**Data Preprocessing in R**

▷ Data preprocessing is the initial phase of Machine Learning where data is prepared for machine learning models.

▷ This part is crucial and needs to be performed properly and systematically.

▷ If not, we will end up building models that are not accurate for their purpose.

▷ Prerequisites

* To perform data preprocessing, you should have the following:
  + RStudio installed on your computer.
  + Packages ‘caTools’, ‘tidyverse’, readr, dplyr, ggplot2 installed.

### ▷ Steps in data preprocessing

* Step 1: Importing the Dataset
* Step 2: Handling the Missing Data
* Step 3: Encoding Categorical Data.
* Step 4: Splitting the Dataset into the Training and Test sets
* Step 5: Feature Scaling

### Step 1: Importing the dataset

▷ Loading data from csv ﬁle dfdata<-read.csv("data.csv") dfdata

## ▷ You can get a count of the number of records with the nrow() function

nrow(dfdata)

## ▷ You can get other information about dataset using following functions

dim(dfdata) names(dfdata) rownames(dfdata)

▷ It will often be the case that you don’t need all the columns in the data that you import.

▷ The dplyr package includes a select() function that can be used to limit the ﬁelds in the data frame.

▷ Selecting speciﬁc columns

dfdata = select(dfdata,'Country','Age','Purchased') View(dfdata)

▷ You may also want to rename columns to make them more reader friendly or perhaps simplify the names.

▷ The select() function can be used to do this as well.

▷ You simply pass in the new name of the column followed by an equal sign and then the old column name.

▷ Renaming Columns

dfdata = select(dfdata,'country'='Country','age'='Age','Purchased') View(dfdata)

▷ Filtering a dataset

▷ In addition to limiting the columns that are part of a data frame, it’s also common to subset or ﬁlter the rows using a where clause.

▷ Filtering the dataset enables you to focus on a subset of the rows instead of the entire dataset.

▷ The dplyr package includes a ﬁlter() function that supports this capability. dfdata1=ﬁlter(dfdata,Country=='France')

View(dfdata1)

▷ You can also include multiple expressions in a ﬁlter() function. dfdata2=ﬁlter(dfdata,country=='France',age<=40) View(dfdata2)

### Step 2: Handling the missing data

▷ From the dataset, the Age and Salary column report missing data.

▷ Before implementing our machine learning models, this problem needs to be solved, otherwise it will cause a serious problem to our machine learning models.

▷ Therefore, it’s our responsibility to ensure this missing data is eliminated from our dataset using the most appropriate technique.

▷ Here are two techniques we can use to handle missing data:

* Delete the observation reporting the missing data:
  + This technique is suitable when dealing with big datasets and with very few missing values

i.e. deleting one row from a dataset with thousands of observations can not affect the quality of the data.

* + When the dataset reports many missing values, it can be very dangerous to use this technique.
  + Deleting many rows from a dataset can lead to the loss of crucial information contained in the data.
  + To ensure this does not happen, we make use of an appropriate technique that has no harm to the quality of the data.
* Replace the missing data with the average of the feature in which the data is missing:
  + This technique is the best way so far to deal with the missing values.
  + Many statisticians make use of this technique over that of the ﬁrst one.

▷ If you want to check if a value is missing, you must use the function is.na:

is.na(NA)

▷ To get the total number of NAs present in the dataset

sum(is.na(dfdata))

▷ The following function can be used to replace the NA with the column mean for all the numeric columns. The numeric columns are identiﬁed by the sapply(data, is.numeric) function

sapply(dfdata, is.numeric)

▷ If you want just to ignore the NA values, there is often a parameter for specifying this

sum(dfdata$age,na.rm = TRUE)

▷ Let’s start by replacing the missing data in the Age column with the mean of that column.

dfdata$age <- ifelse(is.na(dfdata$age),ave(dfdata$age, FUN = function(x) mean(x, na.rm = TRUE)), dfdata$age)

▷ What does the code above really do?

▷ The above code blocks check for missing values in the age and salary columns and update the missing cells with the column-wise average.

* dfdata$age : Selects the column in the dataset speciﬁed after $
* is.na(dfdata$age): This method returns true for all the cells in the speciﬁed column with no values.
* ave(dfdata$age, FUN = function(x) mean(x, na.rm = ‘TRUE’)): This method calculates the average of the column passed as argument.

### Step 3: Encoding categorical data

▷ Encoding refers to transforming text data into numeric data.

▷ Encoding Categorical data simply means we are transforming data that fall into categories into numeric data.

▷ In our dataset, the Country column is Categorical data with 3 levels i.e. France, Spain, and Germany.

▷ The purchased column is Categorical data as well with 2 categories, i.e. YES and NO.

▷ The machine models we built on our dataset are based on mathematical equations and it only take numbers in those equations.

▷ Keeping texts of a categorical variable in the equation can cause some troubles to the machine learning models and this why we encode those variables.

▷ To transform a categorical variable into numeric, we use the factor() function.

dfdata$country = factor(dfdata$country, levels = c('France','Spain','Germany'), labels = c(1.0, 2.0 , 3.0 ))

▷ Our country names were successfully replaced with numbers.

▷ We do the same for the purchased column.

dfdata$Purchased = factor(dfdata$Purchased,levels = c('No', 'Yes'),labels = c(0, 1))

### Step 4: Splitting the dataset into the training and test set

▷ In machine learning, we split data into two parts:

* Training set: The part of the data that we implement our machine learning model on.
* Test set: The part of the data that we evaluate the performance of our machine learning model on.

▷ Using our dataset, let’s split it into the training and test sets.

▷ To begin with, we ﬁrst load the required library.

▷ Install package caTools install.packages(“caTools”)

library(caTools)# required library for data splition set.seed(123)

split = sample.split(dfdata$Purchased, SplitRatio = 0.8)# returns true if observation goes to the Training set and false if observation goes to the test set.

#Creating the training set and test set separately training\_set = subset(dfdata, split == TRUE) test\_set = subset(dfdata, split == FALSE) training\_set

test\_set

### Step 5: Feature scaling

▷ It’s a common case that in most datasets, features also known as inputs, are not on the same scale.

▷ Many machine learning models are Euclidian distant-based.

▷ It happens that, the features with the large units dominate those with small units when it comes to calculation of the Euclidian distance and it will be as if those features with small units do not exist.

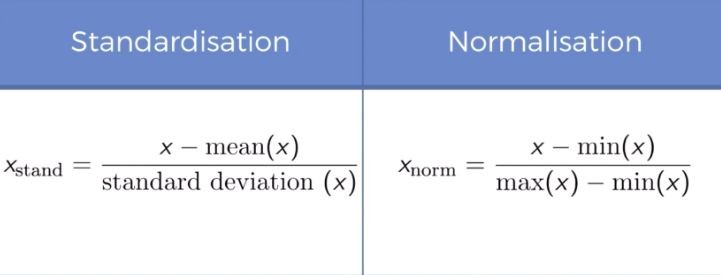
▷ To ensure this does not occur, we need to encode our features so that they all fall in the range between -3 and 3.

▷ There are several ways we can use to scale our features.

▷ The most used one is the standardization and normalization technique.

▷ The normalization technique is used when the data is normally distributed while standardization works with both normally distributed and the data that is not normally distributed.

▷ The formula for these two techniques is shown below.



# Data Preprocessing in R

▷ Now, let’s scale both the training set and test set of our dataset separately.

▷ Here is how we achieve this:

training\_set[, 2] = scale(training\_set[, 2])

test\_set[, 2] = scale(test\_set[, 2]) training\_set

test\_set

Our training and test set were successfully scaled.

▷ Note that in our code we speciﬁed the columns to be scale.

Any Doubts?