DATA SCIENCE

(Customer Segmentation based on Purchase Behaviour)

Summer Internship Report Submitted in partial fulfillment of the requirement for undergraduate degree of

Bachelor of Technology

In

COMPUTER SCIENCE AND ENGINEERING

By

ADAVENI BHAVANA

221710313001

Under the Guidance of

Assistant Professor

Department of Computer Science and Engineering

GITAM School of Technology

GITAM(Deemed to be University)

Hyderabad-502329

July 2020

DECLARATION

I submit this industrial training work entitled “PREDICTION OF LOAN APPROVAL” to GITAM(Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of “Bachelor of Technology ” in “ Computer Science Engineering”. I declare that it was carried out independently by me under the guidance of ________, _________, GITAM(Deemed to be University),Hyderabad,India.

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

Place: Hyderabad Name: Adaveni Bhavana

Date: RollNo:221710313001

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 Prof. N. Seetha Ramaiah, Principal, GITAM Hyderabad.

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

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

Adaveni Bhavana
221710313001

ABSTRACT

Knowing its customers is essential for a company’s success. Especially in the age of e-commerce, where customers became just rows of lists, it is important to target them more individually. The traces buyers leave in an online-store, allow detailed insights on their habits how they interact with an organization. Their purchase history contains all transactional data to build meaningful customer segments. These segments allow more targeted communication and actions towards customers.

The motivation behind this thesis is to investigate the value of clustering in the machine learning /data mining context for customer segmentation. Classical database marketing methods are combined with data mining tools. Data mining techniques can be used to create the segments automatically. The outcome shows that machine learning can be applied successfully for the needs of small and medium organizations and can help in handling a growing customer base.

Table of Contents

List of figures

CHAPTER-1

1. INFORMATION ABOUT DATA SCIENCE

1.1 What is Data Science?

Data science is an interdisciplinary field that uses scientific methods, processes, Algorithms and systems to extract knowledge and insights from many structural and Unstructured data.

Data science provides meaningful information based on large amounts of complex data or big data. Data science, or data-driven science, combines different fields of work in statistics and computation to interpret data for decision-making purposes.

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.

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. Virtual assistance for patients and customer support:

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.

3. Targeted Advertising:

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.

CHAPTER-2
INFORMATION ABOUT MACHINE LEARNING

2.1 Introduction of Machine Learning

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so.

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 analyse 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 following diagram depicts about the machine learning process

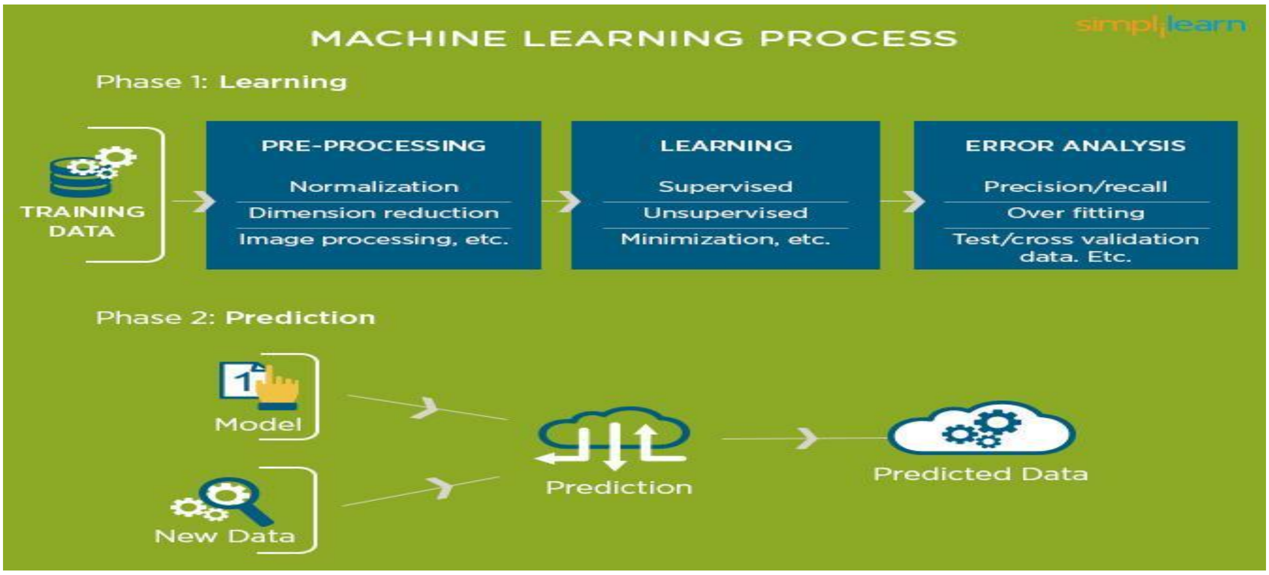


Figure 2.2.1 Machine Learning process

2.3 Uses of Machine Learning

Machine learning is a buzzword for today's technology, and it is growing very rapidly day by day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc.

Below are some most trending real-world applications of Machine Learning:

- Image Recognition
- Speech Recognition
- Traffic prediction
- Product recommendations
- Self – driving cars
- Email Spam and Malware Filtering
- Virtual Personal Assistant
- Online Fraud Detection
- Stock Market trading

2.4 Types of Machine Learning

It's essential to know the types of machine learning so that for any given task you may encounter, you can craft the proper learning environment and understand why what you did worked.

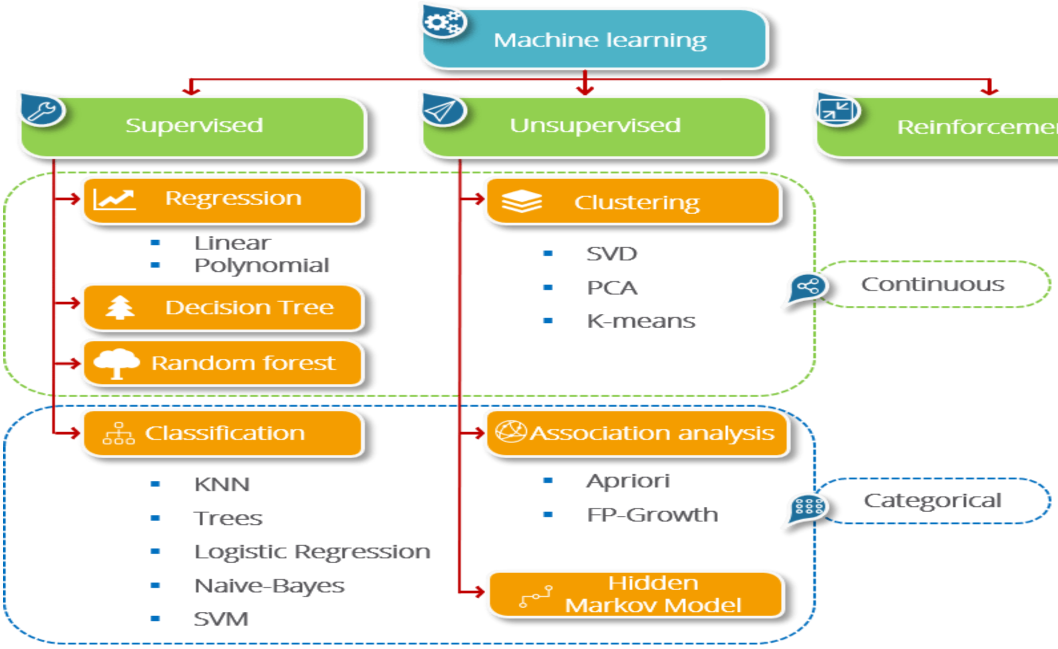


Figure 2.4.1 Types of machine learning

1. Supervised Learning

Supervised learning is one of the most basic types of machine learning. In this type, the machine learning algorithm is trained on labeled data. Even though the data needs to be labeled accurately for this method to work, supervised learning is extremely powerful when used in the right circumstances. In supervised learning, the ML algorithm is given a small training dataset to work with. This training dataset is a smaller part of the bigger dataset and serves to give the algorithm a basic idea of the problem, solution, and data points to be dealt with. The training dataset is also very similar to the final dataset in its characteristics and provides the algorithm with the labeled parameters required for the problem. The algorithm then finds relationships between the parameters given, essentially establishing a cause and effect relationship between the variables in the dataset. At the end of the training, the algorithm has an idea of how the data works and the relationship between the input and the output.

2. Unsupervised learning

Unsupervised machine learning holds the advantage of being able to work with unlabeled data. This means that human labor is not required to make the dataset machine-readable, allowing much larger datasets to be worked on by the program. In supervised learning, the labels allow the algorithm to find the exact nature of the relationship between any two data points. However, unsupervised learning does not have labels to work off of, resulting in the creation of hidden structures. Relationships between data points are perceived by the algorithm in an abstract manner, with no input required from human beings. The creation of these hidden structures is what makes unsupervised learning algorithms versatile. Instead of a defined and set problem statement, unsupervised learning algorithms can adapt to the data by dynamically changing hidden structures. This offers more post-deployment development than supervised learning algorithms.

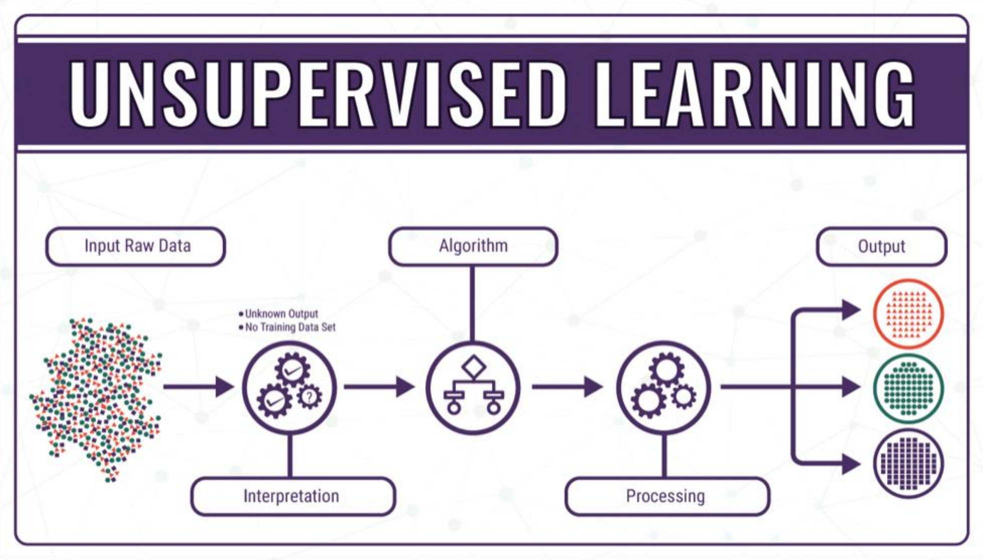


Figure 2.4.2 Unsupervised learning

3 Reinforcement Learning

Reinforcement learning directly takes inspiration from how human beings learn from data in their lives. It features an algorithm that improves upon itself and learns from new situations using a trial-and-error method. Favorable outputs are encouraged or ‘reinforced’, and non-favorable outputs are discouraged or ‘punished.

2.5 Relation between data mining, machine learning and deep learning

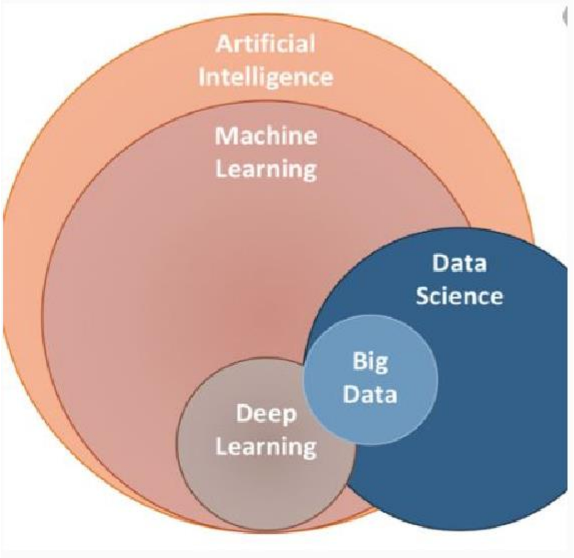


Figure 2.5.1 Relation between machine learning, deep learning and data science

Data Mining is the process of extracting useful information or knowledge (not so obvious patterns or insights) from huge sets of data. This knowledge could further be used for business applications such as Market analysis, Risk management, Fraud detection etc. Data mining could be accomplished by analysis, visualization or Machine Learning modelling.

Machine Learning is the process of gaining knowledge from past data, and using that knowledge to make future predictions. Example Algorithm: Support Vector Machines

Deep learning is a type of machine learning technique with more capabilities since it tries to mimic the neurons in human brain. It tries to learn a phenomenon as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts. Example Algorithm: Convolution Neural Networks

Let me quickly make a **distinction** between Machine Learning and Deep Learning -:

1. **Feature Engineering**

   It is the process of extracting the important features of the data using domain knowledge and then feeding them into the learning algorithm. This process ensures that the patterns in data are more visible to the learning algorithm.

   In Machine Learning, most of the applied features need to be identified by an expert. This process requires substantial time and effort.

   In Deep Learning, the algorithm would extract the features on its own.

2. **Problem Solving approach**

When solving a problem using traditional machine learning algorithm, it is recommended to break the problem down into smaller parts, solve them and later combine their output to get the final result. Whereas, Deep learning recommends solving the problem end to end.

Example, suppose there is an animal recognition problem and one image can have multiple animals

Machine learning would first identify the possible objects in the image and then predict or recognize each one of them separately. Whereas Deep learning would ingest the complete image and output the location and name of the object in one go.

3. **Data**

Deep learning algorithms generally need more data to perform well with respect to traditional machine learning algorithms.

4. **Hardware**

Deep learning is computationally intensive (because of more numbers of matrix multiplications) when compared to traditional machine learning and hence need high end machines for better performance.

5. **Training Time**

Deep Learning takes more time to train.

CHAPTER-3

PYTHON

3.1 What can Python do?

- Python can be used on a server to create web applications.

- Python can be used alongside software to create workflows.

- Python can connect to database systems. It can also read and modify files.

- Python can be used to handle big data and perform complex mathematics.

- Python can be used for rapid prototyping, or for production-ready software development.

3.1.1 Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

- Python has a simple syntax similar to the English language.

- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.

- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.

- Python can be treated in a procedural way, an object-orientated way or a functional way.

3.1.2 Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing large collections of Python files.

3.1.3 Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.

- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.

- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

3.2 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.

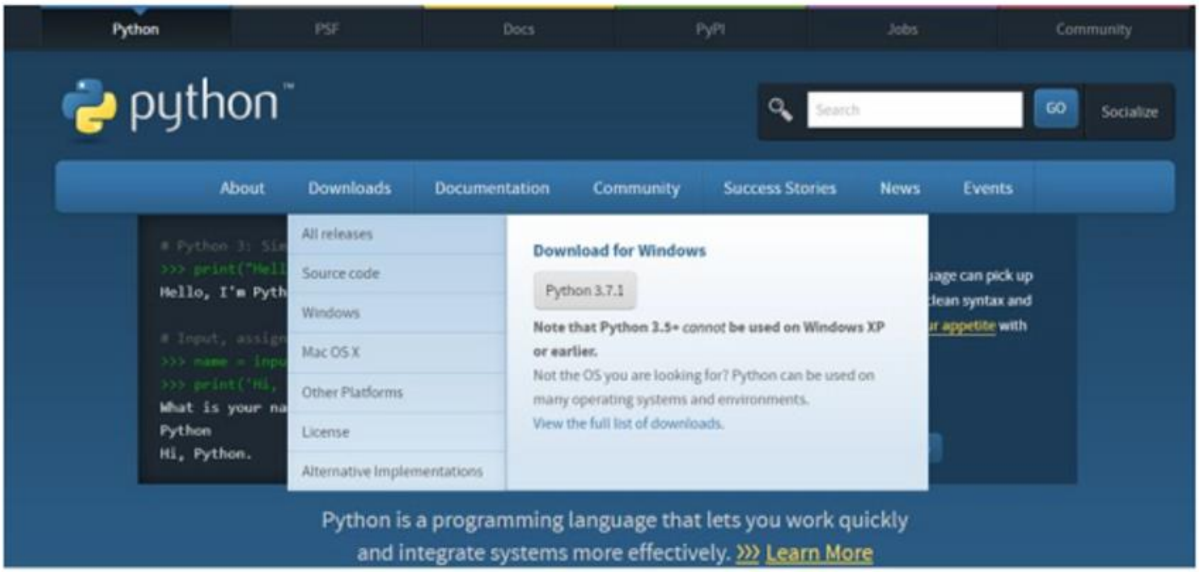
3.2.1 Installation (using python IDLE)

- 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.



```
# Python 3: Si
>>> print("Hell
Hello, I'm Pyth

# Input, assig
>>> name = inpu
>>> print('Hi,
What is your na
Python
Hi, Python.
```

Download for Windows

Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

uage can pick up
clean syntax and
ur appetite with

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Figure 3.2.1 Installation of python

3.2.2 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

.

- Anaconda is a package manager that quickly installs and manages packages.

In WINDOWS:

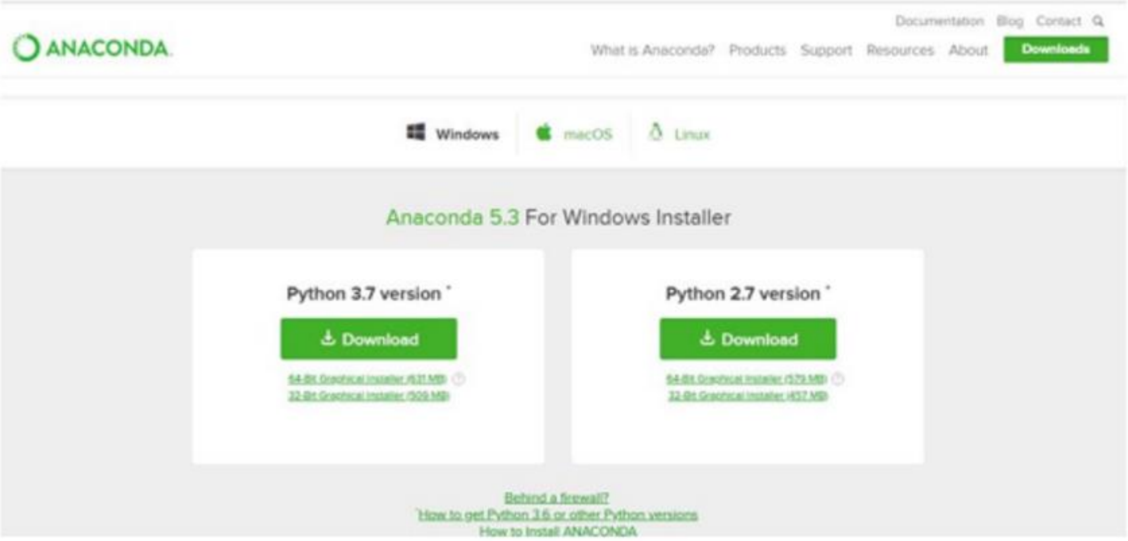
- Step 1: Open Anaconda.com/downloads in a web browser.

- 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)



ANACONDA.
Documentation
Blog
Contact
What is Anaconda?
Products
Support
Resources
About
Downloads
Windows
macOS
Linux
Anaconda 5.3 For Windows Installer
Python 3.7 version *
Download
64-Bit Graphical Installer (631 MB)
32-Bit Graphical Installer (509 MB)
Python 2.7 version *
Download
64-Bit Graphical Installer (579 MB)
32-Bit Graphical Installer (457 MB)
Behind a firewall?
*How to get Python 3.6 or other Python versions
How to Install ANACONDA

Figure 3.2.2 Installation of Anaconda

3.3 Features

Easy to code:

Python is a high-level programming language. Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.

Free and open source:

Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword.

Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.

Object-Oriented Language:

One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.

GUI Programming Support:

Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tkip Python. PyQt5 is the most popular option for creating graphical apps with Python.

High-Level Language:

Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.

Extensible feature:

Python is an Extensible language. We can write some Python code into the C or C++ language and also, we can compile that code in C/C++ language.

Python is Portable language:

Python is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.

Python is Integrated language:

Python is also an Integrated language because we can easily integrate python with other languages like c, c++, etc.

Interpreted Language:

Python is an Interpreted Language because Python code is executed line by line at a time. Like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called bytecode.

Large Standard Library:

Python has a large standard library which provides a rich set of modules and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

Dynamically Typed Language:

Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don’t need to specify the type of variable.

3.4 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.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Standard Data Types:

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. 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 −

1. String
2. List
3. Tuple
4. Dictionary
5. Numbers

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)

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

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.

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.

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists −

```
tuple=('abcd',786,2.23,'john',70.2)
list=['abcd',786,2.23,'john',70.2] tuple[2]=1000  #
Invalidsyntaxwithtuple list[2]=1000               #
Valid syntaxwithlist
```

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 ([]).

3.5 Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

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 (( )).

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

The first statement of a function can be an optional statement - the documentation string of the function or docstring.The code block within every function starts with a colon (:) and is indented.

Syntax:

```
def functionname (parameters):
        --------------
        return [expression]
```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Example:

The following function takes a string as input parameter and prints it on standard screen.

```
def printme(str):
  #"This prints a passed string into this function"
  print str
  return
```

Function Arguments:

You can call a function by using the following types of formal arguments –

Required arguments: Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

Keyword arguments: Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Default arguments: A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Variable-length arguments: You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

The return Statement:

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.

Example:

```
 # Function definition is here def sum (arg1, arg2):
 # Add both the parameters and return them." total = arg1 + arg2
     print "Inside the function : ", total return total;
   # Now you can call sum function total = sum( 10, 20 );
     print "Outside the function : ", total
```

3.6 OOPs Concepts

Python is an object-oriented programming language. What this mean is we can solve a problem in Python by creating objects in our programs. In this guide, we will discuss OOPs terms such as class, objects, methods etc. along with the Object-oriented programming features such as inheritance, polymorphism, abstraction, encapsulation.

Object:

An object is an entity that has attributes and behaviour. For example, Ram is an object who has attributes such as height, weight, color etc. and has certain behaviours such as walking, talking etc,

Class:

A class is a blueprint for the objects. For example, Ram, Shyam, Steve, Rick are all objects so we can define a template (blueprint) class Human for these objects. The class can define the common attributes and behaviours of all the objects.

Methods:

As we discussed above, an object has attributes and behaviours. These behaviours are called methods in programming.

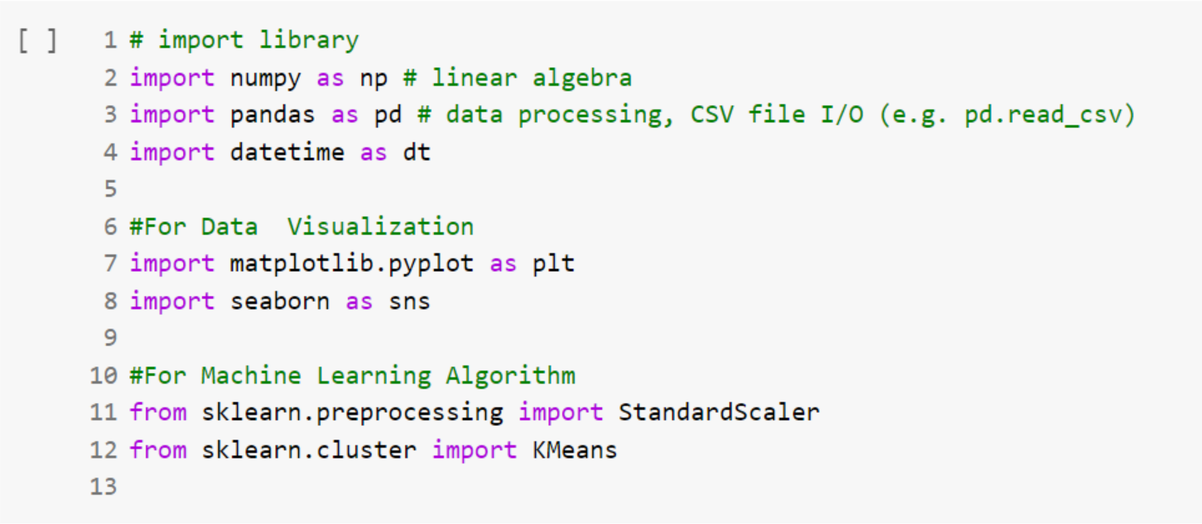
CHAPTER-4

Customer Segmentation based on purchase behaviour

4.1 Project requirements:

4.1.1Packages Used:

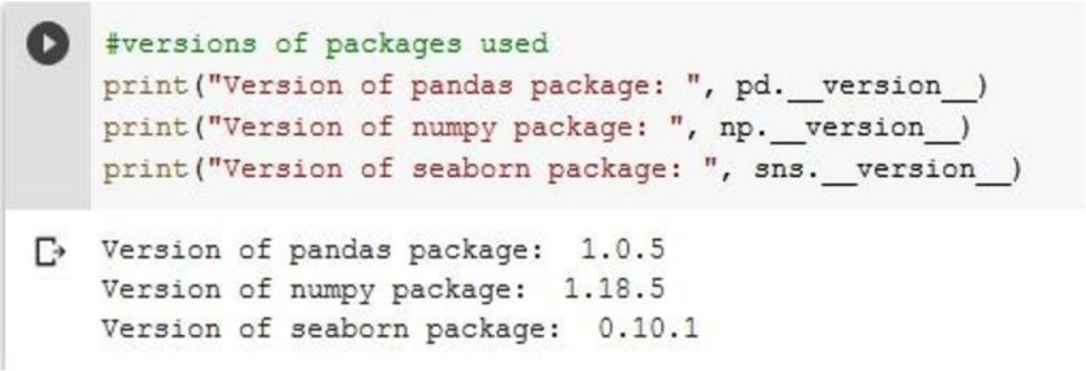
I have used the below mentioned packages initially.



```
# import library
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import datetime as dt

#For Data  Visualization
import matplotlib.pyplot as plt
import seaborn as sns

#For Machine Learning Algorithm
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

```

Figure 4.1.1 Packages used

4.1.2 Versions of the packages



```
#versions of packages used
print("Version of pandas package: ", pd.__version__)
print("Version of numpy package: ", np.__version__)
print("Version of seaborn package: ", sns.__version__)
```

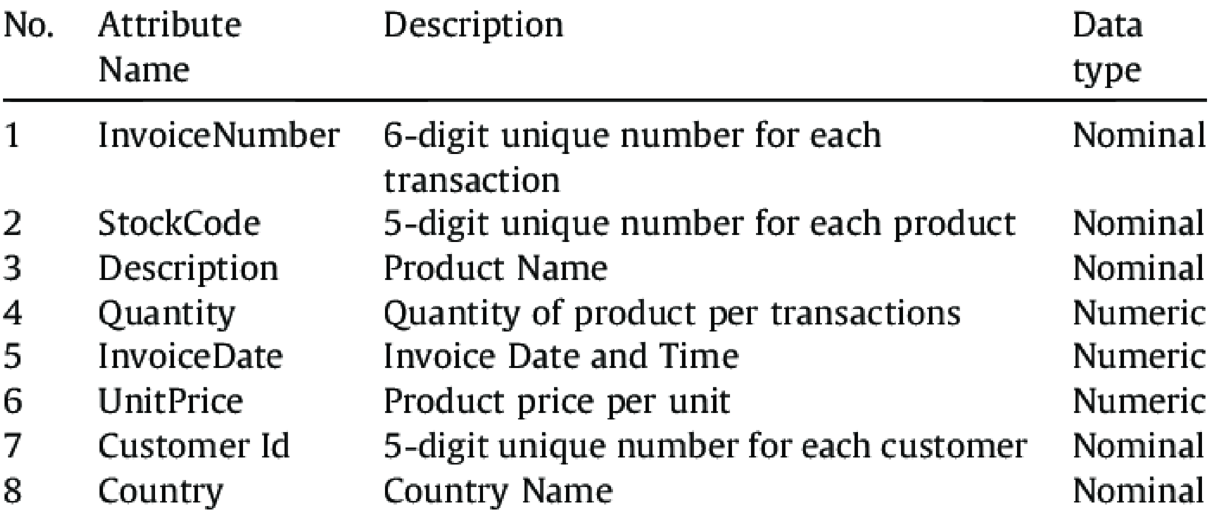
```
Version of pandas package:  1.0.5
Version of numpy package:  1.18.5
Version of seaborn package:  0.10.1
```

Figure 4.1.2 Version of packages

4.2 Project Statement

We have to perform cohort and RFM analysis to understand the value derived from different customer segments. Further, we will divide customers in different clusters based on the analysis by using K-means algorithm.

4.3 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

Figure 4.3.1 Description of variables

CHAPTER-5

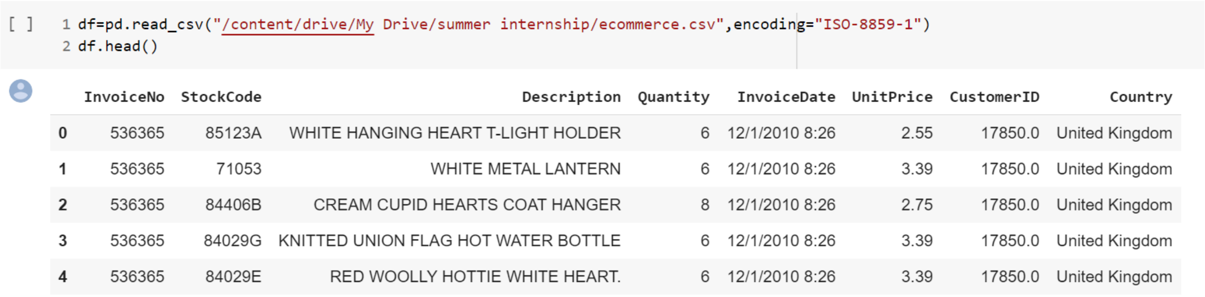
DATA PREPROCESSING

• 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:

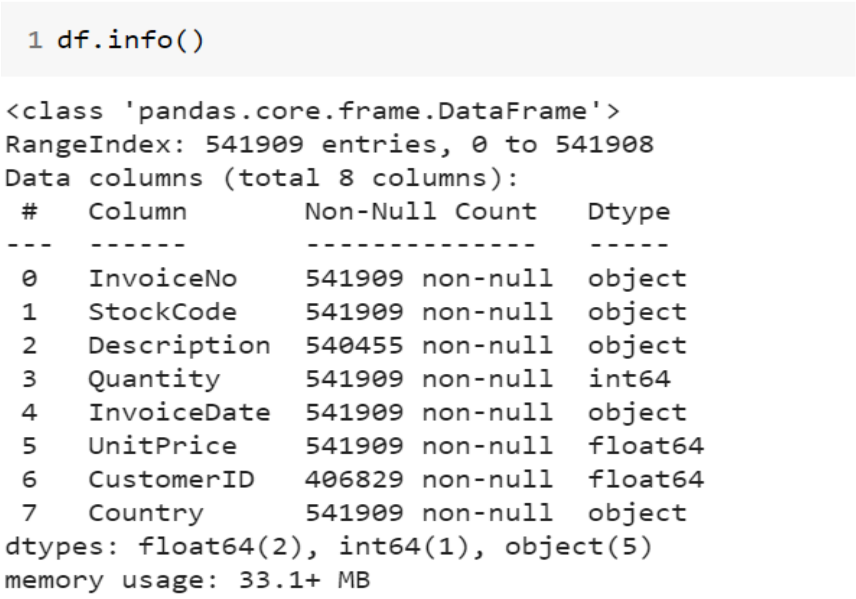
Pandas in python provide an interesting method read_csv(). The read_csv function reads the entire dataset from a comma separated values file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every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



```
df=pd.read_csv("/content/drive/My Drive/summer internship/ecommerce.csv",encoding="ISO-8859-1")
df.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

Figure 5.1.1 reading the dataset

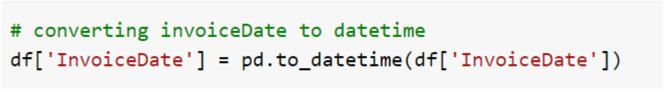


```
df.info()
```

```
<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  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

Figure 5.1.2 df.info()

From the above output we can see that InvoiceDate is off object data type but is should be of datetime data type.



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

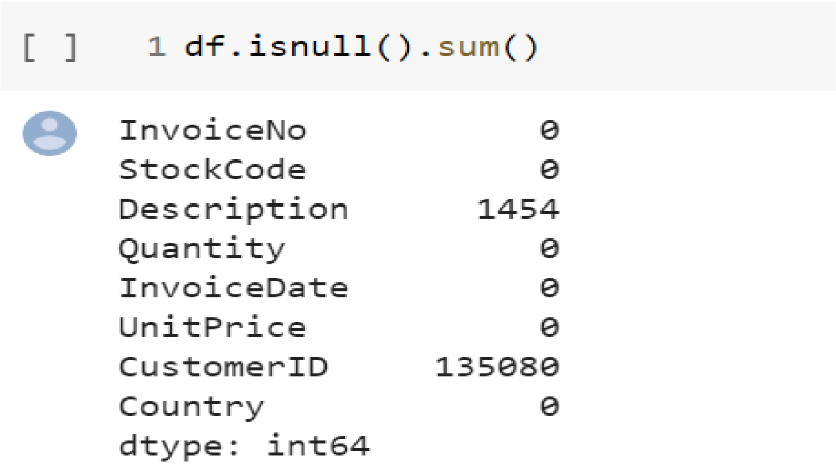
Figure 5.1.3 Datetime conversion

5.2 Handling Missing Values

There are a number of schemes that have been developed to indicate the presence of missing data in a table or DataFrame. Generally, they revolve around one of two strategies: using a mask that globally indicates missing values, or choosing a sentinel value that indicates a missing entry. In the masking approach, the mask might be an entirely separate Boolean array, or it may involve appropriation of one bit in the data representation to locally indicate the null status of a value.In the sentinel approach, the sentinel value could be some data-specific convention, such as indicating a missing integer value with -9999 or some rare bit pattern, or it could be a more global convention, such as indicating a missing floating-point value with NaN (Not a Number), a special value which is part of the IEEE floating-point specification.

isnull( ) function returns true if there is a null value in a cell. And sum( ) is used to count the number of true values returned by the isnull( ) function.

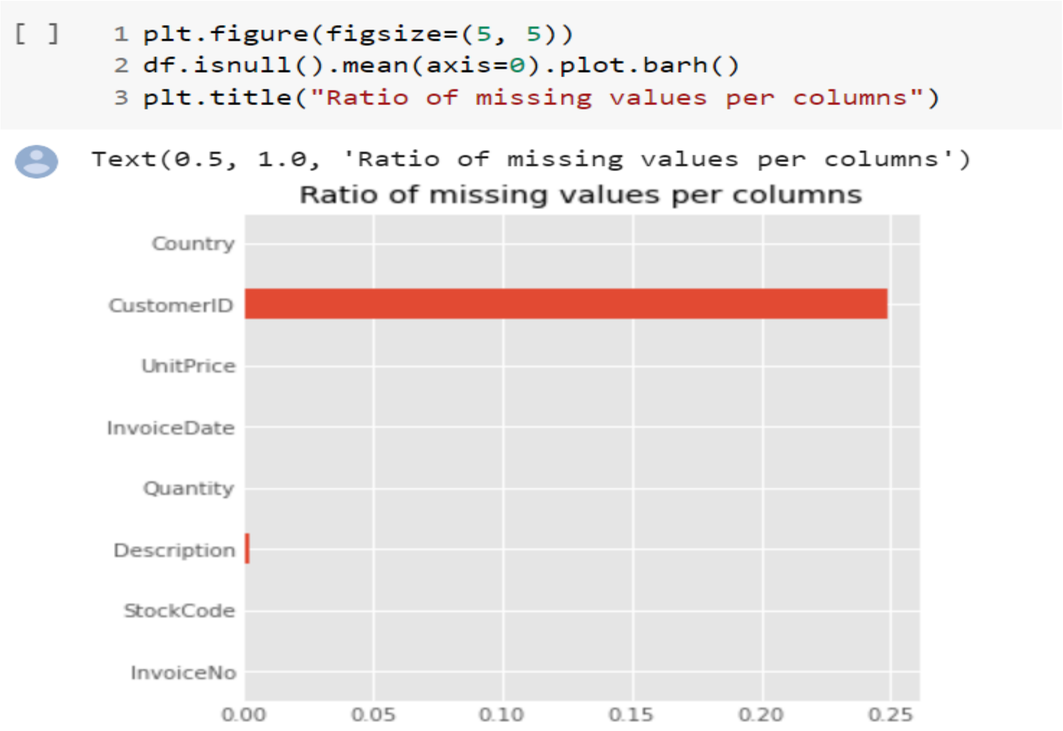
So, by executing the below statement we get the number of null values in each column.



```
[ ]   df.isnull().sum()
```

```
InvoiceNo           0
StockCode           0
Description      1454
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     135080
Country             0
dtype: int64
```

Figure 5.2.1 Sum of null values



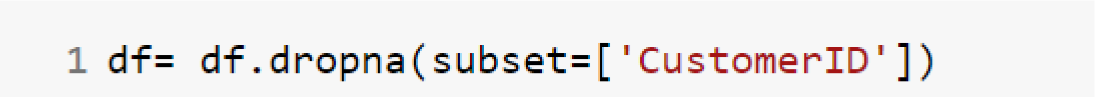
```
[ ]   plt.figure(figsize=(5, 5))
      df.isnull().mean(axis=0).plot.barh()
      plt.title("Ratio of missing values per columns")
```

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

Figure 5.2.2 Bar graph for missing values

From the above output we can say that the columns Description, CustomerID have null values. So we have to drop them.

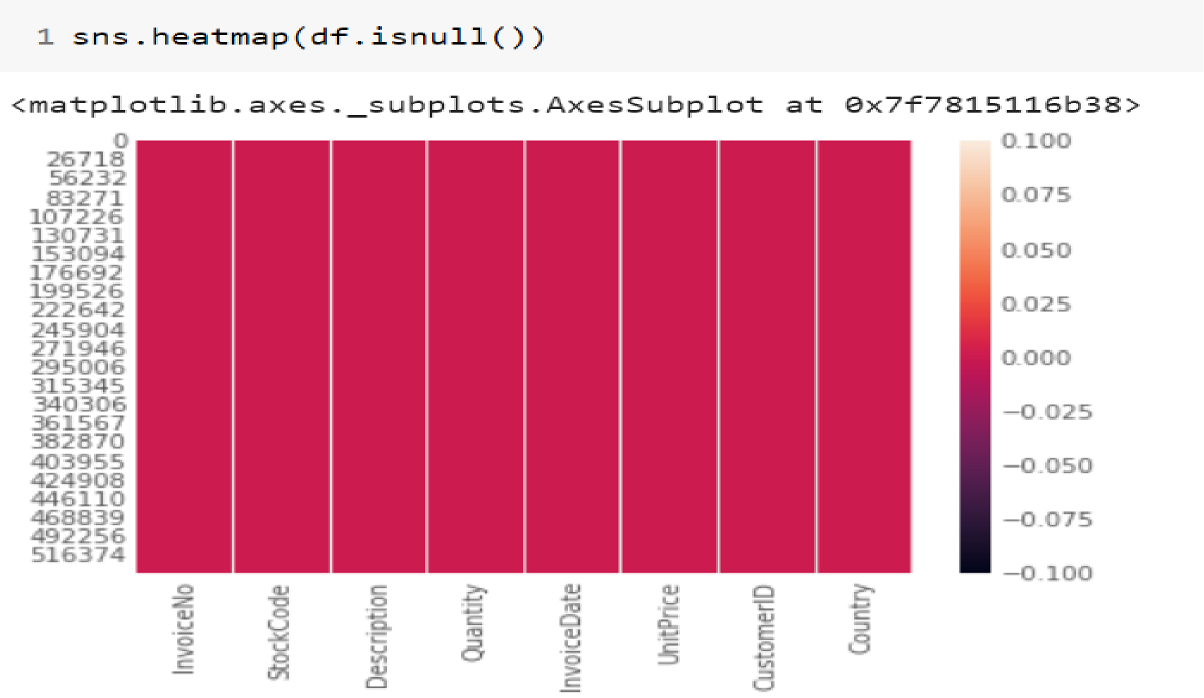
The dropna() function is used to remove missing values.



```
df= df.dropna(subset=['CustomerID'])
```

Figure 5.2.3 Dropping missing values

After this command all the missing values are removed. In order to check whether all the missing values are removed or not we can use heatmap for visualization.



```
sns.heatmap(df.isnull())
```

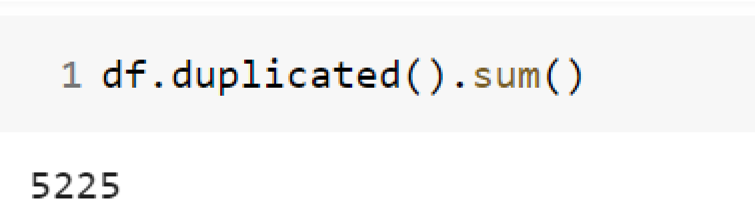
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7815116b38>
```

Figure 5.2.4 Heatmap for missing values

From the above heatmap we can say that there are no missing values.

5.3 Handling duplicate values

To know how many duplicate values are present in our dataframe, we have to use the following command:



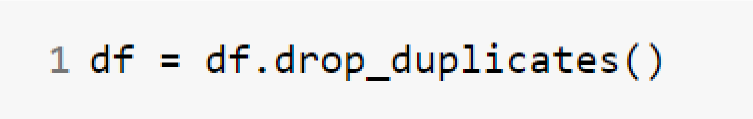
```
df.duplicated().sum()
```

```
5225
```

Figure 5.2.5 No.of duplicates

So, from the above output we can know that there are 5225 missing values in our dataframe.

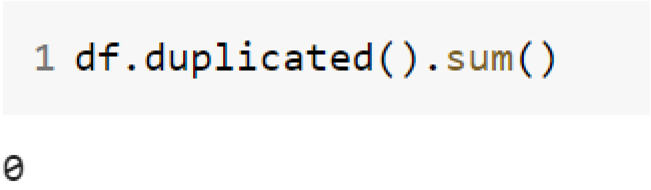
In order to get rid of duplicate values, we have to use the following command:



```
df = df.drop_duplicates()
```

Figure 5.2.6 Dropping duplicates

Let’s check again whether the duplicates are removed or not. Let’s again use the command:



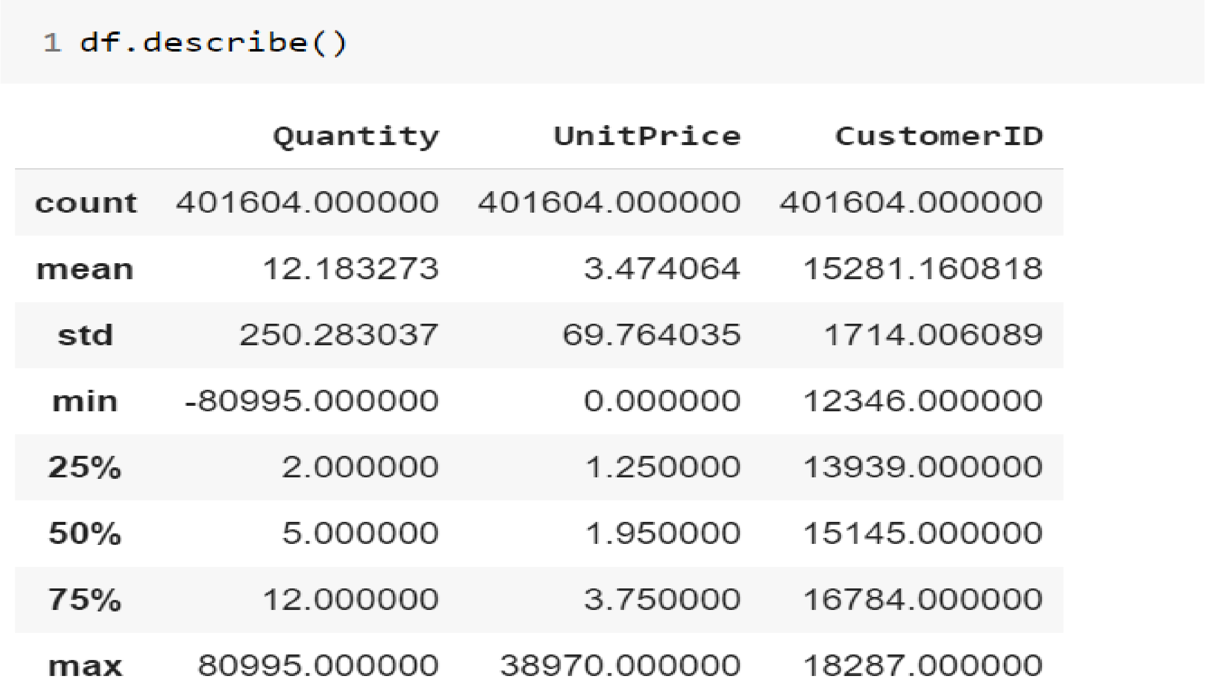
```
df.duplicated().sum()
```

```
0
```

Figure 5.2.7 No. of duplicates

Finally, we don’t have any missing values.

5.4 Data Exploration



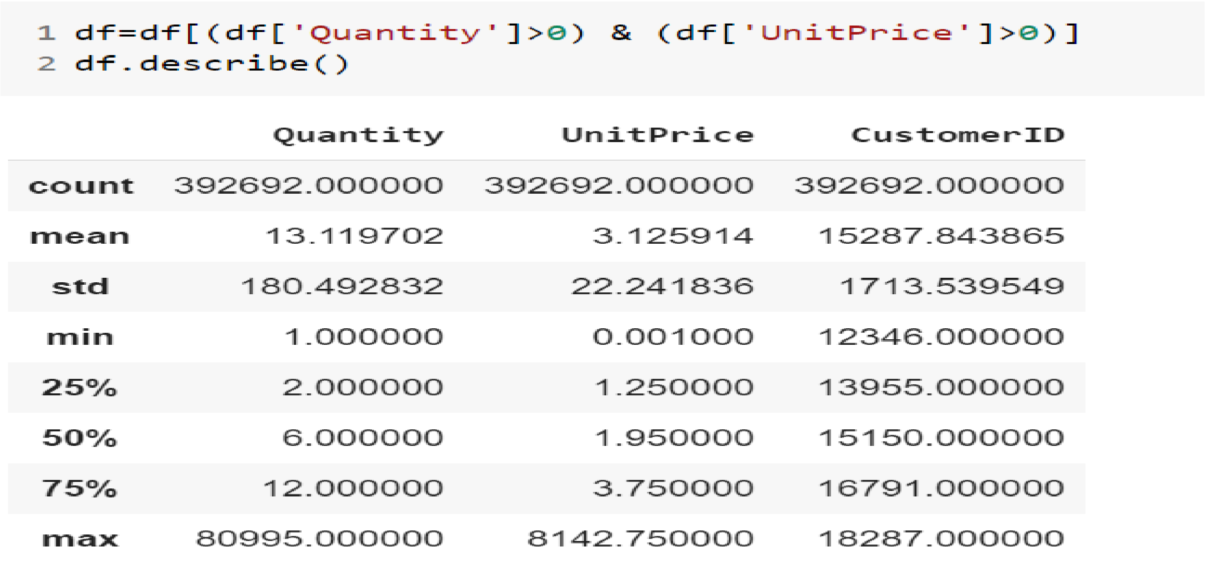
```
df.describe()
```

	Quantity	UnitPrice	CustomerID
count	401604.000000	401604.000000	401604.000000
mean	12.183273	3.474064	15281.160818
std	250.283037	69.764035	1714.006089
min	-80995.000000	0.000000	12346.000000
25%	2.000000	1.250000	13939.000000
50%	5.000000	1.950000	15145.000000
75%	12.000000	3.750000	16784.000000
max	80995.000000	38970.000000	18287.000000

Figure 5.2.8 df.describe()

From the above output the Quantity column has a negative value in it. But logically it should not, because the customer can not buy a negative amount of merchant. It is impossible. But maybe it is a way that store register such sales as instalment plan. But maybe I am wrong.

Also the UnitPrice column have zero price rows. It means that we sell for free.So we should consider only the data which has Quantity>0 and UnitPrice>0.

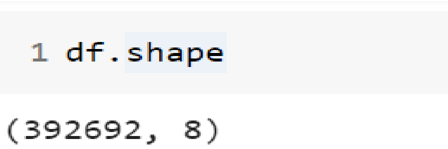


```
df=df[(df['Quantity']>0) & (df['UnitPrice']>0)]
df.describe()
```

	Quantity	UnitPrice	CustomerID
count	392692.000000	392692.000000	392692.000000
mean	13.119702	3.125914	15287.843865
std	180.492832	22.241836	1713.539549
min	1.000000	0.001000	12346.000000
25%	2.000000	1.250000	13955.000000
50%	6.000000	1.950000	15150.000000
75%	12.000000	3.750000	16791.000000
max	80995.000000	8142.750000	18287.000000

Figure 5.2.9 Quality level check

So, here Quantity>0.



```
df.shape
```

```
(392692, 8)
```

Figure 5.2.10 df.shape()

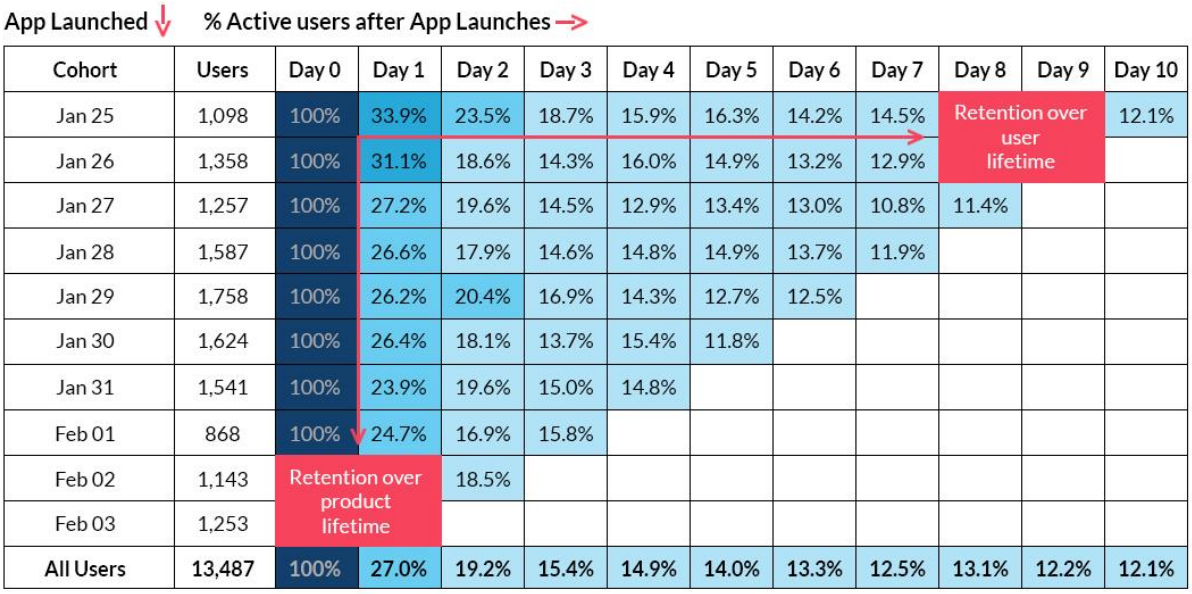
CHAPTER-6
Cohort Analysis

6.1 What is cohort analysis

A cohort is simply a subset of users grouped by shared characteristics. In the context of business analytics, a cohort usually refers to a subset of users specifically segmented by acquisition date (i.e. the first time a user visits your website).

A “cohort analysis,” then, simply allows you to compare the behavior and metrics of different cohorts over time. You can then find the highest-performing (or lowest-performing) cohorts, and what factors are driving this performance.

This is an example for cohort analysis is done.



App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Figure 6.1.1 Cohort analysis

Types of cohorts

1. Time Cohorts are customers who signed up for a product or service during a particular time frame. Analyzing these cohorts shows the customers’ behaviour depending on the time they started using the company’s products or services. The time may be monthly or quarterly even daily.

2. Behaviour cohorts are customers who purchased a product or subscribed to a service in the past. It groups customers by the type of product or service they signed up. Customers who signed up for basic level services might have different needs than those who signed up for advanced services. Understanding the needs of the various cohorts can help a company design custom-made services or products for particular segments.

3. Size cohorts refer to the various sizes of customers who purchase company's products or services. This categorization can be based on the amount of spending in some periodic time after acquisition or the product type that the customer spent most of their order amount in some period of time.

Since, we will be performing Cohort Analysis based on Transaction records of Customers, we will be Dealing with Mainly:

1. Invoice Date
2. CustomerID
3. Price
4. and Quantity columns in this Analsyis.

The Following steps will performed to generate the Cohort Chart of Retention Rate :

1. Month Extraction from InvioceDate column
2. Assigning Cohort to Each Transaction
3. Assigning Cohort Index to each transaction
4. Calculating number of unique customers in each Group of (ChortDate,Index)
5. Creating Cohort Table for Retention Rate
6. Creating the Cohort Chart using the Cohort Table.

6.2 Performing cohort analysis

For cohort analysis, there are a few labels that we have to create:

1.Invoice period: A string representation of the year and month of a single transaction/invoice.

2.Cohort group: A string representation of the year and month of a customer's first purchase. This label is common across all invoices for a particular customer.

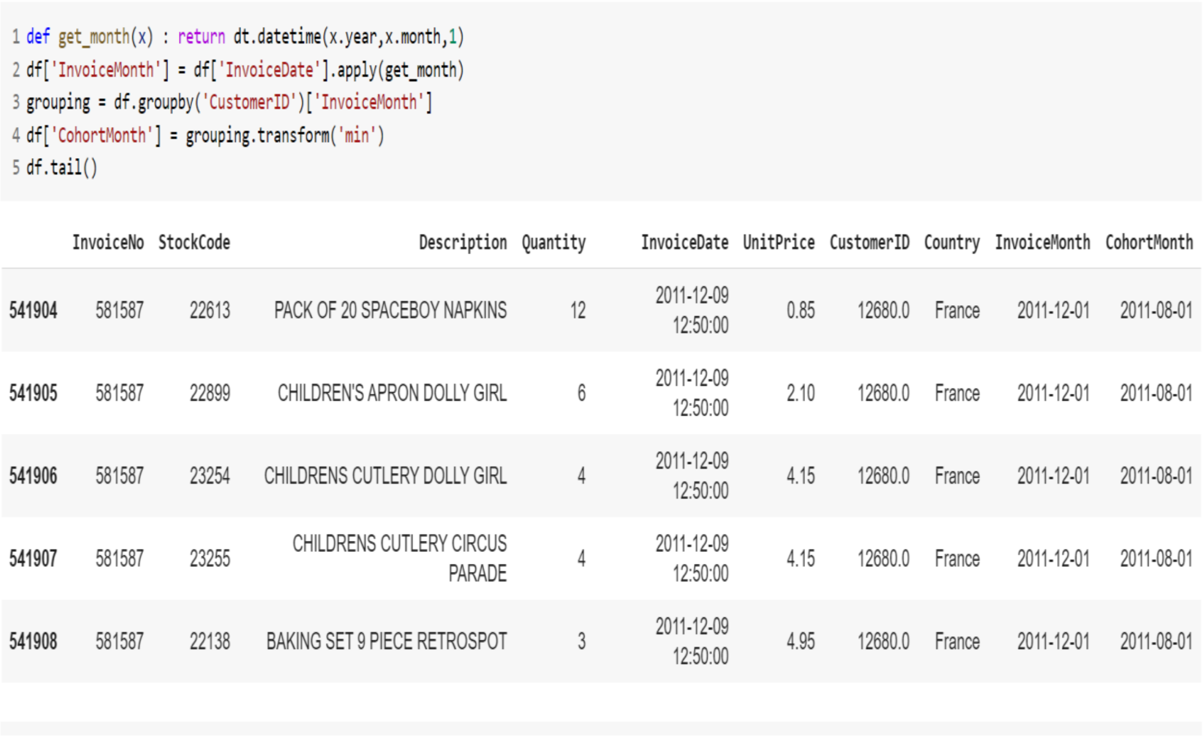
3.Cohort period / Cohort Index: A integer representation a customer's stage in its "lifetime". The number represents the number of months passed since the first purchase.

Step 1: Let's extract the Month of the Purchase of each transaction.

First, we will create a function, which takes any date and returns the formatted date with day value as 1st of the same month and Year. Later we will use the function created above to convert all the invoice dates into respective month date format.

Now we will use the Invoice Month created to generate cohort Date values Creating a groupby object with respect to CustomerID variable, and selecting InvoiceDay for further calculations.

Now let's assign the acquisition Cohort value to each transaction based on the minimum value of InvoiceMonth of the group it belongs to.



```
def get_month(x) : return dt.datetime(x.year,x.month,1)
df['InvoiceMonth'] = df['InvoiceDate'].apply(get_month)
grouping = df.groupby('CustomerID')['InvoiceMonth']
df['CohortMonth'] = grouping.transform('min')
df.tail()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth
541904	581587	22613	PACK OF 20 SPACEBOY NAPKINS	12	2011-12-09 12:50:00	0.85	12680.0	France	2011-12-01	2011-08-01
541905	581587	22899	CHILDREN'S APRON DOLLY GIRL	6	2011-12-09 12:50:00	2.10	12680.0	France	2011-12-01	2011-08-01
541906	581587	23254	CHILDRENS CUTLERY DOLLY GIRL	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01
541907	581587	23255	CHILDRENS CUTLERY CIRCUS PARADE	4	2011-12-09 12:50:00	4.15	12680.0	France	2011-12-01	2011-08-01
541908	581587	22138	BAKING SET 9 PIECE RETROSPOT	3	2011-12-09 12:50:00	4.95	12680.0	France	2011-12-01	2011-08-01

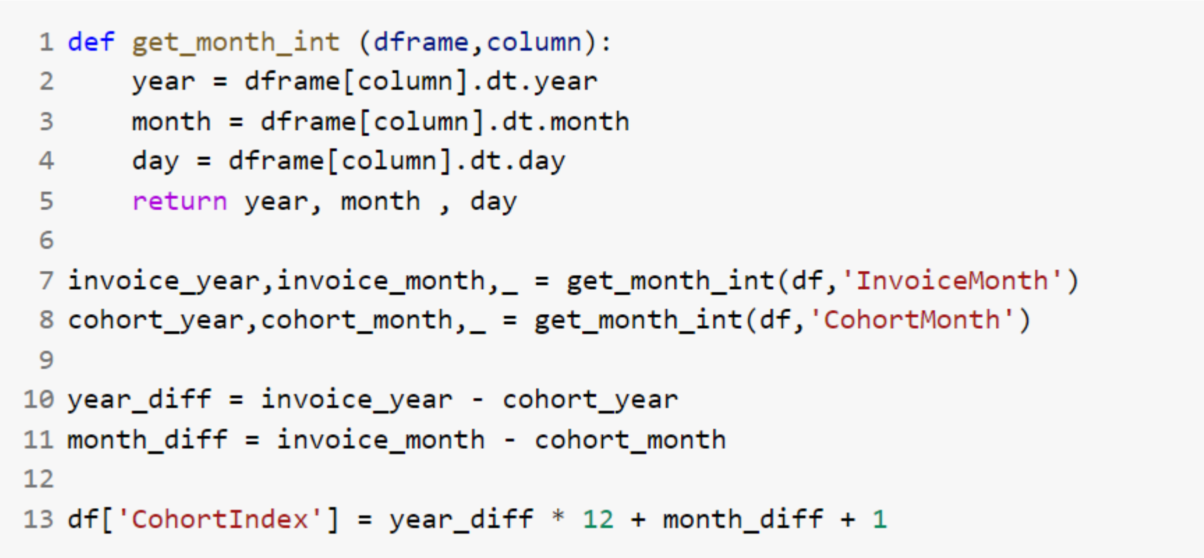
Figure 6.4.2 Function get_month()

Step 2: Calculating time offset in Months i.e. Cohort Index:

Calculating time offset for each transaction will allows us to report the metrics for each cohort in a comparable fashion.First, we will create 4 variables that capture the integer value of years, months for Invoice and Cohort Date using the get_date_int() function .
We will use this function to extract the integer values for Invoice as well as Cohort Date in 3 separate series for each of the two columns

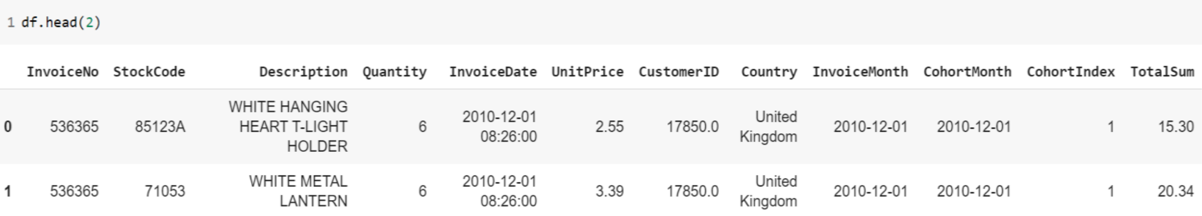
We will use the integer values extracted above to calculate business metrics for our time cohorts:

We will calculate the difference between the Invoice Dates and Cohort dates in years, months separately. Then calculate the total Months difference between the two.
This will be our Months offset or cohort Index, which we will use in the next section to calculate retention rate.



```
def get_month_int (dframe,column):
    year = dframe[column].dt.year
    month = dframe[column].dt.month
    day = dframe[column].dt.day
    return year, month , day

invoice_year,invoice_month,_ = get_month_int(df,'InvoiceMonth')
cohort_year,cohort_month,_ = get_month_int(df,'CohortMonth')

year_diff = invoice_year - cohort_year
month_diff = invoice_month - cohort_month

df['CohortIndex'] = year_diff * 12 + month_diff + 1
```

Figure 6 .1.3 Function get_month_int



```
df.head(2)
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth	CohortIndex	TotalSum
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01	2010-12-01	1	15.30
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	2010-12-01	1	20.34

Figure 6.1.4 Displaying

Step 3: Calculating Retention Count:

The percentage of active customers compared to the total number of customers after a specific time interval is called retention rate. In the this section, we will calculate retention count for each cohort Month paired with cohort Index.

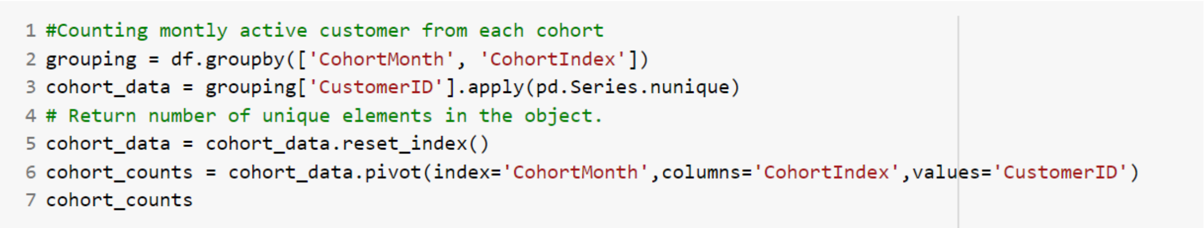
First, we will group by our dataset with respect to CohortMonth and CohortIndex.

Next, we will count number of unique customer Id's falling in each group of CohortMonth and CohortIndex.

This will give us number of customers (Retained Customers) from each cohort who bought items after a n Months where n is CohortIndex and store them in a new dataframe cohort Data.

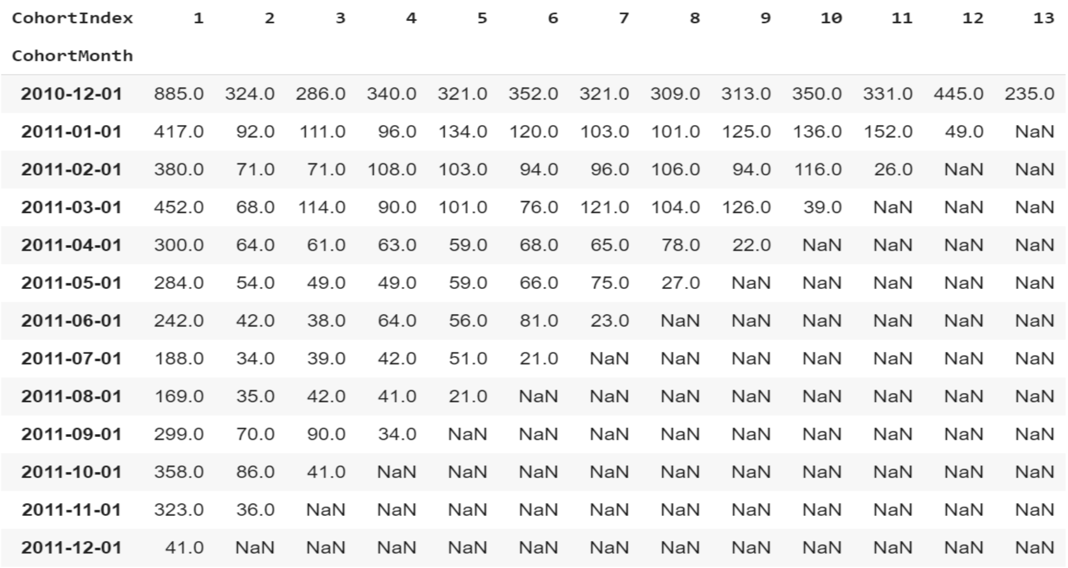
Now that we have count of retained customer for each cohortMonth and cohortIndex let's convert it into suitable format for readability and Calculation of retention Rate for each Cohort.

We will create a pivot table for this purpose



```
#Counting montly active customer from each cohort
grouping = df.groupby(['CohortMonth', 'CohortIndex'])
cohort_data = grouping['CustomerID'].apply(pd.Series.nunique)
# Return number of unique elements in the object.
cohort_data = cohort_data.reset_index()
cohort_counts = cohort_data.pivot(index='CohortMonth',columns='CohortIndex',values='CustomerID')
cohort_counts
```

Figure 6.1.5 Calculation of retention count



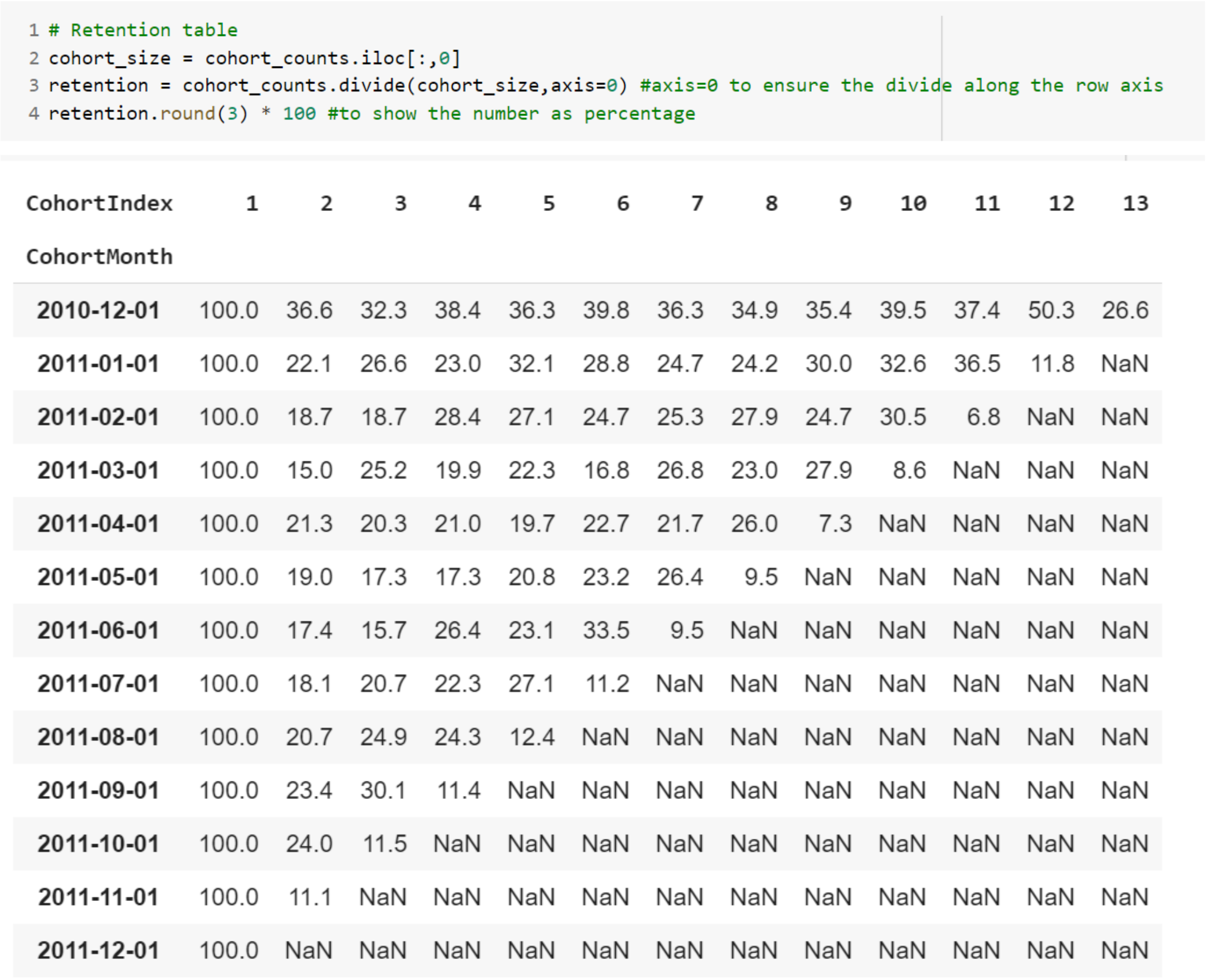
CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	885.0	324.0	286.0	340.0	321.0	352.0	321.0	309.0	313.0	350.0	331.0	445.0	235.0
2011-01-01	417.0	92.0	111.0	96.0	134.0	120.0	103.0	101.0	125.0	136.0	152.0	49.0	NaN
2011-02-01	380.0	71.0	71.0	108.0	103.0	94.0	96.0	106.0	94.0	116.0	26.0	NaN	NaN
2011-03-01	452.0	68.0	114.0	90.0	101.0	76.0	121.0	104.0	126.0	39.0	NaN	NaN	NaN
2011-04-01	300.0	64.0	61.0	63.0	59.0	68.0	65.0	78.0	22.0	NaN	NaN	NaN	NaN
2011-05-01	284.0	54.0	49.0	49.0	59.0	66.0	75.0	27.0	NaN	NaN	NaN	NaN	NaN
2011-06-01	242.0	42.0	38.0	64.0	56.0	81.0	23.0	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	188.0	34.0	39.0	42.0	51.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	169.0	35.0	42.0	41.0	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	299.0	70.0	90.0	34.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	358.0	86.0	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	323.0	36.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	41.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 6.1.6 Pivot table for retention count

Step:4 Now we will use the the Cohort Count dataframe to calculate the retention rate

We will store the 1st column as Cohort size i.e total Number of Customers in that Cohort. We will divide the values in other columns with Cohort Size in order to calculate the retention rate i.e Number of Customers in Each Cohort Index.

Now we will divide the values in all the columns with values in Column 1 Row-Wise



```python
# Retention table
cohort_size = cohort_counts.iloc[:,0]
retention = cohort_counts.divide(cohort_size,axis=0) #axis=0 to ensure the divide along the row axis
retention.round(3) * 100 #to show the number as percentage
```

CohortIndex	1	2	3	4	5	6	7	8	9	10	11	12	13
CohortMonth													
2010-12-01	100.0	36.6	32.3	38.4	36.3	39.8	36.3	34.9	35.4	39.5	37.4	50.3	26.6
2011-01-01	100.0	22.1	26.6	23.0	32.1	28.8	24.7	24.2	30.0	32.6	36.5	11.8	NaN
2011-02-01	100.0	18.7	18.7	28.4	27.1	24.7	25.3	27.9	24.7	30.5	6.8	NaN	NaN
2011-03-01	100.0	15.0	25.2	19.9	22.3	16.8	26.8	23.0	27.9	8.6	NaN	NaN	NaN
2011-04-01	100.0	21.3	20.3	21.0	19.7	22.7	21.7	26.0	7.3	NaN	NaN	NaN	NaN
2011-05-01	100.0	19.0	17.3	17.3	20.8	23.2	26.4	9.5	NaN	NaN	NaN	NaN	NaN
2011-06-01	100.0	17.4	15.7	26.4	23.1	33.5	9.5	NaN	NaN	NaN	NaN	NaN	NaN
2011-07-01	100.0	18.1	20.7	22.3	27.1	11.2	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-08-01	100.0	20.7	24.9	24.3	12.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-09-01	100.0	23.4	30.1	11.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-10-01	100.0	24.0	11.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-11-01	100.0	11.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2011-12-01	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Figure 6.1.5 Percentage of retention rate

The retention Rate dataframe represent Customer retained across Cohurts. We can read it as following:

1. Index value represents the Cohort
2. Columns represent the number of months since current Cohort

for example, the value at index 2010-12-01 column 7 is 34.8 and Represents 34.8% of customers from cohort 2010-12 were retained in 7th Month.

Also, you can see from the retention Rate DataFrame:

Retention Rate 1st index i.e 1st month is 100% as all the customers for that particular customers signed up in 1st Month

Retention Rate may increase or decrease in Subsequent Indexes.

Values towards Bottom Right have a lot of NaN values.

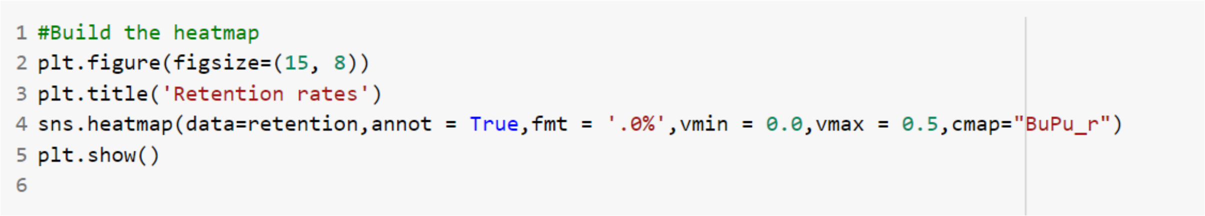
Can You think of the reason for the number of NaN values towards RIGHT BOTTOM?

Yes, as you might have Guessed, The Cohort Indexes towards Right represent Number of Months away the Values are from Current Cohort. Thus, the values at Bottom Right cells don't have data of acquisition as they are too recent.

Step 5: Visualizing the Above Retention rate

Visualization allows decision makers to read and interpret patterns in data easily. This allows the organization to act identify and Act on Trends Faster. In this section, we will visualize cohort analysis using HeatMap as it is very easy to read and Contains color Mapping representing the Values.

Before we starting plotting our HeatMap, let's set the index of our Retention rate dataframe to a more readable string format



```
#Build the heatmap
plt.figure(figsize=(15, 8))
plt.title('Retention rates')
sns.heatmap(data=retention,annot = True,fmt = '.0%',vmin = 0.0,vmax = 0.5,cmap="BuPu_r")
plt.show()

```

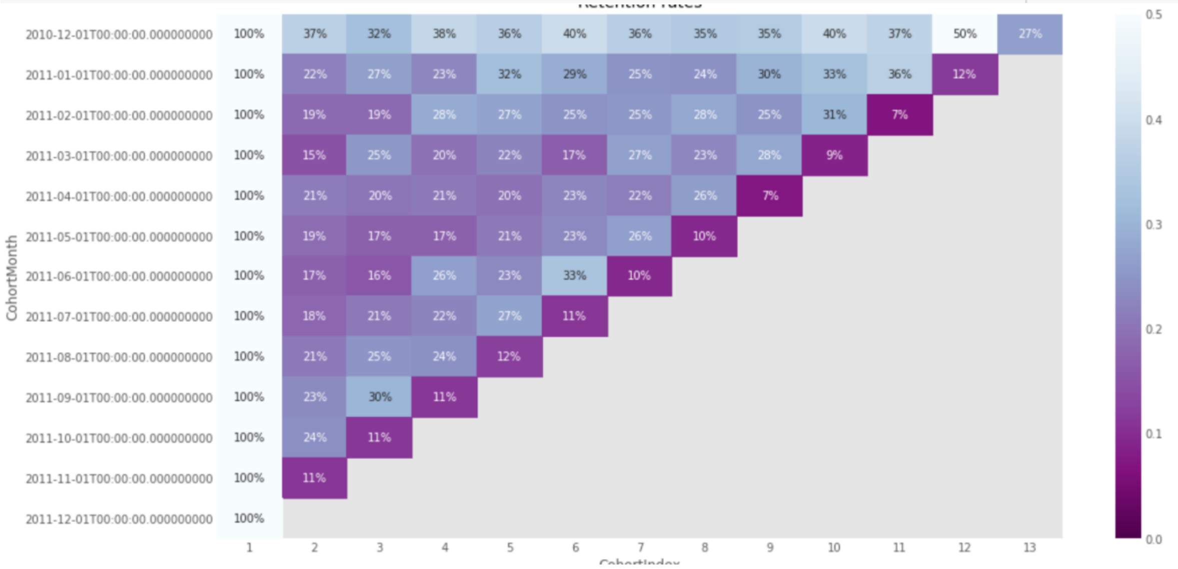


Figure 6.1.6 Heatmap of retention

This marks the end of Cohort Analysis for Retention Rate of Customers.

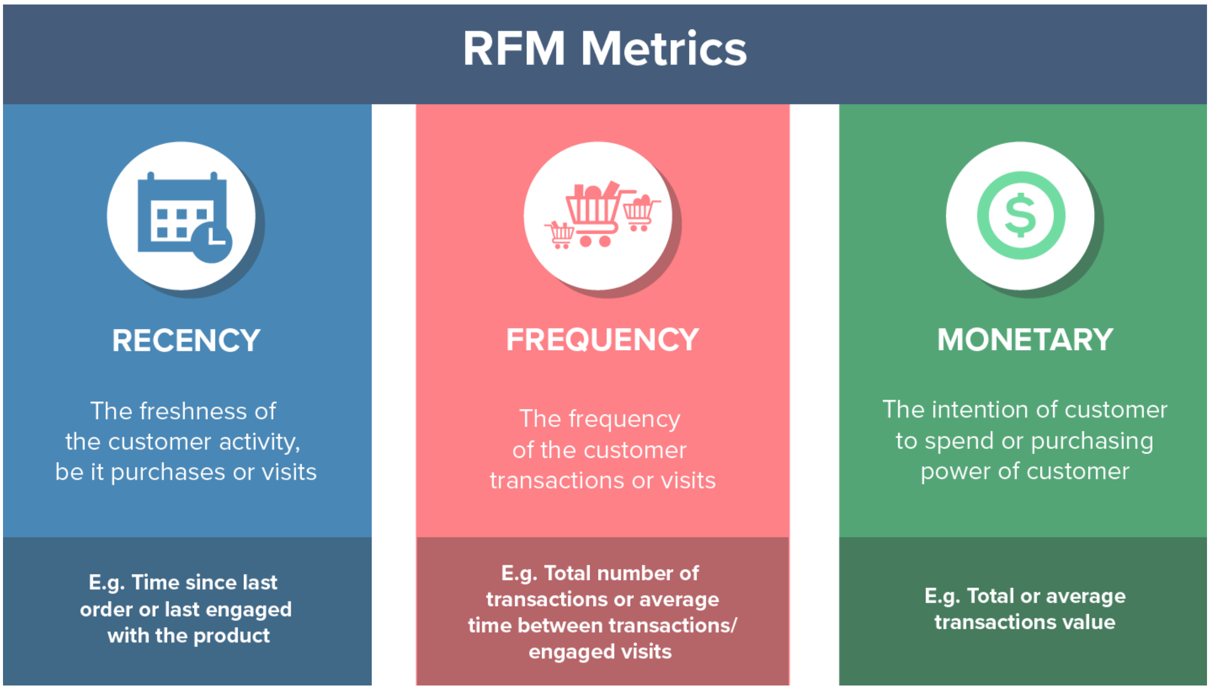
The Heatmap generated can be used to present the cohort Analysis result to Decision Makers and Executives in order to generate insights and Take actions

AS we discussed at the start of this Analysis, Cohort Analytics can be used not only for performing Customer Retention rate but also many other business metrics.

CHAPTER 7
RFM ANALYSIS

7.1 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 behaviour because frequency and monetary value affects a customer's lifetime value, and recency affects retention, a measure of engagement.



RFM Metrics

RECENCY

The freshness of the customer activity, be it purchases or visits

E.g. Time since last order or last engaged with the product

FREQUENCY

The frequency of the customer transactions or visits

E.g. Total number of transactions or average time between transactions/ engaged visits

MONETARY

The intention of customer to spend or purchasing power of customer

E.g. Total or average transactions value

Figure 7.1.1 RFM Analysis

- **Recency** is about when was the last order of a customer. It means the number of days since a customer made the last purchase. If it's a case for a website or an app, this could be interpreted as the last visit day or the last login time.

- **Frequency** is about the number of purchases in a given period. It could be 3 months, 6 months or 1 year. So we can understand this value as for how often or how many a customer used the product of a company. The bigger the value is, the more engaged the customers are. Could we say them as our VIP? Not necessary.

Cause we also have to think about how much they actually paid for each purchase, which means monetary value.

- **Monetary** is the total amount of money a customer spent in that given period. Therefore, big spenders will be differentiated with other customers such as MVP or VIP.

RFM factors illustrate these facts:

- the more recent the purchase, the more responsive the customer is to promotions
- the more frequently the customer buys, the more engaged and satisfied they are
- monetary value differentiates heavy spenders from low-value purchasers

The RFM values can be grouped in several ways:

1.Percentiles e.g. quantiles

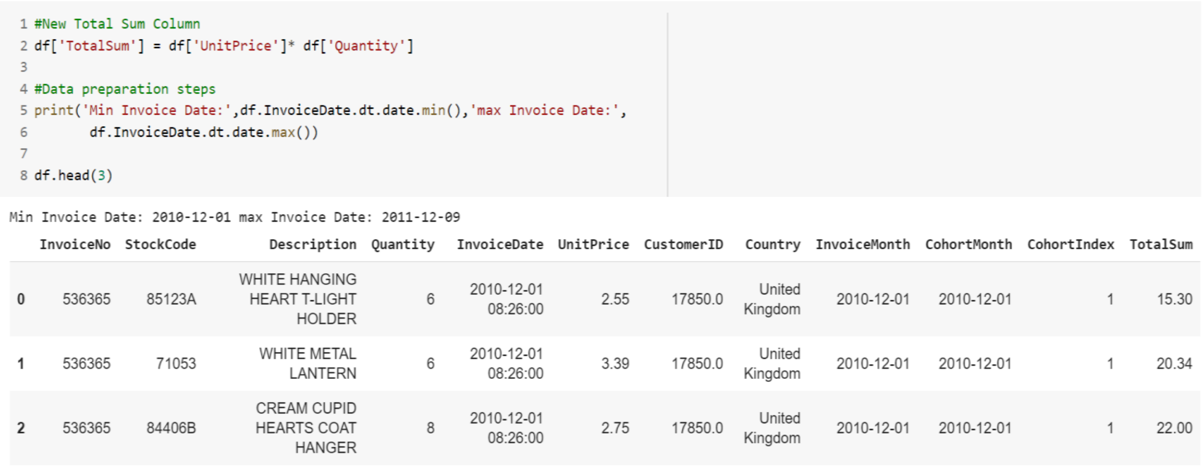
2.Pareto 80/20 cut

3.Custom - based on business knowledge

We are going to implement percentile-based grouping.

Process of calculating percentiles:

1. Sort customers based on that metric
2. Break customers into a pre-defined number of groups of equal size
3. Assign a label to each group



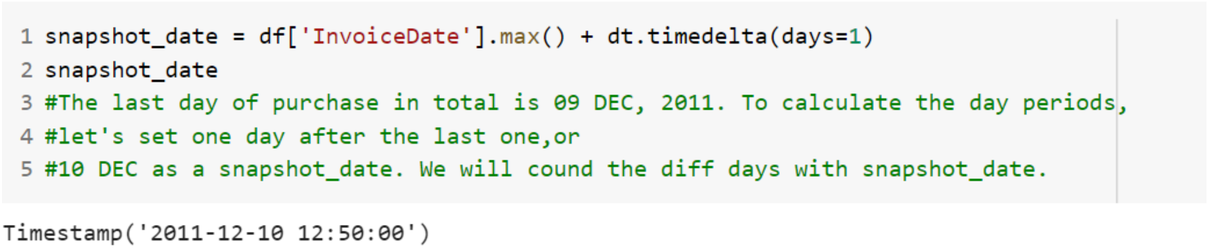
```
#New Total Sum Column
df['TotalSum'] = df['UnitPrice']* df['Quantity']

#Data preparation steps
print('Min Invoice Date:',df.InvoiceDate.dt.date.min(),'max Invoice Date:',
       df.InvoiceDate.dt.date.max())

df.head(3)
```

```
Min Invoice Date: 2010-12-01 max Invoice Date: 2011-12-09
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	InvoiceMonth	CohortMonth	CohortIndex	TotalSum
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010-12-01	2010-12-01	1	15.30
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010-12-01	2010-12-01	1	20.34
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010-12-01	2010-12-01	1	22.00

Figure 7.4.2 Adding TotalSum column

In the real world, we would be working with the most recent snapshot of the data of today or yesterday.

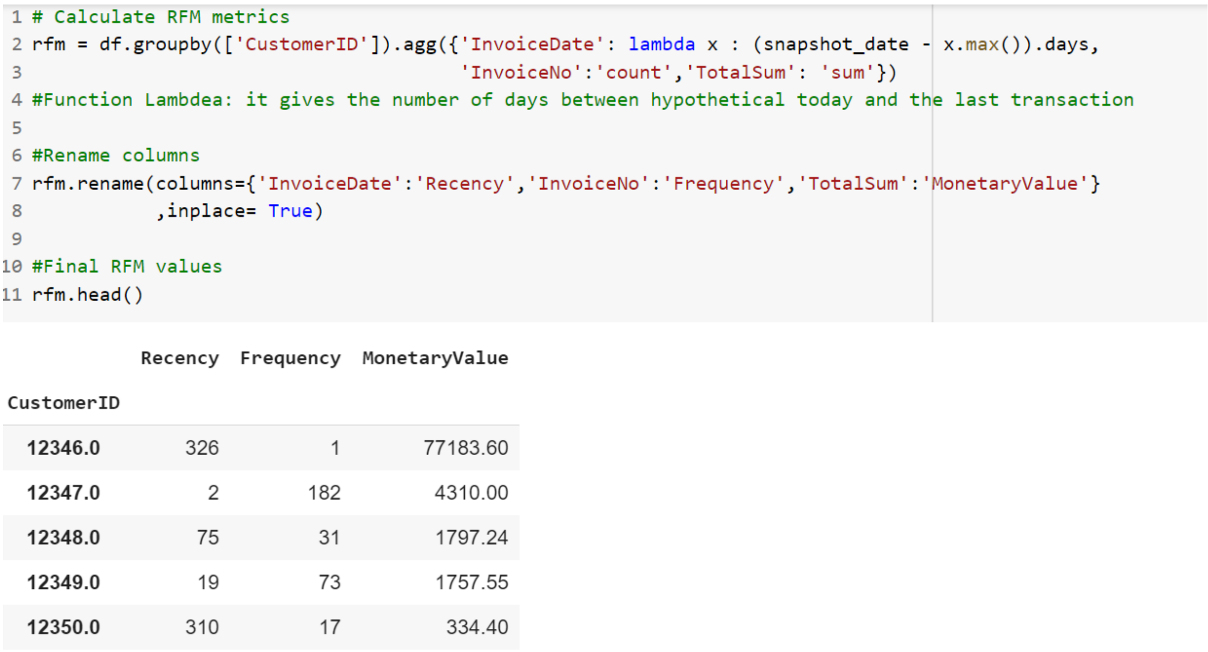


```
snapshot_date = df['InvoiceDate'].max() + dt.timedelta(days=1)
snapshot_date
#The last day of purchase in total is 09 DEC, 2011. To calculate the day periods,
#let's set one day after the last one,or
#10 DEC as a snapshot_date. We will cound the diff days with snapshot_date.
```

```
Timestamp('2011-12-10 12:50:00')
```

Figure 7.1.3 Recent snapshot

Calculation of RFM metrics:



```
# Calculate RFM metrics
rfm = df.groupby(['CustomerID']).agg({'InvoiceDate': lambda x : (snapshot_date - x.max()).days,
                                      'InvoiceNo':'count','TotalSum': 'sum'})
#Function Lambdea: it gives the number of days between hypothetical today and the last transaction

#Rename columns
rfm.rename(columns={'InvoiceDate':'Recency','InvoiceNo':'Frequency','TotalSum':'MonetaryValue'}
           ,inplace= True)

#Final RFM values
rfm.head()
```

	Recency	Frequency	MonetaryValue
CustomerID			
12346.0	326	1	77183.60
12347.0	2	182	4310.00
12348.0	75	31	1797.24
12349.0	19	73	1757.55
12350.0	310	17	334.40

Figure 7.1.4 Calculation of RFM metrics

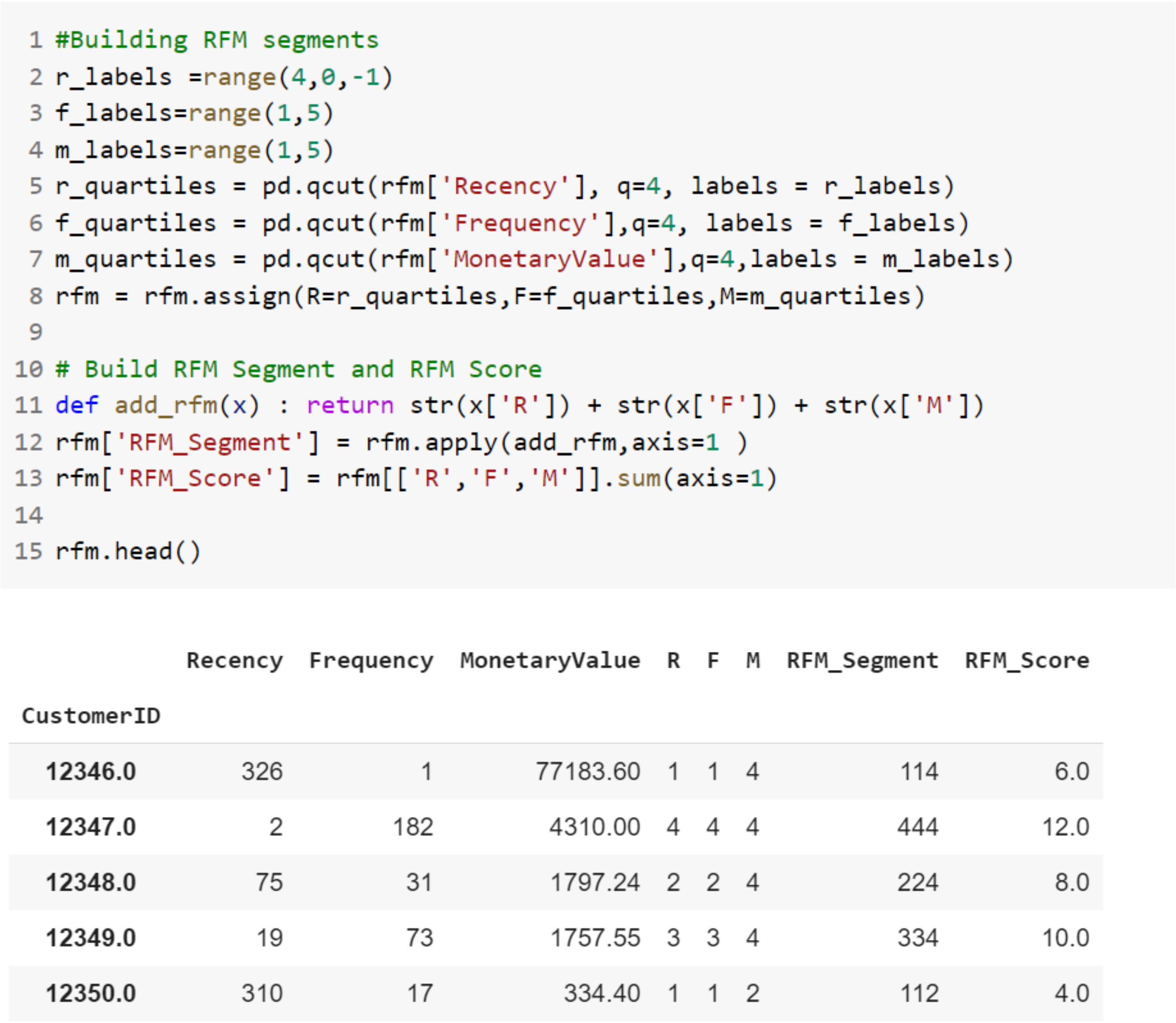
Note That:

We will rate "Recency" customer who have been active more recently better than the less recent customer, because each company wants its customers to be recent

We will rate "Frequency" and "Monetary Value" higher label because we want Customer to spend more money and visit more often(that is different order than recency).

Customer segments with RFM Model:

The simplest way to create customers segments from RFM Model is to use Quantiles. We assign a score from 1 to 4 to Recency, Frequency and Monetary. Four is the best/highest value, and one is the lowest/worst value. A final RFM score is calculated simply by combining individual RFM score numbers.



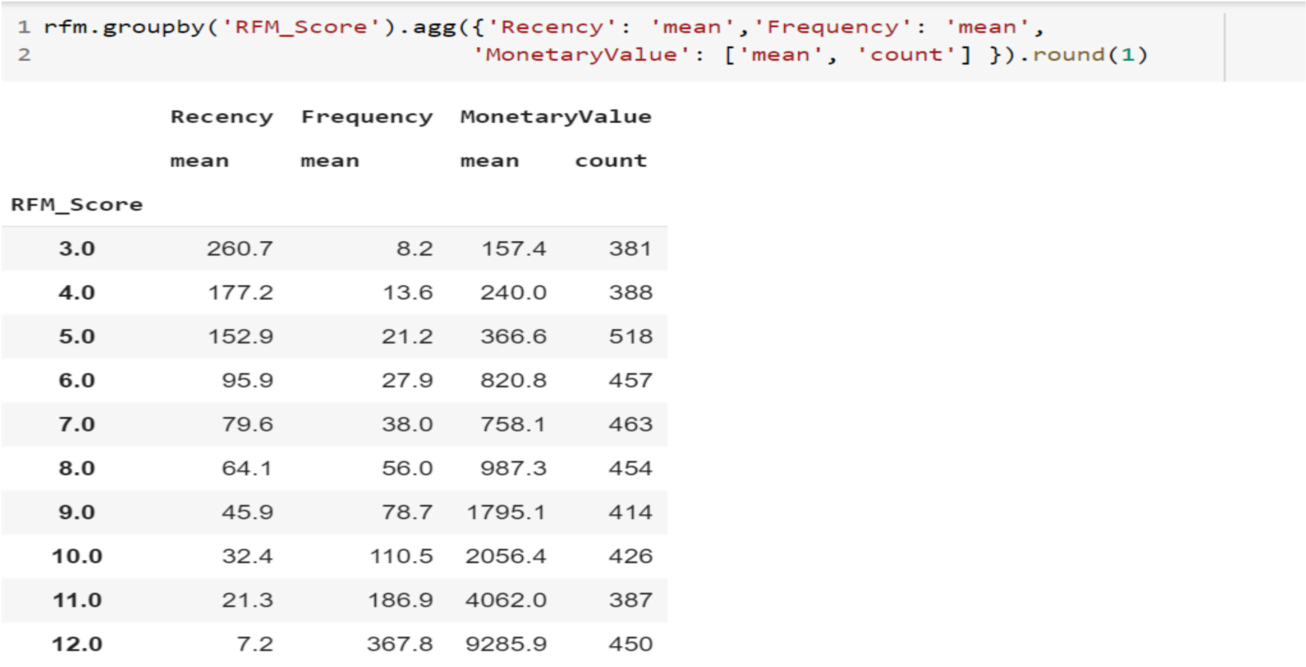
```python
#Building RFM segments
r_labels =range(4,0,-1)
f_labels=range(1,5)
m_labels=range(1,5)
r_quartiles = pd.qcut(rfm['Recency'], q=4, labels = r_labels)
f_quartiles = pd.qcut(rfm['Frequency'],q=4, labels = f_labels)
m_quartiles = pd.qcut(rfm['MonetaryValue'],q=4,labels = m_labels)
rfm = rfm.assign(R=r_quartiles,F=f_quartiles,M=m_quartiles)

# Build RFM Segment and RFM Score
def add_rfm(x) : return str(x['R']) + str(x['F']) + str(x['M'])
rfm['RFM_Segment'] = rfm.apply(add_rfm,axis=1 )
rfm['RFM_Score'] = rfm[['R','F','M']].sum(axis=1)

rfm.head()
```

```
            Recency  Frequency  MonetaryValue  R  F  M  RFM_Segment  RFM_Score
CustomerID
   12346.0      326          1       77183.60  1  1  4          114        6.0
   12347.0        2        182        4310.00  4  4  4          444       12.0
   12348.0       75         31        1797.24  2  2  4          224        8.0
   12349.0       19         73        1757.55  3  3  4          334       10.0
   12350.0      310         17         334.40  1  1  2          112        4.0
```

Figure 7.1.5 Building of RFM segments

RFM score is a simply sum of the Recency, Frequency and Monetary Values and this sum is a result of integer values like 10, 9, 8 and etc. This score will indicate the value of RFM score that will allow us to make decisions on a business product or on our customers. This is very important metric due to future decision-making process concerning the users or customers.

Metrics for RFM score:

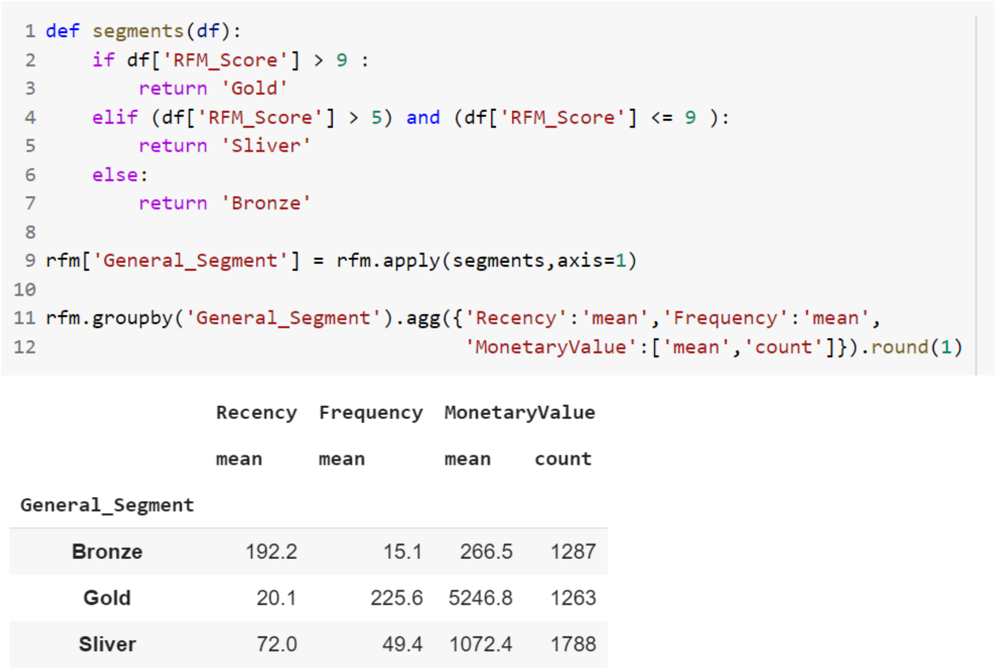


```
rfm.groupby('RFM_Score').agg({'Recency': 'mean','Frequency': 'mean',
                              'MonetaryValue': ['mean', 'count'] }).round(1)
```

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
RFM_Score				
3.0	260.7	8.2	157.4	381
4.0	177.2	13.6	240.0	388
5.0	152.9	21.2	366.6	518
6.0	95.9	27.9	820.8	457
7.0	79.6	38.0	758.1	463
8.0	64.1	56.0	987.3	454
9.0	45.9	78.7	1795.1	414
10.0	32.4	110.5	2056.4	426
11.0	21.3	186.9	4062.0	387
12.0	7.2	367.8	9285.9	450

Figure 7.1.5 Metrics for RFM score

Use RFM score to group customers into Gold, Silver and Bronze segments:



```
def segments(df):
    if df['RFM_Score'] > 9 :
        return 'Gold'
    elif (df['RFM_Score'] > 5) and (df['RFM_Score'] <= 9 ):
        return 'Sliver'
    else:
        return 'Bronze'

rfm['General_Segment'] = rfm.apply(segments,axis=1)

rfm.groupby('General_Segment').agg({'Recency':'mean','Frequency':'mean',
                                    'MonetaryValue':['mean','count']}).round(1)
```

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
General_Segment				
Bronze	192.2	15.1	266.5	1287
Gold	20.1	225.6	5246.8	1263
Sliver	72.0	49.4	1072.4	1788

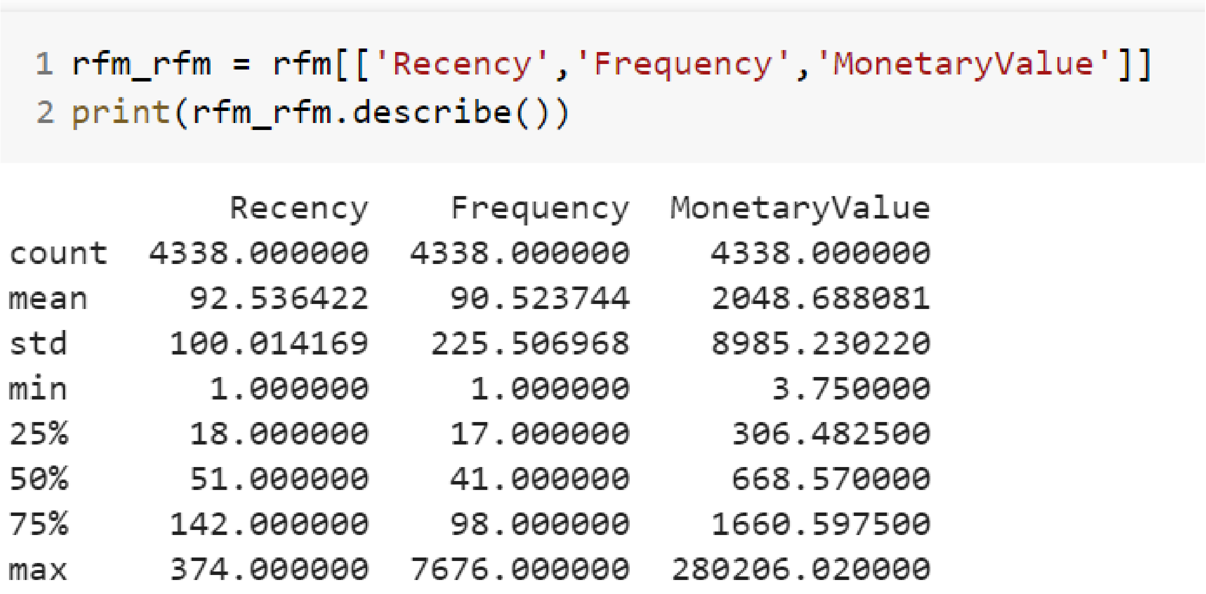
Figure 7.1.6 Grouping the customers

CHAPTER-8

Data preprocessing for k-Means

There are some critical assumptions with K-Means algorithm before building it:

1. All variable must have symmetrical distribution and should not be skewed.
2. All variables should have the same or almost the same average values.
3. All variables should have the same level of variance.



```
rfm_rfm = rfm[['Recency','Frequency','MonetaryValue']]
print(rfm_rfm.describe())
```

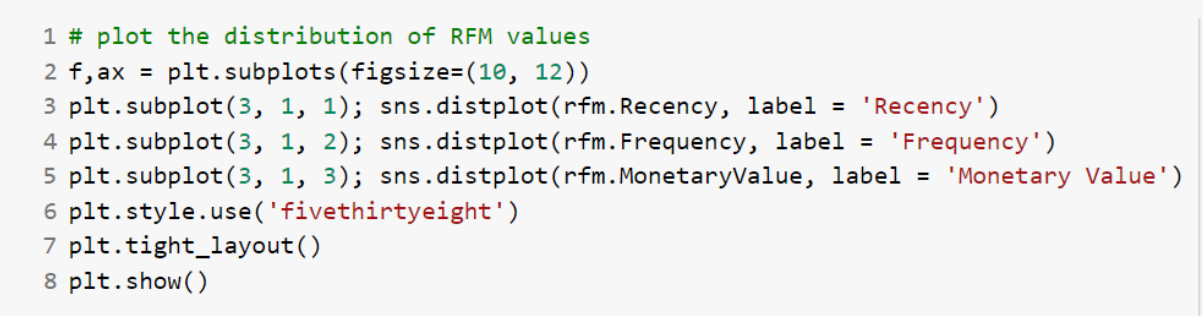
```
          Recency    Frequency  MonetaryValue
count  4338.000000  4338.000000    4338.000000
mean     92.536422    90.523744    2048.688081
std     100.014169   225.506968    8985.230220
min       1.000000     1.000000       3.750000
25%      18.000000    17.000000     306.482500
50%      51.000000    41.000000     668.570000
75%     142.000000    98.000000    1660.597500
max     374.000000  7676.000000  280206.020000
```

Figure 8 4.1 Check for mean and variance

From this table, we find this Problem: Mean and Variance are not Equal

Solution: Scaling variables by using a scaler from scikit-learn library

We will now cluster our customers with K-Means! So let's visualize the data distribution in RFM table.



```python
# plot the distribution of RFM values
f,ax = plt.subplots(figsize=(10, 12))
plt.subplot(3, 1, 1); sns.distplot(rfm.Recency, label = 'Recency')
plt.subplot(3, 1, 2); sns.distplot(rfm.Frequency, label = 'Frequency')
plt.subplot(3, 1, 3); sns.distplot(rfm.MonetaryValue, label = 'Monetary Value')
plt.style.use('fivethirtyeight')
plt.tight_layout()
plt.show()
```

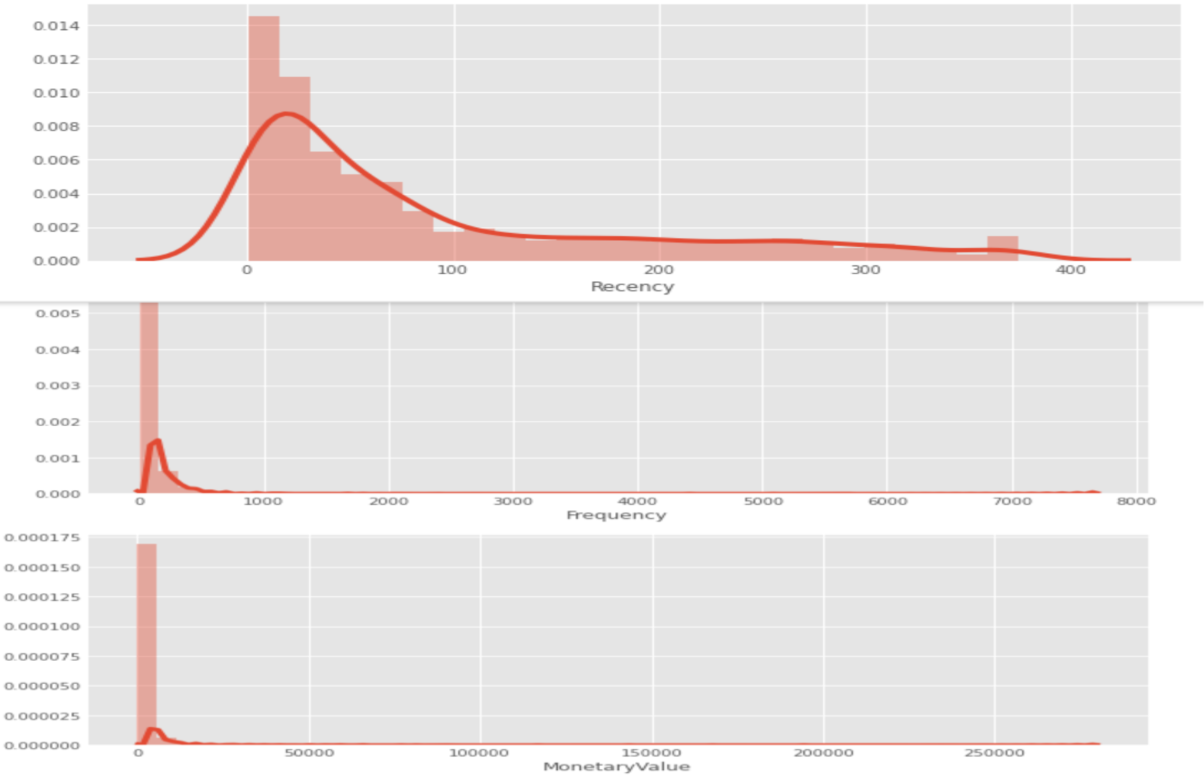


Figure 8.1.5 Plot of distribution of RFM variables

Also, there is another Problem: UnSymmetric distribution of variables (data skewed)

Solution: Logarithmic transformation (positive values only) will manage skewness

We use these Sequence of structuring pre-processing steps:

1. Unskew the data - log transformation

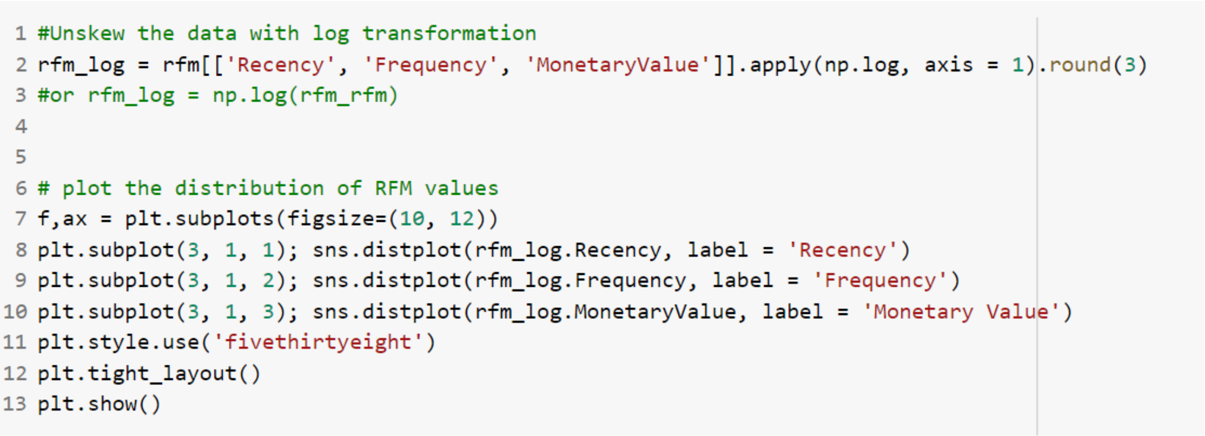
2. Standardize to the same average values

3. Scale to the same standard deviation

4. Store as a separate array to be used for clustering

Why the sequence matters?

- Log transformation only works with positive data
- Normalization forces data to have negative values and log will not work



```
#Unskew the data with log transformation
rfm_log = rfm[['Recency', 'Frequency', 'MonetaryValue']].apply(np.log, axis = 1).round(3)
#or rfm_log = np.log(rfm_rfm)


# plot the distribution of RFM values
f,ax = plt.subplots(figsize=(10, 12))
plt.subplot(3, 1, 1); sns.distplot(rfm_log.Recency, label = 'Recency')
plt.subplot(3, 1, 2); sns.distplot(rfm_log.Frequency, label = 'Frequency')
plt.subplot(3, 1, 3); sns.distplot(rfm_log.MonetaryValue, label = 'Monetary Value')
plt.style.use('fivethirtyeight')
plt.tight_layout()
plt.show()
```

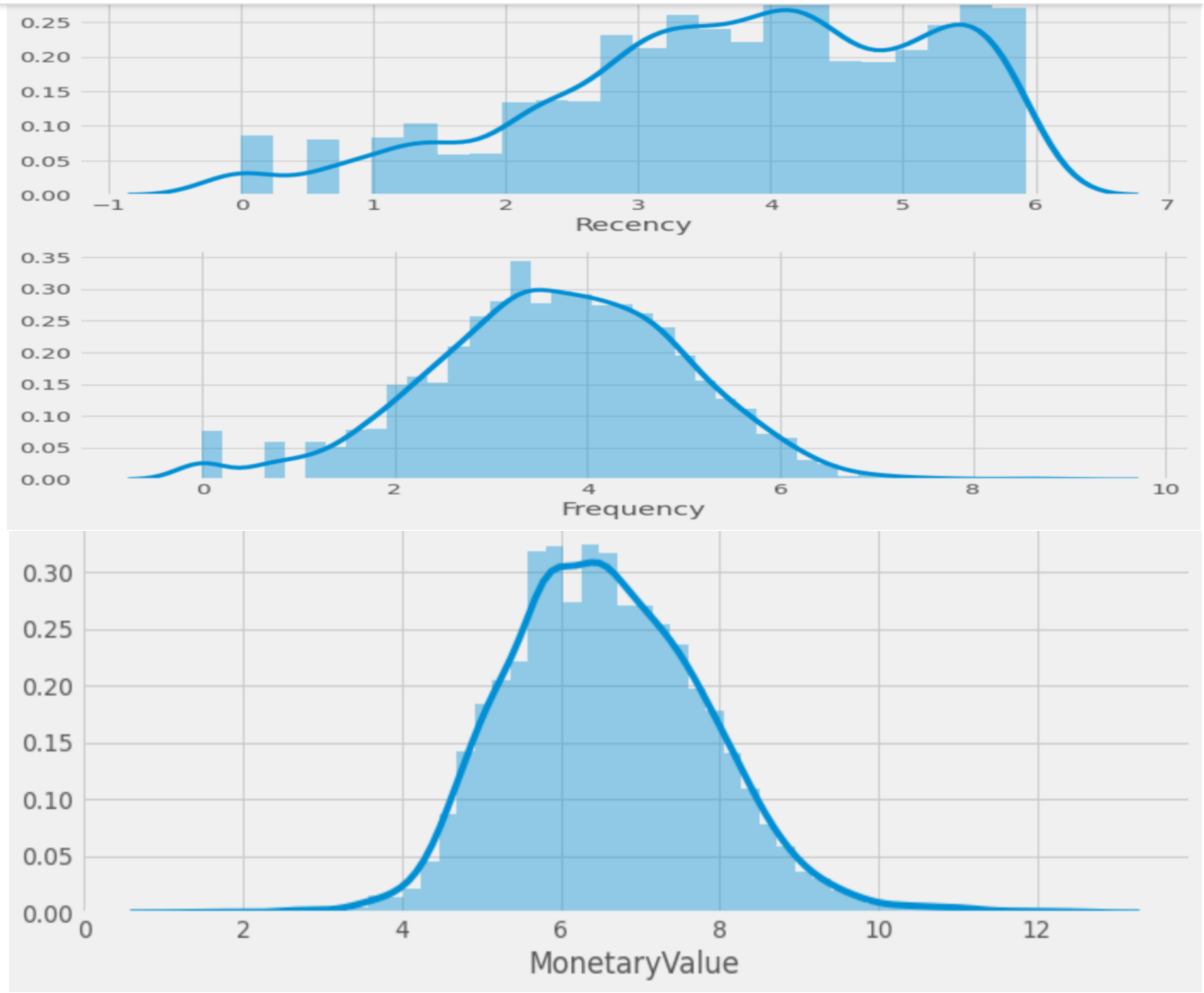


Figure 8.1.6 Plot of distribution RFM variables after log transformation

CHAPTER-9
K-Means clustering

In data science, we often think about how to use data to make predictions on new data points. This is called “supervised learning.” Sometimes, however, rather than ‘making predictions’, we instead want to categorize data into buckets. This is termed “unsupervised learning.”

To illustrate the difference, let’s say we’re at a major pizza chain and we’ve been tasked with creating a feature in the order management software that will predict delivery times for customers. In order to achieve this, we are given a dataset that has delivery times, distances travelled, day of week, time of day, staff on hand, and volume of sales for several deliveries in the past. From this data, we can make predictions on future delivery times. This is a good example of supervised learning.

Now, let’s say the pizza chain wants to send out targeted coupons to customers. It wants to segment its customers into 4 groups: large families, small families, singles, and college students. We are given prior ordering data (e.g. size of order, price, frequency, etc) and we’re tasked with putting each customer into one of the four buckets. This would be an example of “unsupervised learning” since we’re not making predictions; we’re merely categorizing the customers into groups.

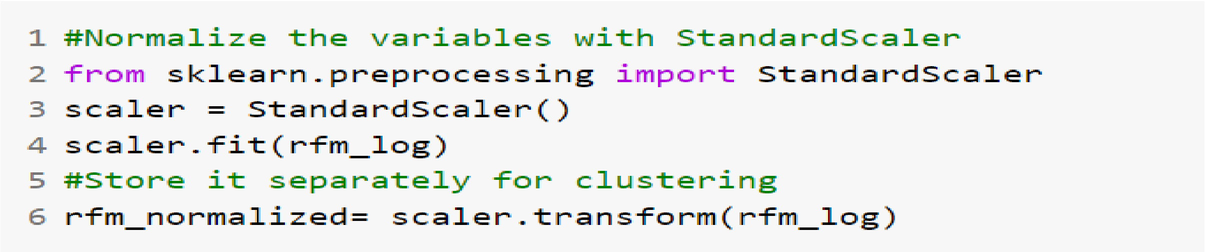
Clustering is one of the most frequently utilized forms of unsupervised learning. In this article, we’ll explore two of the most common forms of clustering: k-means and hierarchical.

9.1 Implementation the K-Means Clustering Algorithm

Key steps

1. Data pre-processing
2. Choosing a number of clusters
3. Running k-means clustering on pre-processed data
4. Analyzing average RFM values of each cluster.

9.1.1 Data processing



```
#Normalize the variables with StandardScaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(rfm_log)
#Store it separately for clustering
rfm_normalized= scaler.transform(rfm_log)
```

Figure 9.3.1 Standard scaling

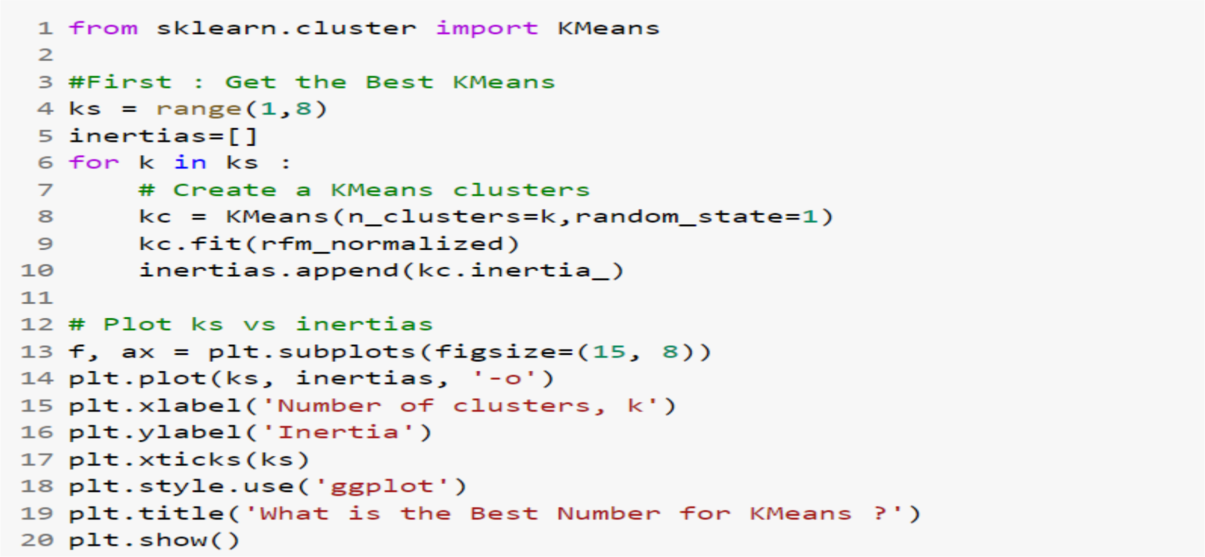
9.2 Choosing no. of clusters

Methods to define the number of clusters

- Visual methods - elbow criterion
- Mathematical methods - silhouette coefficient
- Experimentation and interpretation

Elbow criterion method

- Plot the number of clusters against within-cluster sum-of-squared-errors (SSE) - sum of squared distances from every data point to their cluster center
- Identify an "elbow" in the plot
- Elbow - a point representing an "optimal" number of clusters



```
from sklearn.cluster import KMeans

#First : Get the Best KMeans
ks = range(1,8)
inertias=[]
for k in ks :
    # Create a KMeans clusters
    kc = KMeans(n_clusters=k,random_state=1)
    kc.fit(rfm_normalized)
    inertias.append(kc.inertia_)

# Plot ks vs inertias
f, ax = plt.subplots(figsize=(15, 8))
plt.plot(ks, inertias, '-o')
plt.xlabel('Number of clusters, k')
plt.ylabel('Inertia')
plt.xticks(ks)
plt.style.use('ggplot')
plt.title('What is the Best Number for KMeans ?')
plt.show()
```

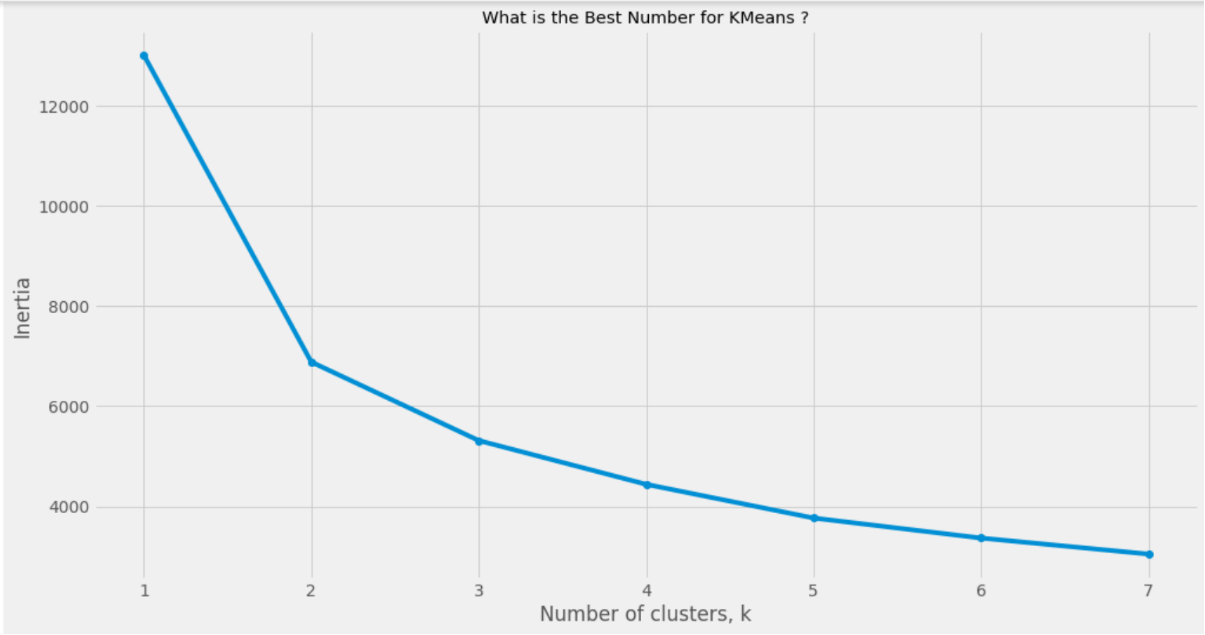
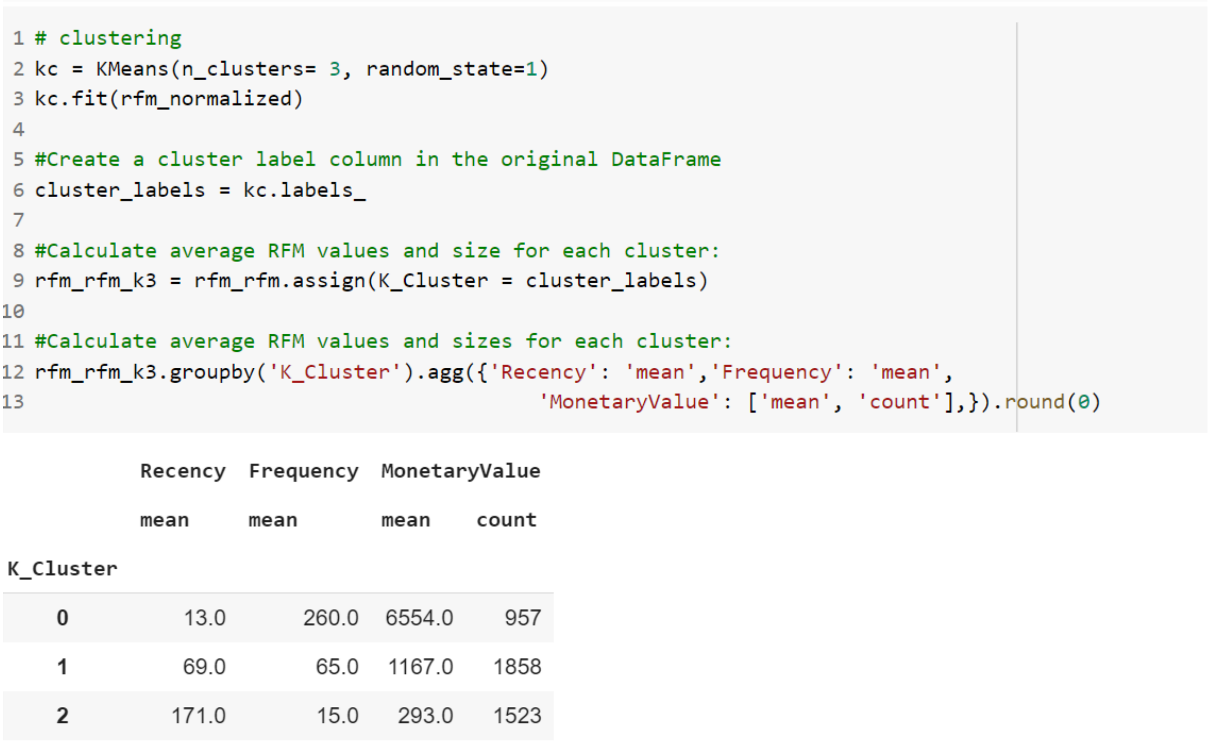


Figure 9.4.1 Elbow method

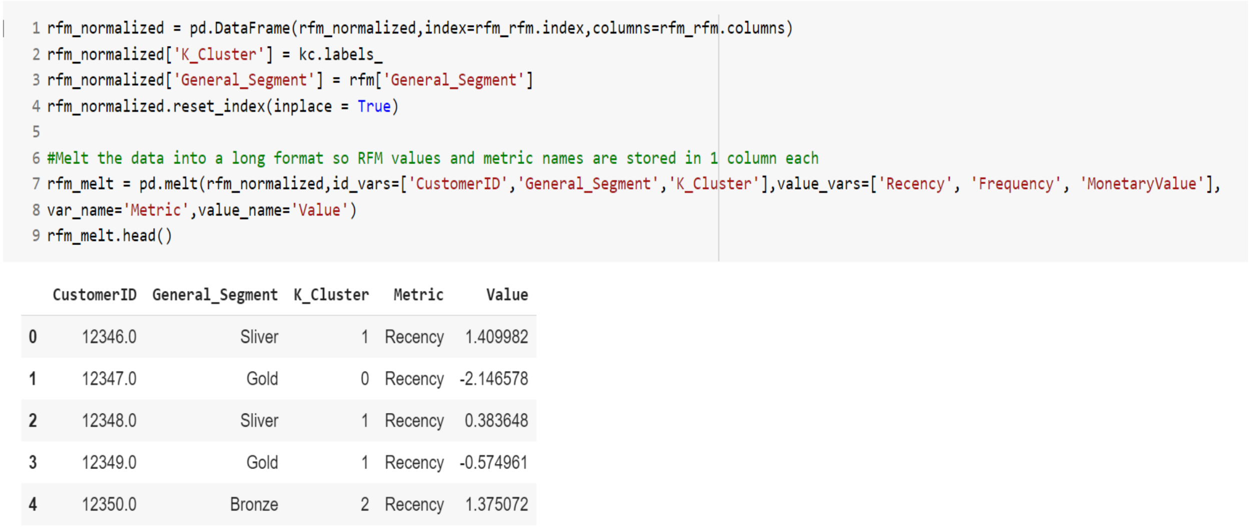
Here, we are using no.of clusters=3 because The scope of the Elbow Method is to choose the number of cluster where the graph breaks. Usually it is number 3 so 3 clusters must be in the algorithm.



```
# clustering
kc = KMeans(n_clusters= 3, random_state=1)
kc.fit(rfm_normalized)

#Create a cluster label column in the original DataFrame
cluster_labels = kc.labels_

#Calculate average RFM values and size for each cluster:
rfm_rfm_k3 = rfm_rfm.assign(K_Cluster = cluster_labels)

#Calculate average RFM values and sizes for each cluster:
rfm_rfm_k3.groupby('K_Cluster').agg({'Recency': 'mean','Frequency': 'mean',
                                          'MonetaryValue': ['mean', 'count'],}).round(0)
```

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
K_Cluster				
0	13.0	260.0	6554.0	957
1	69.0	65.0	1167.0	1858
2	171.0	15.0	293.0	1523

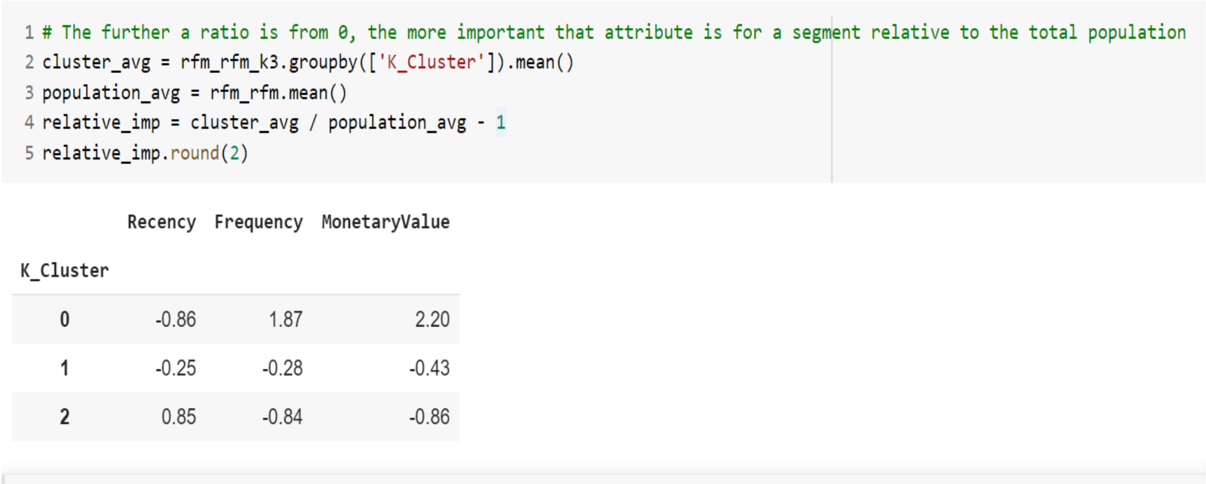
Figure 9.2.2 Avg RFM values for each cluster



```python
rfm_normalized = pd.DataFrame(rfm_normalized,index=rfm_rfm.index,columns=rfm_rfm.columns)
rfm_normalized['K_Cluster'] = kc.labels_
rfm_normalized['General_Segment'] = rfm['General_Segment']
rfm_normalized.reset_index(inplace = True)

#Melt the data into a long format so RFM values and metric names are stored in 1 column each
rfm_melt = pd.melt(rfm_normalized,id_vars=['CustomerID','General_Segment','K_Cluster'],value_vars=['Recency', 'Frequency', 'MonetaryValue'],
var_name='Metric',value_name='Value')
rfm_melt.head()
```

```
   CustomerID  General_Segment  K_Cluster   Metric      Value
0     12346.0           Sliver          1  Recency   1.409982
1     12347.0             Gold          0  Recency  -2.146578
2     12348.0           Sliver          1  Recency   0.383648
3     12349.0             Gold          1  Recency  -0.574961
4     12350.0           Bronze          2  Recency   1.375072
```

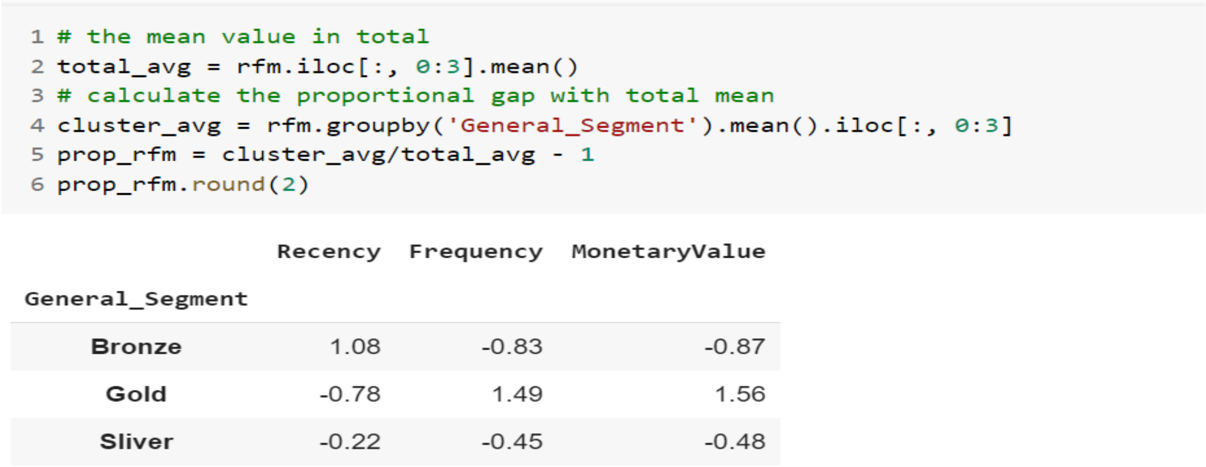
Figure 9.2.3 Normalizing the data



```python
# The further a ratio is from 0, the more important that attribute is for a segment relative to the total population
cluster_avg = rfm_rfm_k3.groupby(['K_Cluster']).mean()
population_avg = rfm_rfm.mean()
relative_imp = cluster_avg / population_avg - 1
relative_imp.round(2)
```

```
           Recency  Frequency  MonetaryValue
K_Cluster
0            -0.86       1.87           2.20
1            -0.25      -0.28          -0.43
2             0.85      -0.84          -0.86
```

Figure 9.2.4 Segment relative to total population

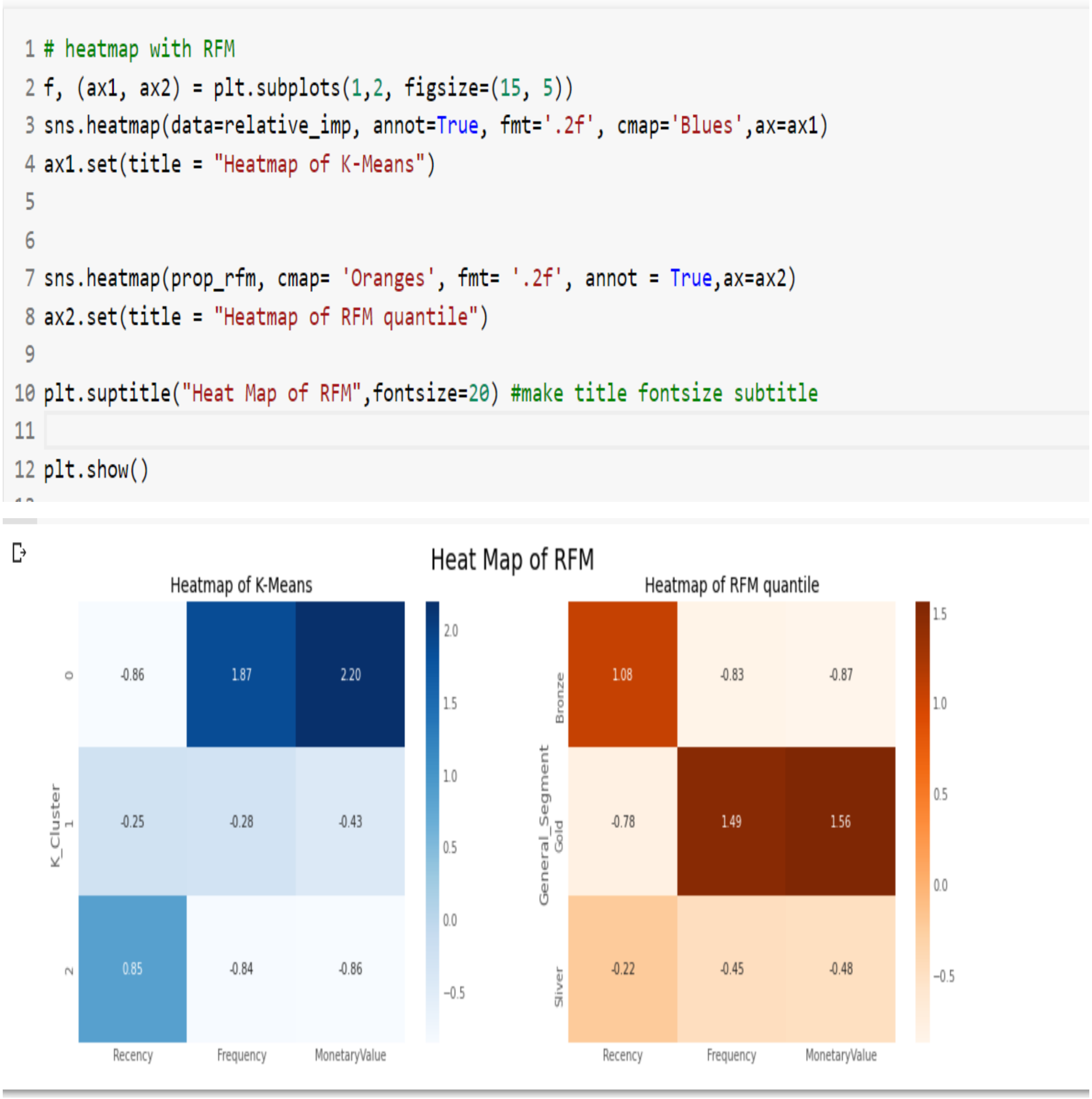


```python
# the mean value in total
total_avg = rfm.iloc[:, 0:3].mean()
# calculate the proportional gap with total mean
cluster_avg = rfm.groupby('General_Segment').mean().iloc[:, 0:3]
prop_rfm = cluster_avg/total_avg - 1
prop_rfm.round(2)
```

```
                 Recency  Frequency  MonetaryValue
General_Segment
Bronze              1.08      -0.83          -0.87
Gold               -0.78       1.49           1.56
Sliver             -0.22      -0.45          -0.48
```

Figure 9.2.5 Proportional gap with total mean

Heat maps are a graphical representation of data where larger values were colored in darker scales and smaller values in lighter scales. We can compare the variance between the groups quite intuitively by colors.

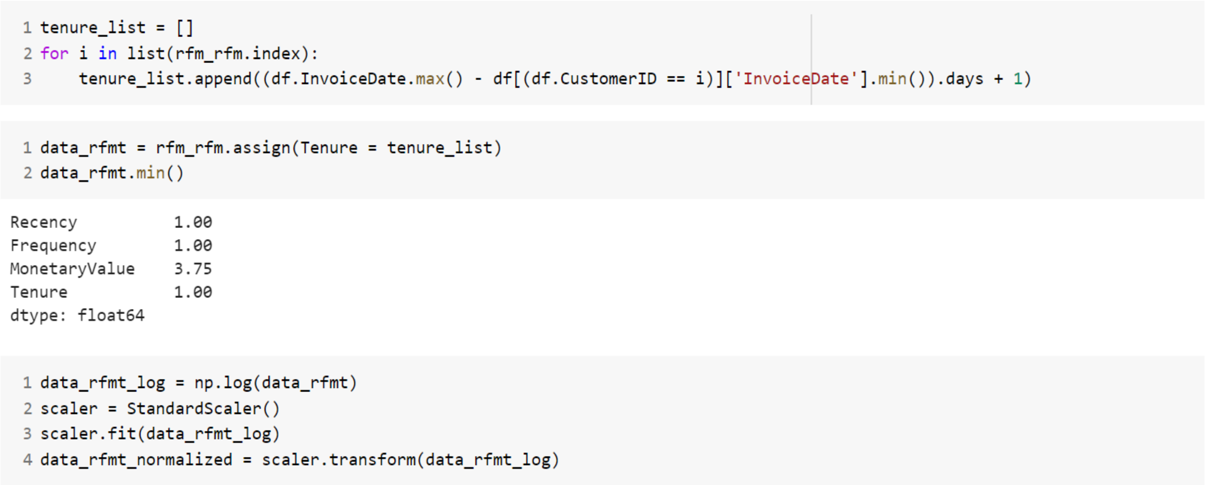


```
# heatmap with RFM
f, (ax1, ax2) = plt.subplots(1,2, figsize=(15, 5))
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='Blues',ax=ax1)
ax1.set(title = "Heatmap of K-Means")


sns.heatmap(prop_rfm, cmap= 'Oranges', fmt= '.2f', annot = True,ax=ax2)
ax2.set(title = "Heatmap of RFM quantile")

plt.suptitle("Heat Map of RFM",fontsize=20) #make title fontsize subtitle

plt.show()
```

Figure 9.2.6 Heatmap for K-Means and RFM quantile

CHAPTER-10

Tenure



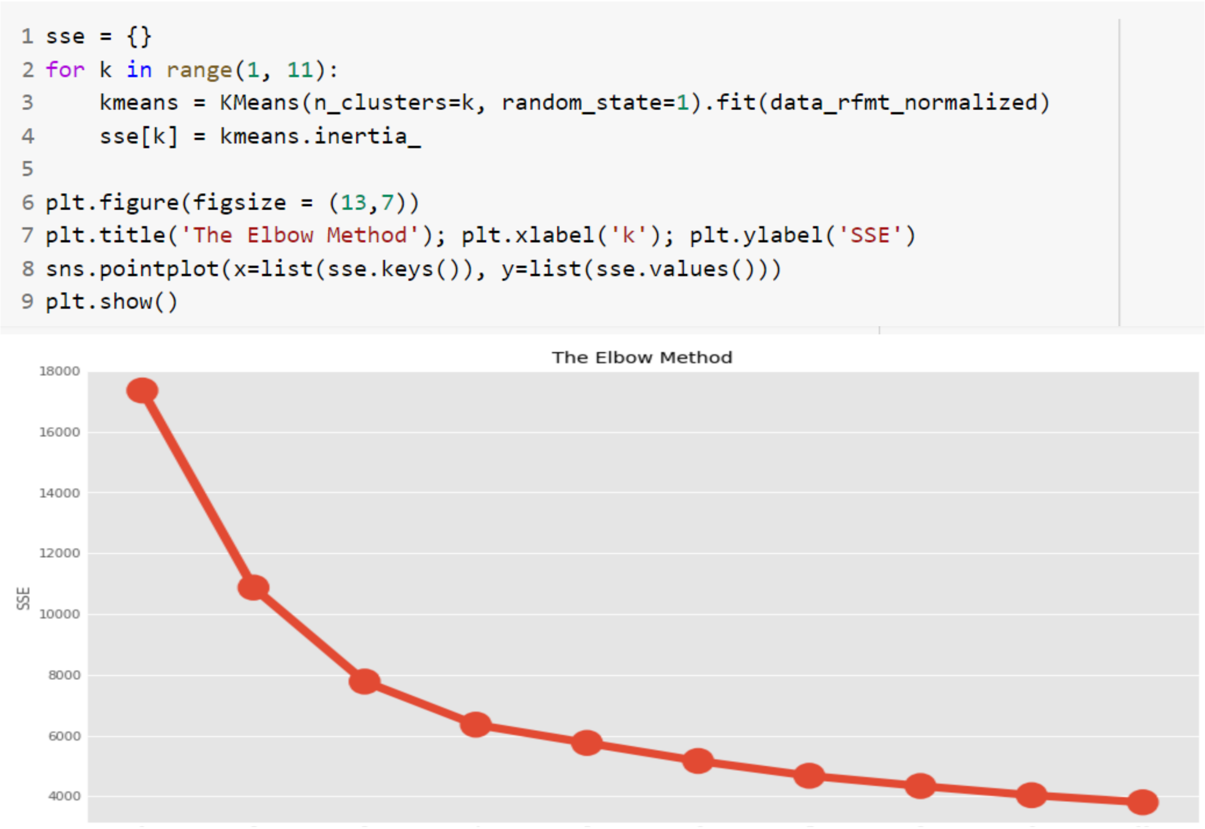
```
tenure_list = []
for i in list(rfm_rfm.index):
    tenure_list.append((df.InvoiceDate.max() - df[(df.CustomerID == i)]['InvoiceDate'].min()).days + 1)
```

```
data_rfmt = rfm_rfm.assign(Tenure = tenure_list)
data_rfmt.min()
```

```
Recency          1.00
Frequency        1.00
MonetaryValue    3.75
Tenure           1.00
dtype: float64
```

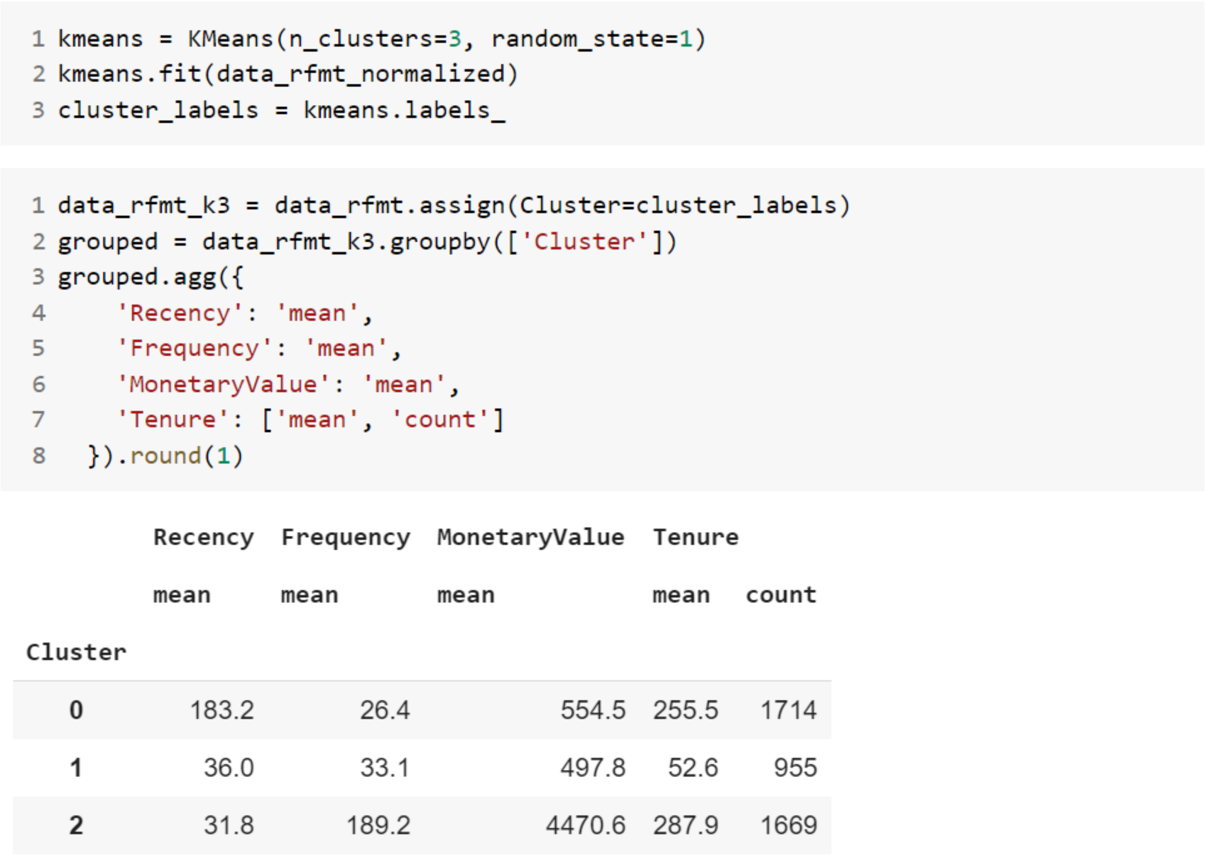
```
data_rfmt_log = np.log(data_rfmt)
scaler = StandardScaler()
scaler.fit(data_rfmt_log)
data_rfmt_normalized = scaler.transform(data_rfmt_log)
```

Figure 10.3.1 Adding tenure



```
sse = {}
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=1).fit(data_rfmt_normalized)
    sse[k] = kmeans.inertia_

plt.figure(figsize = (13,7))
plt.title('The Elbow Method'); plt.xlabel('k'); plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()
```

Figure 10.1.4 Elbow method for tenure



```python
kmeans = KMeans(n_clusters=3, random_state=1)
kmeans.fit(data_rfmt_normalized)
cluster_labels = kmeans.labels_
```

```python
data_rfmt_k3 = data_rfmt.assign(Cluster=cluster_labels)
grouped = data_rfmt_k3.groupby(['Cluster'])
grouped.agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'MonetaryValue': 'mean',
    'Tenure': ['mean', 'count']
  }).round(1)
```

```
         Recency  Frequency  MonetaryValue  Tenure
         mean     mean       mean           mean    count
Cluster
   0      183.2      26.4          554.5    255.5    1714
   1       36.0      33.1          497.8     52.6     955
   2       31.8     189.2         4470.6    287.9    1669
```

Figure 10.1.3 RFM and Tenure values for each cluster

Conclusion

We talked about how to get RFM values from customer purchase data, and we made two kinds of segmentation with RFM quantiles and K-Means clustering methods.

By performing Cohort analysis and RFM you can get following answers to following questions:

- How much effective was a marketing campaign held in a particular time period.
- Did the strategy employed to improve the conversion rates of Customers worked?
- Should I focus more on retention rather than acquiring new customers?
- Are my customer nurturing strategies effective?
- Which marketing channels bring me the best results?
- Is there a seasonality pattern in Customer behaviour?
- along with various performance measures/metrics for your organization.

References

1. https://towardsdatascience.com/an-introduction-to-clustering-algorithms-in-python-123438574097
2. https://clevertap.com/blog/rfm-analysis/
3. https://medium.com/analytics-for-humans/a-beginners-guide-to-cohort-analysis-the-most-actionable-and-underrated-report-on-google-c0797d826bf4
4. https://www.tutorialspoint.com/python/index.htm
5. https://medium.com/code-heroku/introduction-to-exploratory-data-analysis-eda-c025
6. https://en.wikipedia.org/wiki/Machine_learning
7. https://www.edureka.co/blog/data-science-applications/

Github:

https://github.com/adavenibhavana/Customer-Segmentation-Based-on-Purchase-Behaviour/tree/master

Google Site:

https://sites.google.com/view/customer-segmentation-based-on/home

Dataset link:

https://drive.google.com/drive/u/0/folders/1ME8bcQk8_vMNE7BVtN1om0W-8NSujkDa