

# CHAPTER 1

## MACHINE LEARNING

### 1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

### 1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that’s in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works

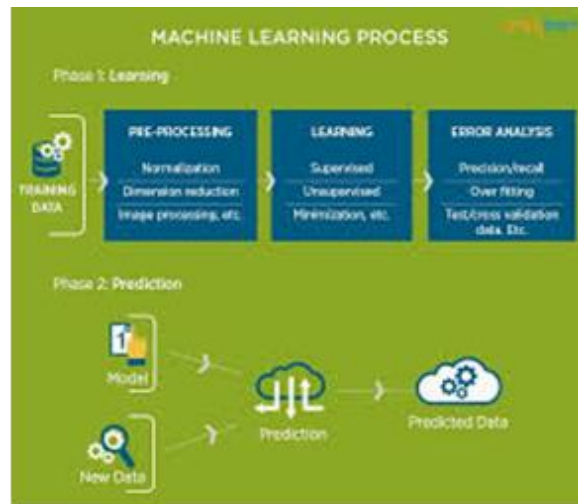


Figure 1 : The Process Flow

### 1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data.

By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

## **1.4 TYPES OF LEARNING ALGORITHMS:**

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

### **1.4.1 Supervised Learning :**

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to “learn” how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

### 1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

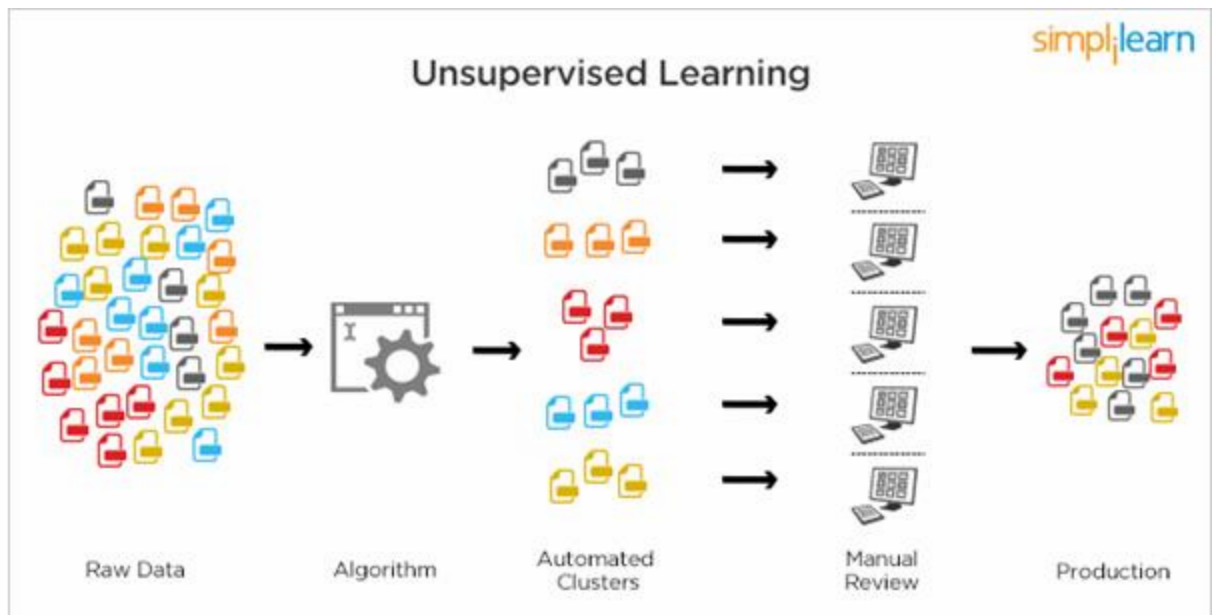


Figure 2 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

### 1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

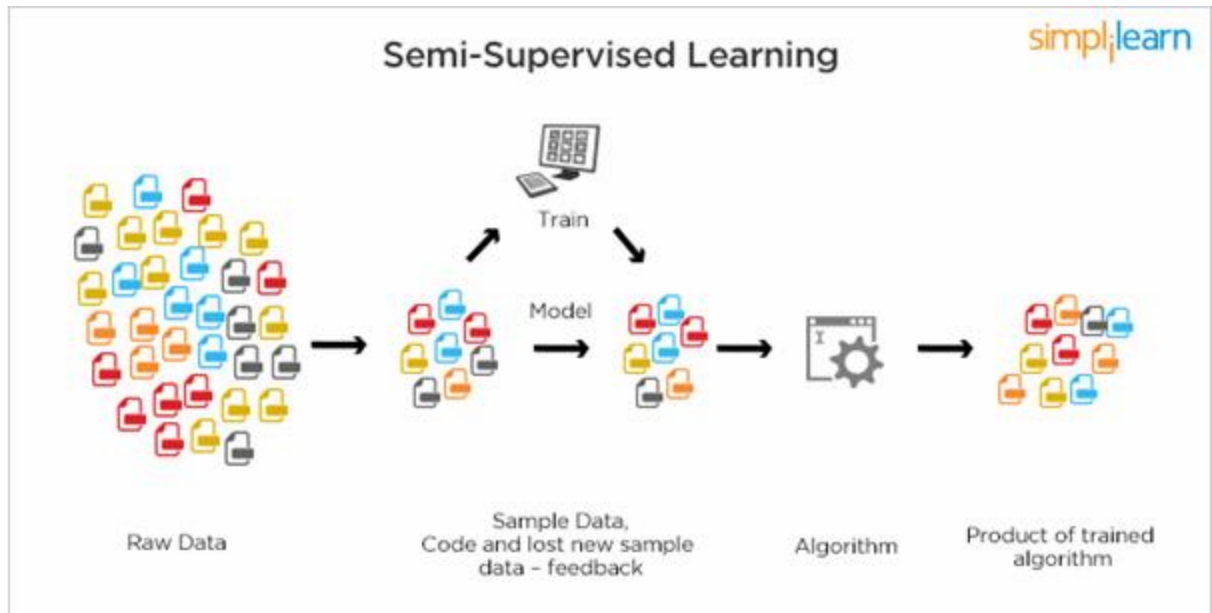


Figure 3 : Semi Supervised Learning

## 1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING:

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovers previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special

types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

## **CHAPTER 2**

### **PYTHON**

Basic programming language used for machine learning is : PYTHON

#### **2.1 INTRODUCTION TO PYHTON:**

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

#### **2.2 HISTORY OF PYTHON:**

- Python was developed by GUIDO VAN ROSSUM in early 1990's
- Its latest version is 3.7 , it is generally called as python3

#### **2.3 FEATURES OF PYTHON:**

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

## **2.4 HOW TO SETUP PYTHON:**

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

### **2.4.1 Installation(using python IDLE):**

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from [www.python.org](http://www.python.org)
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

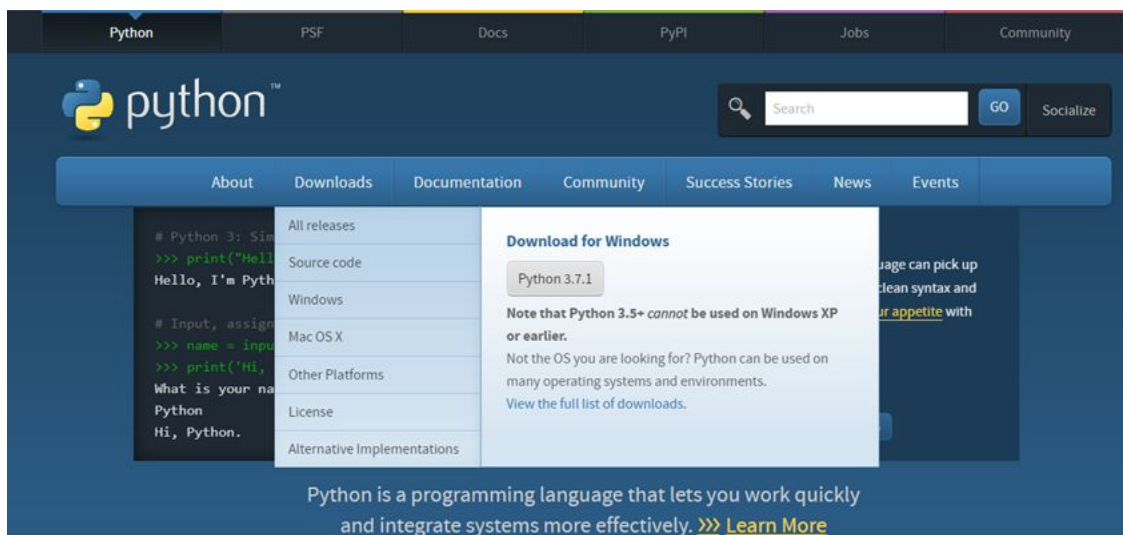


Figure 4 : Python download

## 2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager quickly installs and manages packages.



- In WINDOWS:
- In windows
  - Step 1: Open Anaconda.com/downloads in web browser.
  - Step 2: Download python 3.4 version for (32-bitgraphic installer/64 -bit graphic installer)
  - Step 3: select installation type( all users)
  - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
  - Step 5: Open jupyter notebook ( it opens in default browser)

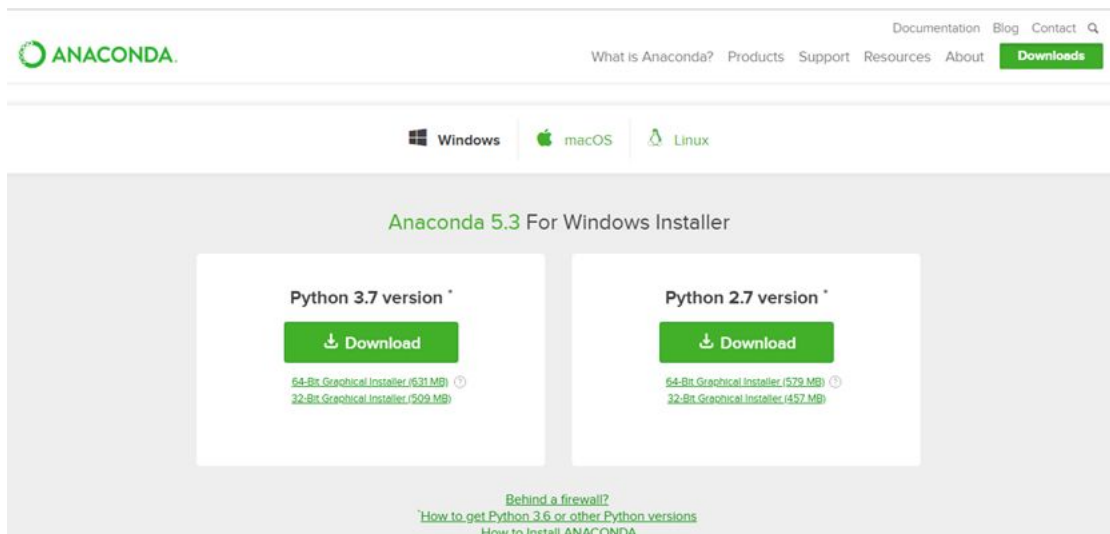


Figure 5 : Anaconda download



Figure 6 : Jupyter notebook

## 2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
  - Numbers
  - Strings
  - Lists

- Tuples
- Dictionary

### **2.5.1 Python Numbers:**

- Number data types store numeric values. Number objects are created when you assign a value to them.
- Python supports four different numerical types – int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

### **2.5.2 Python Strings:**

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (\*) is the repetition operator.

### **2.5.3 Python Lists:**

- Lists are the most versatile of Python's compound data types.
- A list contains items separated by commas and enclosed within square brackets

([]).

- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.
- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.
- The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator.

#### **2.5.4 Python Tuples:**

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example – Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

#### **2.5.5 Python Dictionary:**

- Python's dictionaries are kind of hash table type. They work like associative arrays

or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

- Dictionaries are enclosed by curly braces (`{ }`) and values can be assigned and accessed using square braces (`[]`).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

## **2.6 PYTHON FUNCTION:**

### **2.6.1 Defining a Function:**

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword `def` followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses

The code block within every function starts with a colon (`:`) and is indented. The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

### 2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

## 2.7 PYTHON USING OOP's CONCEPTS:

### 2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- Data member: A class variable or instance variable that holds data associated with a class and its objects.
- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.
- Defining a Class:
  - We define a class in a very similar way how we define a function.
  - Just like a function ,we use parentheses and a colon after the class name(i.e. ()) when we define a class. Similarly, the body of our class is

indented like a functions body is.

```
def my_function():  
    # the details of the  
    # function go here
```

```
class MyClass():  
    # the details of the  
    # class go here
```

Figure 7 : Defining a Class

### 2.7.2 `__init__` method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.
- The init method has a special name that starts and ends with two underscores: `__init__()`.

## CHAPTER 3

### CASE STUDY

#### 3.1 PROBLEM STATEMENT:

To predict the amount of purchases in Retail shop using Machine Learning algorithm called  
MULTIPLE LINEAR REGRESSION

### **3.2 DATA SET:**

The given data set consists of the following parameters:

A - User\_ID

B - Product\_ID

C - Gender

D - Age

E - Occupation

F - City\_Category

G - Stay\_In\_Current\_City\_Years

H - Marital\_Status

I - Product\_Category\_1

J - Product\_Category\_2

K - Product\_Category\_3

L - Purchase

### **3.3 OBJECTIVE OF THE CASE STUDY:**



To get a better understanding and chalking out a plan of action for solution of the client, we have adapted the view point of looking at product categories and for further deep understanding of the problem, we have also considered gender age of the customer and reasoned out the various factors of choice of the products and they purchase , and our primary objective of this case study was to look up the factors which were dampening the sale of products and correlate them to product categories and draft out an outcome report to client regarding the various accepts of a product purchases.

## **CHAPTER 4**

# MODEL BUILDING

## 4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

### 4.1.1 GETTING THE DATASET:

We can get the data set from the database or

we can get the data from client.

### 4.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirement of the algorithm.

## IMPORTING THE LIBRARIES

```
In [ ]: ▶ 1 import pandas as pd  
          2 import numpy as np  
          3 import matplotlib.pyplot as plt  
          4 import seaborn as sns
```

Figure 8 : Importing Libraries

### 4.1.3 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method `read_csv()`. The `read_csv` function reads the entire dataset from a comma separated values file and we can assign it to a `DataFrame` to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be access using the dataframe. Any missing value or NaN value have to be cleaned.

### READING THE DATA-SET

```
[3]: ▶ 1 df=pd.read_csv('datafiles/train.csv')  
      2 df.head()
```

Out[3]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status
0	1000001	P00069042	F	0-17	10	A	2	0
1	1000001	P00248942	F	0-17	10	A	2	0
2	1000001	P00087842	F	0-17	10	A	2	0
3	1000001	P00085442	F	0-17	10	A	2	0
4	1000002	P00285442	M	55+	16	C	4+	0

Figure 9 : Reading the dataset

### 4.1.4 HANDLING MISSING VALUES:

Missing values can be handled in many ways using some inbuilt methods:

(a)`dropna()`

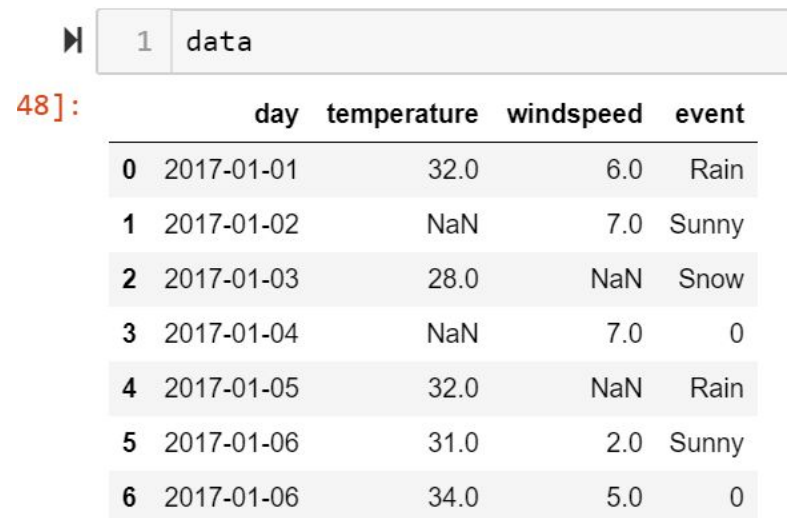
(b)`fillna()`

(c)interpolate()

(d)mean imputation and median imputation

**(a)dropna():**

dropna() is a function which drops all the rows and columns which are having the missing values(i.e. NaN)



The image shows a Jupyter Notebook cell with a code icon and the number 1. The code is `data`. Below the code, the output is displayed as a table with 5 columns: `day`, `temperature`, `windspeed`, and `event`. The table contains 7 rows of data. The first row has values for all columns. The second row has a missing value (NaN) for `temperature`. The third row has a missing value (NaN) for `windspeed`. The fourth row has a missing value (NaN) for `temperature` and a value of 0 for `event`. The fifth row has a missing value (NaN) for `windspeed` and a value of 0 for `event`. The sixth row has values for all columns. The seventh row has values for all columns.

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-01-02	NaN	7.0	Sunny
2	2017-01-03	28.0	NaN	Snow
3	2017-01-04	NaN	7.0	0
4	2017-01-05	32.0	NaN	Rain
5	2017-01-06	31.0	2.0	Sunny
6	2017-01-06	34.0	5.0	0

Figure 10 : data before using dropna()

```

In [52]: 1 data.dropna()
Out[52]:

```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
5	2017-01-06	31.0	2.0	Sunny
6	2017-01-06	34.0	5.0	0

Figure 11 : data after using dropna()

- dropna() function has a parameter called how which works as follows
- if how = 'all' is passed then it drops the rows where all the columns of the particular row are missing
- if how = 'any' is passed then it drops the rows where all the columns of the particular row are missing

#### (b)fillna():

fillna() is a function which replaces all the missing values using different ways.

```
In [45]: 1 data.fillna(0)
```

Out[45]:

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-01-02	0.0	7.0	Sunny
2	2017-01-03	28.0	0.0	Snow
3	2017-01-04	0.0	7.0	0
4	2017-01-05	32.0	0.0	Rain
5	2017-01-06	31.0	2.0	Sunny
6	2017-01-06	34.0	5.0	0

Figure 12 : functioning of fillna(0)

- fillna() also have parameters called method and axis
- if we use method = 'ffill' where ffill is a method called forward fill, which carry forwards the previous row's value
- if we use method = 'bfill' where bfill is a method called backward fill, which carry backward the next row's value
- if we use method = 'ffill' , axis = 'columns' then it carry forwards the previous column's value
- if we use method = 'bfill' , axis = 'columns' then it carry backward the next column's value

### (c)interpolate():

- interpolate() is a function which comes up with a guess value based on the other values in the dataset and fills those guess values in the place of missing values

```
In [51]: 1 data.interpolate()
```

Out[51]:

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-01-02	30.0	7.0	Sunny
2	2017-01-03	28.0	7.0	Snow
3	2017-01-04	30.0	7.0	0
4	2017-01-05	32.0	4.5	Rain
5	2017-01-06	31.0	2.0	Sunny
6	2017-01-06	34.0	5.0	0

Figure 13 : functioning of interpolate()

### (d)mean and median imputation

- mean and median imputation can be performed by using fillna().
- mean imputation calculates the mean for the entire column and replaces the missing values in that column with the calculated mean.
- median imputation calculates the median for the entire column and replaces the missing values in that column with the calculated median.

## Mean Imputation

```
[77]: 1 data.fillna(data.mean())
```

```
Out[77]:
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-01-02	31.4	7.0	Sunny
2	2017-01-03	28.0	5.4	Snow
3	2017-01-04	31.4	7.0	0
4	2017-01-05	32.0	5.4	Rain
5	2017-01-06	31.0	2.0	Sunny
6	2017-01-06	34.0	5.0	0

Figure 14 : mean imputation

## Median Imputation

```
[78]: 1 data.fillna(data.median())
```

```
Out[78]:
```

	day	temperature	windspeed	event
0	2017-01-01	32.0	6.0	Rain
1	2017-01-02	31.4	7.0	Sunny
2	2017-01-03	28.0	5.4	Snow
3	2017-01-04	31.4	7.0	0
4	2017-01-05	32.0	5.4	Rain
5	2017-01-06	31.0	2.0	Sunny
6	2017-01-06	34.0	5.0	0

Figure 15 : median imputation



### 4.1.5 CATEGORICAL DATA:

- Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.
- Categorical Variables are of two types: Nominal and Ordinal
- **Nominal:** The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour
- **Ordinal:** The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less than Ph.D. customer satisfaction survey, high low medium
- Categorical data can be handled by using dummy variables, which are also called as indicator variables.
- Handling categorical data using dummies:

In pandas library we have a method called `get_dummies()` which creates dummy variables for those categorical data in the form of 0's and 1's.

Once these dummies got created we have to concat this dummy set to our dataframe or we can add that dummy set to the dataframe.

```
In [88]: 1 price.head()
```

```
Out[88]:
```

	town	area	price
0	monroe township	2600	550000
1	monroe township	3000	565000
2	monroe township	3200	610000
3	monroe township	3600	680000
4	monroe township	4000	725000

Figure 16 : categorical data

```
In [89]: 1 dummy_set = pd.get_dummies(price.town)
          2 dummy_set
```

```
Out[89]:
```

	monroe township	robinsville	west windsor
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0
5	0	0	1
6	0	0	1
7	0	0	1
8	0	0	1
9	0	1	0
10	0	1	0
11	0	1	0
12	0	1	0

Figure 17 : dummy set for the above data

```
In [90]: 1 merged = pd.concat([price, dummy_set], axis = 1)
```

```
In [91]: 1 merged
```

```
Out[91]:
```

	town	area	price	monroe township	robinsville	west windsor
0	monroe township	2600	550000	1	0	0
1	monroe township	3000	565000	1	0	0
2	monroe township	3200	610000	1	0	0
3	monroe township	3600	680000	1	0	0
4	monroe township	4000	725000	1	0	0
5	west windsor	2600	585000	0	0	1
6	west windsor	2800	615000	0	0	1
7	west windsor	3300	650000	0	0	1
8	west windsor	3600	710000	0	0	1
9	robinsville	2600	575000	0	1	0
10	robinsville	2900	600000	0	1	0
11	robinsville	3100	620000	0	1	0
12	robinsville	3600	695000	0	1	0

Figure 18 : adding the dummy set to dataframe

- Getting dummies using label encoder from scikit learn package

We have a method called label encoder in scikit learn package .we need to import the label encoder method from scikitlearn package and after that we have to fit and transform the data frame to make the categorical data into dummies.

If we use this method to get dummies then in place of categorical data we get the numerical values (0,1,2....)

## Dummy Variables using LabelEncoder

```
In [95]: 1 from sklearn.preprocessing import LabelEncoder  
        2 le = LabelEncoder()
```

Figure 19 : importing the method

```
]: 1 dfle = price  
   2 dfle.town = le.fit_transform(price.town)  
   3 dfle
```

```
t[96]:
```

	town	area	price
0	0	2600	550000
1	0	3000	565000
2	0	3200	610000
3	0	3600	680000
4	0	4000	725000
5	2	2600	585000
6	2	2800	615000
7	2	3300	650000
8	2	3600	710000
9	1	2600	575000
10	1	2900	600000
11	1	3100	620000
12	1	3600	695000

Figure 20 : handling categorical data

## 4.2 TRAINING THE MODEL:

### 4.2.1 Method 1:

- Splitting the data : after the preprocessing is done then the data is split into train and test sets
- In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.
- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt )
- The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%)
- First we need to identify the input and output variables and we need to separate the input set and output set
- In scikit learn library we have a package called model\_selection in which train\_test\_split method is available .we need to import this method
- This method splits the input and output data to train and test based on the percentage specified by the user and assigns them to four different variables(we need to mention the variables)

```

1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size= 0.2, random_state = 0)

```

Figure 21 : importing train\_test\_split

- Then we need to import linear regression method from linear\_model package from scikit learn library
- We need to train the model based on our train set (that we have obtained from splitting)
- Then we have to test the model for the test set ,that is done as follows
  - We have a method called predict , using this method we need to predict the output for input test set and we need to compare the out but with the output test data
  - If the predicted values and the original values are close then we can say that model is trained with good accuracy

```

1 pred=reg.predict(X_test)
2 pred

```

34]: array([10481.08806468, 6353.84743449, 11087.34680048, ..., 12617.06615915, 11490.83221345, 10154.2145273 ])

Figure 22 : Predicting

```
1 np.mean(pred)
79]: 32.764638615780484

1 np.mean(y_test)
30]: 32.246424178823524
```

Figure 23 : comparing the predicted value with the original one

#### 4.2.2 Method 2:

- Using OLS: OLS is Ordinary Least Squares
- This method is available in formula.api package in statsmodel library. So to use this method we have to import statsmodel.formula.api
- In this method we need to first mention the output column names followed by “~” followed by input column names, and this entire thing should be mentioned in double quotes.
- Multiple input and output columns can be mentioned using concat(+) symbol.
- This method gives the summary of the model.
- From this summary we have to observe the R-squared value, R-squared value gives the accuracy mentioned by the client(i.e. if accuracy is 80% then R-squared value must be greater than 0.8)

```

1 import statsmodels.formula.api as smf
2 a = smf.ols("AT ~ Waist", data = df).fit()
3 a.summary()

```

7]: OLS Regression Results

<b>Dep. Variable:</b>	AT	<b>R-squared:</b>	0.670
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.667
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	217.3
<b>Date:</b>	Sat, 08 Jun 2019	<b>Prob (F-statistic):</b>	1.62e-27
<b>Time:</b>	14:29:40	<b>Log-Likelihood:</b>	-534.99
<b>No. Observations:</b>	109	<b>AIC:</b>	1074.
<b>Df Residuals:</b>	107	<b>BIC:</b>	1079.
<b>Df Model:</b>	1		
<b>Covariance Type:</b>	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
<b>Intercept</b>	-215.9815	21.796	-9.909	0.000	-259.190	-172.773
<b>Waist</b>	3.4589	0.235	14.740	0.000	2.994	3.924

Figure 24 : ols and its functioning

- If the model doesn't meet the accuracy we can perform mathematical operations on input and output columns, so that it meets the accuracy



```

1 sqrta = smf.ols("np.sqrt(AT) ~ Waist", data=df).fit()
2 sqrta.summary()

```

]: OLS Regression Results

Dep. Variable:	np.sqrt(AT)	R-squared:	0.710
Model:	OLS	Adj. R-squared:	0.707
Method:	Least Squares	F-statistic:	261.5
Date:	Sat, 08 Jun 2019	Prob (F-statistic):	1.69e-30
Time:	14:34:03	Log-Likelihood:	-202.91
No. Observations:	109	AIC:	409.8
Df Residuals:	107	BIC:	415.2
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-6.8984	1.036	-6.660	0.000	-8.952	-4.845
Waist	0.1803	0.011	16.170	0.000	0.158	0.202

Figure 25 : mathematical operation on input or output

- Correlation: Correlation is a statistical technique that can show whether and how strongly pairs of variables are related. Correlation is described as the analysis which lets us know the association or the absence of the relationship between two variables 'x' and 'y'. It is a statistical measure that represents the strength of the connection between pairs of variables.
- We can also find the column which is effecting the R-Squared value so that we can perform operations on that specific column or we can remove that column , this can be done using pair plot.
- In pair plot we need to find the correlation between two variables and we can do some

operations on that variables

- pairplot is a method which is available in seaborn library, so to use this pairplot method we have to import seaborn library.

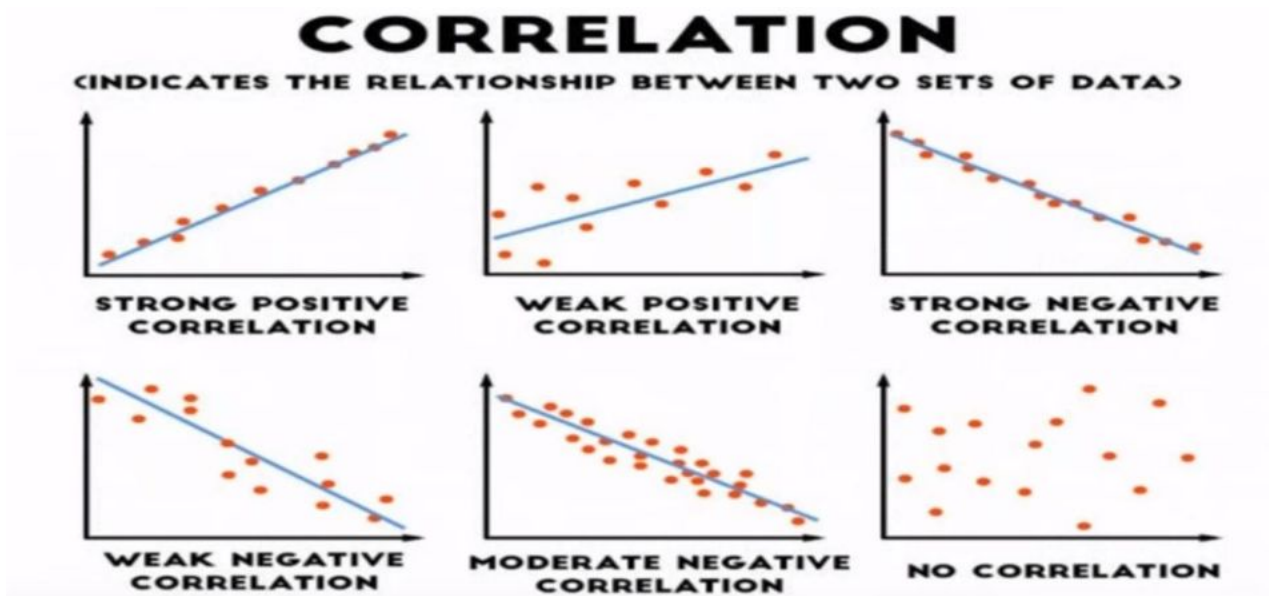


Figure 26 : Correlation

```
1 seaborn.pairplot(df.iloc[:, :])
```

<seaborn.axisgrid.PairGrid at 0x134fcb099e8>

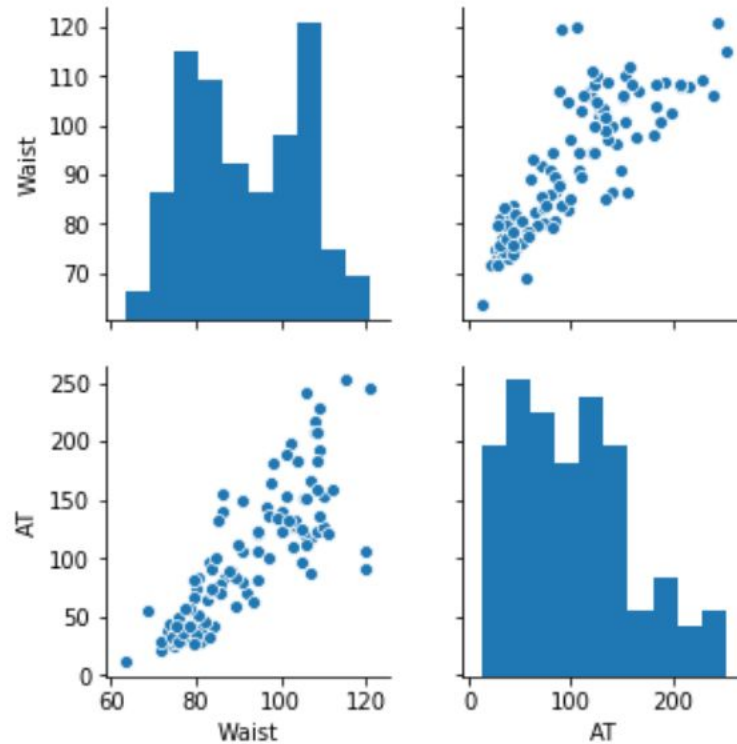


Figure 27 : pairplot

### 4.3 EVALUATING THE CASE STUDY:

- **MULTIPLE LINEAR REGRESSION:** A multiple linear regression model allows us to capture the relationship between multiple feature columns and the target column. Here's what the formula looks like:

$$\hat{y} = a_0 + a_1x_1 + a_2x_2 + \dots + a_nx_n$$

- Importing the required libraries

```

1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns

```

Figure 28 : Importing the libraries

- Reading the Data-Set

```

1 df=pd.read_csv('datafiles/train.csv')
2 df.head()

```

Out[ ]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Cat
0	1000001	P00069042	F	0-17	10	A	2	0	
1	1000001	P00248942	F	0-17	10	A	2	0	
2	1000001	P00087842	F	0-17	10	A	2	0	
3	1000001	P00085442	F	0-17	10	A	2	0	
4	1000002	P00285442	M	55+	16	C	4+	0	

Figure 29 : Reading the Data-Set

- Handling the missing values

- There is a method called `isnull()` which gives the number of missing values in each and every column.
- Using `fillna()` method each and every missing value is replaced by 0.

```

1 df.isnull().sum()
3]: User_ID          0
   Product_ID        0
   Gender            0
   Age              0
   Occupation        0
   City_Category     0
   Stay_In_Current_City_Years  0
   Marital_Status    0
   Product_Category_1  0
   Product_Category_2 173638
   Product_Category_3 383247
   Purchase          0
dtype: int64

```

Figure 30 : Before handling the missing the values

```

1 df1=df.iloc[:,:].apply(lambda x:x.fillna(0))
2 df1.isnull().sum()

In [ ]: User_ID          0
        Product_ID      0
        Gender          0
        Age             0
        Occupation      0
        City_Category   0
        Stay_In_Current_City_Years  0
        Marital_Status  0
        Product_Category_1  0
        Product_Category_2  0
        Product_Category_3  0
        Purchase        0
        dtype: int64

```

Figure 31 : After handling the missing values

- Dealing with categorical data
- Using labelencoder from preprocessing package which is present in scikit learn, we can get dummies in place of categorical data
- Once we get dummies we need to fit and transform that to our dataframe

```

1 from sklearn.preprocessing import LabelEncoder
2 le=LabelEncoder()
3 df1.Gender=le.fit_transform(df1.Gender)
4 df1.head()

```

5]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Cat
0	1000001	P00069042	0	0-17	10	A	2	0	
1	1000001	P00248942	0	0-17	10	A	2	0	
2	1000001	P00087842	0	0-17	10	A	2	0	
3	1000001	P00085442	0	0-17	10	A	2	0	
4	1000002	P00285442	1	55+	16	C	4+	0	

Figure 32 : Getting dummies for Gender column

```

1 df1.Age=le.fit_transform(df1.Age)
2 df1.head()

```

7]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Cat
0	1000001	P00069042	0	0	10	A	2	0	
1	1000001	P00248942	0	0	10	A	2	0	
2	1000001	P00087842	0	0	10	A	2	0	
3	1000001	P00085442	0	0	10	A	2	0	
4	1000002	P00285442	1	6	16	C	4+	0	

Figure 33 : Getting dummies for Age column

```
1 df1.City_Category=le.fit_transform(df1.City_Category)
2 df1.head()
```

8]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Cat
0	1000001	P00069042	0	0	10	0	2	0	
1	1000001	P00248942	0	0	10	0	2	0	
2	1000001	P00087842	0	0	10	0	2	0	
3	1000001	P00085442	0	0	10	0	2	0	
4	1000002	P00285442	1	6	16	2	4+	0	

Figure 34 : Getting dummies for City\_Category column

- Removing the columns which are not required using drop method

```
1 df1=df1.drop(['Stay_In_Current_City_Years', 'User_ID', 'Product_ID'],axis=1)
2 df1.head()
```

9]:

	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3
0	0	0	10	0	0	3	0.0	0.0
1	0	0	10	0	0	1	6.0	14.0
2	0	0	10	0	0	12	0.0	0.0
3	0	0	10	0	0	12	14.0	0.0
4	1	6	16	2	0	8	0.0	0.0

Figure 35 : dropping the columns which are not required



- To get the correlation we have a method called `corr()` which gives the correlation between each and every column

1 `df1.corr()`

	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
Gender	1.000000	-0.004262	0.117291	-0.004515	-0.011603	-0.045594	-0.000954	0.036146	0.060346
Age	-0.004262	1.000000	0.091463	0.123079	0.311738	0.061197	0.018770	-0.007422	0.015839
Occupation	0.117291	0.091463	1.000000	0.034479	0.024280	-0.007618	0.006712	0.012269	0.020833
City_Category	-0.004515	0.123079	0.034479	1.000000	0.039790	-0.014364	0.016003	0.035525	0.061914
Marital_Status	-0.011603	0.311738	0.024280	0.039790	1.000000	0.019888	0.001099	-0.004629	-0.000463
Product_Category_1	-0.045594	0.061197	-0.007618	-0.014364	0.019888	1.000000	-0.067877	-0.385534	-0.343703
Product_Category_2	-0.000954	0.018770	0.006712	0.016003	0.001099	-0.067877	1.000000	0.094234	0.052336
Product_Category_3	0.036146	-0.007422	0.012269	0.035525	-0.004629	-0.385534	0.094234	1.000000	0.094234
Purchase	0.060346	0.015839	0.020833	0.061914	-0.000463	-0.343703	0.052336	0.094234	1.000000

Figure 36 : correlation

#### 4.3.1 Building the model(using the statsmodel library):

- We need to import the `formula.api` package from `statsmodel` library
- By importing this package we can use `ols`(ordinary least squares) to get the summary which gives R-squared value.
- As we have to predict the amount of purchases we have to take `Purchase` column as output and the remaining columns as input
- These output and input columns are separated by “`~`” symbol and the multiple inputs are taken using `concat(+)` symbol.
- This entire thing should be in quotations.

```

1 import statsmodels.formula.api as smf
2 import numpy as np
3 df2=smf.ols("Purchase ~ Gender + Age + Occupation + City_Category+Marital_Status+Product_Category")
4 df2.summary()

```

]: OLS Regression Results

<b>Dep. Variable:</b>	Purchase	<b>R-squared:</b>	0.170
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.170
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	2436.
<b>Date:</b>	Fri, 14 Jun 2019	<b>Prob (F-statistic):</b>	0.00
<b>Time:</b>	15:14:25	<b>Log-Likelihood:</b>	-1.6448e+06
<b>No. Observations:</b>	166821	<b>AIC:</b>	3.290e+06
<b>Df Residuals:</b>	166806	<b>BIC:</b>	3.290e+06
<b>Df Model:</b>	14		
<b>Covariance Type:</b>	nonrobust		

Figure 37 : Getting R-squared value using ols

- Applying mathematical function to improve R-squared value.

```

1 exp=smf.ols("np.log(Purchase) ~ Gender + Age + Occupation + City_Category+Marital_Status+Product_Category")
2 exp.summary()

```

]: OLS Regression Results

<b>Dep. Variable:</b>	np.log(Purchase)	<b>R-squared:</b>	0.219
<b>Model:</b>	OLS	<b>Adj. R-squared:</b>	0.219
<b>Method:</b>	Least Squares	<b>F-statistic:</b>	3348.
<b>Date:</b>	Fri, 14 Jun 2019	<b>Prob (F-statistic):</b>	0.00
<b>Time:</b>	15:24:31	<b>Log-Likelihood:</b>	-1.3006e+05
<b>No. Observations:</b>	166821	<b>AIC:</b>	2.601e+05
<b>Df Residuals:</b>	166806	<b>BIC:</b>	2.603e+05
<b>Df Model:</b>	14		
<b>Covariance Type:</b>	nonrobust		

Figure 38 : Improved R-squared value

### 4.3.2 Building the model (using splitting):

- First we have to retrieve the input and output sets from the given dataset

```
1 X=df.iloc[:,0:8]
2 X.head()
```

	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3
0	0	0	10	0	0	3	0.0	0.0
1	0	0	10	0	0	1	6.0	14.0
2	0	0	10	0	0	12	0.0	0.0
3	0	0	10	0	0	12	14.0	0.0
4	1	6	16	2	0	8	0.0	0.0

Figure 39 : Retrieving the input columns

```
1 y=df.iloc[:,8]
2 y.head()
```

```
0    8370
1   15200
2    1422
3    1057
4    7969
Name: Purchase, dtype: int64
```

Figure 40 : Retrieving the output columns

- Import the train\_test\_split from model\_selection package from scikitlearn library
- Then assigning the output to four different variables, before assigning we have to mention the train size or test size as a parameter to train\_test\_split. Then this method will split according to the size and assigns it to four variables.

```

1 from sklearn.model_selection import train_test_split
2 X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
3 print(X_train.shape)
4 print(y_train.shape)
5 print(X_test.shape)
6 print(y_test.shape)

```

(440054, 8)  
 (440054,)  
 (110014, 8)  
 (110014,)

Figure 41 : Splitting the data

- Import linear regression method which is available in linear\_model package from scikit learn library

```

1 from sklearn.linear_model import LinearRegression
2 reg=LinearRegression()
3 reg.fit(X_train,y_train)

```

In [ ]: LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=1, normalize=False)

Figure 42 : Importing linear regression

- Once the model is built we need to check for accuracy.
- This can be done using predict method which is used to predict the output for input test set, and compare the predicted output with original output test set.

```
1 pred=reg.predict(X_test)
2 pred
: array([10481.08806468,  6353.84743449, 11087.34680048, ...,
        12617.06615915, 11490.83221345, 10154.2145273 ])
```

Figure 43 : Predicting the output

```
1 np.mean(pred)
5]: 9261.993616616272

1 np.mean(y_test)
7]: 9269.135110076899
```

Figure 44 : Comparing with original output

## **CONCLUSION:**

It is concluded after performing thorough Exploratory Data analysis which include Stats models which are computed to get accuracy and also Heat maps which are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case) and its come to point of getting the solution for the problem statement being , that the retail shopkeeper should strategical plan on the marketing in such a way that he could offer complementary products on the purchase of that particular product and if in possibility of bearing expenses offer more quantity of the product at the same price.

## REFERENCES:

[1] <https://www.kaggle.com/semakulapaul/cereals-dataset>

[2] <https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c0257f888676>

[3] [https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning)





