_reduced = pca.fit_transform(train)

#Reduced the dimension of test dataset

test_reduced = pca.transform(test)

#For more detail on this, please refer  this link.
```

***R Code***

```
library(stats)

pca <- princomp(train, cor = TRUE)

train_reduced  <- predict(pca,train)

test_reduced  <- predict(pca,test)
```

**10. Gradient Boosting Algorithms**

**10.1. GBM**

GBM is a boosting algorithm used when we deal with plenty of data to make a prediction with high prediction power. Boosting is actually an ensemble of learning algorithms which combines the prediction of several base estimators in order to improve robustness over a single estimator. It combines multiple weak or average predictors to a build strong predictor. These boosting algorithms always work well in data science competitions like Kaggle, AV Hackathon, CrowdAnalytix.

***Python Code***

```
#Import Library

from sklearn.ensemble import GradientBoostingClassifier

#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dat
aset

# Create Gradient Boosting Classifier object

model= GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_stat
e=0)

# Train the model using the training sets and check score

model.fit(X, y)

#Predict Output

predicted= model.predict(x_test)
```

***R Code***

```
library(caret)

x <- cbind(x_train,y_train)

# Fitting model

fitControl <- trainControl( method = "repeatedcv", number = 4, repeats = 4)

fit <- train(y ~ ., data = x, method = "gbm", trControl = fitControl,verbose = FALSE)

predicted= predict(fit,x_test,type= "prob")[,2]
```

**10.2. XGBoost**

Another classic gradient boosting algorithm that's known to be the decisive choice between winning and losing in some Kaggle competitions.

The XGBoost has an immensely high predictive power which makes it the best choice for accuracy in events as it possesses both linear model and the tree learning algorithm, making the algorithm almost 10x faster than existing gradient booster techniques.

The support includes various objective functions, including regression, classification and ranking.

One of the most interesting things about the XGBoost is that it is also called a regularized boosting technique. This helps to reduce overfit modelling and has a massive support for a range of languages such as Scala, Java, R, Python, Julia and C++.

Supports distributed and widespread training on many machines that encompass GCE, AWS, Azure and Yarn clusters. XGBoost can also be integrated with Spark, Flink and other cloud dataflow systems with a built in cross validation at each iteration of the boosting process.

**Python Code:**

```
from xgboost import XGBClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

X = dataset[:,0:10]

Y = dataset[:,10:]
```

```
seed = 1

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, random_state=seed)

model = XGBClassifier()

model.fit(X_train, y_train)

#Make predictions for test data

y_pred = model.predict(X_test)
```

**R Code:**

```
require(caret)

x <- cbind(x_train,y_train)

# Fitting model

TrainControl <- trainControl( method = "repeatedcv", number = 10, repeats = 4)

model<- train(y ~ ., data = x, method = "xgbLinear", trControl = TrainControl,verbose = FALSE)

OR

model<- train(y ~ ., data = x, method = "xgbTree", trControl = TrainControl,verbose = FALSE)

predicted <- predict(model, x_test)
```

### 10.3. LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Parallel and GPU learning supported
- Capable of handling large-scale data

The framework is a fast and high-performance gradient boosting one based on decision tree algorithms, used for ranking, classification and many other machine learning tasks. It was developed under the Distributed Machine Learning Toolkit Project of Microsoft.

Since the LightGBM is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.

Also, it is surprisingly very fast, hence the word ‘Light’.

Python Code:

```
data = np.random.rand(500, 10) # 500 entities, each contains 10 features

label = np.random.randint(2, size=500) # binary target

train_data = lgb.Dataset(data, label=label)

test_data = train_data.create_valid('test.svm')

param = {'num_leaves':31, 'num_trees':100, 'objective':'binary'}

param['metric'] = 'auc'

num_round = 10

bst = lgb.train(param, train_data, num_round, valid_sets=[test_data])

bst.save_model('model.txt')

# 7 entities, each contains 10 features

data = np.random.rand(7, 10)

ypred = bst.predict(data)
```

R Code:

```
library(RLightGBM)

data(example.binary)
```

```
#Parameters

num_iterations <- 100

config <- list(objective = "binary",  metric="binary_logloss,auc", learning_rate = 0.1, num_leaves = 63, tree_learner = "serial", feature_fraction = 0.8, bagging_freq = 5, bagging_fraction = 0.8, min_data_in_leaf = 50, min_sum_hessian_in_leaf = 5.0)

#Create data handle and booster

handle.data <- lgbm.data.create(x)

lgbm.data.setField(handle.data, "label", y)

handle.booster <- lgbm.booster.create(handle.data, lapply(config, as.character))

#Train for num_iterations iterations and eval every 5 steps

lgbm.booster.train(handle.booster, num_iterations, 5)

#Predict

pred <- lgbm.booster.predict(handle.booster, x.test)

#Test accuracy

sum(y.test == (y.pred > 0.5)) / length(y.test)

#Save model (can be loaded again via lgbm.booster.load(filename))

lgbm.booster.save(handle.booster, filename = "/tmp/model.txt")
```

If you’re familiar with the Caret package in R, this is another way of implementing the LightGBM.

```
require(caret)

require(RLightGBM)

data(iris)

model <-caretModel.LGBM()

fit <- train(Species ~ ., data = iris, method=model, verbosity = 0)
```

```
print(fit)

y.pred <- predict(fit, iris[,1:4])

library(Matrix)

model.sparse <- caretModel.LGBM.sparse()

#Generate a sparse matrix

mat <- Matrix(as.matrix(iris[,1:4]), sparse = T)

fit <- train(data.frame(idx = 1:nrow(iris)), iris$Species, method = model.sparse, matrix = mat, verbosity = 0)

print(fit)
```

**10.4. Catboost**

CatBoost is a recently open-sourced machine learning algorithm from Yandex. It can easily integrate with deep learning frameworks like Google's TensorFlow and Apple's Core ML.

The best part about CatBoost is that it does not require extensive data training like other ML models, and can work on a variety of data formats; not undermining how robust it can be.

Make sure you handle missing data well before you proceed with the implementation.

Catboost can automatically deal with categorical variables without showing the type conversion error, which helps you to focus on tuning your model better rather than sorting out trivial errors.

Python Code:

```
import pandas as pd

import numpy as np

from catboost import CatBoostRegressor

#Read training and testing files

train = pd.read_csv("train.csv")

test = pd.read_csv("test.csv")
```

```
#Imputing missing values for both train and test

train.fillna(-999, inplace=True)

test.fillna(-999,inplace=True)

#Creating a training set for modeling and validation set to check model performance

X = train.drop(['Item_Outlet_Sales'], axis=1)

y = train.Item_Outlet_Sales

from sklearn.model_selection import train_test_split



X_train, X_validation, y_train, y_validation = train_test_split(X, y, train_size=0.7, random_state=1234)

categorical_features_indices = np.where(X.dtypes != np.float)[0]

#importing library and building model

from catboost import CatBoostRegressormodel=CatBoostRegressor(iterations=50, depth=3, learning_
rate=0.1, loss_function='RMSE')

model.fit(X_train, y_train,cat_features=categorical_features_indices,eval_set=(X_validation, y_validati
on),plot=True)

submission = pd.DataFrame()

submission['Item_Identifier'] = test['Item_Identifier']

submission['Outlet_Identifier'] = test['Outlet_Identifier']

submission['Item_Outlet_Sales'] = model.predict(test)
```

**R Code:**

```
set.seed(1)

require(titanic)

require(caret)
```

```
require(catboost)

tt <- titanic::titanic_train[complete.cases(titanic::titanic_train),]

data <- as.data.frame(as.matrix(tt), stringsAsFactors = TRUE)

drop_columns = c("PassengerId", "Survived", "Name", "Ticket", "Cabin")

x <- data[,!(names(data) %in% drop_columns)]y <- data[,c("Survived")]

fit_control <- trainControl(method = "cv", number = 4,classProbs = TRUE)

grid <- expand.grid(depth = c(4, 6, 8),learning_rate = 0.1,iterations = 100, l2_leaf_reg = 1e-3,           r
sm = 0.95, border_count = 64)

report <- train(x, as.factor(make.names(y)),method = catboost.caret,verbose = TRUE, preProc = NUL
L,tuneGrid = grid, trControl = fit_control)

print(report)

importance <- varImp(report, scale = FALSE)

print(importance)
```

## Additional Reading

### Artificial intelligence (AI), deep learning, and neural networks

Artificial intelligence (AI), deep learning, and neural networks represent incredibly exciting and powerful machine learning-based techniques used to solve many real-world problems.

While human-like deductive reasoning, inference, and decision-making by a computer is still a long time away, there have been remarkable gains in the application of AI techniques and associated algorithms.

The primary motivation and driving force for these areas of study, and for developing these techniques further, is that the solutions required to solve certain problems are incredibly complicated, not well understood, nor easy to determine manually.

Increasingly, we rely on these techniques and machine learning to solve these problems for us, without requiring explicit programming instructions. This is critical for two reasons. The first is that we likely wouldn't be able, or at least know how to write the programs required to model and solve many problems that AI techniques are able to solve. Second, even if we did know how to write the programs, they would be inordinately complex and nearly impossible to get right.

Luckily for us, machine learning and AI algorithms, along with properly selected and prepared training data, are able to do this for us.

Artificial Intelligence Overview

In order to define AI, we must first define the concept of *intelligence* in general. A paraphrased definition based on Wikipedia is:

Intelligence can be generally described as the ability to perceive information, and retain it as knowledge to be applied towards adaptive behaviors within an environment or context.

While there are many different definitions of intelligence, they all essentially involve learning, understanding, and the application of the knowledge learned to achieve one or more goals.

It's therefore a natural extension to say that AI can be described as intelligence exhibited by machines. So what does that mean exactly, when is it useful, and how does it work?

A familiar instance of an AI solution includes *IBM's Watson*, which was made famous by beating the two greatest *Jeopardy* champions in history, and is now being used as a *question answering computing system* for commercial applications. *Apple's Siri* and*Amazon's Alexa* are similar examples as well.

In addition to speech recognition and natural language (processing, generation, and understanding) applications, AI is also used for other recognition tasks (pattern, text, audio, image, video, facial, …), autonomous vehicles, medical diagnoses, gaming, search engines, spam filtering, crime fighting, marketing, robotics, remote sensing, computer vision, transportation, music recognition, classification, and so on.

Something worth mentioning is a concept known as the *AI effect*. This describes the case where once an AI application has become somewhat mainstream, it's no longer considered by many as AI. It happens because people's tendency is to no longer think of the solution as involving real intelligence, and only being a application of normal computing.

This despite the fact that these applications still fit the definition of AI regardless of widespread usage. The key takeaway here is that today's AI is not necessarily tomorrow's AI, at least not in some people's minds anyway.

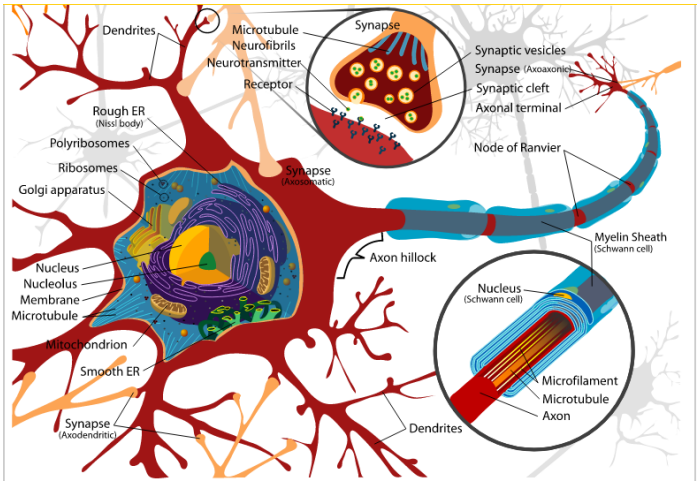
There are many different goals of AI as mentioned, with different techniques used for each. The primary topics of this article are artificial neural networks and an advanced version known as deep learning.

## Biological Neural Networks Overview

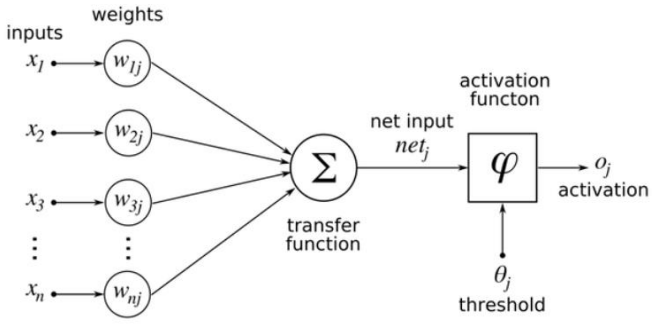
The human brain is exceptionally complex and quite literally the most powerful computing machine known.

The inner-workings of the human brain are often modeled around the concept of*neurons* and the networks of neurons known as *biological neural networks*. According to Wikipedia, it's estimated that the human brain contains roughly 100 billion neurons, which are connected along pathways throughout these networks.

At a very high level, neurons interact and communicate with one another through an interface consisting of axon terminals that are connected to dendrites across a gap (synapse) as shown here.




In plain english, a single neuron will pass a message to another neuron across this interface if the sum of *weighted input signals* from one or more neurons (*summation*) into it is great enough (exceeds a *threshold*) to cause the message transmission. This is called *activation* when the threshold is exceeded and the message is passed along to the next neuron.




The *summation* process can be mathematically complex. Each neuron's input signal is actually a *weighted combination* of potentially many input signals, and the weighting of each input means that that input can have a different influence on any subsequent calculations, and ultimately on the final output of the entire network.

In addition, each neuron applies a function or *transformation* to the weighted inputs, which means that the combined weighted input signal is transformed mathematically prior to evaluating if the *activation threshold* has been exceeded. This combination of weighted input signals and the functions applied are typically either linear or nonlinear.

These input signals can originate in many ways, with our senses being some of the most important, as well as ingestion of gases (breathing), liquids (drinking), and solids (eating) for example. A single neuron may receive hundreds of thousands of input signals at once that undergo the summation process to determine if the message gets passed along, and ultimately causes the brain to instruct actions, memory recollection, and so on.

The 'thinking' or processing that our brain carries out, and the subsequent instructions given to our muscles, organs, and body are the result of these neural networks in action. In addition, the brain's neural networks continuously change and update themselves in many ways, including modifications to the amount of weighting applied between neurons. This happens as a direct result of learning and experience.

Given this, it's a natural assumption that for a computing machine to replicate the brain's functionality and capabilities, including being 'intelligent', it must successfully implement a computer-based or artificial version of this network of neurons.

This is the genesis of the advanced statistical technique and term known as *artificial neural networks*.

**Artificial Neural Networks Overview**

*Artificial neural networks* (*ANNs*) are statistical models directly inspired by, and partially modeled on biological neural networks. They are capable of modeling and processing nonlinear relationships between inputs and outputs in parallel. The related algorithms are part of the broader field of machine learning, and can be used in many applications as discussed.

Artificial neural networks are characterized by containing *adaptive weights* along paths between neurons that can be tuned by a *learning algorithm* that *learns* from observed data in order to improve the model. In addition to the learning algorithm itself, one must choose an appropriate *cost function*.

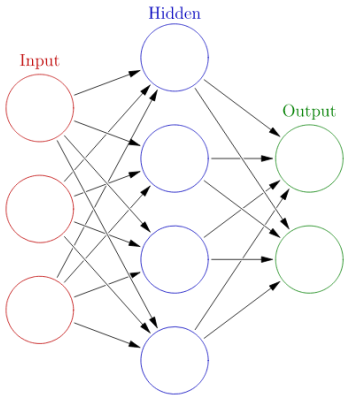
The cost function is what's used to *learn* the optimal solution to the problem being solved. This involves determining the best values for all of the tunable model parameters, with neuron path adaptive weights being the primary target, along with algorithm tuning parameters such as the *learning rate*. It's usually done through*optimization* techniques such as *gradient descent* or *stochastic gradient descent*.

These optimization techniques basically try to make the ANN solution be as close as possible to the optimal solution, which when successful means that the ANN is able to solve the intended problem with high performance.

Architecturally, an artificial neural network is modeled using layers of *artificial neurons*, or computational units able to receive input and apply an activation function along with a threshold to determine if messages are passed along.

In a simple model, the first layer is the *input* layer, followed by one *hidden* layer, and lastly by an *output* layer. Each layer can contain one or more neurons.

Models can become increasingly complex, and with increased abstraction and problem solving capabilities by increasing the number of *hidden layers*, the number of neurons in any given layer, and/or the number of paths between neurons. Note that an increased chance of *overfitting* can also occur with increased model complexity.



Model architecture and tuning are therefore major components of ANN techniques, in addition to the actual learning algorithms themselves. All of these characteristics of an ANN can have significant impact on the performance of the model.

Additionally, models are characterized and tunable by the *activation function* used to convert a neuron's weighted input to its output activation. There are many different types of transformations that can be used as the activation function, and a discussion of them is out of scope for this article.

The abstraction of the output as a result of the transformations of input data through neurons and layers is a form of *distributed representation*, as contrasted with*local representation*. The meaning represented by a single artificial neuron for example is a form of local representation. The meaning of the entire network however, is a form of distributed representation due to the many transformations across neurons and layers.

One thing worth noting is that while ANNs are extremely powerful, they can also be very complex and are considered *black box* algorithms, which means that their inner-workings are very difficult to understand and explain. Choosing whether to employ ANNs to solve problems should therefore be chosen with that in mind.

**Deep Learning Introduction**

*Deep learning*, while sounding flashy, is really just a term to describe certain types of neural networks and related algorithms that consume often very *raw* input data. They process this data through many layers of nonlinear transformations of the input data in order to calculate a target output.
Unsupervised *feature extraction* is also an area where deep learning excels. Feature extraction is when an algorithm is able to automatically derive or construct meaningful features of the data to be used for further learning, generalization, and understanding. The burden is traditionally on the data scientist or programmer to carry out the feature extraction process in most other machine learning approaches, along with feature selection and engineering.

Feature extraction usually involves some amount dimensionality reduction as well, which is reducing the amount of input features and data required to generate meaningful results. This has many benefits, which include simplification, computational and memory power reduction, and so on.

More generally, deep learning falls under the group of techniques known as *feature learning* or *representation learning*. As discussed so far, feature extraction is used to 'learn' which features to focus on and use in machine learning solutions. The machine learning algorithms themselves 'learn' the optimal parameters to create the best performing model.

Paraphrasing Wikipedia, *feature learning* algorithms allow a machine to both learn for a specific task using a well-suited set of features, and also learn the features themselves. In other words, these algorithms *learn how to learn*!
Deep learning has been used successfully in many applications, and is considered to be one of the most cutting-edge machine learning and AI techniques at the time of this writing. The associated algorithms are often used for *supervised*, *unsupervised*, and *semi-supervised* learning problems.

For neural network-based deep learning models, the number of layers are greater than in so-called *shallow learning* algorithms. Shallow algorithms tend to be less complex and require more up-front knowledge of optimal features to use, which typically involves feature selection and engineering.
In contrast, deep learning algorithms rely more on optimal model selection and optimization through model tuning. They are more well suited to solve problems where prior knowledge of features is less desired or necessary, and where labeled data is unavailable or not required for the primary use case.

In addition to statistical techniques, neural networks and deep learning leverage concepts and techniques from signal processing as well, including nonlinear processing and/or transformations.

You may recall that a nonlinear function is one that is not characterized simply by a straight line. It therefore requires more than just a slope to model the relationship between the input, or independent variable, and the output, or dependent variable. Nonlinear functions can include polynomial, logarithmic, and exponential terms, as well as any other transformation that isn't linear.

Many phenomena observed in the physical universe are actually best modeled with nonlinear transformations. This is true as well for transformations between inputs and the target output in machine learning and AI solutions.

**A Deeper Dive into Deep Learning**

As mentioned, input data is transformed throughout the layers of a deep learning neural network by artificial neurons or processing units. The chain of transformations that occur from input to output is known as the *credit assignment path*, or *CAP*.

The CAP value is a proxy for the measurement or concept of 'depth' in a deep learning model architecture. According to Wikipedia, most researchers in the field agree that deep learning has multiple nonlinear layers with a CAP greater than two, and some consider a CAP greater than ten to be *very deep learning*.

While a detailed discussion of the many different deep-learning model architectures and learning algorithms is beyond the scope of this article, some of the more notable ones include:

- Feed-forward neural networks
- Recurrent neural network
- Multi-layer perceptrons (MLP)
- Convolutional neural networks
- Recursive neural networks
- Deep belief networks
- Convolutional deep belief networks
- Self-Organizing Maps
- Deep Boltzmann machines
- Stacked de-noising auto-encoders

It's worth pointing out that due to the relative increase in complexity, deep learning and neural network algorithms can be prone to overfitting. In addition, increased model and algorithmic complexity can result in very significant computational resource and time requirements.

It's also important to consider that solutions may represent *local minima* as opposed to a global optimal solution. This is due to the complex nature of these models when combined with optimization techniques such as gradient descent.

Given all of this, proper care must be taken when leveraging artificial intelligence algorithms to solve problems, including the selection, implementation, and performance assessment of algorithms themselves. While out of scope for this article, the field of machine learning includes many techniques that can help with these areas.

## Cognitive Computing

The goal of cognitive computing is to simulate human thought processes in a computerized model. Using self-learning algorithms that use data mining, pattern recognition and natural language processing, the computer can mimic the way the human brain works.

While computers have been faster at calculations and processing than humans for decades, they haven't been able to accomplish tasks that humans take for granted as simple, like understanding natural language, or recognizing unique objects in an image.

Some people say that cognitive computing represents the third era of computing: we went from computers that could tabulate sums (1900s) to programmable systems (1950s), and now to cognitive systems.

These cognitive systems, most notably IBM Watson, rely on deep learning algorithms and neural networks to process information by comparing it to a teaching set of data. The more data the system is exposed to, the more it learns, and the more accurate it becomes over time, and the neural network is a complex "tree" of decisions the computer can make to arrive at an answer.

**What can cognitive computing do?**

For example, according to a TED Talk video from IBM, Watson could eventually be applied in a healthcare setting to help collate the span of knowledge around a condition, including patient history, journal articles, best practices, diagnostic tools, etc., analyze that vast quantity of information, and provide a recommendation.

The doctor is then able to look at evidence-based treatment options based on a large number of factors including the individual patient's presentation and history, to hopefully make better treatment decisions.

In other words, the goal (at this point) is not to replace the doctor, but expand the doctor's capabilities by processing the humongous amount of data available that no human could reasonably process and retain, and provide a summary and potential application.

This sort of process could be done for any field in which large quantities of complex data need to be processed and analyzed to solve problems, including finance, law, and education.

These systems will also be applied in other areas of business including consumer behavior analysis, personal shopping bots, customer support bots, travel agents, tutors, security, and diagnostics. Hilton Hotels recently debuted the first concierge robot, Connie, which can answer questions about the hotel, local attractions, and restaurants posed to it in natural language.

The personal digital assistants we have on our phones and computers now (Siri and Google among others) are not true cognitive systems; they have a pre-programmed set of responses and can only respond to a preset number of requests. But the time is coming in the near future when we will be able to address our phones, our computers, our cars, or our smart houses and get a real, thoughtful response rather than a pre-programmed one.

As computers become more able to think like human beings, they will also expand our capabilities and knowledge. Just as the heroes of science fiction movies rely on their computers to make accurate predictions, gather data, and draw conclusions, so we will move into an era when computers can augment human knowledge and ingenuity in entirely new ways.