**DATA PREPROCESSING**

**What is Data Preprocessing ?**

* **Data preprocessing** is a **data** mining technique that involves transforming raw **data** into an understandable format.
* Real-world **data** is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors.
* **Data preprocessing** is a proven method of resolving such issues.

**Steps in Data Preprocessing**

* **Step 1 :** Import the libraries
* **Step 2 :** Import the data-set
* **Step 3 :** Check out the missing values
* **Step 4 :** See the Categorical Values
* **Step 5 :** Splitting the data-set into Training and Test Set
* **Step 6 :** Feature Scaling

**STEP 1 :  IMPORT THE LIBRARIES**

**import numpy as np**

**import pandas as pd**

**import matplotlib.pyplot as plt**

**import warnings**

**warnings.filterwarnings('ignore')**

**NumPy** is the fundamental package for scientific computing with Python. It contains among other things:

1. A powerful N-dimensional array object
2. Sophisticated (broadcasting) functions
3. Tools for integrating C/C++ and FORTRAN code
4. Useful linear algebra, Fourier transform, and random number capabilities

**Pandas :**

1. Pandas is for data manipulation and analysis.
2. Pandas is an open source
3. BSD-licensed library providing high-performance
4. easy-to-use data structures and data analysis tools for the [Python](https://www.python.org/) programming language.

**Matplotlib**

1. Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hard copy formats and interactive environments across platforms.
2. Matplotlib can be used in Python scripts, the Python and [IPython](http://ipython.org/" \t "_blank) shells, the [Jupyter](http://jupyter.org/" \t "_blank) notebook, web application servers, and four graphical user interface toolkits.

**Seaborn**

1. Seaborn is a Python data visualization library based on [matplotlib](https://matplotlib.org/" \t "_blank).
2. It provides a high-level interface for drawing attractive and informative statistical graphics.

**Warning:-**

1. [Warning](https://docs.python.org/3.1/library/warnings.html) messages are typically issued in situations where it is useful to alert the user of some condition in a program
2. For example, one might want to issue a warning when a program uses an obsolete module.

**Import the Data-set**

**dataset= pd.read\_csv('WorldCupMatches.csv')**

**dataset.head(5)**

**dataset.shape**

**dataset.index**

**dataset.columns**

* By using Pandas we import our data-set and the file I used here is .csv file [Note: It’s not necessarily every-time you deal with **CSV** file, sometimes you deal with **Html or Xlsx(Excel file)** ].
* However, to access and to use fastly we use CSV files because of their light weights. After importing the dataset, you can see we use head function ( This function returns the first n rows for the object based on position.
* It is useful for quickly testing if your object has the right type of data in it. By default it returns 5 rows. )

**Check out the Missing Values**

* The concept of missing values is important to understand in order to successfully [manage](http://www.statisticssolutions.com/academic-solutions/resources/dissertation-resources/data-entry-and-management/multiple-imputation-for-missing-data/) data.
* If the missing values are not handled properly by the researcher, then he/she may end up drawing an inaccurate inference about the data.
* Due to improper handling, the result obtained by the researcher will differ from ones where the missing values are present.

**Two ways to handle Missing Values**

**Method1 :-**

dataset.dropna(inplace=True)

dataset.isnull().sum()

* This method commonly used to handle the null values.
* Here, we either delete a particular row if it has a null value for a particular feature and a particular column if it has more than 75% of missing values.
* This method is advised only when there are enough samples in the data set.
* One has to make sure that after we have deleted the data, there is no addition of bias. Removing the data will lead to loss of information which will not give the expected results while predicting the output.

**Method2:-**

**dataset['Year'].mean()**

**dataset['Year'].tail()**

**dataset['Year'].replace(np.NaN,dataset['Year'].mean()).tail()**

* This strategy can be applied on a feature which has numeric data like the year column or Home team goal column.
* We can calculate the **mean, median or mode** of the feature and replace it with the missing values.
* This is an approximation which can add variance to the data set.
* But the loss of the data can be negated by this method which yields better results compared to removal of rows and columns.
* Replacing with the above three approximations are a statistical approach of handling the missing values.
* This method is also called as **leaking the data** while training.
* Another way is to approximate it with the deviation of neighbouring values. This works better if the data is linear.

**See the Categorical Values**

* **Machine learning models are based on Mathematical equations and you can intuitively understand that it would cause some problem if we can keep the Categorical data in the equations because we would only want numbers in the equations.**
* **So, we need to encode the Categorical Variable…..**
* **To convert Categorical variable into Numerical data we can use LabelEncoder() class from preprocessing library.**

**# working on categorical Data**

**from sklearn.preprocessing import LabelEncoder**

**label\_encoder = LabelEncoder()**

**X[:,0] =label\_encoder.fit\_transform(X[:,0])**

* label\_encoder is object which is I use and help us in transferring Categorical data into Numerical data.
* Next, I fitted this label\_encoder object to the first column of our matrix X and all this return the first column country of the matrix X encoded.
* But there is a problem in it, the problem is still the same, machine learning models are based on equations and that’s good that we replaced the text by numbers so that we can include the numbers in the equations.
* However, since 1>0 and 2>1(See the above data-set) , the equations in the model will think that Spain has a higher value than Germany and France, and Germany has a higher value than France.
* Actually, this is a not the case, these are actually three Categories and there is no relational order between the three. So , we have to prevent this, we’re going to use what are **Dummy Variables.**

**What is Dummy Variables ?**

* **Dummy Variables** is one that takes the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome.

***Number of Columns = Number of Categories***

* To create dummy variable we are going to use OneHotEncoder Class from sklearn.preprocessing or you can use pandas get dummies method.
* I will show you with pandas how to use **get\_dummies( )**for creating Dummy Variables.

**Working on Dummy Variable**

from sklearn.preprocessing import OneHotEncoder

onehotencoder = OneHotEncoder(categorical\_features=[0])

X = onehotencoder.fit\_transform(X)

dummy =pd.get\_dummies(dataset['Country'])

dummy

**concatnating dummy variables to data set**

dataset = pd.concat([dataset,dummy],axis=1)

dataset.Purchased.replace(('Yes', 'No'), (1, 0), inplace=True)

dataset.drop(['Country'],axis=1)

y=dataset.iloc[:,3].values

**Splitting the data-set into Training and Test Set**

In any Machine Learning model is that we’re going to split data-set into two separate sets

1. Training Set

2. Test Set

**Why we need splitting ?**

* Well here it’s your algorithm model that is going to learn from your data to make predictions.
* Generally we split the data-set into 70:30 ratio or 80:20 what does it mean, 70 percent data take in train and 30 percent data take in test.
* However, this Splitting can be varies according to the data-set shape and size.

**X\_train** 🡪 is the training part of the matrix of features.

**X\_test 🡪** is the test part of the matrix of features.

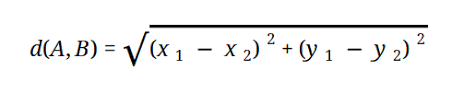
**y\_train** 🡪 is the training part of the dependent variable that is associated to X\_train here.

**y\_test** **🡪** is the test part of the dependent variable that is associated to X\_train here.

**Feature Scaling**

**What is Feature Scaling ?**

* **Feature scaling** is the method to limit the range of variables so that they can be compared on common grounds.
* Suppose we have this data-set
* See the Age and Salary column. You can easily noticed Salary and Age variable don’t have the same scale and this will cause some issue in your machine learning model.
* Because most of the Machine Learning models are based on **Euclidean Distance**.

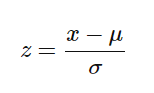


* Let’s say we take two values from Age and Salary column
* One can easily compute and see that Salary column will be dominated in Euclidean Distance. And we don’t want this thing.

**Feature Scaling or Standardization:**

* It is a step of Data Pre Processing which is applied to independent variables or features of data.
* It basically helps to normalise the data within a particular range.
* Sometimes, it also helps in speeding up the calculations in an algorithm.

**from sklearn.preprocessing import StandardScaler**

* Formula used in Backend Standardisation replaces the values by their Z scores.  
  
* Mostly the Fit method is used for Feature scaling

fit(X, y = None)

import pandas as pd

from sklearn.preprocessing import StandardScaler

# Read Data from CSV

data = read\_csv('Data.csv')

data.head()

# Initialise the Scaler

scaler = StandardScaler()

# To scale data

scaler.fit(data)

**Why and Where to Apply Feature Scaling?**

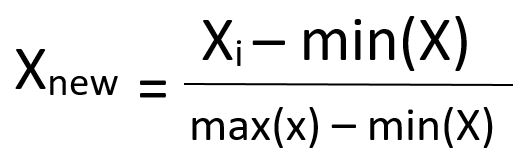
* Real world dataset contains features that highly vary in magnitudes, units, and range.
* Normalisation should be performed when the scale of a feature is irrelevant or misleading and not should Normalise when the the scale is meaningful.
* The algorithms which use Euclidean Distance measure are sensitive to Magnitudes. Here feature scaling helps to weigh all the features equally.
* Formally, If a feature in the dataset is big in scale compared to others then in algorithms where Euclidean distance is measured this big scaled feature becomes dominating and needs to be normalized.

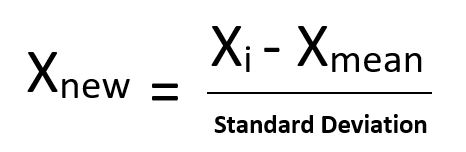
**Examples of Algorithms where Feature Scaling**    
1. K-Means uses the Euclidean distance measure here feature scaling matters.  
2. K-Nearest-Neighbours also require feature scaling.  
3. Principal Component Analysis (PCA): Tries to get the feature with maximum variance, here too feature scaling is required.  
4. Gradient Descent: Calculation speed increase as Theta calculation becomes faster after feature scaling.

* Note: Naive Bayes, Linear Discriminant Analysis, and Tree-Based models are not affected by feature scaling.  
  In Short, any Algorithm which is Not Distance based is Not affected by Feature Scaling.

**Techniques to perform Feature Scaling**  
Consider the two most important ones:

* **Min-Max Normalisation:**This technique re-scales

a feature or observation value with distribution value between 0 and 1.  


* **Standardisation:**It is a very effective technique which re-scales a feature value so that it has distribution with 0 mean value and variance equals to 1.  
  

# here Features - Age and Salary columns

# are taken using slicing

# to handle values with varying magnitude

x = data\_set.iloc[:, 1:3].values

print ("\nOriginal data values : \n",  x)

""" PART 4

    Handling the missing values """

from sklearn import preprocessing

""" MIN MAX SCALER """

min\_max\_scaler = preprocessing.MinMaxScaler(feature\_range=(0, 1))

  # Scaled feature

x\_after\_min\_max\_scaler = min\_max\_scaler.fit\_transform(x)

 print ("\nAfter min max Scaling : \n", x\_after\_min\_max\_scaler)

""" Standardisation """

Standardisation = preprocessing.StandardScaler()

# Scaled feature

x\_after\_Standardisation = Standardisation.fit\_transform(x)

print ("\nAfter Standardisation : \n", x\_after\_Standardisation)