**Introduction to Data Preprocessing**

Present 1-2: The Title slide and the Learning Objectives slide. An overview of what we will achieve in this course.

**Learning Objectives**

By the end of this lesson, you will be able to:

* Describe the preprocessing techniques
* Use the python libraries specific to the learned techniques
* In detail understanding through real-life examples
* Explain the different preprocessing techniques and know When, Where and How to use them

**Introduction**

Announce 3: Welcome to the class. Introduce yourself and discuss what the course will cover. Talk about the topics that will be covered in this lesson.

Data preprocessing is the most important step in the process of Data mining. Data mining is the process of identifying underlying patterns in the data. Since data collection is often not performed in a controlled manner, this could result in outliers (e.g., age: – 700), non-sense data combination (e.g., Model: Bike, type: 4Wheeler), missing values, Scale problems, etc.

So, having quality data is the first and foremost step in producing a reliable result. You as a data Scientist may not be spending most of your time in modeling, but in pre-processing. Interestingly, different approaches in pre-processing will lead to different results during training.

Say, you want to make up a good body so you work out daily with a healthy diet and a better lifestyle. Comparing this example with data science paradigm the workout you do is like training a model and the healthy diet and better lifestyle is like preprocessing.

Now consider if you feed junk food and lazy lifestyle even though you do hard work out you may not get fit. So now, you can understand the fact even though you have a powerful machine learning algorithm if your data is not proper, you may not going to get a perfect result. In fact, if you give garbage data, the result will also be garbage.

Considering the above fact, this lesson begins with an explanation of various data preprocessing techniques with examples and then moves on to provide a brief explanation of the same. The above information will be useful to understand the different steps taken to develop a machine learning model.

**Data Preprocessing**

Present 4: Slide introducing the topic

Data Preprocessing is not a single step, it includes multiple steps and handling different problems, and It is not possible to define certain steps and follow the same for all data. So, it is important to read all the preprocessing steps and then use it when and where it is applicable.

**Note**

“In God we trust. All others must bring data.”- [W. Edwards Deming](https://www.google.com/url?q=https://en.wikipedia.org/wiki/W._Edwards_Deming&sa=D&ust=1463700706970000&usg=AFQjCNGZqsO5eD2PW7Zhcjxo-i92jAVjcw), Data Preprocessing is a skill of making the data reliable and it can only be achieved through in detail understanding of the preprocessing concepts.

Discuss 6: Discuss why we data preprocessing with an interactive example with the class

**Why is it important?**

Present 6: This slide explains the importance of data preprocessing in Data Analytics

We will look at a Spam and ham classification problem to understand the importance of preprocessing.

**Email Sample:**

>>>>>> Last Call for Full body checks at very low price <<<<<<< It depends on how fast you book a sitting. This can be anywhere between ₹500 and ₹2000. Visit our website

<http://www.fullbodycheckup.com>......

To unsubscribe yourself from this mailing list, send an email to: [groupname-unsubscribe@egroups.com](mailto:groupname-unsubscribe@egroups.com)

The email contains an Email address at the end, a URL, some special characters, and numbers. This would be hard for the model to understand. Let us do some preprocessing to this so that we can make it easy for the model to understand this for classification.

We can convert the text into lower case and then do URL and Email address Normalization (where we convert the ‘link’ or the ‘email address’ to ‘webaddr’ and ‘emailaddr’, word stemming(process of reducing inflected words to their word stem), Normalizing Numbers, removal of non-words and special characters. After this preprocessing we get the result as:

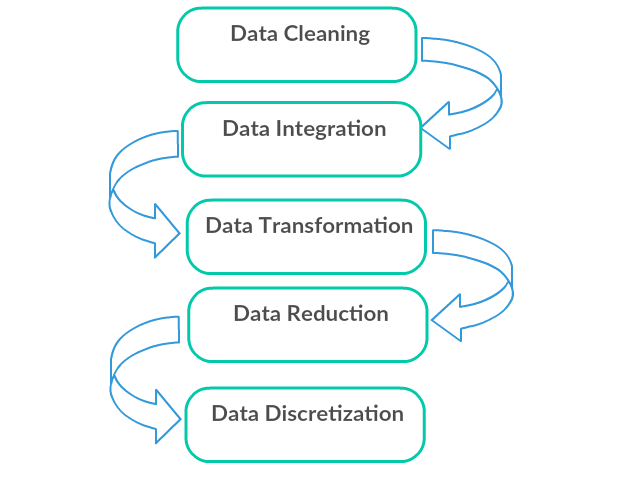
“last call for full body check at very low price it depend on how fast you book a sit can be anywhere between and visit our website webaddress to unsubscribe yourself from mail list send an email to groupname unsubscribe emailaddress”

Data preprocessing refers to the transformation of data before feeding it into the model. It deals with the techniques that are used to convert the raw – unusable data into clean reliable data.

Announce 7: Steps involved in data preprocessing

Present 7: Explaining various steps in data preprocessing

The various steps involved in preprocessing are:

* **Data Cleaning:** Data Cleaning includes processes such as filling the missing values, handling inconsistencies, etc.,
* **Data Integration:** This is the process of putting together data with different representations
* **Data Transformations:** Data Normalization, Encoding etc.,
* **Data Representations:** Presenting a reduced representation of data in the warehouse
* **Data Discretization:** This is the process of representing the continuous variables by bucketing them in terms of interval or range
* Data Cleaning, Integration and Transformation are the most important steps out of all the steps in Data Preprocessing.

**Note**

You can visit the following website to find out how Google and Amazon turn data into a competitive advantage: <https://www.inc.com/jeremy-goldman/how-companies-like-amazon-google-turn-data-into-a-competitive-advantage-how-you-can-too.html>

**Data Problems**

Present 8: Slide introducing the new topic

Discuss 10: Discuss the different forms of data representation, and the importance of representing data in a specific way.

The main objective of machine learning is to build models by interpreting data. In order to do so, it is highly important to feed the data in a way that is readable by the computer. To feed data into a model, it must be represented as a table or matrix of the required dimension, which will be discussed below.

**Tables of Data**

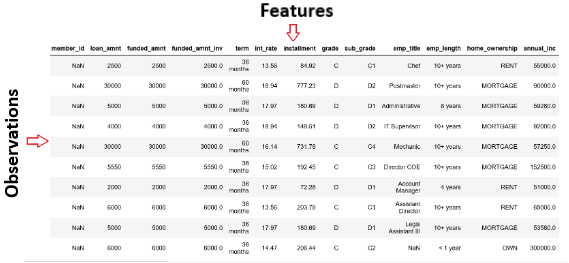
Present 10-11: This slide shows how data can be represented as a table

We will load the data in two dimensional tables called data frames (using pandas in python) i.e., in terms of rows and columns.

**Note**

Learn more about pandas in python <http://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html>.

The table below shows a sample data with lot of impurities. We will use this dataset to predict whether the loan can be issued or not (which we will not be doing here). Hence, in the table, each row embodies a customer information and each column denotes the that specific feature for that customer.

****

**Figure 1.1: An image showing the first 10 instances and first 13 features of the dataset**

The dataset is very large both in terms of number of customers applied and, on their features, (i.e., rows and columns). This is a very good dataset as it provides good scope for data preprocessing techniques. Let us explore the data more and understand its features and target to be predicted.

**Independent and Dependent Variables**

Present 12-19: This slide explains the concept of dependent and independent variables

The dataframe that we use will have variables or features which can be classified into two categories the **Independent variables** (Ideally) or the **predictor variables** which are the catalysts in predicting the **dependent variable** or the **predict and variable** or the **target variable**. The independent variables, as name suggests should be independent from each other if not that is also a kind of impurity, we need to clean.

**Independent Variables**

These are all the features except the **Target Variable** and of size (m,n) where m is the number of observations and n is the number of features respectively, these variables ideally must be normally distributed and should NOT contain:

1. Missing or NULL values
2. Highly Categorical data feature or High Cardinality
3. Outliers
4. Data in different scale
5. Human Error
6. Multi-collinearity
7. Very large independent feature set
8. Sparse data
9. Special characters

**Exercise 1: Loading a sample dataset and creating the features matrix and target matrix.**

**Note**

All exercises and activities will be primarily developed in Jupyter Notebook. It is recommended to keep a separate notebook for different assignments, unless advised not to. Also, to load a sample dataset, the seaborn library will be used, as it displays the data as a table. Other ways to load data will be explained in further sections.

In this exercise, we will be loading the House\_price\_prediction dataset into the pandas dataframe and creating features and target matrices using this dataset.

**Note**

For the exercises and activities within this lesson, you will need to have Python >= 3, seaborn, Jupyter, matplotlib, and Pandas installed on your system.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

dataset = “[USA\_Housing.csv](https://raw.githubusercontent.com/nursnaaz/Inceptez/master/Batch-6/13-Linear%20Regression/USA_Housing.csv)”

df = pd.read\_csv(dataset, header = 0)

The function pd.read\_csv() loads datasets from an online Github repository also. The data from the repository is stored in a variable named df.

1. Print all the column names of the data frame. Using the command df.columns

df.columns

A screenshot of a cell phone

Description automatically generated

1. Print the index names of the data frame. Using the command df.index

df.index

A screenshot of a cell phone

Description automatically generated

We have total 5000 rows starts from index 0 to 5000.

**Note**

How can we give an index name to dataframe instead of index number?

We can just use set\_index() function in pandas to convert a column as an index of rows in dataframe.

Dataframe.set\_index(‘column name’, inplace = True)

1. Make the address column as index and reset it back to the original dataframe.

df.set\_index(‘Address’, inplace=True)

A screenshot of a cell phone

Description automatically generated

**Note**

What is the need of inplace in the above function?

Inplace by default is False. If Inplace is True, whatever operation we do it changes directly the content of a given dataframe without making a copy.

df.reset\_index(inplace=True)

A screenshot of a cell phone

Description automatically generated

**Not**

Explain the concept of Indexing.

There are lot of ways to pull the elements, rows and columns from Data Frame.

Index is like a name given to row and column. Rows and columns both have an index, **row indices are called as index** and for **column indices are called** as **column names** by itself. One can index by row/column **number** or row/column **name**.

1. Retrieve first 2 rows and first 3 columns using a row number and column number. Use iloc function in pandas.

df.iloc[0:4 , 0:3]

A screenshot of a cell phone

Description automatically generated

1. Retrieve first 2 rows and Income and Age column using the name of column. Use loc function in pandas.

df.loc[0:4 , ["Avg. Area Income", "Avg. Area House Age"]]

A screenshot of a cell phone

Description automatically generated

1. Create a variable X to store the independent features. Use the drop()function to include all features but the dependent or the target variable, which in this case is named Price. Then, print out the top 5 instances of the variable.

X = df.drop('Price', axis=1)

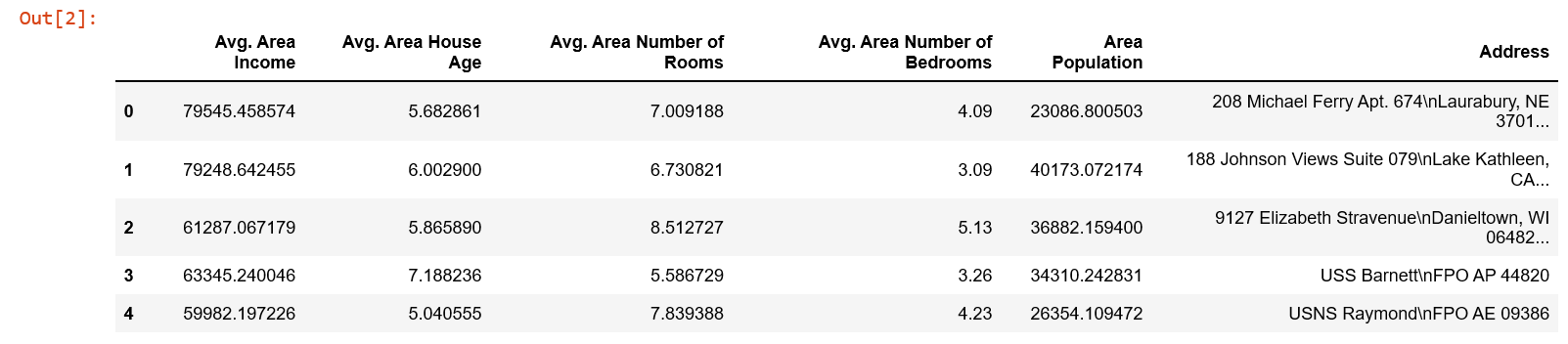
X.head()

**Note**

The default number of instances that will be taken for head is 5, so if you don’t specify the number then it will by default output 5 observations

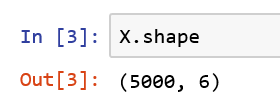
The axis parameter in the above screenshot denotes whether you want to drop the label from rows (axis = 0) or columns (axis = 1).

The printed output should look as shown below:

****

**Figure 1.13: A screenshot showing the first 5 instances of the features matrix**

1. Print the shape of your new created feature matrix using the command X.shape. The first value indicates the number of observations in the dataset (5000), and the second value represents the number of features (6).

****

**Figure 1.14: A screenshot showing the shape of the feature’s matrix**

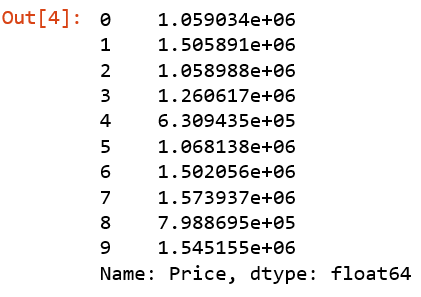
1. Similarly, we will create a variable y that will store the target values. We will just use indexing to grab only the target column. Indexing allows you to access a section of a larger element. In this case, we want to grab the column named Price from the dataframe df.

Then, print out the top 10 values of the variable.

y = df['Price']

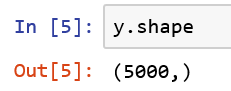
y.head(10)

The printed output should look as below:

****

**Figure 1.15: A screenshot showing the first 10 instances of the target matrix**

1. Print the shape of your new variable using the command: y.shape. The shape should be one-dimensional with length equal to the number of observations (5000) only.

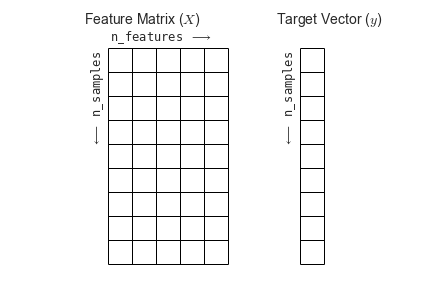
****

**Figure 1.16: A screenshot showing the shape of the target matrix**

Congratulations! You have successfully created the features and target matrices of a dataset.

**Feature and target matrices:**

Present 12-19: This slide explains the concept of Feature and target matrices



**Figure 1.17: A screenshot showing the shape of the target matrix**

Generally, the preferred way to represent data is by using two-dimensional tables, where the rows represent the number of observations, also known as instances, and the columns represent the characteristics of those instances, commonly known as features.

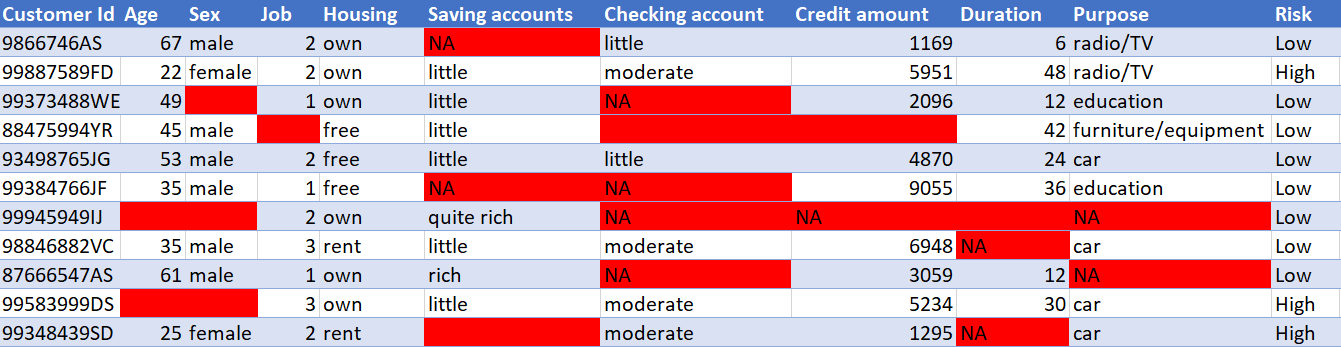
For data problems that require target labels, the data frame needs to be partitioned into a feature’s matrix and a target matrix. The features matrix will contain the values of all features but the target, for each instance, making it a two-dimensional matrix. On the other hand, the target matrix will only contain the value of the target feature for all entries, making it a one-dimensional matrix.

**Missing Values**

Wherever there is a compromise in information delivery of the data we call it missing data. For example, when there are missing values in data, faulty human input and so on. This is one of the most common problems faced by analysts as they can result in the compromise of the statistical inference and thereby derailing from one inference to another. The concept of missing values is important to understand in order to master the skill of successful management and understanding of the data.

**Example:**

This is the data collected for a bank where each row is a separate customer and we have their Gender, Age, Phone no. and so on. Now, we need to predict the Risk, for providing them with the loan. If the risk is ‘Low’ then it means we can issue them the loan, else if the risk is High then there is no return of our amount. Here ‘NA’ or empty record both depicts missing data.



You see this is very crucial for the bank. So, there are lots of important information missing in the data, and the prediction has been made with these. Consider Customer no. 6, His age, sec, checking account balance, credit amount, Duration, purpose all of it missing yet the risk is marked ‘Low’ what would happen if this guy is issued with the loan?

## **Handling of Missing Data**

Present 21-25 : Slide introducing the new topic

Intelligent Handling of Missing data would result in building a robust model capable enough in handling complex tasks. Even though there are many ways to handle missing data, let us now look at some of the ways to take care of it.

### **Removing:**

### These are very simple and commonly used method to handle the missing values. We delete the row if the missing value corresponds to the places in the row, or we delete the column if the column has more than 70%-75% of missing data. Again, the threshold value is not fixed and depends on how much one wishes to fix.

**Advantages of Removing missing data**:

* Removal of missing data would help us build a robust model because removal of rows or columns that give us insufficient information is far better than having them for modelling
* This is the simplest method of handling the missing data to build a highly reliable and accurate model

**Disadvantages**:

* Removing of missing data might result in loss of information from the data
* Removing should only be done if the percentage of missing data is less than 30% of the whole dataset

**Exercise 2: How to remove the missing data**

In this exercise, we will be loading the Banking\_Marketing.csv dataset into the pandas dataframe and handle the missing data properly and make it clean.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

dataset = 'Banking\_Marketing.csv'

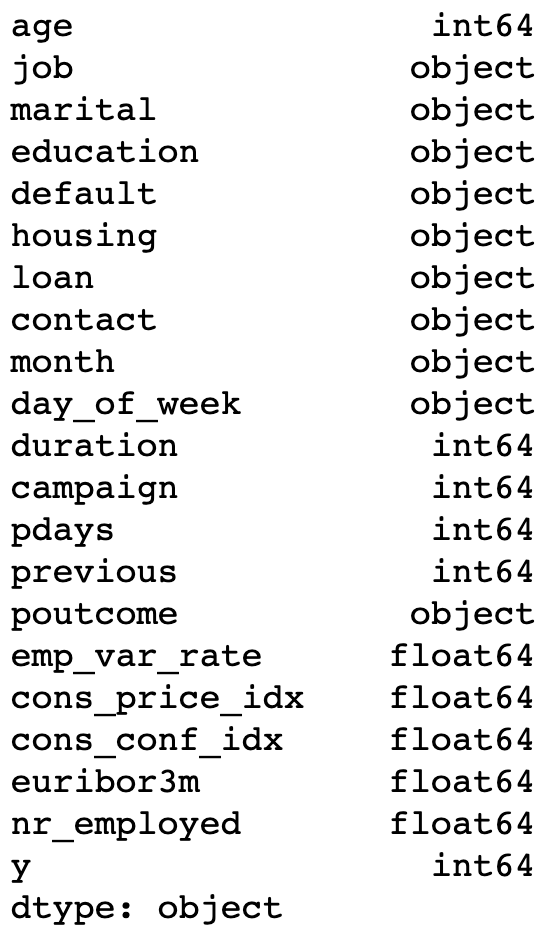
#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Print the datatype of each column. To do so, use dtypes attribute from pandas dataframe.

#finding the data types of each column

df.dtypes



1. Print how many missing values on each column. To do so, use isna() function from pandas dataframe

#finding the data types of each column and checking for null

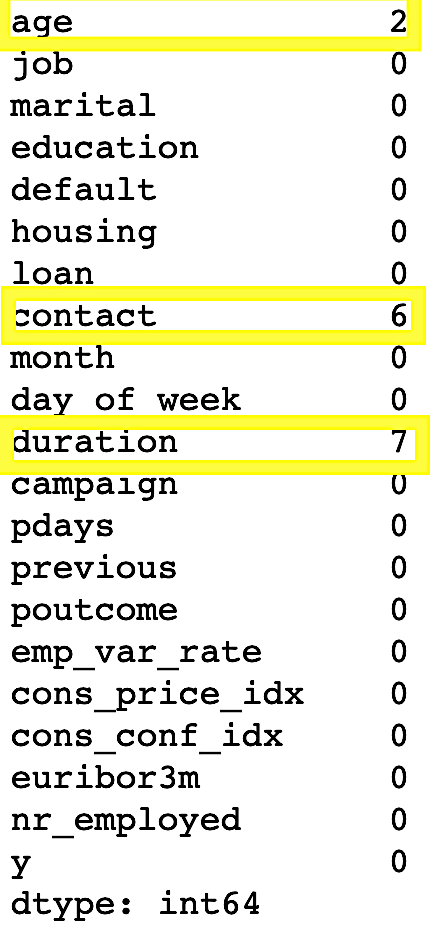
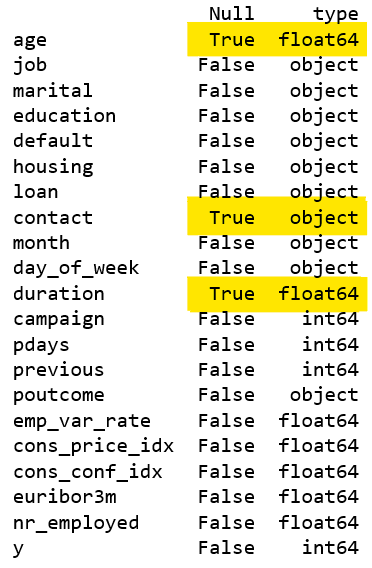
null\_ = df.isna().any()

dtypes = df.dtypes

info = pd.concat([null\_,dtypes],axis = 1,keys = ['Null','type'])

print(info)

df.isna().sum()



From the result there are three columns age’, ‘contact’ and ‘duration’ have a missing data.

There are two NA’s in age column, 6 NA’s in contact and 7 NA’s in duration

1. Remove all the missing rows from the dataframe. To do so, we make use of the function dropna().

#removing Null values

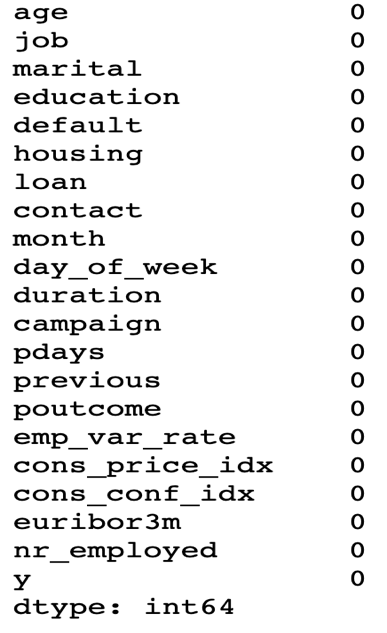
data = data.dropna()

#or

df.dropna( inplace=True)

#Let us check again if NA’s still available

df.isna().sum()



Congratulations! You have successfully removed all missing data from the dataframe.

**Mean/Median/Mode Imputation:**

In the case of **numerical data,** we compute its **mean** or **median** and use the result to replace the missing value and in case of the **categorical data,** we compute its **mode** to replace the missing value.

**Note**

When to use mean and When to use median?

The mean is affected by extreme values. So, if the data contain “outliers”(We will discuss in next section) it will not describe the actual centrality whereas the median is not affected by extreme values

Example:

Salary of 10 employees including a CEO salary (5005000) given below with a missing salary of an employee:

20000, NaN, 40000, 10000, 20000, 15000, 14000, 5005000, 8000, 11000

How can we impute NaN in the above data?

Mean of above data = 570944.44

Which is close to CEO salary, Mean is influenced by extreme values

Median of above data = 15000

This make sense, Median have less or no effect by extreme values

So, the point here is when there is sure about **not having an extreme value** in the data go for **mean** and when there is an uncertainty of **having extreme values** in your data go for **median**

**Advantages of Mean/Median/Mode Imputation:**

* This is a better method as these are statistical approaches in handling the missing data rather than just removing it.
* They also prevent data losses as they just replace the missing values and therefore works well with small datasets also

**Dis-Advantages:**

* They may add bias and variance to the data
* This is not the best among handling the missing values. To have a nearly precise model we can try predicting the missing values

**Exercise 3: How to Impute the missing data**

In this exercise, we will be loading the Banking\_Marketing.csv dataset into the pandas dataframe and handle the missing data properly and make it clean.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

dataset = ' Banking\_Marketing.csv'

#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Impute the numerical data of age column with its mean. To do so, first find the mean of age column using mean() function of pandas dataframe and impute the missing data with its mean using fillna() function

#Find the mean of age column

mean\_age = df.age.mean()

A screenshot of a cell phone

Description automatically generated

#Imputing the missing value in age column with its mean

df.age.fillna(mean\_age,inplace=True)

1. Impute the numerical data of duration column with its median. To do so, first find the median of duration column using median() function of pandas dataframe and impute the missing data with its mean using fillna() function

#Find the median of duration column

median\_duration = df.duration.median()

A screenshot of a cell phone

Description automatically generated

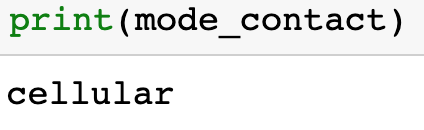
#Imputing the missing value in age column with its mean

df. duration.fillna(median\_duration,inplace=True)

1. Impute the categorical data of contact column with its mode. To do so, first find the mode of contact column using mode() function of pandas dataframe and impute the missing data with its mode using fillna() function

#Find the mode of contact column

mode\_contact = df.contact.mode()[0]



Unlike mean and median. There may be more than one mode in a column, so we just take the first mode we take so we index 0 when we find mode.

#Imputing the missing value in contact column with its mode

df.contact.fillna(mode\_contact,inplace=True)

Congratulations! You have successfully imputed the missing data with different ways and made the data complete and clean.

**Predicting the missing values:**

In this method we replace the missing data with the help of the other available data by using suitable predictive modelling technique (depending on the nature of the data either numeric or categorical). The machine learning modelling will be learnt in coming chapters.

**Advantages of Predicting missing values**:

* This is better method of handling the missing values as the result is going to be based upon the other available data also
* Bias may be eliminated by using this method of imputation

**Dis-Advantages:**

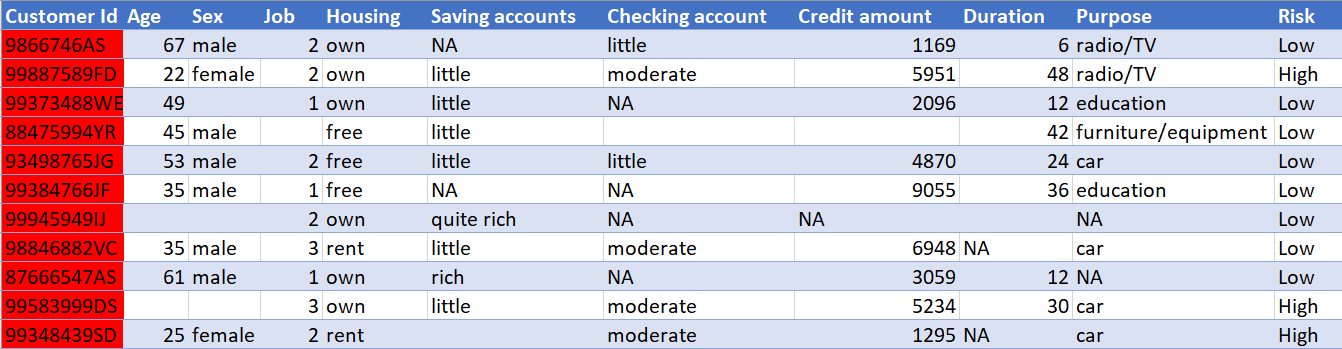
* Even though we try to predict the data statistically, we must keep in mind that this is just a proxy and not the real data

**High Cardinality or Highly Categorical Data Feature:**

A categorical data is one which can be divided into one or more groups or categories. Gender (male or Female), Age Group (<10 yrs., 10-20yrs., 20+yrs.), etc., are some examples of the categorical data. Many of the Machine learning algorithms accept numbers as input, so when our data contains categorical variables like Name, Gender, Age Group or so, we first need to encode them to numbers before giving them into the machine learning model.

**Example:**

This is the same Customer credit dataset taken for the previous example. Here, the ‘Customer Id’ is highly categorical, i.e. each row is unique and will not help the model any way as the but rather just increase the complexity. So, it is very important to handle them before feeding the data into the model.



# **Handling Categorical Data**

Present 26-28: Slide introducing the new topic

There are some algorithms that can work well with categorical data, For example, Decision tree. But majority of Machine learning Algorithms cannot operate directly with categorical data.

These Algorithms require the input and output both to be in numerical form. If the output to be predicted is categorical, then after prediction we convert them back to categorical from numerical.

**Key Challenges with Categorical Data:**

* **High Cardinality:** The Categorical data column in this case might have lots of levels such that their handling becomes very difficult
* **Rare Occurrences:** These data columns might have variables which occur very rarely and thereby would not be significant enough to cause impact on the model
* **Frequent Occurrences:** There might be a category in the data columns that occur a lot of times and thereby with very low variance would fail to make impact on the model.
* **Won’t Fit:** These categorical data left unprocessed won’t fit to our Regression model

**Encoding:**

These are some of the key challenges that occur when you have a categorical feature present in the data. So, to get rid of these problems we will be using Encoding just like the use of Translator in the “Translate Please” example. It is the process in which we convert the categorical variable into its equivalent numerical form.

We will be discussing the different type of Encodings in detail in the next chapter.

**Types of Encoding:**

As discussed in the previous chapter, Encoding is a technique in which we convert the categorical data into its equivalent numerical form. Although there are many ways of handling the categorical data, let’s us now look at three Simple methods of handling it:

### **Replacing**

This is a technique in which we replace the categorical data with numbers that are of with reference with the business use-case. This is a simple replace and does not involve any heavy codes.

**Exercise 4: Simple replacement of a categorical data with number**

In this exercise, we will be loading the Student dataset into the pandas dataframe and simply replace all the categorical data to a number

**Note**

For the exercises and activities within this lesson, you will need to have Python >= 3, seaborn, Jupyter, matplotlib, Numpy and Pandas installed on your system.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

import numpy as np

dataset = “student.csv”

df = pd.read\_csv(dataset, header = 0)

1. Find the categorical column and separate out with different dataframe. To do so, use select\_dtypes() function from pandas dataframe

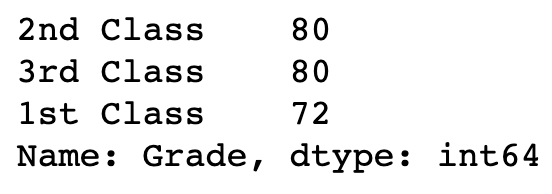
df\_categorical = df.select\_dtypes(exclude=np.number)

A screenshot of a cell phone

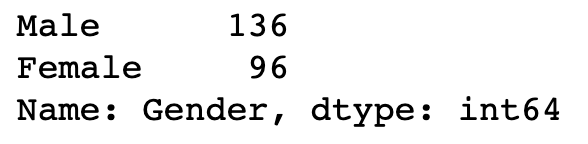
Description automatically generated

1. Find the frequency distribution of each categorical column. To do so, use value\_counts() function on each column.

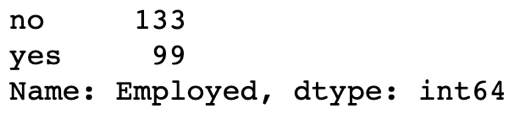
df\_categorical.Grade.value\_counts()



df\_categorical.Gender.value\_counts()



df\_categorical.Employed.value\_counts()



1. Replace the number in Grade column. ‘1st class’ with ‘3’, ‘2nd class’ with ‘2’ and ‘3rd class’ with ‘1’. To do so, use replace() function with the dataframe column

df\_categorical.Grade.replace({"1st Class":3, "2nd Class":2, "3rd Class":1}, inplace= True)

1. Replace the number in Gender column. ‘Male’ with ‘0’ and ‘Female’ with ‘1’. To do so, use replace() function with the dataframe column

df\_categorical.Gender.replace({"yes":1,"no":0}, inplace= True)

1. Replace the number in Employed Employed column. ‘no’ with ‘0’ and ‘yes’ with ‘1’. To do so, use replace() function with the dataframe column

df\_categorical.Employed.replace({"yes":1,"no":0}, inplace = True

### Print the df\_categorical dataframe after replacement

df\_ categorical.head()

### 

Congratulations! You have successfully converted the categorical data to numerical data using simple replacement method manually.

### **From Label Encoding**

This is a technique in which we replace each value in the categorical column with the numbers from 0 to N-1 where N is the level of the categorical data. But this might not be suitable for all the cases as the model might consider this to be the weights assigned to the data. Label encoding is the best method to use for ordinal data.

**Note**

Discuss about types of categorical data.

Nominal data and Ordinal Data.

Nominal data names something without assigning it to order in relation to other numbered objects. An example of nominal data might be a "Male" or "female"

Ordinal data, unlike nominal data, includes some order. ordinal numbers hold in relation to each other in a ranked manner. For example, suppose you receive a survey from your favourite restaurant that asks you to provide feedback on the service you received. You can rank the quality of service as "1" for poor, "2" for below average, "3" for average, "4" for very good and "5" for excellent.

**Exercise 5: Converting the categorical data to numerical using Label Encoding.**

In this exercise, we will be loading the Banking\_Marketing.csv dataset into the pandas dataframe and convert categorical data to a numeric data using Label Encoding.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

dataset = ' Banking\_Marketing.csv'

#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Before doing the encoding, remove all the missing data, To do so, use dropna() function

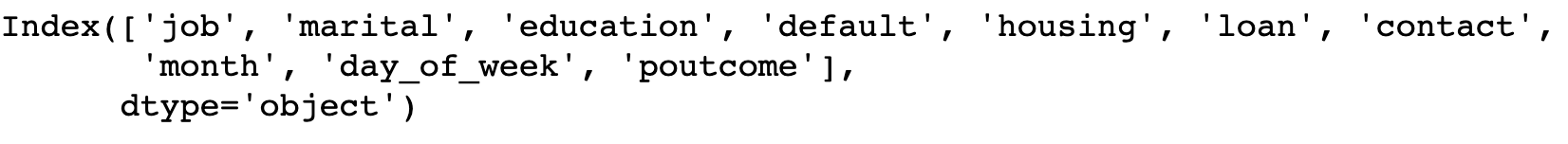
#Dropping Missing Values

df = df.dropna()

1. Select all the columns which is not numeric.

data\_column\_category = df.select\_dtypes(exclude=[np.number]).columns

data\_column\_category



1. Iterate through this category column and convert it to the numeric using Label Encoder. To do so, use import sklearn.preprocessing package and avail the LabelEncoder() class to transform the needed.

#import the LabelEncoder class

from sklearn.preprocessing import LabelEncoder

#Creating the object instance

label\_encoder = LabelEncoder()

for i in data\_column\_category:

df[i] = label\_encoder.fit\_transform(df[i])

print("Label Encoded Data: ")

print(df)

### 

Congratulations! You have successfully converted the categorical data to numerical data using LabelEncoder method.

### **One-Hot Encoding**

This overcomes the problem of weighting the value Improperly by converting each of the level or category to a new column with 0 or 1 (which is True or False) in the columns depicting 0 as absence of value in that field and 1 as presence of value in that field. For every level or category, a new column will be created. We will do one hot encoding mostly to Nominal data.

**Exercise 6: Converting the categorical data to numerical using OneHot Encoding.**

In this exercise, we will be loading the Banking\_Marketing.csv dataset into the pandas dataframe and convert categorical data to a numeric data using Label Encoding.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

dataset = ' Banking\_Marketing.csv'

#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Before doing the encoding, remove all the missing data, To do so, use dropna() function

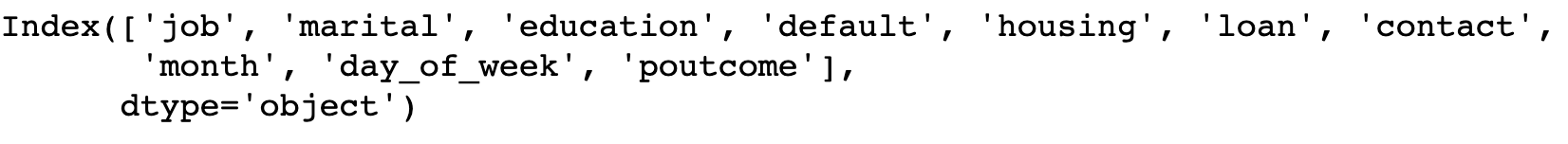
#Dropping Missing Values

df = df.dropna()

1. Select all the columns which is not numeric.

data\_column\_category = df.select\_dtypes(exclude=[np.number]).columns

data\_column\_category



1. Iterate through this category column and convert it to the numeric using OneHot Encoder. To do so, use import sklearn.preprocessing package and avail the OneHotEncoder() class to transform the needed.

**Note**

In order to perform sklearn OnehotEncoder. We need to perform LabelEncoder first.

#performing label encoding

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

for i in data\_column\_category:

df[i] = label\_encoder.fit\_transform(df[i])

#Performing Onehot Encoding

onehot\_encoder = OneHotEncoder(sparse=False)

onehot\_encoded = onehot\_encoder.fit\_transform(df[data\_column\_category])

#Creating a dataframe with encoded data with new column name

onehot\_encoded\_frame = pd.DataFrame(onehot\_encoded, columns = onehot\_encoder.get\_feature\_names(data\_column\_category))

onehot\_encoded\_frame

A close up of a keyboard

Description automatically generated

1. Print all the columns created. To do so, use columns attribute from pandas dataframe

onehot\_encoded\_frame.columns

A close up of text on a white background

Description automatically generated

See for every level or category, a new column is created.

Congratulations! You have successfully converted the categorical data to numerical data using OneHotEncoder method.

**Dummy Variable Encoding**

[One-hot encoding](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) converts the independent feature variables into n variables, whereas [dummy encoding](https://en.wikiversity.org/wiki/Dummy_variable_(statistics)) turns it into n-1 variables. If we have k categorical variables, each of which has n values. One hot encoding ends up with kn variables, whereas dummy encoding results with kn-k variables.

**Exercise 7: Converting the categorical data to numerical using get\_dummies() function.**

In this exercise, we will be loading the Banking\_Marketing.csv dataset into the pandas dataframe and convert categorical data to a numeric data using get\_dummies().

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

dataset = ' Banking\_Marketing.csv'

#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Before doing the encoding, remove all the missing data, To do so, use dropna() function

#Dropping Missing Values

df = df.dropna()

1. Select all the columns which is not numeric.

data\_column\_category = df.select\_dtypes(exclude=[np.number]).columns

data\_column\_category

1. Iterate through this category column and convert it to the numeric using get\_dummies. To do so, just use get\_dummies() function from pandas to transform the needed

#performing Dummy variable encoding

for var in data\_column\_category:

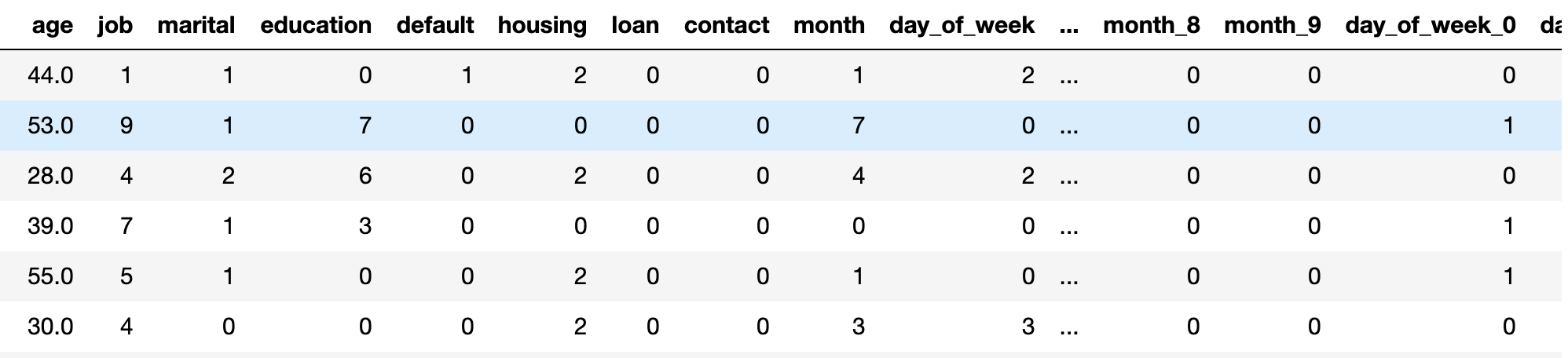
cat\_list='var'+'\_'+var

cat\_list = pd.get\_dummies(df[var], prefix=var)

data1=df.join(cat\_list)

df=data1

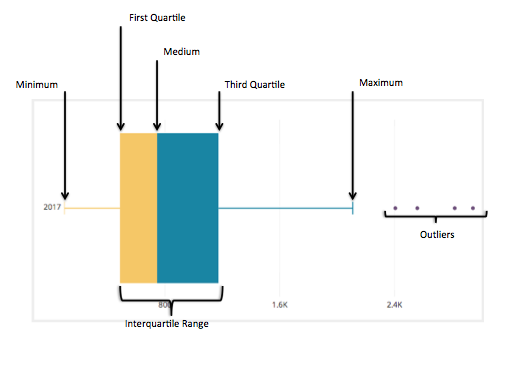
df



Congratulations! You have successfully converted the categorical data to numerical data using get\_dummies method.

**Outliers:**

These are values that are very large or very small with respect to the distribution of the other data. We can find outliers only for the numerical data. Box plots are one good way to find the outliers in the dataset.



**Note**

Outlier is not always a bad data! With the help of business understanding and client interaction you can conclude either to remove or retain the outlier.

Let us learn how to find outliers using a simple example

Consider a sample dataset of temperatures from a place in different time.

71, 70, 73, 70, 70, 59, 70, 72, 71, 320, 71, 69

Step:1 – Sort the data

59, 69, 70, 70, 70, 70, 71, 71, 71, 72, 73, 320

Step:2 – Calculate the median(Q2)

Median is the middle data after sorting.

The middle 2 terms are points 6 and 7 - 70 and 71, respectively. So, the median for data set is the average of these two points: ((70 + 71) / 2), = **70.5**.

Step:3 - Calculate the lower quartile(Q1)

**Q1** is the middle value(median) of the first half of the data set.

First half of the data = 59, 69, 70, 70, 70, 70

Points 3 and 4 of the bottom 6 are both equal to 70

Average is ((70 + 70) / 2), = **70**.

**Q1 = 70**

Step:4 - Calculate the upper quartile(Q3)

**Q1** is the middle value(median) of the second half of the data set.

First half of the data = 71, 71, 71, 72, 73, 320

Points 3 and 4 of the upper 6 are 71, 72

Average is ((71 + 72) / 2), = **71.5**.

**Q3 = 71.5**

Step:5 – Find the Inter Quartile Range (IQR)

IQR = Q3 – Q1

= 71.5 – 70

**IQR = 1.5**

Step:6 – Find Upper and Lower fences

Lower Fence = Q1 – 1.5 (IQR)

Upper Fence = Q3 + 1.5 (IQR)

Lower Fence = 70 – 1.5(1.5) = 67.75

Upper Fence = 71.5 + 1.5(1.5) = 73.75

Boundaries of our fences = **67.75 and 73.75**

**Any data that lie outside the outer fences are considered as a major outliers**

Thus the outliers from our example is **59 and 320**

**Exercise 8: Draw the boxplot and find the outliers and remove it using IQR method and Z-score method**

In this exercise, we will be loading the german\_credit\_data.csv dataset into the pandas dataframe and remove the outliers

**Note**

For the exercises and activities within this lesson, you will need to have Python >= 3, seaborn, Jupyter, matplotlib, seaborn and Pandas installed on your system.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

import numpy as np

# magicfunction essential if running on notebook to make the plot visible in the notebook

%matplotlib inline

import seaborn as sbn

dataset = 'german\_credit\_data.csv'

#reading the data into the dataframe into the object data

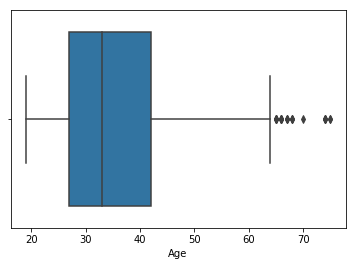
df = pd.read\_csv(dataset, header=0)

1. Plot the boxplot of age column. To do so, use boxplot() function from seaborn library

**Note**

Boxplot use IQR method to display data and outliers (shape of the data) but in order to get list of outlier, we use mathematical formula to retrieve it.

sbn.boxplot(df[‘Age’])



We can see some data points are outliers from the above boxplot

1. Find the outliers of Age column based on IQR method.

Q1 = df ["Age"].quantile(0.25)

Q3 = df["Age"].quantile(0.75)

IQR = Q3 - Q1

A screenshot of a cell phone

Description automatically generated

1. Find the Upper Fence and Lower Fence

Lower\_Fence = Q1 - (1.5 \* IQR)

Upper\_Fence = Q3 + (1.5 \* IQR)

1. Print all the data below the Lower fence and above the Upper fence

df[((df["Age"] < Lower\_Fence) |(df["Age"] > Upper\_Fence))]

A screenshot of a cell phone

Description automatically generated

1. Filter out the outlier data and print only the potential data. To do so, just negate the above result using ~ operator

df[~((df ["Age"] < Lower\_Fence) |(df["Age"] > Upper\_Fence))]

A screenshot of a cell phone

Description automatically generated

1. Find the outliers of Age column based on Z score method. To do so, zscore method from stats package and filter out data with zscore greater than 3

from scipy import stats

import numpy as np

threshold = 3

z = np.abs(stats.zscore(df['Age']))

df[~(z < 3)]

A screenshot of a cell phone

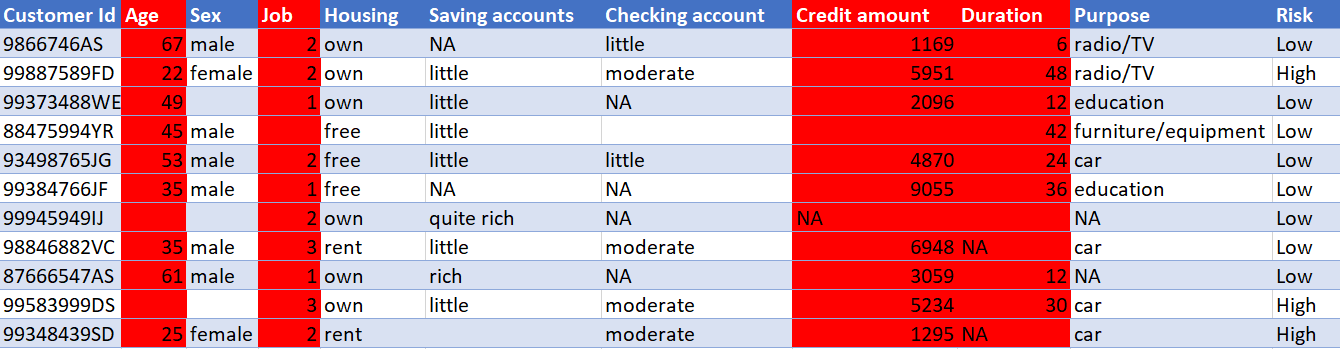
Description automatically generated

Congratulations! You have successfully found the outlier using IQR and Z-score method.

**Data in Different Scales:**

In real life data might be in different magnitude, range or scale. So Algorithms that use distance may not weigh all these in a same way.

Feature Scaling and Normalization are types of Data Transformation techniques which are used to put the features of our data in same scale, magnitude or range. Some features in our data might have the value magnitude to be high by nature such as salary, fees, etc., and might change the outcome of the prediction. This doesn’t mean the data smaller in magnitude is not significant. Thus, so to make our prediction not vary because of different magnitudes of features in our data we do Scaling or Normalization.



Here if we see Age, Job, Credit amount and Duration are at different magnitudes. This is a good real-life example for the data in different scales.

**Feature Scaling and Normalization**

Present 28-29: Slide introducing the new topic

These are two similar but different techniques, as in both cases you are transforming features, but the difference comes only when we know what we transform.

### **Scaling**:

Scaling of data would reduce the interference of magnitude of the feature to a great extent in prediction or inference made in the algorithm, because there are algorithms use distance formula as their parameter to compute the distance between two points and the magnitude of data might affect this as the feature with large magnitude would take a higher weightage

There might be different types of data. For example, one feature might be greater in magnitude but less significant to the inference made by the model and some features might be smaller in magnitude and be of greater significance to the prediction

If left untouched, these algorithms would consider the features greater in magnitude and neglect the features lesser in magnitude as the features with greater magnitude may weigh more in when applied to the distance formula

Example for such Algorithms:

* + - KNN, K-Means, PCA, C- Means, etc.,

So, in order to handle such type of inequalities we transform the range of the Data.

**Standardization:**

**Standardization**, on the other hand, is a scaling technique that transforms the data into a Gaussian distribution with mean equal to 0 and standard deviation equal to 1.

One simple way of standardizing a feature is shown in the equation below:

**C:\Users\nehanair\OneDrive - Packt Publishing Pvt. Ltd\Documents\Titles\Beginning Machine Learning with scikit-learn\Development\Lesson 01\New Styled Final Lessons\Images\Lesson 1_files\image029.png**

**Figure 1.19: The standardization equation**

Here, zi corresponds to the ith standardized value, and x represents all values.

**Exercise 9: Implementing scaling using standard scaler method**

In this exercise, we will be loading the Wholesale customers data.csv dataset into the pandas dataframe and perform the scaling using standard scaler method

**Note**

For the exercises and activities within this lesson, you will need to have Python >= 3, seaborn, Jupyter, matplotlib, and Pandas installed on your system.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

import numpy as np

dataset = ' Wholesale customers data.csv

#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Check if there is missing data available if yes drop the missing data.

#finding the data types of each column and checking for null

null\_ = df.isna().any()

dtypes = df.dtypes

info = pd.concat([null\_,dtypes],axis = 1,keys = ['Null','type'])

print(info)

A screenshot of a cell phone

Description automatically generated

There is no missing data available. Thus no need perform dropna()

1. Perform the standard scaling. To do so, use StandardScaler() class from sklearn.preprocessing and implement fit\_transorm() method

from sklearn import preprocessing

std\_scale = preprocessing.StandardScaler().fit\_transform(df)

scaled\_frame = pd.DataFrame(std\_scale,columns=df.columns)

scaled\_frame

A screenshot of a cell phone

Description automatically generated

Congratulations! You have successfully scaled the data using StandardScaler method.

### **Normalization:**

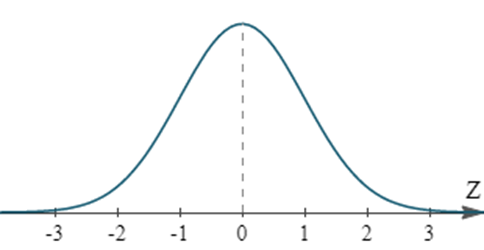
### **Normalization** is a technique in which we alter the data values in the features to fit the values into a normal distribution.

Making the data zero mean and then having unit variance among each feature is what is done in normalization. The data after normalization would be spread roughly equally above and below the mean. Normalization helps the model building by making the train less prone to the scale of the feature value. This also helps for Well-Controlled optimization

The following equation allows you to normalize the values of a feature:

**C:\Users\nehanair\OneDrive - Packt Publishing Pvt. Ltd\Documents\Titles\Beginning Machine Learning with scikit-learn\Development\Lesson 01\New Styled Final Lessons\Images\Lesson 1_files\image026.png**

**Figure 1.20: The normalization equation**



**Figure 1.21: The data is altered and is made to fit into this curve**

Standardization or Min-max Scaling?? Well, that depends on the data and place of application

Image processing and other popular applications make use of min-max scaling while cluster analysis, PCA etc., make use of Standard Scaler

**Exercise 10: Implementing scaling using MinMax scaler method**

In this exercise, we will be loading the Wholesale customers data.csv dataset into the pandas dataframe and perform the scaling using standard scaler method

**Note**

For the exercises and activities within this lesson, you will need to have Python >= 3, seaborn, Jupyter, matplotlib, and Pandas installed on your system.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

import numpy as np

dataset = ' Wholesale customers data.csv

#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Check if there is missing data available if yes drop the missing data.

#finding the data types of each column and checking for null

null\_ = df.isna().any()

dtypes = df.dtypes

info = pd.concat([null\_,dtypes],axis = 1,keys = ['Null','type'])

print(info)

A screenshot of a cell phone

Description automatically generated

There is no missing data available. Thus no need perform dropna()

1. Perform the standard scaling. To do so, use StandardScaler() class from sklearn.preprocessing and implement fit\_transorm() method

from sklearn import preprocessing

minmax\_scale = preprocessing.MinMaxScaler().fit\_transform(df)

scaled\_frame = pd.DataFrame(minmax\_scale,columns=df.columns)

scaled\_frame

A screenshot of a cell phone

Description automatically generated

Congratulations! You have successfully scaled the data using MinMaxScaler method.

**Train, Test and Validation Split**

Present 32: Slide introducing the new topic

Discuss 17: Why do we need to test and validate periodically?



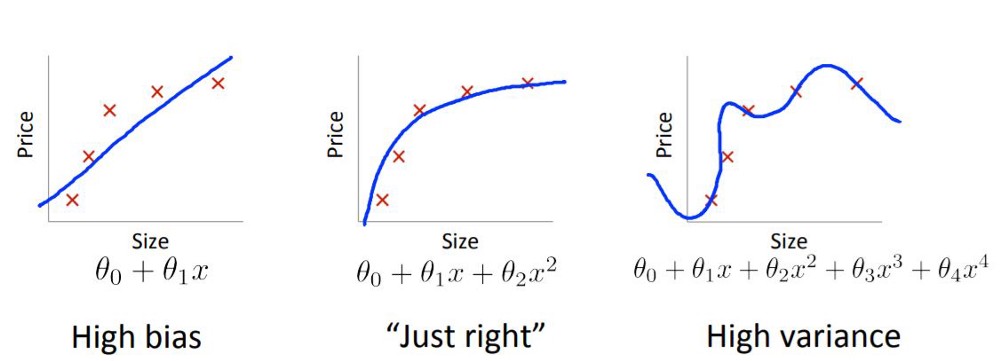
**Figure 1.24: A Figure showing Train, Test and Validation split**

Construction of algorithms that feed on the data and do the predictions are the very nature of Machine learning. Data to these algorithms don’t come from single sources. They come through three namely train, test and validation sets.

But one common problem of machine learning is ‘Overfitting’. We shall now understand this with a classical example.

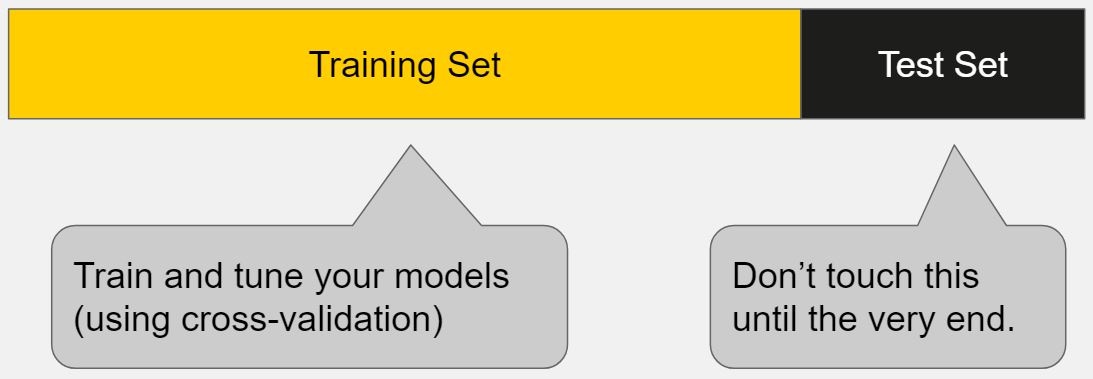
Assume you are building a Linear Regression model for predicting the market price of cars in a country. Let’s say you have a large dataset about the cars and their prices but there are still some more cars whose price needs to be predicted.

When we train our model with the dataset, we want our model to just find that ‘pattern’ within the dataset nothing more. Because if it goes beyond that it will start to memorize the training set.



Here our model will work towards minimizing the error and maximizing the accuracy with the validation set. The first image shows that the model has not learnt enough to predict well in the test set. The third one shows the model has memorized the train dataset. Which means the accuracy score will be 100 with 0 error.

But if we predict on the test the middle model will outperform the third. So that is the reason we keep our train and test separately.



## **Difference between Train, Test and Validation sets**

The machine learning algorithm will feed on data from multiple sets, such as the training set and the validation set before going through the prediction. After which it will perform the prediction in the Test set.

### **Training Dataset:**

This is the part of data that is used to fit the model and thereby “training” the model.

### **Validation Dataset:**

This is the part of the data that is used to give unbiased evaluation of the model with respect to the training data set and thereby paving the way for the improvement of the skill of the model by tuning the model “parameters”.

### **Test Dataset:**

Test is the final and ‘single’ unbiased evaluation of the skill of the model and thereby possess very less capability to mark the uncertainty in the result. The percentage split for each set is not fixed and is purely dependent on situation based on what it demands.

**Exercise 11: How to split the data into train and test?**

In this exercise, we will be loading the USA\_Housing.csv dataset into the pandas dataframe and perform train test split

**Note**

For the exercises and activities within this lesson, you will need to have Python >= 3, seaborn, Jupyter, matplotlib, and Pandas installed on your system.

1. Open a Jupyter Notebook to implement this exercise.

In the cmd or terminal, navigate to the desired path and use the following command:

jupyter notebook

1. Load the dataset into the pandas dataframe. To do so, you first need to import the pandas library, and then, use the function pd.read\_csv(), as shown below:

import pandas as pd

import numpy as np

dataset = 'USA\_Housing.csv'

#reading the data into the dataframe into the object data

df = pd.read\_csv(dataset, header=0)

1. Create a variable X to store the independent features. Use the drop()function to include all features but the dependent or the target variable, which in this case is named Price. Then, print out the top 5 instances of the variable.

X = df.drop('Price', axis=1)

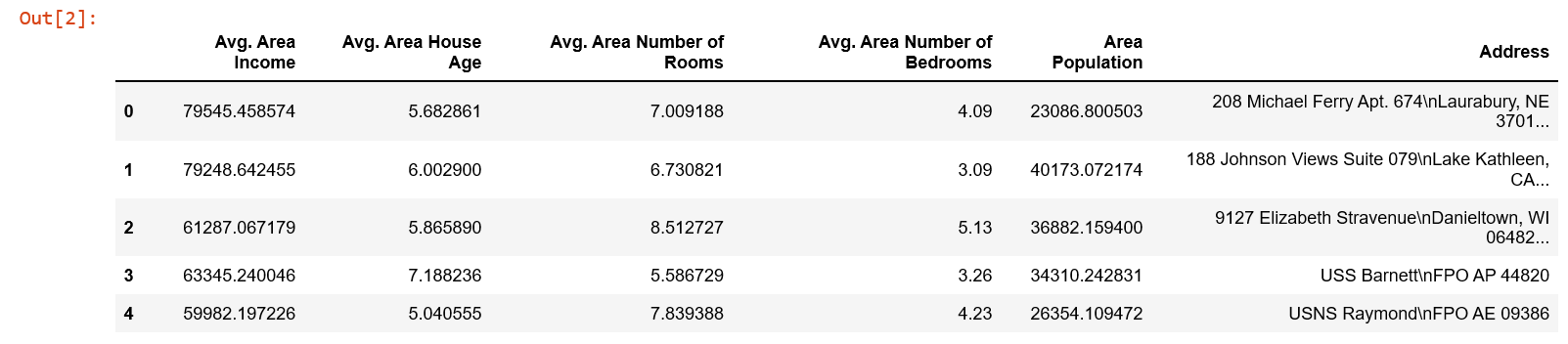
X.head()

**Note**

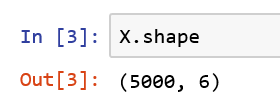
The default number of instances that will be taken for head is 5, so if you don’t specify the number then it will by default output 5 observations

The axis parameter in the above screenshot denotes whether you want to drop the label from rows (axis = 0) or columns (axis = 1).

The printed output should look as shown below:

****

1. Print the shape of your new created feature matrix using the command X.shape. The first value indicates the number of observations in the dataset (5000), and the second value represents the number of features (6).

****

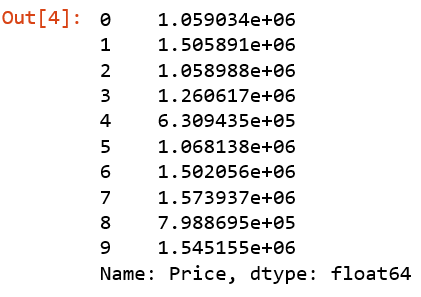
1. Similarly, we will create a variable y that will store the target values. We will just use indexing to grab only the target column. Indexing allows you to access a section of a larger element. In this case, we want to grab the column named Price from the dataframe df.

Then, print out the top 10 values of the variable.

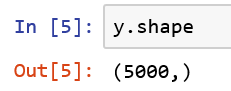
y = df['Price']

y.head(10)

The printed output should look as below:

****

1. Print the shape of your new variable using the command: y.shape. The shape should be one-dimensional with length equal to the number of observations (5000) only.

****

1. Split the train and test with 80:20 proportion. To do so, use train\_test\_split() function from sklearn.model\_selection package.

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

**Note**

What is test\_size?

A floating point value, which defines the size of test data in proportion. If the value is 0.2 then it is a 80-20 split.

What is random\_state?

test\_teain\_split splits the arrays or matrices into train and test subsets in a random way. Each time we run the code without random\_state we will get a different result.

1. Print the shape of X\_train, X\_test, y\_train, y\_test.

print(X\_train.shape)

print("X\_train : ",X\_train.shape)

print("X\_test : ",X\_test.shape)

print("y\_train : ",y\_train.shape)

print("y\_test : ",y\_test.shape)

A close up of a logo

Description automatically generated

Congratulations! You have successfully spitted the train test data.

* 1. **Activity 1:** Pre-Processing using Customer Segment Dataset:

This project is going to be clustering problem. Use the [Customer Segments Dataset (A dataset of shopper spending on groceries)](https://archive.ics.uci.edu/ml/datasets/Wholesale+customers) and then perform:

1. Load the Data

**Attribute Information:**

1) FRESH: annual spending (m.u.) on fresh products (Continuous);   
2) MILK: annual spending (m.u.) on milk products (Continuous);   
3) GROCERY: annual spending (m.u.)on grocery products (Continuous);   
4) FROZEN: annual spending (m.u.)on frozen products (Continuous)   
5) DETERGENTS\_PAPER: annual spending (m.u.) on detergents and paper products (Continuous)   
6) DELICATESSEN: annual spending (m.u.)on and delicatessen products (Continuous);   
7) CHANNEL: customersâ€™ Channel - Horeca (Hotel/Restaurant/CafÃ©) or Retail channel (Nominal)   
8) REGION: customersâ€™ Region â€“ Lisnon, Oporto or Other (Nominal)   
Descriptive Statistics:

1. Understand the Data Features
2. Check for NULL values and their datatypes
3. Remove the missing values (if any)
4. Perform Feature Scaling

**Solution for Activity 1:**

1. Reading the data into the pandas dataframe f

#Reading Data into the pandas Dataframe

import pandas as pd

df = pd.read\_csv("Wholesale customers data.csv")

1. Let’s check for the missing values and type of each feature

#finding the data types of each column and checking for null

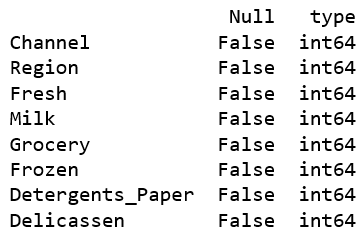
null\_ = df.isna().any()

dtypes = df.dtypes

info = pd.concat([null\_,dtypes],axis = 1,keys = ['Null','type'])

print(info)

OUTPUT:



1. Since there are no missing values in the data set we have nothing to handle
2. Feature Scaling using Standard scaler

#Performng standard scaling

from sklearn import preprocessing

std\_scale = preprocessing.StandardScaler().fit(df)

df\_std = std\_scale.transform(df)

* 1. **Activity 2:** Pre-Processing using Market Prediction Dataset

We will be using the data related to ‘**Market prediction’**.

The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe (1/0) a term deposit (variable y).

This dataset provides the customer information. It includes 41188 records and 21 fields.

1. Load the data set from the link given below into a pandas dataframe : 'https://raw.githubusercontent.com/madmashup/targeted-marketing-predictive-engine/master/banking.csv'.
2. Feature Overview: Understand the Features well.

Input Variables

1 - age (numeric)

2 - job : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')

3 - marital : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced or widowed)

4 - education (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')

5 - default: has credit in default? (categorical: 'no','yes','unknown')

6 - housing: has housing loan? (categorical: 'no','yes','unknown')

7 - loan: has personal loan? (categorical: 'no','yes','unknown')

8 - contact: contact communication type (categorical: 'cellular','telephone')

9 - month: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')

10 - day\_of\_week: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')

11 - duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

12 - campaign: number of contacts performed during this campaign and for this client (numeric, includes last contact)

13 - pdays: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)

14 - previous: number of contacts performed before this campaign and for this client (numeric)

15 - poutcome: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')

16 - emp.var.rate: employment variation rate - (numeric)

17 - cons.price.idx: consumer price index - (numeric)

18 - cons.conf.idx: consumer confidence index - (numeric)

19 - euribor3m: euribor 3 month rate - (numeric)

20 - nr.employed: number of employees - (numeric)

Predict variable (desired target):

y - has the client subscribed a term deposit? (binary: '1','0')

* Check for NULL values and their data types
* Remove the missing values
* The education column of the dataset has many categories and we need to reduce the categories for a better modelling
* Select and perform a suitable encoding method for the data
* Split the data into Train and test
* Perform RFE and select important columns
* Find the feature importance using Random Forest and plot it

**Solution for Activity 2:**

1. Reading the data into the pandas dataframe from Github link

Link = 'https://raw.githubusercontent.com/madmashup/targeted-marketing-predictive-engine/master/banking.csv'

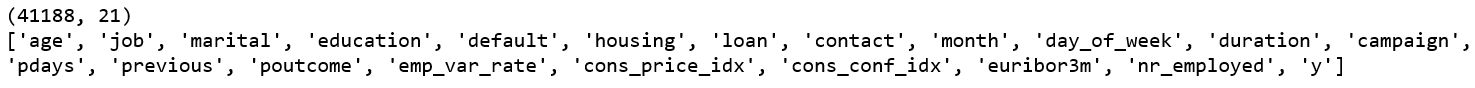
#reading the data into the dataframe into the object data

data = pd.read\_csv(Link, header=0)

#printing data shape and columns

print(data.shape)

print(list(data.columns))

OUTPUT:

1. Let’s check for the missing values and type of each feature

#finding the data types of each column and checking for null

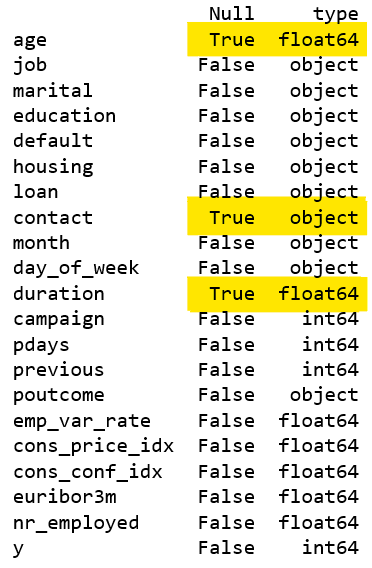
null\_ = df.isna().any()

dtypes = df.dtypes

info = pd.concat([null\_,dtypes],axis = 1,keys = ['Null','type'])

print(info)

OUTPUT:

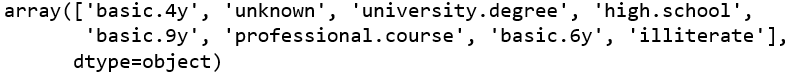


1. Since we have loaded the dataset into the object “data” we will remove the null values from the dataset

#removing Null values

data = data.dropna()

1. The education column of the dataset has many categories and we need to reduce the categories for a better modelling. The education column has the following categories:



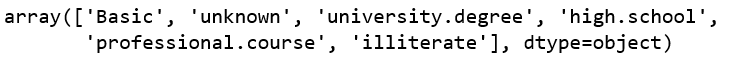
Let us group "basic.4y", "basic.9y" and "basic.6y" together and call them "basic".

data['education']= np.where(data['education'] =='basic.9y', 'Basic', data['education'])

data['education']= np.where(data['education'] =='basic.6y', 'Basic', data['education'])

data['education']= np.where(data['education'] =='basic.4y', 'Basic', data['education']

After grouping, this is the column



1. Lets now handle categorical variables

#Defining a list with all the names of the categorical features in the data

cat\_vars=['job','marital','education','default','housing','loan','contact','month','day\_of\_week','poutcome']

#for every variable in the list getting dummy variable encoded output

for var in cat\_vars:

cat\_list='var'+'\_'+var

cat\_list = pd.get\_dummies(data[var], prefix=var)

data1=data.join(cat\_list)

data=data1

1. Now let us make a list of columns to keep and so that we can just have those columns

Output

#Categorical features

cat\_vars=['job','marital','education','default','housing','loan','contact','month','day\_of\_week','poutcome']

#Numerical features

data\_vars=data.columns.values.tolist()

#creating a list to keep

to\_keep=[i for i in data\_vars if i not in cat\_vars]

#creating a new dataframe called data\_final with features just to keep

data\_final=data[to\_keep]

1. Tran, Test Split

data\_final\_vars=data\_final.columns.values.tolist()

y=['y']

X=[i for i in data\_final\_vars if i not in y]

1. Feature Selection: We make use of the function Recursive Feature Elimination which gives us the ranking of features in ascending order

from sklearn import datasets

from sklearn.feature\_selection import RFE

from sklearn.linear\_model import LogisticRegression

#defining logistic regressing

logreg = LogisticRegression()

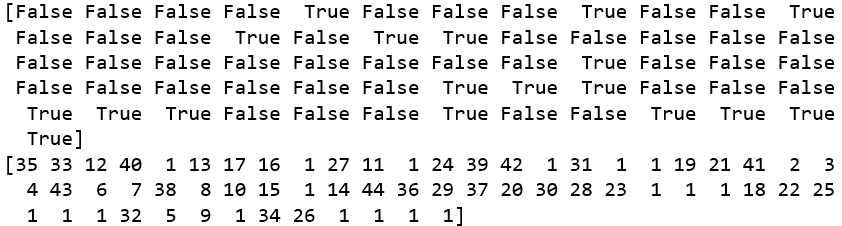
rfe = RFE(logreg, 18)

rfe = rfe.fit(data\_final[X], data\_final[y] )

print(rfe.support\_)

print(rfe.ranking\_)

OUTPUT:



The Recursive Feature Elimination (RFE) has helped us select the following

features: "previous", "euribor3m", "job\_blue-collar", "job\_retired", "job\_services", "job\_student", "default\_no", "month\_aug", "month\_dec", "month\_jul", "month\_nov", "month\_oct", "month\_sep", "day\_of\_week\_fri", "day\_of\_week\_wed", "poutcome\_failure", "poutcome\_nonexistent", "poutcome\_success".

1. Taking just the features ranked 1 and 2 from as per the output of RFE

cols=["previous", "euribor3m", "job\_blue-collar", "job\_retired", "job\_services", "job\_student", "default\_no", "month\_aug", "month\_dec", "month\_jul", "month\_nov", "month\_oct", "month\_sep", "day\_of\_week\_fri", "day\_of\_week\_wed", "poutcome\_failure", "poutcome\_nonexistent", "poutcome\_success"]

#Redefining X and y

X=data\_final[cols]

y=data\_final['y']

1. Feature Importance using Random Forest

from sklearn.ensemble import RandomForestClassifier

#defining the dataframe to an object r

r = pd.DataFrame(columns=['Feature','Importance'])

ncomp = 30

#Random forest object

rf\_loan\_app= RandomForestClassifier(n\_estimators=100,class\_weight="balanced")

#fitting with the model object with the variable X and y

rf\_loan\_app.fit(X, y)

#defining the columns of the dataframe feature and Importance

r['Feature'] = feat\_labels = X.columns

r['Importance'] = rf\_loan\_app.feature\_importances\_

#Setting index as feature

r.set\_index(r['Feature'], inplace=True)

#sorting the important feature as a graph

ax = r.sort\_values('Importance', ascending=False)[:ncomp].plot.bar(width=0.9, legend=False, figsize=(15,8))

ax.set\_ylabel('Relative Importance')

OUTPUT Plot:

