

# PROJECT REPORT

## "DATA ANALYSIS"

### TRENDY GO



MADE BY: OMEMA DANIYAL

## TABLE OF CONTENTS

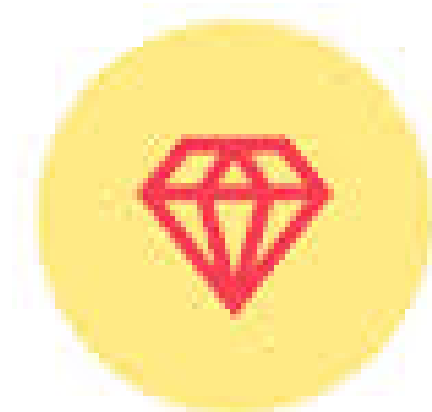
➤ Abstract	➤ Uses of data analysis in our Business
➤ Data analysis Overview	➤ Benefits of data analysis
➤ The Scope of data analysis	➤ Methods and techniques
➤ Python for data analysis	➤ Data Visualization and Analysis
➤ Data collection	➤ Conclusion

**SAMPLE IMAGE OF OUR SETUP**

**TRENDYGO**



**SAMPLE LOGO OF OUR BRAND**



**TRENDYGO**

# ABSTRACT

As each and every sector of the market is growing, data is building up day by day, we need to keep the record of the data which can be helpful for the analytics and evaluation. Now we don't have data in gigabyte or terabyte but in zeta byte and petabyte and this data cannot be handled with the day to day software such as Excel or Mat lab. Therefore in this report we will be dealing with large data sets with the high-level programming language 'Python'.

The main goal of this project is to aggregate and analyze the data collected from the different data sources available on the internet. This project mainly focuses on the usage of the python programming language. This language has not only it's application in the field of just analyzing the data but also for the prediction of the upcoming scenarios in the BUSINESS OF FASHION,

The purpose of using this specific language is due to its versatility, vast libraries (Pandas, Numpy, Matplotlib, etc.), speed limitations, and ease of learning. We will be analyzing large energy data sets in this project which cannot be easily analyzed in other tools as compared to python. Python does not have it's limitation to only data analytics but also in many other fields such as Artificial intelligence, Machine learning, and many more

# DATA ANALYSIS OVERVIEW

Data analysis is the field of data analytics and data visualization in which raw data or the un-structured data is cleaned and made ready for the analysis purpose. Data scientists use this data to get the required information for the future purpose.

## STAGES OF DATA ANALYSIS

- Identifying the problem.
- Identify available data sources
- Identify available data sources
- Identify if additional data sources are needed.
- Statistical analysis
- Implementation, development
- Communicate results
- Maintenance



# SCOPE OF DATA ANALYSIS

Data analytics is the Art and Science of drawing actionable insights from the data.

Data Analytics + Business Knowledge = Impact/Value Creation for the Business.

It involves following steps:

1. Understand business problem that we are trying to solve
2. Obtain and prepare the data to solve the business problem
3. Do data mining, statistical analysis, machine learning etc to build models and dashboards
4. Draw insights and create decks/visualizations
5. Share and convince stakeholder.

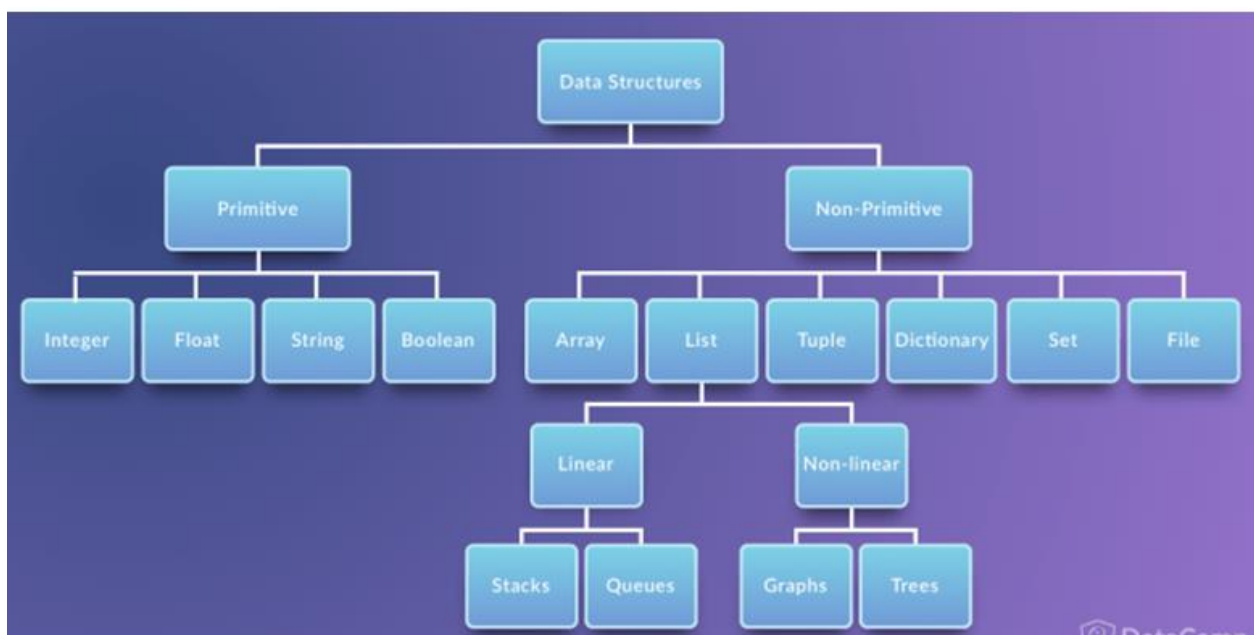
# PYTHON FOR DATA ANALYSIS

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics”. This language consist of mainly data structures which make it very easy for the data scientists to analyses the data very effectively. It does not only help in forecasting and analysis it also helps in connecting the two different languages.

## DATA STRUCTURES IN PYTHON

Data structures are the way of storing the data so that we can easily perform different operations on the data whenever its required. When the data has been collected from the data source the data is available in different forms. So later it is easy for the data scientists to perform different operation on the data once it is sorted in to different data structures.

Data structures are mainly classified in to two categories and then further their subcategories shown below.



- Operators in python

- Condition statements in python
- Loops in python
- Module, Package and Functions in python

## **PRIMITIVE DATA STRUCTURES.**

They are also called as basic data structures. This type of data structures contains simple values of the data

---

**Integers - All the whole numbers from negative infinity to positive infinity comes under integer data types. for example 4,9,-2,-6.**

- **Float - The decimal figure numbers or rational numbers comes under float data types. for example 3.1,2.2,8.96**
- **Strings - Collection of alphabets or characters are called strings. We enclose the string either in single or double quotes in python. for example 'hello' and "bread".**
- **Boolean- These are the built in data types which take two values that are 'True' and 'False'. True represents the 1 and False represents 0 in python.**

---

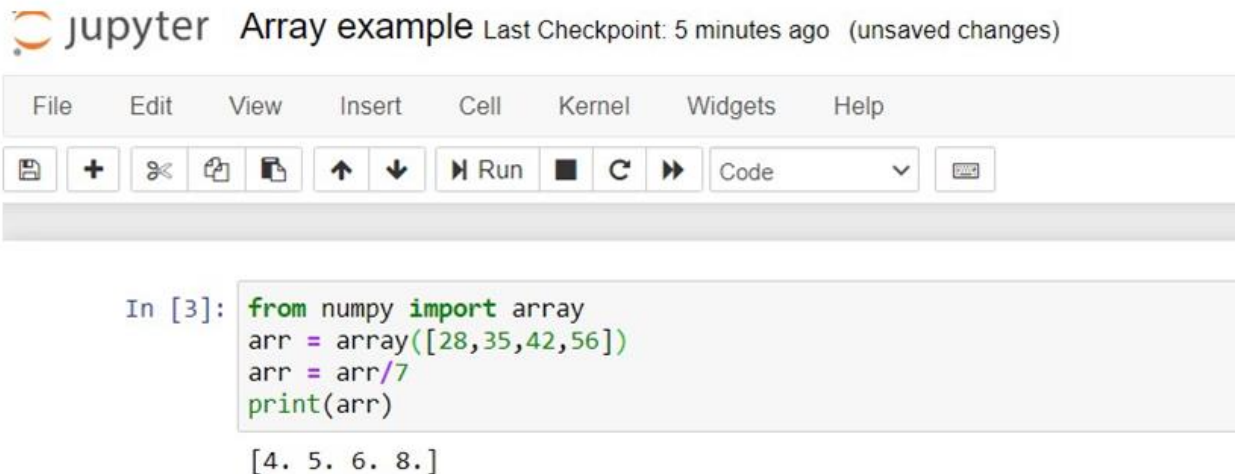
## **NON-PRIMITIVE DATA STRUCTURES**

These are the derived type or reference variables data structures. They are called derived data structures because they derived from the basic data structures such



as integer and float. Python has mainly five types of data structures.

**ARRAY** - Array is the collection of data types of same type. Arrays data structures are used mostly in the NumPy library of python. In the below example we have first imported the pack- age array from numpy library and defined the array as variable are then divided the array by 7 and we have printed our array to get output.



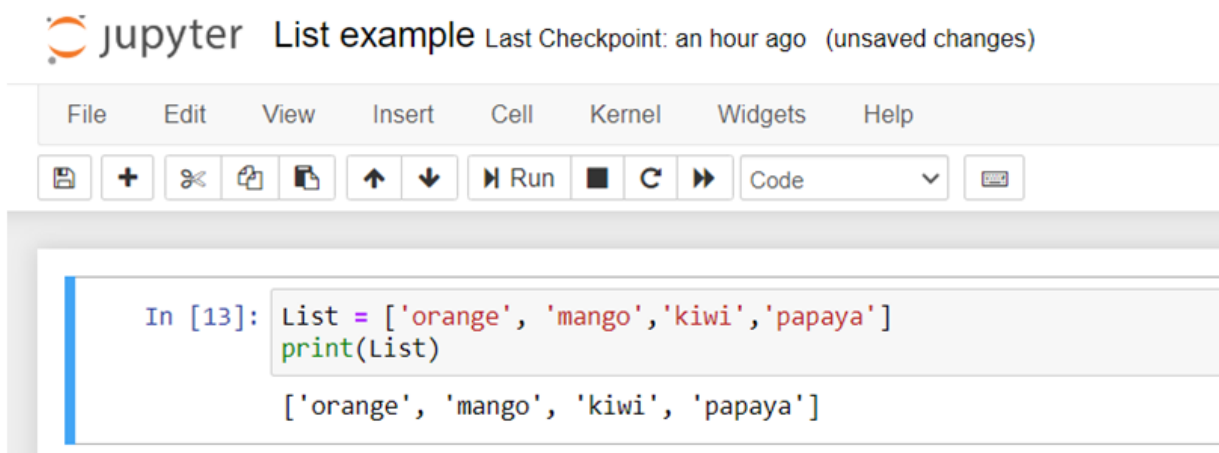
The image shows a Jupyter Notebook interface with the title "Array example" and a subtitle "Last Checkpoint: 5 minutes ago (unsaved changes)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The code cell contains the following Python code:

```
In [3]: from numpy import array
arr = array([28,35,42,56])
arr = arr/7
print(arr)
```

The output of the code is displayed below the code cell:

```
[4. 5. 6. 8.]
```

**LIST** list is a value that contains multiple values in an ordered sequence”. Values in the list referred to list itself that is the value can be stored in a variable or passed to a function. Lists are changeable and values in the list are enclosed inside a square bracket, we can perform multiple operations such as indexing, slicing, adding and multiplying



The image shows a Jupyter Notebook interface with the title "List example" and a subtitle "Last Checkpoint: an hour ago (unsaved changes)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The code cell contains the following Python code:

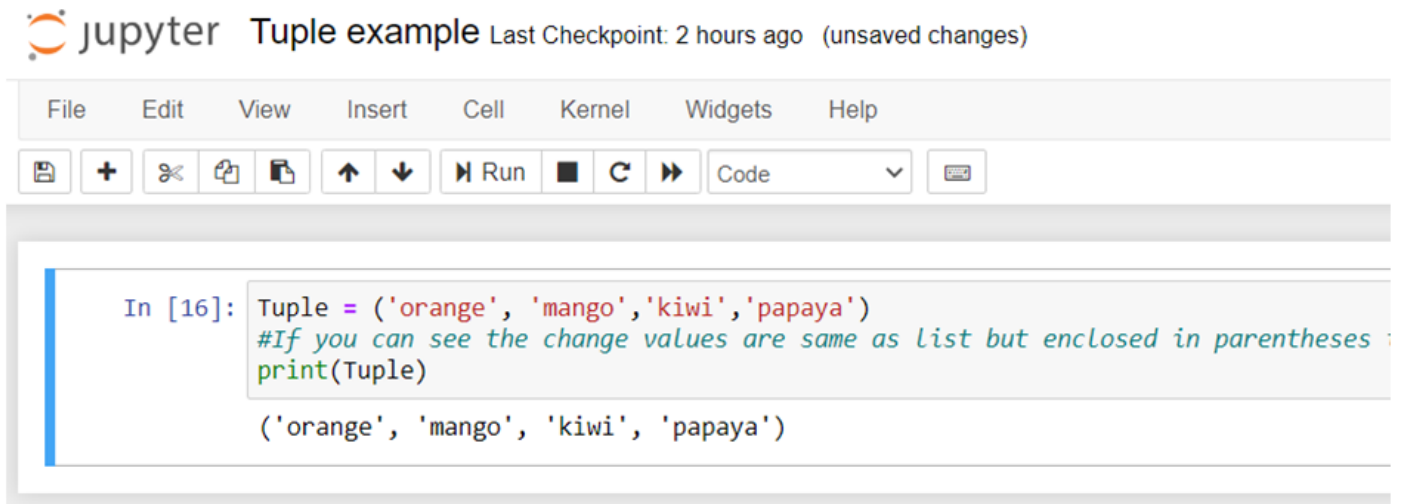
```
In [13]: List = ['orange', 'mango', 'kiwi', 'papaya']
print(List)
```

The output of the code is displayed below the code cell:

```
['orange', 'mango', 'kiwi', 'papaya']
```

**TUPLE** - A tuple is a list of non-changeable objects. The differences between

tuples and lists are that the tuples cannot be changed, tuples use parentheses, whereas list uses square brackets.

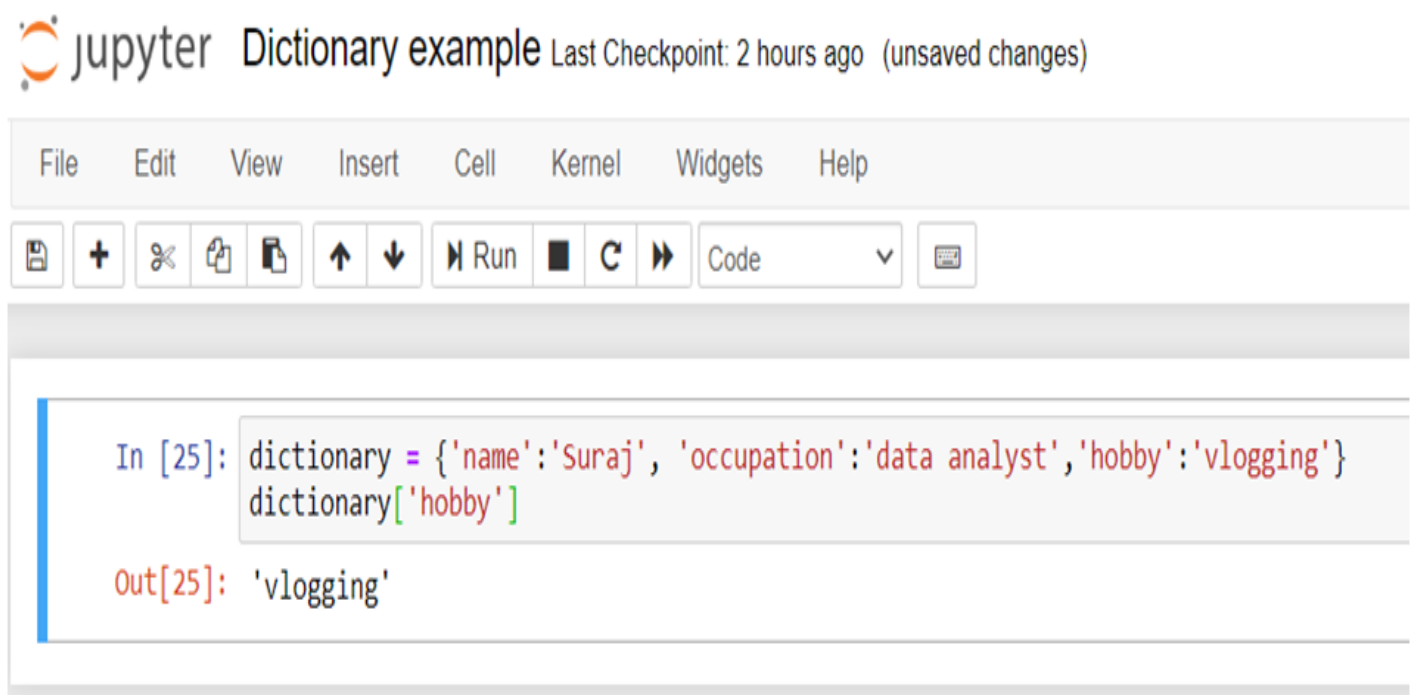


The image shows a Jupyter Notebook interface with the title "Tuple example". The top bar includes the Jupyter logo, the title, and the status "Last Checkpoint: 2 hours ago (unsaved changes)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Under the menu bar is a toolbar with icons for saving, adding cells, undo, redo, copy, paste, and navigation. The main area contains a code cell with the following text:

```
In [16]: Tuple = ('orange', 'mango', 'kiwi', 'papaya')
         #If you can see the change values are same as list but enclosed in parentheses
         print(Tuple)

         ('orange', 'mango', 'kiwi', 'papaya')
```

**DICTIONARY**- These are nothing but a type of data structure which consist of key value pairs enclosed in the curly brackets. It is same as the any dictionary we use in day to day life in which we find the meaning of the particular words. So if I compare normal dictionary to this python dictionary data structure then the a word in a dictionary will be our key and its meaning will be the value of the dictionary. In the figure name, occupation and hobby are the keys and Suraj, data analyst and vlogging are the values assigned to the keys



The image shows a Jupyter Notebook interface with the title "Dictionary example". The top bar includes the Jupyter logo, the title, and the status "Last Checkpoint: 2 hours ago (unsaved changes)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Under the menu bar is a toolbar with icons for saving, adding cells, undo, redo, copy, paste, and navigation. The main area contains a code cell with the following text:

```
In [25]: dictionary = {'name': 'Suraj', 'occupation': 'data analyst', 'hobby': 'vlogging'}
         dictionary['hobby']

Out[25]: 'vlogging'
```

# OPERATORS

**OPERATORS** - Operators are the symbols in python that are used to perform Arithmetic or logical operations. Following are the different types of operators in python.

**Arithmetic operators** - Arithmetic operators carry out mathematical operations and they are mostly used with the numeric values.

Arithmetic operators		
Operator	Name	Example
+	Addition	A+B
-	Subtraction	A-B
*	Multiplication	A*B
/	Division	A/B
%	Modulus	A%B
**	Exponentiation	A**B
//	Quotient	A//B

**Assignment operators** - As the name decides this operators are used for assigning the values to the variable

ASSIGNMENT OPERATORS		
Operator	Example	may also be written
=	a = 6	a = 6
+=	a += 3	a = a + 3
-=	a -= 4	a = a - 4
*=	a *= 5	a = a * 5
/=	a /= 6	a = a / 6
%=	a %= 7	a = a % 7
//=	a //= 8	a = a // 8
**=	a **= 9	a = a ** 9
&=	a &= 1	a = a & 1

**Logical operators** - These operators are used to join conditional statements

Logical Operators		
Operator	Description	Example
and	if both statements are true it returns true	<code>x &lt;5 and x &lt;10</code>
or	if any of the two statement is true it returns true	<code>x &lt;4 or x &lt;8</code>
not	if the result is true it reverses the result and gives false	<code>not (x &lt;4 and x &lt;8)</code>

**Comparison operators** - These operators are used to compare two different values.

Comparison operators		
Operator	Name	Example
<code>==</code>	Equal	<code>a == b</code>
<code>!=</code>	Not equal	<code>a!=b</code>
<code>&gt;</code>	Greater than	<code>a &gt;b</code>
<code>&lt;</code>	less than	<code>a &lt;b</code>
<code>&gt;=</code>	Greater than equal to	<code>a&gt;= b</code>
<code>&lt;=</code>	less than equal to	<code>a &lt;=b</code>

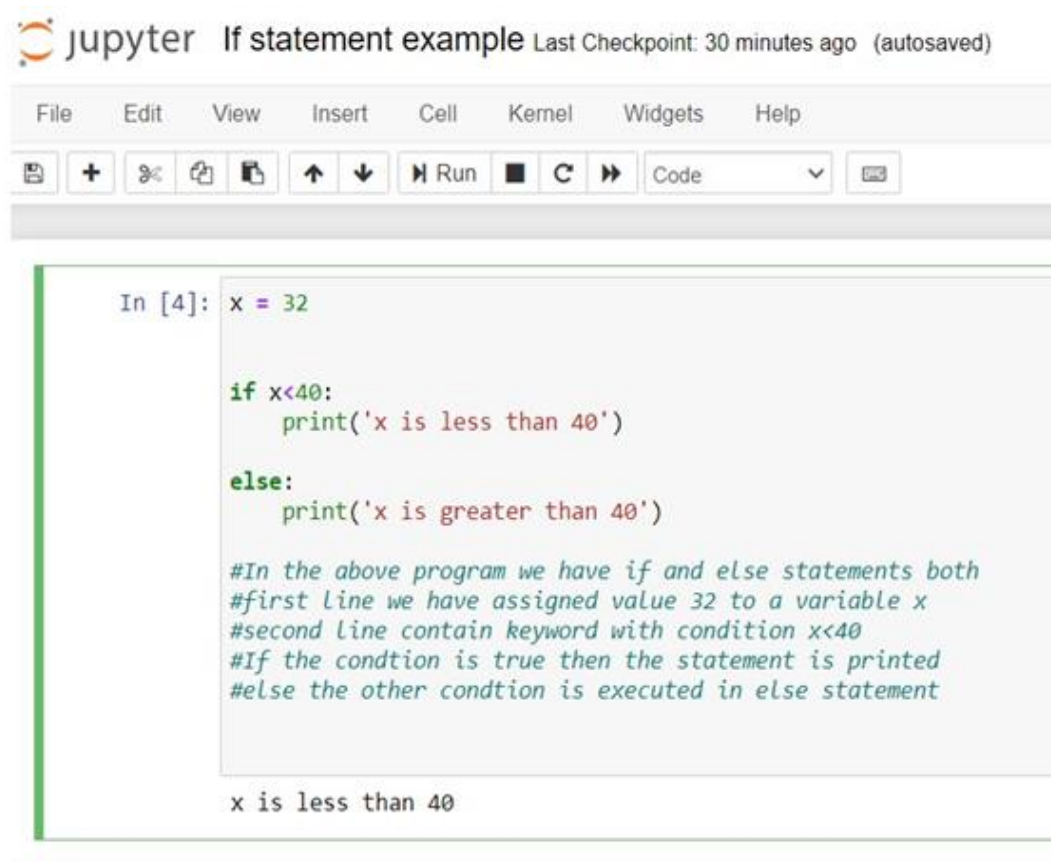
# CONDITION STATEMENTS

## If else statements

”The most common type of statement is the if statement. if statement consist of a block which is called as **clause**”.it is the block after if statement, it executed the statement if the condition is true. The statement is omitted if the condition is False. then the statement in the else part is printed

If statement consist of following -

- ✓ If keyword itself
- ✓ Condition which may be True or False
- ✓ Colon
- ✓ If clause or a block of code



The image shows a Jupyter Notebook interface with the title "If statement example" and a subtitle "Last Checkpoint: 30 minutes ago (autosaved)". The notebook has a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for file operations, code execution, and cell management. The main area of the notebook displays a code cell with the following content:

```
In [4]: x = 32

if x<40:
    print('x is less than 40')
else:
    print('x is greater than 40')
```

Below the code cell, there is a text block explaining the code:

```
#In the above program we have if and else statements both
#first line we have assigned value 32 to a variable x
#second line contain keyword with condition x<40
#If the condition is true then the statement is printed
#else the other condition is executed in else statement
```

At the bottom of the notebook, the output of the code is displayed:

```
x is less than 40
```

## elif statements

In this statement only one statement is executed, There are many cases in which there is only one possibility to execute. "The elif statement is an else if statement that always follows an if or another elif statement" The elif statement provides another condition that is checked only if any of the previous conditions were False. In code, an elif statement always consists of the following: . The only difference between if else and elif statement is that in elif statement we have the condition where as in else statement we do not have any condition.

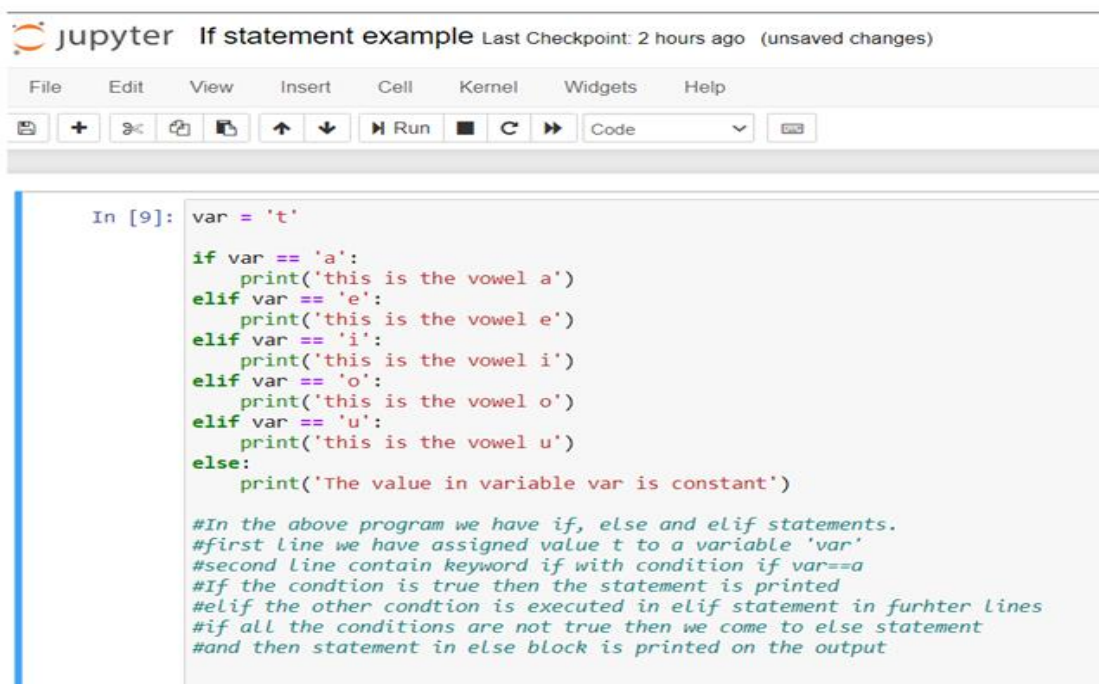
elif statement consist of following -

elif keyword itself

Condition which may be True or False

Colon

elif clause or a block of code



The screenshot shows a Jupyter Notebook window titled "If statement example" with a "Last Checkpoint: 2 hours ago (unsaved changes)" status. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, code execution, and navigation. The code cell contains the following Python code:

```
In [9]: var = 't'

if var == 'a':
    print('this is the vowel a')
elif var == 'e':
    print('this is the vowel e')
elif var == 'i':
    print('this is the vowel i')
elif var == 'o':
    print('this is the vowel o')
elif var == 'u':
    print('this is the vowel u')
else:
    print('The value in variable var is constant')

#In the above program we have if, else and elif statements.
#first line we have assigned value t to a variable 'var'
#second line contain keyword if with condition if var==a
#If the condition is true then the statement is printed
#elif the other condition is executed in elif statement in further lines
#if all the conditions are not true then we come to else statement
#and then statement in else block is printed on the output
```

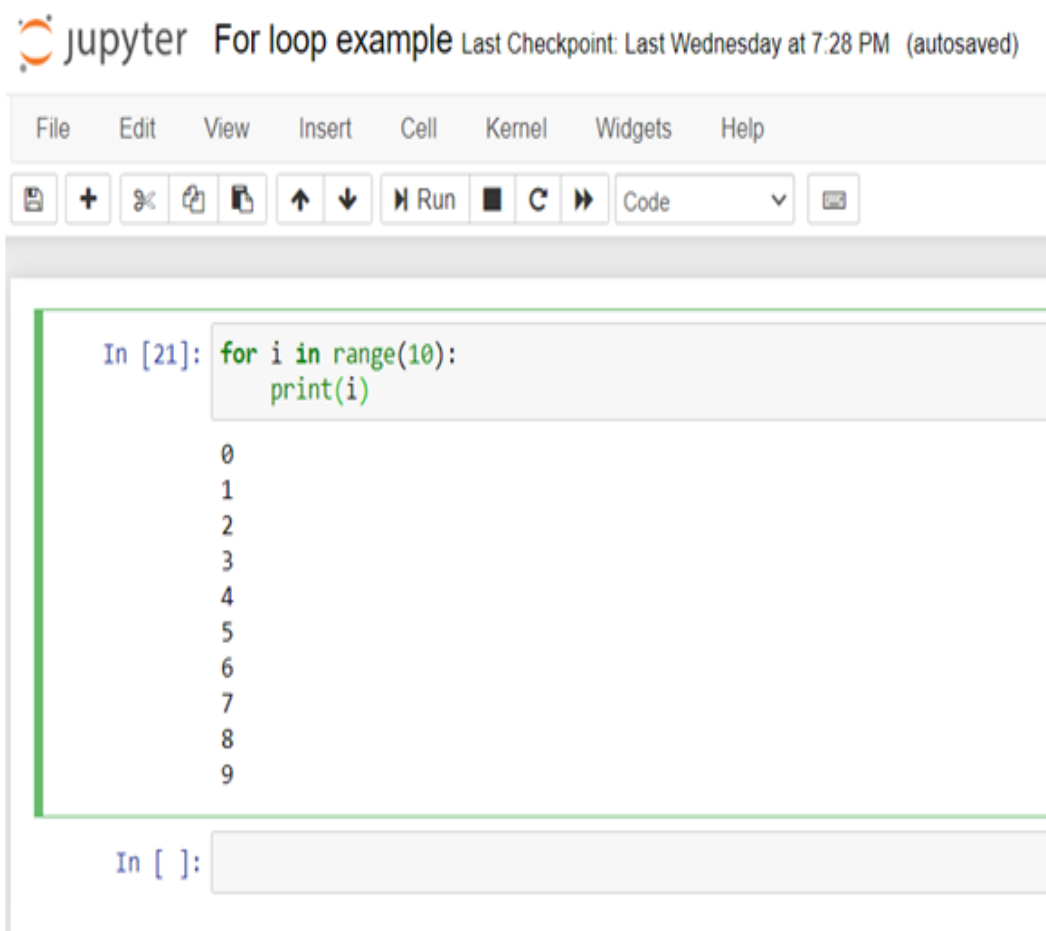
## LOOPS IN PYTHON

### For loop

When do we use for loops ?

for loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. The Python for statement iterates over the members of a sequence in order, executing the block each time.

**Range statement** - This statement 'range()' is used with for loop statements where you can specify one value. For example, if you specify 10, the loop statement starts from 1 and ends with 9, which is n-1. Also, you can specify the start and end values. The following examples demonstrate loop statements.

A screenshot of a Jupyter Notebook interface. The title bar says "Jupyter For loop example" and "Last Checkpoint: Last Wednesday at 7:28 PM (autosaved)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar has icons for saving, adding cells, undo, redo, and running code. The code cell shows a for loop that prints numbers from 0 to 9. The output of the code is displayed below the cell.

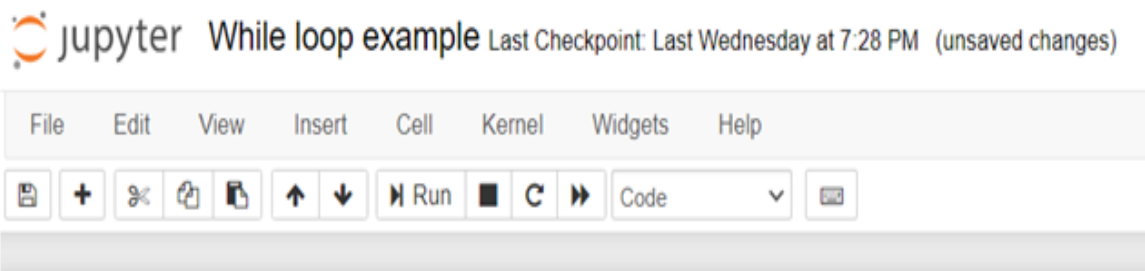
```
In [21]: for i in range(10):  
         print(i)
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

```
In [ ]:
```

## While loop

While loops are used for repeating the section of code but not same as for loop, the while loop does not run n times, but until a defined condition is no longer met. If the condition is initially false, the loop body will not be executed at all.



```
In [30]: i=1 #we have assigned value one to a variable i
while i<=5: #entering in while loop if condition is true
    print(i)
    i=i+1
print('done')
i=2
while i<=10:
    print('*'*i)
    i=i+1
print('done')
```

```
1
2
3
4
5
done
**
***
****
*****
*****
*****
*****
*****
*****
*****
done
```

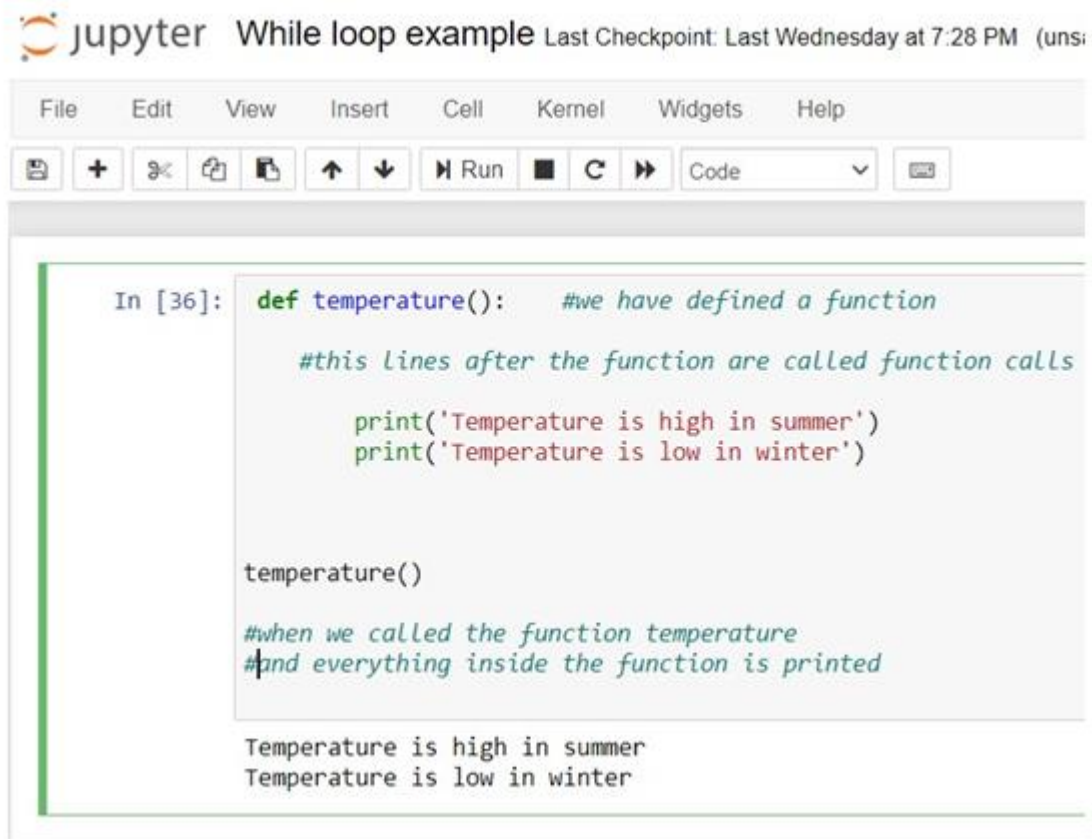
## FUNCTIONS

A function is a python code which can be reused at any anytime in the whole python code. Function performs specific task whenever it is called during the program. With the help of function the program is divided in to multiple codes.

- ✓ Built in functions - The functions which are already in the python programming and have specific action to perform are called as built in functions. This function are immutable. Some examples of this functions are -
- ✓ chr() - used to get string



- ✓ print() - used to print an object in terminal min() - used to get minimum value in terminal
- ✓ User defined functions - These functions are user-defined functions and they start with the key word 'def' as shown in the example below. We have defined the function name temperature and its task to be performed when called. Below is the example of it.



The screenshot shows a Jupyter Notebook interface with the title "While loop example". The top bar includes the Jupyter logo, the title, and the last checkpoint information: "Last Checkpoint: Last Wednesday at 7:28 PM (unsaved)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Under the menu bar is a toolbar with icons for saving, adding a new cell, undo, redo, copy, paste, up, down, run, interrupt, and a dropdown menu currently set to "Code". The main area of the notebook contains a code cell with the following content:

```
In [36]: def temperature():    #we have defined a function

        #this lines after the function are called function calls

        print('Temperature is high in summer')
        print('Temperature is low in winter')

temperature()

#when we called the function temperature
#and everything inside the function is printed

Temperature is high in summer
Temperature is low in winter
```

# LIBRARIES IN PYTHON



A (software) library is a collection of files (called modules) that contains functions for use by other programs

## TYPES OF LIBRARIES

- **PANDAS** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series
- **MATPLOTLIB** is a plotting library for the Python programming language and its numerical mathematics extension Numpy.
- **NUMPY** is a python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices
- **SEABORN** is a library for making statistical graphics in Python
- **STATSMODELS** is a python module that provides classes and functions for the estimation of many different statistical models

# DATA COLLECTION

Before analyzing and visualization we need the raw data and this raw data can be gathered from different open source data websites available on the internet. This data will be in raw form, it may be the PV solar panel sales, renewable energy consumption or production in any specific area or regions where solar or wind which one is more favorable. As here we are focusing on the renewable energy data sets so we will be considering following websites where this data is available.

<https://www.eia.gov/>

This website contains the energy data mostly of US. EIA is the abbreviated form of Energy Information Administration. Here we have different data of prices, consumption, production, exports and imports of the energy data.

<https://www.energy.gov/>

Energy.gov is the other website for data related to renewable energy, This Energy Department is responsible to make sure USA's Energy Future and solve the energy related problems

<https://openei.org/>

Open Energy Information is a website for policy makers, researchers, technology investors, venture capitalists, and market professionals with energy data, information, analyzes, tools, images, maps, and other resources.

<https://data.world/>

Here we can find data related to each and every field, it is the most widely used website for data analysis. We can also gather energy related data from this website

<https://catalog.data.gov/dataset>

Data.gov is powered by two open source applications, CKAN and WordPress, and it is developed publicly on GitHub. Data.gov is managed and hosted by the U.S. General Services Administration, Technology Transformation Service.

<https://www.kaggle.com/>

The data sets available here is not specifically for renewable energy. Kaggle is general data sets website, here you can get generalize

# USES OF DATA SCIENCE IN OUR BUSINESS

**INTRODUCTION:** TRENDYGO is one of the most popular and successful stores in the fashion world. They adopt the concept of “fast fashion”, where the whole process of designing a collection to shipping it to stores takes a maximum of three weeks. The success of this brand is attributed to this dynamic concept where the retailer studies the choices and preference of the customer to create a collection catering their tastes. They create what the customer craves instead of selling what they design. The customer themselves may not know what they are particularly looking for, but the smart business analysts and data scientists at TRENDYGO make use of the data to create a collection which the customers will automatically want because it's their “taste”

## USES:

### ✓ COLOR OPTIONS

Using Big data, we can find the colors preferred by the customers to curate a best-selling collection. The range of colors for a particular style, the combination of colors purchased together, etc. can be mined from sales data and online retail data.

### ✓ MEN'S OR WOMEN'S CLOTHING

Each designer targets a different demographic or gender to increase their popularity or sales. Designers need to decide how much items in each collection and the kind of variety they need to create. They have a fixed set of resources like budget and display space and they need data-backed guidelines to decide how much to allot to each category.

### ✓ TURNING RUNWAY STYLES TO RETAILS MERCHANDISE

Many styles featured on the runway are not “wearable” in real-life. Trends on the runway are exaggerated and too over-the-top for retail. The accessories need to be altered before they can be curated for sale in stores. Training algorithms to suggest which features to change like color, fabric, size, length, combination, etc

### ✓ ACCESSORY PRICE

For each accessory, the designers need to understand the prices the customer would be willing to pay given the quality, style, popularity and the brand value. Big Data should be used to average previous sales data to generate suggested pricing. Data from competing brands can also be used to set prices which aren't too high but still contribute to good revenues.

## ✓ **UNCOVERING NEW PRODUCT CATEGORIES**

A brand needs to find new products that will be successful in the market and which product aren't very lucratively promising. Designers need to think whether making a unique new product will be accepted or rejects by customers.

## ✓ **STORE ARRANGEMENT**

Customers exhibit a particular behavior while shopping which can be studied to arrange merchandise in a manner which increases the chances of sales of majority of the pieces. Associative data mining can help us decide where to group products together so customer is likely to pick up most of them.

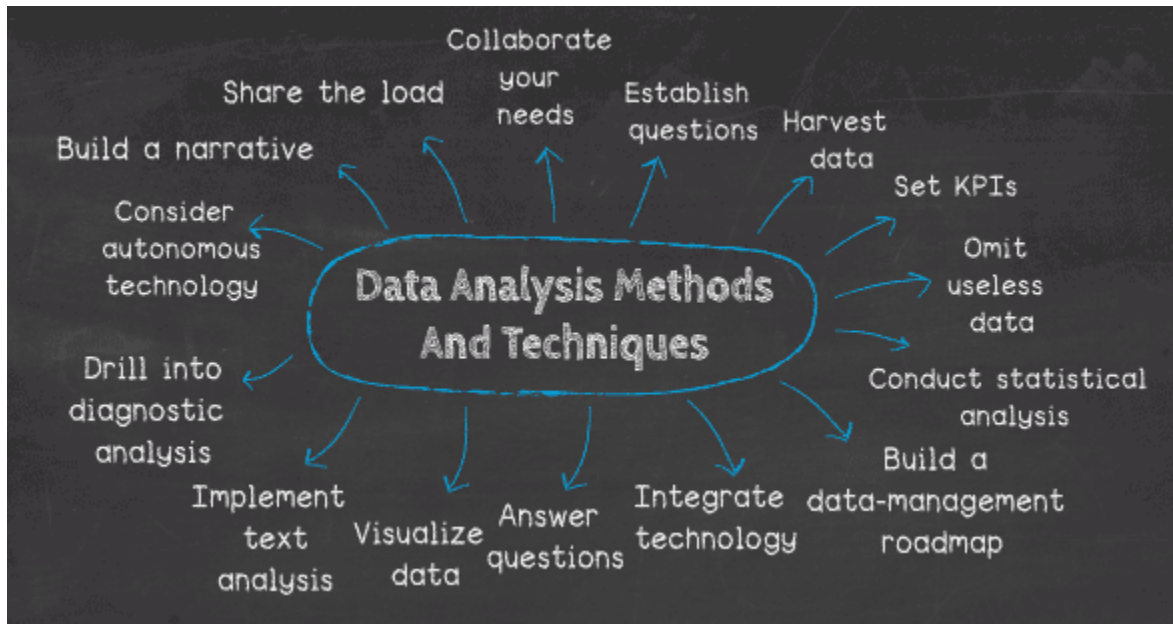
# **BENEFITS OF DATA ANALYSIS**

## **4 BIG DATA ANALYSIS BENEFITS FOR RETAIL**

- **Maintaining a 360-degree view of each customer — Create the kind of personal engagement that customers have come to expect by knowing each individual, at scale.**
- **Optimize pricing — Get the most value out of upcoming trends and know when, and how much, to decrease off-trend product prices**
- **Streamline back-office operations — Imaging maintaining perfect stock levels throughout the year and gathering data from registered products in real-time.**
- **Enhanced customer service — Unlock the customer service data hiding in recorded calls, in-store security footage, and social media comment.**

# DATA ANALYSIS METHODS & TECHNIQUES

## DIFFERENT TYPES OF ANALYSIS METHODS



- Collaborate your needs
- Establish your questions
- Harvest your data
- Omit useless data
- Conduct statistical analysis
- Build a data management roadmap
- Integrate technology
- Answer your questions
- Visualize your data
- Implement text analysis
- Drill into diagnostic analysis
- Consider autonomous technology
- Build a narrative
- Share the load



# DATA VISUALIZATION AND ANALYSIS

## TRENDYGO DATA 2019JAN-2019AUG

We will be analyzing the data with the help of some questions. Below is the figure of the data sheet in excel that will give you the hint that how the data is available to us.

1	Order ID	Accessorie	Quantity Ordered	Price Each	Order Date	Purchase Address
2	176558	Sunglasses	2	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001
3						
4	176559	Necklace	1	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
5	176560	Watch	1	600	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
6	176560	Socks	1	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
7	176561	Tie	1	11.99	4/30/2019 9:27	333 8th St, Los Angeles, CA 90001
8	176562	Bow tie	1	11.95	4/29/2019 13:03	381 Wilson St, San Francisco, CA 94016
9	176563	Ring	1	99.99	4/2/2019 7:46	668 Center St, Seattle, WA 98101
0	176564	Gloves	1	11.95	4/12/2019 10:58	790 Ridge St, Atlanta, GA 30301
1	176565	Scarf	1	1700	4/24/2019 10:38	915 Willow St, San Francisco, CA 94016
2	176566	Umbrella	1	11.99	4/8/2019 14:05	83 7th St, Boston, MA 02215
3	176567	Boots	1	600	4/18/2019 17:18	444 7th St, Los Angeles, CA 90001
4	176568	Mittens	1	14.95	4/15/2019 12:18	438 Elm St, Seattle, WA 98101
5	176569	Stockings	1	389.99	4/16/2019 19:23	657 Hill St, Dallas, TX 75001
6	176570	Earmuffs	1	3.84	4/22/2019 15:09	186 12th St, Dallas, TX 75001
7	176571	Hair band	1	14.95	4/19/2019 14:29	253 Johnson St, Atlanta, GA 30301

## first we import libraries

```
In [1]: import os
import pandas as pd
```

See that how many csv files we have in our folder (Omema\_Data)

```
In [2]: path = "C:/Users/Fujitsu-Pc/Desktop/PARKO/Omema_Data"
os.listdir(path)
```

```
Out[2]: ['TRENDYGO_April_2019.csv',
'TRENDYGO_August_2019.csv',
'TRENDYGO_February_2019..csv',
'TRENDYGO_January_2019.csv',
'TRENDYGO_July_2019.csv',
'TRENDYGO_June_2019.csv',
'TRENDYGO_March_2019.csv',
'TRENDYGO_May_2019.csv']
```

Combined all data from each csv file into one csv file

```
In [3]: path = "C:/Users/Fujitsu-Pc/Desktop/PARKO/Omema_Data"
files = [file for file in os.listdir(path) if not file.startswith('.')]
all_months_data = pd.DataFrame()
for file in files:
    my_data = pd.read_csv(path+"/"+file)
    all_months_data = pd.concat([all_months_data, my_data ])
all_months_data.to_csv('all_data.csv',index=False)
```

## read our file

```
In [4]: all_data = pd.read_csv('all_data.csv')
all_data.head(100)
```

Out[4]:

	Order ID	Accessorie	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558.0	Sunglasses	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001
1	NaN	NaN	NaN	NaN	NaN	NaN
2	176559.0	Necklace	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
3	176560.0	Watch	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560.0	Socks	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
...	...	...	...	...	...	...
95	176648.0	Earmuffs	1.0	149.99	4/24/2019 1:17	732 2nd St, Portland, OR 97035
96	176649.0	Hair band	1.0	11.95	4/9/2019 8:49	702 11th St, San Francisco, CA 94016
97	176650.0	Safety pin	1.0	14.95	4/12/2019 16:47	153 River St, Boston, MA 02215
98	176651.0	Pocket watch	1.0	700.00	4/7/2019 13:14	997 South St, Boston, MA 02215
99	176652.0	Apron	1.0	600.00	4/9/2019 20:04	502 14th St, New York City, NY 10001

## find empty data (nan)

```
In [5]: nan_df = all_data[all_data.isna().any(axis=1)]
display(nan_df.head())
```

	Order ID	Accessorie	Quantity Ordered	Price Each	Order Date	Purchase Address
1	NaN	NaN	NaN	NaN	NaN	NaN
16	NaN	NaN	NaN	NaN	NaN	NaN
25	NaN	NaN	NaN	NaN	NaN	NaN
37	NaN	NaN	NaN	NaN	NaN	NaN
356	NaN	NaN	NaN	NaN	NaN	NaN

## clean our data from (nan)

```
In [6]: all_data = all_data.dropna(how='all')
all_data.head()
```

Out[6]:

	Order ID	Accessorie	Quantity Ordered	Price Each	Order Date	Purchase Address
0	176558.0	Sunglasses	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001
2	176559.0	Necklace	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215
3	176560.0	Watch	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
4	176560.0	Socks	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001
5	176561.0	Tie	1.0	11.99	4/30/2019 9:27	333 8th St, Los Angeles, CA 90001

## Remove text from order date column

```
In [10]: all_data = all_data[all_data['Order Date'].str[0:1]!='Or']
```

```
In [11]: all_data['Quantity Ordered'] = pd.to_numeric(all_data['Quantity Ordered'])
all_data['Price Each'] = pd.to_numeric(all_data['Price Each'])
```

## Separate month column

```
In [12]: all_data['Month'] = all_data['Order Date'].str[0:1]
all_data['Month'] = all_data['Month'].astype('int32')
all_data.head()
```

```
Out[12]:
```

	Order ID	Accessorie	Quantity Ordered	Price Each	Order Date	Purchase Address	Month
0	176558.0	Sunglasses	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001	4
2	176559.0	Necklace	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215	4
3	176560.0	Watch	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001	4
4	176560.0	Socks	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001	4
5	176561.0	Tie	1.0	11.99	4/30/2019 9:27	333 8th St, Los Angeles, CA 90001	4

## which month column you want to make

```
In [34]: all_data['Month 8'] = pd.to_datetime(all_data['Order Date']).dt.month
all_data.head()
```

```
Out[34]:
```

	Order ID	Accessorie	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Month 2	Month 3	Month 8	City	Sales	Hour	Minute	Count
0	176558.0	Sunglasses	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001	4	4	4	4	Dallas (TX)	23.90	8	46	1
2	176559.0	Necklace	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215	4	4	4	4	Boston (MA)	99.99	22	30	1
3	176560.0	Watch	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001	4	4	4	4	Los Angeles (CA)	600.00	14	38	1
4	176560.0	Socks	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001	4	4	4	4	Los Angeles (CA)	11.99	14	38	1
5	176561.0	Tie	1.0	11.99	4/30/2019 9:27	333 8th St, Los Angeles, CA 90001	4	4	4	4	Los Angeles (CA)	11.99	9	27	1

## ADD CITY COLUMN

```
In [ ]: #Add city column
```

```
In [16]: def get_city(address):
          return address.split(",")[1].strip(" ")

          def get_state(address):
              return address.split(",")[2].split(" ")[1]

          all_data['City'] = all_data['Purchase Address'].apply(lambda x: f"{get_city(x)} ({get_state(x)})")
          all_data.head()
```

```
Out[16]:
```

	Order ID	Accessorie	Quantity Ordered	Price Each	Order Date	Purchase Address	Month	Month 2	Month 3	Month 8	City
0	176558.0	Sunglasses	2.0	11.95	4/19/2019 8:46	917 1st St, Dallas, TX 75001	4	4	4	4	Dallas (TX)
2	176559.0	Necklace	1.0	99.99	4/7/2019 22:30	682 Chestnut St, Boston, MA 02215	4	4	4	4	Boston (MA)
3	176560.0	Watch	1.0	600.00	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001	4	4	4	4	Los Angeles (CA)
4	176560.0	Socks	1.0	11.99	4/12/2019 14:38	669 Spruce St, Los Angeles, CA 90001	4	4	4	4	Los Angeles (CA)
5	176561.0	Tie	1.0	11.99	4/30/2019 9:27	333 8th St, Los Angeles, CA 90001	4	4	4	4	Los Angeles (CA)

## find which month had maximum number of sales of accessories

```
In [18]: all_data.groupby(['Month']).sum()
```

```
Out[18]:
```

	Order ID	Quantity Ordered	Price Each	Month 2	Month 3	Month 8	Sales
Month							
1	70597051.0	564.0	89802.02	499	499	499	90451.00
2	74315791.0	552.0	82601.61	986	986	986	83027.79
3	80600343.0	568.0	103662.00	1491	1491	1491	104276.30
4	87322905.0	554.0	79404.47	1976	1976	1976	79968.59
5	96370466.0	569.0	94883.61	2480	2480	2480	95285.07
6	104450160.0	559.0	96303.28	2982	2982	2982	98572.31
7	110905869.0	563.0	77071.62	3479	3479	3479	77655.62
8	117508182.0	574.0	85603.41	3968	3968	3968	86667.00

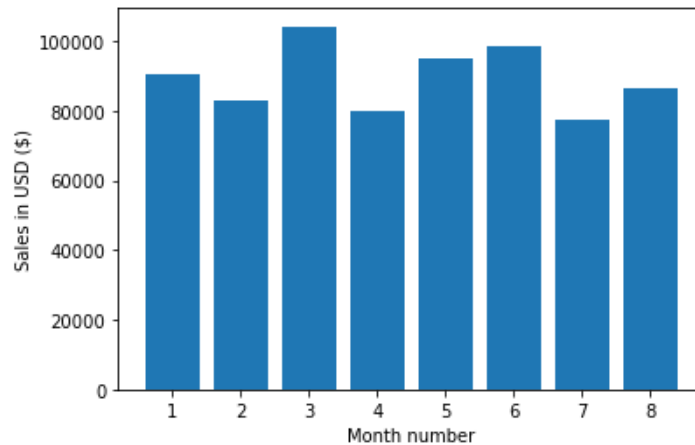
## plotting graph we are showing month that had maximum sales

```
In [20]: import matplotlib.pyplot as plt

months = range(1,9)
print(months)

plt.bar(months,all_data.groupby(['Month']).sum()['Sales'])
plt.xticks(months)
plt.ylabel('Sales in USD ($)')
plt.xlabel('Month number')
plt.show()
```

range(1, 9)



## Which city having maximum num of sales of our accessories

```
In [21]: all_data.groupby(['City']).sum()
```

```
Out[21]:
```

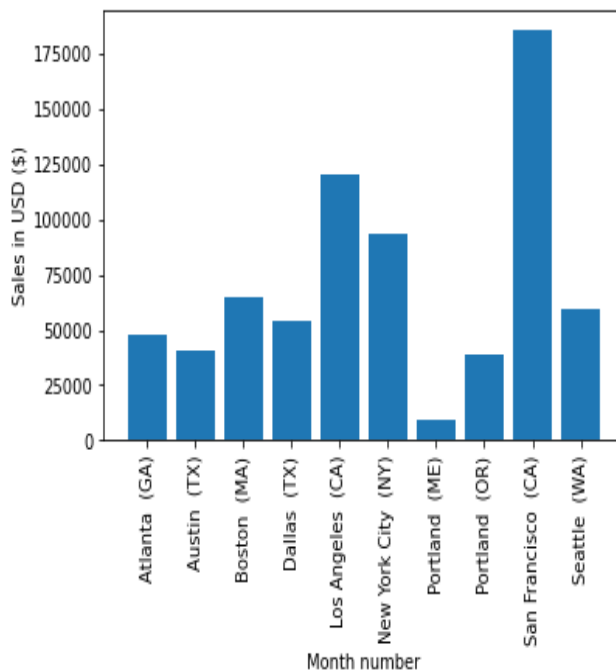
	Order ID	Quantity Ordered	Price Each	Month	Month 2	Month 3	Month 8	Sales
City								
Atlanta (GA)	55470733.0	328.0	47872.81	1295	1295	1295	1295	48157.73
Austin (TX)	40554274.0	253.0	40240.11	957	957	957	957	40500.70
Boston (MA)	77212067.0	462.0	63996.88	1849	1849	1849	1849	64960.83
Dallas (TX)	54052107.0	336.0	54059.80	1277	1277	1277	1277	54411.90
Los Angeles (CA)	126647654.0	756.0	119735.93	3078	3078	3078	3078	120416.05
New York City (NY)	97421987.0	591.0	93326.01	2322	2322	2322	2322	93696.79
Portland (ME)	7030359.0	41.0	9312.98	173	173	173	173	9334.75
Portland (OR)	41834736.0	243.0	38987.86	1047	1047	1047	1047	39291.60
San Francisco (CA)	181085685.0	1097.0	182338.24	4433	4433	4433	4433	185345.63
Seattle (WA)	60761165.0	396.0	59461.40	1430	1430	1430	1430	59787.70

## plotting graph we find which city having maximum num of sales of our accessorie

In [23]: `import matplotlib.pyplot as plt`

```
keys = [city for city, df in all_data.groupby(['City'])]

plt.bar(keys, all_data.groupby(['City']).sum()['Sales'])
plt.ylabel('Sales in USD ($)')
plt.xlabel('Month number')
plt.xticks(keys, rotation='vertical', size=10)
plt.show()
```



## **CONCLUSION**

Without a doubt we can say that data powered decisions will give you an edge in the competitive world of fashion. Before creating any product, data needs to be consulted to see it is economically feasible and promising. Selectively using your data to create and convert product lines your customers are sure to buy in the future would help the retail houses survive in the wake of e-commerce. Some may say that AI can dull the creativity of the collections by just creating what the customer want. But that is why the outcomes should be used only to supplement the human creative insight instead of entirely replacing it. But it doesn't hurt to create the right product at the right price at the right time.



# **WORKING ON DATA ANALYSIS PROJECT**

## **REPORT AND CODING**

**OMEMA DANIYAL**

## **DATA MAKING**

**RAFLA SYED**

## **VIDEO MAKING**

**SARA ZAFAR**