DATA SCIENCE

(Segmentation of customers based on the purchase behaviour)

Summer Internship Report Submitted in partial fulfillment of the

requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Dhundigal Manasa

221710313011

Department of Computer Science and Engineering

GITAM School of Technology GITAM(Deemed to beUniversity)

Hyderabad-502329

June 2020

DECLARATION

I submit this industrial training work entitled **"Segmentation of customers based on the Purchase Behaviour"** to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of **"Bachelor ofTechnology"** in **"Computer Science and Engineering"**. I declare that it was carried out independently by me under the guidance of Mr., Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University Institute for the award of any degree or diploma.

Place: **Dhundigal Manasa**
Hyderabad **RollNo:221710313011**

//certificate

//certificate

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this internship.

I would like to thank respected Dr. N. Siva Prasad, Pro Vice Chancellor, GITAM Hyderabad and Dr. N.Seetharamaiah, Principal, GITAM Hyderabad.

I would like to thank Dr.S.Phani Kumar, Head of the Department of Computer Science and Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a internship report. It helped me a lot to realize what we study for.

I would like to thank the respected faculties Mr. who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Dhundigal Manasa
221710313011

ABSTRACT

In today's fast moving world of marketing from product-orientation to customer-orientation, the management of customer treatment can be seen as a key to achieve revenue growth and profitability. Knowledge of customer behavior can help marketing managers re-evaluate their strategies with the customers and plan to improve and expand their application of the most effective strategies. B2B or business customers are more complex, their buying process is more complicated and their sales value is greater. The business marketers usually prefer to cooperate with fewer but larger buyers than the final consumer marketer. As a business transaction requires more decision making and more professional buying effort than the consumer market does, the efficient relationship with business customers is of paramount importance.

Most customer segmentation approaches based on customer value fail to account for the factor of time and the trend of value changes in their analysis. In this article, we classify customers based on their value using the RFM model and K-means clustering method. Then, an assessment of changes over several periods of time is carried out. The originality of this research lies in its incorporation of time and trend of customer value changes in improving the accuracy of predictions based on the past behavior of customers. For this purpose, we used the POS customer transactions.

Table of Contents:

CHAPTER 1
DATA SCIENCE

1.INFORMATION ABOUT DATA SCIENCE :

1.1 What is Data Science :

Data Science is a blend of various tools, algorithms, and machine learning principles with the goal to discover hidden patterns from the raw data. How is this different from what statisticians have been doing for years? The answer lies in the difference between explaining and predicting.

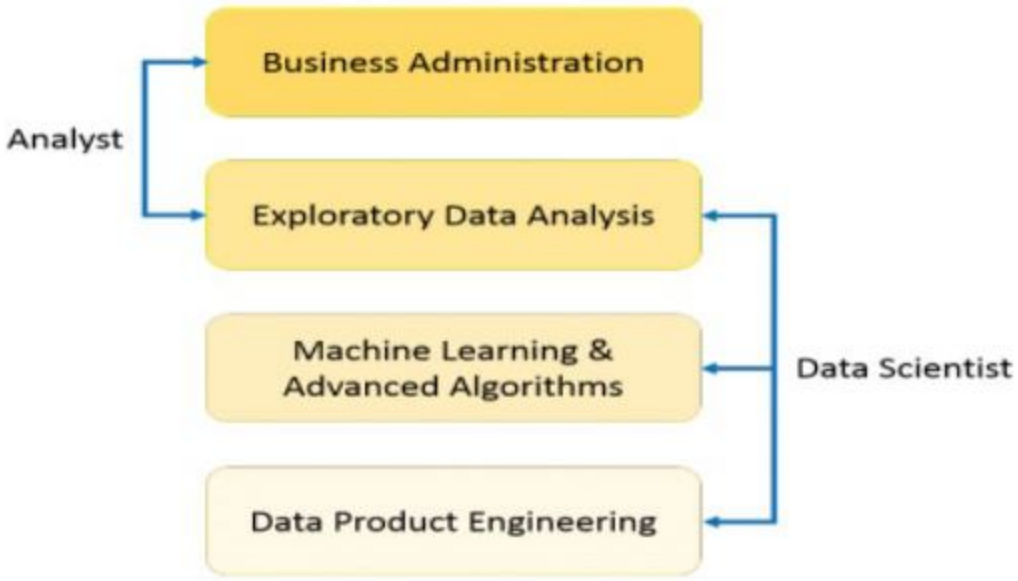


Fig.1.1: Difference between explaining and predicting

As you can see from the above image, a Data Analyst usually explains what is going on by processing the history of the data. On the other hand, Data Scientist not only does exploratory analysis to discover insights from it, but also uses various advanced machine learning algorithms to identify the occurrence of a particular event in the future. A Data Scientist will look at the data from many angles, sometimes angles not known earlier.

1.2 Need of Data Science :

Industries need data to help them make careful decisions. Data Science churns raw data into meaningful insights. Therefore, industries need data science. A Data Scientist is a wizard who knows how to create magic using data. The model will know how to dig out meaningful information with whatever data he comes across. The company requires strong data-driven decisions. The Data Scientist is an expert in various underlying fields of Statistics and Computer Science.

Companies are applying big data and data science to everyday activities to bring value to consumers. Banking institutions are capitalizing on big data to enhance their fraud detection successes. Asset management firms are using big data to predict the likelihood of a security's price moving up or down at a stated time.

Companies such as Netflix mine big data to determine what products to deliver to its users. Netflix also uses algorithms to create personalized recommendations for users based on their viewing history. Data science is evolving at a rapid rate, and its applications will continue to change lives into the future

1.3 Uses of Data Science

Fraud and Risk Detection:

The earliest applications of data science were in Finance. Companies were fed up with bad debts and losses every year. However, they had a lot of data which used to get collected during the initial paperwork while sanctioning loans. They decided to bring in a data scientist in order to rescue them out of losses. Over the years, banking companies learned to divide and conquer data via customer profiling, past expenditures, and other essential variables to analyze the probabilities of risk and default. Moreover, it also helped them to push their banking products based on the customer's purchasing power.

Healthcare:

1.Medical Image Analysis:

Procedures such as detecting tumours, artery stenosis, organ delineation employ various different methods and frameworks like Map Reduce to find optimal parameters for tasks like

lung texture classification. It applies machine learning methods, support vector machines (SVM), content-based medical image indexing, and wavelet analysis for solid texture classification.
2. Genetics & Genomics Data Science applications also enable an advanced level of treatment personalization through research in genetics and genomics. The goal is to understand the impact of the DNA on ourhealth and find individual biological connections between genetics, diseases, and drug response. Data science techniques allow integration of different kinds of data with genomic data in the disease research, which provides a deeper understanding of genetic issues in reactions to particular drugs and diseases. As soon as we acquire reliable personal genome data, we will achieve a deeper understanding of the human DNA. The advanced genetic risk prediction will be a major step towards more individual care.
3. Drug Development The drug discovery process is highly complicated and involves many disciplines. The greatest ideas are often bounded by billions of testing, huge financial and time expenditure. On average, it takes twelve years to make an official submission. Data science applications and machine learning algorithms simplify and shorten this process, adding a perspective to each step from the initial screening of drug compounds to the prediction of the success rate based on the biological factors. Such algorithms can forecast how the compound will act in the body using advanced mathematical modeling and simulations instead of the “lab experiments”. The idea behind the computational drug discovery is to create computer model simulations as a biologically relevant network simplifying the prediction of future outcomes with high accuracy.
4. Virtual assistance for patients and customer supportOptimization of the clinical process builds upon the concept that for many cases it is not actually necessary for patients to visit doctors in person. A mobile application can give a more effective solution by bringing the doctor to the patient instead.The AI-powered mobile apps can provide basic healthcare support, usually as chatbots. You simply describe your symptoms, or ask questions, and then receive key information about your medical condition derived from a wide network linking symptoms to causes. Apps can remind you to take your medicine on time, and if necessary, assign an appointment with a doctor.This approach promotes a healthy lifestyle by encouraging patients to make healthy decisions, saves their time waiting in line for an appointment, and allows doctors to focus on more critical cases.
5.Internet Search Now, this is probably the first thing that strikes your mind when you think Data Science Applications.When we speak of search, we think ‘Google’. Right? But there are many other search engines like Yahoo, Bing, Ask, AOL, and so on. All these search engines (including Google) make use of data science algorithms to deliver the best result for our searched query in a fraction of seconds. Considering the fact that, Google processes more than 20 petabytes of data everyday.

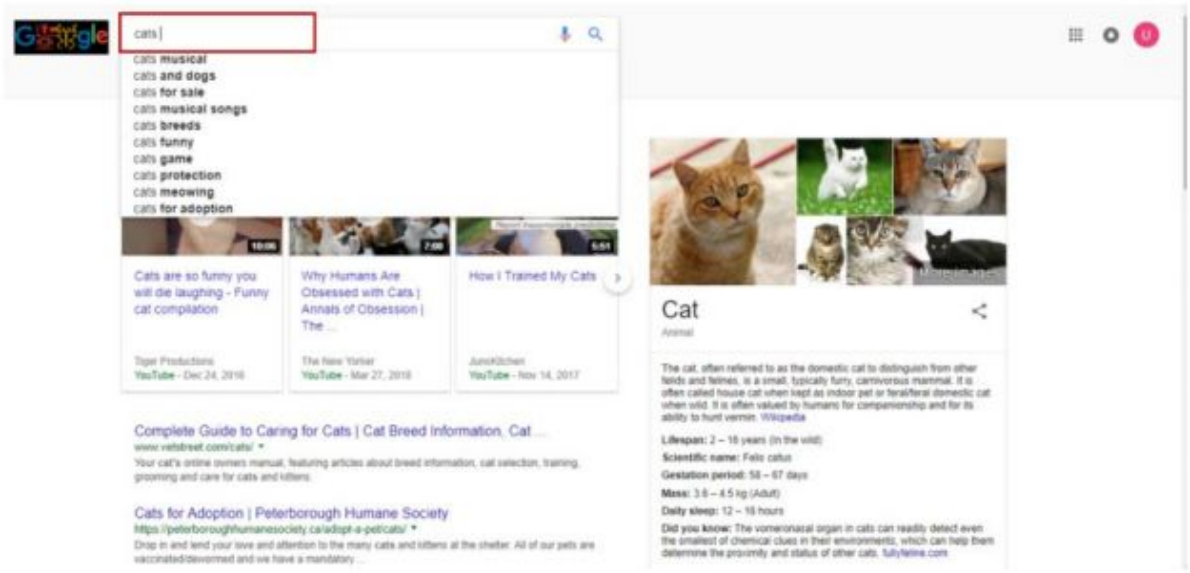


Fig.1.3.1 Internet search

6.Targeted Advertising If you thought Search would have been the biggest of all data science applications, here is a challenger – the entire digital marketing spectrum. Starting from the display banners on various 12 websites to the digital billboards at the airports – almost all of them are decided by using data science algorithms. This is the reason why digital ads have been able to get a lot higher CTR (Call-Through Rate) than traditional advertisements. They can be targeted based on a user's past behavior. This is the reason why you might see ads of Data Science Training Programs while I see an ad of apparels in the same place at the same time.

Fig 1.3.2 Targeted advertising

CHAPTER 2
MACHINE LEARNING

2.Machine Learning

2.1INTRODUCTION :

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).

2.2 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.

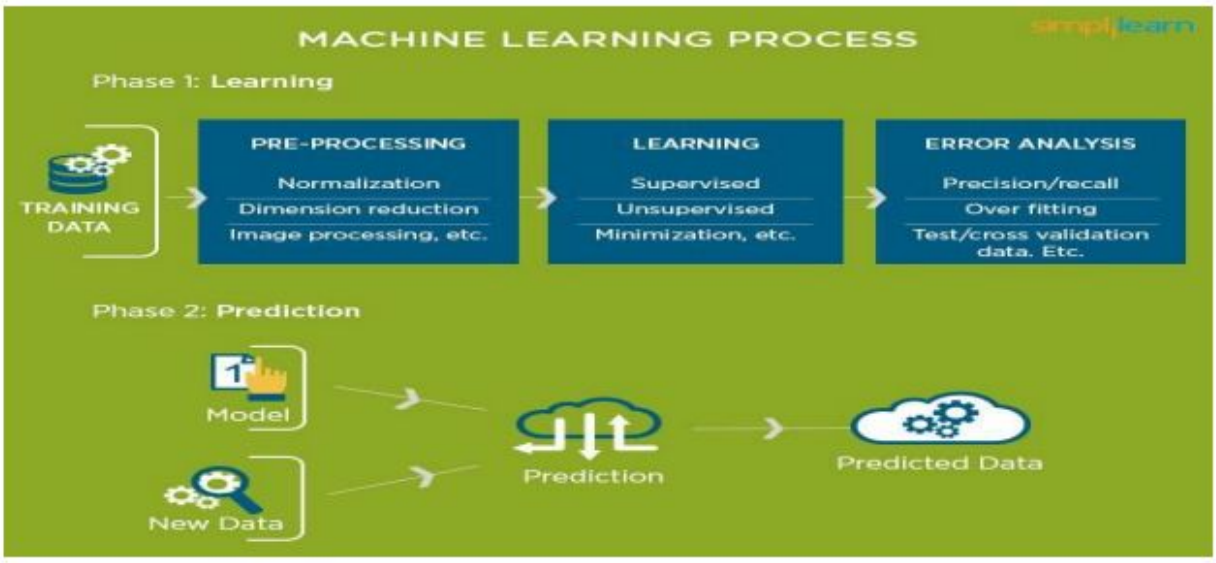


Figure 2.2.1 : The Process Flow

2.3 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 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.

2.4 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.

2.4.1 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.

2.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving 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.

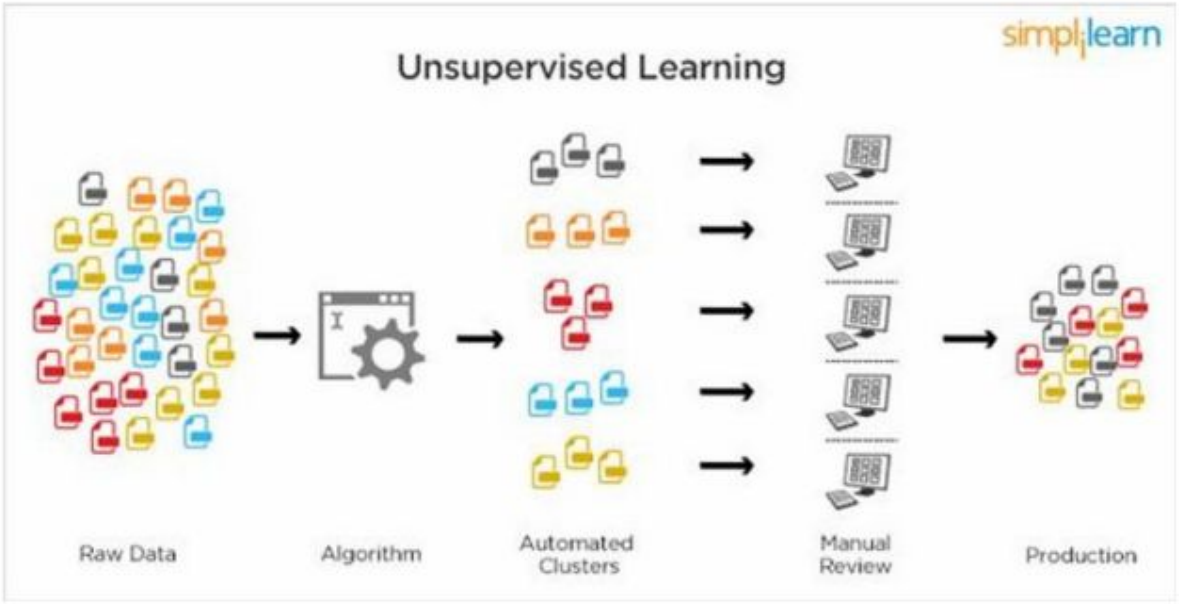


Figure 4.2. 2 : Unsupervised Learning.

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.

2.4.3 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.

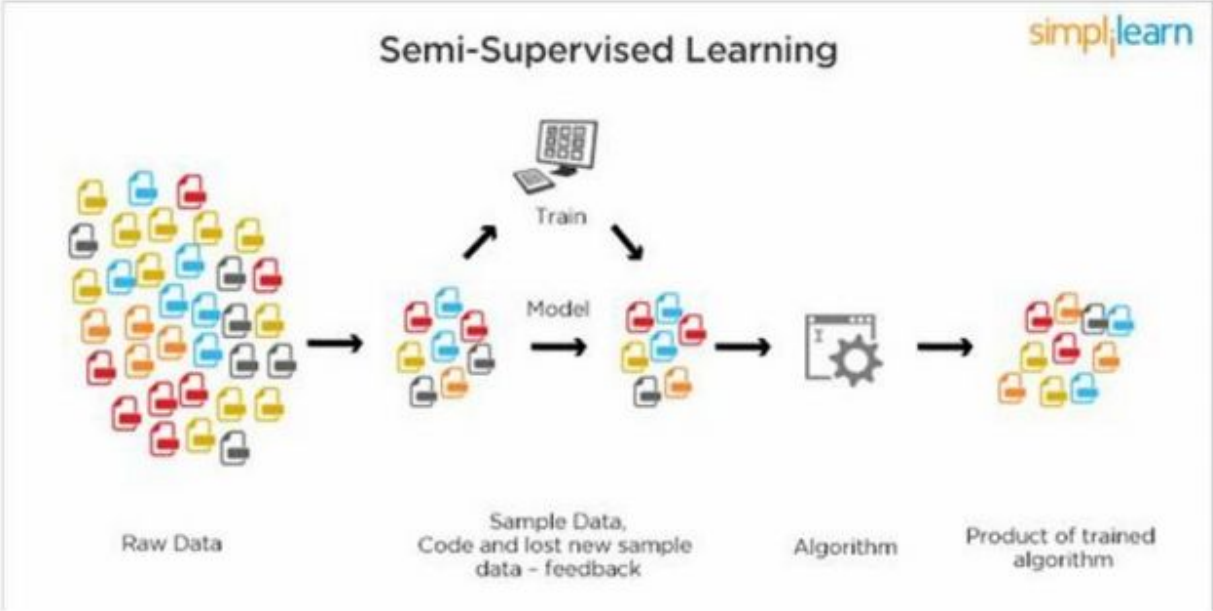


Figure 3 : Semi Supervised Learning

2.5 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 discovered 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 3
PYTHON

3.python

3.1 INFORMATION ABOUT PYTHON :

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.

3.2 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 python

3.3 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.

3.4 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.

3.4.1 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
● 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

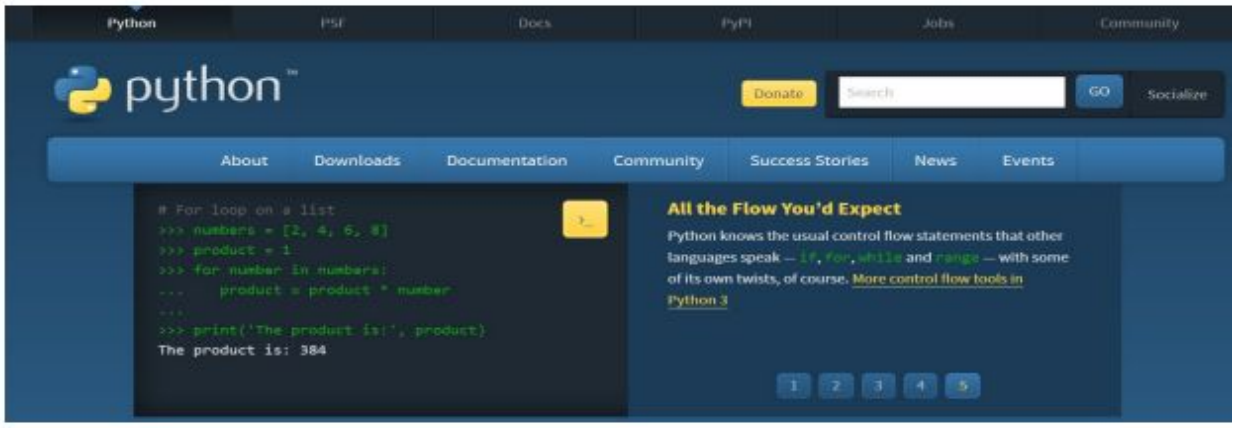


Figure 3.4.1 : Python download

3.4.2 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 that quickly installs and manages packages.

● In WINDOWS:

● Step 1: Open Anaconda.com/downloads in a web browser.
● Step 2: Download python 3.4 version for (32-bits graphic 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)

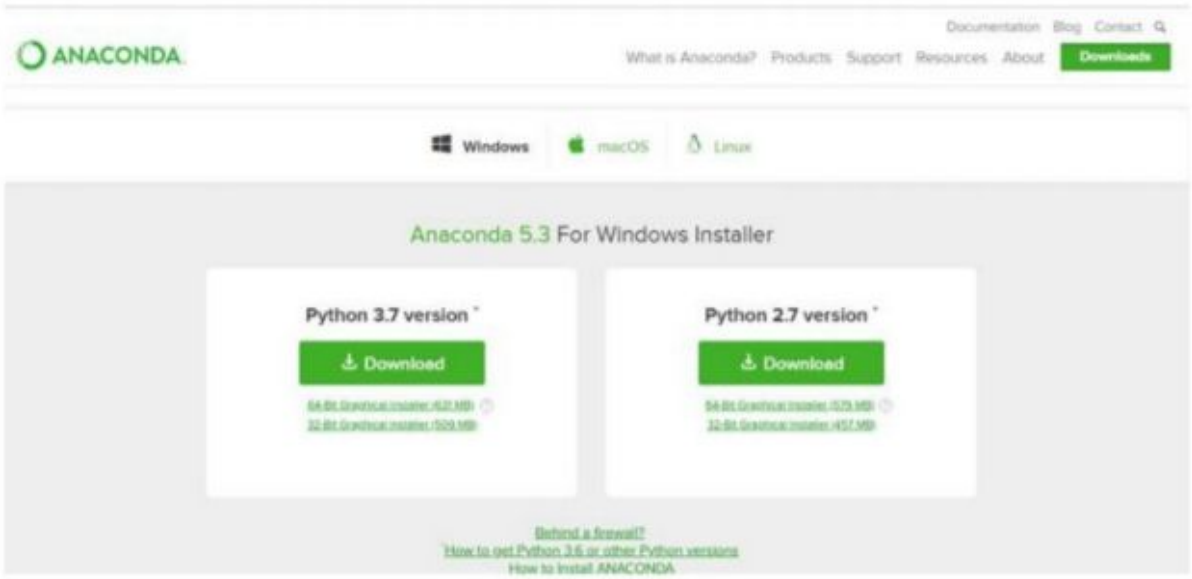


Figure 3.4.2(1) : Anaconda download

Figure 3.4.2(2) : Jupyter notebook

3.5 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

3.5.1 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).

3.5.2 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.

3.5.3 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 types.

● 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.

3.5.4 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.

3.5.5 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.

3.6 PYTHON FUNCTION:

3.6.1 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 returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

3.6.2 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.

3.7 PYTHON USING OOPs CONCEPTS:

3.7.1 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.

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

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

Figure 3.7.1 : Defining a Class

● 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: o We define a class in a very similar way how we define a function. o 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 function body is.
● 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 4

PROJECT NAME(INFORMATION ABOUT THE PROJECT)

4.1 PROJECT REQUIREMENTS

4.1.1DATASET DESCRIPTION

The dataset used for this project consists of 8 columns and their description is given below.

4.1.2Attribute Information:

-->InvoiceNo: 6-digit integer number uniquely assigned to each transaction. The letter ‘c’ at the start of InvoiceNo indicates a cancellation.

-->StockCode: 5-digit integer number uniquely assigned to each distinct product. -->Description: Product name

-->Quantity: The quantities of each product (item) per transaction.

-->InvoiceDate: The day and time when each transaction was generated.

-->UnitPrice: Product price per unit in £ sterling.

-->CustomerID: 5-digit integer number uniquely assigned to each customer.

-->Country: The name of the country where each customer resides.

4.2 Packages Used The packages used are:

- pandas
- numpy
- Seaborn
- matplotlib
- datetime

```
#Importing Libraries
import pandas as pd
import numpy as np

# For Visualisation
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import datetime as dt

# To Scale our data
from sklearn.preprocessing import scale

# To perform KMeans clustering
from sklearn.cluster import KMeans

# To perform Hierarchical clustering
from scipy.cluster.hierarchy import linkage
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import cut_tree
```

Fig 4.2.1 Importing packages

4.3 Versions of the Packages

- pandas version 1.0.5
- numpy version 1.18.5
- seaborn version 0.10.1

```
#Checking the versions of the packages used
print(pd.__version__)
print(np.__version__)
print(sns.__version__)
```

```
1.0.5
1.18.5
0.10.1
```

Fig 4.3.1 Versions

4.4 Project Statement: The aim of the project is to analyze the best set of Customers based on RFM so that the company can target its customers efficiently using k means clustering and hierarchical clustering.

Dataset description:

No.	Attribute Name	Description	Data type
1	InvoiceNumber	6-digit unique number for each transaction	Nominal
2	StockCode	5-digit unique number for each product	Nominal
3	Description	Product Name	Nominal
4	Quantity	Quantity of product per transactions	Numeric
5	InvoiceDate	Invoice Date and Time	Numeric
6	UnitPrice	Product price per unit	Numeric
7	Customer Id	5-digit unique number for each customer	Nominal
8	Country	Country Name	Nominal

Fig 4.4.1:Description of variables

CHAPTER-5

DATA PREPROCESSING

5.Data Preprocessing:

• Pre-processing refers to the transformations applied to our data before feeding it to the algorithm.

• Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

5.1 Reading the dataset:

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 accessed using the dataframe. Any missing value or NaN value has to be cleaned.

```
#Reading the dataset using pandas by G-drive
retail=pd.read_csv("/content/drive/My Drive/Summer Internship/ecommerce.csv",encoding="unicode_escape")
```

Fig 5.1:Reading the data set

5.2 Date time conversion:

```
# converting invoiceDate to datetime
retail['InvoiceDate'] = pd.to_datetime(retail['InvoiceDate'])
```

Fig 5.2:datetime conversion

5.3 CHECKING WITH THE BASIC FUNCTIONS:

Here, the basic functions are considered as head,tail,shape,describe,info,dtype

1.head():It is a function which views the top most rows in our dataset

2.tail():It is a function which views the bottom most rows in our dataset

3.shape():It is a function which views the number of rows and columns in our dataset

4.info():It is a function which views the information in our dataset

5.dtype():It is a function which views our data type format

```
#To view our dataset
retail
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
...	...	...	...	...	...	...	...	...
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France

541909 rows × 8 columns

Fig 5.3.1:view of our dataset

```
#Sanity Check
retail.shape#checking the number of rows and columns present
retail.describe()
retail.info()#checking the information about the data present
retail.shape
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    541909 non-null  object
 1   StockCode    541909 non-null  object
 2   Description  540455 non-null  object
 3   Quantity     541909 non-null  int64
 4   InvoiceDate  541909 non-null  datetime64[ns]
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object
dtypes: datetime64[ns](1), float64(2), int64(1), object(4)
memory usage: 33.1+ MB
(541909, 8)
```

Fig 5.3.2:Sanity check

5.4 HANDLING MISSING VALUES AND DUPLICATE VALUES

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

1. dropna()

2. fillna()

3. interpolate()

4. mean imputation and median imputation .

1. dropna():

● dropna() is a function which drops all the rows and columns which are having the missing values(i.e. NaN). dropna() function has a parameter called how which works as follows:

2. fillna():

● fillna() is a function which replaces all the missing values using different ways fillna() also has parameters called method and axis.

3. 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 .

4. 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.

Missing values can be checked using isna() or isnull() functions which returns the output in a boolean format. Total number of missing values in each column can be calculated using isna().sum() or isnull().sum().

```
#to check whether any null values are present or not
retail.isnull().values.any()
retail.isnull().values.sum()#to check the count of null values values
retail.isnull().sum()*100/retail.shape[0]
```

```
InvoiceNo       0.000000
StockCode       0.000000
Description     0.268311
Quantity        0.000000
InvoiceDate     0.000000
UnitPrice       0.000000
CustomerID     24.926694
Country         0.000000
dtype: float64
```

Fig 5.4.1:All null values

```
#visualizing the missing values by using heatmap
sns.heatmap(retail.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c4b4b50b8>
```

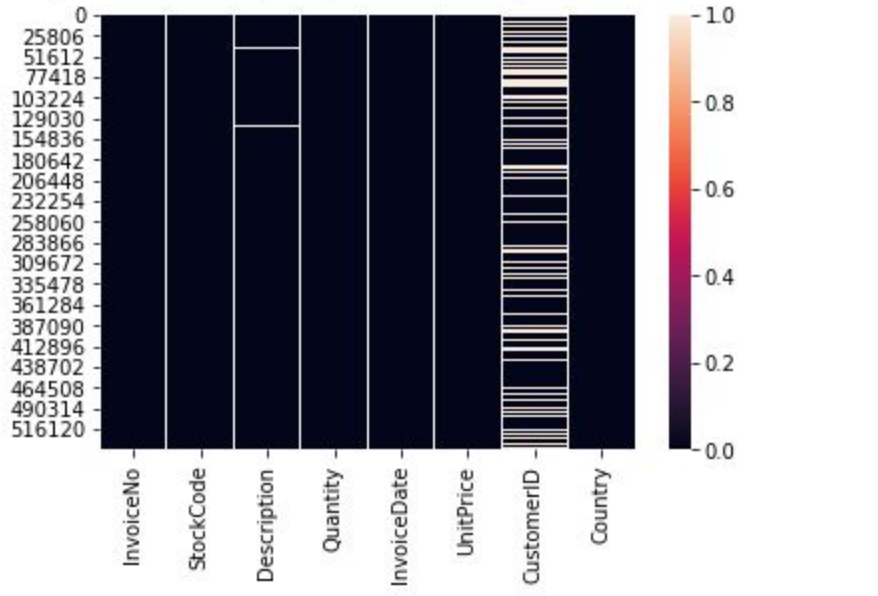


Fig 5.4.2:Heatmap for null values

Here , Whitespaces indicates the number of null values.

```
#Plotting bar graph to know the missing values
plt.figure(figsize=(5, 5))
retail.isnull().mean(axis=0).plot.barh()
plt.title("Ratio of missing values per columns")
```

```
Text(0.5, 1.0, 'Ratio of missing values per columns')
```

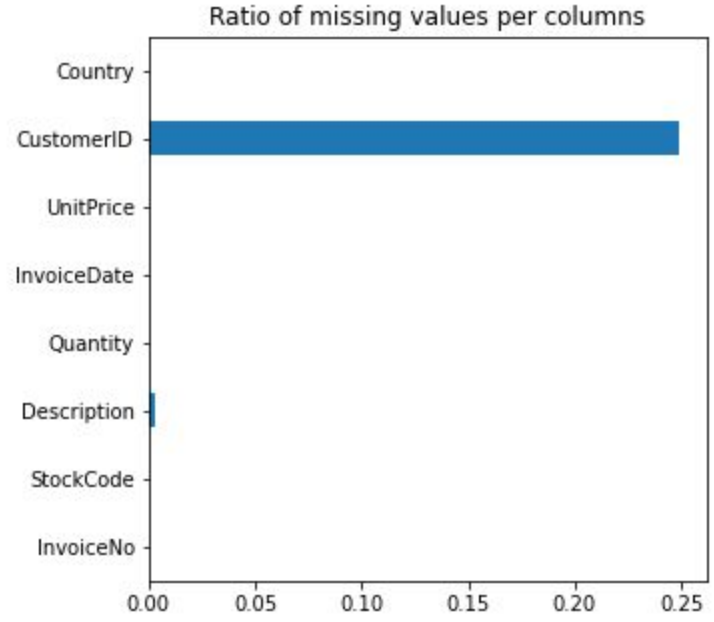


Fig 5.4.3:Bar graph for missing values

from the above bar graph the missing values are more in "CustomerId and Description".

so we have to drop these missing values by 'row wise'.
The dropna() is used to remove the missing values.

```
#Dropping the irrevalent values
order_wise = retail.dropna()
```

Fig 5.4.4:dropping the missing values

After dropping out missing values we should check our dataset whether it consists of any null values or not.To check our dataset we have various functions to view it such as heatmap,isnull().

```
#retriving the data after dropping the column of null values
order_wise.shape
order_wise.isnull().sum()
```

```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

Fig 5.4.5:using isnull()

From the above figure we can see that there are no null values

```
#visualizing the data by using heatmap after dropping the missing values
sns.heatmap(order_wise.isna())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6af4bac18>
```

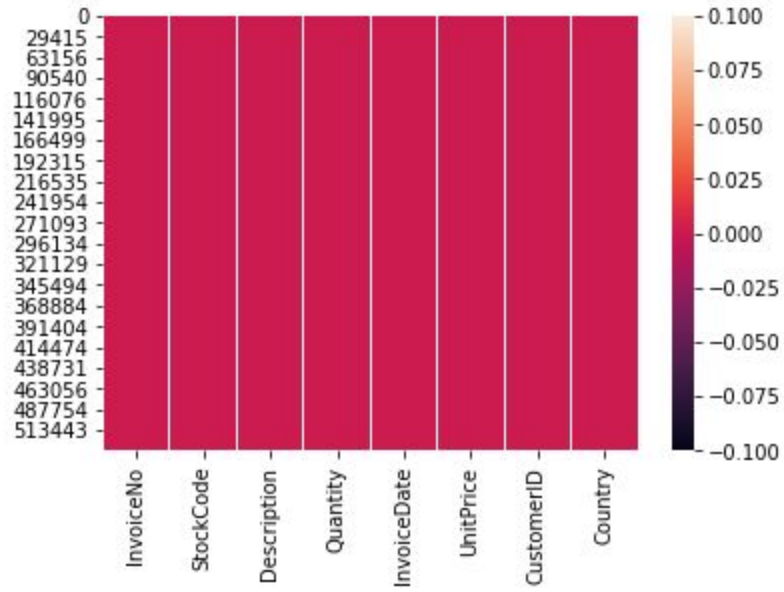


Fig 5.4.6:checking for null values using heatmap

from the above graph,we can clearly say that the are no missing values

To know how many duplicate values are present in our dataframe, we have to

use the following command:

```
#checking the number of duplicate entries
print('Duplicate Entries: {}'.format(order_wise.duplicated().sum()))
```

```
Duplicate Entries: 5225
```

Fig 5.4.7:Checking the duplicate values

So, by this we can say that 5225 duplicates entries which are present in our dataset.

Now, we have to drop these duplicate entries from our dataset by using the below command.

```
# dropping values with duplicate entries
order_wise=order_wise.drop_duplicates()
order_wise.shape
```

Fig 5.4.8: command to drop duplicate entries

After dropping these duplicate values we have to check again for the duplicates whether they are removed or not from our dataset.

```
#checking the duplicate entries after dropping the duplicate entries
print('Duplicate Entries: {}'.format(order_wise.duplicated().sum()))
```

```
Duplicate Entries: 0
```

Fig 5.4.9:No duplicate entries

Now,our data is cleaned with no missing and duplicate values where to move on for further steps to complete our project. These data exploring,handling missing and null values are very important.

```
#to check the number of unique countries
order_wise.Country.nunique()
```

```
37
```

Fig 5. 4.10:unique countries

From the above command we can say that we are having 37 unique countries with the same customer id.

```
#to check the highest number of customer's from a country
order_wise.Country.value_counts()[:37].plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c4b0536d8>
```

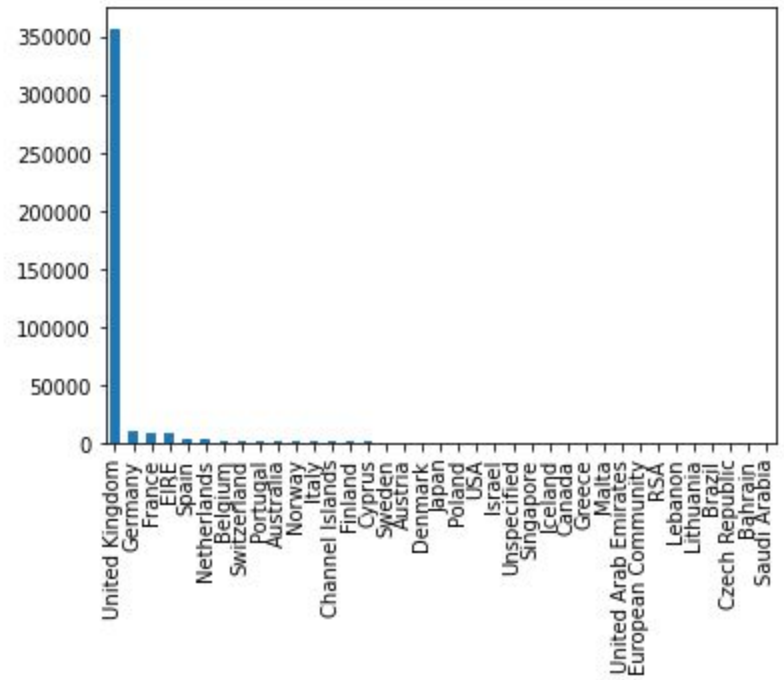


Fig 5. 4.11:customers from a different countries

From the above graph we can observe that the customers from "United kingdom" are more in number.

CHAPTER 6

RFM Analysis

6.RFM Analysis

6.1 RFM Analysis:

Smart marketers understand the importance of “know thy customer.” Instead of simply focusing on generating more clicks, marketers must follow the paradigm shift from increased CTRs (Click-Through Rates) to retention, loyalty, and building customer relationships.

Instead of analyzing the entire customer base as a whole, it's better to segment them into homogeneous groups, understand the traits of each group, and engage them with relevant campaigns rather than segmenting on just customer age or geography.

One of the most popular, easy-to-use, and effective segmentation methods to enable marketers to analyze customer behavior is RFM analysis.

6.1.1 What is RFM Analysis?

RFM stands for Recency, Frequency, and Monetary value, each corresponding to some key customer trait. These RFM metrics are important indicators of a customer's behavior because frequency and monetary value affects a customer's lifetime value, and recency affects retention, a measure of engagement.

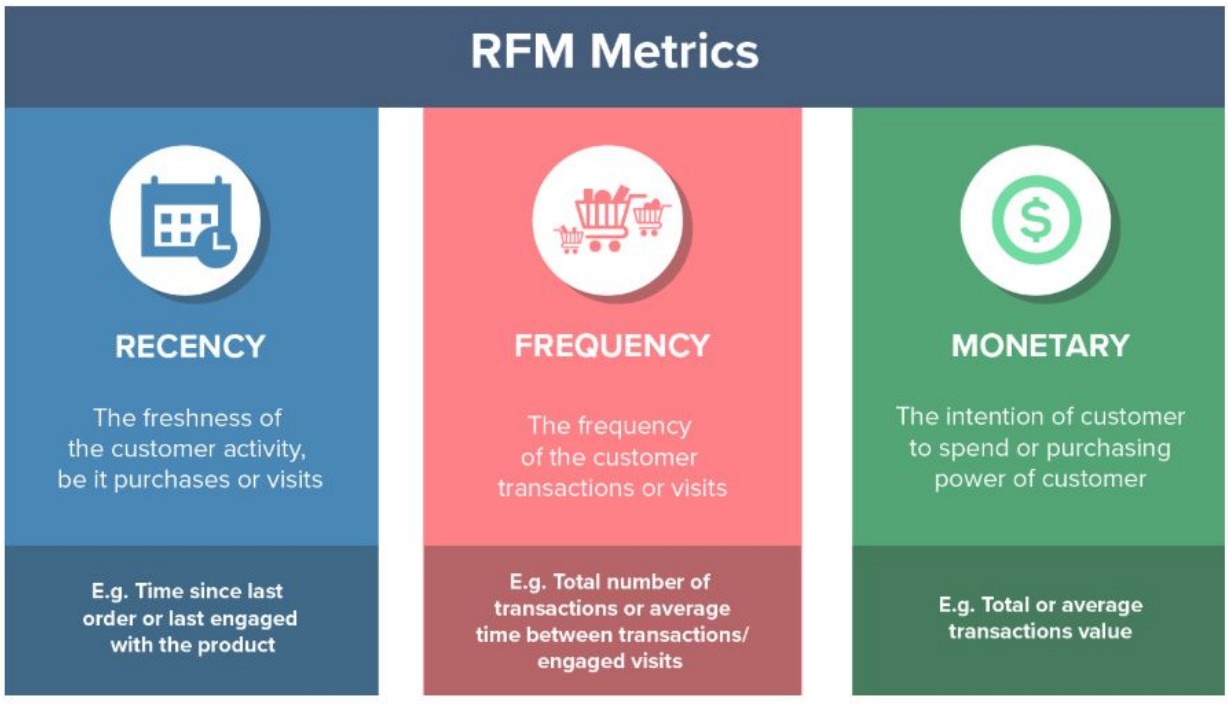


Fig 6.1.1:RFM Metrics

Businesses that lack the monetary aspect, like viewership, readership, or surfing-oriented products, could use Engagement parameters instead of Monetary factors. This results in using RFE – a variation of RFM. Furthermore, this Engagement parameter could be defined as a

composite value based on metrics such as bounce rate, visit duration, number of pages visited, time spent per page, etc.

RFM factors illustrate these facts:

1.The more recent the purchase, the more responsive the customer is to promotions

2.The more frequently the customer buys, the more engaged and satisfied they are

3.monetary value differentiates heavy spenders from low-value purchasers

6.1.2 How to Implement RFM Analysis Used in Customer Segmentation

RFM analysis helps marketers find answers to the following questions:

- Who are your best customers?
- Which of your customers could contribute to your churn rate?
- Who has the potential to become valuable customers?
- Which of your customers can be retained?
- Which of your customers are most likely to respond to the engagement campaign much do they spend?

RFM (Recency, Frequency, Monetary) analysis is a behavior-based approach grouping customers into segments. It groups the customers on the basis of their previous purchase transactions. How recently, how often, and how much did a customer buy. RFM filters customers into various groups for the purpose of better service. It helps managers to identify potential customers to do more profitable business. There is a segment of customers who is the big spender but what if they purchased only once or how recently they purchased? Do they often purchase our product? Also, It helps managers to run an effective promotional campaign for personalized service.

- Recency (R): Who have purchased recently? Number of days since last purchase (least recency)
- Frequency (F): Who has purchased frequently? It means the total number of purchases. (high frequency)
- Monetary Value(M): Who has a high purchase amount? It means the total money customer spent (high monetary value)

```
#RFM implementation

# Extracting amount by multiplying quantity and unit price and saving the data into amount variable.
amount  = pd.DataFrame(order_wise.Quantity * order_wise.UnitPrice, columns = ["Amount"])
amount.head()
```

	Amount
0	15.30
1	20.34
2	22.00
3	20.34
4	20.34

fig 6.2.1:RFM implementation

By implementing rfm we can extract amount by multiplying quantity and unit price and save the data into the amount variable.

```
] #merging amount in order_wise
  order_wise = pd.concat(objs = [order_wise,amount], axis = 1, ignore_index = False)

  #Monetary Function
  # Finding total amount spent per customer
  monetary = order_wise.groupby("CustomerID").Amount.sum()
  monetary = monetary.reset_index()
  monetary.head()
```

	CustomerID	Amount
0	12346.0	0.00
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40

fig 6.2.2:merging amount

We are merging our amount column in our new data i.e, order_wise.

```
] #Frequency function
  frequency = order_wise[['CustomerID', 'InvoiceNo']]
```

```
] # Getting the count of orders made by each customer based on customer ID.
  k = frequency.groupby("CustomerID").InvoiceNo.count()
  k = pd.DataFrame(k)
  k = k.reset_index()
  k.columns = ["CustomerID", "Frequency"]
  k.head()
```

	CustomerID	Frequency
0	12346.0	2
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17

Fig 6.2.3:orders made by each customer

In the above command we are just seeing the orders made by each customer and their frequency.

```
#Merging Amount and Frequency columns
#creating master dataset
master = monetary.merge(k, on = "CustomerID", how = "inner")
master.head()
```

	CustomerID	Amount	Frequency
0	12346.0	0.00	2
1	12347.0	4310.00	182
2	12348.0	1797.24	31
3	12349.0	1757.55	73
4	12350.0	334.40	17

Fig 6.2.4:merging amount and frequency columns

In the above command we just merged our two columns amount and frequency to know the amount that the customer is doing their transactions and to know his frequency of purchasing the product,

```
recency  = order_wise[['CustomerID','InvoiceDate']]
maximum = max(recency.InvoiceDate)
```

```
maximum
```

```
Timestamp('2011-12-09 12:50:00')
```

Fig 6.2.5:timestamp-This timestamp indicates the transaction time of the product by the customer. It shows the maximum and minimum time of transactions means starting and ending transactions of product purchased.

```
#Generating recency function

# Filtering data for customerid and invoice_date
recency  = order_wise[['CustomerID','InvoiceDate']]

# Finding max data
maximum = max(recency.InvoiceDate)

# Adding one more day to the max data, so that the max date will have 1 as the difference and not zero.
maximum = maximum + pd.DateOffset(days=1)
recency['grp'] = maximum - recency.InvoiceDate
recency.head()
```

Fig 6.2.6:generating recency function

The above figure indicates the filtered data for customerid and invoice_date , helps in finding the max date and also adds one more day to the max data ,so that the max date will

have 1 as the difference and not zero.

```
grp_region=recency.groupby('CustomerID')
grp_region.min()
```

CustomerID	InvoiceDate	grp
12346.0	2011-01-18 10:01:00	326 days 02:33:00
12347.0	2010-12-07 14:57:00	2 days 20:58:00
12348.0	2010-12-16 19:09:00	75 days 23:37:00
12349.0	2011-11-21 09:51:00	19 days 02:59:00
12350.0	2011-02-02 16:01:00	310 days 20:49:00
...	...	...
18280.0	2011-03-07 09:52:00	278 days 02:58:00
18281.0	2011-06-12 10:53:00	181 days 01:57:00
18282.0	2011-08-05 13:35:00	8 days 01:07:00
18283.0	2011-01-06 14:14:00	4 days 00:48:00
18287.0	2011-05-22 10:39:00	43 days 03:21:00

4372 rows × 2 columns

Fig6.2.7:grouping

By using group by we are grouping our customerId with invoiceDate and grp

```
#Dataframe merging by recency
df = pd.DataFrame(recency.groupby('CustomerID').grp.min())
```

```
df = df.reset_index()
df.columns = ["CustomerID", "Recency"]
df.head()
```

	CustomerID	Recency
0	12346.0	326 days 02:33:00
1	12347.0	2 days 20:58:00
2	12348.0	75 days 23:37:00
3	12349.0	19 days 02:59:00
4	12350.0	310 days 20:49:00

Fig 6.2.8:merging recency

By the above figure we can say that we are merging the recency column with customerid.

```
#Combining all recency, frequency and monetary parameters
RFM = k.merge(monetary, on = "CustomerID")
RFM = RFM.merge(df, on = "CustomerID")
RFM.head()
```

	CustomerID	Frequency	Amount	Recency
0	12346.0	2	0.00	326 days 02:33:00
1	12347.0	182	4310.00	2 days 20:58:00
2	12348.0	31	1797.24	75 days 23:37:00
3	12349.0	73	1757.55	19 days 02:59:00
4	12350.0	17	334.40	310 days 20:49:00

Fig6.3.1:combining the parameters

From the above figure we can observe that 3 parameters that are frequency,amount,recency are combined to get the appropriate information about the customers using clusterid.

CHAPTER -7

Outliers

7.Outliers

7.1.1What are outliers?

An **outlier** is an observation that lies an abnormal distance from other values in a random sample from a population. ... Examination of the data for unusual observations that are far removed from the mass of data. These points are often referred to as **outliers**.

7.1.2 Why Do We Care About Outliers?

Detecting outliers or anomalies is one of the core problems in data mining. The emerging expansion and continued growth of data and the spread of IoT devices, make us rethink the way we approach anomalies and the use cases that can be built by looking at those anomalies.

We now have smart watches and wristbands that can detect our heartbeats every few minutes. Detecting anomalies in the heartbeat data can help in predicting heart diseases. Anomalies in traffic patterns can help in predicting accidents. It can also be used to identify bottlenecks in network infrastructure and traffic between servers. Hence, the use cases and solutions built on top of detecting anomalies are limitless.Another reason why we need to detect anomalies is that when preparing datasets for machine learning models, it is really important to detect all the outliers and either get rid of them or analyze them to know why you had them there in the first place.Now, let’s explore 5 common ways to detect anomalies starting with the most simple way.

7.2 METHODS TO DETECT Outliers:

Method-1:Standard Deviation:

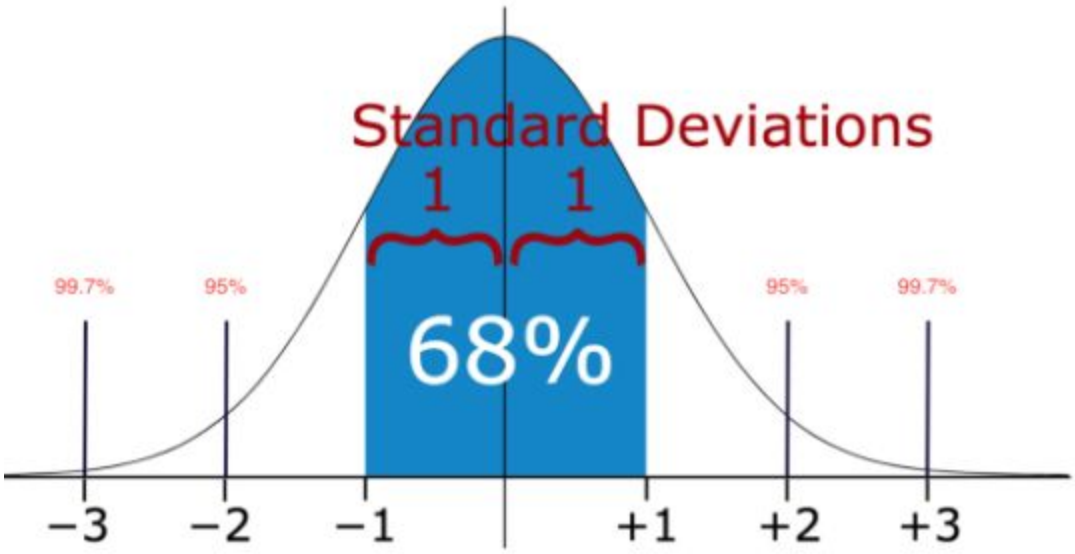


Fig 7.2.1:standard deviation

Method-2:Box plots:

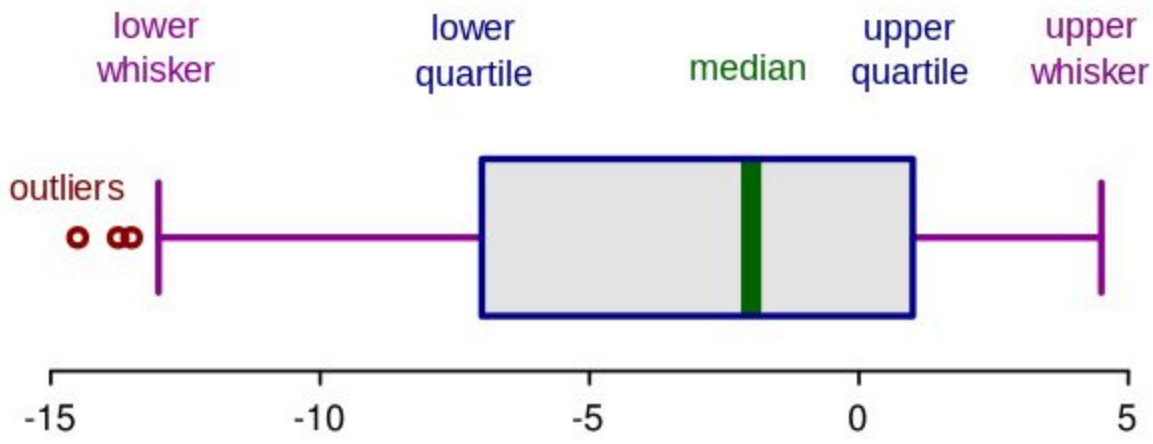


Fig 7.2.2:box plots

Boxplot Anatomy:

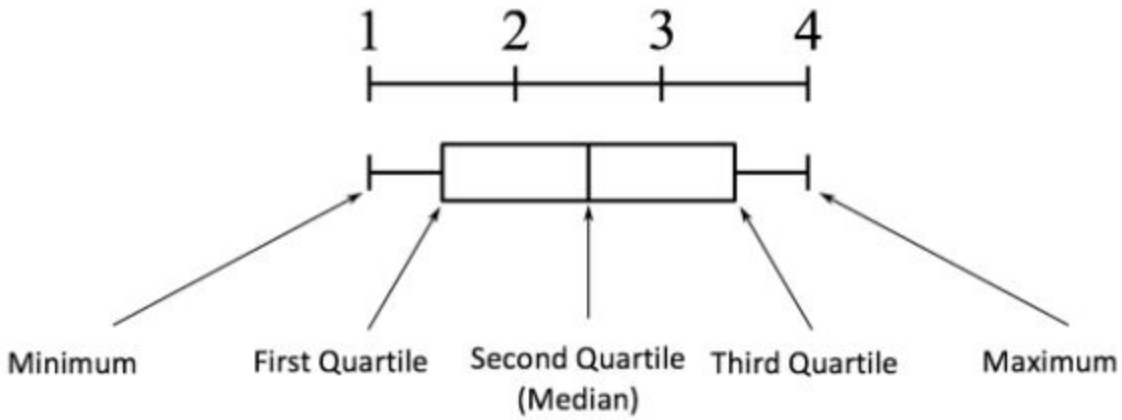


Fig 7.2.3:boxplot anatomy

Method-3:DBScan Clustering:

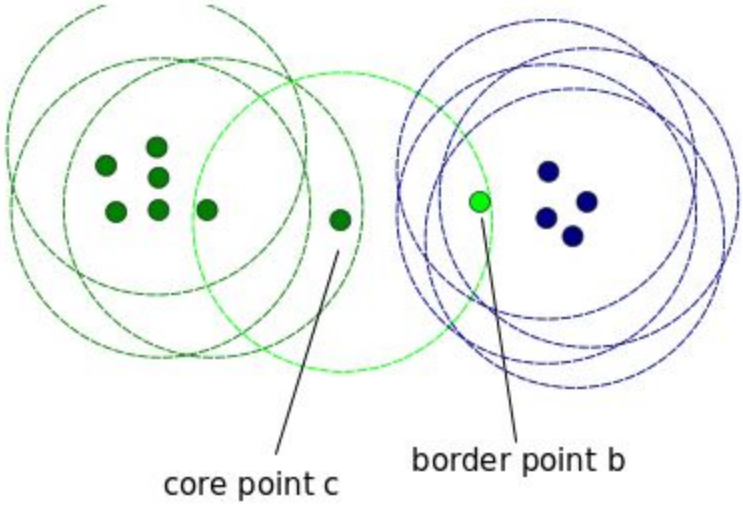


Fig 7.2.4:DBScan Clustering

Method-4: Isolation Forest:

```python
from sklearn.ensemble import IsolationForest
import numpy as np
np.random.seed(1)
random_data = np.random.randn(50000,2)  * 20 + 20

clf = IsolationForest( behaviour = 'new', max_samples=100, random_state = 1, contamination= 'auto
preds = clf.fit_predict(random_data)
preds
```

isolationForest.py hosted with ❤ by GitHub view raw

Fig 7.2.5: Isolation Forest

Method 5— Robust Random Cut Forest:

Table 1. Comparison of Baseline Isolation Forest to proposed Robust Random Cut Forest

Method	Sample Size	Positive Precision	Positive Recall	Negative Precision	Negative Recall	Accuracy	AUC
IF	256	0.42 (0.05)	0.37 (0.02)	0.96 (0.00)	0.97 (0.01)	0.93 (0.01)	0.83 (0.01)
RRCF	256	0.87 (0.02)	0.44 (0.04)	0.97 (0.00)	1.00 (0.00)	0.96 (0.00)	0.86 (0.00)
IF	512	0.48 (0.05)	0.37 (0.01)	0.97 (0.01)	0.96 (0.00)	0.94 (0.00)	0.86 (0.00)
RRCF	512	0.84 (0.04)	0.50 (0.03)	0.99 (0.00)	0.97 (0.00)	0.96 (0.00)	0.89 (0.00)
IF	1024	0.51 (0.03)	0.37 (0.01)	0.96 (0.00)	0.98 (0.00)	0.94 (0.00)	0.87 (0.00)
RRCF	1024	0.77 (0.03)	0.57 (0.02)	0.97 (0.00)	0.99 (0.00)	0.96 (0.00)	0.90 (0.00)

Method	Segment Precision	Segment Recall	Time to Detect Onset	Time to Detect End	Prec@5	Prec@10	Prec@15	Prec@20
IF	0.40 (0.09)	0.80 (0.09)	22.68 (3.05)	23.30 (1.54)	0.52 (0.10)	0.50 (0.00)	0.34 (0.02)	0.28 (0.03)
RRCF	0.65 (0.14)	0.80 (0.00)	13.53 (2.05)	10.85 (3.89)	0.58 (0.06)	0.49 (0.03)	0.39 (0.02)	0.30 (0.00)

Fig 7.2.6:Robust Random Cut Forest

In my project i have used “BOX PLOTS” outliers

Boxplot Anatomy:

The concept of the I**nterquartile Range** (**IQR**) is used to build the boxplot graphs. IQR is a concept in statistics that is used to measure the statistical dispersion and data variability by dividing the dataset into quartiles.

7.3 Outlier treatment:

```
# outlier treatment for Amount
plt.boxplot(RFM.Amount)
Q1 = RFM.Amount.quantile(0.25)
Q3 = RFM.Amount.quantile(0.75)
IQR = Q3 - Q1
RFM = RFM[(RFM.Amount >= (Q1 - 1.5*IQR)) & (RFM.Amount <= (Q3 + 1.5*IQR))]
```

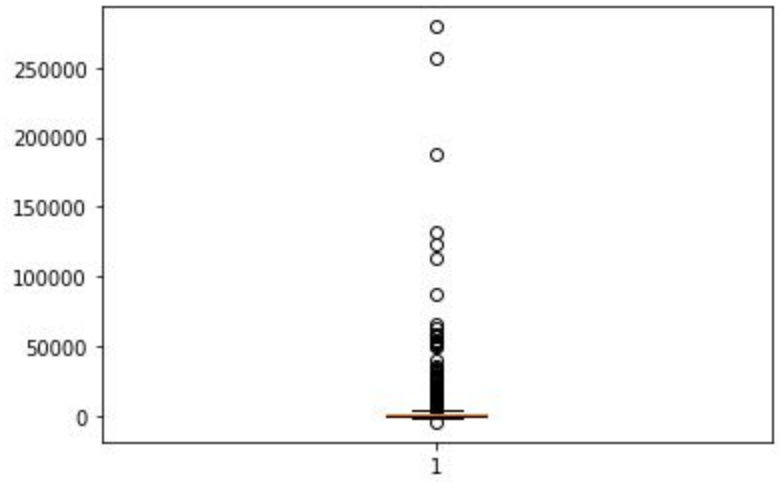


Fig 7.3.1:outlier for amount

From the above graph we can see that the amount values are more than 2,50,000.

```
# outlier treatment for Frequency
plt.boxplot(RFM.Frequency)
Q1 = RFM.Frequency.quantile(0.25)
Q3 = RFM.Frequency.quantile(0.75)
IQR = Q3 - Q1
RFM = RFM[(RFM.Frequency >= Q1 - 1.5*IQR) & (RFM.Frequency <= Q3 + 1.5*IQR)]
```

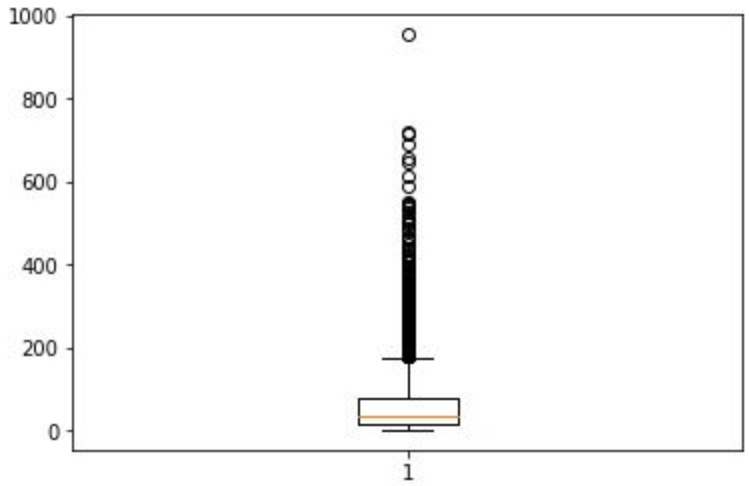


Fig 7.3.2:outlier for frequency

From the above graph we can say that the frequency values are between 800-1000.

```
# outlier treatment for Recency
plt.boxplot(RFM.Recency)
Q1 = RFM.Recency.quantile(0.25)
Q3 = RFM.Recency.quantile(0.75)
IQR = Q3 - Q1
RFM = RFM[(RFM.Recency >= Q1 - 1.5*IQR) & (RFM.Recency <= Q3 + 1.5*IQR)]
```

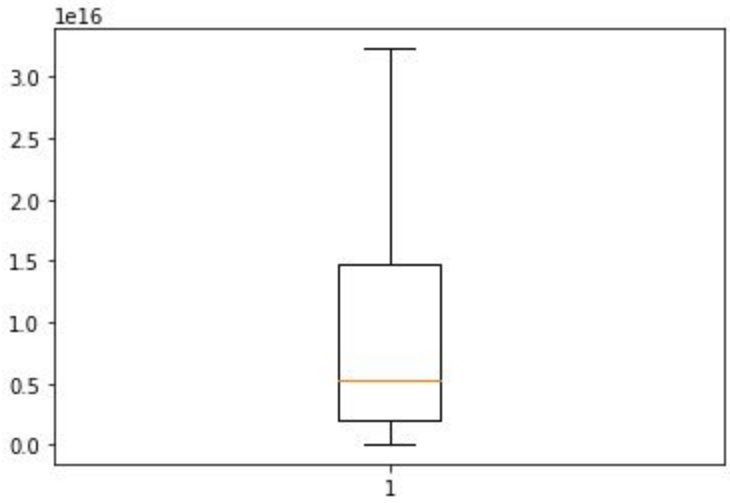


Fig 7.3.3:outlier for recency

From the above graph we can say that the recency value is above the 3.0.

CHAPTER-8

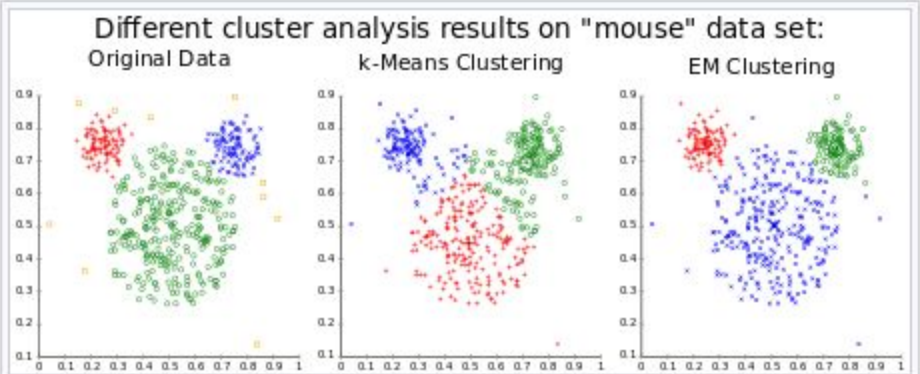
K-Means clustering

8.K-Means clustering

8.1 Introduction:

***k*-means clustering** is a method of vector quantization, originally from single processing that aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean(cluster centers or cluster centroid), serving as a prototype of the cluster. This results in a partitioning of the data space into voronoi cells. It is popular for cluster analysis in data mining *k*-means clustering minimizes within-cluster variances, but not regular Euclidean distances, which would be the more difficult Weber problem: the mean optimizes squared errors, whereas only the geometric median minimizes Euclidean distances. For instance, better Euclidean solutions can be found using k-median and k-medoids The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum These are usually similar to the expectation - maximization algorithms for mixtures of Gaussian Distributions via an iterative refinement approach employed by both k-means and Gaussian mixtures modelling They both use cluster centers to model the data; however, *k*-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the k-nearest neighbour algorithm, a popular Machine Learning technique for classification that is often confused with *k*-means due to the name. Applying the 1-nearest neighbor classifier to the cluster centers obtained by *k*-means classifies new data into the existing clusters. This is known as the nearest centroid classifier or Rocchio algorithm.



k-means clustering vs. EM clustering on an artificial dataset ("mouse"). The tendency of *k*-means to produce equal-sized clusters leads to bad results here, while EM benefits from the Gaussian distributions with different radius present in the data set.

Fig 8.1:cluster analysis

8.2 Applications:

k-means clustering is rather easy to apply to even large data sets, particularly when using heuristics such as Lloyd's algorithm. It has been successfully used in market

segmentation,computer vision and astronomy among many other domains. It often is used as a preprocessing step for other algorithms, for example to find a starting configuration.

```
# standardise all parameters
RFM_norm1 = RFM.drop(["CustomerID"], axis=1)
RFM_norm1.Recency = RFM_norm1.Recency.dt.days

from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
RFM_norm1 = standard_scaler.fit_transform(RFM_norm1)
```

```
RFM_norm1 = pd.DataFrame(RFM_norm1)
RFM_norm1.columns = ['Frequency','Amount','Recency']
RFM_norm1.head()
```

	Frequency	Amount	Recency
0	-1.074138	-1.041762	2.137274
1	-0.370228	1.399756	-0.281781
2	0.649227	1.345838	-0.821490
3	-0.710046	-0.587486	1.983071
4	1.183227	1.057650	-0.657650

Fig 8.2.1:Scaling the rfm data

8.3 Determining The Optimal Number Of Clusters:

Determining the optimal number of clusters in a data set is a fundamental issue in partitioning clustering, such as k-means clustering which requires the user to specify the number of clusters k to be generated.

Unfortunately, there is no definitive answer to this question. The optimal number of clusters is somehow subjective and depends on the method used for measuring similarities and the parameters used for partitioning

A simple and popular solution consists of inspecting the dendrogram produced using hierarchical clustering to see if it suggests a particular number of clusters. Unfortunately, this approach is also subjective.

In this chapter, we'll describe different methods for determining the optimal number of clusters for k-means, k-medoids (PAM) and hierarchical clustering.

These methods include direct methods and statistical testing methods:

1. Direct methods: consists of optimizing a criterion, such as the within cluster sums of squares or the average silhouette. The corresponding methods are named elbow and silhouette methods, respectively.

2. Statistical testing methods: consists of comparing evidence against null hypothesis. An example is the gap statistic.

In addition to elbow, silhouette and gap statistic methods, there are more than thirty other indices and methods that have been published for identifying the optimal number of clusters. We'll provide R codes for computing all these 30 indices in order to decide the best number of clusters using the "majority rule".

For each of these methods:

- We'll describe the basic idea and the algorithm
- We'll provide easy-o-use R codes with many examples for determining the optimal number of clusters and visualizing the output.

8.4 Elbow method

Recall that, the basic idea behind partitioning methods, such as k-means clustering, is to define clusters such that the total intra-cluster variation [or total within-cluster sum of square (WSS)] is minimized. The total WSS measures the compactness of the clustering and we want it to be as small as possible.

The Elbow method looks at the total WSS as a function of the number of clusters: One should choose a number of clusters so that adding another cluster doesn't improve much better the total WSS.

The optimal number of clusters can be defined as follow:

1. Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters.
2. For each k, calculate the total within-cluster sum of square (wss).
3. Plot the curve of wss according to the number of clusters k.
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

8.5 Silhouette Analysis

$$\text{silhouette score} = \frac{p - q}{max(p, q)}$$

p is the mean distance to the points in the nearest cluster that the data point is not a part of
q is the mean intra-cluster distance to all the points in its own cluster

- The value of the silhouette score range lies between -1 to 1.
- A score closer to 1 indicates that the data point is very similar to other data points in the cluster,
- A score closer to -1 indicates that the data point is not similar to the data points in its cluster.

```
# sum of squared distances
ssd = []
for num_clusters in list(range(1,21)):
    model_clus = KMeans(n_clusters = num_clusters, max_iter=50)
    model_clus.fit(RFM_norm1)
    ssd.append(model_clus.inertia_)

plt.plot(ssd)
```

```
[<matplotlib.lines.Line2D at 0x7fb6aef11c18>]
```

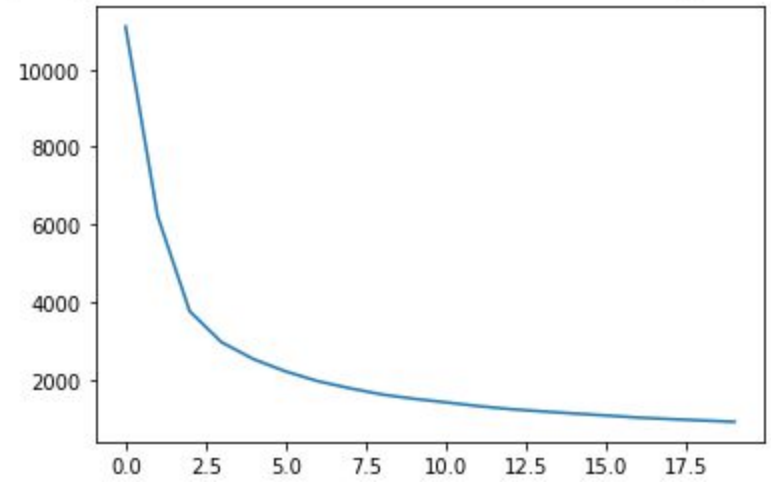


Fig 8.4.1:elbow graph

From the above graph we can observe that there are 3 deviations,so we are having 3 numbers of clusters.

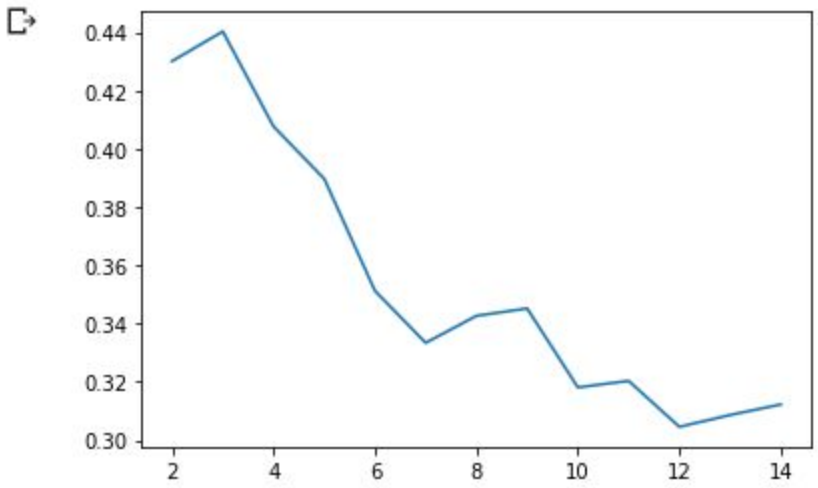
```
# applying k-means clustering
model_clus3 = KMeans(n_clusters = 3, max_iter=50)# Kmeans with K=3
model_clus3.fit(RFM_norm1)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=50,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

```
##Silhouette Analysis
from sklearn.metrics import silhouette_score
sse_ = []
for k in range(2, 15):
    kmeans = KMeans(n_clusters=k).fit(RFM_norm1)
    sse_.append([k, silhouette_score(RFM_norm1, kmeans.labels_)])
```

Fig 8.5.1:Silhouette Analysis

```
[43] #plotting a linkage graph
     plt.plot(pd.DataFrame(sse_)[0], pd.DataFrame(sse_)[1]);
```



From the above graph we can say that the average linkage is between 3-7.

Fig 8.5.2:linkage graph

```
# analysis of clusters formed
RFM.index = pd.RangeIndex(len(RFM.index))
RFM_km = pd.concat([RFM, pd.Series(model_clus.labels_)], axis=1)
RFM_km.columns = ['CustomerID', 'Frequency', 'Amount', 'Recency', 'ClusterID']

RFM_km.Recency = RFM_km.Recency.dt.days
km_clusters_amount =  pd.DataFrame(RFM_km.groupby(["ClusterID"]).Amount.mean())
km_clusters_frequency =   pd.DataFrame(RFM_km.groupby(["ClusterID"]).Frequency.mean())
km_clusters_recency =   pd.DataFrame(RFM_km.groupby(["ClusterID"]).Recency.mean())
```

```
km_clusters_amount
```

ClusterID	Amount
0	1369.520465
1	243.310429
2	2871.027081
3	2042.652240
4	1057.353681

Fig 8.5.3:clusters formed

From the above we can analyse the cluster's forms and these are grouped by clusterid.

```
#plottiong the bar graph between cluserid and amount-mean
sns.barplot(x=df.ClusterID, y=df.Amount_mean)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6ad236e48>
```

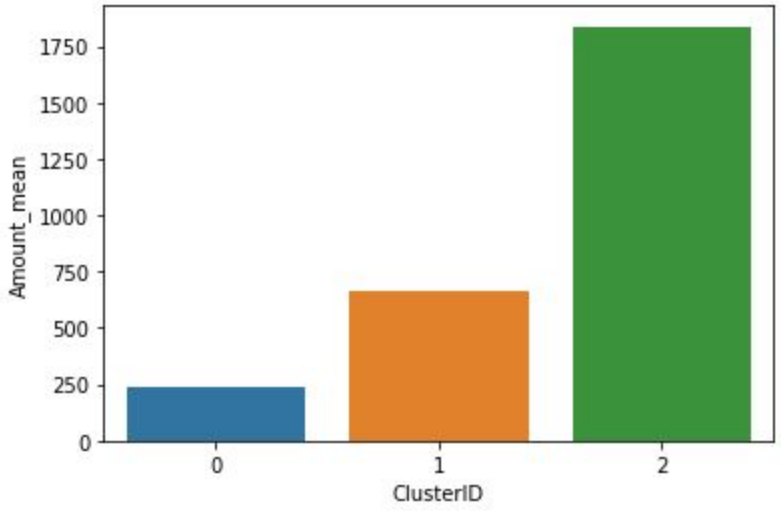


Fig 8.6.1:amount_mean graph

customers with clusterID 2 are the customers with a high amount of transactions as compared to other customers.

```
#plottiong the bar graph between cluserid and frequency_mean
sns.barplot(x=df.ClusterID, y=df.Frequency_mean)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6acb2a9e8>
```

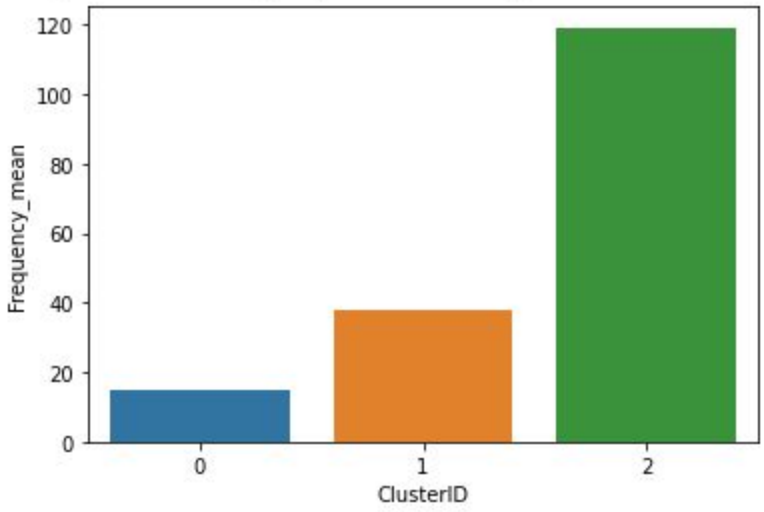


Fig8.6.2:bar graph for frequency mean

Customers with ClusterID 2 are frequent buyers

```
#plottiong the bar graph between cluserid and recency-mean
sns.barplot(x=df.ClusterID, y=df.Recency_mean)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6ad1ac630>
```

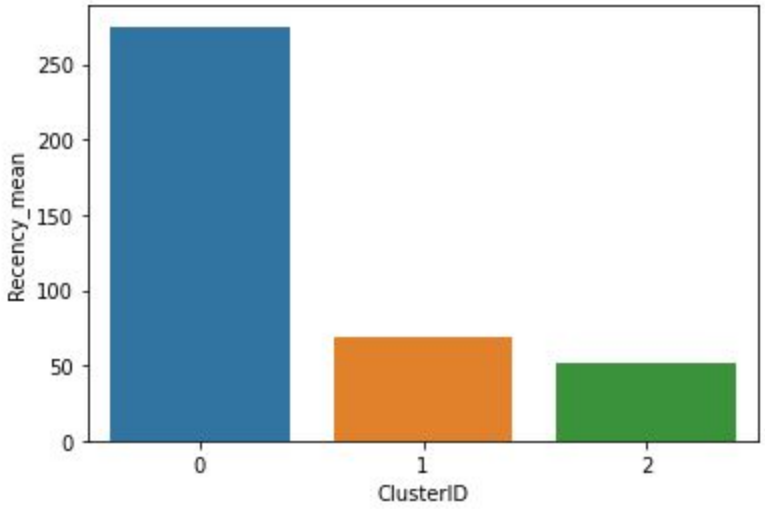


Fig 8.6.3:bar graph for recency mean

customers with ClusterID 0 are recent buyers and hence highly important from a business point of view.

CHAPTER-9

Hierarchical clustering

9.Hierarchical clustering

9.1 Introduction:

Hierarchical clustering, also known as hierarchical cluster analysis, is an algorithm that groups similar objects into groups called clusters. The endpoint is a set of clusters, where each cluster is distinct from each other cluster, and the objects within each cluster are broadly similar to each other.

In data mining and statistics, hierarchical clustering (also called hierarchical cluster analysis or HCA) is a method of cluster analysis which seeks to build a hierarchy of clusters. Strategies for hierarchical clustering generally fall into two types:

1.Agglomerative Clustering

2.Divisive Clustering

In general, the merges and splits are determined in a greedy manner. The results of hierarchical clustering are usually presented in a dendrogram .

9.2 Applications of Hierarchical Clustering:

1) US Senator Clustering through Twitter

Can we find the party lines through Twitter?Following the controversial “Twitter mood predicts the stock market” paper, researchers have been looking at Twitter as a source of highly valuable data. In this example, we use Twitter to cluster US senators into their respective parties.Our data is simple: we only look at which senators follow which senators. That defines a graph structure with senators as the nodes and follows as the edges.On this graph, we use the Walktrap algorithm by ponestal, which does a random walk through graph, and estimates the senator similarity by the number of times you end up at a certain senator starting from a senator.After getting these similarities, we can use agglomerative clustering to find the dendrogram.

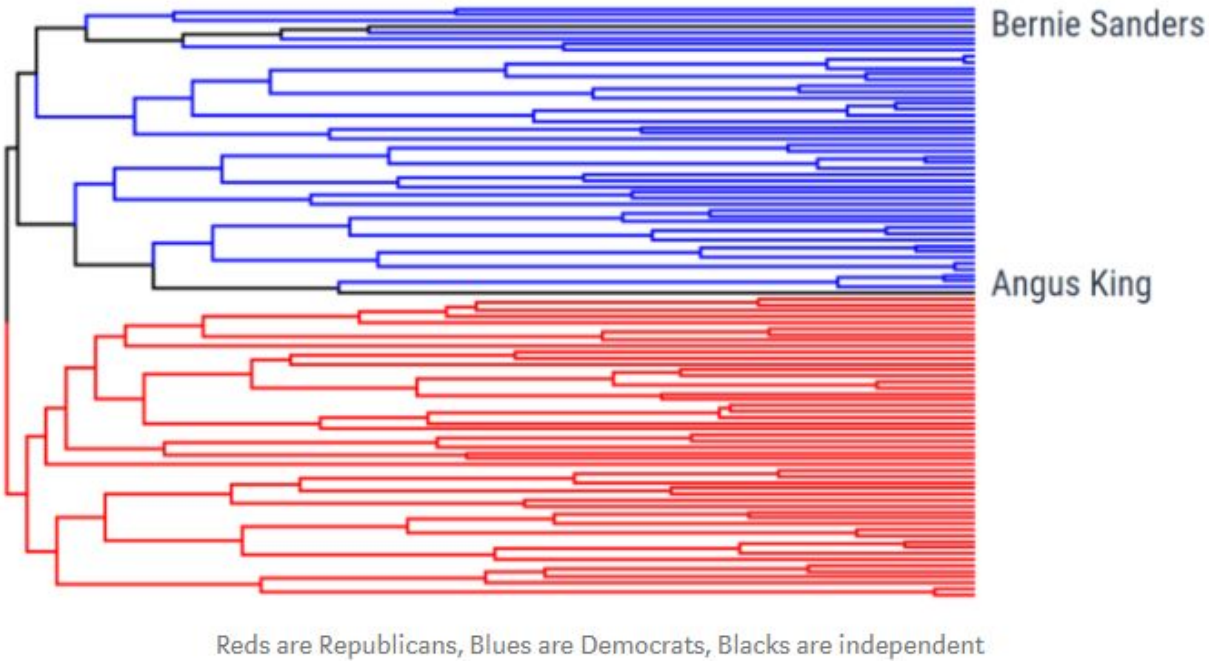


Fig 9.2.1:dendrograms

In order to measure how well our clustering worked, we can color the results with the party colors. As you can see, Democrats and Republicans are very clearly split from the top, showing the success of this method.

You also might've noticed the two black lines, denoting the independent senators. These are a little trickier to evaluate, but both Sen. Bernie Sanders and Sen. Angus King caucus with the Democratic Party, meaning that it is natural that they are in the Democratic Party branch of the tree.

2) Charting Evolution through Phylogenetic Trees

How can we relate different species together?In the decades before DNA sequencing was reliable, the scientists struggled to answer a seemingly simple question: Are giant pandas closer to bears or racoons?

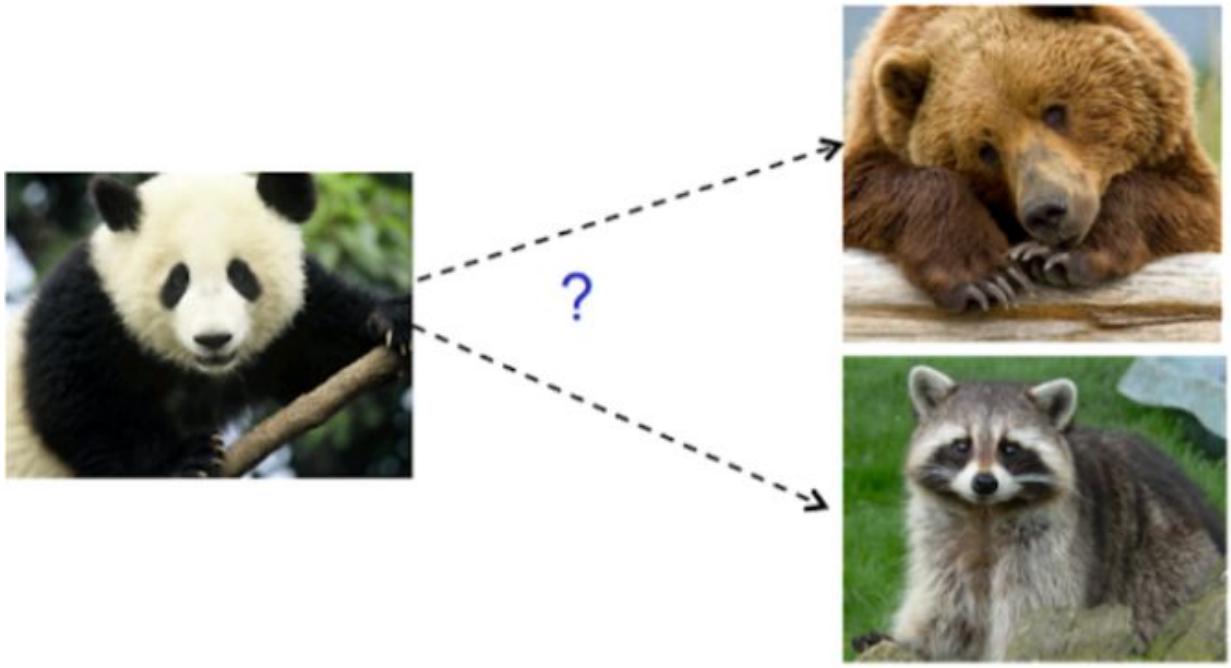


Fig 9.2.2:DNA sequencing

Nowadays, we can use DNA sequencing and hierarchical clustering to find the phylogenetic tree of animal evolution:

1. Generate the DNA sequences Calculate the edit distances between all sequences.
2. Calculate the DNA similarities based on the edit distances.

3. Construct the phylogenetic tree.

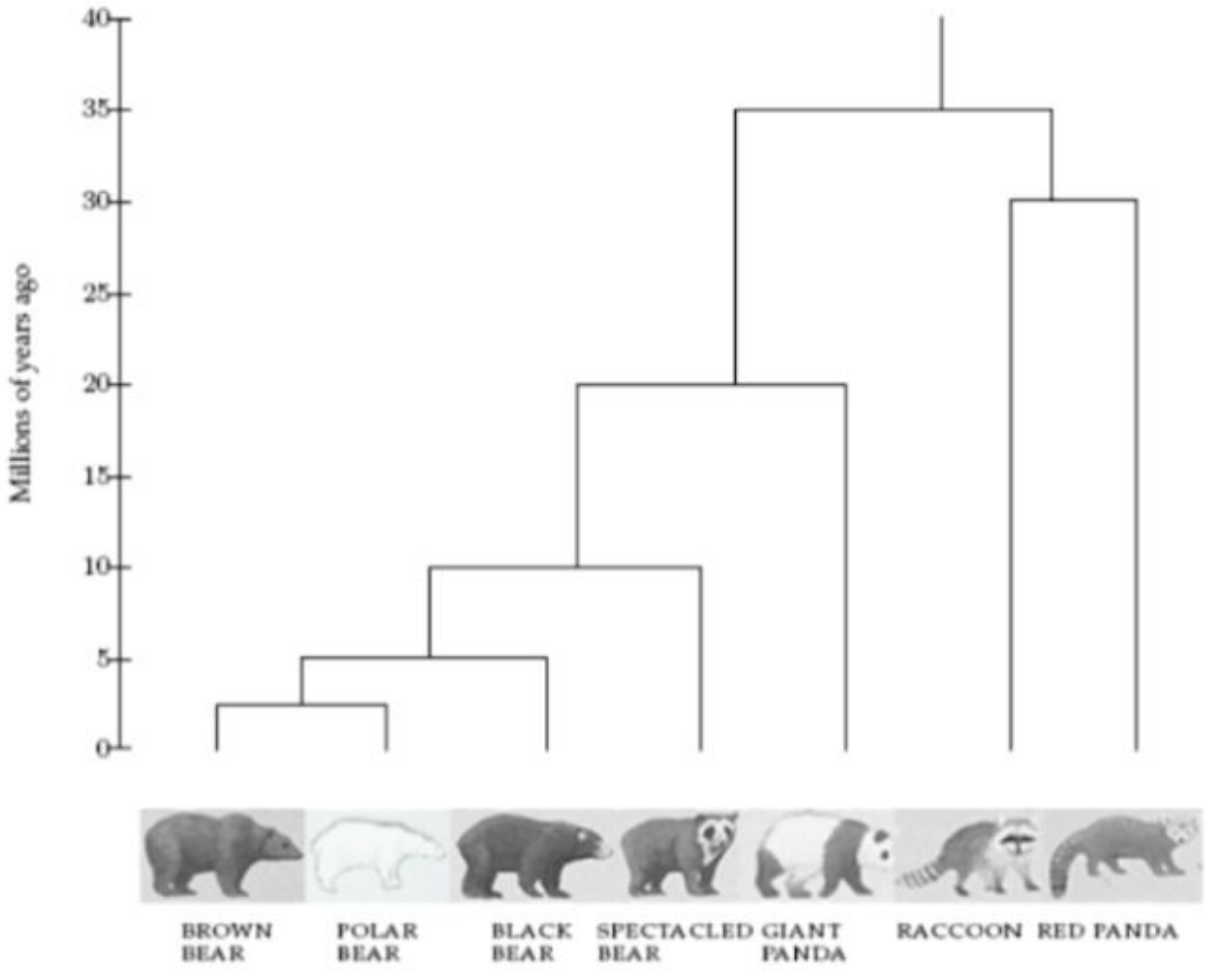


Fig 9.2.3: phylogenetic tree.

As a result of this experiment, the researchers were able to place the giant pandas closer to bears.

3) Tracking Viruses through Phylogenetic Trees

Can we find where a viral outbreak originated?

Tracking viral outbreaks and their sources is a major health challenge. Tracing these outbreaks to their source can give scientists additional data as to why and how the outbreak began, potentially saving lives.

Viruses such as HIV have high mutation rates, which means the similarity of the DNA sequence of the same virus depends on the time since it was transmitted. This can be used to trace paths of transmission.

This method was used as evidence in our court case, wherein the victim's strand of HIV was found to be more similar to the accused patient's strand, compared to a control group.

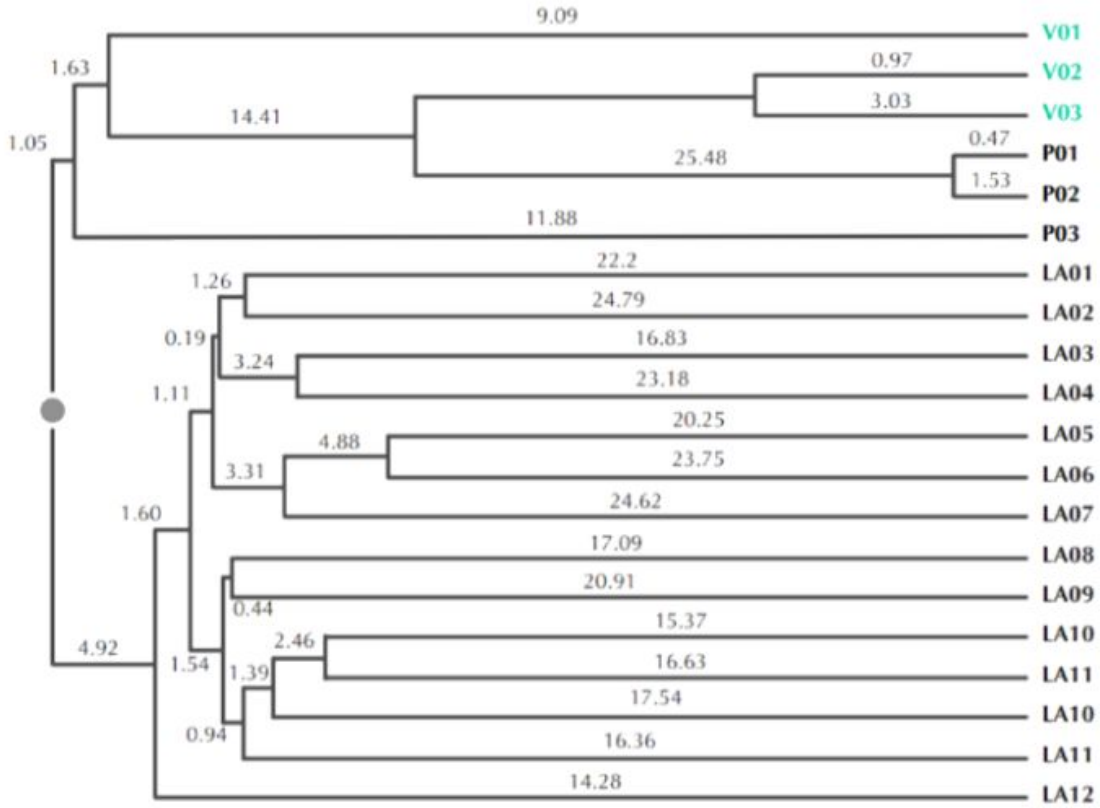


Fig 9. 2.4:V1–3 are victim's strands, P1–3 are accused patient's, and LA1–12 are the control group

A similar study was also done for finding the animal that gave the humans the SARS virus

9.3 Dendrograms in Python:

A **dendrogram** is a diagram representing a tree. The figure factory called create_dendrogram performs hierarchical clustering on data and represents the resulting tree. Values on the tree depth axis correspond to distances between clusters.

Why is Dendrogram used for?

A **dendrogram** is a diagram that shows the hierarchical relationship between objects. It is most commonly created as an output from hierarchical clustering. The main **use** of a dendrogram is to work out the best way to allocate objects to clusters.

WORKS: A dendrogram is a diagram that shows the attribute distances between each pair of sequentially merged classes. ... After each merging, the distances between all pairs of classes are updated. The distances at which the signatures of classes are merged are used to construct a dendrogram.

```
# heirarchical clustering
mergings = linkage(RFM_norm1, method = "single", metric='euclidean')
dendrogram(mergings)
plt.show()
```

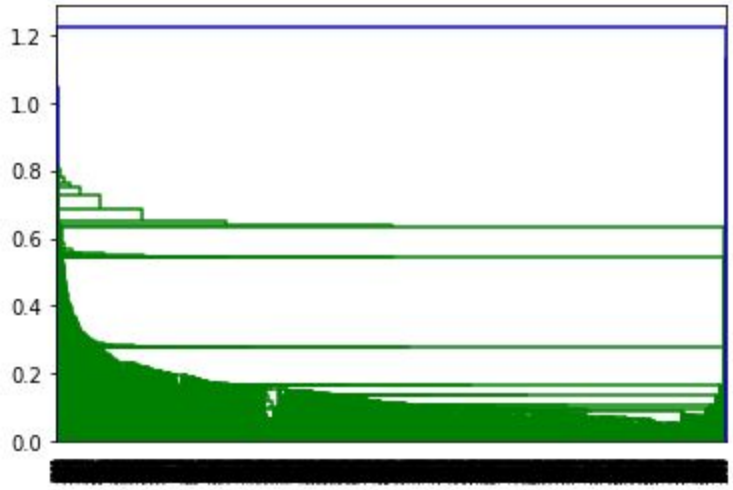


Fig 9.3.1:plotting the dendrogram using hierarchical clustering

From the above graph we can say that ,the above graph represents our dataset in this way in a hierarchical cluster.

```
] #plotting the dendrogram
  mergings = linkage(RFM_norm1, method = "complete", metric='euclidean')
  dendrogram(mergings)
  plt.show()
```

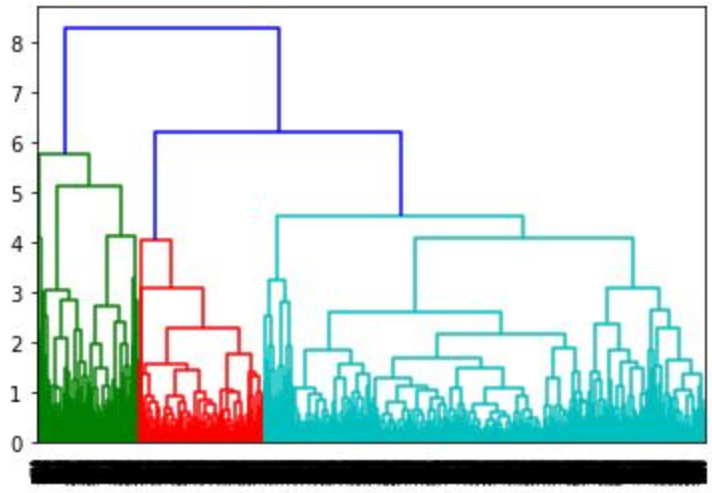


Fig 9.3.2:dendrogram

Dendrograms are used to divide into multiple clusters depending upon the problem.

```
#summarise
RFM_hc.Recency = RFM_hc.Recency.dt.days
km_clusters_amount =  pd.DataFrame(RFM_hc.groupby(["ClusterID"]).Amount.mean())
km_clusters_frequency =   pd.DataFrame(RFM_hc.groupby(["ClusterID"]).Frequency.mean())
km_clusters_recency =   pd.DataFrame(RFM_hc.groupby(["ClusterID"]).Recency.mean())
```

```
df = pd.concat([pd.Series([0,1,2]), km_clusters_amount, km_clusters_frequency, km_clusters_recency], axis=1)
df.columns = ["ClusterID", "Amount_mean", "Frequency_mean", "Recency_mean"]
df.head()
```

	ClusterID	Amount_mean	Frequency_mean	Recency_mean
0	0	234.770072	14.840810	275.073806
1	1	666.123689	38.002461	68.297785
2	2	1837.121328	119.237347	51.136126

Fig 9.3.3:summarise

From the above figure we can summarize our data and this contains 3 clusters with clusterid-0,1,2

Finally, we can say that clusterid 0 has high recency mean,clusterid 1 has high amount-mean,clusterid 2 also has high amount mean.

```
#plottiong the bar graph between cluserid and amount-mean
sns.barplot(x=df.ClusterID, y=df.Amount_mean)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6aedc1a20>
```

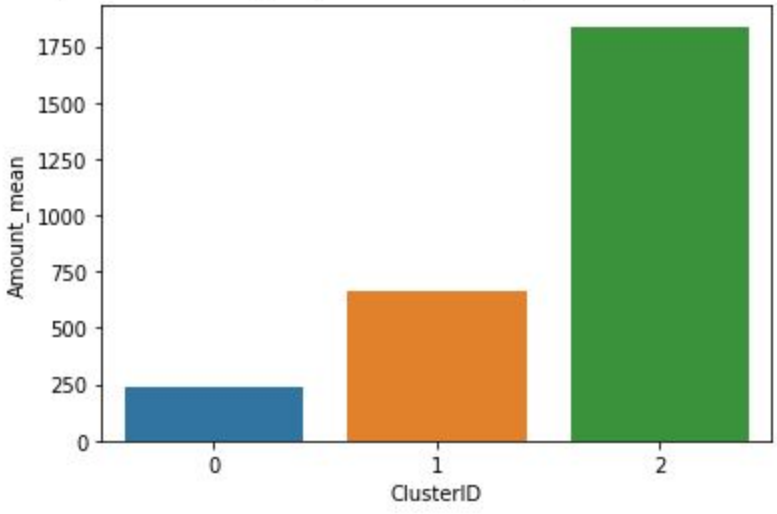


Fig 9.4.1:amount-mean bar graph

Customers with ClusterId 2 are the customers with a high amount of transactions as compared to other customers.

```
#plottiong the bar graph between cluserid and frequency-mean
sns.barplot(x=df.ClusterID, y=df.Frequency_mean)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6acca82e8>
```

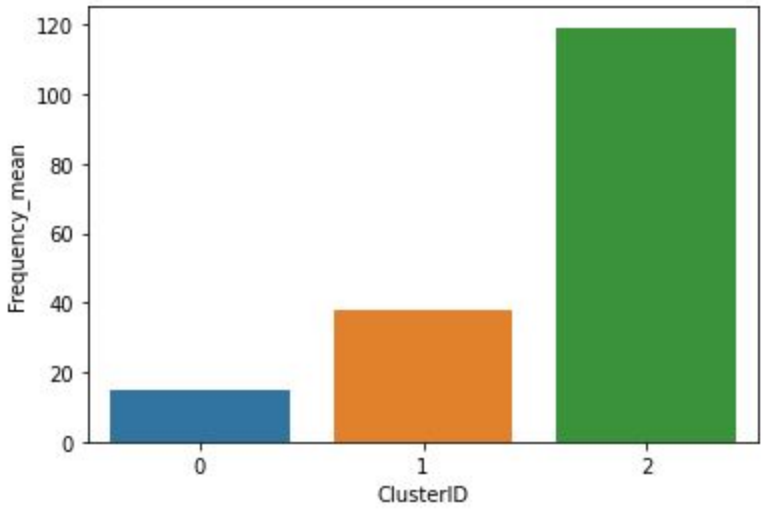


Fig 9.4.2:frequency_mean bar graph

Customers with ClusterId 2 are frequent buyers.

```
#plotting the bargraph between recency_mean and clusterid
sns.barplot(x=df.ClusterID, y=df.Recency_mean)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6ae9c86d8>
```

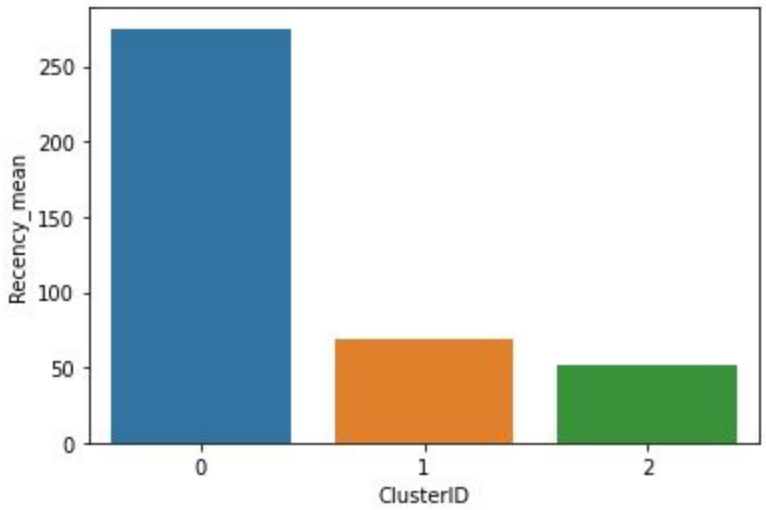


Fig 9.4.3:recency_mean bar graph-

Customers with ClusterID 0 are not recent buyers and hence least of importance from a business point of view.

CONCLUSION:

By this i can conclude that,

1.The customers with clusterid 0 has high amount of transactions

2.The customers with clusterid 1 are frequent buyers

3.And,The Customers with clusterid 2 is high recency buyers who are not the recent buyers

hence these are very important segmentations which are helpful from a business point of view for the companies.

REFERENCES:

1.https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c025
2. https://en.wikipedia.org/wiki/Machine_learning

3. https://www.edureka.co/blog/data-science-applications/

4.https://clevertap.com/blog/rfm-analysis/