EXPERIMENT NO. 01

Title: Installation of Python and study of basics.

Outcome: Students will be able to install Python and study of basics.

Theory:

Python is a widely used high-level programming language first launched in 1991. Since then, Python has been gaining popularity and is considered as one of the most popular and flexible server-side programming languages.

Prerequisites

- A system running Windows 10 with admin privileges
- Command Prompt (comes with Windows by default)

Python 3 Installation on Windows

Step 1: Select Version of Python to Install

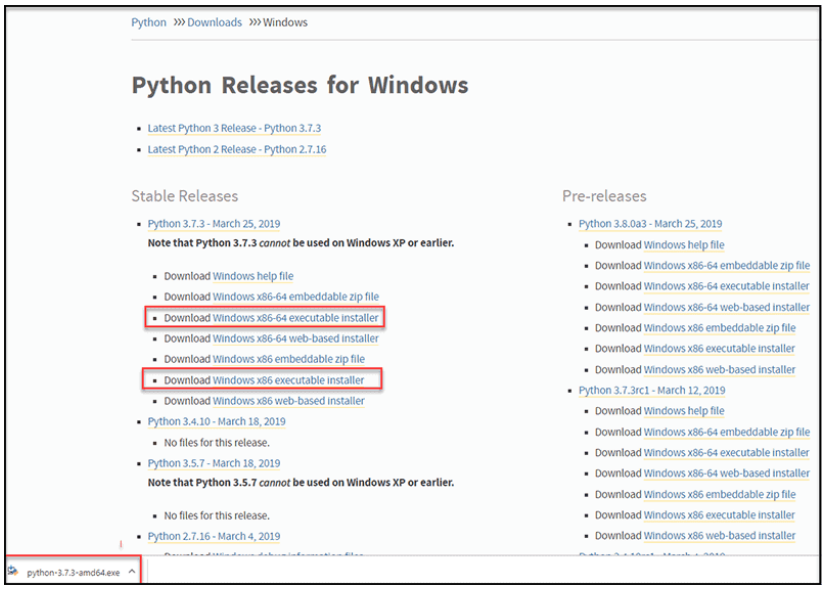
The installation procedure involves downloading the official Python .exe installer and running it on your system.

The version you need depends on what you want to do in Python. For example, if you are working on a project coded in Python version 2.6, you probably need that version. If you are starting a project from scratch, you have the freedom to choose.

If you are learning to code in Python, we recommend you **download both the latest version of Python 2 and 3**. Working with Python 2 enables you to work on older projects or test new projects for backward compatibility.

Step 2: Download Python Executable Installer

1. Open your web browser and navigate to the Downloads for Windows section of the official Python website.
2. Search for your desired version of Python. At the time of publishing this article, the latest Python 3 release is version 3.7.3, while the latest Python 2 release is version 2.7.16.
3. Select a link to download either the **Windows x86-64 executable installer** or **Windows x86 executable installer**. The download is approximately 25MB.

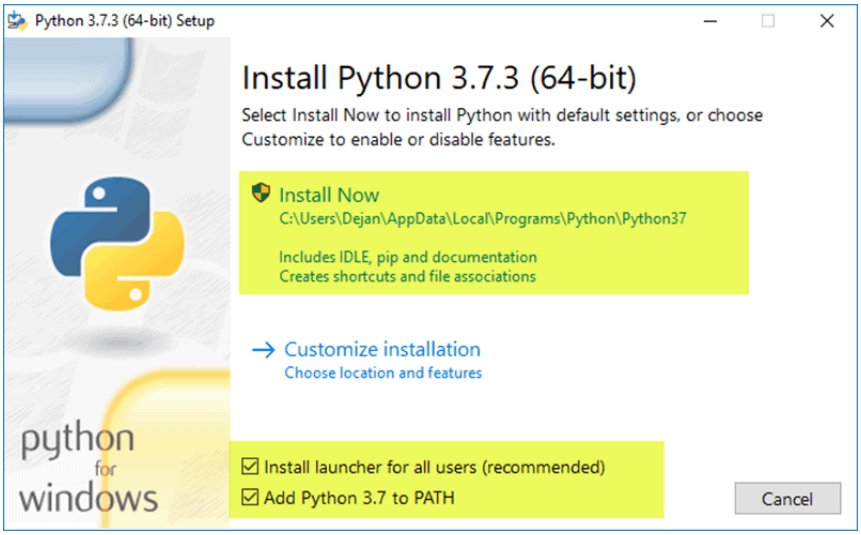


Step 3: Run Executable Installer

1. Run the **Python Installer** once downloaded. (In this example, we have downloaded Python 3.7.3.)

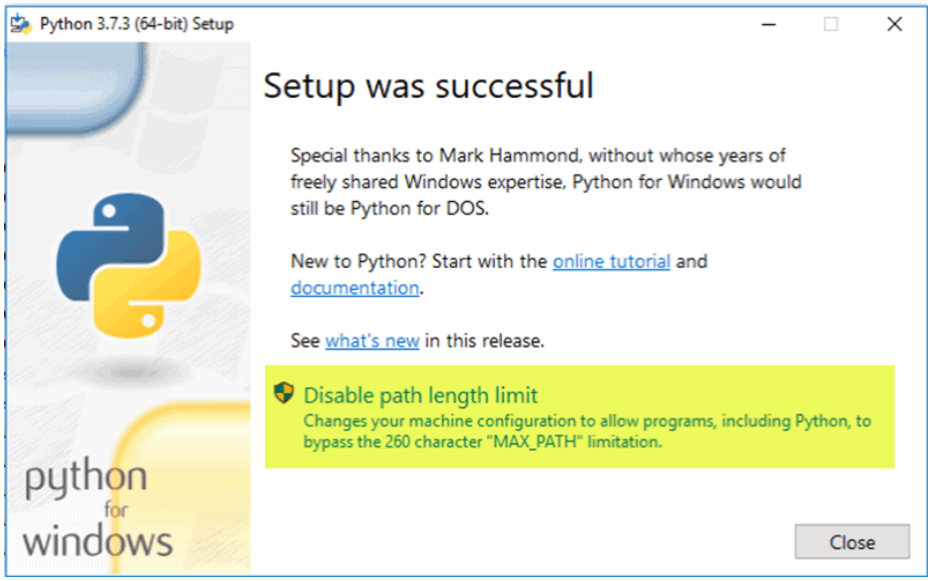
2. Make sure you select the **Install launcher for all users** and **Add Python 3.7 to PATH** checkboxes. The latter places the interpreter in the execution path. For older versions of Python that do not support the **Add Python to Path** checkbox, see Step 6.

3. Select **Install Now** – the recommended installation options.



For all recent versions of Python, the recommended installation options include **Pip** and **IDLE**. Older versions might not include such additional features.

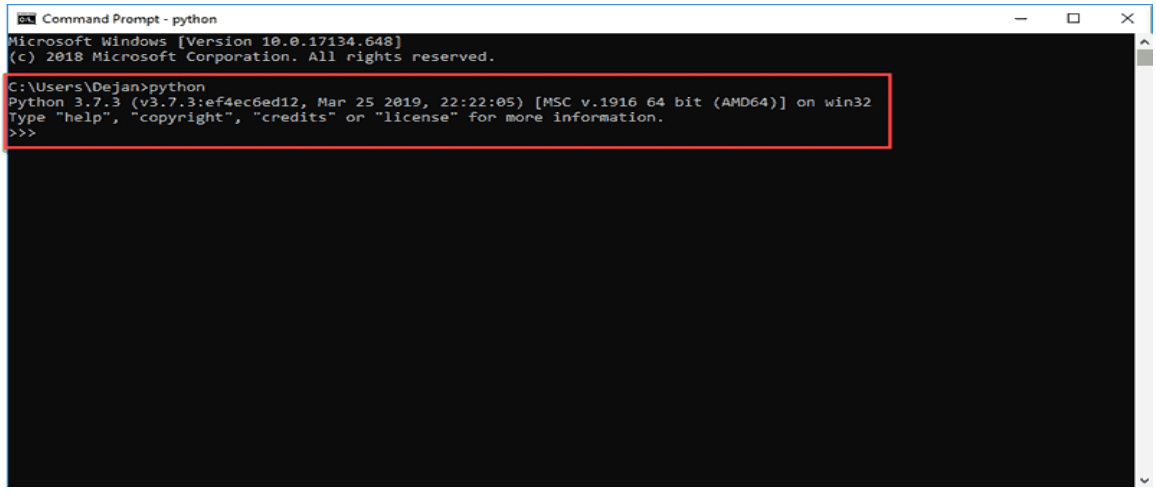
4. The next dialog will prompt you to select whether to **Disable path length limit**. Choosing this option will allow Python to bypass the 260-character MAX_PATH limit. Effectively, it will enable Python to use long path names.



The **Disable path length limit** option will not affect any other system settings. Turning it on will resolve potential name length issues that may arise with Python projects developed in Linux.

Step 4: Verify Python Was Installed On Windows

1. Navigate to the directory in which Python was installed on the system. In our case, it is **C:\Users*Username*\AppData\Local\Programs\Python\Python37** since we have installed the latest version.
2. Double-click **python.exe**.
3. The output should be similar to what you can see below:



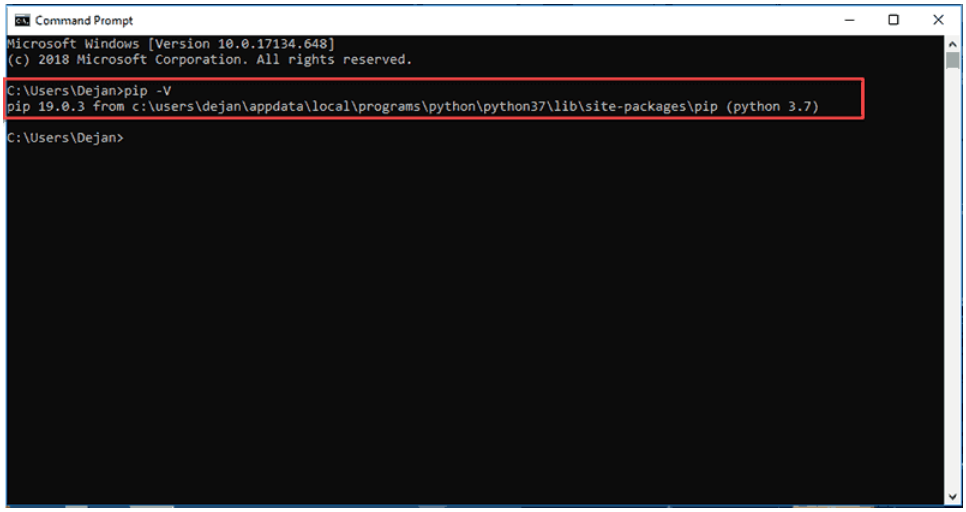
Step 5: Verify Pip Was Installed

If you opted to install an older version of Python, it is possible that it did not come with Pip preinstalled. Pip is a powerful package management system for Python software packages. Thus, make sure that you have it installed.

We recommend using Pip for most Python packages, especially when working in virtual environments.

To verify whether Pip was installed:

1. Open the **Start** menu and type "**cmd.**"
2. Select the **Command Prompt** application.
3. Enter **pip -V** in the console. If Pip was installed successfully, you should see the following output:



Pip has not been installed yet if you get the following output:

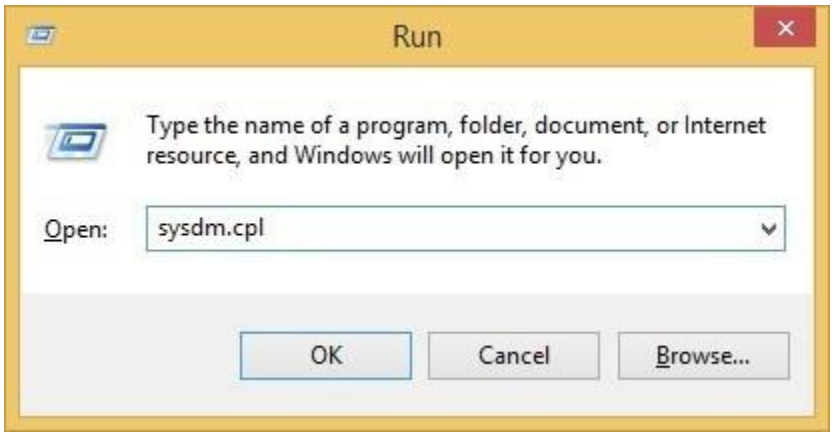
```
'pip' is not recognized as an internal or external command,
Operable program or batch file.
```

Step 6: Add Python Path to Environment Variables (Optional)

We recommend you go through this step if your version of the Python installer does not include the **Add Python to PATH** checkbox or if you have not selected that option.

Setting up the Python path to system variables alleviates the need for using full paths. It instructs Windows to look through all the PATH folders for “python” and find the install folder that contains the python.exe file.

1. Open the **Start** menu and start the **Run** app.



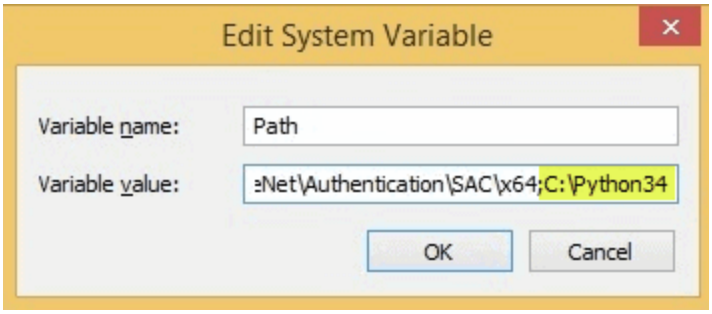
2. Type **sysdm.cpl** and click **OK**. This opens the **System Properties** window.

3. Navigate to the **Advanced** tab and select **Environment Variables**.

4. Under **System Variables**, find and select the **Path** variable.

5. Click **Edit**.

6. Select the **Variable value** field. Add the path to the **python.exe** file preceded with a **semicolon (;)**. For example, in the image below, we have added "**;C:\Python34.**"



7. Click **OK** and close all windows.

By setting this up, you can execute Python scripts like this: **Python script.py**

Install Python 3 on Ubuntu

Prerequisites

- A system running Ubuntu 18.04 or Ubuntu 20.04
- A user account with **sudo** privileges
- Access to a terminal window/command-line (**Ctrl**–**Alt**–**T**)

Option 1: Install Python 3 Using apt (Easier)

This process uses the **apt package manager** to install Python. There are fewer steps, but it's dependent on a third party hosting software updates. You may not see new releases as quickly on a third-party repository.

Most factory versions of Ubuntu 18.04 or Ubuntu 20.04 come with Python pre-installed. Check your version of Python by entering the following:

```
python --version
```

If the revision level is lower than 3.7.x, or if Python is not installed, continue to the next step.

Step 1: Update and Refresh Repository Lists

Open a terminal window, and enter the following:

```
sudo apt update
```

Step 2: Install Supporting Software

The software-properties-common package gives you better control over your package manager by letting you add PPA (Personal Package Archive) repositories. Install the supporting software with the command:

```
sudo apt install software-properties-common
```

```
sofija@sofija-VirtualBox:~$ sudo apt install build-essential zlib1g-dev libncur
ses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev libffi-dev wget
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cpp cpp-7 dpkg-dev fakeroot g++ g++-7 gcc gcc-7 gcc-7-base gcc-8-base
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libasan4 libatomic1 libc-dev-bin libc6-dev libcc1-0 libcilkrts5
  libdpkg-perl libfakeroot libgcc-7-dev libgcc1 libgomp1 libitm1 liblsan0
  libmpx2 libnspr4-dev libnss3 libquadmath0 libssl1.1 libstdc++-7-dev
  libstdc++6 libtinfo-dev libtsan0 libubsan0 linux-libc-dev make manpages-dev
Suggested packages:
  cpp-doc gcc-7-locales debian-keyring g++-multilib g++-7-multilib gcc-7-doc
```

Step 3: Add Deadsnakes PPA

Deadsnakes is a PPA with newer releases than the default Ubuntu repositories. Add the PPA by entering the following:

```
sudo add-apt-repository ppa:deadsnakes/ppa
```

The system will prompt you to press enter to continue. Do so, and allow it to finish. Refresh the package lists again:

```
sudo apt update
```

Step 4: Install Python 3

Now you can start the installation of Python 3.8 with the command:

```
sudo apt install python3.8
```

Allow the process to complete and verify the Python version was installed sucessfully::

```
python --version
```

```
sofija@sofija-VirtualBox:~$ python3 --version
Python 3.8.2
```

Option 2: Install Python 3.7 From Source Code (Latest Version)

Use this process to download and compile the source code from the developer. It's a bit more complicated, but the trade-off is accessing a newer release of Python.

Step 1: Update Local Repositories

To update local repositories, use the command:

```
sudo apt update
```

Step 2: Install Supporting Software

Compiling a package from source code requires additional software.
Enter the following to install the required packages for Python:

```
sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl
-dev libreadline-dev libffi-dev wget
```

```
sofija@sofija-VirtualBox:~$ sudo apt install build-essential zlib1g-dev libncur
ses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev libffi-dev wget
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cpp cpp-7 dpkg-dev fakeroot g++ g++-7 gcc gcc-7 gcc-7-base gcc-8-base
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl
  libasan4 libatomic1 libc-dev-bin libc6-dev libcc1-0 libcilkrts5
  libdpkg-perl libfakeroot libgcc-7-dev libgcc1 libgomp1 libitm1 liblsan0
  libmpx2 libnspr4-dev libnss3 libquadmath0 libssl1.1 libstdc++-7-dev
  libstdc++6 libtinfo-dev libtsan0 libubsan0 linux-libc-dev make manpages-dev
Suggested packages:
  cpp-doc gcc-7-locales debian-keyring g++-multilib g++-7-multilib gcc-7-doc
```

Step 3: Download the Latest Version of Python Source Code

To download the newest release of Python Source Code, navigate to the **/tmp** directory and use the **wget** command:

```
cd /tmp
wget https://www.python.org/ftp/python/3.7.5/Python-3.7.5.tgz
```

```
sofija@sofija-VirtualBox:/tmp$ wget https://www.python.org/ftp/python/3.8.3/Pyt
hon-3.8.3.tgz
--2020-06-05 16:45:55--  https://www.python.org/ftp/python/3.8.3/Python-3.8.3.t
gz
Resolving www.python.org (www.python.org)... 151.101.196.223, 2a04:4e42:2e::223
Connecting to www.python.org (www.python.org)|151.101.196.223|:443... connected
.
HTTP request sent, awaiting response... 200 OK
Length: 24067487 (23M) [application/octet-stream]
Saving to: ‘Python-3.8.3.tgz’

Python-3.8.3.tgz    100%[==================>]  22,95M  4,01MB/s    in 7,4s

2020-06-05 16:46:03 (3,09 MB/s) - ‘Python-3.8.3.tgz’ saved [24067487/24067487]
```

Step 4: Extract Compressed Files

Next, you need to extract the tgz file you downloaded, with the command:

```
tar -xf Python-3.8.3.tgz
```

Step 5: Test System and Optimize Python

Before you install the software, make sure you test the system and optimize Python.
The **./configure** command evaluates and prepares Python to install on your system. Using the **--optimization** option speeds code execution by 10-20%.
Enter the following:

```
cd python-3.8.3
./configure --enable-optimizations
```

This step can take up to 30 minutes to complete.

Step 6: Install a Second Instance of Python (recommended)

To create a second installation of Python 3.835, in addition to your current Python installation, enter the following:

```
sudo make altinstall
```

It is recommended that you use the **altinstall** method. Your Ubuntu system may have software packages dependent on Python 2.x.

(Option) Overwrite Default Python Installation

To install Python 3.8.3 over the top of your existing Python, enter the following:

```
sudo make install
```

Allow the process to complete.

Step 7: Verify Python Version

Enter the following:

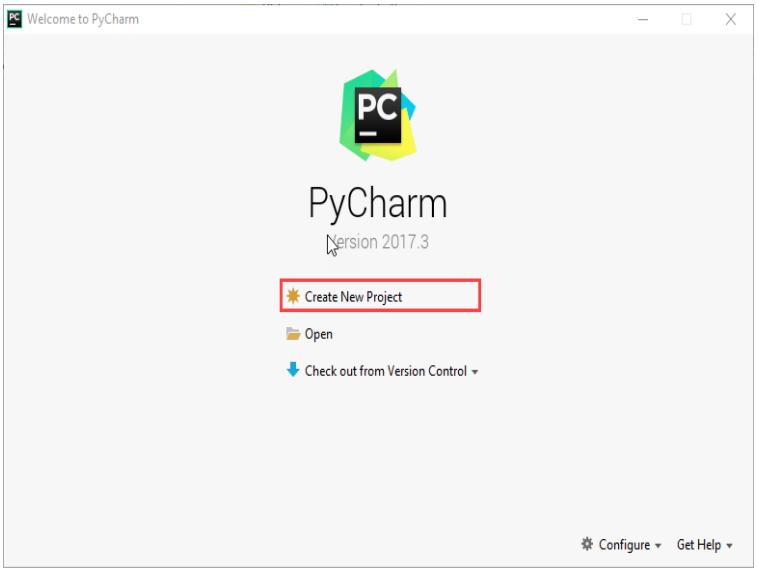
```
python3 --version
```

Basics of Python:-

Hello World: Create your First Python Program

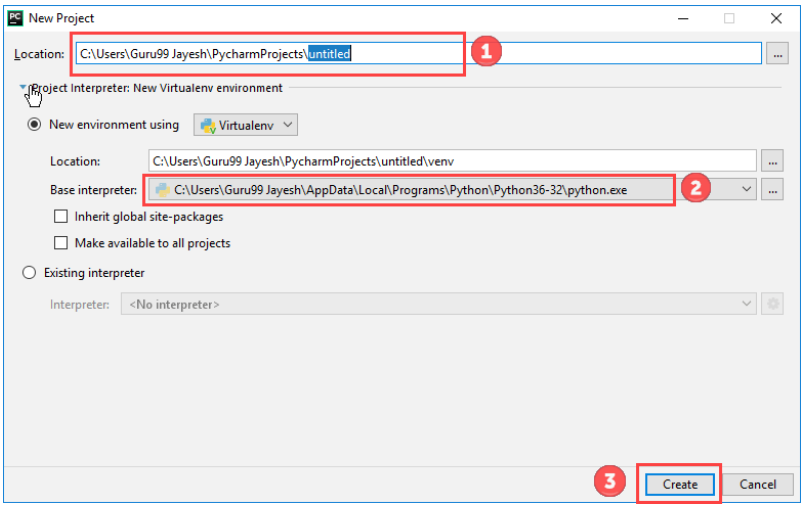
Creating First Program

Step 1) Open PyCharm Editor. You can see the introductory screen for PyCharm. To create a new project, click on "Create New Project".

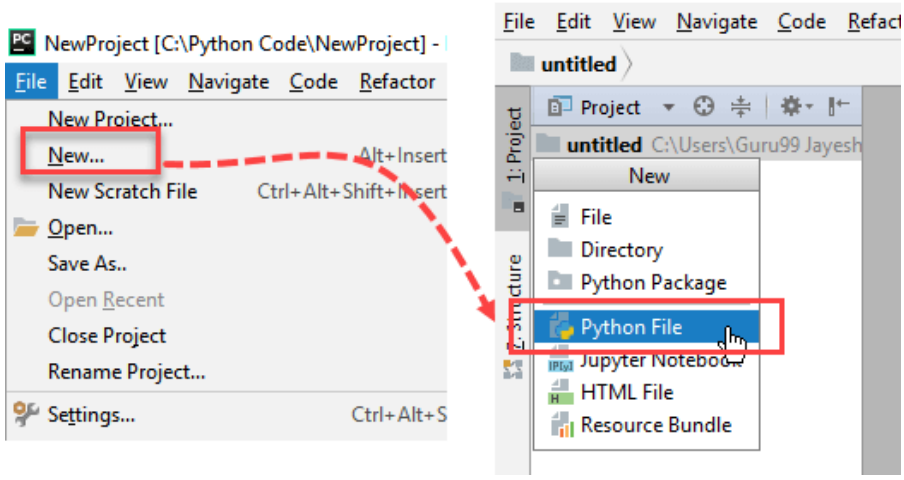


Step 2) You will need to select a location.

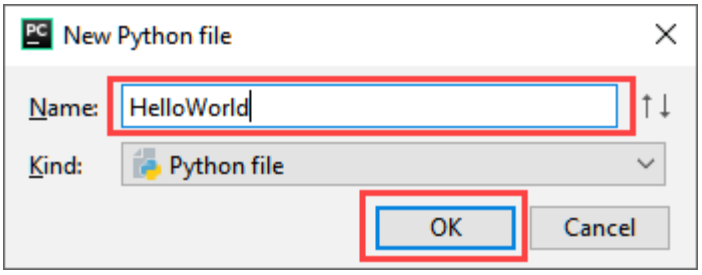
1. You can select the location where you want the project to be created. If you don't want to change location than keep it as it is but at least change the name from "untitled" to something more meaningful, like "FirstProject".
2. PyCharm should have found the Python interpreter you installed earlier.
3. Next Click the "Create" Button.



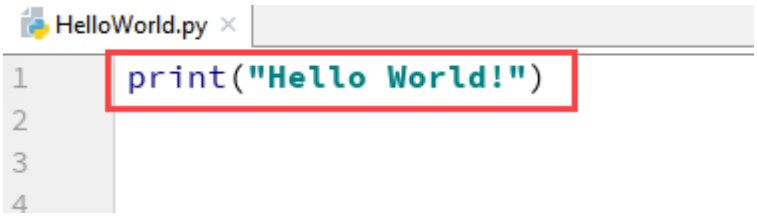
Step 3) Now Go up to the "File" menu and select "New". Next, select "Python File".



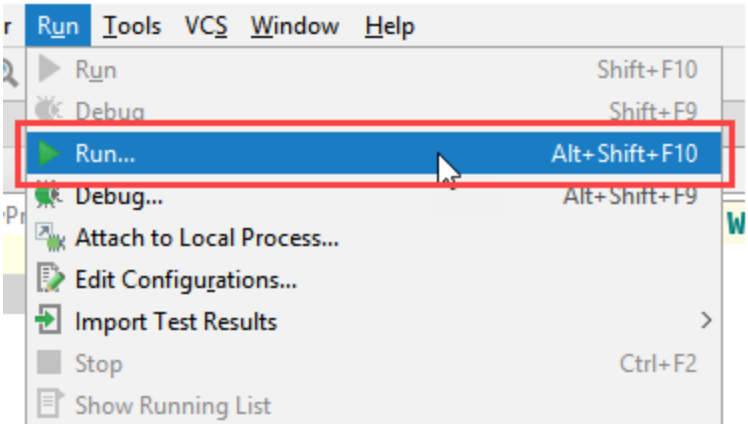
Step 4) A new pop up will appear. Now type the name of the file you want (Here we give "HelloWorld") and hit "OK".



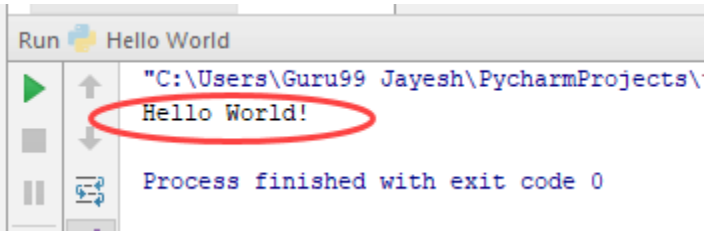
Step 5) Now type a simple program – print ('Hello World!').



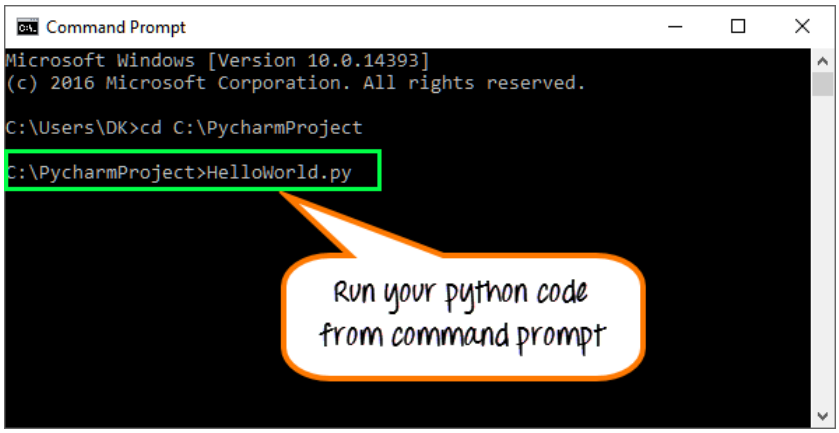
Step 6) Now Go up to the "Run" menu and select "Run" to run your program.



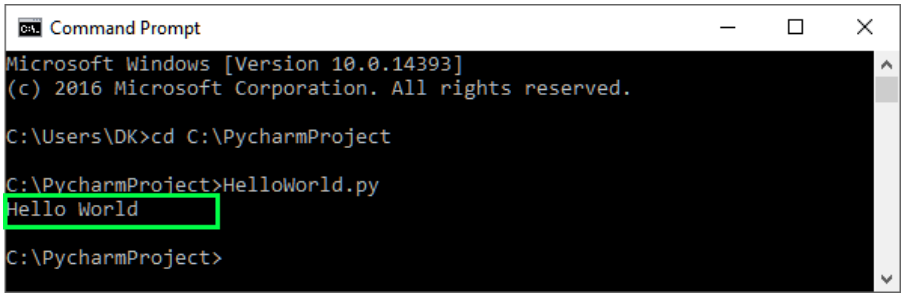
Step 7) You can see the output of your program at the bottom of the screen.



Step 8) Don't worry if you don't have Pycharm Editor installed, you can still run the code from the command prompt. Enter the correct path of a file in command prompt to run the program.



The output of the code would be



Step 9) If you are still not able to run the program, we have Python Editor for you.

print("Hello World")

Variable in Python

A Python variable is a reserved memory location to store values. In other words, a variable in a python program gives data to the computer for processing.

Python Variable Types

Every value in Python has a datatype. Different data types in Python are Numbers, List, Tuple, Strings, Dictionary, etc. Variables in Python can be declared by any name or even alphabets like a, aa, abc, etc.

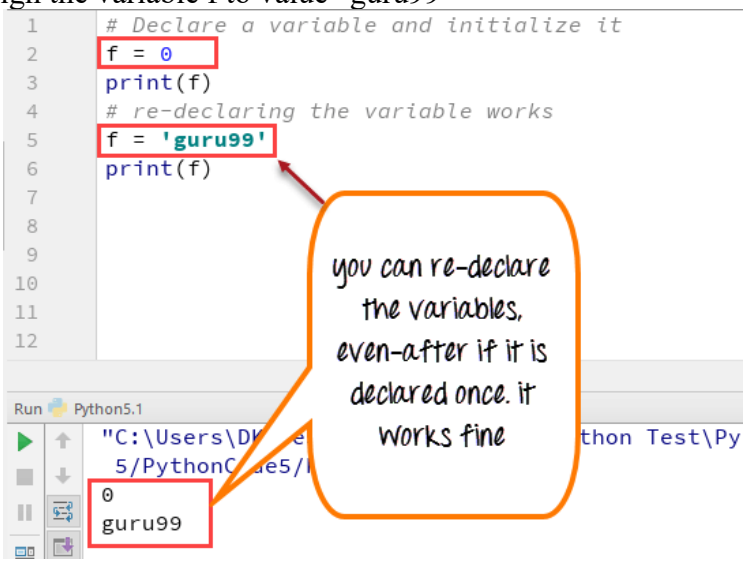
How to Declare and use a Variable

Let see an example. We will define variable in Python and declare it as "a" and print it.

a=100
print (a)

Re-declare a Variable

You can re-declare Python variables even after you have declared once.
Here we have Python declare variable initialized to f=0.
Later, we re-assign the variable f to value "guru99"



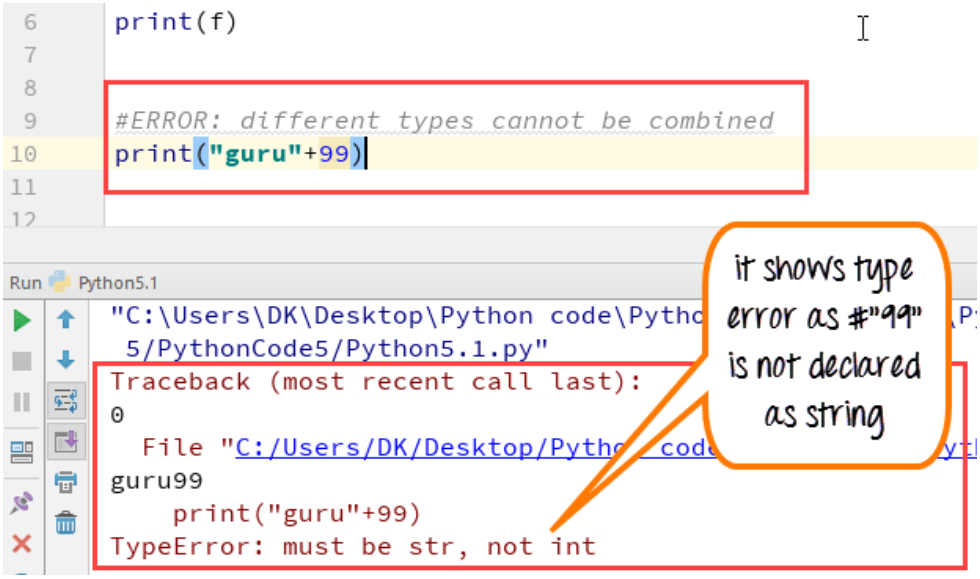
Python 2 Example

```
# Declare a variable and initialize it
f = 0
print f
# re-declaring the variable works
f = 'guru99'
print f
```

Python String Concatenation and Variable

Let's see whether you can concatenate different data types like string and number together. For example, we will concatenate "Guru" with the number "99".

Unlike Java, which concatenates number with string without declaring number as string, while declaring variables in Python requires declaring the number as string otherwise it will show a TypeError



For the following code, you will get undefined output –

```
a="Guru"
b = 99
print a+b
```

Once the integer is declared as string, it can concatenate both "Guru" + **str**("99")= "Guru99" in the output.

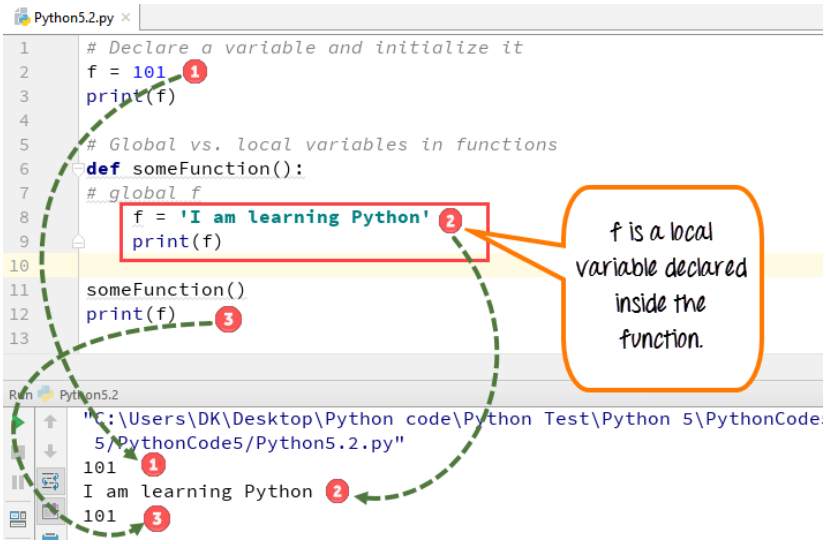
Python Variable Types: Local & Global

There are two types of variables in Python, Global variable and Local variable. When you want to use the same variable for rest of your program or module you declare it as a

global variable, while if you want to use the variable in a specific function or method, you use a local variable while Python variable declaration.

Let's understand this Python variable types with the difference between local and global variables in the below program.

1. Let us define variable in Python where the variable "f" is **global** in scope and is assigned value 101 which is printed in output
2. Variable f is again declared in function and assumes **local** scope. It is assigned value "I am learning Python." which is printed out as an output. This Python declare variable is different from the global variable "f" defined earlier
3. Once the function call is over, the local variable f is destroyed. At line 12, when we again, print the value of "f" is it displays the value of global variable f=101



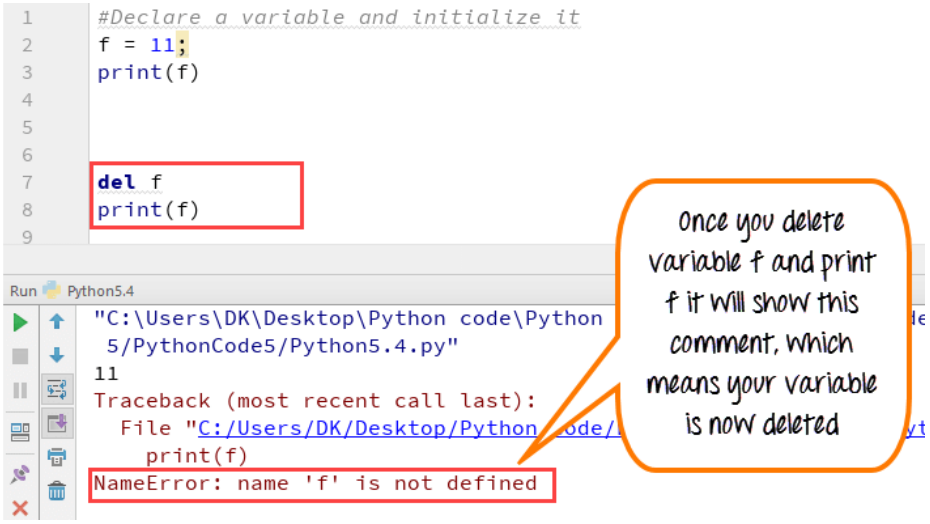
Python 2 Example

```
# Declare a variable and initialize it
f = 101
print f
# Global vs. local variables in functions
def someFunction():
# global f
   f = 'I am learning Python'
   print f
someFunction()
print f
```

Delete a variable

You can also delete Python variables using the command **del** "variable name".

In the below example of Python delete variable, we deleted variable f, and when we proceed to print it, we get error "**variable name is not defined**" which means you have deleted the variable.



Example of Python delete variable or Python clear variable :

Python Data Structure

Python TUPLE – Pack, Unpack, Compare, Slicing, Delete, Key

Tuple Matching in Python is a method of grouping the tuples by matching the second element in the tuples. It is achieved by using a dictionary by checking the second element in each tuple in python programming. However, we can make new tuples by taking portions of existing tuples.

Tuple Syntax

Tup = ('Jan','feb','march')
To write an empty tuple, you need to write as two parentheses containing nothing-

tup1 = ();
For writing tuple for a single value, you need to include a comma, even though there is a single value. Also at the end you need to write semicolon as shown below.

Tup1 = (50,);
Tuple indices begin at 0, and they can be concatenated, sliced and so on.

Tuple Assignment

Python has tuple assignment feature which enables you to assign more than one variable at a time. In here, we have assigned tuple 1 with the persons information like name,

surname, birth year, etc. and another tuple 2 with the values in it like number (1,2,3,….,7).

For Example,

(name, surname, birth year, favorite movie and year, profession, birthplace) = Robert

Here is the code,

```
tup1 = ('Robert', 'Carlos','1965','Terminator 1995', 'Actor','Florida');
tup2 = (1,2,3,4,5,6,7);
print(tup1[0])
print(tup2[1:4])
```

- Tuple 1 includes list of information of Robert
- Tuple 2 includes list of numbers in it
- We call the value for [0] in tuple and for tuple 2 we call the value between 1 and 4
- Run the code- It gives name Robert for first tuple while for second tuple it gives number (2,3 and 4)

Packing and Unpacking

In packing, we place value into a new tuple while in unpacking we extract those values back into variables.

```
x = ("Guru99", 20, "Education")    # tuple packing
(company, emp, profile) = x    # tuple unpacking
print(company)
print(emp)
print(profile)
```

Comparing tuples

A comparison operator in Python can work with tuples.

The comparison starts with a first element of each tuple. If they do not compare to =,< or > then it proceed to the second element and so on.

It starts with comparing the first element from each of the tuples

Let’s study this with an example-

#case 1

```
a=(5,6)
b=(1,4)
if (a>b):print("a is bigger")
else: print("b is bigger")
```

#case 2

```
a=(5,6)
b=(5,4)
if (a>b):print("a is bigger")
else: print ("b is bigger")
```

#case 3

```
a=(5,6)
b=(6,4)
if (a>b):print("a is bigger")
else: print("b is bigger")
```

Case1: Comparison starts with a first element of each tuple. In this case 5>1, so the output a is bigger

Case 2: Comparison starts with a first element of each tuple. In this case 5>5 which is inconclusive. So it proceeds to the next element. 6>4, so the output a is bigger

Case 3: Comparison starts with a first element of each tuple. In this case 5>6 which is false. So it goes into the else block and prints "b is bigger."

Using tuples as keys in dictionaries

Since tuples are hashable, and list is not, we must use tuple as the key if we need to create a composite key to use in a dictionary.

Example: We would come across a composite key if we need to create a telephone directory that maps, first-name, last-name, pairs of telephone numbers, etc. Assuming that we have declared the variables as last and first number, we could write a dictionary assignment statement as shown below:

```
directory[last,first] = number
```

Inside the brackets, the expression is a tuple. We could use tuple assignment in a for loop to navigate this dictionary.

```
for last, first in directory:
print first, last, directory[last, first]
```

This loop navigates the keys in the directory, which are tuples. It assigns the elements of each tuple to last and first and then prints the name and corresponding telephone number.

Tuples and dictionary

Dictionary can return the list of tuples by calling items, where each tuple is a key value pair.

```
a = {'x':100, 'y':200}
```

```
b = list(a.items())
print(b)
```

Deleting Tuples

Tuples are immutable and cannot be deleted. You cannot delete or remove items from a tuple. But deleting tuple entirely is possible by using the keyword

del

Slicing of Tuple

To fetch specific sets of sub-elements from tuple or list, we use this unique function called slicing. Slicing is not only applicable to tuple but also for array and list.

```
x = ("a", "b","c", "d", "e")
print(x[2:4])
```
The output of this code will be (‘c’, ‘e’).

Here is the Python 2 Code for all above example

```
tup1 = ('Robert', 'Carlos','1965','Terminator 1995', 'Actor','Florida');
tup2 = (1,2,3,4,5,6,7);
print  tup1[0]
print  tup2[1:4]

#Packing and Unpacking
x = ("Guru99", 20, "Education")    # tuple packing
(company, emp, profile) = x    # tuple unpacking
print company
print emp
print profile

#Comparing tuples
#case 1
a=(5,6)
b=(1,4)
if (a>b):print "a is bigger"
else: print "b is bigger"

#case 2
a=(5,6)
b=(5,4)
if (a>b):print "a is bigger"
else: print "b is bigger"

#case 3
a=(5,6)
```

```
b=(6,4)
if (a>b):print "a is bigger"
else: print "b is bigger"

#Tuples and dictionary
a = {'x':100, 'y':200}
b = a.items()
print b

#Slicing of Tuple
x = ("a", "b","c", "d", "e")
print x[2:4]
```

Built-in functions with Tuple

To perform different task, tuple allows you to use many built-in functions like all(), any(), enumerate(), max(), min(), sorted(), len(), tuple(), etc.

Dictionary(Dict): Update, Cmp, Len, Sort, Copy, Items, str

A Dictionary in Python is the unordered and changeable collection of data values that holds key-value pairs. Each key-value pair in the dictionary maps the key to its associated value making it more optimized. A Dictionary in python is declared by enclosing a comma-separated list of key-value pairs using curly braces({}). Python Dictionary is classified into two elements: Keys and Values.

Syntax for Python Dictionary

Dict = { ' Tim': 18, xyz,.. }

Dictionary is listed in curly brackets, inside these curly brackets, keys and values are declared. Each key is separated from its value by a colon (:), while commas separate each element.

Properties of Dictionary Keys

There are two important points while using dictionary keys

- More than one entry per key is not allowed (no duplicate key is allowed)
- The values in the dictionary can be of any type, while the keys must be immutable like numbers, tuples, or strings.
- Dictionary keys are case sensitive- Same key name but with the different cases are treated as different keys in Python dictionaries.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print (Dict['Tiffany'])
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print((Dict['Tiffany']))
```

- In code, we have dictionary name "Dict"
- We declared the name and age of the person in the dictionary, where name is "Keys" and age is the"value"
- Now run the code
- It retrieves the age of tiffany from the dictionary.

Python Dictionary Methods

Copying dictionary

You can also copy the entire dictionary to a new dictionary. For example, here we have copied our original dictionary to the new dictionary name "Boys" and "Girls".

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
studentX=Boys.copy()
studentY=Girls.copy()
print studentX
print studentY
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
studentX=Boys.copy()
studentY=Girls.copy()
print(studentX)
print(studentY)
```

- We have the original dictionary (Dict) with the name and age of the boys and girls together
- But we want boys list separate from girls list, so we defined the element of boys and girls in a separate dictionary name "Boys" and "Girls."
- Now again we have created new dictionary name "student X" and "student Y," where all the keys and values of boy dictionary are copied into student X, and the girls will be copied in studentY
- So now you don't have to look into the whole list in the main dictionary(Dict) to check who is a boy and who is girl, you just have to print student X if you want boys list and StudentY if you want girls list

- So, when you run the student X and studentY dictionary, it will give all the elements present in the dictionary of “boys” and “girls” separately

Updating Dictionary

You can also update a dictionary by adding a new entry or a key-value pair to an existing entry or by deleting an existing entry. Here in the example, we will add another name, “Sarah” to our existing dictionary.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Dict.update({"Sarah":9})
print Dict
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Dict.update({"Sarah":9})
print(Dict)
```

- Our existing dictionary “Dict” does not have the name “Sarah.”
- We use the method Dict.update to add Sarah to our existing dictionary
- Now run the code, it adds Sarah to our existing dictionary

Delete Keys from the dictionary

Python dictionary gives you the liberty to delete any element from the dictionary list. Suppose you don’t want the name Charlie in the list, so you can remove the key element by the following code.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
del Dict ['Charlie']
print Dict
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
del Dict ['Charlie']
print(Dict)
```

When you run this code, it should print the dictionary list without Charlie.

- We used the code del Dict
- When code executed, it has deleted the Charlie from the main dictionary

Dictionary items() Method

The items() method returns a list of tuple pairs (Keys, Value) in the dictionary.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print "Students Name: %s" % Dict.items()
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print("Students Name: %s" % list(Dict.items()))
```

- We use the code items() method for our Dict.
- When code was executed, it returns a list of items (keys and values) from the dictionary

Check if a given key already exists in a dictionary

For a given list, you can also check whether our child dictionary exists in the main dictionary or not. Here we have two sub-dictionaries "Boys" and "Girls", now we want to check whether our dictionary Boys exist in our main "Dict" or not. For that, we use the for loop method with else if method.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
for key in Boys.keys():
   if key in Dict.keys():
      print True
   else:
      print False
```

Sorting the Dictionary

In the dictionary, you can also sort the elements. For example, if we want to print the name of the elements of our dictionary alphabetically, we have to use the for a loop. It will sort each element of the dictionary accordingly.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
Students = Dict.keys()
Students.sort()
for S in Students:
    print":".join((S,str(Dict[S])))
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
Students = list(Dict.keys())
Students.sort()
for S in Students:
    print(":".join((S,str(Dict[S]))))
```

- We declared the variable students for our dictionary “Dict.”
- Then we use the code Students.sort, which will sort the element inside our dictionary
- But to sort each element in the dictionary, we run the for a loop by declaring variable S
- Now, when we execute the code, the for loop will call each element from the dictionary, and it will print the string and value in an order

Python Dictionary in-built Functions

Dictionary len() Method

The len() function gives the number of pairs in the dictionary.

For example,

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print "Length : %d" % len (Dict)
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print("Length : %d" % len (Dict))
```

When len (Dict) function is executed it gives the output at “4” as there are four elements in our dictionary

Variable Types

Python does not require to explicitly declare the reserve memory space; it happens automatically. The assign values to variable “=” equal sign are used. The code to determine the variable type is ” %type (Dict).”

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print "variable Type: %s" %type (Dict)
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print("variable Type: %s" %type (Dict))
```

- Use the code %type to know the variable type
- When code was executed, it tells a variable type is a dictionary

Python List cmp() Method

The compare method cmp() is used in Python to compare values and keys of two dictionaries. If method returns 0 if both dictionaries are equal, 1 if dic1 > dict2 and -1 if dict1 < dict2.

Python 2 Example

```
Boys = {'Tim': 18,'Charlie':12,'Robert':25}
Girls = {'Tiffany':22}
print cmp(Girls, Boys)
```

Python 3 Example

cmp is not supported in Python 3

- We have two dictionary name, “Boys” and “Girls.”
- Whichever you declare first in code “cmp(Girls, Boys)” will be considered as dictionary 1. In our case, we declared “Girls” first, so it will be considered as dictionary 1 and boys as dictionary 2
- When code is executed, it prints out -1, It indicates that our dictionary 1 is less than dictionary 2.

Dictionary Str(dict)

With Str() method, you can make a dictionary into a printable string format.

Python 2 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print "printable string:%s" % str (Dict)
```

Python 3 Example

```
Dict = {'Tim': 18,'Charlie':12,'Tiffany':22,'Robert':25}
print("printable string:%s" % str (Dict))
```

- Use the code % str (Dict)
- It will return the dictionary elements into a printable string format

Here is the list of all Dictionary Methods

Method	Description	Syntax
copy()	Copy the entire dictionary to new dictionary	dict.copy()
update()	Update a dictionary by adding a new entry or a key-value pair to anexisting entry or by deleting an existing entry.	Dict.update([other])
items()	Returns a list of tuple pairs (Keys, Value) in the dictionary.	dictionary.items()
sort()	You can sort the elements	dictionary.sort()
len()	Gives the number of pairs in the dictionary.	len(dict)
cmp()	Compare the values and keys of two dictionaries	cmp(dict1, dict2)
Str()	Make a dictionary into a printable string format	Str(dict)

Arrays: Create, Reverse, Pop with Python Array Examples

A **Python Array** is a collection of common type of data structures having elements with same data type. It is used to store collections of data. In Python programming, an arrays are handled by the "array" module. If you create arrays using the array module, elements of the array must be of the same numeric type.

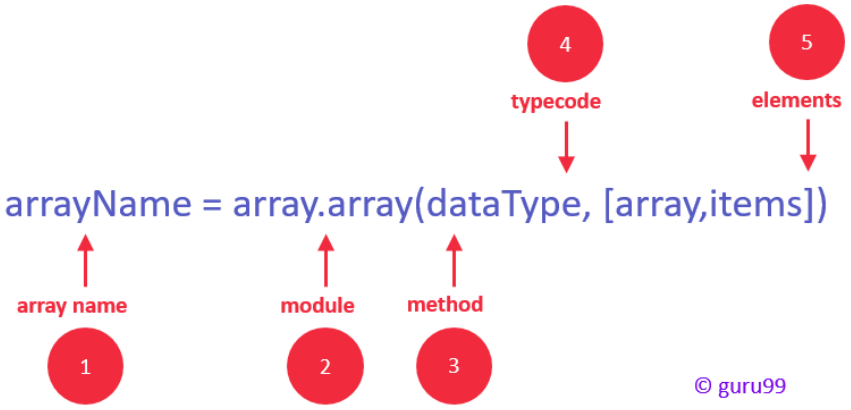
Python arrays are used when you need to use many variables which are of the same type. It can also be used to store a collection of data. The arrays are especially useful when you have to process the data dynamically. Python arrays are much faster than list as it uses less memory.

Syntax to Create an Array in Python

You can declare an array in Python while initializing it using the following syntax.

arrayName = array.array(type code for data type, [array,items])

The following image explains the syntax.



Array Syntax

1. **Identifier**: specify a name like usually, you do for variables
2. **Module**: Python has a special module for creating array in Python, called "array" – you must import it before using it
3. **Method**: the array module has a method for initializing the array. It takes two arguments, type code, and elements.
4. **Type Code**: specify the data type using the type codes available (see list below)
5. **Elements**: specify the array elements within the square brackets, for example [130,450,103]

How to create arrays in Python?

In Python, we use following syntax to create arrays:

Class array.array(type code[,initializer])

For Example

```
import array as myarray
abc = myarray.array('d', [2.5, 4.9, 6.7])
```

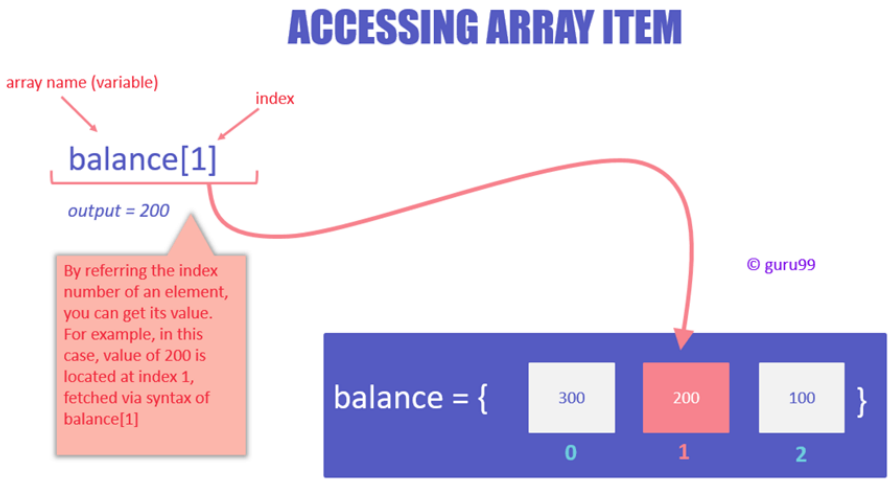
The above code creates an array having integer type. The letter ‘d’ is a type code.
Following tables show the type codes:

Type code	Python type	C Type	Min size(bytes)
‘u’	Unicode character	Py_UNICODE	2
‘b’	Int	Signed char	1
‘B’	Int	Unsigned char	1
‘h’	Int	Signed short	2
‘l’	Int	Signed long	4
‘L’	Int	Unsigned long	4
‘q’	Int	Signed long long	8
‘Q’	Int	Unsigned long long	8
‘H’	Int	Unsigned short	2
‘f’	Float	Float	4
‘d’	Float	Double	8
‘i’	Int	Signed int	2
‘I’	Int	Unsigned int	2

How to access array elements?
You can access any array item by using its index.
The syntax is
arrayName[indexNum]
For example,

```
import array
balance = array.array('i', [300,200,100])
print(balance[1])
```

Output:
200
The following image illustrates the basic concept of accessing arrays items by their index.



Accessing Array Item

Here, we have accessed the second value of the array using its index, which is 1. The output of this will be 200, which is basically the second value of the balanced array.

The array index starts with 0. You can also access the last element of an array using the -1 index.

Example:

```
import array as myarray
abc = myarray.array('d', [2.5, 4.9, 6.7])
print("Array first element is:",abc[0])
print("Array last element is:",abc[-1])
```

Output:

```
Array first element is: 2.5
Array last element is: 6.7
```

You can also access elements by using the ':' operator as shown in below Python arrays examples.

Example:

```
import array as myarray
abc= myarray.array('q',[3,9,6,5,20,13,19,22,30,25])
print(abc[1:4])
print(abc[7:10])
```

Output:

```
array('q', [9, 6, 5])
array('q', [22, 30, 25])
```

This operation is called a **slicing** operation.

How to insert elements?

Python array insert operation enables you to insert one or more items into an array at the beginning, end, or any given index of the array. This method expects two arguments index and value.

The syntax is

```
arrayName.insert(index, value)
```

Example:

Let's add a new value right after the second item of the array. Currently, our balance array has three items 300, 200, and 100. Consider the second array item with a value of 200 and index 1.

In order to insert the new value right "after" index 1, you need to reference index 2 in your insert method, as shown in the below Python array example:

```
import array
balance = array.array('i', [300,200,100])
balance.insert(2, 150)
print(balance)
```

Output:

```
array('i', [300,200,150,100])
```

Example 2:

```
import array as myarr
a=myarr.array('b',[2,4,6,8,10,12,14,16,18,20])
```

```
a.insert(2,56)
print(a)
```
Output:
```
array('b', [2, 4, 56, 6, 8, 10, 12, 14, 16, 18, 20])
```
How to modify elements?

Arrays in Python are mutable. They can be modified by the following syntax:

```
Object_name[index]=value;
```
Example:
```
import array as myarr
a=myarr.array('b',[3,6,4,8,10,12,14,16,18,20])
a[0]=99
print(a)
```
Output:
```
array('b', [99, 6, 4, 8, 10, 12, 14, 16, 18, 20])
```
We can also perform concatenation operations on arrays in Python.

Example:
```
import array as myarr
first = myarr.array('b', [4, 6, 8])
second = myarr.array('b', [9, 12, 15])
numbers = myarr.array('b')
numbers = first + second
print(numbers)
```
Output:
```
array('b', [4, 6, 8, 9, 12, 15])
```
The above Python array example code concates two variables called “first” and “second”. The result is stored in a variable called “number”.

The last line of code is used to print two arrays.

How to pop an element from Array in Python?

In Python, a developer can use pop() method to pop and element from Python array. Below is an example of pop() method in Python.

Python array pop Example:
```
import array as myarr
first = myarr.array('b', [20, 25, 30])
first.pop(2)
print(first)
```
Output:
```
array('b', [20, 25])
```
You can also use the ‘del’ statement of Python.

Example
```
import array as myarr
no = myarr.array('b', [10, 4, 5, 5, 7])
del no[4]
print(no)
```
Output:
```
array('b', [10, 4, 5, 5])
```

How to delete elements?

With this operation, you can delete one item from an array by value. This method accepts only one argument, value. After running this method, the array items are re-arranged, and indices are re-assigned.

The syntax is

arrayName.remove(value)

Example:

Let's remove the value of "3" from the array

```
import array as myarray
first = myarray.array('b', [2, 3, 4])
first.remove(3)
print(first)
```

Output:

```
array('b', [2, 4])
```

How to Search and get the index of a value in an Array

With this operation, you can search for an item in an array based on its value. This method accepts only one argument, value. It is a non-destructive method, which means it does not affect the array values.

The syntax is

arrayName.index(value)

Example:

Let's find the value of "3" in the array. This method returns the index of the searched value.

```
import array as myarray
number = myarray.array('b', [2, 3, 4, 5, 6])
print(number.index(3))
```

Output:

```
1
```

This operation will return the index of the first occurrence of the mentioned element.

How to Reverse an Array in Python

This operation will reverse the entire array.

Syntax: array.reverse()

```
import array as myarray
number = myarray.array('b', [1,2, 3])
number.reverse()
print(number)
```

Output:

```
array('b', [3, 2, 1])
```

Questions:

1. List the features of Python?
2. Explain the data structure in Python?

Conclusion : Thus, I have studied installation of Python and study of basics.

EXPERIMENT NO. 02

Title: Implement python program that loads any dataset and plot the graph.

Outcome: Students will be able to implement python program that loads any dataset and plot the graph.

Theory:

Data visualization is the presentation of data in an accessible manner through visual tools like graphs or charts. These visualizations aid the process of communicating insights and relationships within the data, and are an essential part of data analysis. In this article, we treat **Matplotlib**, which is the most popular data visualization library within the Python programming language.

1. Preliminaries

Matplotlib is a very well documented package. To make the plotting easier, we make use of the **pyplot** module, that makes Matplotlib work like MATLAB. Essentially, all its functioning can be found HERE. The point of this article is to state its main and most important functions and give examples on how to use pyplot, as the documentation sometimes is pretty hard to navigate through.

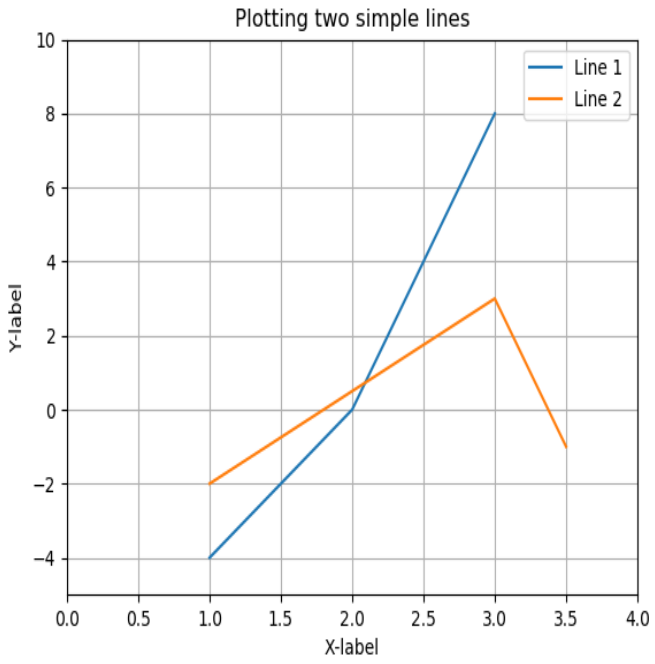
To call the package module, we begin our code with `import matplotlib.pyplot as plt`. Below, we state some of the most important functions when using pyplot:

- plt.title: Set a title, which appears above the plot.
- plt.grid: Configure the grid lines in the figure. To enable grid lines in the plot, use plt.grid(True).
- plt.legend: Place a legend in the figure.
- plt.xlabel and plt.ylabel: Set labels for the axes. For example, plt.xlabel("Age") sets "Age" as the label for the x-axis.
- plt.xlim and plt.ylim: Set the limit ranges for the axes. So, e.g., plt.xlim([-50, 50]) limits the plot to show only x-values between -50 and 50.
- plt.show: Use at the end to display everything in the plot.

To demonstrate the usage of these functions, let us consider a simple example where we want to draw two simple lines. This can be achieved by using the plt.plot function. By using the following code

```
import matplotlib.pyplot as pltx = [1, 2, 3]
x2 = [1, 3, 3.5]
y = [-4, 0, 8]
y2 = [-2, 3, -1]

plt.plot(x, y, label='Line 1')
plt.plot(x2, y2, label='Line 2')

plt.title("Plotting two simple lines")
plt.grid(True)
plt.xlabel("X-label")
plt.ylabel("Y-label")
plt.xlim([0, 4])
plt.ylim([-5, 10])
plt.legend()
plt.show()
```

we obtain the following figure:



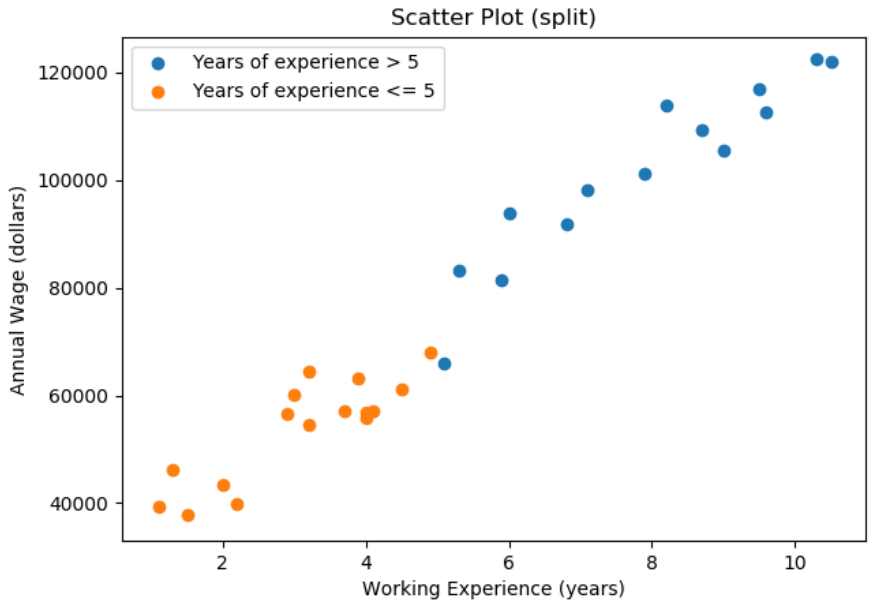
2. Scatter Plots

Now that we have seen the basic pyplot functions, we will start with our first type of plot. The **scatter plot** displays values for, typically, two variables for a set of data. Such kind of plots could be very informative when figuring out relationships between pairs of variables. Consider the following Salary dataset (from Kaggle), which contains 30 observations consisting of years of working experience and the annual wage (in dollars). To create a scatter plot, we make use of the plt.scatter function. Then, we can plot these data points as follows:

```
import matplotlib.pyplot as plt
import pandas as pddata = pd.read_csv("Salary_data.csv")  # load dataset
X = data["YearsExperience"]
Y = data["Salary"]plt.scatter(X, Y)
plt.title("Scatter Plot")
plt.xlabel("Working Experience (years)")
plt.ylabel("Annual Wage (dollars)")
plt.show()
```

We are also able to, for example, distinguish between observations that have more than 5 years of working experience and observations that have less than 5 years of working experience by using different colors. To do this, we create two scatter plots by using the relevant data splits and display them in one single plot. The following code results in the desired plot:

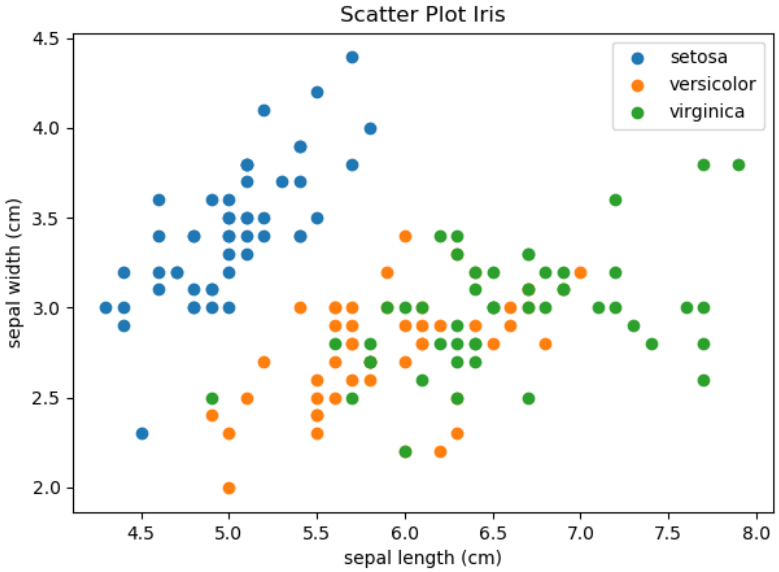
```
X_1 = X[X > 5]
X_2 = X[X <= 5]
Y_1 = Y[X > 5]
Y_2 = Y[X <= 5]plt.scatter(X_1, Y_1, label='Years of experience > 5')
plt.scatter(X_2, Y_2, label='Years of experience <= 5')
plt.title("Scatter Plot (split)")
plt.legend()
plt.xlabel("Working Experience (years)")
plt.ylabel("Annual Wage (dollars)")
plt.show()
```



We are also able to make use of scatter plots within **classification** applications. When we are dealing with data where observations belong to different classes, we can plot a data point and color it accordingly to the class it belongs to. To give an example, we make use of the Iris dataset which is available through the sklearn package in Python. Within this dataset, observations could belong to three different flower (iris) classes. In this example, we consider only two features so that the picture is 2D. These two features of interest are the sepal length and sepal width (in cm). Then, by using the following code, we can make a nice plot for the different observations belonging to different iris classes:

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as npiris = datasets.load_iris()  # load dataset
X_iris = iris.data[:, :2]  # only take the first two features
```

```
Y_iris = iris.target
n_classes = 3for i in range(n_classes):
   index = np.where(Y_iris == i)
   plt.scatter(X_iris[index, 0], X_iris[index, 1],
   label=iris.target_names[i])
plt.legend()
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.show
```



3. Bar Charts

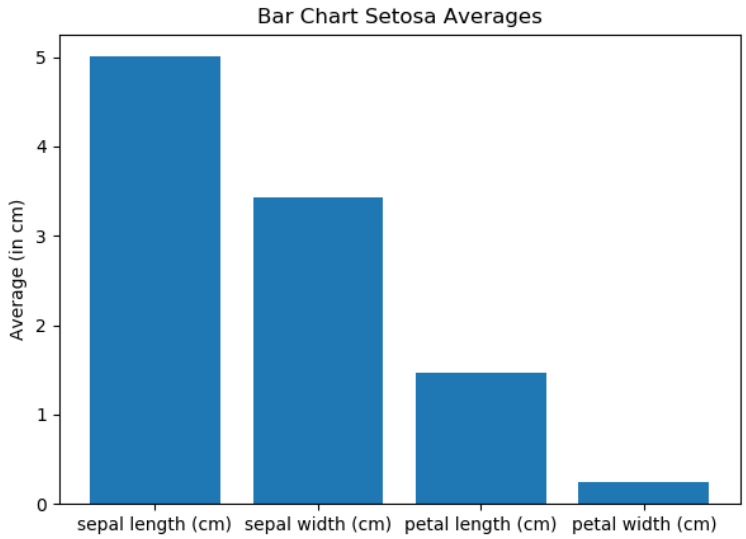
A **bar chart** graphically displays categorical data with rectangular bars of different heights, where the heights or lengths of the bars represent the values of the corresponding measure**.**

Let us once again consider the Iris dataset, where observations belong to either one of three iris flower classes. Assume we want to visualize the average value for each feature of the Setosa iris class. We can do this by using a bar chart, requiring the plt.bar function.

The following code results in the desired bar chart figure:

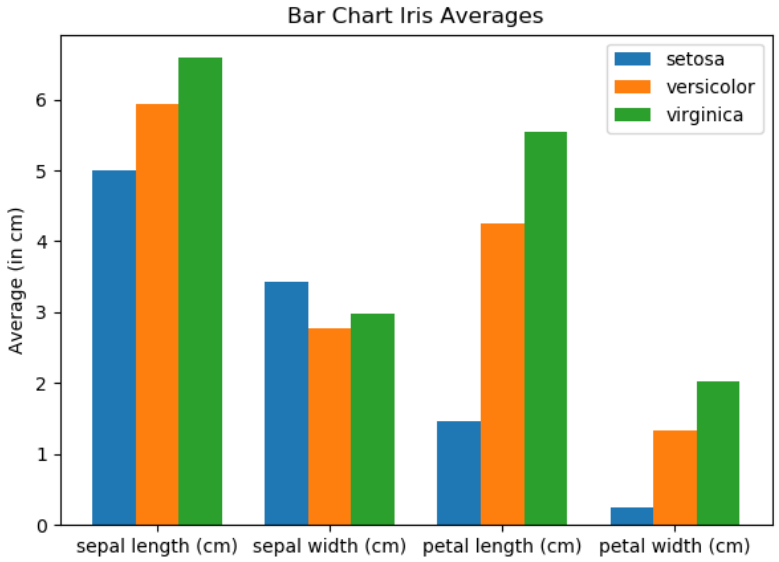
```
from sklearn import datasets
import matplotlib.pyplot as pltiris = datasets.load_iris()
X_iris = iris.data
Y_iris = iris.targetaverage = X_iris[Y_iris == 0].mean(axis=0)

plt.bar(iris.feature_names, average)
```

```
plt.title("Bar Chart Setosa Averages")
plt.ylabel("Average (in cm)")
plt.show()
```



Furthermore, we are also able to nicely display the feature averages for all three iris flowers, by placing the bars next to each other. This takes a bit more effort than the standard bar chart. By using the following code, we obtain the desired plot:

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as npiris = datasets.load_iris()
X_iris = iris.data
Y_iris = iris.target
n_classes = 3averages = [X_iris[Y_iris == i].mean(axis=0) for i in range(n_classes)]
x = np.arange(len(iris.feature_names))

fig = plt.figure()
ax = fig.add_subplot()
bar1 = ax.bar(x - 0.25, averages[0], 0.25, label=iris.target_names[0])
bar2 = ax.bar(x, averages[1], 0.25, label=iris.target_names[1])
bar3 = ax.bar(x + 0.25, averages[2], 0.25, label=iris.target_names[2])
ax.set_xticks(x)
ax.set_xticklabels(iris.feature_names)

plt.legend()
plt.title("Bar Chart Iris Averages")
plt.ylabel("Average")
plt.show
```

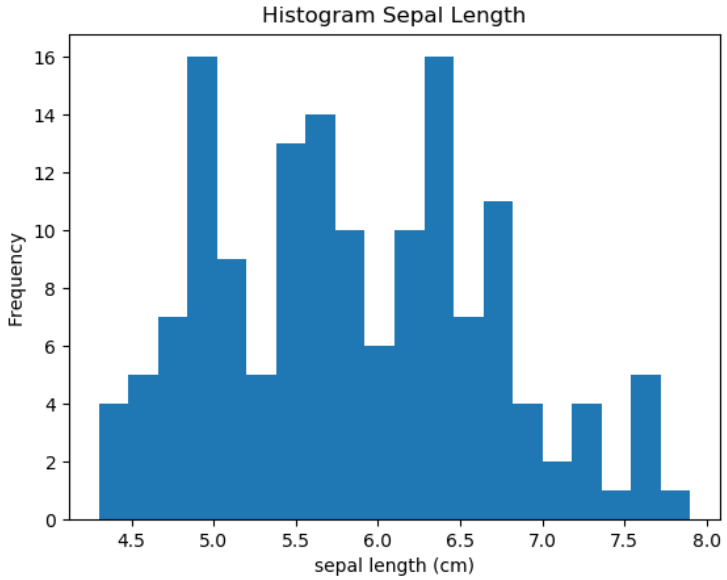


4. Histograms

A **histogram** is used to give an approximate representation of the distribution of the data, based on the sample data at hand. A histogram is constructed by using equally sized 'bins' (intervals), and counting the number of data points that belong to each bin. Creating histograms at the start of a new project is very useful to get familiar with the data, and to get a rough sense of the density of the underlying distribution. To create a histogram, we make use of the plt.hist function.

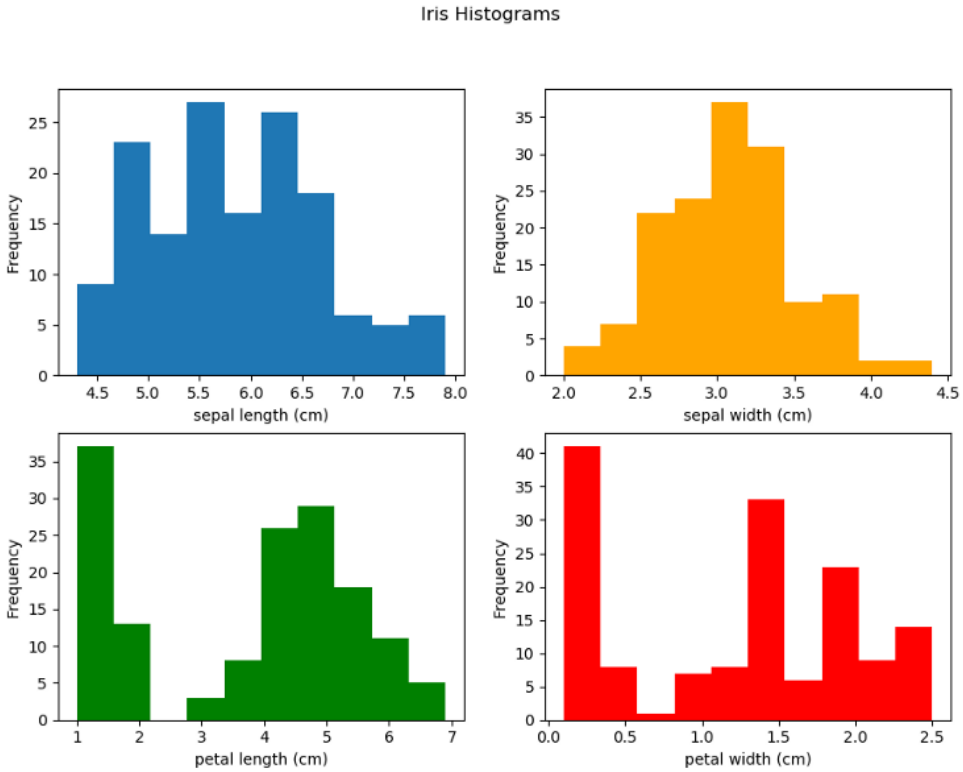
To create a basic histogram on the sepal length of all iris flowers, using 20 equal-length bins, we use the following code:

```
from sklearn import datasets
import matplotlib.pyplot as pltbins = 20
iris = datasets.load_iris()
X_iris = iris.data
X_sepal = X_iris[:, 0]

plt.hist(X_sepal, bins)
plt.title("Histogram Sepal Length")
plt.xlabel(iris.feature_names[0])
plt.ylabel("Frequency")
plt.show
```



Instead of plotting the histogram for a single feature, we can plot the histograms for all features. This can be done by creating separate plots, but here, we will make use of subplots, so that all histograms are shown in one single plot. For this, we make use of the plt.subplots function. By using the following code, we obtain the plot containing the four histograms:

```
from sklearn import datasets
import matplotlib.pyplot as pltbins = 20
iris = datasets.load_iris()
X_iris = iris.datafig, axs = plt.subplots(2, 2)
axs[0, 0].hist(X_iris[:, 0])
axs[0, 1].hist(X_iris[:, 1], color='orange')
axs[1, 0].hist(X_iris[:, 2], color='green')
axs[1, 1].hist(X_iris[:, 3], color='red')

i = 0
for ax in axs.flat:
    ax.set(xlabel=iris.feature_names[i], ylabel='Frequency')
    i += 1

fig.suptitle("Iris Histograms")
```



Visualize data from CSV file in Python

CSV stands for '**Comma-Separated Values**'. It means the data(values) in a CSV file are separated by a delimiter i.e., comma. Data in a CSV file is stored in tabular format with an extension of .csv. Generally, CSV files are used with Google spreadsheets or Microsoft Excel sheets. A CSV file contains a number of records with the data spread across rows and columns. In this article, we are going to visualize data from a CSV file in Python.

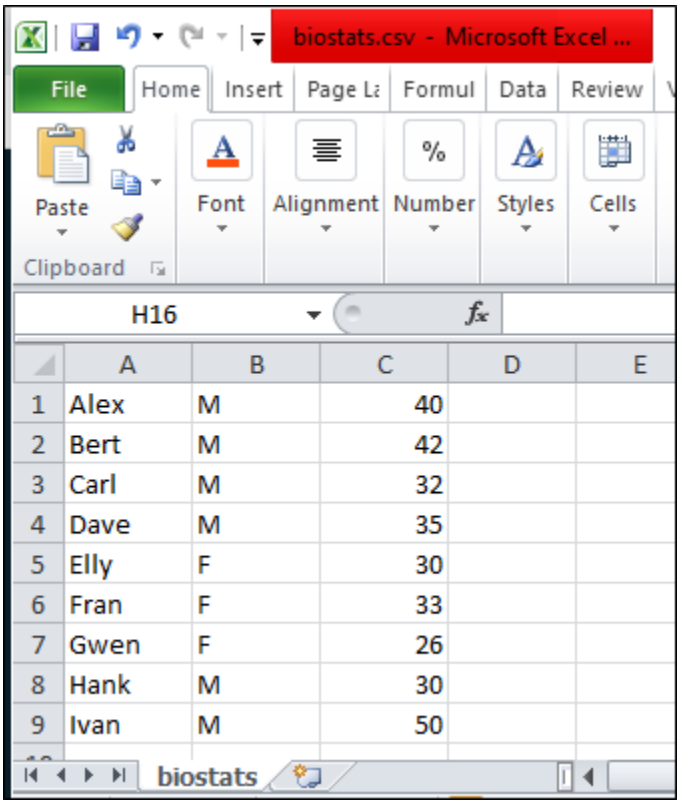
To extract the data in CSV file, CSV module must be imported in our program as follows:

```
import csv


with open('file.csv') as File:
    Line_reader = csv.reader(File)
```

Here, csv.reader( ) function is used to read the program after importing CSV library.

Example 1: Visualizing the column of different persons through bar plot.

The below CSV file contains different person name, gender, and age saved as 'biostats.csv':



The approach of the program:

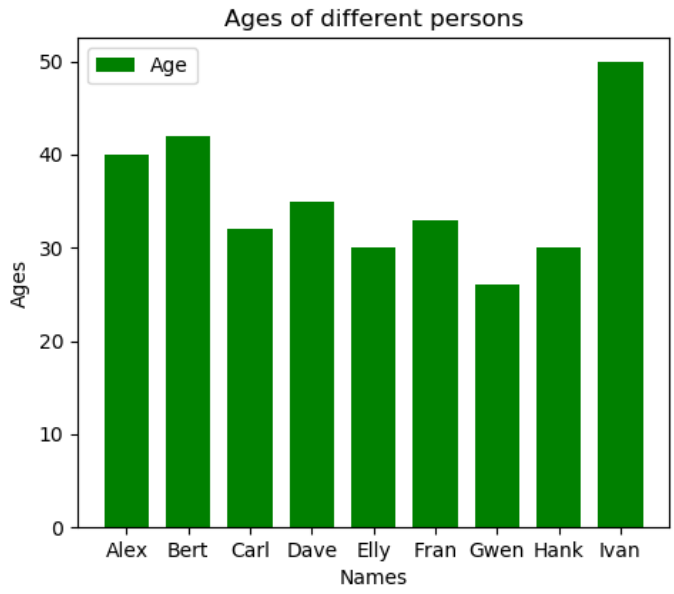
1. Import required libraries, matplotlib library for visualizing, and CSV library for reading CSV data.
2. Open the file using open( ) function with 'r' mode (read-only) from CSV library and read the file using csv.reader( ) function.
3. Read each line in the file using for loop.
4. Append required columns into a list.
5. After reading the whole CSV file, plot the required data as X and Y axis.
6. In this example, we are plotting names as X-axis and ages as Y-axis.

Below is the implementation:

```
import matplotlib.pyplot as plt
import csv

x = []
y = []

with open('biostats.csv','r') as csvfile:
        plots = csv.reader(csvfile, delimiter = ',')

        for row in plots:
                x.append(row[0])
                y.append(int(row[2]))
```

```
plt.bar(x, y, color = 'g', width = 0.72, label = "Age")
plt.xlabel('Names')
plt.ylabel('Ages')
plt.title('Ages of different persons')
plt.legend()
plt.show()
```

Output :



Questions:

1. Explain Data visualization in Python?
2. How to plot different Graphs in Python?

Conclusion : Thus, I have implemented python program that loads any dataset and plot the graph.

EXPERIMENT NO. 03

Title: Implement python program that perform data cleaning on any dataset.

Outcome: Students will be able to implement python program that perform data cleaning on any dataset.

Theory:

Python is an easy-to-learn programming language, which makes it the most preferred choice for beginners in Data Science, Data Analytics, and Machine Learning. It also has a great community of online learners and excellent data-centric libraries.

With so much data being generated, it becomes important that the data we use for Data Science applications like Machine Learning and Predictive Modeling is clean. But what do we mean by clean data? And what makes data dirty in the first place?

Dirty data simply means data that is erroneous. Duplicacy of records, incomplete or outdated data, and improper parsing can make data dirty. This data needs to be cleaned. Data cleaning (or data cleansing) refers to the process of "cleaning" this dirty data, by identifying errors in the data and then rectifying them.

Data cleaning is an important step in and Machine Learning project, and we will cover some basic data cleaning techniques (in Python) in this article.

Cleaning Data in Python

We will learn more about data cleaning in Python with the help of a sample dataset. We will use the Russian housing dataset on Kaggle.

We will start by importing the required libraries.

```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Download the data, and then read it into a Pandas DataFrame by using the read_csv() function, and specifying the file path. Then use the shape attribute to check the number of rows and columns in the dataset. The code for this is as below:

```
df = pd.read_csv('housing_data.csv')
df.shape
```

The dataset has 30,471 rows and 292 columns.

We will now separate the numeric columns from the categorical columns.

```
# select numerical columns
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
# select non-numeric columns
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
```

We are now through with the preliminary steps. We can now move on to data cleaning. We will start by identifying columns that contain missing values and try to fix them.

Missing values

We will start by calculating the percentage of values missing in each column, and then storing this information in a DataFrame.

```
# % of values missing in each column
values_list = list()
cols_list = list()
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())*100
    cols_list.append(col)
    values_list.append(pct_missing)
```

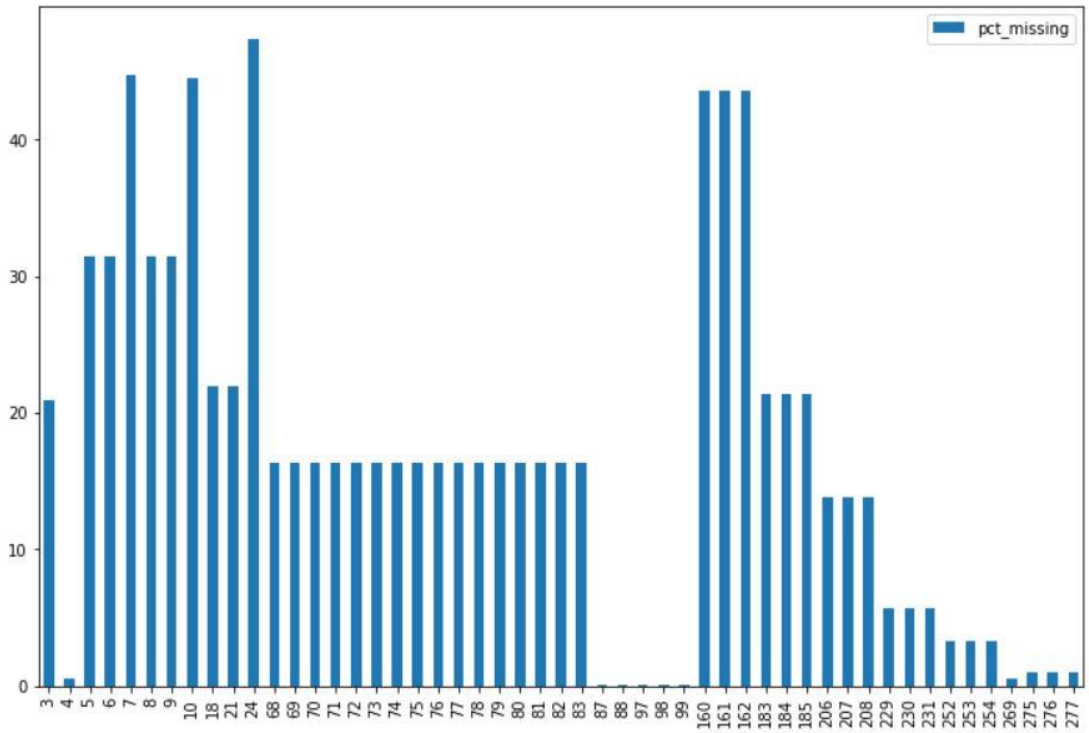
```
pct_missing_df = pd.DataFrame()
pct_missing_df['col'] = cols_list
pct_missing_df['pct_missing'] = values_list
```

The DataFrame pct_missing_df now contains the percentage of missing values in each column along with the column names.

We can also create a visual out of this information for better understanding using the code below:

```
pct_missing_df.loc[pct_missing_df.pct_missing > 0].plot(kind='bar', figsize=(12,8))
plt.show()
```

The output after execution of the above line of code should look like this:



It is clear that some columns have very few values missing, while other columns have a substantial % of values missing. We will now fix these missing values.

There are a number of ways in which we can fix these missing values. Some of them are"

Drop observations

One way could be to drop those observations that contain any null value in them for any of the columns. This will work when the percentage of missing values in each column is very less. We will drop observations that contain null in those columns that have less than 0.5% nulls. These columns would be metro_min_walk, metro_km_walk, railroad_station_walk_km, railroad_station_walk_min, and ID_railroad_station_walk.

```
less_missing_values_cols_list = list(pct_missing_df.loc[(pct_missing_df.pct_missing < 0.5) & (pct_missing_df.pct_missing > 0), 'col'].values)
df.dropna(subset=less_missing_values_cols_list, inplace=True)
```

This will reduce the number of records in our dataset to 30,446 records.

Remove columns (features)

Another way to tackle missing values in a dataset would be to drop those columns or features that have a significant percentage of values missing. Such columns don't contain a lot of information and can be dropped altogether from the dataset. In our case, let us drop all those columns that have more than 40% values missing in them. These columns would be build_year, state, hospital_beds_raion, cafe_sum_500_min_price_avg, cafe_sum_500_max_price_avg, and cafe_avg_price_500.

```
# dropping columns with more than 40% null values
_40_pct_missing_cols_list = list(pct_missing_df.loc[pct_missing_df.pct_missing > 40, 'col'].values)
df.drop(columns=_40_pct_missing_cols_list, inplace=True)
```

The number of features in our dataset is now 286.

Impute missing values

There is still missing data left in our dataset. We will now impute the missing values in each numerical column with the median value of that column.

```
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
for col in numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:  # impute values only for columns that have missing values
        med = df[col].median() #impute with the median
        df[col] = df[col].fillna(med)
```

Missing values in numerical columns are now fixed. In the case of categorical columns, we will replace missing values with the mode values of that column.

```
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
for col in non_numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:  # impute values only for columns that have missing values
        mod = df[col].describe()['top'] # impute with the most frequently occuring value
        df[col] = df[col].fillna(mod)
```

All missing values in our dataset have now been treated. We can verify this by running the following piece of code:

```
df.isnull().sum().sum()
```

If the output is zero, it means that there are no missing values left in our dataset now.

We can also replace missing values with a particular value (like -9999 or 'missing') which will indicate the fact that the data was missing in this place. This can be a substitute for missing value imputation.

Outliers

An outlier is an unusual observation that lies away from the majority of the data. Outliers can affect the performance of a Machine Learning model significantly. Hence, it becomes important to identify outliers and treat them.

Let us take the 'life_sq' column as an example. We will first use the describe() method to look at the descriptive statistics and see if we can gather any information from it.

```
df.life_sq.describe()
```

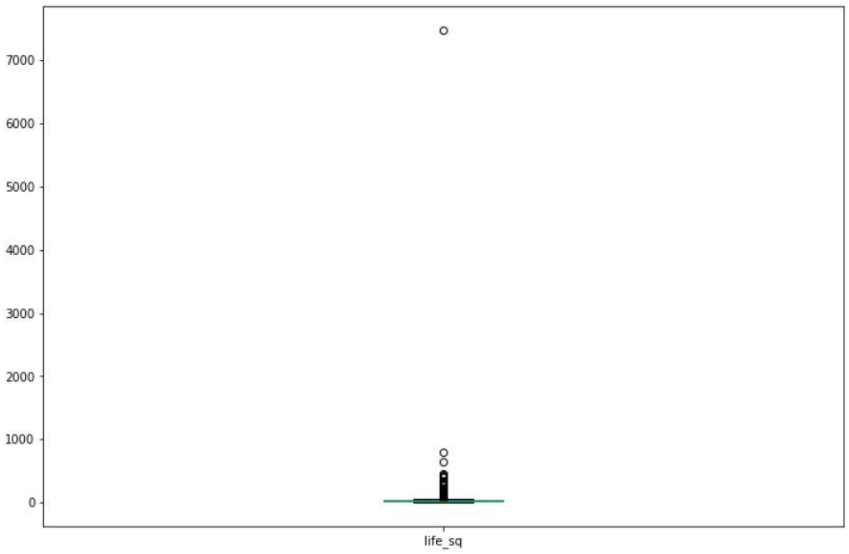
The output will look like this:

```
count    30446.000000
mean        33.482658
std         46.538609
min          0.000000
25%         22.000000
50%         30.000000
75%         38.000000
max       7478.000000
Name: life_sq, dtype: float64
```

From the output, it is clear that something is not correct. The max value seems to be abnormally large compared to the mean and median values. Let us make a boxplot of this data to get a better idea.

```
df.life_sq.plot(kind='box', figsize=(12, 8))
plt.show()
```

The output will look like this:



It is clear from the boxplot that the observation corresponding to the maximum value (7478) is an outlier in this data. Descriptive statistics, boxplots, and scatter plots help us in identifying outliers in the data.

We can deal with outliers just like we dealt with missing values. We can either drop the observations that we think are outliers, or we can replace the outliers with suitable values, or we can perform some sort of transformation on the data (like log or exponential). In our case, let us drop the record where the value of 'life_sq' is 7478.

```
# removing the outlier value in life_sq column
df = df.loc[df.life_sq < 7478]
```

Duplicate records

Data can sometimes contain duplicate values. It is important to remove duplicate records from your dataset before you proceed with any Machine Learning project. In our data,

since the ID column is a unique identifier, we will drop duplicate records by considering all but the ID column.

```
# dropping duplicates by considering all columns other than ID
cols_other_than_id = list(df.columns)[1:]
df.drop_duplicates(subset=cols_other_than_id, inplace=True)
```

This will help us in dropping the duplicate records. By using the shape method, you can check that duplicate records have actually been dropped. The number of observations is 30,434 now.

Fixing data type

Often in the dataset, values are not stored in the correct data type. This can create a problem in later stages, and we may not get the desired output or may get errors while execution. One common data type error is with dates. Dates are often parsed as objects in Python. There is a separate data type for dates in Pandas, called DateTime.

We will first check the data type of the timestamp column in our data.

```
df.timestamp.dtype
```

This returns the data type ‘object’. We now know the timestamp is not stored correctly.

To fix this, let’s convert the timestamp column to the DateTime format.

```
# converting timestamp to datetime format
df['timestamp'] = pd.to_datetime(df.timestamp, format='%Y-%m-%d')
```

Questions:

1. Explain features of Panda Library?
2. Explain the need of data cleaning?

Conclusion : Thus, I have implemented python program that perform data cleaning on any dataset.

EXPERIMENT NO. 04

Title: Installation of R and study of R objects with basic statistics.

Outcome: Students will be able to install R and study of R objects with basic statistics.

Theory:

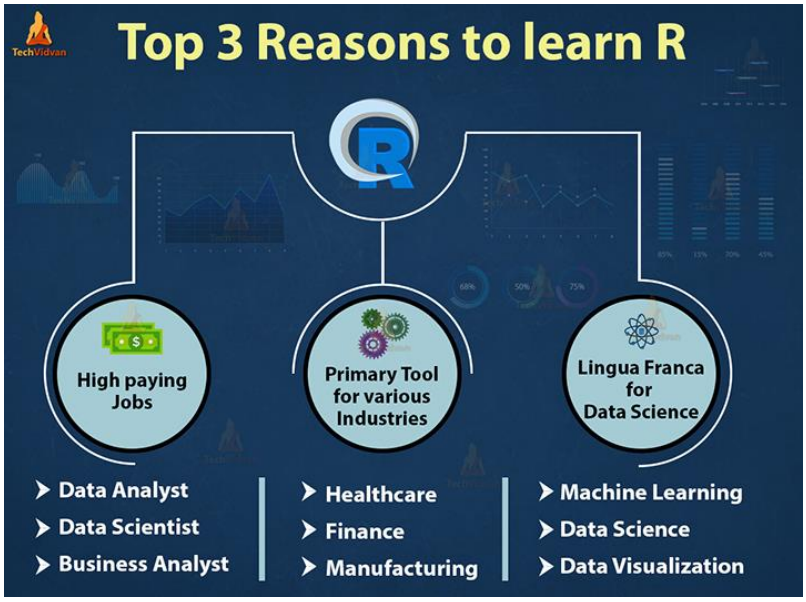
R is an **open-source** programming language and environment used for statistical analysis, data visualization, and data science.
Being open-source, R has a **massive community** that continuously works to improve the environment as well as helps members worldwide to improve and innovate.
R can be used for data analytics, statistical analysis, as well as **machine learning** purposes.
R is compatible with a number of different technologies and is highly flexible.

It has over **10,000** different **libraries** and packages to enhance and add on to its already significant capabilities. It also has graphics libraries for static as well as dynamic graphics.

History of R

R is an extension of the S-programming language, which was created by John Chambers at Bell Laboratories (formerly AT&T, now Lucent Technologies) in 1976. S was a premiere tool for statistical research, but it wasn't very feasible outside scholarly research.

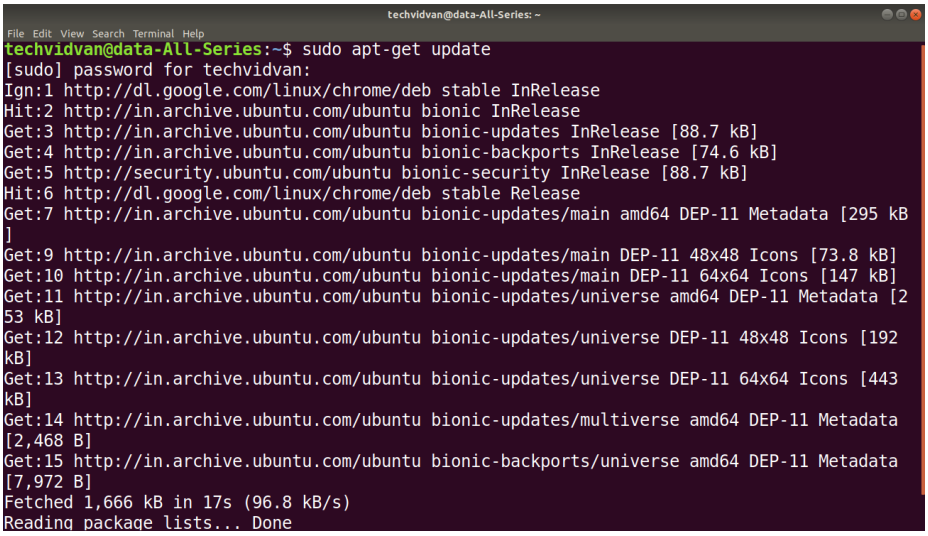
In 1992, **Ross Ihaka** and **Robert Gentleman** created R at the University of Auckland, New Zealand, as a tool that their students could learn and use easily. Ihaka and Gentleman released the initial version in 1995, and a stable beta version was released in 2000. Since then, it is maintained by the R Development Core Team.



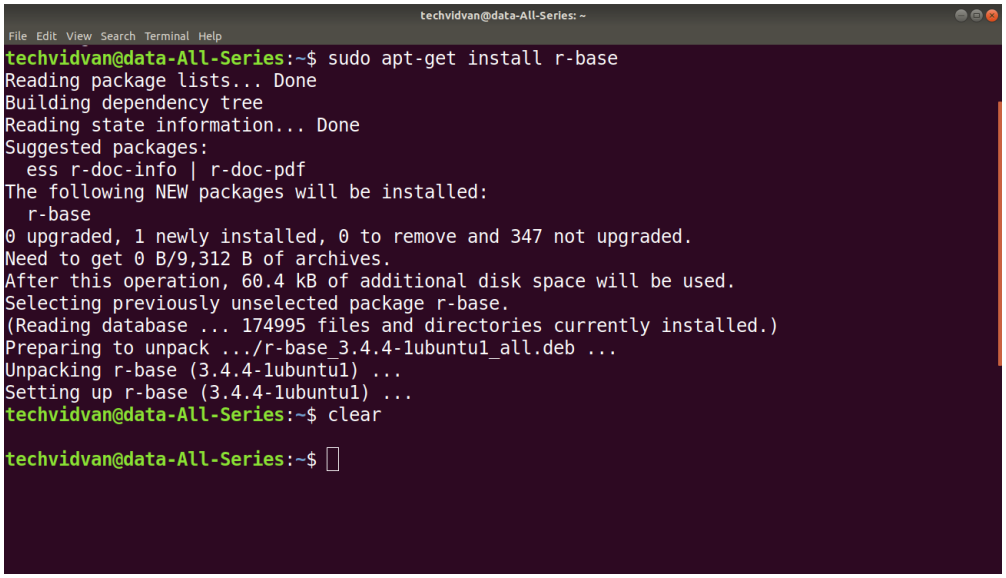
Installing R and RStudio on Linux

Linux software is often distributed as source code and then compiled by package managers like apt or yum. To install R in Ubuntu, we will have to go through the following steps.

Install the **R-base package** using the following code
sudo apt-get update



sudo apt-get install r-base



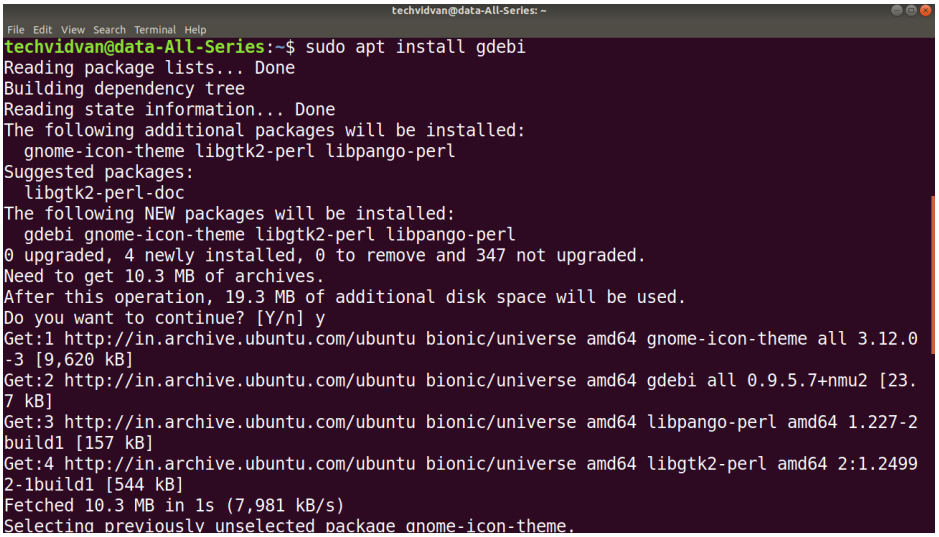
After running the command, a confirmation prompt will appear. Answer it with a '**Y**' for yes.

Install RStudio on Linux

Step – 1 Next comes installing RStudio. To install RStudio, go to download RStudio, click on the download button for RStudio desktop, click the link for the latest R version for your OS and save the .deb file.

Step – 2 Download and install the **gdebi** package using the following commands

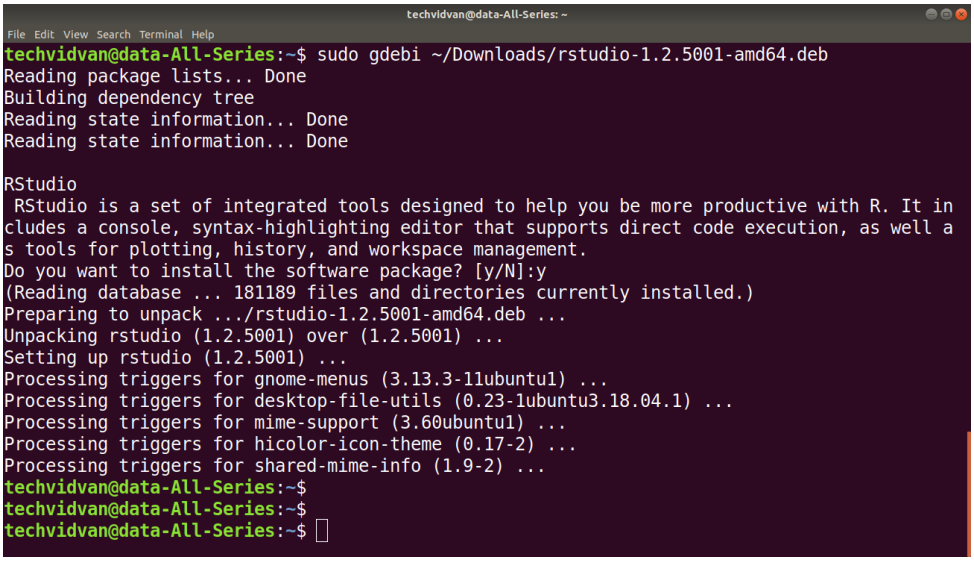
```
sudo apt install gdebi
```



Answer with a '**Y**' for yes to confirm when prompted.

Step – 3 Use the following commands to install the .deb package

```
sudo gdebi /path/to/the/file/.deb
```

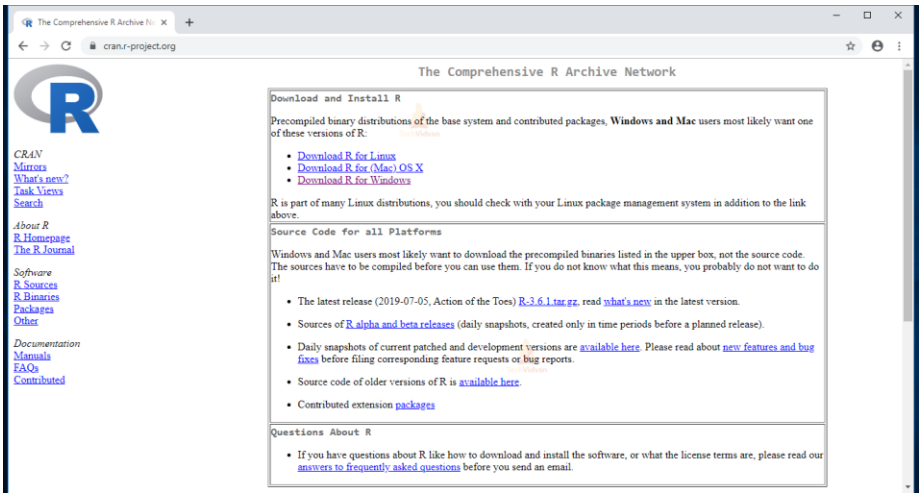


Installing R and RStudio on Windows

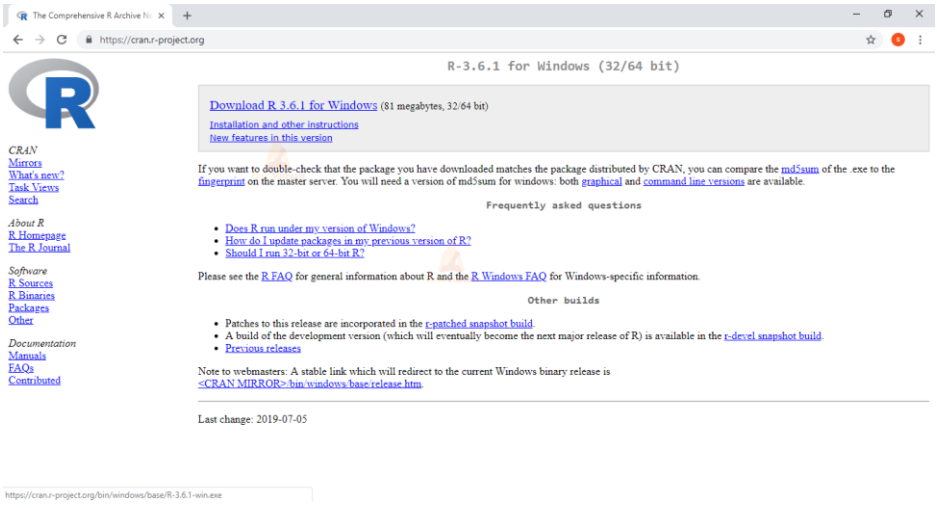
To install R and RStudio on windows, go through the following steps:

Install R on windows

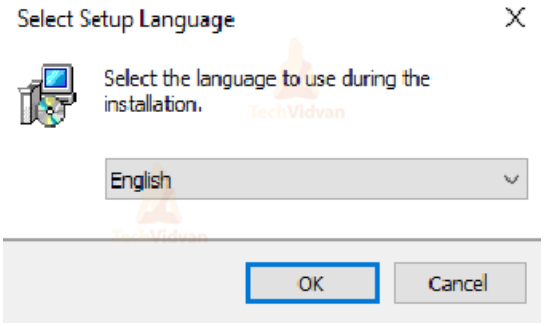
Step – 1: Go to CRAN R project website.



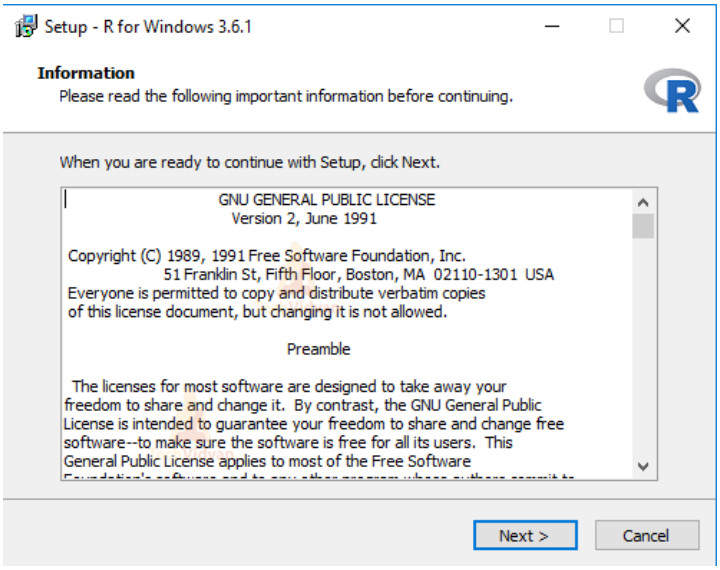
Step – 2: Click on the **Download R for Windows** link.
Step – 3: Click on the **base** subdirectory link or **install R for the first time** link.
Step – 4: Click **Download R X.X.X for Windows** (X.X.X stand for the latest version of R. eg: 3.6.1) and save the executable .exe file.



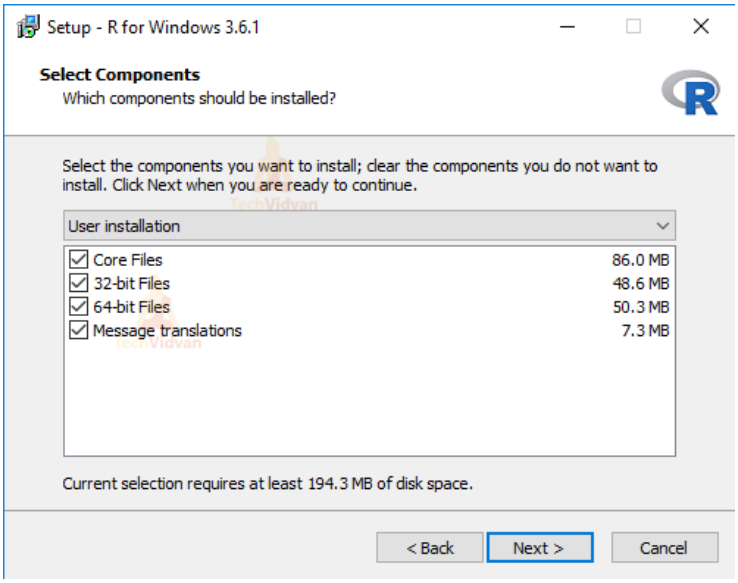
Step – 5: Run the **.exe** file and follow the installation instructions.
5.a. Select the desired language and then click **Next**.



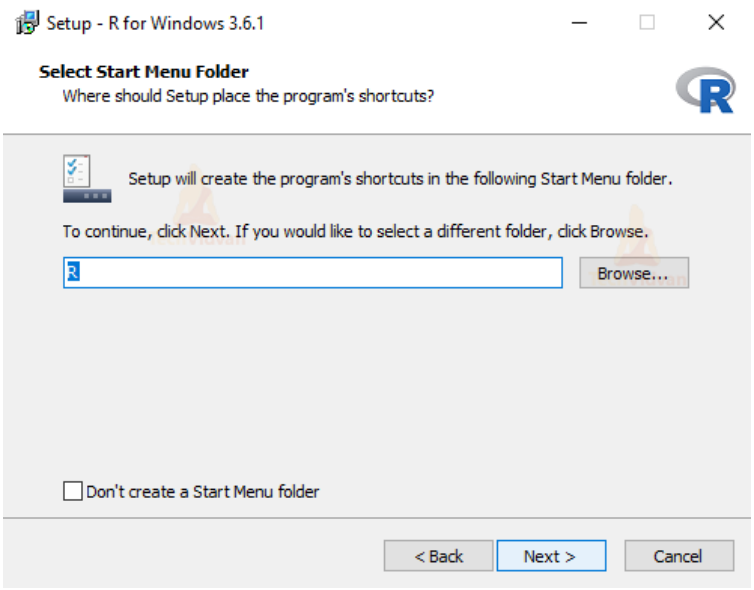
5.b. Read the license agreement and click **Next**.



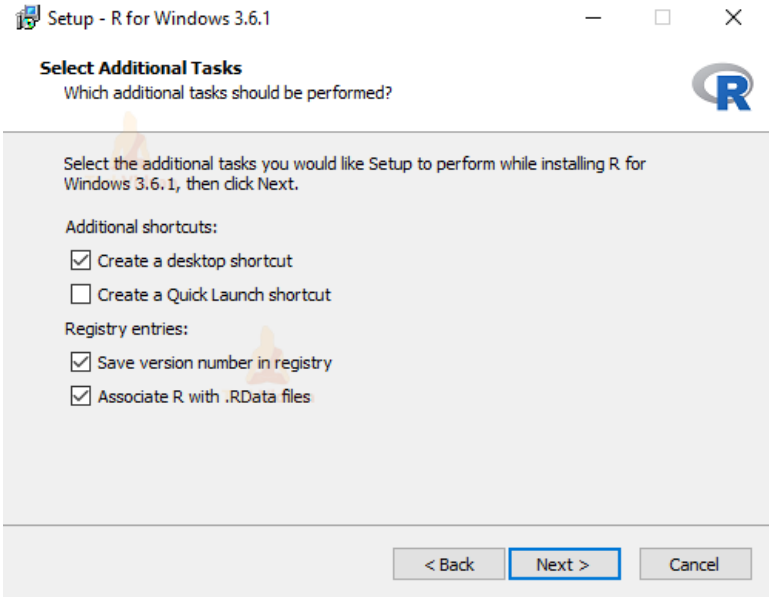
5.c. Select the components you wish to install (it is recommended to install all the components). Click **Next**.



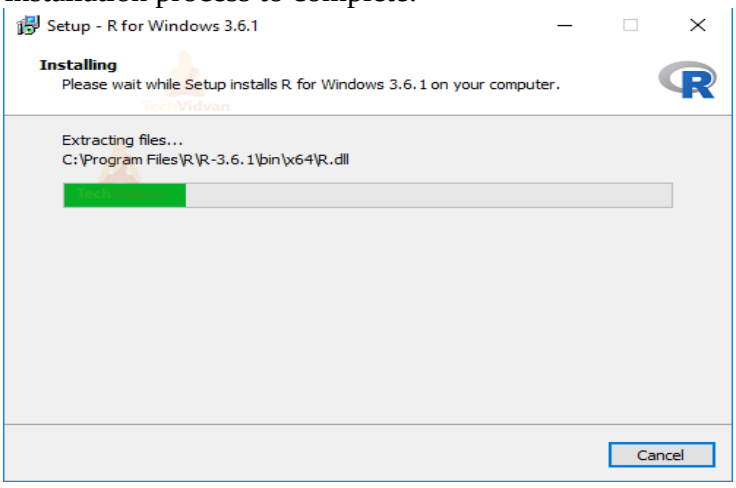
5.d. Enter/browse the folder/path you wish to install R into and then confirm by clicking **Next**.



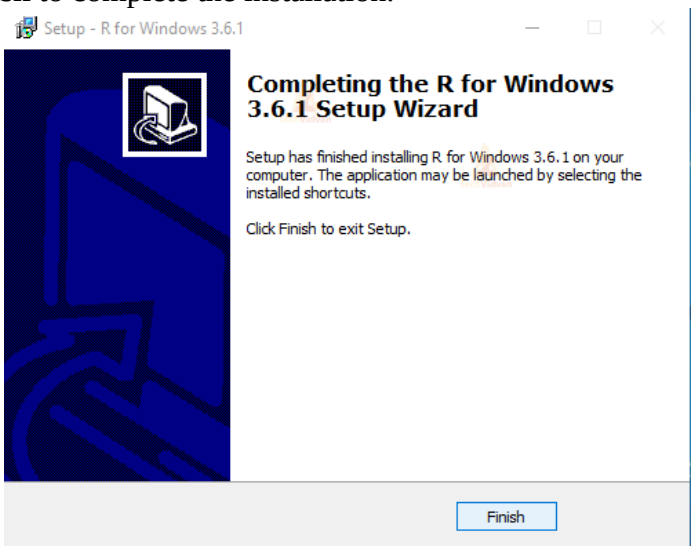
.e. Select additional tasks like creating desktop shortcuts etc. then click **Next**.



5.f. Wait for the installation process to complete.

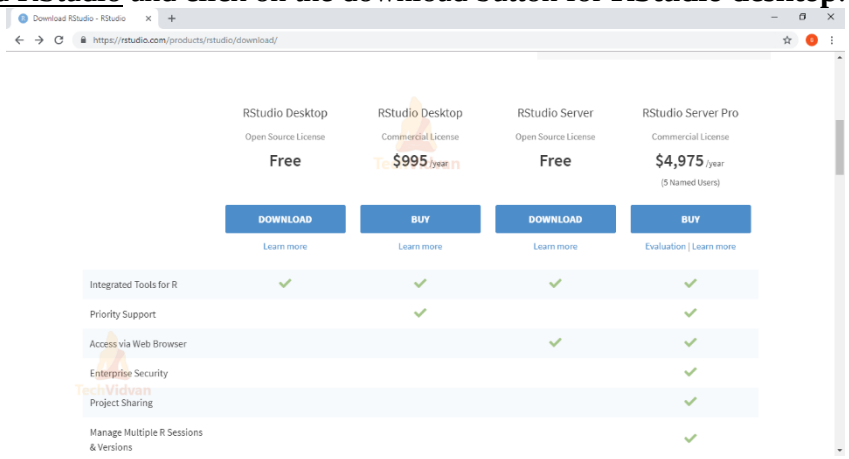


5.g. Click on **Finish** to complete the installation.

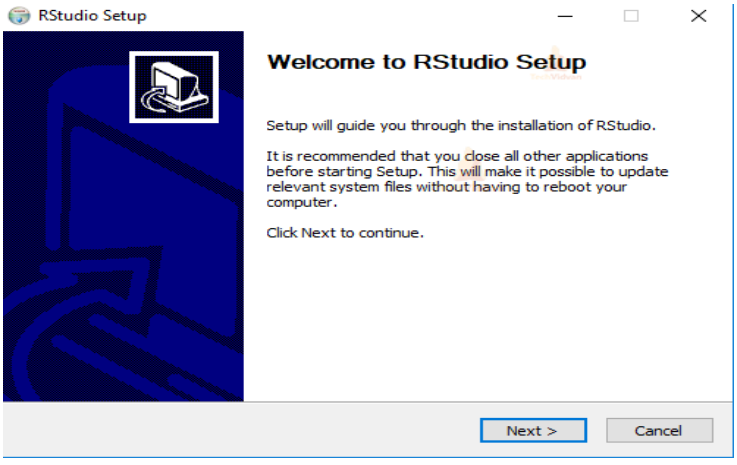


Install RStudio on Windows

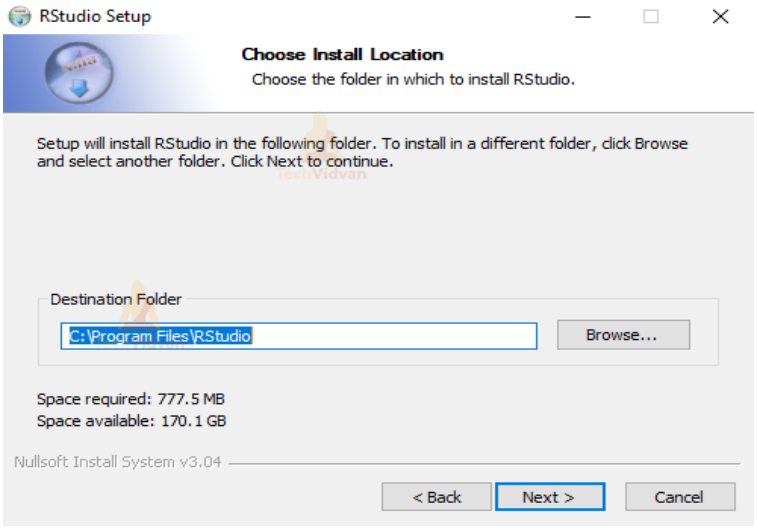
Step – 1: With R-base installed, let's move on to installing RStudio. To begin, go to download RStudio and click on the download button for **RStudio desktop**.



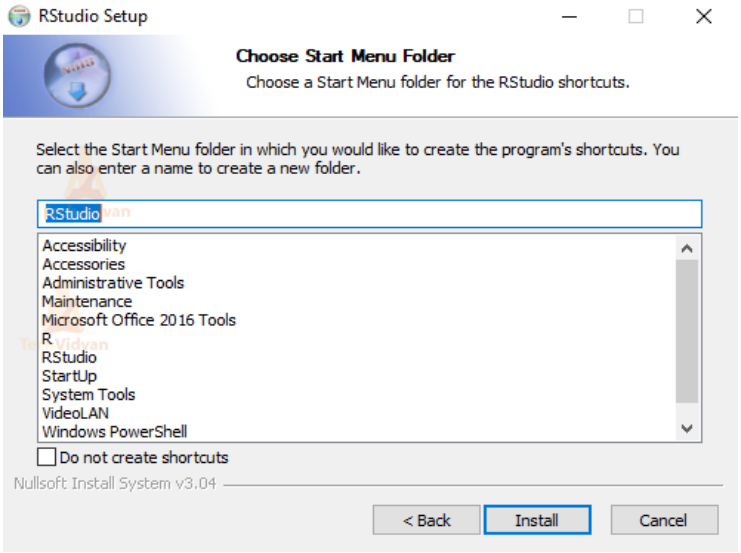
Step – 2: Click on the link for the windows version of RStudio and save the .exe file.
Step – 3: Run the .exe and follow the installation instructions.
3.a. Click **Next** on the welcome window.



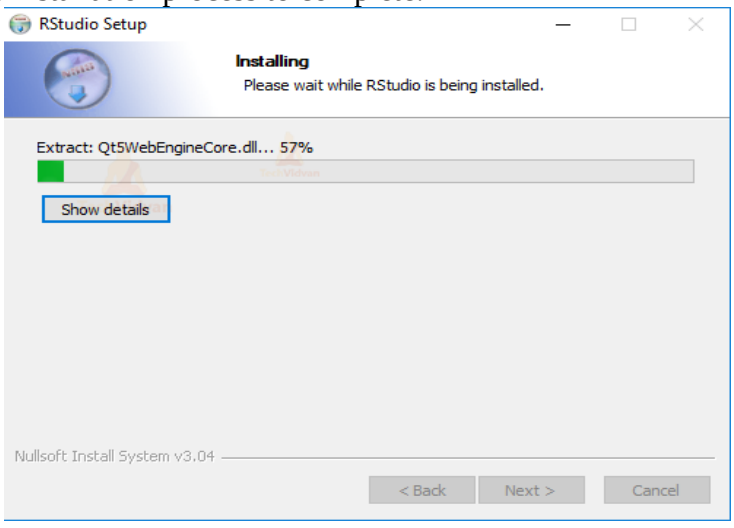
3.b. Enter/browse the path to the installation folder and click **Next** to proceed.



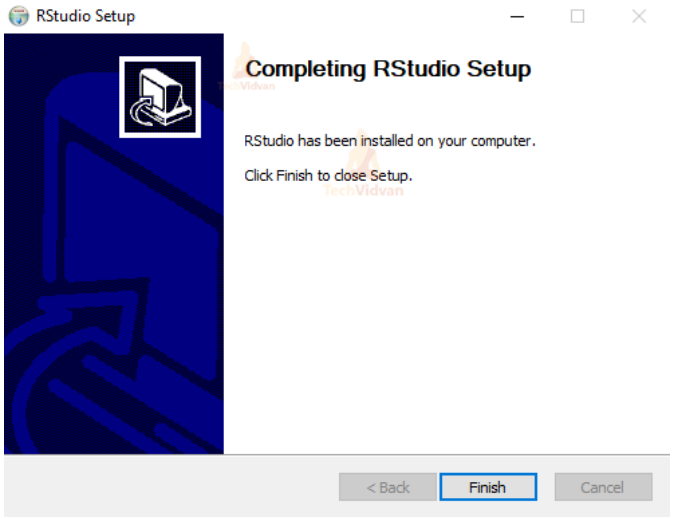
3.c. Select the folder for the start menu shortcut or click on do not create shortcuts and then click **Next**.



3.d. Wait for the installation process to complete.



3.e. Click **Finish** to end the installation.



Data Types in R

here are fundamentally five data types in R. Though straight forward and obvious at first glance, they have a few surprises hidden in them. These elementary data types are:

1. Numeric
2. Integer
3. Complex
4. Character
5. Logical

These data types are often combined to form data structures. Let us explore the meaning of each data type in detail.

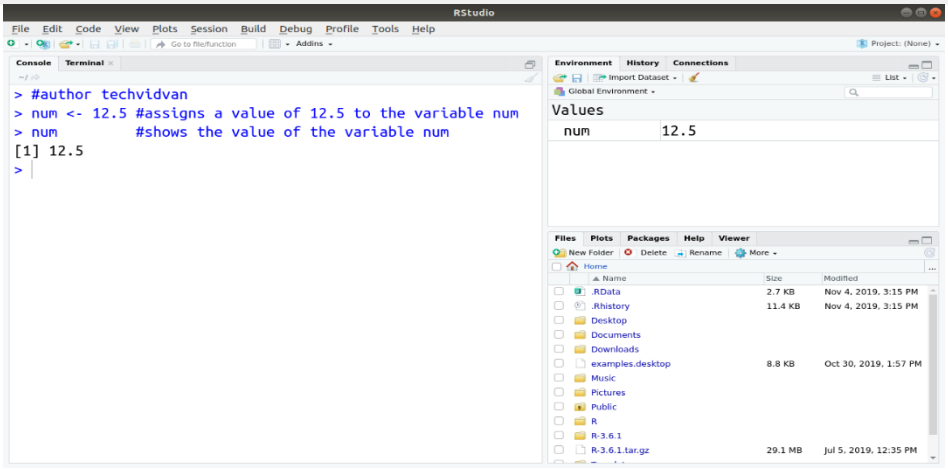
1. Numeric Data Type

Numeric data consists of **decimal values**.

```
> num <- 12.5 #assigns a value of 12.5 to the variable num
> num #shows the value of the variable num
```

Output:

```
[1] 12.5
```

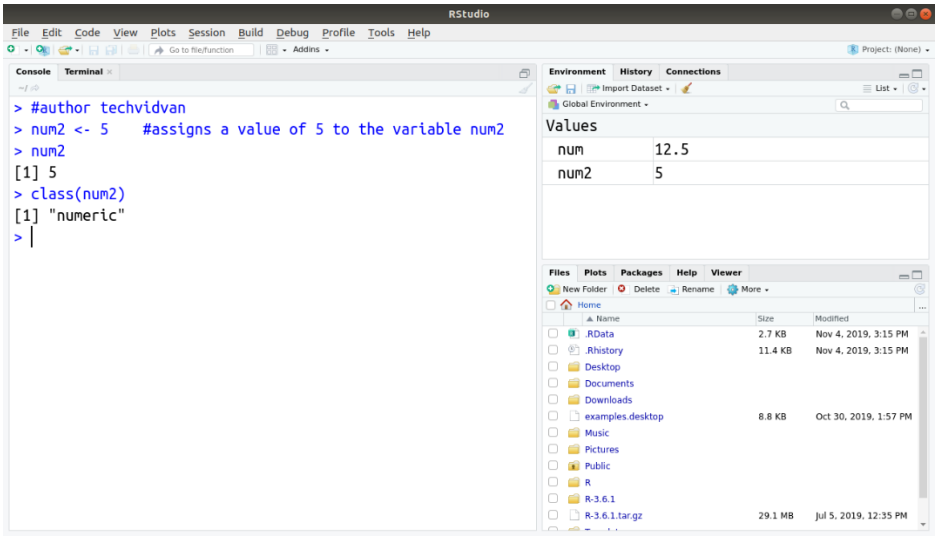


The value doesn't have to be decimal for the variable to be numeric. For example, the following code will result in a numeric value as well.

```
> num2 <- 5 #assigns a value of 5 to the variable num2
> num2
> class(num2) #shows the class or type of the variable num2
```

Output:

```
[1] 5
[1] "numeric"
```



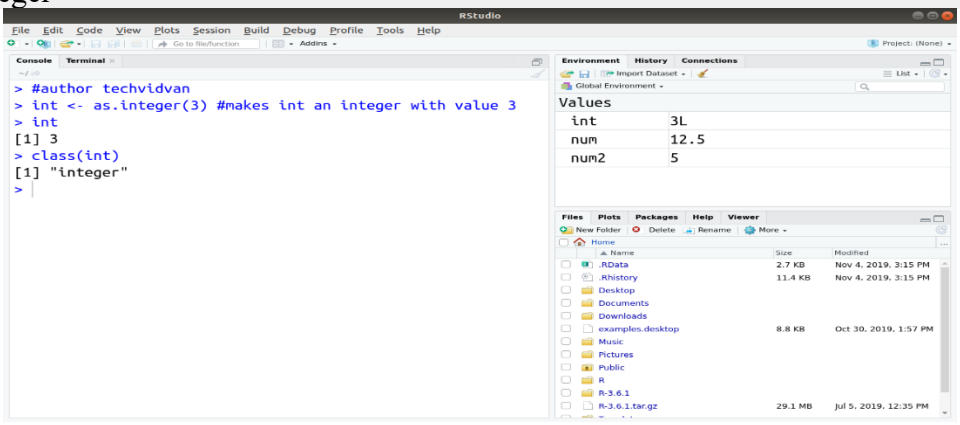
2. Integer data type

In R, there are two ways to create an integer variable. The first is to invoke the **as.integer()** function.

```
> int <- as.integer(3) #makes int an integer with value 3
> int
> class(int)
```

Output:

```
[1] 3
[1] "integer"
```

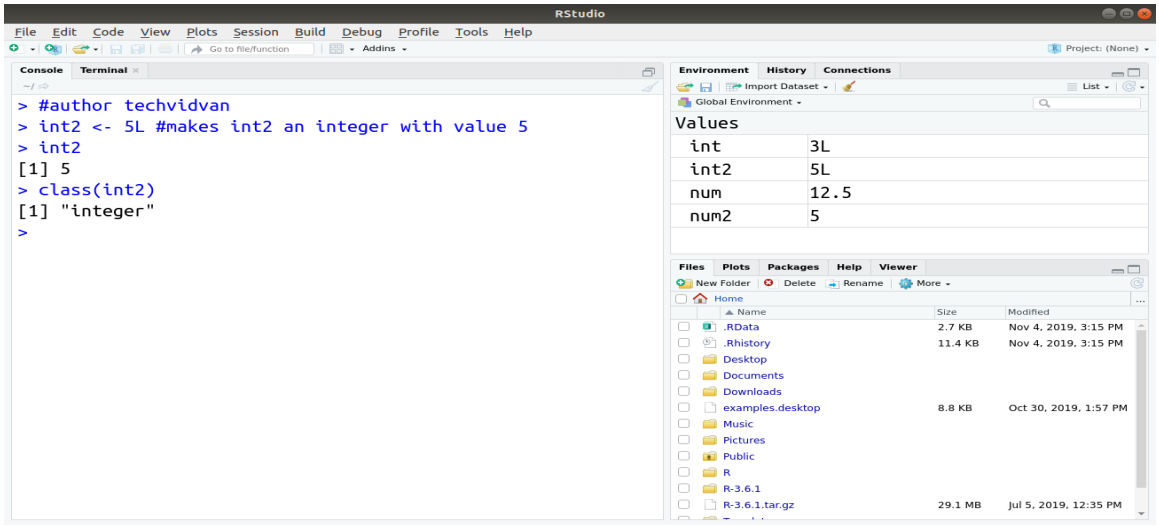


Another method of creating an integer variable is to use 'the capital **L**'.

> int2 <- 5L #makes int2 an integer with value 5
> int2
> **class**(int2)

Output:

[1] 5
[1] "integer"



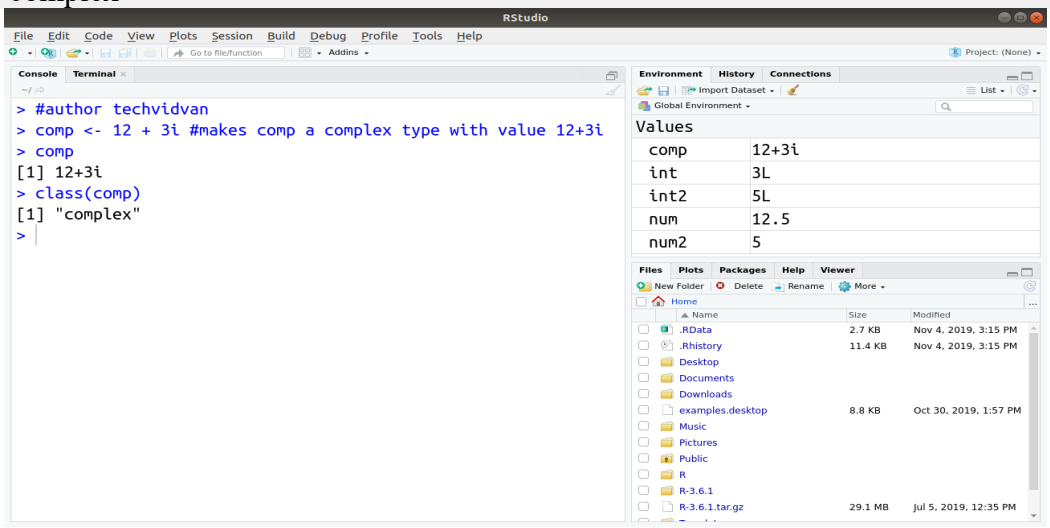
3. Complex data type

The complex data type in R is for complex numbers or numbers with imaginary values.

> comp <- 12 + 3i #makes comp a complex type with value 12+3i
> comp
> **class**(comp)

Output:

[1] 12+3i
[1] "complex"



4. Character data type

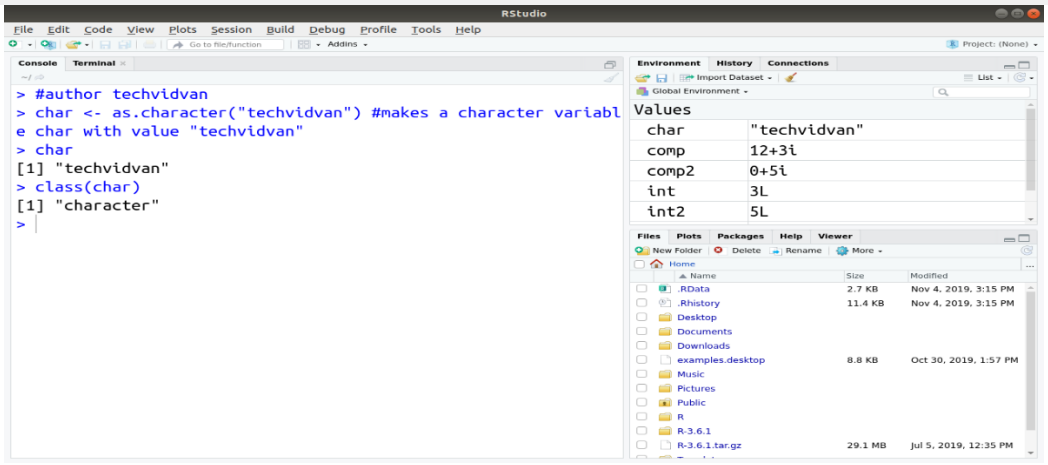
The character data type is used to store strings in R. A character variable can be created in two ways in R.

The first is to invoke the **as.character()** function.

> char <- as.character("techvidvan") #makes a character variable char with value "techvidvan"
> char
> **class**(char)

Output:

[1] “techvidvan”
[1] “character”



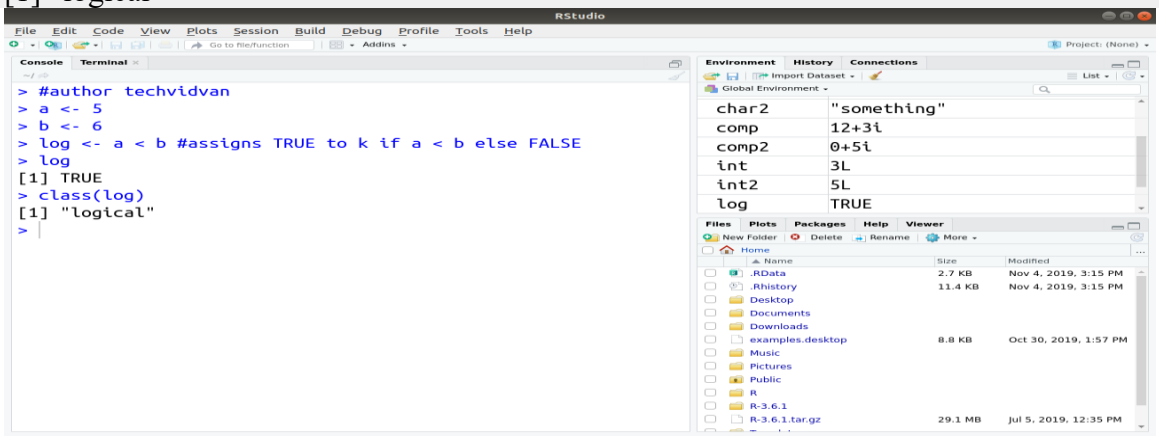
5. Logical data type

A logical variable can have two values either **TRUE** or **FALSE**. Logical are generally created when there is a comparison between variables.

> a <- 5
> b <- 6
> log <- a < b #assigns TRUE to k if a < b else FALSE
> log
> **class**(log)

Output:

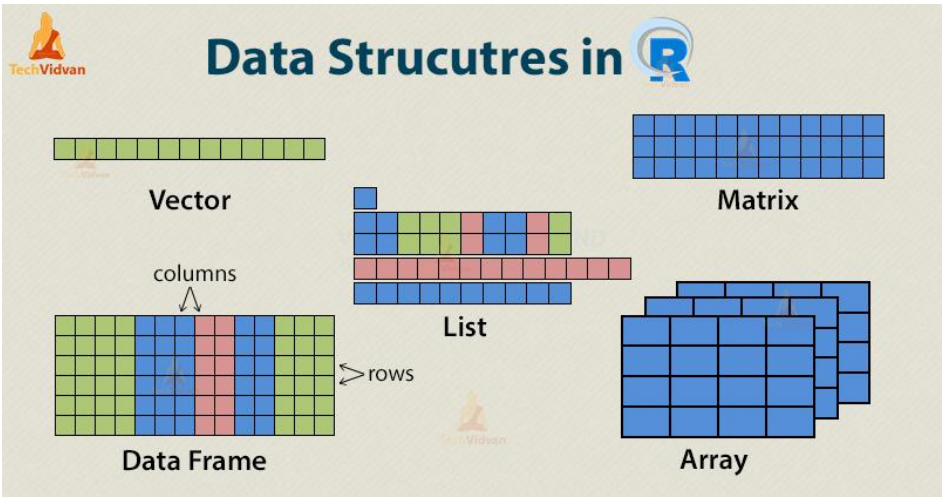
[1] TRUE
[1] “logical”



Data Structures in R

R has the following basic data structures:

1. Vector
2. List
3. Matrix
4. Data frame
5. Array
6. Factor



1. Vectors

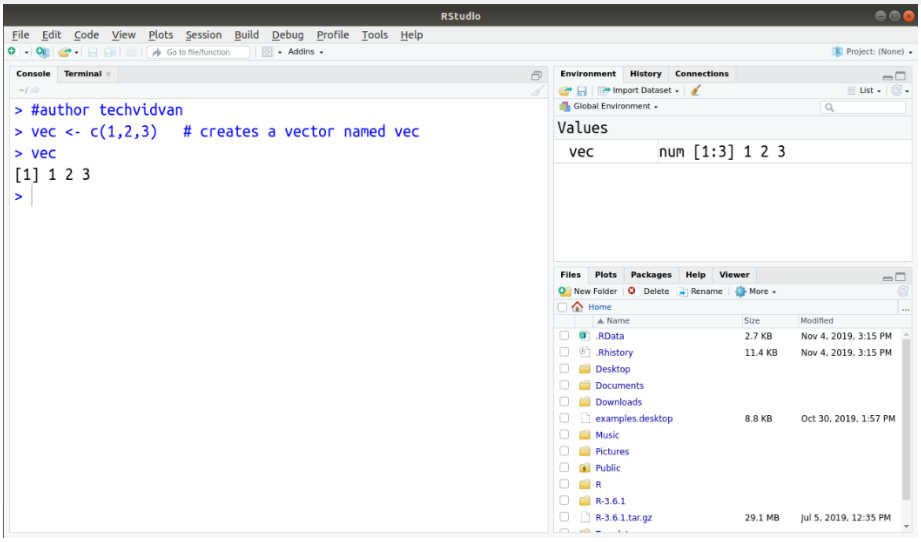
Vectors are **single-dimensional, homogeneous** data structures. To create a vector, use the c() function.

For example:

```
> vec <- c(1,2,3) # creates a vector named vec
> vec
```

Output:

```
[1] 1 2 3
```



The assign() function is another way to create a vector.

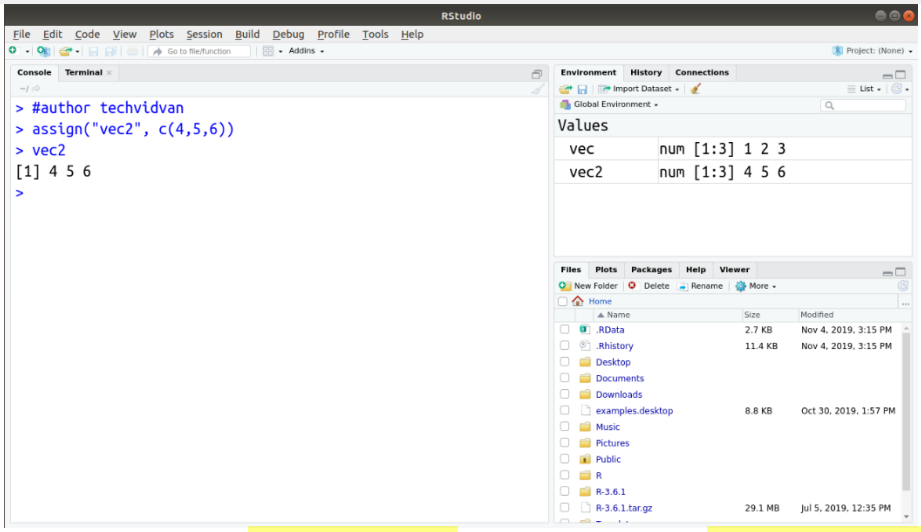
For example:

> assign("vec2", c(4,5,6))

> vec2

Output:

[1] 4 5 6



Vectors can hold values of a single data type. Thus, they can be numeric, logical, character, integer or complex vectors.

For example:

> numeric_vec <- c(1,2,3,4,5)

> integer_vec <- c(1L,2L,3L,4L,5L)

> logical_vec <- c(**TRUE**, **TRUE**, **FALSE**, **FALSE**, **FALSE**)

> complex_vec <- c(12+2i, 3i, 4+1i, 5+12i, 6i)

> character_vec <- c("techvidvan", "this", "is", "a", "character vector")

> numeric_vec

> integer_vec

> logical_vec

> complex_vec

> character_vec

Output:

[1] 1 2 3 4 5

[1] 1 2 3 4 5

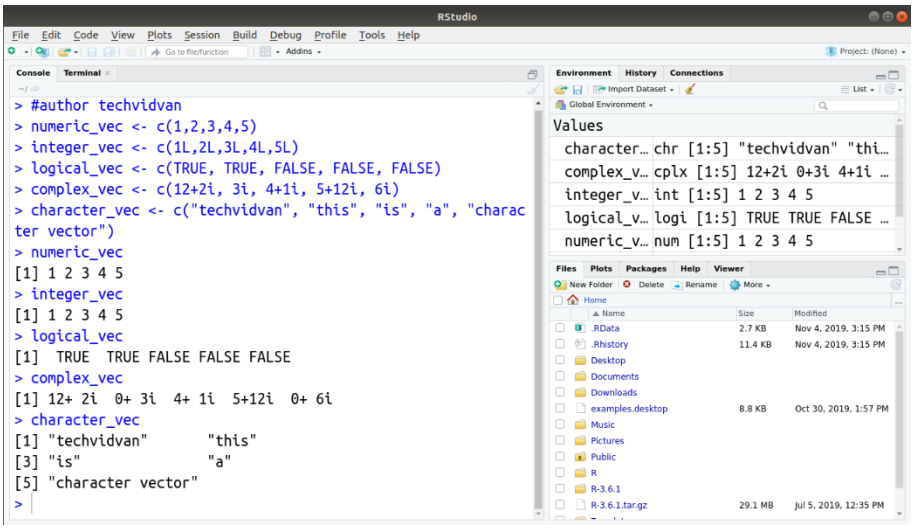
[1] TRUE FALSE TRUE FALSE FALSE

[1] 12+ 2i 0+ 3i 4+ 1i 5+12i 0+ 6i

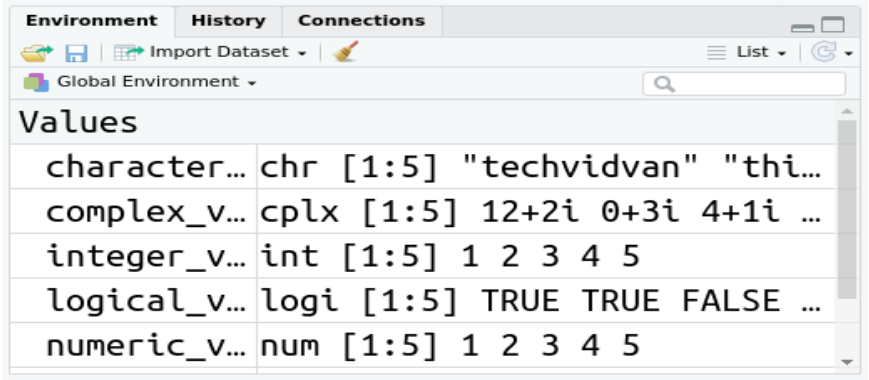
[1] “techvidvan” “this”

[3] “is” “a”

[5] “character vector”



The above code will create the following vectors with corresponding values and types.



2. Lists

Lists are **heterogeneous** data structures. They are very similar to vectors except they can store data of different types. To create a list, we use the list() function.

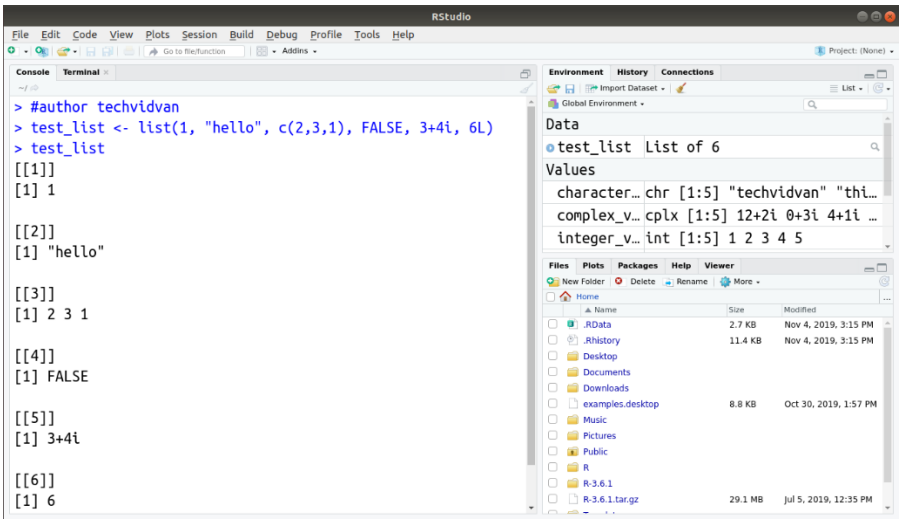
For example

> test_list <- list(1, "hello", c(2,3,1), **FALSE**, 3+4i, 6L)

> test_list

Output:

```
[[1]]
[1] 1
[[2]]
[1] “hello”
[[3]]
[1] 2 3 1
[[4]]
[1] FALSE
[[5]]
[1] 3+4i
[[6]]
[1] 6
```



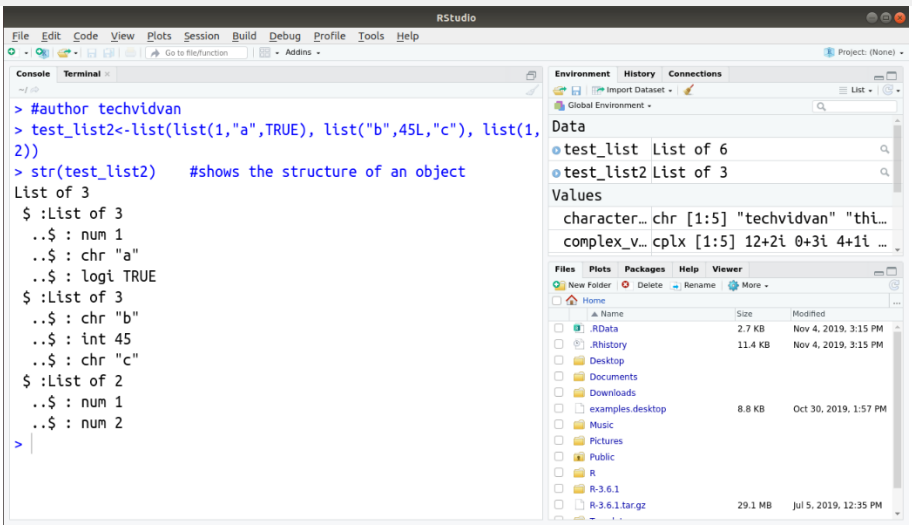
Example:

> test_list2<-list(list(1,"a",**TRUE**), list("b",45L,"c"), list(1,2))

> str(test_list2) #shows the structure of an object

Output:

List of 3
$:List of 3
..$: num 1
..$: chr “a”
..$: logi TRUE
$:List of 3
..$: chr “b”
..$: int 45
..$: chr “c”
$:List of 2
..$: num 1
..$: num 2



3. Matrix

Matrices are **two-dimensional, homogeneous** data structures. This means that all values in a matrix have to be of the same type. Coercion takes place if there is more than one data type. They have rows and columns.
By default, matrices are in **column-wise** order. The basic syntax to create a matrix is:

>matrix(data, nrow, ncol, byrow, dimnames)

Where **data** is the input values in the matrix given as a vector,

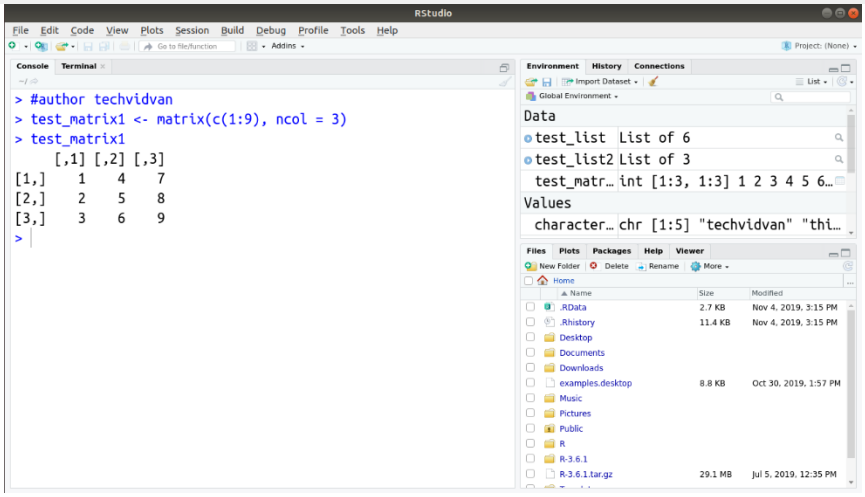
nrow is the number of rows,
ncol is the number of columns,
byrow is a logical which tells the function to arrange the matrix row-wise, by default it is set to FALSE,
dimnames is a list of the names of the rows/columns created.
The following code will create a matrix with 3 rows and values 1 to 9 in a column-wise order.

For example:

```
> test_matrix1 <- matrix(c(1:9), ncol = 3)
> test_matrix1
```

Output:

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```



4. Data Frames

Data frames are **two-dimensional, heterogeneous** data structures. They are lists of vectors of equal lengths. Data frames have the following constraints placed upon them:

1. A data-frame **must have column-names** and each row should have a **unique name**.
2. Each column should have the **same number** of items.
3. Each item in a single column should be of the **same type**.

4. Different columns can have different data types.

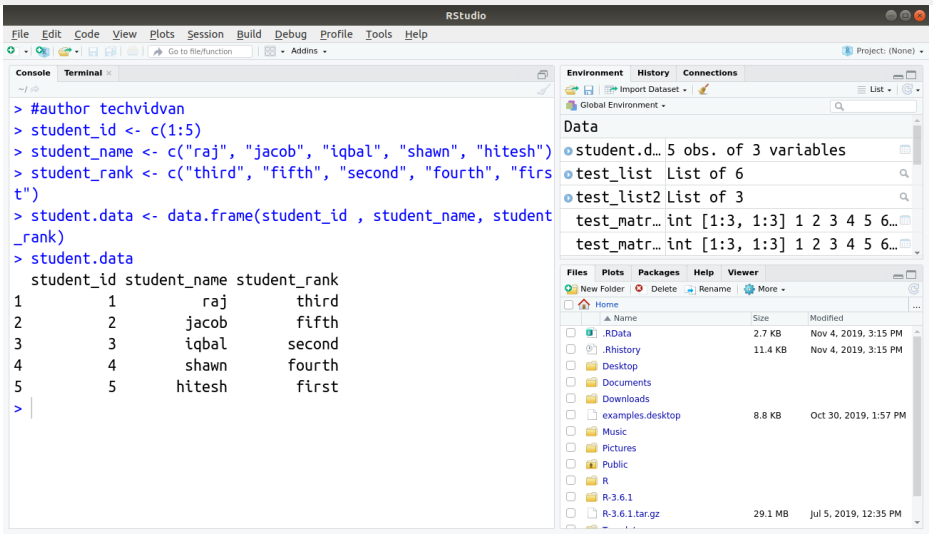
To create a data frame, use the data.frames() function.

For example:

```
> student_id <- c(1:5)
> student_name <- c("raj", "jacob", "iqbal", "shawn", "hitesh")
> student_rank <- c("third", "fifth", "second", "fourth", "first")
> student.data <- data.frame(student_id , student_name, student_rank)
> student.data
```

Output:

	student_id	student_name	student_rank
1	1	raj	third
2	2	jacob	fifth
3	3	iqbal	second
4	4	shawn	fourth
5	5	hitesh	first



5. Arrays

Arrays are **three dimensional, homogeneous** data structures. They are collections of matrices stacked one on top of the other in layers.

You can create an array using the array() function. The following is the syntax of it:

Array_name = array(data,dim,dimnames)

Where **array_name** is the name of the array,

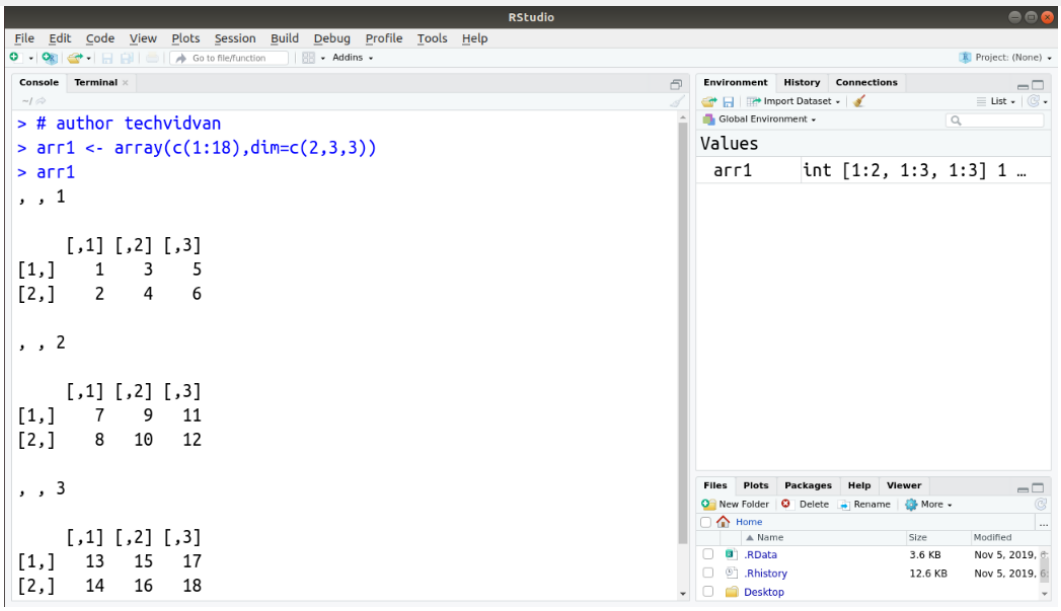
data is the data that is filled inside the array,
dim is a vector containing the dimensions of the array,
and **dimnames** is a list containing the names of the rows, columns, and matrices inside the array.

Here is an **example** of the array() function:

```
> arr1 <- array(c(1:18),dim=c(2,3,3))
> arr1
```

Output:

```
, , 1[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6, , 2[,1] [,2] [,3]
[1,] 7 9 11
[2,] 8 10 12, , 3[,1] [,2] [,3]
[1,] 13 15 17
[2,] 14 16 18
```



6. Factors

Factors are vectors that can only store predefined values. They are useful for storing **categorical data**. Factors have two attributes:

- **Class** – which has a value of "factor", it makes it behave differently than a normal vector.
- **Levels** – which is the set of allowed values

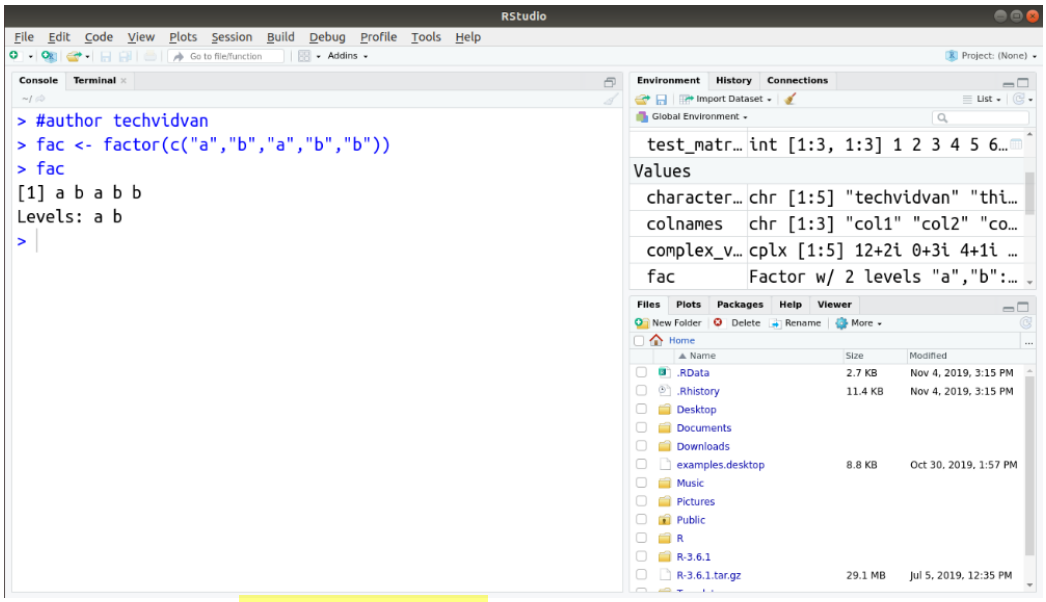
You can create a factor using the factor() function.

For example:

```
> fac <- factor(c("a", "b", "a", "b", "b"))
> fac
```

Output:

```
[1] a b a b b
Levels: a b
```



Factors can store both strings and integers. They are useful to categorize unique values in columns like “TRUE” or “FALSE”, or “MALE” or “FEMALE”, etc..

Questions:

1. List and explain the data types in R?
2. List and explain the data structures in R?

Conclusion : Thus, I have studied installation of R and study of R objects with basic statistics.

EXPERIMENT NO. 05

Title: Implement R Program to perform data pre-processing, analysis and visualization.

Outcome: Students will be able to implement R Program to perform data pre-processing, analysis and visualization.

Theory:

Data preprocessing is the initial phase of Machine Learning where data is prepared for machine learning models. This part is crucial and needs to be performed properly and systematically. If not, we will end up building models that are not accurate for their purpose.

Steps in data preprocessing

- Steps in Data Preprocessing
  - Step 1: Importing the Dataset
  - Step 2: Handling the Missing Data
- Step 3: Encoding Categorical Data.
  - Output
- Step 4: Splitting the Dataset into the Training and Test sets
  - Training set
  - Test set
- Step 5: Feature Scaling
  - training_set
  - test_set

Step 1: Importing the dataset

Before we start preparing our data, first we need to dowload it from here and load it in

RStudio IDE.

Here is how to achieve this.

```
Dataset = read_csv('data.csv')
```

This code imports our data stored in CSV format.

We can have a look at our data using the 'view()' function:

```
view(Dataset)
```

Upon executing we obtain our dataset as below.

Output

	Country	Age	Salary	Purchased
1	France	44	72000	No
2	Spain	27	48000	Yes
3	Germany	30	54000	No
4	Spain	38	61000	No
5	Germany	40	*NA*	Yes
6	France	35	58000	Yes
7	Spain	*NA*	52000	No
8	France	48	79000	Yes
9	Germany	50	83000	No
10	France	37	67000	Yes

Our dataset has four columns and ten observations, it shows how customers from three different countries with different ages and salaries responded to the purchase of a certain product.

Step 2: Handling the missing data

From the dataset, the Age and Salary column report missing data. Before implementing our machine learning models, this problem needs to be solved, otherwise it will cause a serious problem to our machine learning models. Therefore, it's our responsibility to ensure this missing data is eliminated from our dataset using the most appropriate technique.

Here are two techniques we can use to handle missing data:

1. **Delete the observation reporting the missing data:**

This technique is suitable when dealing with big datasets and with very few missing values i.e. deleting one row from a dataset with thousands of observations can not affect the quality of the data. When the dataset reports many missing values, it can be very dangerous to use this technique. Deleting many rows from a dataset can lead to the loss of crucial information contained in the data.

To ensure this does not happen, we make use of an appropriate technique that has no harm to the quality of the data.

2. **Replace the missing data with the average of the feature in which the data is missing:**

This technique is the best way so far to deal with the missing values. Many statisticians make use of this technique over that of the first one.

Now that we know the techniques used to treat the missing data, let's solve this problem from our data. In our case, we shall make use of the second technique.

Let's start by replacing the missing data in the Age column with the mean of that column.

The code below carries out such a task.

```
Dataset$Age = ifelse(is.na(Dataset$Age),
                     ave(Dataset$Age, FUN = function (x)mean(x, na.rm = TRUE)),
                     Dataset$Age)
```

What does the code above really do?

Dataset$Age: simply take the Age column from our dataset.

In the Age column, we've just taken that from our data set, we need to replace the missing data, and at the same time keep the data that is not missing.

This objective is achieved by the use of the if-else statement.

Our ifelse statement is taking three parameters:

- The first parameter is if the condition is true.
- The second parameter is the value we input if the condition is true.
- The third parameter is the action we take if the condition is false.

Our condition is is.na(Dataset$Age). This will tell us if a value in the Dataset$Age is missing or not. It returns a logical output, YES if a value is missing and NO if a value is not missing. The second parameter, the 'ave()' function, finds the mean of the Age column.

Because this column reports NA values, we need to exclude the null data in the calculation of the mean, otherwise we shall obtain the mean as NA.

This is the reason we pass na.rm = TRUE in our mean function just as to declare those values that should be used and those should be excluded when calculating the mean of the vector Age.

The third condition is the value that will be returned if the value in the Age column of the dataset is not missing.

Executing the code we obtain:

	Country	Age	Salary	Purchased
1	France	44.00000	72000	No
2	Spain	27.00000	48000	Yes
3	Germany	30.00000	54000	No
4	Spain	38.00000	61000	No
5	Germany	40.00000	*NA*	Yes
6	France	35.00000	58000	Yes
7	Spain	38.77778	52000	No
8	France	48.00000	79000	Yes
9	Germany	50.00000	83000	No
10	France	37.00000	67000	Yes

The missing value that was in the Age column of our data set has successfully been replaced with the mean of the same column.

We do the same for the Salary column by executing the code below:

```
Dataset$Salary = ifelse(is.na(Dataset$Salary),
                        ave(Dataset$Salary, FUN = function (x)mean(x, na.rm = TRUE)),
                        Dataset$Salary)
```

	Country	Age	Salary	Purchased
1	France	44.00000	72000.00	No
2	Spain	27.00000	48000.00	Yes
3	Germany	30.00000	54000.00	No
4	Spain	38.00000	61000.00	No
5	Germany	40.00000	63777.78	Yes
6	France	35.00000	58000.00	Yes
7	Spain	38.77778	52000.00	No
8	France	48.00000	79000.00	Yes
9	Germany	50.00000	83000.00	No
10	France	37.00000	67000.00	Yes

The missing value that was in the Salary column was successfully replaced with the mean of the same column.

Step 3: Encoding categorical data

Encoding refers to transforming text data into numeric data. Encoding Categorical data simply means we are transforming data that fall into categories into numeric data.

In our dataset, the Country column is Categorical data with 3 levels i.e. France, Spain, and Germany. The purchased column is Categorical data as well with 2 categories, i.e. YES and NO.

The machine models we built on our dataset are based on mathematical equations and it's only take numbers in those equations.

Keeping texts of a categorical variable in the equation can cause some troubles to the machine learning models and this why we encode those variables. To transform a categorical variable into numeric, we use the factor() function.

Let start by encoding the Country column.

```
Dataset$Country = factor(Dataset$Country,
                         levels = c('France','Spain','Germany'),
                         labels = c(1.0, 2.0 , 3.0 ))
```

Executing the code above we obtain.

Output

	Country	Age	Salary	Purchased
1	1	44.00000	72000.00	No
2	2	27.00000	48000.00	Yes
3	3	30.00000	54000.00	No
4	2	38.00000	61000.00	No
5	3	40.00000	63777.78	Yes
6	1	35.00000	58000.00	Yes
7	2	38.77778	52000.00	No
8	1	48.00000	79000.00	Yes
9	3	50.00000	83000.00	No
10	1	37.00000	67000.00	Yes

Our country names were successfully replaced with numbers.

We do the same for the purchased column.

```
Dataset$Purchased = factor(Dataset$Purchased,
```

```
                    levels = c('No', 'Yes'),
                    labels = c(0, 1))
Dataset$Purchased[is.na(Dataset$Purchased)] <- 0
as.factor(Dataset$Purchased)
```

Using the view() function we obtain.

	Country	Age	Salary	Purchased
1	1	44.00000	72000.00	0
2	2	27.00000	48000.00	1
3	3	30.00000	54000.00	0
4	2	38.00000	61000.00	0
5	3	40.00000	63777.78	1
6	1	35.00000	58000.00	1
7	2	38.77778	52000.00	0
8	1	48.00000	79000.00	1
9	3	50.00000	83000.00	0
10	1	37.00000	67000.00	1

Our purchased column was successfully encoded into 0,s, and 1,s.

Step 4: Splitting the dataset into the training and test set

In machine learning, we split data into two parts:

- Training set: The part of the data that we implement our machine learning model on.
- Test set: The part of the data that we evaluate the performance of our machine learning model on.

The reason we split this data is to ensure that our machine learning model does not overlearn the correlation of data it's trained on. If we let it learn too much on the data, it may perform poorly when tested on a new dataset with a different correlation.

Therefore, whenever we are building a machine learning model, the idea is to implement it on the training set and evaluate it on the test set. We expect the performance in the training set and test set to be different and if this is the case the model can adapt to new datasets.

Using our dataset, let's split it into the training and test sets.

To begin with, we first load the required library.

```
library(caTools)# required library for data splition
set.seed(123)
split = sample.split(Dataset$Purchased, SplitRatio = 0.8)# returns true if observation goes to the Training set and false if observation goes to the test set.

#Creating the training set and test set separately
training_set = subset(Dataset, split == TRUE)
test_set = subset(Dataset, split == FALSE)
training_set
test_set
```

Executing our code yields:

Training Set:

```
> training_set
   Country      Age   Salary Purchased
1        1 44.00000 72000.00         0
2        2 27.00000 48000.00         1
3        3 30.00000 54000.00         0
4        2 38.00000 61000.00         0
5        3 40.00000 63777.78         1
7        2 38.77778 52000.00         0
8        1 48.00000 79000.00         1
10       1 37.00000 67000.00         1
```

From the results it clear that eight observations, 0.8 of our dataset observations, were split into the training set.

Test Set:

```
> test_set
   Country Age Salary Purchased
6        1  35  58000         1
9        3  50  83000         0
>
```

From the output it clear that two observations went to the test set.

Step 5: Feature scaling

It's a common case that in most datasets, features also known as inputs, are not on the same scale. Many machine learning models are Euclidian distant-based.

It happens that, the features with the large units dominate those with small units when it comes to calculation of the Euclidian distance and it will be as if those features with small units do not exist.

To ensure this does not occur, we need to encode our features so that they all fall in the range between -3 and 3. There are several ways we can use to scale our features. The most used one is the standardization and normalization technique.

The normalization technique is used when the data is normally distributed while standardization works with both normally distributed and the data that is not normally distributed.

The formula for these two techniques is shown below.

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

Now, let's scale both the training set and test set of our dataset separately.

Here is how we achieve this:

```
training_set[, 2:3] = scale(training_set[, 2:3])
test_set[, 2:3] = scale(test_set[, 2:3])
training_set
test_set
```

Executing our code we obtain:

Training Set:

```
> training_set
   Country         Age     Salary Purchased
1        1  0.90101716  0.9392746         0
2        2 -1.58847494 -1.3371160         1
3        3 -1.14915281 -0.7680183         0
4        2  0.02237289 -0.1040711         0
5        3  0.31525431  0.1594000         1
7        2  0.13627122 -0.9577176         0
8        1  1.48678000  1.6032218         1
10       1 -0.12406783  0.4650265         1
```

Test Set:

```
> test_set
  Country        Age     Salary Purchased
6       1 -0.7071068 -0.7071068         1
9       3  0.7071068  0.7071068         0
>
```

Data Visualization in R

Data visualization is the technique used to deliver insights in data using visual cues such as graphs, charts, maps, and many others. This is useful as it helps in intuitive and easy understanding of the large quantities of data and thereby make better decisions regarding it.

Data Visualization in R Programming Language

The popular data visualization tools that are available are Tableau, Plotly, R, Google Charts, Infogram, and Kibana. The various data visualization platforms have different capabilities, functionality, and use cases. They also require a different skill set. This article discusses the use of R for data visualization.

R is a language that is designed for statistical computing, graphical data analysis, and scientific research. It is usually preferred for data visualization as it offers flexibility and minimum required coding through its packages.

Consider the following *airquality* data set for visualization in R:

Ozone	Solar R.	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

Types of Data Visualizations

Some of the various types of visualizations offered by R are:

Bar Plot

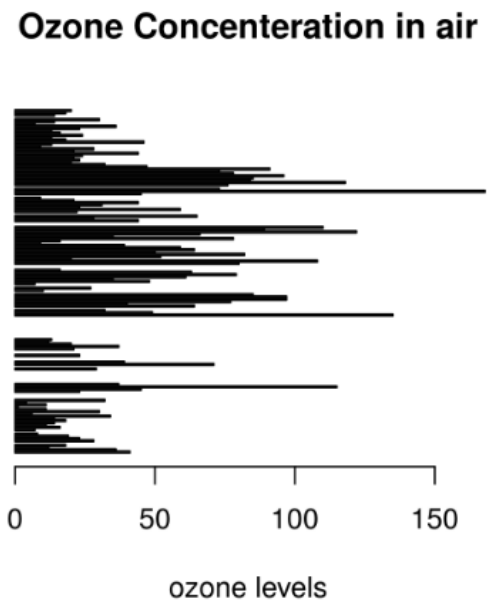
There are two types of bar plots- horizontal and vertical which represent data points as horizontal or vertical bars of certain lengths proportional to the value of the data item. They are generally used for continuous and categorical variable plotting. By setting the **horiz** parameter to true and false, we can get horizontal and vertical bar plots respectively.

Example 1:

```
# Horizontal Bar Plot for
# Ozone concentration in air
barplot(airquality$Ozone,
        main = 'Ozone Concenteration in air',
```

```
                xlab = 'ozone levels', horiz = TRUE)
```

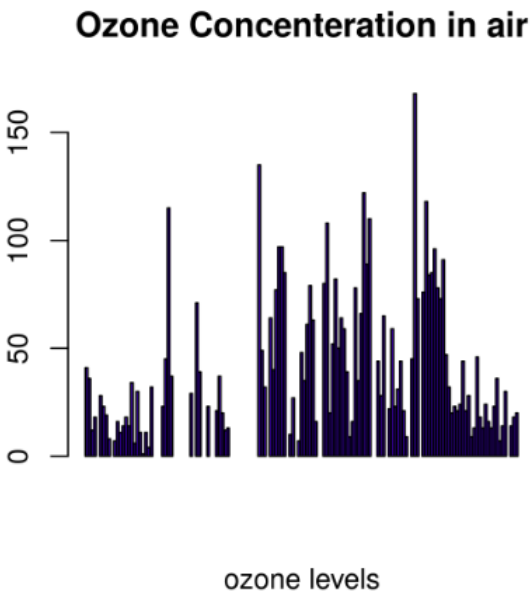
Output:



Example 2:

```
# Vertical Bar Plot for
# Ozone concentration in air
barplot(airquality$Ozone, main = 'Ozone Concenteration in air',
                xlab = 'ozone levels', col ='blue', horiz = FALSE)
```

Output:



Bar plots are used for the following scenarios:

- To perform a comparative study between the various data categories in the data set.
- To analyze the change of a variable over time in months or years.

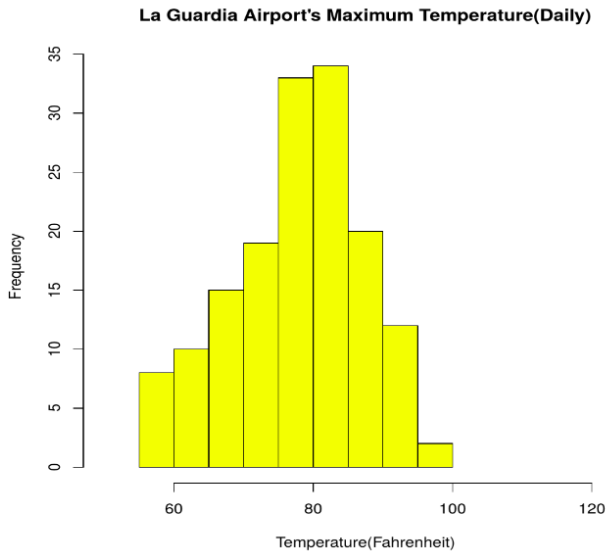
Histogram

A histogram is like a bar chart as it uses bars of varying height to represent data distribution. However, in a histogram values are grouped into consecutive intervals called bins. In a Histogram, continuous values are grouped and displayed in these bins whose size can be varied.

Example:

```
# Histogram for Maximum Daily Temperature
data(airquality)

hist(airquality$Temp, main ="La Guardia Airport's\
Maximum Temperature(Daily)",
        xlab ="Temperature(Fahrenheit)",
        xlim = c(50, 125), col ="yellow",
        freq = TRUE)
```

Output:



For a histogram, the parameter **xlim** can be used to specify the interval within which all values are to be displayed. Another parameter **freq** when set to *TRUE* denotes the frequency of the various values in the histogram and when set to *FALSE*, the probability densities are represented on the y-axis such that they are of the histogram adds up to one.

Histograms are used in the following scenarios:

- To verify an equal and symmetric distribution of the data.

- To identify deviations from expected values.

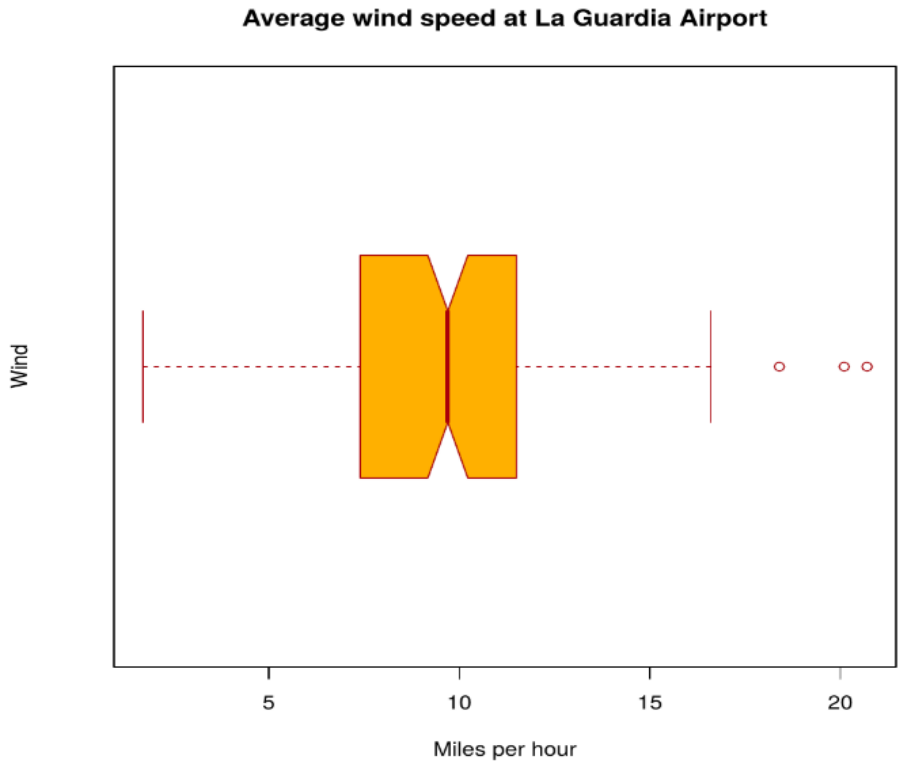
Box Plot

The statistical summary of the given data is presented graphically using a boxplot. A boxplot depicts information like the minimum and maximum data point, the median value, first and third quartile, and interquartile range.

Example:

```
# Box plot for average wind speed
data(airquality)

boxplot(airquality$Wind, main = "Average wind speed\
at La Guardia Airport",
                xlab = "Miles per hour", ylab = "Wind",
                col = "orange", border = "brown",
                horizontal = TRUE, notch = TRUE)
```

Output:

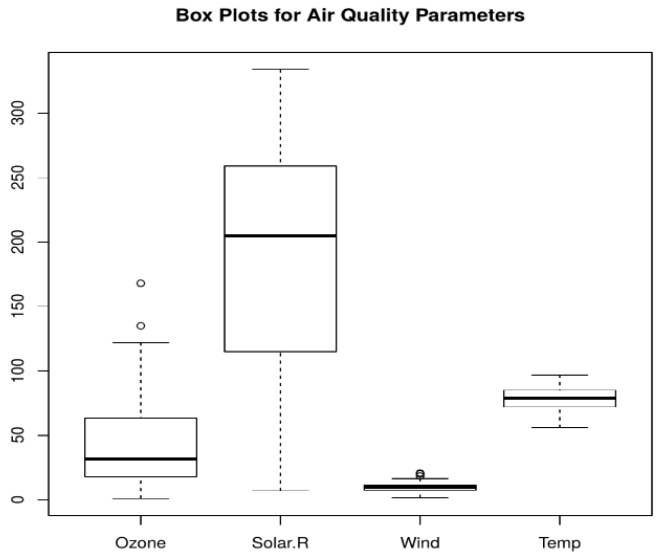


Multiple box plots can also be generated at once through the following code:

Example:

```
# Multiple Box plots, each representing
# an Air Quality Parameter
boxplot(airquality[, 0:4],
                main ='Box Plots for Air Quality Parameters')
```

Output:



Box Plots are used for:

- To give a comprehensive statistical description of the data through a visual cue.
- To identify the outlier points that do not lie in the inter-quartile range of data.

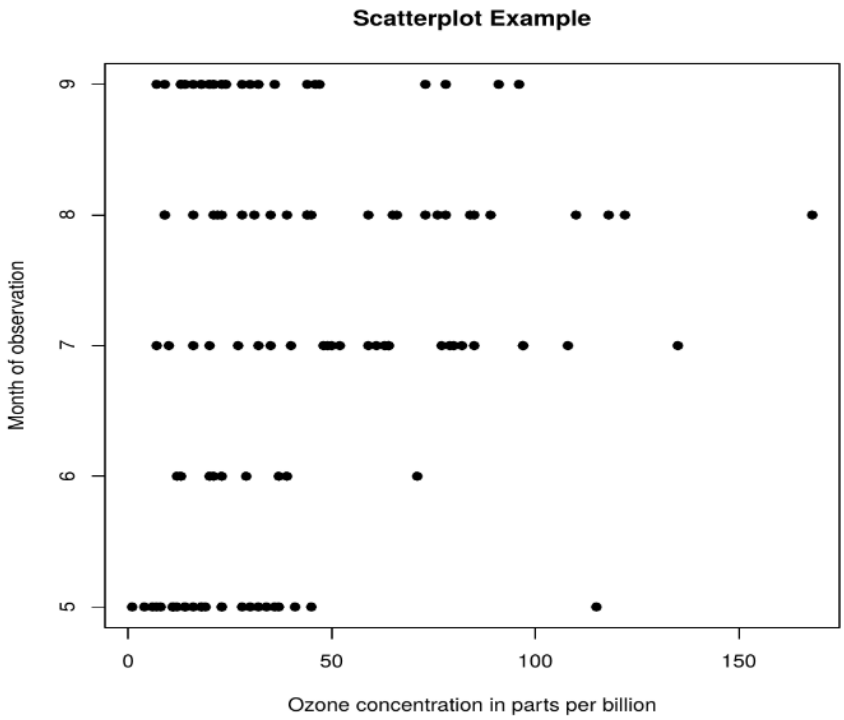
Scatter Plot

A scatter plot is composed of many points on a Cartesian plane. Each point denotes the value taken by two parameters and helps us easily identify the relationship between them.

Example:

```
# Scatter plot for Ozone Concentration per month
data(airquality)

plot(airquality$Ozone, airquality$Month,
        main ="Scatterplot Example",
        xlab ="Ozone Concentration in parts per billion",
        ylab =" Month of observation ", pch = 19)
```

Output:



Scatter Plots are used in the following scenarios:

- To show whether an association exists between bivariate data.
- To measure the strength and direction of such a relationship.

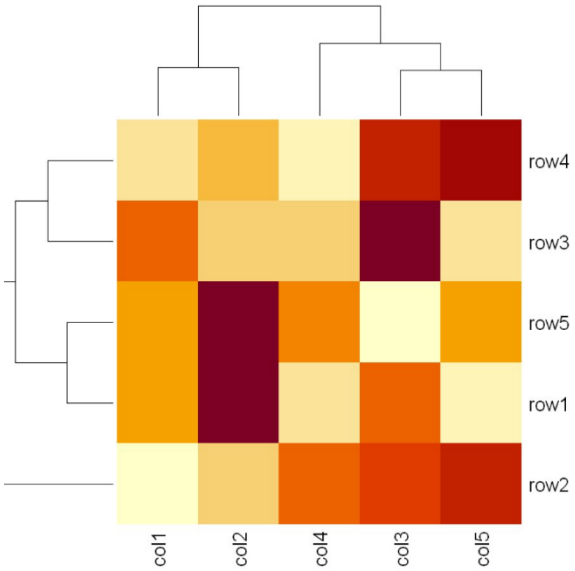
Heat Map

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. heatmap() function is used to plot heatmap.

Syntax: *heatmap(data)*
Parameters: *data: It represent matrix data, such as values of rows and columns*
Return: *This function draws a heatmap.*

```
# Set seed for reproducibility
# set.seed(110)

# Create example data
data <- matrix(rnorm(50, 0, 5), nrow = 5, ncol = 5)

# Column names
colnames(data) <- paste0("col", 1:5)
rownames(data) <- paste0("row", 1:5)

# Draw a heatmap
heatmap(data)
```

Output:

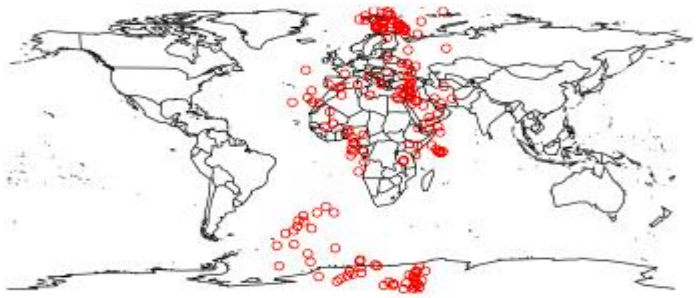


Map visualization in R

Here we are using maps package to visualize and display geographical maps using an R programming language.
install.packages("maps")
Link of the dataset: worldcities.csv

```
# Read dataset and convert it into
# Dataframe
data <- read.csv("worldcities.csv")
df <- data.frame(data)

# Load the required libraries
library(maps)
map(database = "world")

# marking points on map
points(x = df$lat[1:500], y = df$lng[1:500], col = "Red")
```

Output:



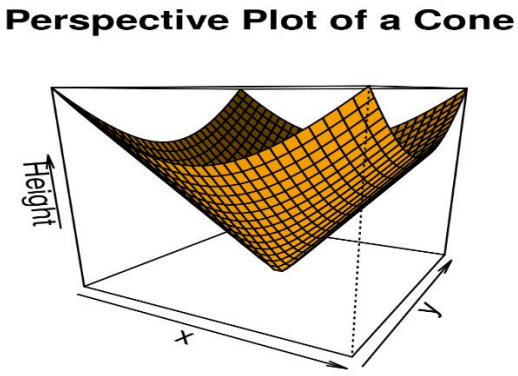
3D Graphs in R

Here we will use preps() function, This function is used to create 3D surfaces in perspective view. This function will draw perspective plots of a surface over the x–y plane.

Syntax: *persp(x, y, z)*
Parameter: *This function accepts different parameters i.e. x, y and z where x and y are vectors defining the location along x- and y-axis. z-axis will be the height of the surface in the matrix z.*
Return Value: *persp() returns the viewing transformation matrix for projecting 3D coo rdinates (x, y, z) into the 2D plane using homogeneous 4D coordinates (x, y, z, t).*

```
# Adding Titles and Labeling Axes to Plot
cone <- function(x, y){
sqrt(x ^ 2 + y ^ 2)
}
# prepare variables.
x <- y <- seq(-1, 1, length = 30)
z <- outer(x, y, cone)
# plot the 3D surface
# Adding Titles and Labeling Axes to Plot
persp(x, y, z,
main="Perspective Plot of a Cone",
zlab = "Height",
theta = 30, phi = 15,
col = "orange", shade = 0.4)
```

Output:



Questions:

1. Explain data pre-processing in R?
2. How to visualize the data in R?

Conclusion : Thus, I have implemented R Program to perform data pre-processing, analysis and visualization.

EXPERIMENT NO. 06

Title: Implement R program for Correlation and regression analysis.

Outcome: Students will be able to implement R program for Correlation and regression analysis.

Theory:

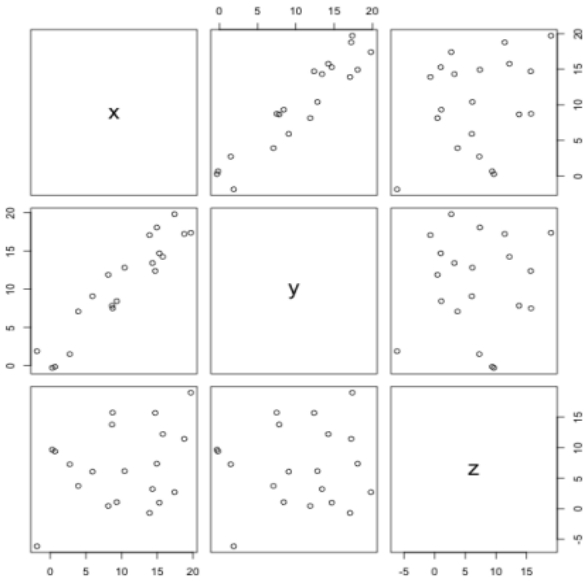
Correlation look at trends shared between two variables, and regression look at causal relation between a predictor (independent variable) and a response (dependent) variable.

Correlation

As mentioned above correlation look at global movement shared between two variables, for example when one variable increases and the other increases as well, then these two variables are said to be positively correlated. The other way round when a variable increase and the other decrease then these two variables are negatively correlated. In the case of no correlation no pattern will be seen between the two variable.
Let's look at some code before introducing correlation measure:

```
x<-sample(1:20,20)+rnorm(10,sd=2)
y<-x+rnorm(10,sd=3)
z<-(sample(1:20,20)/2)+rnorm(20,sd=5)
df<-data.frame(x,y,z)
plot(df[,1:3])
```

Here is the plot:



From the plot we get we see that when we plot the variable y with x, the points form some kind of line, when the value of x get bigger the value of y get somehow proportionally bigger too, we can suspect a positive correlation between x and y.

The measure of this correlation is called the coefficient of correlation and can calculated in different ways, the most usual measure is the Pearson coefficient, it is the covariance of the two variable divided by the product of their variance, it is scaled between 1 (for a perfect positive correlation) to -1 (for a perfect negative correlation), 0 would be complete randomness. We can get the Pearson coefficient of correlation using the function cor()

```
cor(df,method="pearson")
          x         y         z
x  1.0000000  0.8736874 -0.2485967
y  0.8736874  1.0000000 -0.2376243
z -0.2485967 -0.2376243  1.0000000

cor(df[,1:3],method="spearman")
          x         y         z
x  1.0000000  0.8887218 -0.3323308
y  0.8887218  1.0000000 -0.2992481
z -0.3323308 -0.2992481  1.0000000
```

From these outputs our suspicion is confirmed x and y have a high positive correlation, but as always in statistics we can test if this coefficient is significant. Using parametric assumptions (Pearson, dividing the coefficient by its standard error, giving a value that follow a t-distribution) or when data violate parametric assumptions using Spearman rank coefficient.

```
cor.test(df$x,df$y,method="pearson")

        Pearson's product-moment correlation

data:  df$x and df$y
t = 7.6194, df = 18, p-value = 4.872e-07
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7029411 0.9492172
sample estimates:
      cor
0.8736874

cor.test(df$x,df$y,method="spearman")
```

```
        Spearman's rank correlation rho

data:  df$x and df$y
S = 148, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.8887218
```

An extension of the Pearson coefficient of correlation is when we square it we obtain the amount of variation in y explained by x (this is not true for the spearman rank based coefficient where squaring it has no statistical meanings). In our case we have around 75% of the variance in y that is explained by x.

However such results do not allow any causal explanation of the effect of x on y, indeed x could act on y in various way that are not always direct, all we can say from the correlation is that these two variables are linked somehow, to really explain and measure causal effect of x on y we need to use regression method, which will come next.

Linear Regression

Regression is different from correlation because it try to put variables into equation and thus explain causal relationship between them, for example the most simple linear equation is written : Y=aX+b, so for every variation of unit in X, Y value change by aX. Because we are trying to explain natural processes by equations that represent only part of the whole picture we are actually building a model that's why linear regression are also called linear modelling.

In R we can build and test the significance of linear models.

```
m1<-lm(mpg~cyl,data=mtcars)
summary(m1)

Call:
lm(formula = mpg ~ cyl, data = mtcars)

Residuals:
   Min     1Q  Median     3Q    Max
-4.9814 -2.1185  0.2217  1.0717  7.5186

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.8846     2.0738   18.27  < 2e-16 ***
cyl          -2.8758     0.3224   -8.92 6.11e-10 ***
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.206 on 30 degrees of freedom
Multiple R-squared: 0.7262,     Adjusted R-squared: 0.7171
F-statistic: 79.56 on 1 and 30 DF,  p-value: 6.113e-10
```

The basic function to build linear model (linear regression) in R is to use the lm() function, you provide to it a formula in the form of y~x and optionally a data argument.

Using the summary() function we get all information about our model: the formula called, the distribution of the residuals (the error of our models), the value of the coefficient and their significance plus an information on the overall model performance with the adjusted R-squared (0,71 in our case) that represent the amount of variation in y explained by x, so 71% of the variation in ‘mpg’ can be explain by the variable ‘cyl’.

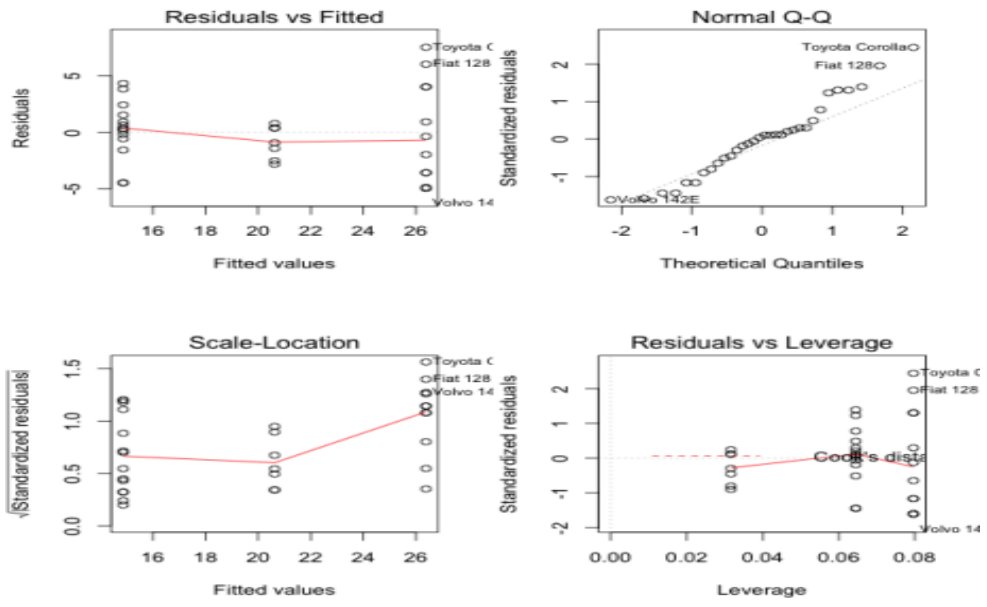
Before shouting ‘Eureka’ we should first check that the models assumptions are met, indeed linear models make a few assumptions on your data, the first one is that your data are normally distributed, the second one is that the variance in y is homogeneous over all x values (sometimes called homoscedasticity) and independence which means that a y value at a certain x value should not influence other y values.

There is a marvelous built-in methods to check all this with linear models:

```
par(mfrow=c(2,2))

plot(m1)
```

The par() argument is just to put all graphs in one window, the plot function is the real one.

Here is the plot:



The graphs on the first columns look at variance homogeneity among other things, normally you should see no pattern in the dots but just a random clouds of points. In this example this is clearly not the case since we see that the spreads of dots increase with higher values of cyl, our homogeneity assumptions is violated we can go back at the beginning and build new models this one cannot be interpreted… Sorry m1 you looked so great…

For the record the graph on the top right check the normality assumptions, if your data are normally distributed the point should fall (more or less) in a straight line, in this case the data are normal.

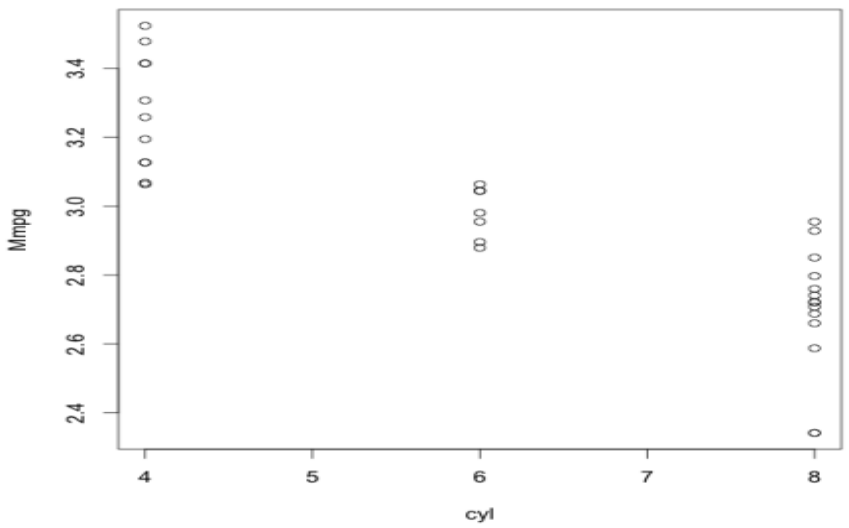
The final graph show how each y influence the model, each points is removed at a time and the new model is compared to the one with the point, if the point is very influential then it will have a high leverage value. Points with too high leverage value should be removed from the dataset to remove their outlying effect on the model.

Transforming the data

There are a few basics mathematical transformations that can be applied to non normal or heterogeneous data, usually it is a trial and error process;

```
mtcars$Mmpg<-log(mtcars$mpg)
plot(Mmpg~cyl,mtcars)
```

Here is the plot we get:



In our case this looks ok, but we can still remove the two outliers in ‘cyl’ categorie 8;

```
n<-rownames(mtcars)[mtcars$Mmpg!=min(mtcars$Mmpg[mtcars$cyl==8])]

mtcars2<-subset(mtcars,rownames(mtcars)%in%n)
```

The first line ask for row names in 'mtcars'

```
(rownames(mtcars))
```

, but only return the one where the value of the variable 'Mmpg' is not equal

```
(!=)
```

to the minimum value of the variable 'Mmpg' which fall in the category of 8 cylinders. Then the list 'n' contain all these rownames and the next step is to make a new data frame that only contain rows with rownames present in the list 'n'.

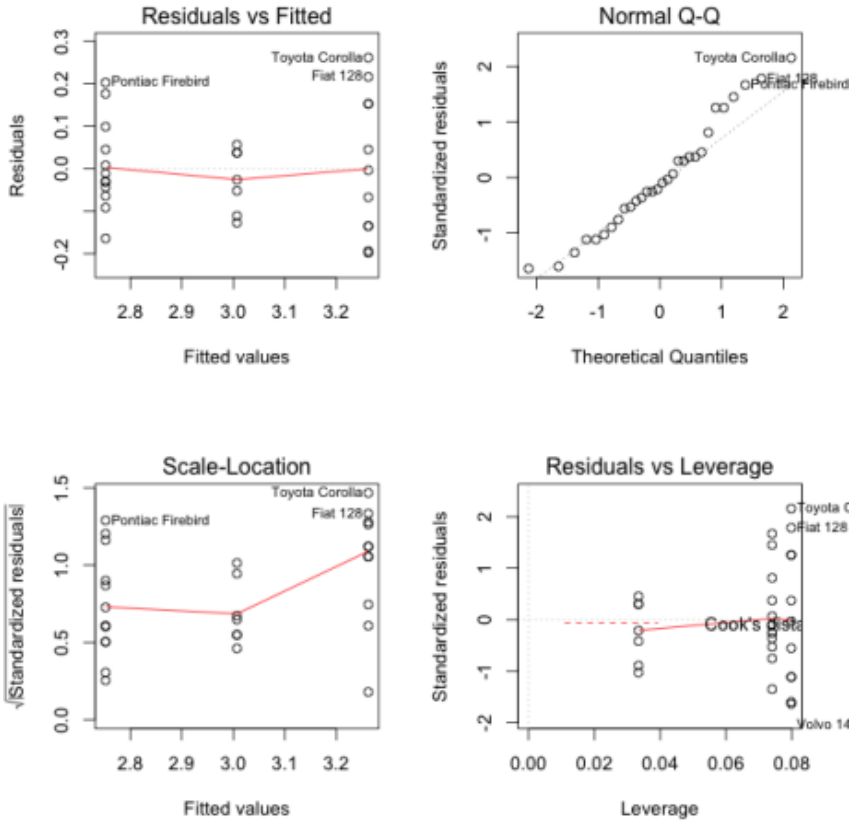
In this stage of transforming and removing outliers from the data you should use and abuse of plots to help you through the process.

Now let's look back at our bivariate linear regression model from this new dataset;

```
model<-lm(Mmpg~cyl,mtcars2)
summary(model)

Call:
lm(formula = Mmpg ~ cyl, data = mtcars2)

Residuals:
    Min      1Q  Median      3Q     Max
-0.19859 -0.08576 -0.01887  0.05354  0.26143

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.77183    0.08328  45.292  < 2e-16 ***
cyl         -0.12746    0.01319  -9.664 2.04e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1264 on 28 degrees of freedom
Multiple R-squared: 0.7693,     Adjusted R-squared: 0.7611
F-statistic: 93.39 on 1 and 28 DF,  p-value: 2.036e-10

plot(model)
```

Here is the plot for the

```
model
```



Again we have highly significant intercept and slope, the model explain 76% of the variance in log(mpg) and is overall significant. Now we biologist are trained to love and worship ANOVA table, in R there are several way to do it (as always an easy and straightforward way and another with more possibilities for tuning);

```
anova(model)

Analysis of Variance Table

Response: Mmpg
          Df  Sum Sq Mean Sq F value    Pr(>F)
cyl        1 1.49252 1.49252  93.393 2.036e-10 ***
Residuals 28 0.44747 0.01598

library(car)
Le chargement a nécessité le package : MASS
Le chargement a nécessité le package : nnet

Anova(model)
```

```
Anova Table (Type II tests)

Response: Mmpg
          Sum Sq Df F value    Pr(>F)
cyl       1.49252  1  93.393 2.036e-10 ***
Residuals 0.44747 28
```

The second function

```
Anova()
```

allow you to define which type of sum-of-square you want to calculate (here is a nice explanation of their different assumptions) and also to correct for variance heterogeneity;

```
Anova(model,white.adjust=TRUE)

Analysis of Deviance Table (Type II tests)

Response: Mmpg
          Df      F    Pr(>F)
cyl       1 69.328 4.649e-09 ***
Residuals 28
```

Questions:

1. What is Correlation?
2. Explain regression in detail?

Conclusion : Thus, I have implemented R program for Correlation and regression analysis.

EXPERIMENT NO. 07

Title: Data analysis using R for available data set. (Apply machine learning algorithm).

Outcome: Students will be able to perform Data analysis using R for available data set.

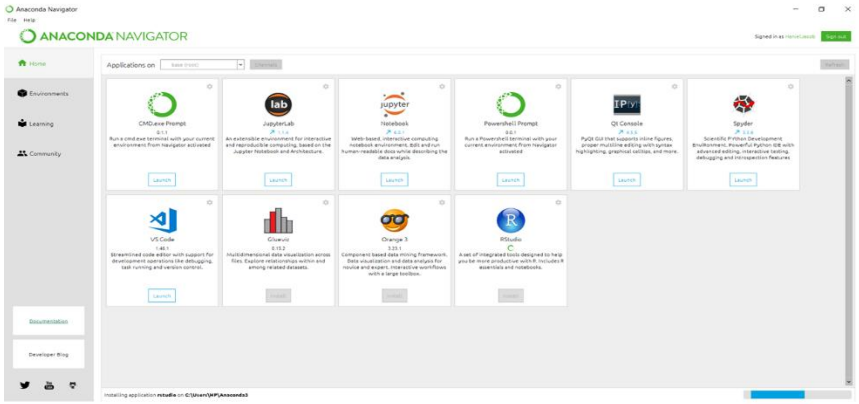
Theory:

Machine Learning is a subset of Artificial Intelligence (AI), which is used to create intelligent systems that are able to learn without being programmed explicitly. In machine learning, we create algorithms and models which is used by an intelligent system to predict outcomes based on particular patterns or trends which are observed from the given data. Machine learning follows a unique principle of using data and the outcomes from the data to predict the rules which are stored in a model. This model is then used to predict outcomes from a different set of data. In R programming the environment for machine learning can be set easily through RStudio.

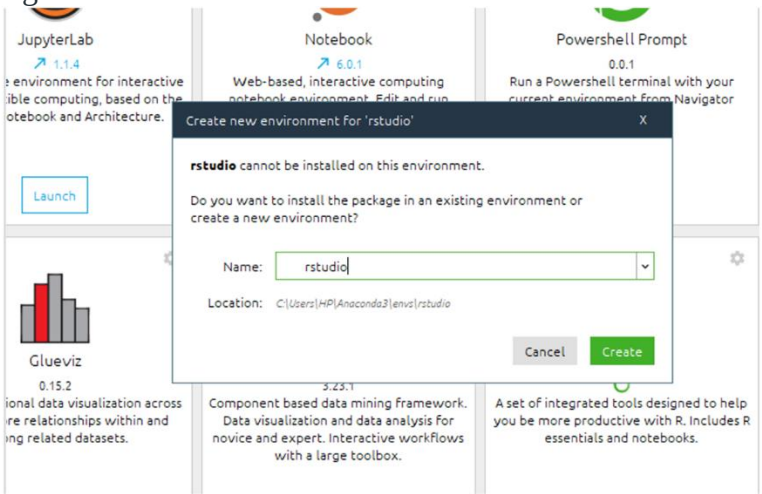
Setting up an environment for machine learning using Anaconda

Step 1: Install Anaconda (Linux, Windows) and launch the navigator.

Step 2: Open Anaconda Navigator and click the **Install** button for Rstudio.

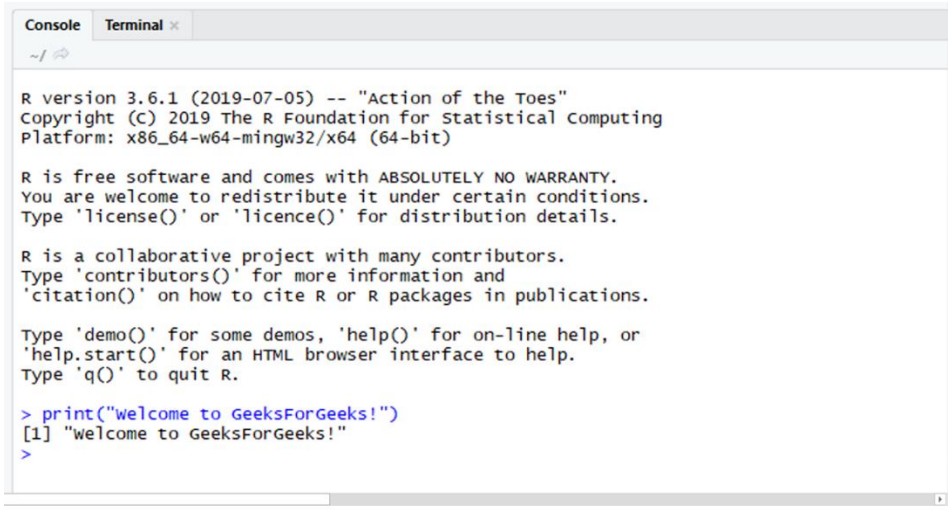


Step 3: After installation, create a new environment. Anaconda will then send a prompt asking to enter a name for the new environment and the lunch the R studio.



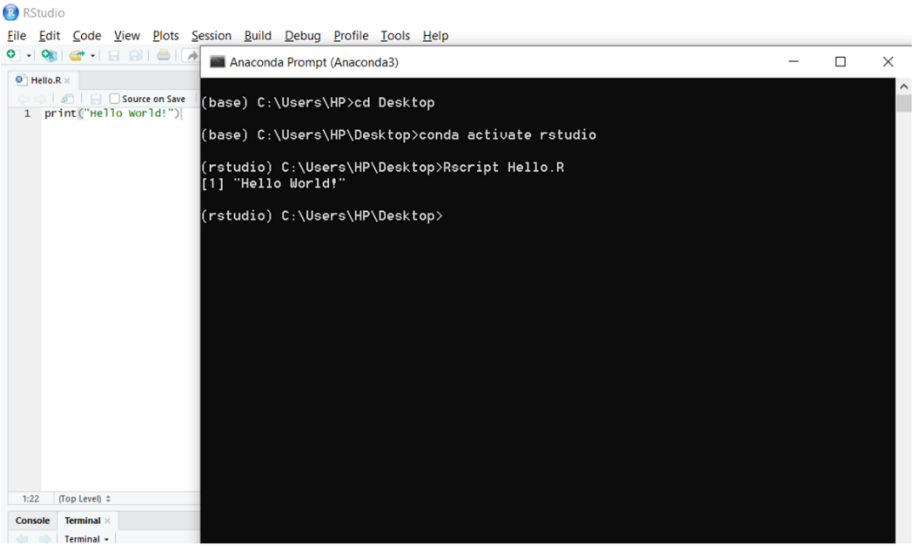
Running R commands

Method 1: R commands can run from the console provided in R studio. After opening Rstudio simply type R commands to the console.



Method 2: R commands can be stored in a file and can be executed in an anaconda prompt. This can be achieved by the following steps.

1. Open an anaconda prompt
2. Go to the directory where the R file is located
3. Activate the anaconda environment by using the command:
   conda activate <ENVIRONMENT_NAME>
4. Run the file by using the command:
   Rscript <FILE_NAME>.R

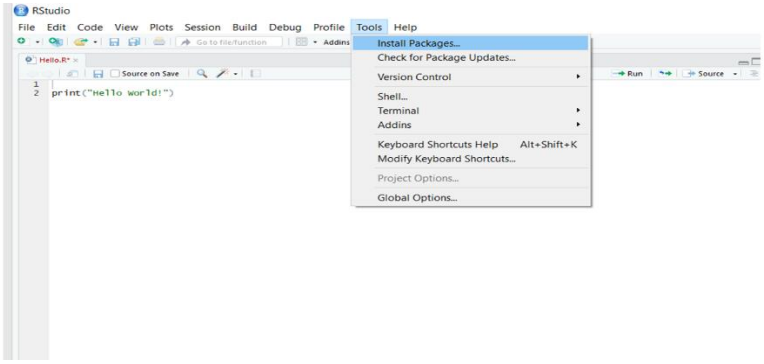


Installing machine learning packages in R

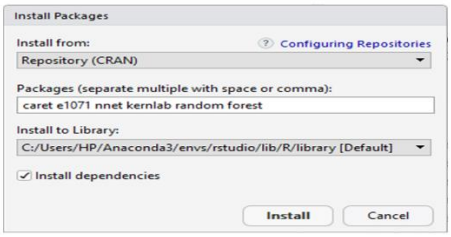
Packages help make code easier to write as they contain a set of predefined functions that perform various tasks. The most used machine learning packages are **Caret, e1071, net, kernlab, and randomforest**. There are two methods that can be used to install these packages for your R program.

Method 1: Installing Packages through Rstudio

1. Open Rstudio and click the **Install Packages** option under **Tools** which is present in the menubar.



2. Enter the names of all the packages you want to install separated by spaces or commas and then click install.



Method 2: Installing Packages through Anaconda prompt/Rstudio console

1. Open an Anaconda prompt.
2. Switch the environment to the environment you used for Rstudio using the command:
   conda activate <ENVIRONMENT_NAME>
3. Enter the command **r** to open the R console.
4. Install the required packages using the command:
   install.packages(c("<PACKAGE_1>", "<PACKAGE_2>", ..., "<PACKAGE_N>"))

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> install.packages(c("caret", "e1071", "nnet", "kernlab", "random forest"))
--- Please select a CRAN mirror for use in this session ---
trying URL 'https://cloud.r-project.org/bin/windows/contrib/3.6/caret_6.0-86.zip'
Content type 'application/zip' length 6259100 bytes (6.0 MB)
downloaded 6.0 MB

trying URL 'https://cloud.r-project.org/bin/windows/contrib/3.6/e1071_1.7-3.zip'
Content type 'application/zip' length 1022788 bytes (998 KB)
downloaded 998 KB

trying URL 'https://cloud.r-project.org/bin/windows/contrib/3.6/nnet_7.3-14.zip'
Content type 'application/zip' length 138022 bytes (134 KB)
downloaded 134 KB

trying URL 'https://cloud.r-project.org/bin/windows/contrib/3.6/kernlab_0.9-29.zip'
Content type 'application/zip' length 2641401 bytes (2.5 MB)
downloaded 2.5 MB

package 'caret' successfully unpacked and MD5 sums checked
package 'e1071' successfully unpacked and MD5 sums checked
package 'nnet' successfully unpacked and MD5 sums checked
package 'kernlab' successfully unpacked and MD5 sums checked
```

While downloading the packages you might be prompted to choose a **CRAN** mirror. It is recommended to choose the location closest to you for a faster download.

Machine Learning packages in R

There are many R libraries that contain a host of functions, tools, and methods to manage and analyze data. Each of these libraries has a particular focus with some libraries managing image and textual data, data manipulation, data visualization, web crawling, machine learning, and so on. Here let's discuss some of the important machine learning packages by demonstrating an example.

Example:

Preparing the Data Set:

Before using these packages first of all import the data set into RStudio, cleaning the data set, and split the data into train and test data set. Download the CSV file from this link.

```
# Import the data set
Data <- read.csv("GenderClassification.csv",
                 stringsAsFactors = TRUE)
# Using set.seed()
# Generating random number
set.seed(10)

# Cleaning the data set
Data$Favorite.Color <- as.numeric
                       (Data$Favorite.Color)
Data$Favorite.Music.Genre <- as.numeric
                       (Data$Favorite.Music.Genre)
Data$Favorite.Beverage <- as.numeric
                       (Data$Favorite.Beverage)
Data$Favorite.Soft.Drink <- as.numeric
                       (Data$Favorite.Soft.Drink)

# Split into train and test data set
TrainingSize <- createDataPartition(Data$Gender,
                                    p = 0.8,
                                    list = FALSE)
TrainingData <- Data[TrainingSize,]
TestingData <- Data[-TrainingSize,]
```

CARET: Caret stands for classification and regression training. The **CARET** package is used for performing classification and for regression tasks. It consists of many other built-in packages.

```
# Using CARET package

# Importing the library
library(caret)

# Using the train() available in
# Caret package
model <- train(Gender ~ ., data = TrainingData,
               method = "svmPoly",
               na.action = na.omit,
               preProcess = c("scale", "center"),
               trControl = trainControl(method = "none"),
               tuneGrid = data.frame(degree = 1,
                                     scale = 1,
                                     C = 1)
)
model.cv <- train(Gender ~ ., data = TrainingData,
                  method = "svmPoly",
                  na.action = na.omit,
                  preProcess = c("scale", "center"),
                  trControl = trainControl(method = "cv",
                                           number = 6),
                  tuneGrid = data.frame(degree = 1,
                                        scale = 1,
                                        C = 1)
)

# Printing the models
print(model)
print(model.cv)
```

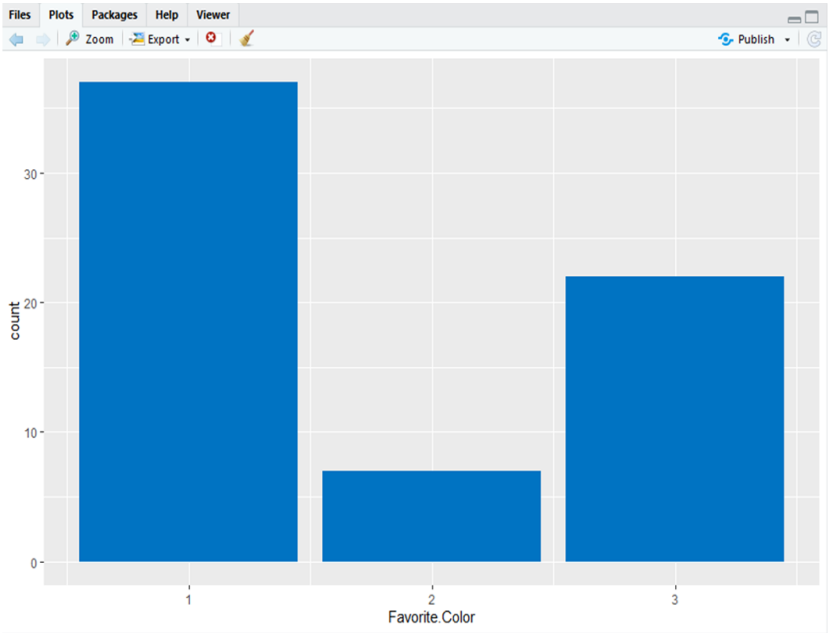
Output:

```
> print(model)
Support Vector Machines with Polynomial Kernel

54 samples
 4 predictor
 2 classes: 'F', 'M'

Pre-processing: scaled (4), centered (4)
Resampling: None
> print(model.cv)
Support Vector Machines with Polynomial Kernel

54 samples
 4 predictor
 2 classes: 'F', 'M'

Pre-processing: scaled (4), centered (4)
Resampling: Cross-Validated (6 fold)
Summary of sample sizes: 46, 44, 45, 45, 45, 45, ...
Resampling results:

  Accuracy   Kappa
  0.4259259  -0.1393453

Tuning parameter 'degree' was held constant at a value of 1
Tuning parameter 'scale' was held constant at a value of
 1
Tuning parameter 'C' was held constant at a value of 1
>
```

ggplot2: R is most famous for its visualization library ggplot2. It provides an aesthetic set of graphics that are also interactive. **The ggplot2** package is used for creating plots and for visualising data.

```
# Using ggplot2

# Creating a bar plot from the
# Data's Favorite.Color attribute
ggplot(Data, aes(Favorite.Color)) +
  geom_bar(fill = "#0073C2FF")
```

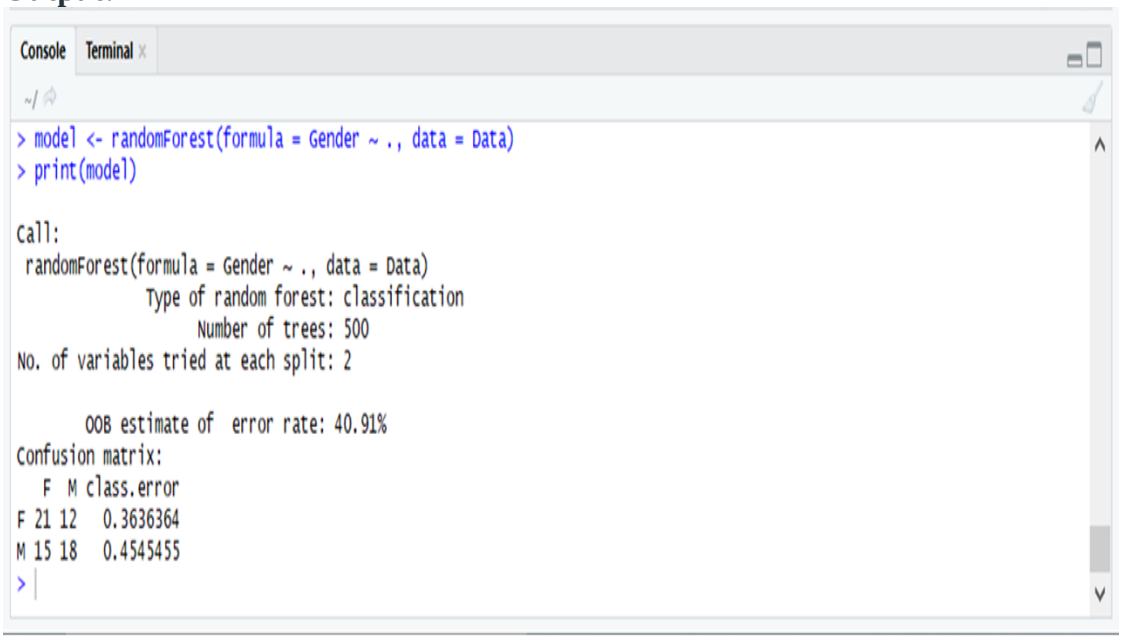
Output:



randomForest: The **randomForest** package allows us to use the random forest algorithm easily.

```
# Using randomforset

# Importing the randomForest package
library(randomForest)

# Using the randomForest function
# From the randomForest package
model <- randomForest(formula = Gender ~ .,
                      data = Data)
print(model)
```

Output:



nnet: The **nnet** package uses neural networks in deep learning to create layers which help in training and predicting models. The loss (the difference between the actual value and predicted value) decreases after every iteration of training.

```
# Using nnet

# Importing the nnet package
library(nnet)

# Using the nnet function
# In the nnet package
model <- nnet(formula = Gender ~ .,
              data = Data,
              size = 30)
print(model)
```

Output:

```
> model <- nnet(formula = Gender ~ ., data = Data, size = 30)
# weights:  181
initial  value 58.214998
iter  10 value 39.771566
iter  20 value 31.015013
iter  30 value 18.575582
iter  40 value 9.893588
iter  50 value 5.859889
iter  60 value 4.199062
iter  70 value 4.161215
iter  80 value 4.159124
iter  90 value 4.158951
iter 100 value 4.158895
final  value 4.158895
stopped after 100 iterations
> print(model)
a 4-30-1 network with 181 weights
inputs: Favorite.Color Favorite.Music.Genre Favorite.Beverage Favorite.Soft.Drink
output(s): Gender
options were - entropy fitting
>
```

e1071: The **e1071** package is used to implement the support vector machines, naive bayes algorithm and many other algorithms.

```
# Using e1071

# Importing the e1071 package
library(e1071)

# Using the svm function
# In the e1071 package
model <- svm(formula = Gender ~ .,
             data = Data)
print(model)
```

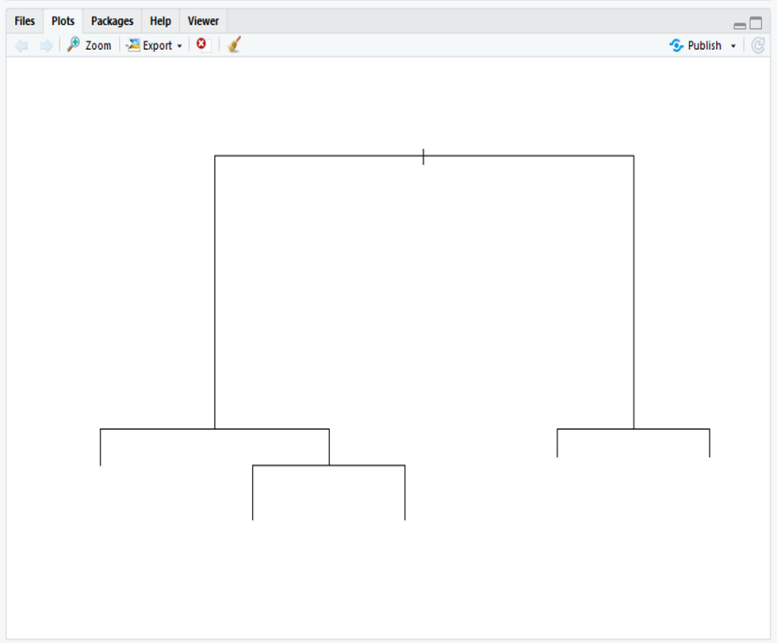
Output:

```
> model <- svm(formula = Gender ~ ., data = Data)
> print(model)

Call:
svm(formula = Gender ~ ., data = Data)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  57

>
```

rpart: The **rpart** package is used to partition data. It is used for classification and regression tasks. The resultant model is in the form of a binary tree.

```
# Using rpart

# Importing the rpart package
library(rpart)

# Using the rpart function
# To partition data
partition <- rpart(formula = Gender~.,
                   data = Data)
plot(partition)
```

Output:



dplyr: Like **rpart** the **dplyr** package is also a data manipulation package. It helps manipulate data by using functions such as **filter, select, and arrange.**

```
# Using dplyr

# Importing the dplyr package
library(dplyr)

# Using the filter function
# From the dplyr package
Data %>%
  filter(Gender == "M")
```

Output:

```
> Data %>%
+   filter( Gender == "M")
   Favorite.Color Favorite.Music.Genre Favorite.Beverage Favorite.Soft.Drink Gender
1               3                    6                 6                   4      M
2               2                    3                 1                   1      M
3               3                    1                 3                   2      M
4               2                    7                 2                   2      M
5               1                    5                 3                   3      M
6               1                    5                 5                   3      M
7               3                    7                 4                   1      M
8               1                    7                 4                   2      M
9               2                    5                 2                   1      M
10              3                    6                 2                   2      M
11              1                    7                 6                   1      M
12              1                    2                 1                   4      M
13              1                    3                 1                   2      M
14              1                    3                 6                   2      M
15              1                    6                 5                   1      M
16              1                    7                 2                   4      M
17              3                    3                 1                   2      M
18              1                    6                 2                   2      M
19              1                    7                 2                   2      M
20              1                    3                 2                   4      M
21              3                    7                 1                   3      M
22              1                    1                 2                   3      M
23              1                    1                 3                   3      M
24              3                    2                 3                   3      M
25              3                    1                 4                   3      M
26              3                    4                 4                   2      M
27              1                    5                 5                   4      M
28              1                    1                 5                   2      M
29              1                    7                 4                   2      M
30              1                    3                 1                   2      M
31              2                    3                 2                   3      M
32              1                    7                 6                   2      M
33              1                    1                 1                   2      M
>
```

Questions:

1. What is Machine Learning?
2. Explain different machine learning algorithms?

Conclusion : Thus, I have performed Data analysis using R for available data set.

EXPERIMENT NO. 08

Title: Implement a GNU Octave program for Solving Systems of Linear Equations.

Outcome: Students will be able to implement a GNU Octave program for Solving Systems of Linear Equations.

Theory:

GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language.

Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customizable via user-defined functions written in Octave's own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages.

GNU Octave is also freely redistributable software. You may redistribute it and/or modify it under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation.

Octave was written by John W. Eaton and many others. Because Octave is free software you are encouraged to help make Octave more useful by writing and contributing additional functions for it, and by reporting any problems you may have.

History

Octave was originally conceived (in about 1988) to be companion software for an undergraduate-level textbook on chemical reactor design being written by James B. Rawlings of the University of Wisconsin-Madison and John G. Ekerdt of the University of Texas. We originally envisioned some very specialized tools for the solution of chemical reactor design problems. Later, after seeing the limitations of that approach, we opted to attempt to build a much more flexible tool.

There were still some people who said that we should just be using Fortran instead, because it is the computer language of engineering, but every time we had tried that, the students spent far too much time trying to figure out why their Fortran code failed and not enough time learning about chemical engineering. We believed that with an interactive environment like Octave, most students would be able to pick up the basics quickly, and begin using it confidently in just a few hours.

Full-time development began in the Spring of 1992. The first alpha release was January 4, 1993, and version 1.0 was released February 17, 1994. Since then, Octave has been through several major revisions, is included with Debian GNU/Linux, openSUSE, and many other GNU/Linux distributions. Octave was reviewed in the July, 1997 issue of the Linux Journal.

Clearly, Octave is now much more than just another courseware package with limited utility beyond the classroom. Although our initial goals were somewhat vague, we knew that we wanted to create something that would enable students to solve realistic problems, and that they could use for many things other than chemical reactor design problems. Today, thousands of people worldwide are using Octave in teaching, research, and commercial applications.

Just about everyone thinks that the name Octave has something to do with music, but it is actually the name of one of the author's former professor Octave Levenspiel who wrote a famous textbook on chemical reaction engineering, and who was also well known for his ability to do quick "back of the envelope" calculations. We hope that this software will make it possible for many people to do more ambitious computations just as easily.

Everyone is encouraged to share this software with others under the terms of the GNU General Public License (GPL). You are also encouraged to help make Octave more useful by writing and contributing additional functions for it, and by reporting any problems you may have.

Installers for Microsoft Windows

The easiest way to install GNU Octave on Microsoft Windows is by using MXE builds. For the current release, both 32-bit and 64-bit installers and zip archived packages (.zip and .7z formats) can be found at https://www.gnu.org/software/octave/download.html under the Windows tab.

- For executable (.exe) installers: the user can simply run the downloaded file and follow the on-screen installation prompts. It is recommended that the installation path does not include spaces or non-ASCII characters. Shortcuts to the program will be created automatically.

- For the 7z/zip archives:

    1. Extract the file content to a directory on the harddrive (such as **C:\Octave**). Spaces or non-ASCII characters in the path are discouraged and may cause program errors.
    2. Manually create a shortcut to the **octave.vbs** file in the main installation directory. (Right-click on the file, select 'Create Shortcut', and move the new shortcut to your desired location.)
    3. If a command-line only instance of Octave is desired, the user can create another shortcut as stated above, right-click on the shortcut, select Properties, and add `--no-gui` to the end of the Target field.

4. IMPORTANT: Run the **post-install.bat** file before running Octave the first time to reduce plot delays due to the Windows font cache and make the pre-installed packages visible to the system.

- note: users who have problems running .vbs files due to local security policy or software can instead make a shortcut to the **octave.bat** file located in **\mingw32\bin** or **\mingw64\bin**, depending on the version of Octave installed.

Solving System of Equations in Octave

We have a little different approach to solve a system of 'n' linear equations in 'n' unknowns. Let us take up a simple example to demonstrate this use.

Let us solve the equations −

5x + 9y = 5

3x – 6y = 4

Such a system of linear equations can be written as the single matrix equation Ax = b, where A is the coefficient matrix, b is the column vector containing the right-hand side of the linear equations and x is the column vector representing the solution as shown in the below program −

Create a script file and type the following code –

```
A = [5, 9; 3, -6];
b = [5;4];
A \ b
```

When you run the file, it displays the following result −

```
ans =

   1.157895
  -0.087719
```

In same way, you can solve larger linear systems as given below −

x + 3y -2z = 5

3x + 5y + 6z = 7

2x + 4y + 3z = 8

Expanding and Collecting Equations in Octave

You need to have symbolic package, which provides expand and the collectfunction to expand and collect an equation, respectively. The following example demonstrates the concepts −

When you work with many symbolic functions, you should declare that your variables are symbolic but Octave has different approach to define symbolic variables. Notice the use of Sin and Cos, which are also defined in symbolic package.
Create a script file and type the following code –

```
% first of all load the package, make sure its installed.
pkg load symbolic

% make symbols module available
symbols

% define symbolic variables
x = sym ('x');
y = sym ('y');
z = sym ('z');

% expanding equations
expand((x-5)*(x+9))
expand((x+2)*(x-3)*(x-5)*(x+7))
expand(Sin(2*x))
expand(Cos(x+y))

% collecting equations
collect(x^3 *(x-7), z)
collect(x^4*(x-3)*(x-5), z)
```

When you run the file, it displays the following result –

```
ans =


-45.0+x^2+(4.0)*x
ans =


210.0+x^4-(43.0)*x^2+x^3+(23.0)*x
ans =


sin((2.0)*x)
ans =


cos(y+x)
ans =


x^(3.0)*(-7.0+x)
ans =


(-3.0+x)*x^(4.0)*(-5.0+x)
```

Factorization and Simplification of Algebraic Expressions

The factor function factorizes an expression and the simplify function simplifies an expression. The following example demonstrates the concept −

Example

Create a script file and type the following code –

```
syms x
syms y
factor(x^3 - y^3)
factor([x^2-y^2,x^3+y^3])
simplify((x^4-16)/(x^2-4))
```

When you run the file, it displays the following result −

```
ans =
 (x - y)*(x^2 + x*y + y^2)
 ans =
 [ (x - y)*(x + y), (x + y)*(x^2 - x*y + y^2)]
 ans =
 x^2 + 4
```

Questions:

1. What is GNU Octave?
2. How to solve system of linear equations?

Conclusion : Thus, I have implemented a GNU Octave program for Solving Systems of Linear Equations.

EXPERIMENT NO. 09

Title: Implement a GNU Octave program for Integrating Differential Equations and produce the Graphical Output of solution.

Outcome: Students will be able to implement a GNU Octave program for Integrating Differential Equations and produce the Graphical Output of solution.

Theory:

Integrating Differential Equations

Octave has built-in functions for solving nonlinear differential equations of the form.

For Octave to integrate equations of this form, you must first provide a definition of the function This is straightforward, and may be accomplished by entering the function body directly on the command line. For example, the following commands define the right hand side function for an interesting pair of nonlinear differential equations. Note that while you are entering a function, Octave responds with a different prompt, to indicate that it is waiting for you to complete your input.

```
octave:8> function xdot = f (x, t)
>
>  r = 0.25;
>  k = 1.4;
>  a = 1.5;
>  b = 0.16;
>  c = 0.9;
>  d = 0.8;
>
>  xdot(1) = r*x(1)*(1 - x(1)/k) - a*x(1)*x(2)/(1 + b*x(1));
>  xdot(2) = c*a*x(1)*x(2)/(1 + b*x(1)) - d*x(2);
>
> endfunction
```

Given the initial condition

```
x0 = [1; 2];
```

and the set of output times as a column vector (note that the first output time corresponds to the initial condition given above)

```
t = linspace (0, 50, 200)';
```

it is easy to integrate the set of differential equations:

```
x = lsode ("f", x0, t);
```

Producing Graphical Output

To display the solution of the previous example graphically, use the command

```
plot (t, x)
```

If you are using the X Window System, Octave will automatically create a separate window to display the plot. If you are using a terminal that supports some other graphics commands, you will need to tell Octave what kind of terminal you have. Type the command

```
gset term
```

to see a list of the supported terminal types. Octave uses gnuplot to display graphics, and can display graphics on any terminal that is supported by gnuplot.

To capture the output of the plot command in a file rather than sending the output directly to your terminal, you can use a set of commands like this

```
gset term postscript
gset output "foo.ps"
replot
```

This will work for other types of output devices as well. Octave's gset command is really just piped to the gnuplot subprocess, so that once you have a plot on the screen that you like, you should be able to do something like this to create an output file suitable for your graphics printer.

Or, you can eliminate the intermediate file by using commands like this

```
gset term postscript
gset output "|lpr -Pname_of_your_graphics_printer"
replot
```

Questions:

1. What are Differential Equations?
2. How to plot graphs in Octave?

Conclusion : Thus, I have implemented a GNU Octave program for Integrating Differential Equations and produce the Graphical Output of solution.

EXPERIMENT NO. 10

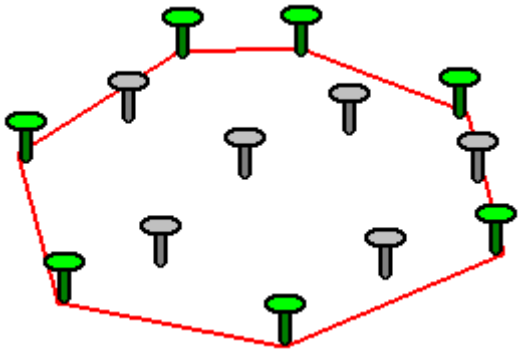
Title: Implement a GNU Octave program for solving convex hull problem and produce the Graphical Output of solution.

Outcome: Students will be able to implement a GNU Octave program for solving convex hull problem and produce the Graphical Output of solution.

Theory:

The convex hull is a ubiquitous structure in computational geometry. Even though it is a useful tool in its own right, it is also helpful in constructing other structures like Voronoi diagrams, and in applications like unsupervised image analysis.

We can visualize what the convex hull looks like by a thought experiment. Imagine that the points are nails sticking out of the plane, take an elastic rubber band, stretch it around the nails and let it go. It will snap around the nails and assume a shape that minimizes its length. The area enclosed by the rubber band is called the convex hull of P*P*. This leads to an alternative definition of the convex hull of a finite set P*P* of points in the plane: it is the unique convex polygon whose vertices are points from P*P* and which contains all points of P*P*.

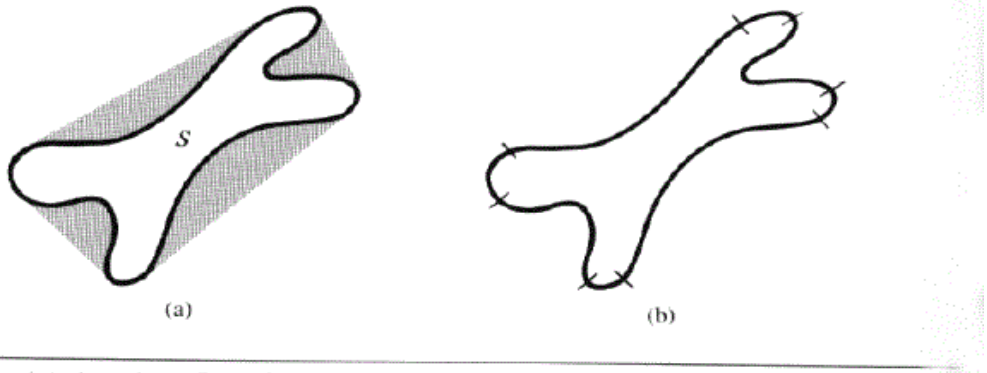


Applications

A few of the applications of the convex hull are:

- **Collision avoidance**: If the convex hull of a car avoids collision with obstacles then so does the car. Since the computation of paths that avoid collision is much easier with a convex car, then it is often used to plan paths.

- **Smallest box**: The smallest area rectangle that encloses a polygon has at least one side flush with the convex hull of the polygon, and so the hull is computed at the first step of minimum rectangle algorithms. Similarly, finding the smallest three-dimensional box surrounding an object depends on the 3D-convex hull.
- **Shape analysis**: Shapes may be classified for the purposes of matching by their "convex deficiency trees", structures that depend for their computation on convex hulls.



(a) A region (S) and its convex deficiency (shaded); (b) partitioned boundary.

Convex Hull

The convex hull of a set of points is the minimum convex envelope containing all of the points. Octave has the functions *convhull* and *convhulln* to calculate the convex hull of 2-dimensional and N-dimensional sets of points.

Function File: *H* = **convhull** *(x, y)*
Function File: *H* = **convhull** *(x, y, options)*

Compute the convex hull of the set of points defined by the arrays *x* and *y*. The hull *H* is an index vector into the set of points and specifies which points form the enclosing hull.

An optional third argument, which must be a string or cell array of strings, contains options passed to the underlying qhull command. See the documentation for the Qhull library for details http://www.qhull.org/html/qh-quick.htm#options. The default option is *{"Qt"}*.

If *options* is not present or *[]* then the default arguments are used. Otherwise, *options* replaces the default argument list. To append user options to the defaults it is necessary to repeat the default arguments in *options*. Use a null string to pass no arguments.

Loadable Function: *h* = **convhulln** *(pts)*
Loadable Function: *h* = **convhulln** *(pts, options)*
Loadable Function: *[h, v]* = **convhulln** *(...)*

Compute the convex hull of the set of points *pts*.

pts is a matrix of size [n, dim] containing n points in a space of dimension dim.

The hull *h* is an index vector into the set of points and specifies which points form the enclosing hull.

An optional second argument, which must be a string or cell array of strings, contains options passed to the underlying qhull command. See the documentation for the Qhull library for details http://www.qhull.org/html/qh-quick.htm#options. The default options depend on the dimension of the input:

- 2D, 3D, 4D: *options = {"Qt"}*
- 5D and higher: *options = {"Qt", "Qx"}*

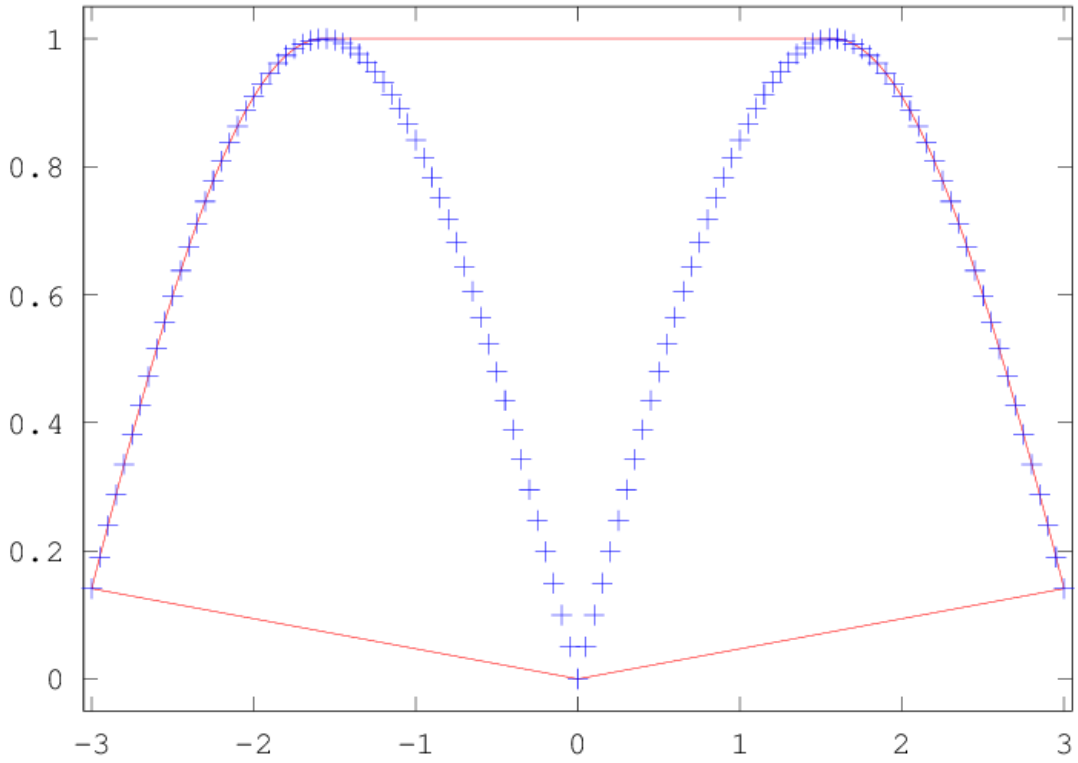
If *options* is not present or *[]* then the default arguments are used. Otherwise, *options* replaces the default argument list. To append user options to the defaults it is necessary to repeat the default arguments in *options*. Use a null string to pass no arguments.

If the second output *v* is requested the volume of the enclosing convex hull is calculated.

An example of the use of *convhull* is

```
x = -3:0.05:3;

y = abs (sin (x));

k = convhull (x, y);

plot (x(k), y(k), "r-", x, y, "b+");

axis ([-3.05, 3.05, -0.05, 1.05]);
```

The output of the above can be seen in



Questions:

1. What is Convex Hull Problem?
2. Explain different algorithms for solving Convex Hull Problem?

Conclusion : Thus, I have implemented a GNU Octave program for solving convex hull problem and produce the Graphical Output of solution.

EXPERIMENT NO. 11

Title: Implement a GNU Octave program to apply Image Processing functions like load, display, represent, plot and color.

Outcome: Students will be able to implement a GNU Octave program to apply Image Processing functions like load, display, represent, plot and color.

Theory:

Image Processing

Since an image is basically a matrix, Octave is a very powerful environment for processing and analyzing images. To illustrate how easy it is to do image processing in Octave, the following example will load an image, smooth it by a 5-by-5 averaging filter, and compute the gradient of the smoothed image.

```
I = imread ("myimage.jpg");
S = conv2 (I, ones (5, 5) / 25, "same");
[Dx, Dy] = gradient (S);
```

Loading and Saving Images

The first step in most image processing tasks is to load an image into Octave which is done with the *imread* function. The *imwrite* function is the corresponding function for writing images to the disk.

In summary, most image processing code will follow the structure of this code

```
I = imread ("my_input_image.img");

J = process_my_image (I);

imwrite (J, "my_output_image.img");
```

: *[img, map, alpha]* = **imread** *(filename)*
: *[...]* = **imread** *(url)*
: *[...]* = **imread** *(..., ext)*
: *[...]* = **imread** *(..., idx)*
: *[...]* = **imread** *(..., param1, value1, ...)*

Read images from various file formats.

Read an image as a matrix from the file *filename* or from the online resource *url*. If neither is given, but *ext* was specified, look for a file with the extension *ext*.

The size and class of the output depends on the format of the image. A color image is returned as an MxNx3 matrix. Grayscale and black-and-white images are of size MxN. Multipage images will have an additional 4th dimension.

The bit depth of the image determines the class of the output: *"uint8"*, *"uint16"*, or *"single"* for grayscale and color, and *"logical"* for black-and-white. Note that indexed images always return the indexes for a colormap, independent of whether *map* is a requested output. To obtain the actual RGB image, use *ind2rgb*. When more than one indexed image is being read, *map* is obtained from the first. In some rare cases this may be incorrect and *imfinfo* can be used to obtain the colormap of each image.

Some file formats, such as TIFF and GIF, are able to store multiple images in a single file. *idx* can be a scalar or vector specifying the index of the images to read. By default, Octave will read only the first page.

Depending on the file format, it is possible to configure the reading of images with *parameter*, *value* pairs. The following options are supported:

"Frames" or *"Index"*

This is an alternative method to specify *idx*. When specifying it in this way, its value can also be the string *"all"*.

"Info"

This option exists for MATLAB compatibility, but has no effect. For maximum performance when reading multiple images from a single file, use the *"Index"* option.

"PixelRegion"

Controls the image region that is read. The value must be a cell array with two arrays of 3 elements *{[rows], [cols]}*. The elements in the array are the start, increment, and end pixel to be read. If the increment value is omitted it defaults to 1. For example, the following are all equivalent:

```
imread (filename, "PixelRegion", {[200 600], [300 700]});

imread (filename, "PixelRegion", {[200 1 600], [300 1 700]});

imread (filename)(200:600, 300:700);
```

: **imwrite** *(img, filename)*
: **imwrite** *(img, filename, ext)*
: **imwrite** *(img, map, filename)*
: **imwrite** *(…, param1, val1, …)*

Write images in various file formats.

The image *img* can be a binary, grayscale, RGB, or multi-dimensional image. The size and class of *img* should be the same as what should be expected when reading it

with *imread*: the 3rd and 4th dimensions reserved for color space, and multiple pages respectively. If it's an indexed image, the colormap *map* must also be specified.

If *ext* is not supplied, the file extension of *filename* is used to determine the format. The actual supported formats are dependent on options made during the build of Octave. Use *imformats* to check the support of the different image formats.

Depending on the file format, it is possible to configure the writing of images with *param*, *val* pairs. The following options are supported:

'*Alpha*'

Alpha (transparency) channel for the image. This must be a matrix with same class, and number of rows and columns of *img*. In case of a multipage image, the size of the 4th dimension must also match and the third dimension must be a singleton. By default, image will be completely opaque.

'*Compression*'

Compression to use one the image. Can be one of the following: "none" (default), "bzip", "fax3", "fax4", "jpeg", "lzw", "rle", or "deflate". Note that not all compression types are available for all image formats in which it defaults to your Magick library.

'*DelayTime*'

For formats that accept animations (such as GIF), controls for how long a frame is displayed until it moves to the next one. The value must be scalar (which will applied to all frames in *img*), or a vector of length equal to the number of frames in *im*. The value is in seconds, must be between 0 and 655.35, and defaults to 0.5.

'*DisposalMethod*'

For formats that accept animations (such as GIF), controls what happens to a frame before drawing the next one. Its value can be one of the following strings: "doNotSpecify" (default); "leaveInPlace"; "restoreBG"; and "restorePrevious", or a cell array of those string with length equal to the number of frames in *img*.

'*LoopCount*'

For formats that accept animations (such as GIF), controls how many times the sequence is repeated. A value of Inf means an infinite loop (default), a value of 0 or 1 that the sequence is played only once (loops zero times), while a value of 2 or above loops that number of times (looping twice means it plays the complete sequence 3 times). This option is ignored when there is only a single image at the end of writing the file.

'*Quality*'

Set the quality of the compression. The value should be an integer between 0 and 100, with larger values indicating higher visual quality and lower compression. Defaults to 75.

'*WriteMode*'

Some file formats, such as TIFF and GIF, are able to store multiple images in a single file. This option specifies if *img* should be appended to the file (if it exists) or if a new file should be created for it (possibly overwriting an existing file). The value should be the string *"Overwrite"* (default), or *"Append"*.

Despite this option, the most efficient method of writing a multipage image is to pass a 4 dimensional *img* to *imwrite*, the same matrix that could be expected when using *imread* with the option *"Index"* set to *"all"*.

Displaying Images

A natural part of image processing is visualization of an image. The most basic function for this is the *imshow* function that shows the image given in the first input argument.

: **imshow** *(im)*
: **imshow** *(im, limits)*
: **imshow** *(im, map)*
: **imshow** *(rgb, ...)*
: **imshow** *(filename)*
: **imshow** *(..., string_param1, value1, ...)*
: *h* = **imshow** *(...)*

Display the image *im*, where *im* can be a 2-dimensional (grayscale image) or a 3-dimensional (RGB image) matrix.

If *limits* is a 2-element vector *[low, high]*, the image is shown using a display range between *low* and *high*. If an empty matrix is passed for *limits*, the display range is computed as the range between the minimal and the maximal value in the image.

If *map* is a valid color map, the image will be shown as an indexed image using the supplied color map.

If a filename is given instead of an image, the file will be read and shown.

If given, the parameter *string_param1* has value *value1*. *string_param1* can be any of the following:

"displayrange"

value1 is the display range as described above.

"colormap"

value1 is the colormap to use when displaying an indexed image.

"xdata"

If *value1* is a 2-element vector, it must contain horizontal image limits in the form [xfirst, xlast], where xfirst and xlast are the abscissa of the centers of the corner pixels.

Otherwise *value1* must be a vector and only the first and last elements will be used for xfirst and xlast respectively.

"ydata"

If *value1* is a 2-element vector, it must contain vertical image limits in the form [yfirst, ylast], where yfirst and ylast are the ordinates of the center of the corner pixels. Otherwise *value1* must be a vector and only the first and last elements will be used for yfirst and ylast respectively.

The optional return value *h* is a graphics handle to the image.

: **image** *(img)*
: **image** *(x, y, img)*
: **image** *(..., "prop", val, ...)*
: **image** *("prop1", val1, ...)*
: *h =* **image** *(...)*

Display a matrix as an indexed color image.

The elements of *img* are indices into the current colormap.

x and *y* are optional 2-element vectors, *[min, max]*, which specify the coordinates of the centers of the corner pixels. If a range is specified as *[max, min]* then the image will be reversed along that axis. For convenience, *x* and *y* may be specified as N-element vectors matching the length of the data in *img*. However, only the first and last elements will be used to determine the axis limits.

Multiple property/value pairs may be specified for the image object, but they must appear in pairs.

The optional return value *h* is a graphics handle to the image.

Implementation Note: The origin (0, 0) for images is located in the upper left. For ordinary plots, the origin is located in the lower left. Octave handles this inversion by plotting the data normally, and then reversing the direction of the y-axis by setting the *ydir* property to *"reverse"*. This has implications whenever an image and an ordinary plot need to be overlaid. The recommended solution is to display the image and then plot the reversed ydata using, for example, *flipud (ydata)*.

Calling Forms: The *image* function can be called in two forms: High-Level and Low-Level. When invoked with normal options, the High-Level form is used which first calls *newplot* to prepare the graphic figure and axes. When the only inputs to *image* are property/value pairs the Low-Level form is used which creates a new instance of an image object and inserts it in the current axes.

Representing Images

In general Octave supports four different kinds of images, grayscale images, RGB images, binary images, and indexed images. A grayscale image is represented with an M-by-N matrix in which each element corresponds to the intensity of a pixel. An RGB image is represented with an M-by-N-by-3 array where each 3-vector corresponds to the red, green, and blue intensities of each pixel.

The actual meaning of the value of a pixel in a grayscale or RGB image depends on the class of the matrix. If the matrix is of class *double* pixel intensities are between 0 and 1, if it is of class *uint8* intensities are between 0 and 255, and if it is of class *uint16* intensities are between 0 and 65535.

A binary image is an M-by-N matrix of class *logical*. A pixel in a binary image is black if it is *false* and white if it is *true*.

An indexed image consists of an M-by-N matrix of integers and a C-by-3 color map. Each integer corresponds to an index in the color map, and each row in the color map corresponds to an RGB color. The color map must be of class *double* with values between 0 and 1.

The following convenience functions are available for conversion between image formats.

: **im2double** *(img)*
: **im2double** *(img, "indexed")*

Convert image to double precision.

The conversion of *img* to double precision, is dependent on the type of input image. The following input classes are supported:

‘*uint8, uint16, and int16*’

The range of values from the class is scaled to the interval [0 1].

‘*logical*’

True and false values are assigned a value of 0 and 1 respectively.

‘*single*’

Values are cast to double.

‘*double*’

Returns the same image.

If *img* is an indexed image, then the second argument should be the string *"indexed"*. If so, then *img* must either be of floating point class, or unsigned integer class and it will simply be cast to double. If it is an integer class, a +1 offset is applied.

: *img* = **gray2ind** *(I)*
: *img* = **gray2ind** *(I, n)*
: *img* = **gray2ind** *(BW)*
: *img* = **gray2ind** *(BW, n)*
: *[img, map]* = **gray2ind** *(...)*

Convert a grayscale or binary intensity image to an indexed image.

The indexed image will consist of *n* different intensity values. If not given *n* defaults to 64 for grayscale images or 2 for binary black and white images.

The output *img* is of class uint8 if *n* is less than or equal to 256; Otherwise the return class is uint16.

Octave also provides tools to produce and work with movie frame structures. Those structures encapsulate the image data (*"cdata"* field) together with the corresponding colormap (*"colormap"* field).

: *frame* = **getframe** *()*
: *frame* = **getframe** *(hax)*
: *frame* = **getframe** *(hfig)*
: *frame* = **getframe** *(..., rect)*

Capture a figure or axes as a movie frame structure.

Without an argument, capture the current axes excluding ticklabels, title, and x/y/zlabels. The returned structure *frame* has a field *cdata*, which contains the actual image data in the form of an NxMx3 (RGB) uint8 matrix, and a field *colormap* which is provided for MATLAB compatibility but is always empty.

If the first argument *hax* is an axes handle, then capture this axes, rather than the current axes returned by *gca*.

If the first argument *hfig* is a figure handle then the entire corresponding figure canvas is captured.

Finally, if a second argument *rect* is provided it must be a four-element vector ([left bottom width height]) defining the region inside the figure to be captured. Regardless of the figure *"units"* property, *rect* must be defined in **pixels**.

Plotting on top of Images

If gnuplot is being used to display images it is possible to plot on top of images. Since an image is a matrix it is indexed by row and column values. The plotting system is,

however, based on the traditional *(x, y)* system. To minimize the difference between the two systems Octave places the origin of the coordinate system in the point corresponding to the pixel at *(1, 1)*. So, to plot points given by row and column values on top of an image, one should simply call *plot* with the column values as the first argument and the row values as the second. As an example the following code generates an image with random intensities between 0 and 1, and shows the image with red circles over pixels with an intensity above *0.99*.

```
I = rand (100, 100);

[row, col] = find (I > 0.99);

hold ("on");

imshow (I);

plot (col, row, "ro");

hold ("off");
```

Color Conversion

Octave supports conversion from the RGB color system to the HSV color system and vice versa. It is also possible to convert from a color RGB image to a grayscale image.

: *hsv_map* = **rgb2hsv** *(rgb_map)*
: *hsv_img* = **rgb2hsv** *(rgb_img)*

Transform a colormap or image from RGB to HSV color space.

A color in the RGB space consists of red, green, and blue intensities.

A color in HSV space is represented by hue, saturation and value (brightness) levels in a cylindrical coordinate system. Hue is the azimuth and describes the dominant color. Saturation is the radial distance and gives the amount of hue mixed into the color. Value is the height and is the amount of light in the color.

Output class and size will be the same as input.

: *rgb_map* = **hsv2rgb** *(hsv_map)*
: *rgb_img* = **hsv2rgb** *(hsv_img)*

Transform a colormap or image from HSV to RGB color space.

A color in HSV space is represented by hue, saturation and value (brightness) levels in a cylindrical coordinate system. Hue is the azimuth and describes the dominant color. Saturation is the radial distance and gives the amount of hue mixed into the color. Value is the height and is the amount of light in the color.

The input can be both a colormap or RGB image. In the case of floating point input, values are expected to be on the [0 1] range. In the case of hue (azimuth), since the value corresponds to an angle, *mod (h, 1)* is used.

```
>> hsv2rgb ([0.5 1 1])

⇒ ans = 0 1 1


>> hsv2rgb ([2.5 1 1])

⇒ ans = 0 1 1


>> hsv2rgb ([3.5 1 1])

⇒ ans = 0 1 1
```

Output class and size will be the same as input.

: *I* = **rgb2gray** *(rgb_img)*
: *gray_map* = **rgb2gray** *(rgb_map)*

Transform an image or colormap from red-green-blue (RGB) color space to a grayscale intensity image.

The input may be of class uint8, int8, uint16, int16, single, or double. The output is of the same class as the input.

Implementation Note: The grayscale intensity is calculated as

$$I = 0.298936*R + 0.587043*G + 0.114021*B$$

Questions:

1. Explain different Octave functions for displaying images?
2. Explain different color conversion Octave functions?

Conclusion : Thus, I have implemented a GNU Octave program to apply Image Processing functions like load, display, represent, plot and color.

EXPERIMENT NO. 12

Title: Implement a GNU Octave program to apply Audio Processing functions like record, retrieval, and audio data processing.

Outcome: Students will be able to implement a GNU Octave program to apply Audio Processing functions like record, retrieval, and audio data processing.

Theory:
Audio File Utilities

The following functions allow you to read, write and retrieve information about audio files. Various formats are supported including wav, flac and ogg vorbis.

: *info* = **audioinfo** *(filename)*

Return information about an audio file specified by *filename*.

The output *info* is a structure containing the following fields:

'*Filename*'

Name of the audio file.

'*CompressionMethod*'

Audio compression method. Unused, only present for compatibility with MATLAB.

'*NumChannels*'

Number of audio channels.

'*SampleRate*'

Sample rate of the audio, in Hertz.

'*TotalSamples*'

Number of samples in the file.

'*Duration*'

Duration of the audio, in seconds.

'*BitsPerSample*'

Number of bits per sample.

'*BitRate*'

Audio bit rate. Unused, only present for compatibility with MATLAB.

'*Title*'

"Title" audio metadata value as a string, or empty if not present.

'*Artist*'

"Artist" audio metadata value as a string, or empty if not present.

'*Comment*'

"Comment" audio metadata value as a string, or empty if not present.

: *[y, fs] =* **audioread** *(filename)*
: *[y, fs] =* **audioread** *(filename, samples)*
: *[y, fs] =* **audioread** *(filename, datatype)*
: *[y, fs] =* **audioread** *(filename, samples, datatype)*

Read the audio file *filename* and return the audio data *y* and sampling rate *fs*.

The audio data is stored as matrix with rows corresponding to audio frames and columns corresponding to channels.

The optional two-element vector argument *samples* specifies starting and ending frames.

The optional argument *datatype* specifies the datatype to return. If it is *"native"*, then the type of data depends on how the data is stored in the audio file.

: **audiowrite** *(filename, y, fs)*
: **audiowrite** *(filename, y, fs, name, value, ...)*

Write audio data from the matrix *y* to *filename* at sampling rate *fs* with the file format determined by the file extension.

Additional name/value argument pairs may be used to specify the following options:

'*BitsPerSample*'

Number of bits per sample. Valid values are 8, 16, 24, and 32. Default is 16.

'*BitRate*'

Valid argument name, but ignored. Left for compatibility with MATLAB.

'*Quality*'

Quality setting for the Ogg Vorbis compressor. Values can range between 0 and 100 with 100 being the highest quality setting. Default is 75.

'*Title*'

Title for the audio file.

'*Artist*'

Artist name.

'*Comment*'

Comment.

Audio Device Information

: *devinfo* = **audiodevinfo** *()*
: *devs* = **audiodevinfo** *(io)*
: *name* = **audiodevinfo** *(io, id)*
: *id* = **audiodevinfo** *(io, name)*
: *driverversion* = **audiodevinfo** *(io, id, "DriverVersion")*
: *id* = **audiodevinfo** *(io, rate, bits, chans)*
: *supports* = **audiodevinfo** *(io, id, rate, bits, chans)*

Return a structure describing the available audio input and output devices.

The *devinfo* structure has two fields *"input"* and *"output"*. The value of each field is a structure array with fields *"Name"*, *"DriverVersion"* and *"ID"* describing an audio device.

If the optional argument *io* is 1, return information about input devices only. If it is 0, return information about output devices only. If *io* is the only argument supplied, return the number of input or output devices available.

If the optional argument *id* is provided, return information about the corresponding device.

If the optional argument *name* is provided, return the ID of the named device.

If the optional argument *"DriverVersion"* is given, return the name of the driver for the specified device.

Given a sampling rate, bits per sample, and number of channels for an input or output device, return the ID of the first device that supports playback or recording using the specified parameters.

If also given a device ID, return true if the device supports playback or recording using those parameters.

Audio Player

The following methods are used to create and use audioplayer objects. These objects can be used to play back audio data stored in Octave matrices and arrays. The audioplayer

object supports playback from various devices available to the system, blocking and non-blocking playback, convenient pausing and resuming and much more.

: *player* = **audioplayer** *(y, fs)*
: *player* = **audioplayer** *(y, fs, nbits)*
: *player* = **audioplayer** *(y, fs, nbits, id)*
: *player* = **audioplayer** *(recorder)*
: *player* = **audioplayer** *(recorder, id)*

Create an audioplayer object that will play back data *y* at sample rate *fs*.

The optional arguments *nbits*, and *id* specify the bit depth and player device id, respectively. Device IDs may be found using the audiodevinfo function. Given an audioplayer object, use the data from the object to initialize the player.

The signal *y* can be a vector or a two-dimensional array.

The following example will create an audioplayer object that will play back one second of white noise at 44100 sample rate using 8 bits per sample.

```
y = 0.25 * randn (2, 44100);

player = audioplayer (y, 44100, 8);

play (player);
```

Audio Recorder

The following methods are used to create and use audiorecorder objects. These objects can be used to record audio data from various devices available to the system. You can use convenient methods to retrieve that data or audioplayer objects created from that data. Methods for blocking and non-blocking recording, pausing and resuming recording and much more is available.

: *recorder* = **audiorecorder** *()*
: *recorder* = **audiorecorder** *(fs, nbits, channels)*
: *recorder* = **audiorecorder** *(fs, nbits, channels, id)*

Create an audiorecorder object recording 8 bit mono audio at 8000 Hz sample rate.

The optional arguments *fs*, *nbits*, *channels*, and *id* specify the sample rate, bit depth, number of channels and recording device id, respectively. Device IDs may be found using the audiodevinfo function.

Audio Data Processing

Octave provides a few functions for dealing with audio data. An audio 'sample' is a single output value from an A/D converter, i.e., a small integer number (usually 8 or 16 bits), and audio data is just a series of such samples. It can be characterized by three parameters: the sampling rate (measured in samples per second or Hz, e.g., 8000 or 44100), the number of bits per sample (e.g., 8 or 16), and the number of channels (1 for mono, 2 for stereo, etc.).

There are many different formats for representing such data. Currently, only the two most popular, *linear encoding* and *mu-law encoding*, are supported by Octave. There is an excellent FAQ on audio formats by Guido van Rossum guido@cwi.nl which can be found at any FAQ ftp site, in particular in the directory */pub/usenet/news.answers/audio-fmts* of the archive site *rtfm.mit.edu*.

Octave simply treats audio data as vectors of samples (non-mono data are not supported yet). It is assumed that audio files using linear encoding have one of the extensions *lin* or *raw*, and that files holding data in mu-law encoding end in *au*, *mu*, or *snd*.

: *y* = **lin2mu** *(x, n)*

Convert audio data from linear to mu-law.

Mu-law values use 8-bit unsigned integers. Linear values use *n*-bit signed integers or floating point values in the range $-1 \le x \le 1$ if *n* is 0.

If *n* is not specified it defaults to 0, 8, or 16 depending on the range of values in *x*.

See also: mu2lin.

: *y* = **mu2lin** *(x, n)*

Convert audio data from mu-law to linear.

Mu-law values are 8-bit unsigned integers. Linear values use *n*-bit signed integers or floating point values in the range $-1 \le y \le 1$ if *n* is 0.

If *n* is not specified it defaults to 0.

See also: lin2mu.

: **record** *(sec)*
: **record** *(sec, fs)*

Record *sec* seconds of audio from the system's default audio input at a sampling rate of 8000 samples per second.

If the optional argument *fs* is given, it specifies the sampling rate for recording.

For more control over audio recording, use the *audiorecorder* class.

See also: sound, soundsc.

: **sound** *(y)*
: **sound** *(y, fs)*
: **sound** *(y, fs, nbits)*

Play audio data *y* at sample rate *fs* to the default audio device.

The audio signal *y* can be a vector or a two-column array, representing mono or stereo audio, respectively.

If *fs* is not given, a default sample rate of 8000 samples per second is used.

The optional argument *nbits* specifies the bit depth to play to the audio device and defaults to 8 bits.

For more control over audio playback, use the *audioplayer* class.

Questions:

1. Explain Audio File Utilities in Octave?
2. Explain Audio Data Processing functions in Octave?

Conclusion : Thus, I have implemented a GNU Octave program to apply Audio Processing functions like record, retrieval, and audio data processing.

EXPERIMENT NO. 13

Title: Connecting to Database and Extracting Data in Tableau.

Outcome: Students will be able to Connect to Database and Extracting Data in Tableau.

Theory:

Tableau is a Business Intelligence tool for visually analyzing the data. Users can create and distribute an interactive and shareable dashboard, which depict the trends, variations, and density of the data in the form of graphs and charts. Tableau can connect to files, relational and Big Data sources to acquire and process data. The software allows data blending and real-time collaboration, which makes it very unique. It is used by businesses, academic researchers, and many government organizations for visual data analysis. It is also positioned as a leader Business Intelligence and Analytics Platform in Gartner Magic Quadrant.

Tableau Features

Tableau provides solutions for all kinds of industries, departments, and data environments. Following are some unique features which enable Tableau to handle diverse scenarios.

- **Speed of Analysis** − As it does not require high level of programming expertise, any user with access to data can start using it to derive value from the data.
- **Self-Reliant** − Tableau does not need a complex software setup. The desktop version which is used by most users is easily installed and contains all the features needed to start and complete data analysis.
- **Visual Discovery** − The user explores and analyzes the data by using visual tools like colors, trend lines, charts, and graphs. There is very little script to be written as nearly everything is done by drag and drop.
- **Blend Diverse Data Sets** − Tableau allows you to blend different relational, semistructured and raw data sources in real time, without expensive up-front integration costs. The users don't need to know the details of how data is stored.
- **Architecture Agnostic** − Tableau works in all kinds of devices where data flows. Hence, the user need not worry about specific hardware or software requirements to use Tableau.
- **Real-Time Collaboration** − Tableau can filter, sort, and discuss data on the fly and embed a live dashboard in portals like SharePoint site or Salesforce. You can save your view of data and allow colleagues to subscribe to your interactive dashboards so they see the very latest data just by refreshing their web browser.
- **Centralized Data** − Tableau server provides a centralized location to manage all of the organization's published data sources. You can delete, change permissions, add tags, and manage schedules in one convenient location. It's easy to schedule extract refreshes and manage them in the data server.

Administrators can centrally define a schedule for extracts on the server for both incremental and full refreshes.

Download Tableau Desktop

The Free Personal Edition of Tableau Desktop can be downloaded from Tableau Desktop. You need to register with your details to be able to download.

After downloading, the installation is a very straightforward process in which you need to accept the license agreement and provide the target folder for installation. The following steps and screenshots describe the entire setup process.

Start the Installation Wizard

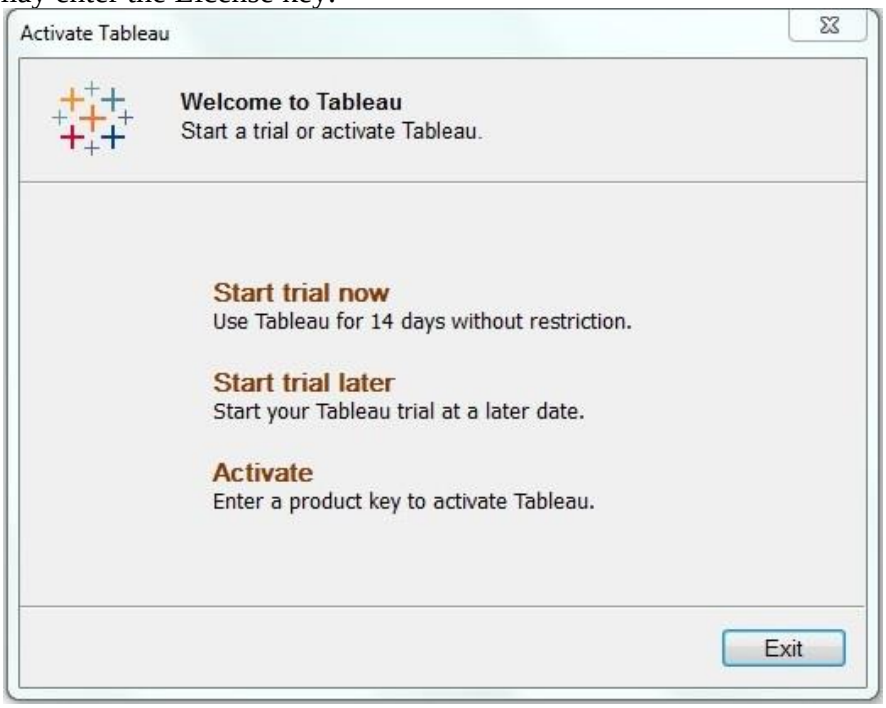
Double-click the **TableauDesktop-64bit-9-2-2.exe**. It will present a screen to allow the installation program to run. Click "Run".

Accept the License Agreement

Read the license agreement and if you agree, choose the "I have read and accept the terms of this license agreement" option. Then, click "Install".

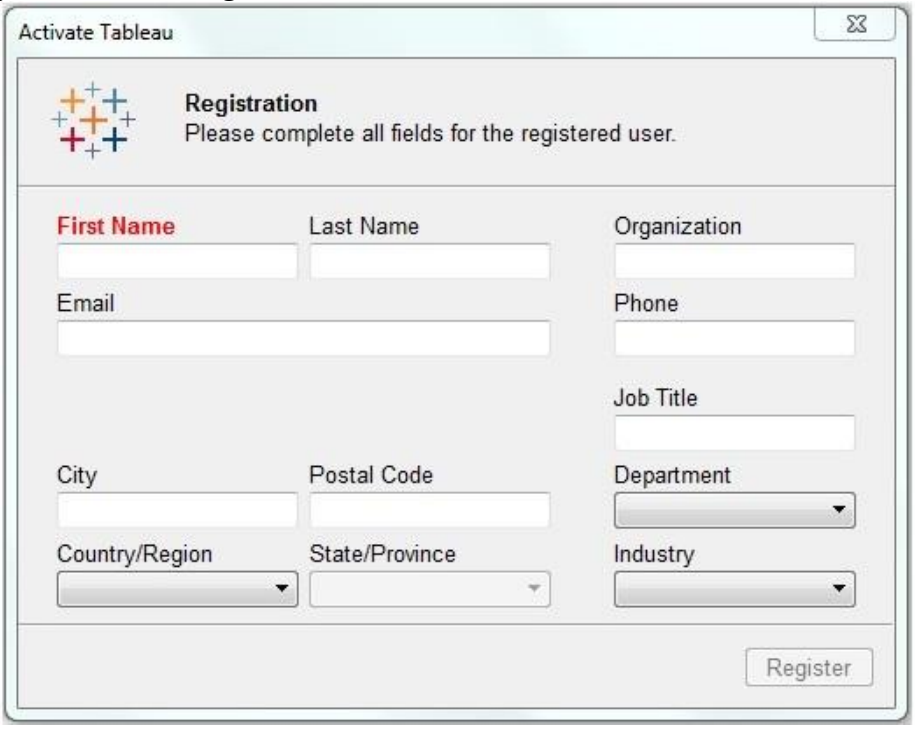
Start Trial

On completion of the installation, the screen prompts you with the option to Start the trial now or later. You may choose to start it now. Also, if you have purchased Tableau then you may enter the License key.



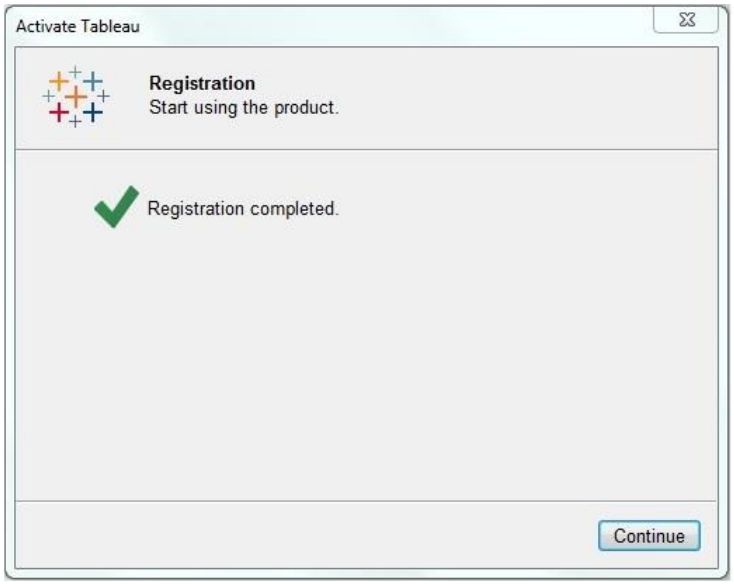
Provide Your Details

Provide your name and organization details. Then, click "Next".



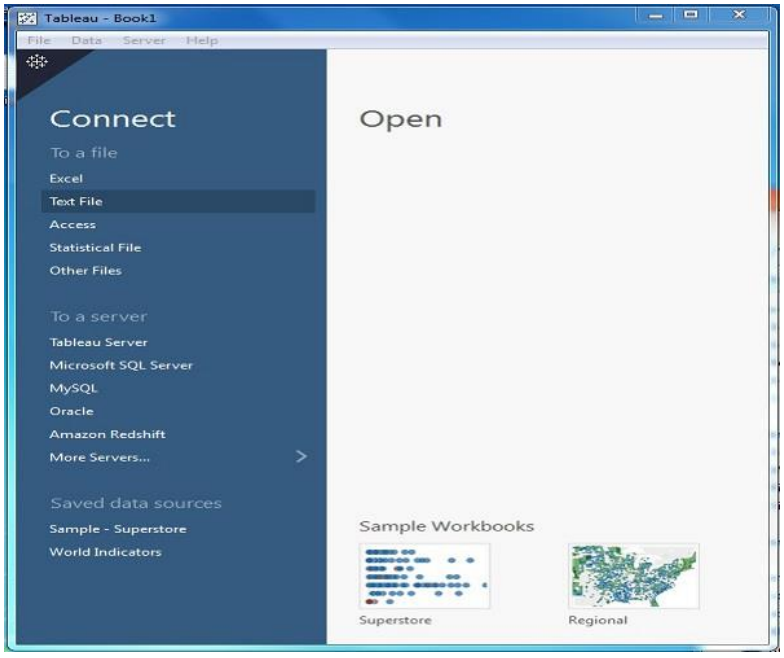
Registration Complete

The registration completion screen appears. Click "Continue".



Verify the Installation

You can verify the installation by going to the Windows start menu. Click the Tableau icon. The following screen appears.



You are now ready to learn Tableau.

here are three basic steps involved in creating any Tableau data analysis report.

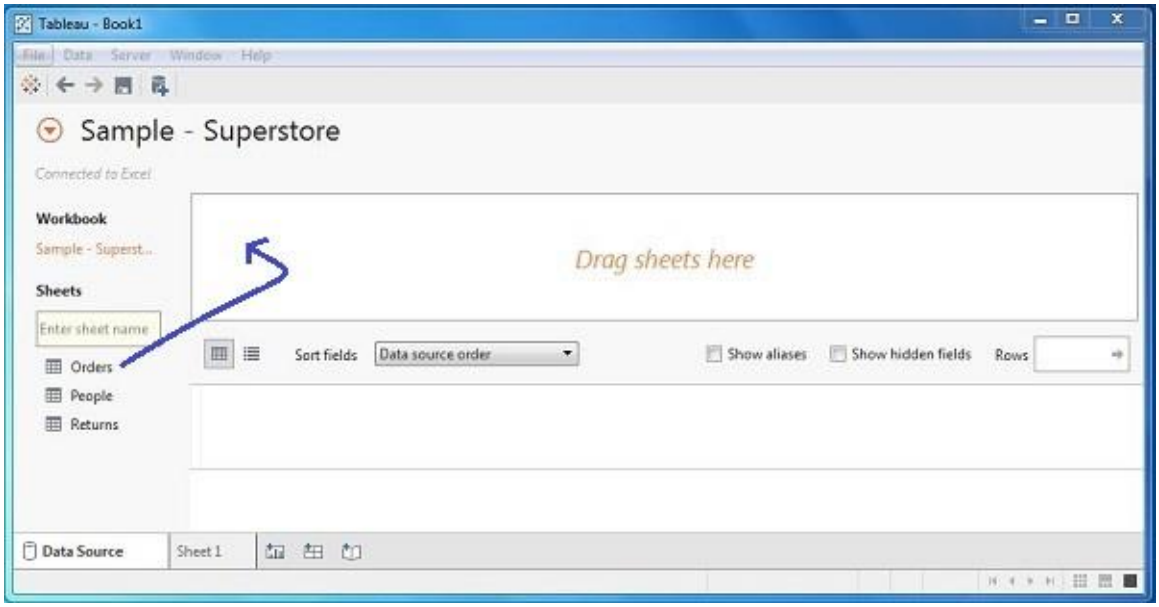
These three steps are −

- **Connect to a data source** − It involves locating the data and using an appropriate type of connection to read the data.
- **Choose dimensions and measures** − This involves selecting the required columns from the source data for analysis.
- **Apply visualization technique** − This involves applying required visualization methods, such as a specific chart or graph type to the data being analyzed.

For convenience, let's use the sample data set that comes with Tableau installation named sample – superstore.xls. Locate the installation folder of Tableau and go to **My Tableau Repository**. Under it, you will find the above file at **Datasources\9.2\en_US-US**.

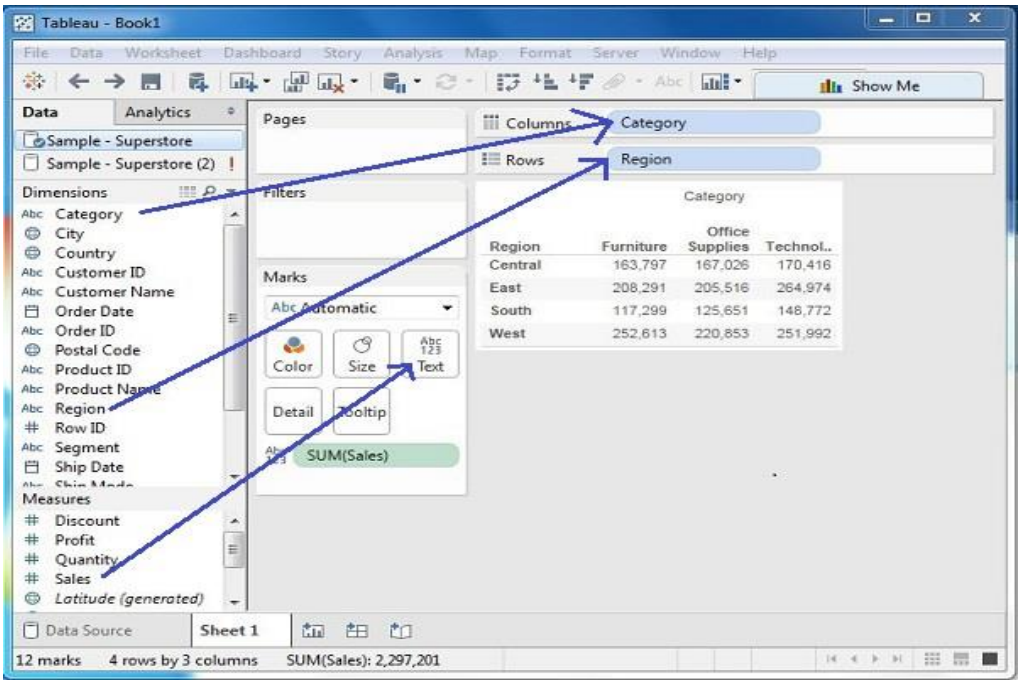
Connect to a Data Source

On opening Tableau, you will get the start page showing various data sources. Under the header **"Connect"**, you have options to choose a file or server or saved data source. Under Files, choose excel. Then navigate to the file **"Sample – Superstore.xls"** as mentioned above. The excel file has three sheets named Orders, People and Returns. Choose **Orders**.



Choose the Dimensions and Measures

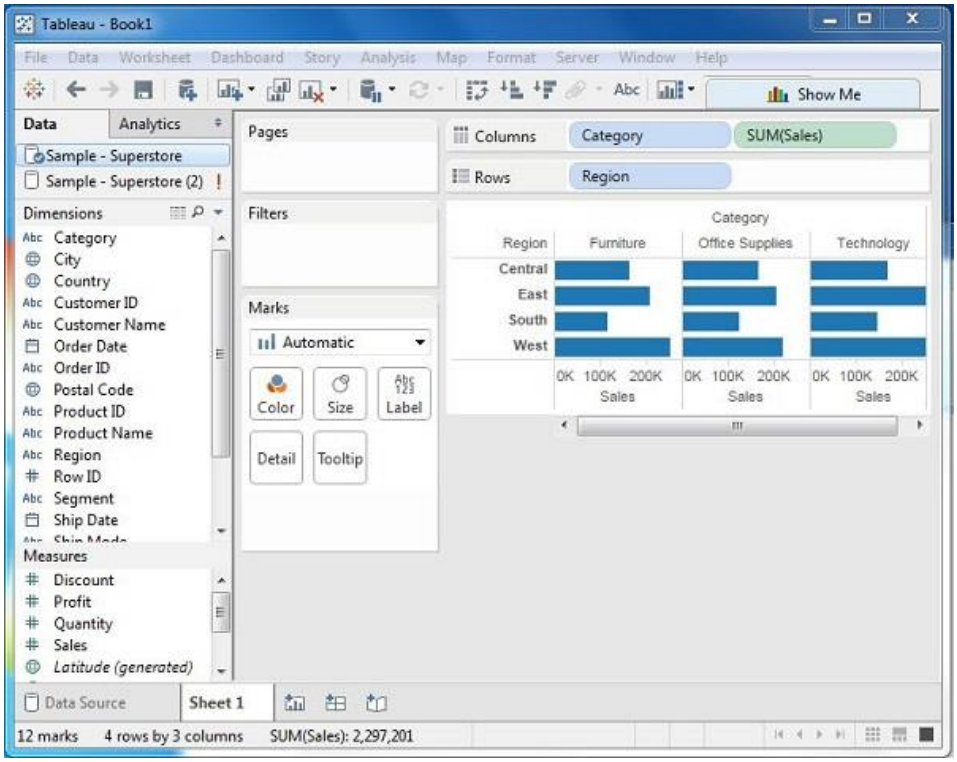
Next, choose the data to be analyzed by deciding on the dimensions and measures. Dimensions are the descriptive data while measures are numeric data. When put together, they help visualize the performance of the dimensional data with respect to the data which are measures. Choose **Category** and **Region** as the dimensions and **Sales** as the measure. Drag and drop them as shown in the following screenshot. The result shows the total sales in each category for each region.



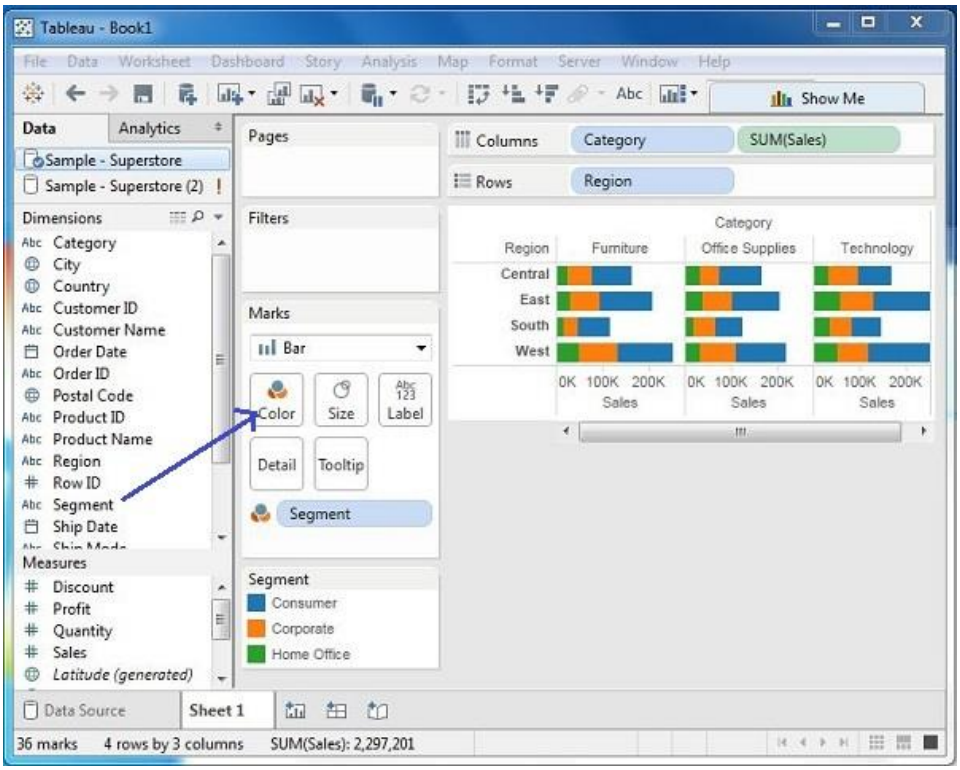
Apply Visualization Technique

In the previous step, you can see that the data is available only as numbers. You have to read and calculate each of the values to judge the performance. However, you can see them as graphs or charts with different colors to make a quicker judgment.

We drag and drop the sum (sales) column from the Marks tab to the Columns shelf. The table showing the numeric values of sales now turns into a bar chart automatically.



You can apply a technique of adding another dimension to the existing data. This will add more colors to the existing bar chart as shown in the following screenshot.



Questions:

1. What is Tableau?
2. Explain different methods for creating Tableau data analysis report?

Conclusion : Thus, I have implemented a program for connection to Database and Extracting Data in Tableau.

EXPERIMENT NO. 14

Title: Develop Tableau worksheet, add filters and create chart using dataset.

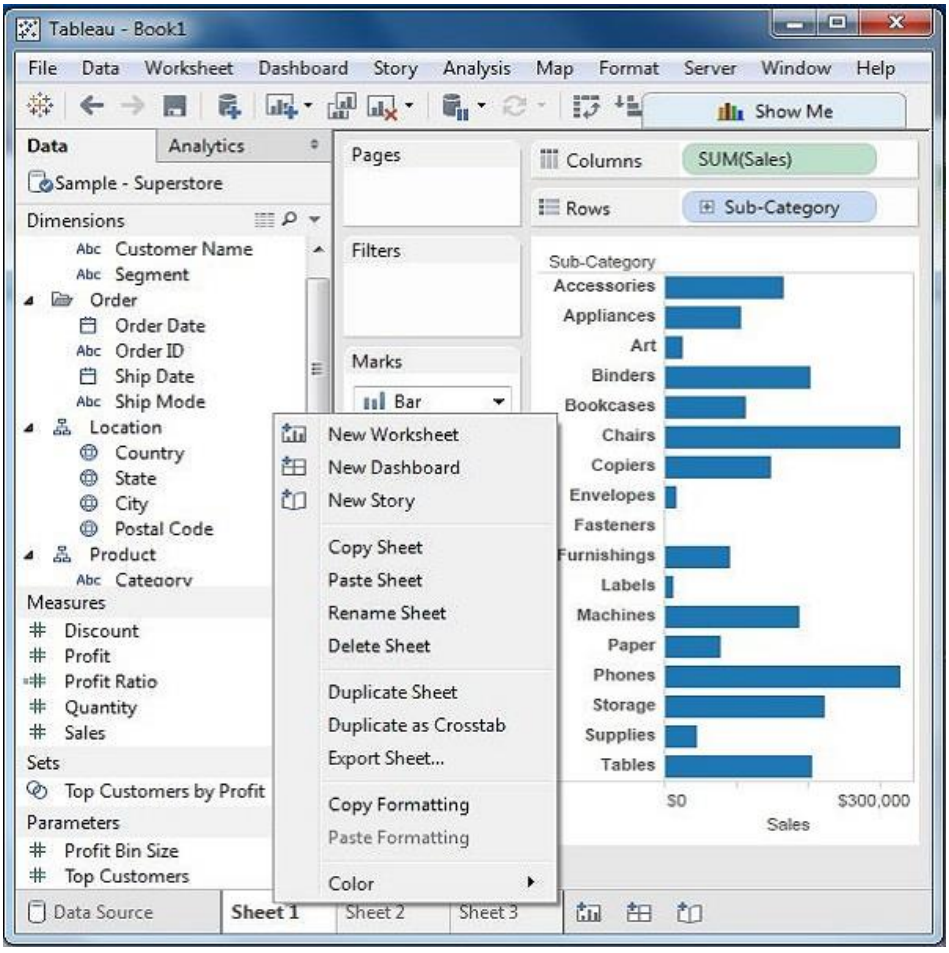
Outcome: Students will be able to develop Tableau worksheet, add filters and create chart using dataset.

Theory:

Worksheet in the Tableau screen is the area where you create the views for data analysis. By default, Tableau provides three blank worksheets when you have established a connection to data source. You can go on adding multiple worksheets to look at different data views in the same screen, one after another.

Adding a Worksheet

You can add a worksheet in two ways. Right-click on the name of the current worksheet and choose the option New Worksheet from the pop-up menu. You can also click on the small icon to the right of the last sheet name to add a worksheet.



Quick Preview of a Worksheet

Staying in one worksheet, you can have a quick preview of another worksheet by hovering the mouse on the name of the other worksheet.

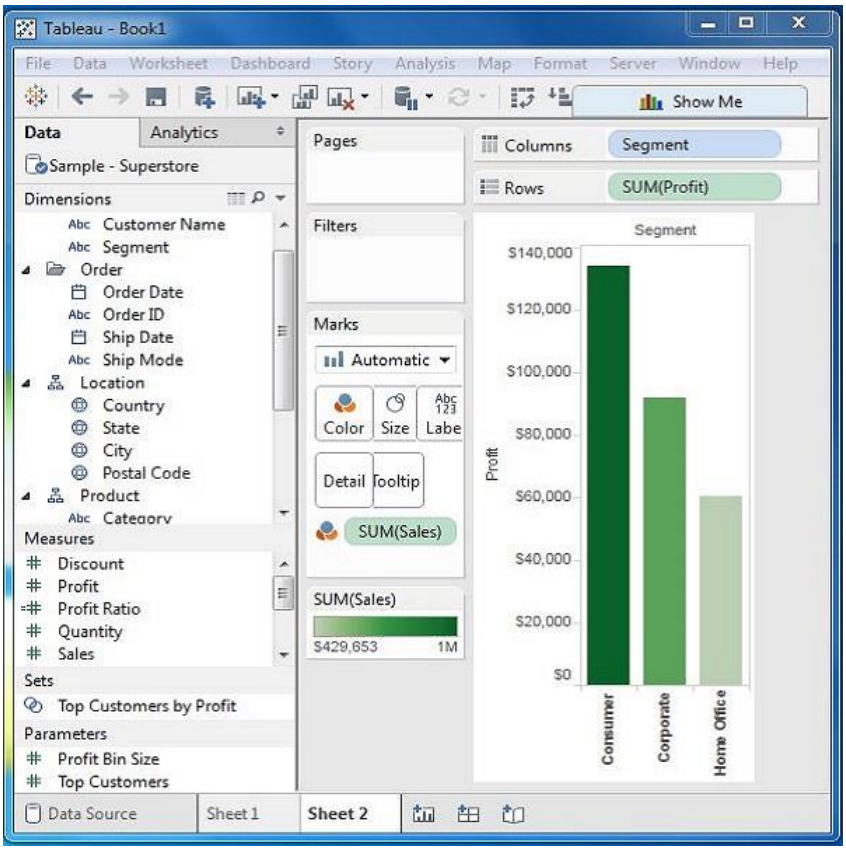


Tableau - Basic Filters

Filtering is the process of removing certain values or range of values from a result set. Tableau filtering feature allows both simple scenarios using field values as well as advanced calculation or context-based filters. In this chapter, you will learn about the basic filters available in Tableau.

There are three types of basic filters available in Tableau. They are as follows −

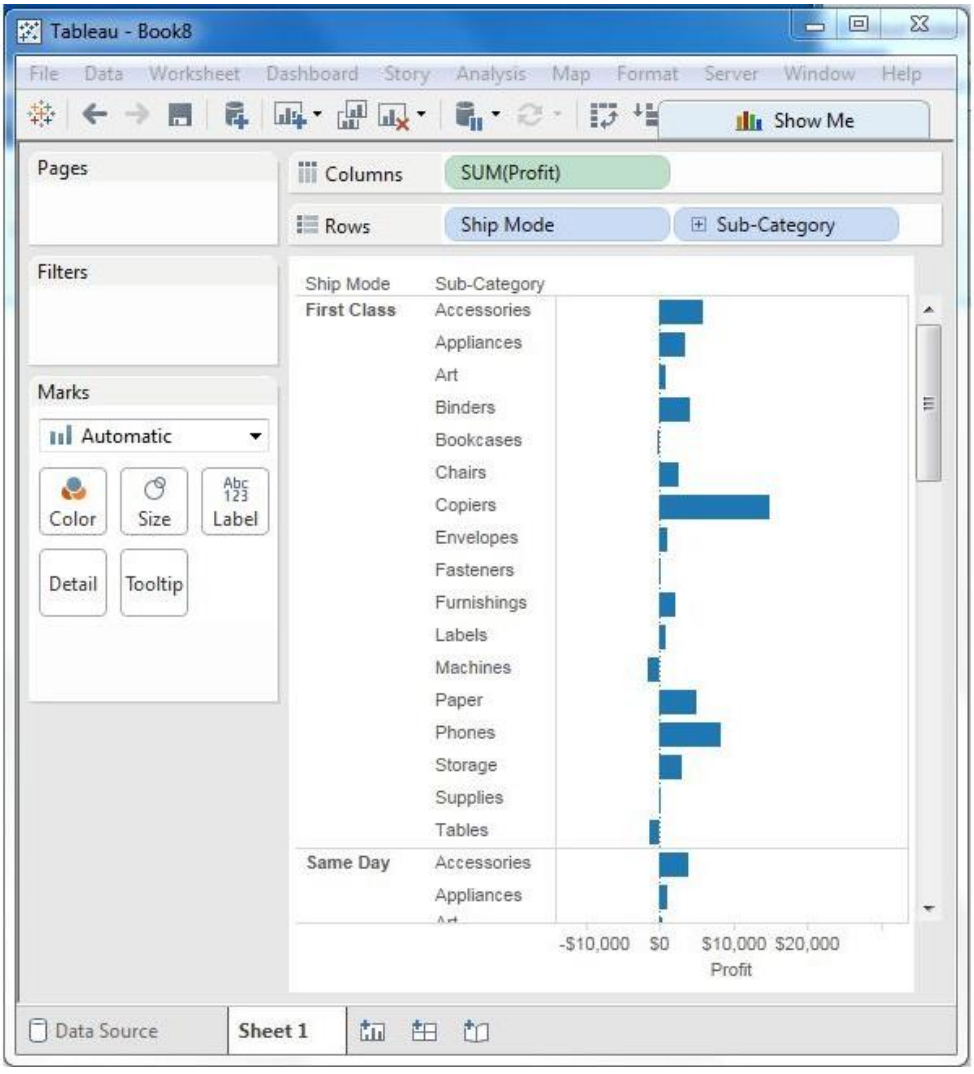
- **Filter Dimensions** are the filters applied on the dimension fields.
- **Filter Measures** are the filters applied on the measure fields.
- **Filter Dates** are the filters applied on the date fields.

Filter Dimensions

These filters are applied on the dimension fields. Typical examples include filtering based on categories of text or numeric values with logical expressions greater than or less than conditions.

Example

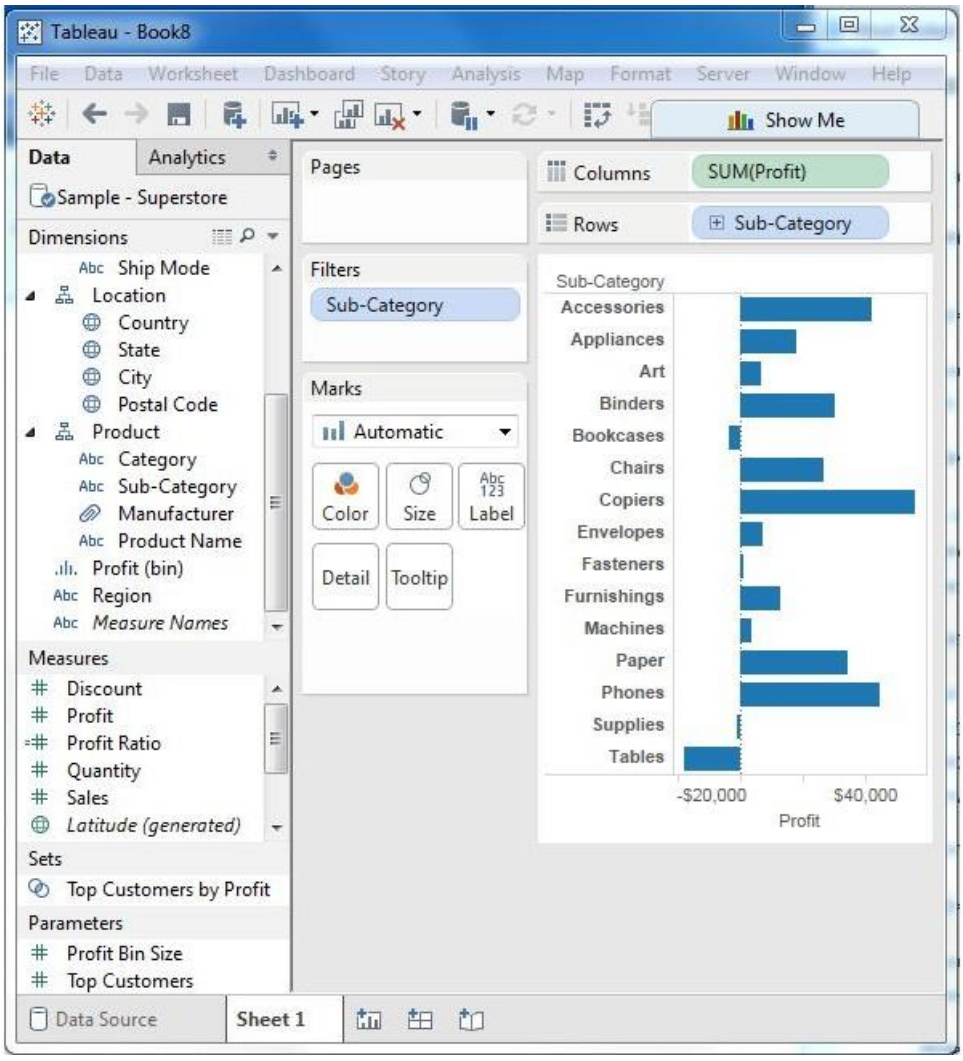
We use the Sample - Superstore data source to apply dimension filters on the sub-category of products. We create a view for showing profit for each sub-category of products according to their shipping mode. For it, drag the dimension field "Sub-Category" to the Rows shelf and the measure field "profit" to the Columns shelf.



Next, drag the Sub-Category dimension to the Filters shelf to open the Filter dialog box. Click the None button at the bottom of the list to deselect all segments.

Then, select the Exclude option in the lower right corner of the dialog box. Finally, select Labels and Storage and then click OK.

The following screenshot shows the result with the above two categories excluded.

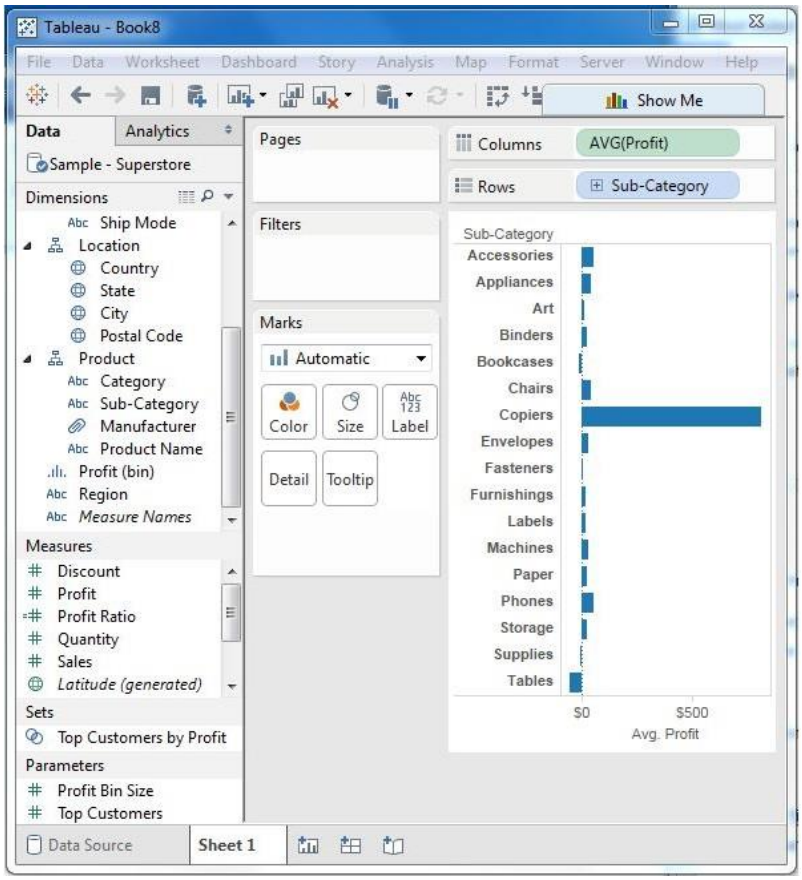


Filter Measures

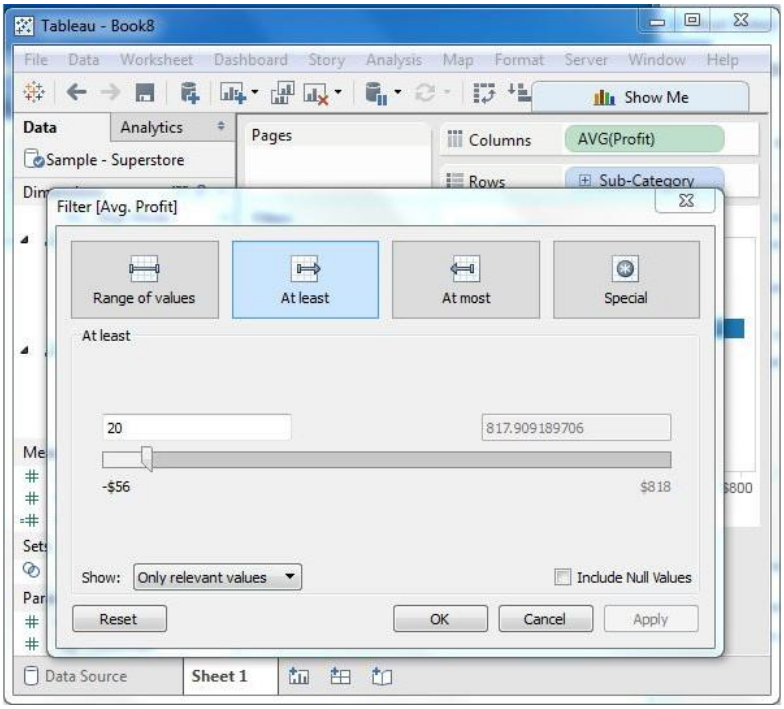
These filters are applied on the measure fields. Filtering is based on the calculations applied to the measure fields. Hence, while in dimension filters you use only values to filter, in measures filter you use calculations based on fields.

Example

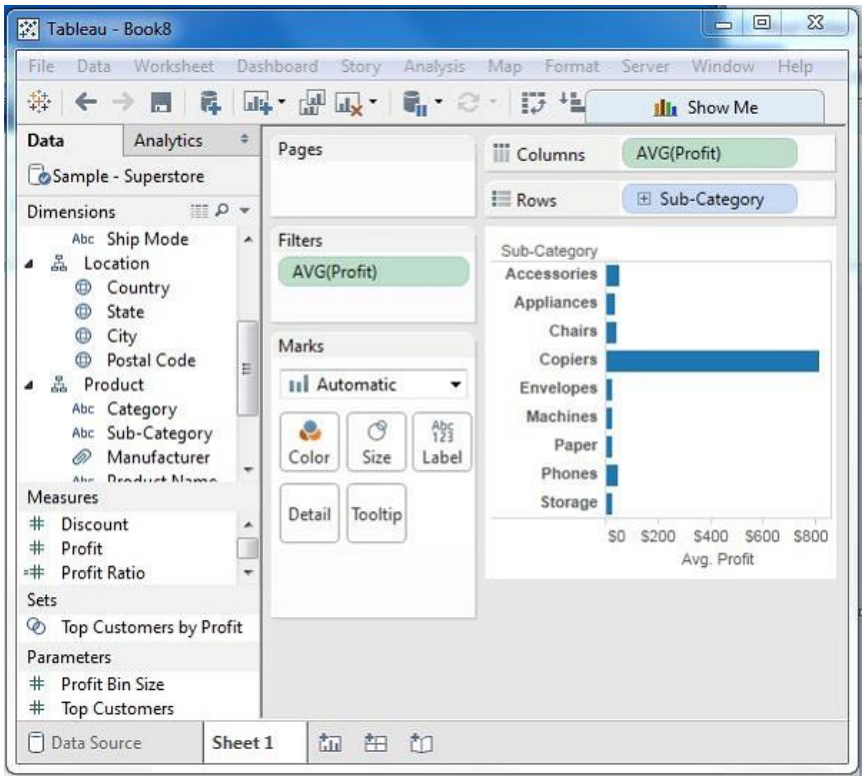
You can use the Sample - Superstore data source to apply dimension filters on the average value of the profits. First, create a view with ship mode and subcategory as dimensions and Average of profit as shown in the following screenshot.



Next, drag the AVG (profit) value to the filter pane. Choose Average as the filter mode. Next, choose "At least" and give a value to filter the rows, which meet these criteria.



After completion of the above steps, we get the final view below showing only the subcategories whose average profit is greater than 20.

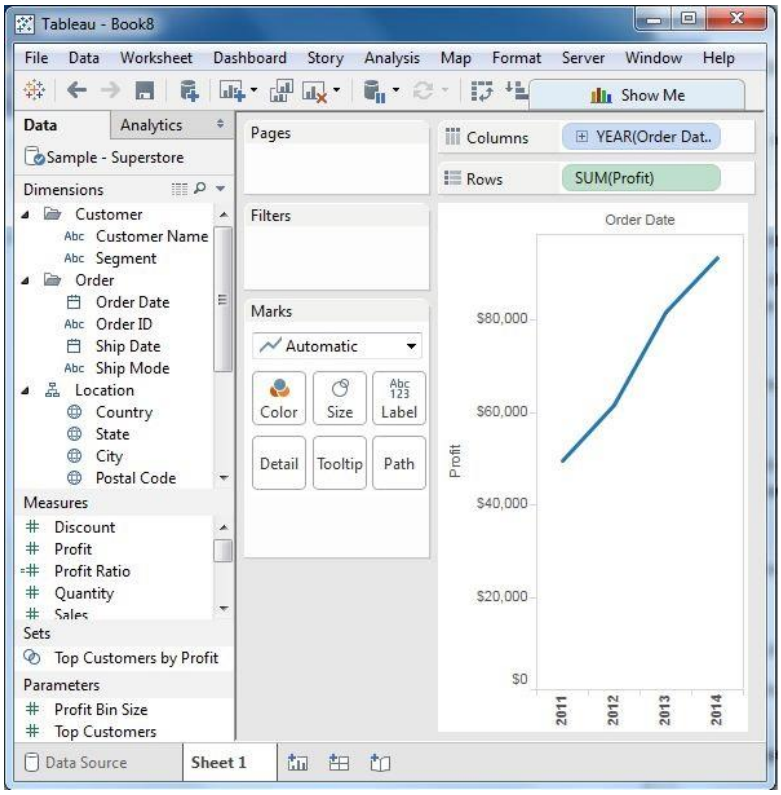


Filter Dates

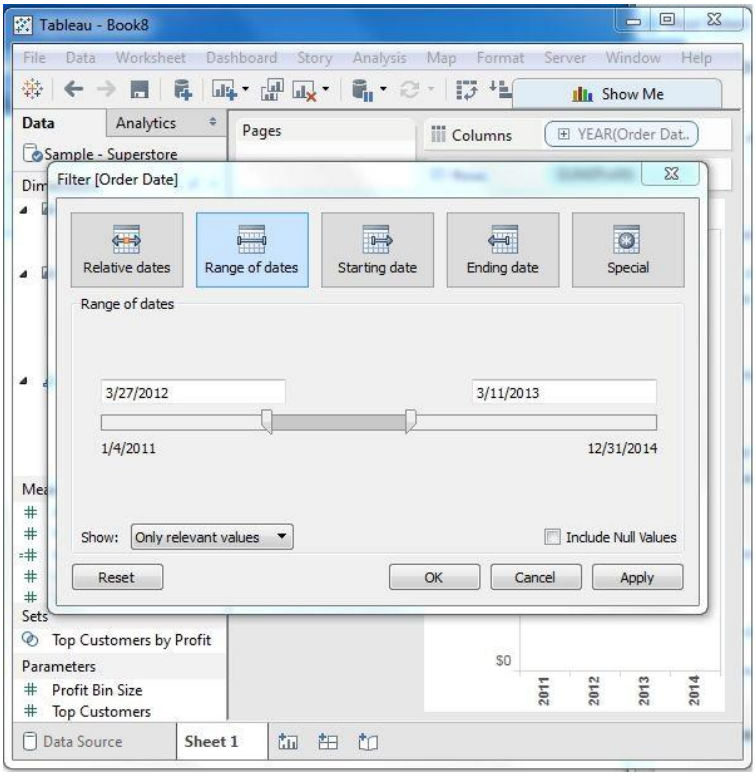
Tableau treats the date field in three different ways while applying the date field. It can apply filter by taking a relative date as compared to today, an absolute date, or range of dates. Each of this option is presented when a date field is dragged out of the filter pane.

Example

We choose the sample - Superstore data source and create a view with order date in the column shelf and profit in the rows shelf as shown in the following screenshot.



Next, drag the "order date" field to the filter shelf and choose Range of dates in the filter dialog box. Choose the dates as shown in the following screenshot.



On clicking OK, the final view appears showing the result for the chosen range of dates as seen in the following screenshot.

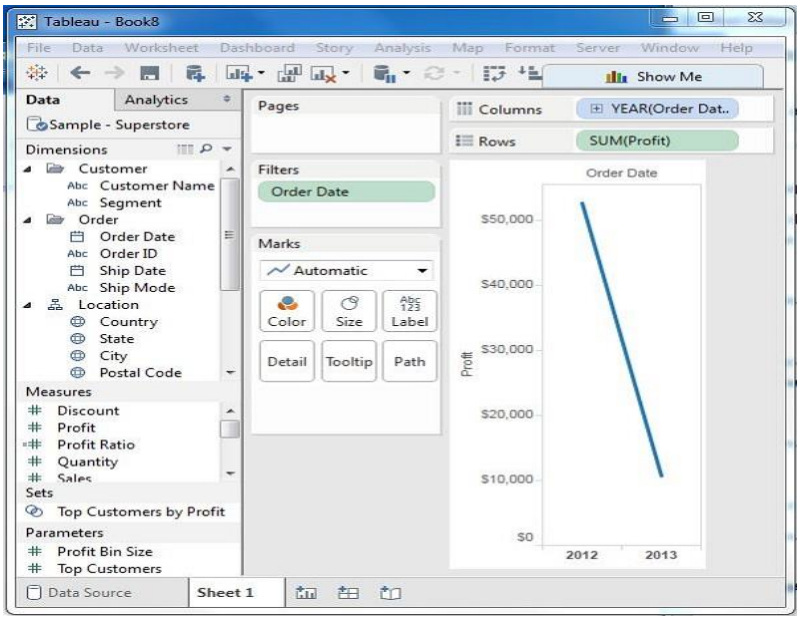
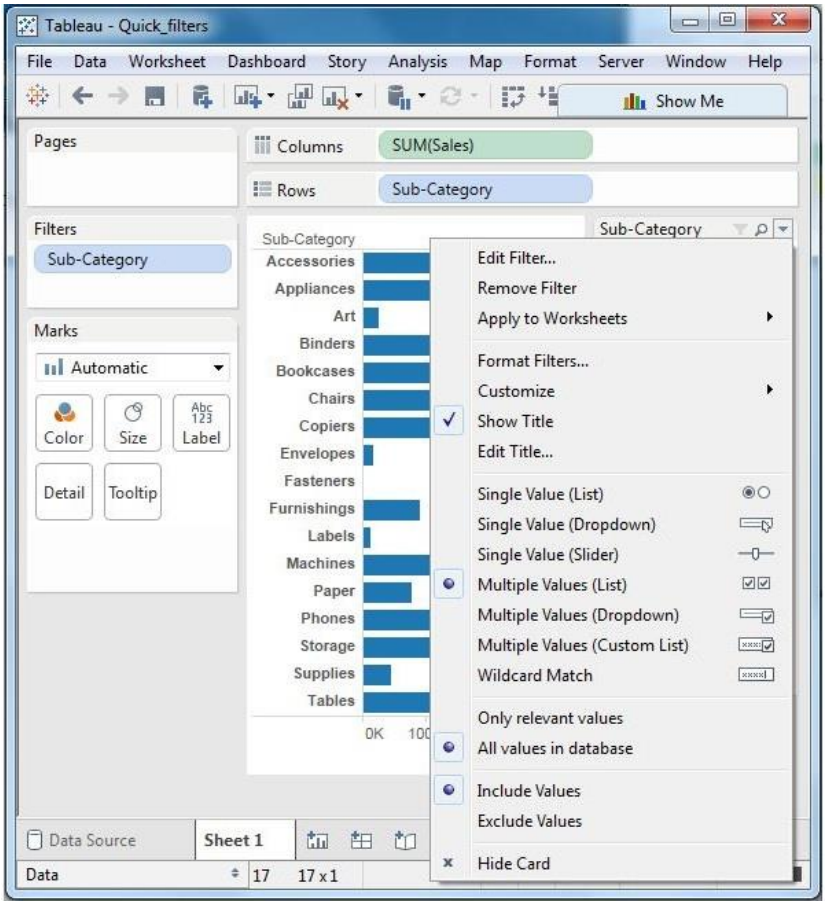


Tableau - Quick Filters

Many filter types in Tableau are quickly available using the right-click option on the dimension or measure. These filters known as Quick filters have enough functionality to solve most of the common filtering needs.

The following screenshot shows how the quick filters are accessed.



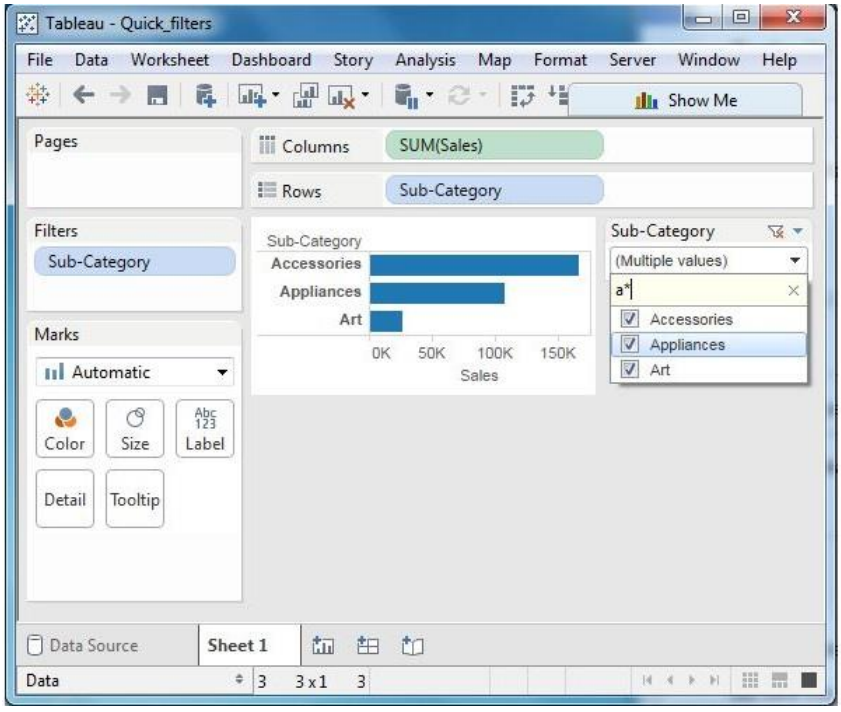
Following is a list of various quick filters and their use.

Filter name	Purpose
Single Value (List)	Select one value at a time in a list.
Single Value (Dropdown)	Select a single value in a drop-down list.
Multiple Values (List)	Select one or more values in a list.
Multiple Values (Dropdown)	Select one or more values in a drop-down list.
Multiple Values (Custom List)	Search and select one or more values.
Single Value (Slider)	Drag a horizontal slider to select a single value.
Wildcard Match	Select values containing the specified characters.

Example

Consider the Sample-Superstore data source to apply some quick filters. In the following example, choose sub-category as the row and sales as the column which by default produces a horizontal bar chart. Next, drag the sub-category field to the filters pane. All the subcategories appear next to the chart. Apply wildcard filtering using the expression **a*** which selects all subcategory name starting with **"a"**.

The below screen shows the result of applying this filter where only the sub-categories starting with **"A"** are displayed.



Clearing the Filter

Once the analysis is complete by applying the filter, remove it by using the clear filter option. For this, go to the filter Pane, right-click on the field name and choose Clear Filter as shown in the following screenshot.

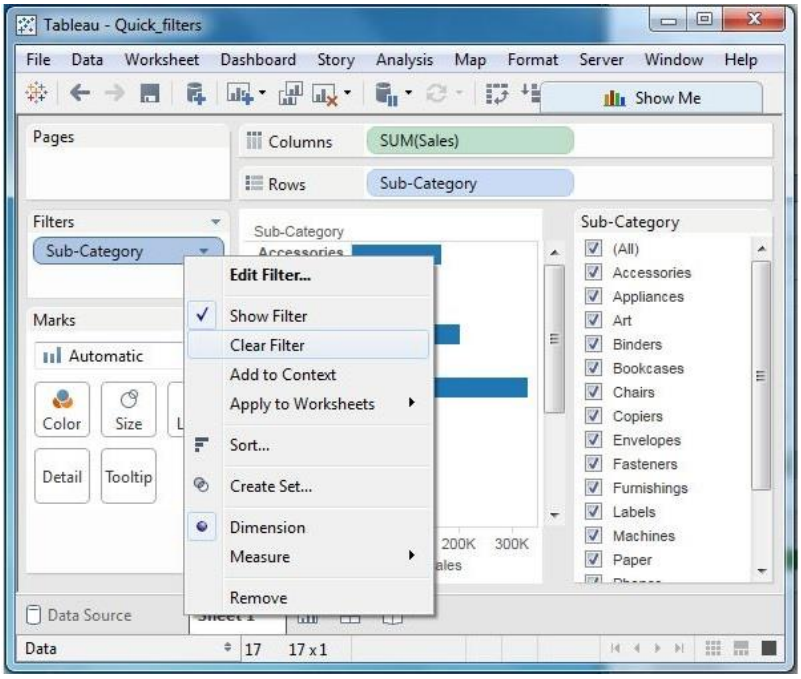


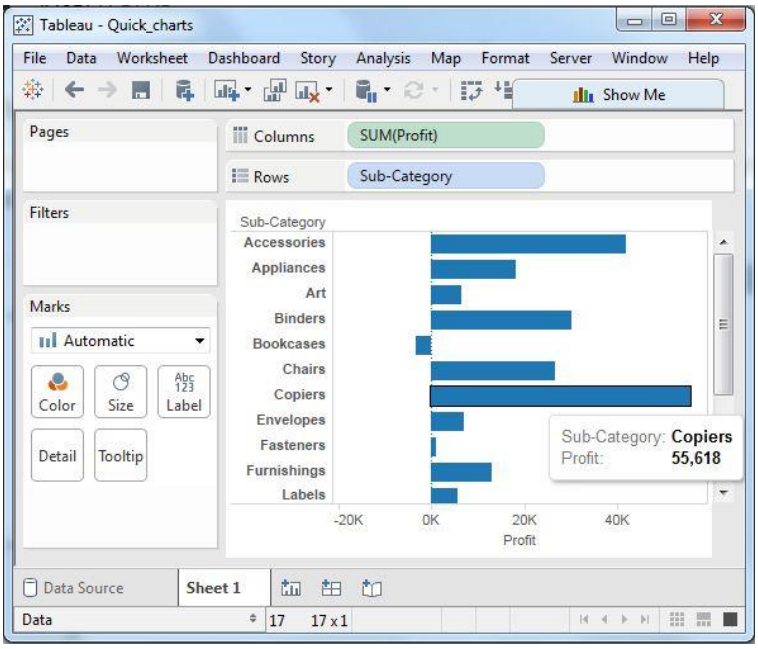
Tableau - Bar Chart

A bar chart represents data in rectangular bars with the length of the bar proportional to the value of the variable. Tableau automatically produces a bar chart when you drag a dimension to the Row shelf and measure to the Column shelf. We can also use the bar chart option present in the Show Me button. If the data is not appropriate for bar chart, then this option will be automatically greyed out.

In Tableau, various types of bar charts can be created by using a dimension and a measure.

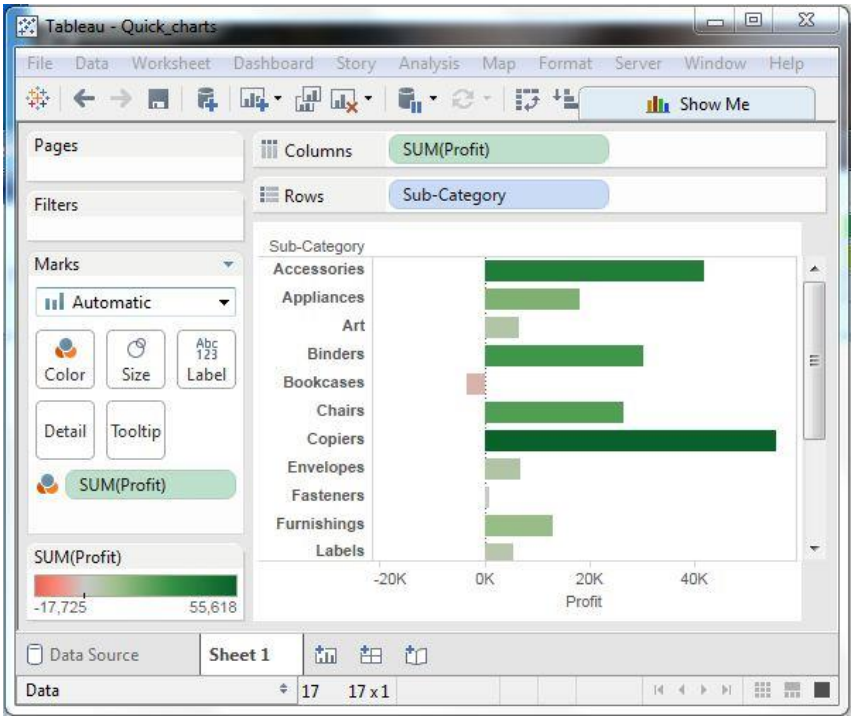
Simple Bar Chart

From the Sample-Superstore, choose the dimension, take profit to the columns shelf and Sub-Category to the rows shelf. It automatically produces a horizontal bar chart as shown in the following screenshot. In case, it does not, you can choose the chart type from the Show Me tool to get the following result.



Bar Chart with Color Range

You can apply colors to the bars based on their ranges. The longer bars get darker shades and the smaller bars get the lighter shades. To do this, drag the profit field to the color palette under the Marks Pane. Also note that, it produces a different color for negative bars.



Stacked Bar Chart

You can add another dimension to the above bar chart to produce a stacked bar chart, which shows different colors in each bar. Drag the dimension field named segment to the Marks pane and drop it in colors. The following chart appears which shows the distribution of each segment in each bar.

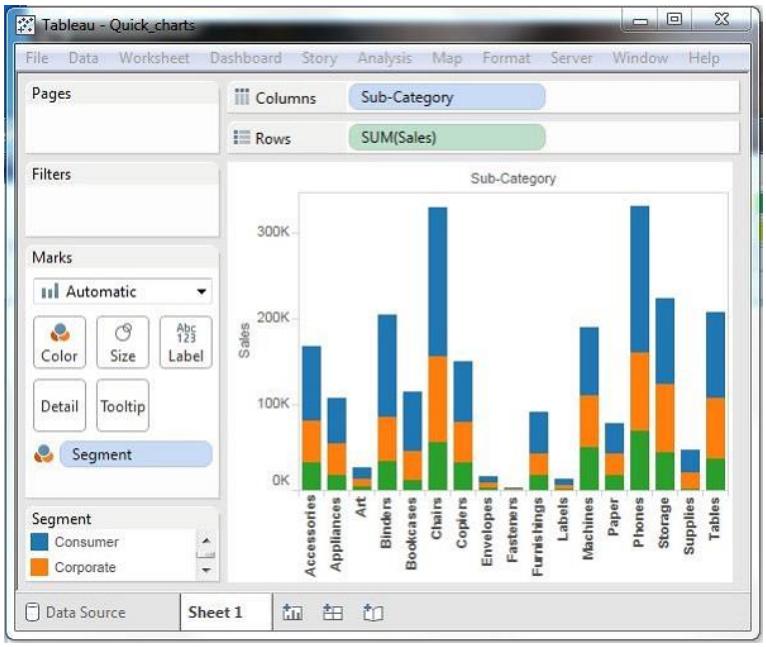
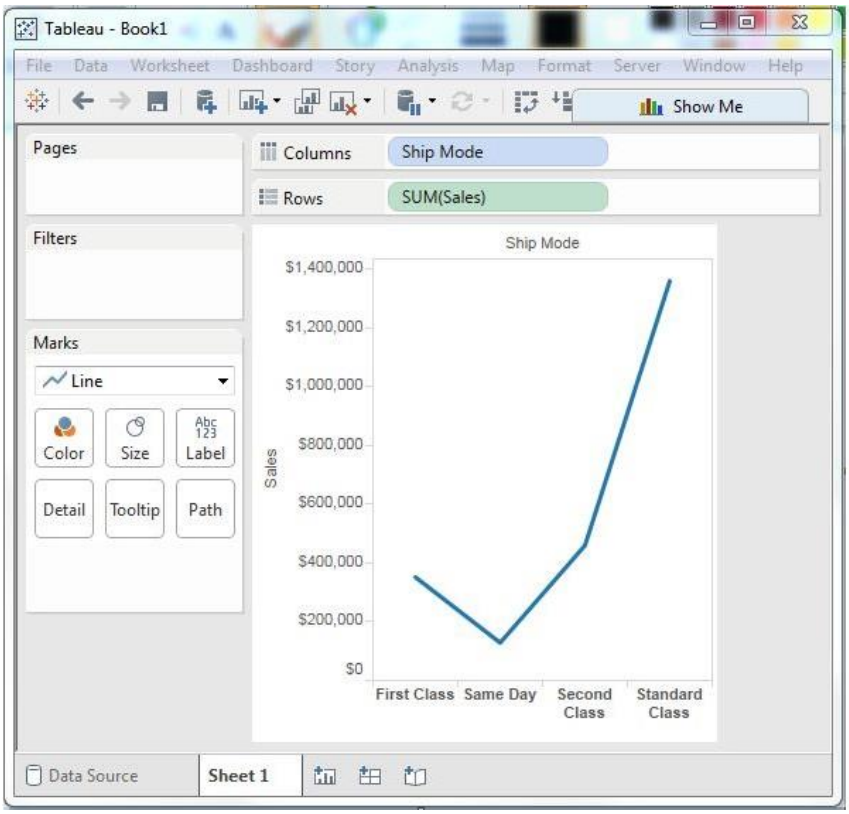


Tableau - Line Chart

In a line chart, a measure and a dimension are taken along the two axes of the chart area. The pair of values for each observation becomes a point and the joining of all these points create a line showing the variation or relationship between the dimensions and measures chosen.

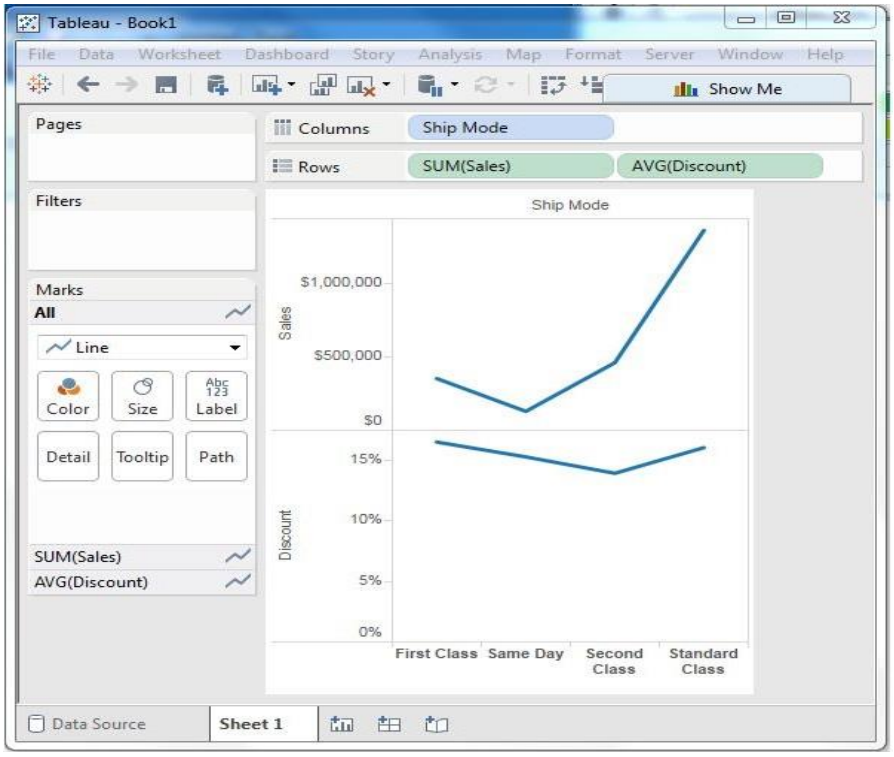
Simple Line Chart

Choose one dimension and one measure to create a simple line chart. Drag the dimension Ship Mode to Columns Shelf and Sales to the Rows shelf. Choose the Line chart from the Marks card. You will get the following line chart, which shows the variation of Sales for different Ship modes.



Multiple Measure Line Chart

You can use one dimension with two or more measures in a line chart. This will produce multiple line charts, each in one pane. Each pane represents the variation of the dimension with one of the measures.



Line Chart with Label

Each of the points making the line chart can be labeled to make the values of the measure visible. In this case, drop another measure Profit Ratio into the labels pane in the Marks card. Choose average as the aggregation and you will get the following chart showing the labels.

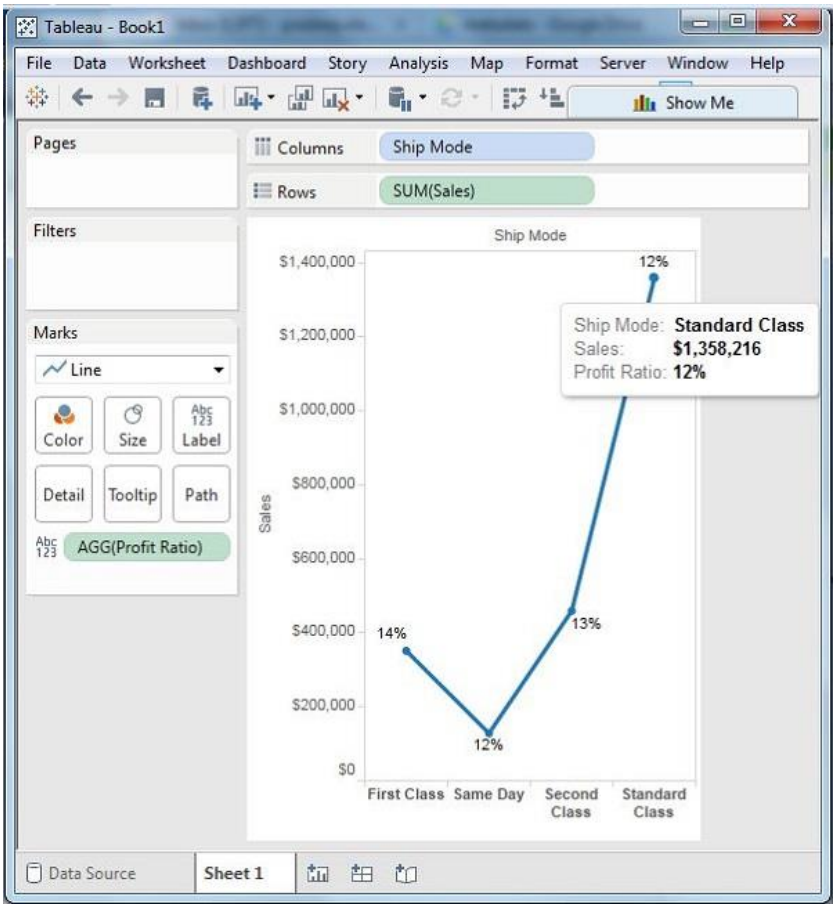
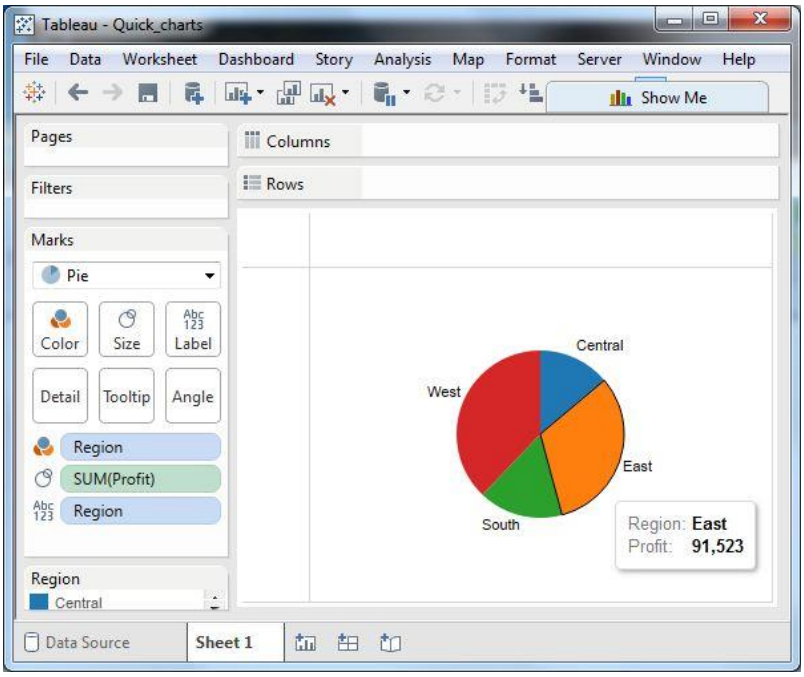


Tableau - Pie Chart

A pie chart represents data as slices of a circle with different sizes and colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart. You can select the pie chart option from the Marks card to create a pie chart.

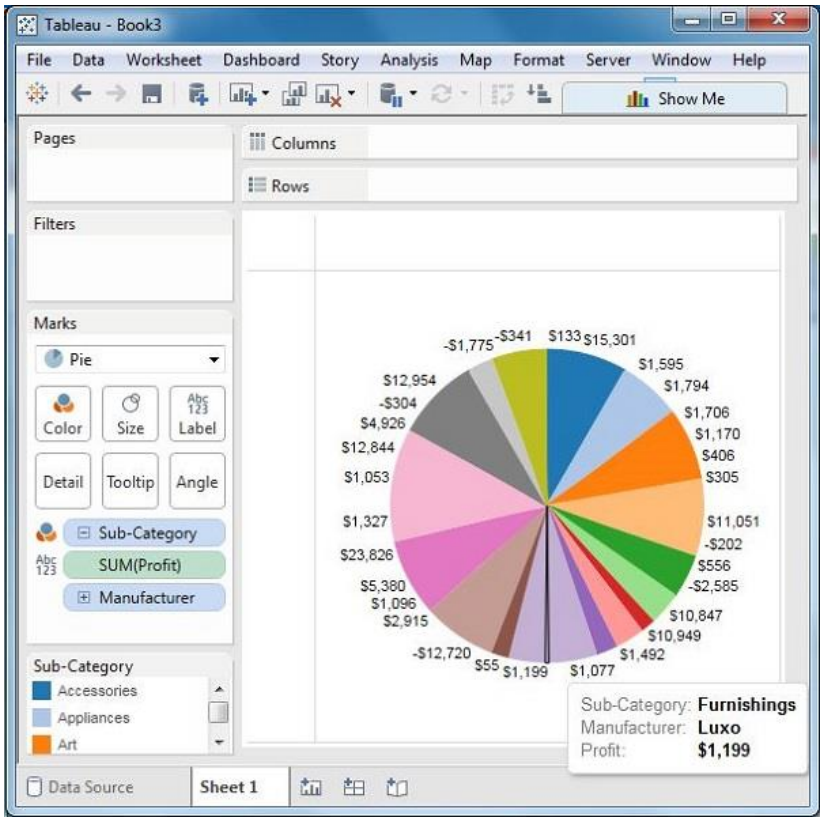
Simple Pie Chart

Choose one dimension and one measure to create a simple pie chart. For example, take the dimension named region with the measure named profit. Drop the Region dimension in the colors and label marks. Drop the Profit measure into the size mark. Choose the chart type as Pie. The following chart appears which shows the 4 regions in different colors.



Drill-Down Pie Chart

You can choose a dimension with hierarchy and as you go deeper into the hierarchy, the chart changes reflect the level of the dimension chosen. In the following example, we take the dimension Sub-Category which has two more levels - Manufacturer and Product Name. Take the measure profit and drop it to the Labels mark. The following pie chart appears which shows the values for each slice.



Going one more level into the hierarchy, we get the manufacturer as the label and the above pie chart changes to the following one.

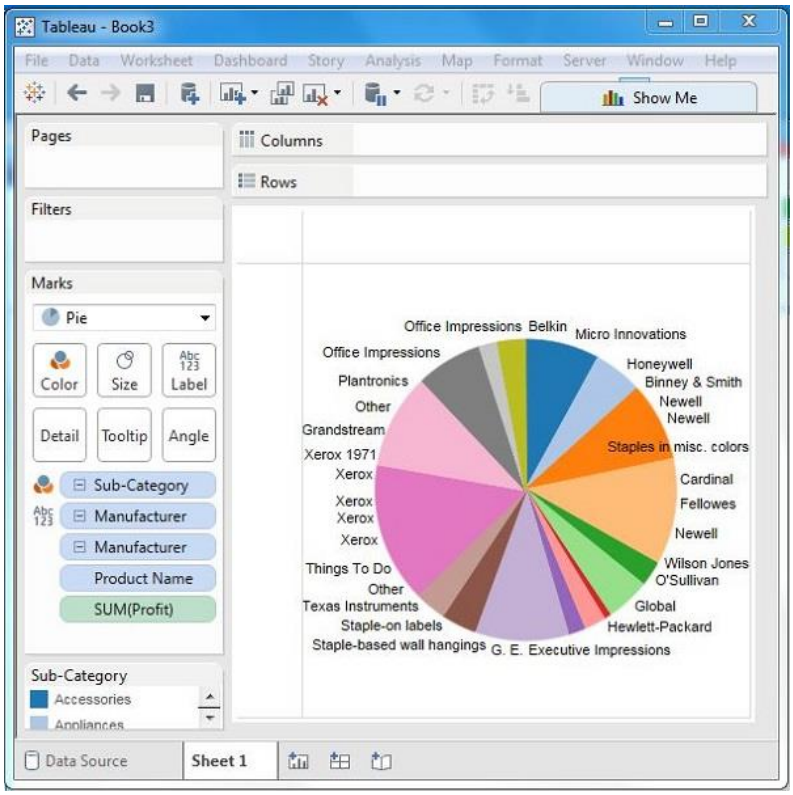


Tableau - Scatter Plot

As the name suggests, a scatter plot shows many points scattered in the Cartesian plane. It is created by plotting values of numerical variables as X and Y coordinates in the Cartesian plane. Tableau takes at least one measure in the Rows shelf and one measure in the Columns shelf to create a scatter plot. However, we can add dimension fields to the scatter plot which play a role in marking different colors for the already existing points in the scatter graph.

Simple Scatter Plot

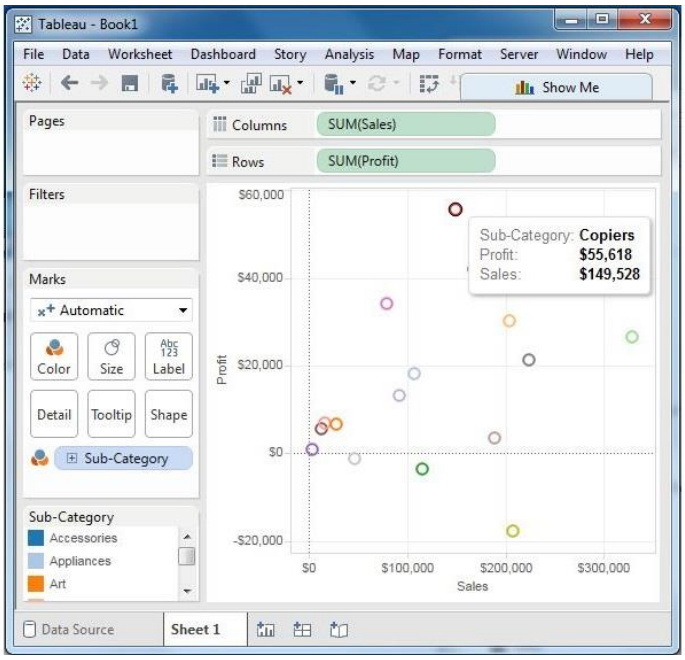
Using the Sample-superstore, let's aim to find the variation of sales and profit figures as the two axes of the Cartesian plane is distributed according to their Sub-Category. To achieve this objective, following are the steps.

Step 1 − Drag and drop the measure Sales to the Columns shelf.

Step 2 − Drag and drop the measure Profit to the Rows shelf.

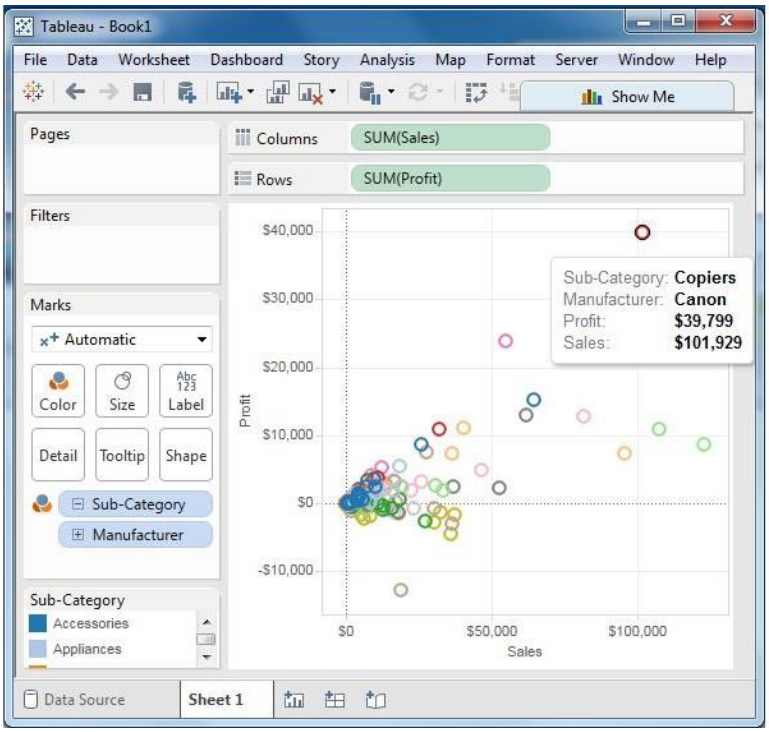
Step 3 − Pull the dimension Sub-Category to the labels Shelf under Marks.

The following chart appears which shows how profit and sales is distributed across the Sub-Category of products.



Scatter Plot - Color Encoded

You can get the values color encoded by dragging the dimension Sub-Category to the color Shelf under the Marks card. This chart shows the scatter points with different color for each point.



Drill-Down Scatter Plot

The same scatter plot can show different values when you choose a dimension with hierarchy. In the following example, we expand the Sub-Category field to show the scatter plot values for the Manufacturers.

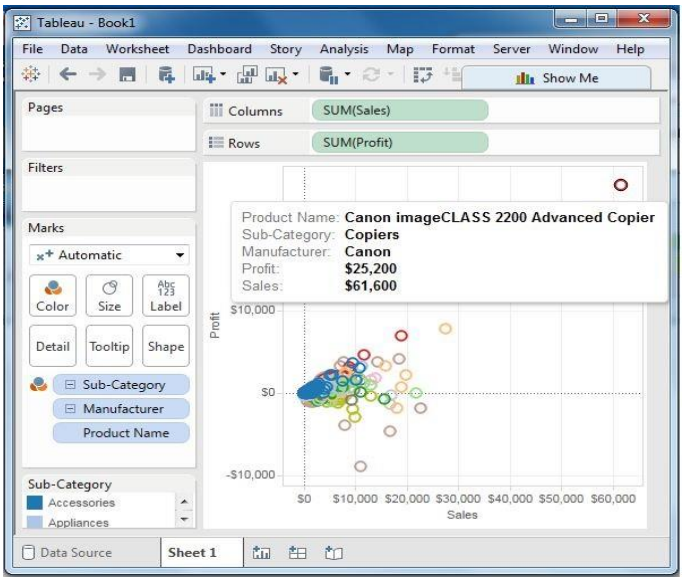


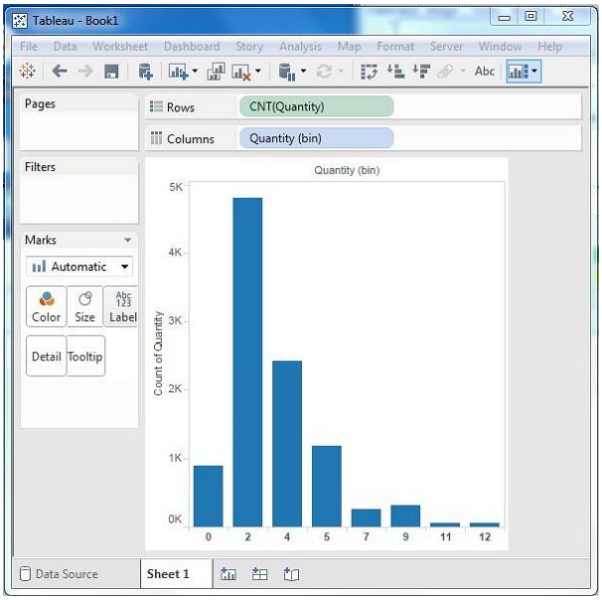
Tableau - Histogram

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chart but it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

Tableau creates a histogram by taking one measure. It creates an additional bin field for the measure used in creating a histogram.

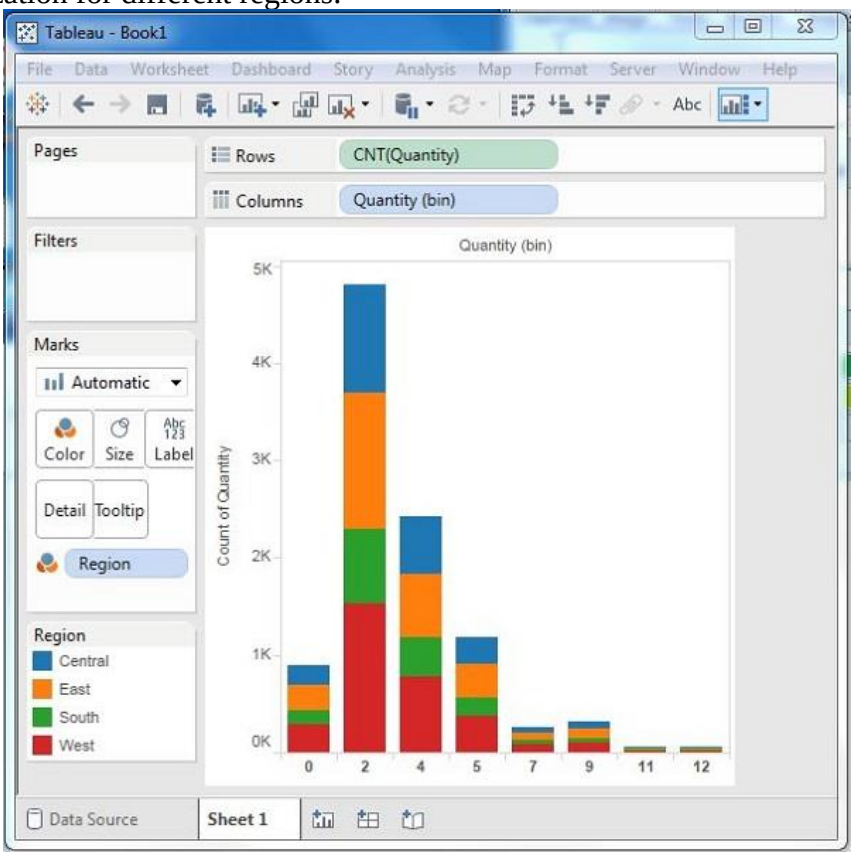
Creating a Histogram

Using the Sample-superstore, plan to find the quantities of sales for different regions. To achieve this, drag the Measure named Quantity to the Rows shelf. Then open Show Me and select the Histogram chart. The following diagram shows the chart created. It shows the quantities automatically bucketed into values ranging from 0 to 4811 and divided into 12 bins.



Creating a Histogram with Dimension

You can also add Dimensions to Measures to create histograms. This will create a stacked histogram. Each bar will have stacks representing the values of the dimension. Following the steps of the above example, add the Region Dimension to the color Shelf under Marks Card. This creates the following histogram where each bar also includes the visualization for different regions.



Questions:

1. How to add worksheets in Tableau?
2. How to add filters and create chart in Tableau?

Conclusion : Thus, I have developed Tableau worksheet, add filters and create chart using dataset.

EXPERIMENT NO. 15

Title: Creating dashboard in Tableau by Adding Sheets with defining Global Filters and Layout Design.

Outcome: Students will be able to create dashboard in Tableau by Adding Sheets with defining Global Filters and Layout Design

Theory:

Tableau – Dashboard

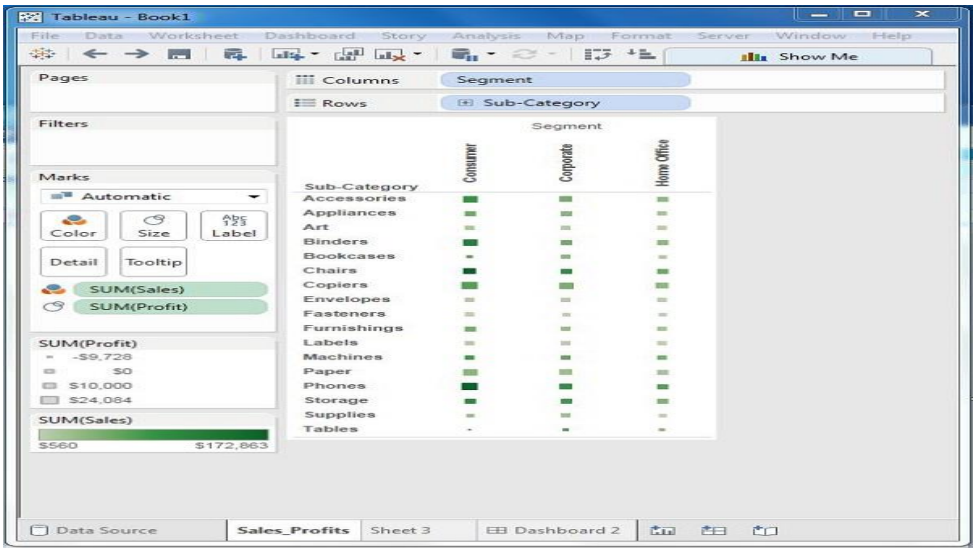
A dashboard is a consolidated display of many worksheets and related information in a single place. It is used to compare and monitor a variety of data simultaneously. The different data views are displayed all at once. Dashboards are shown as tabs at the bottom of the workbook and they usually get updated with the most recent data from the data source. While creating a dashboard, you can add views from any worksheet in the workbook along with many supporting objects such as text areas, web pages, and images.

Each view you add to the dashboard is connected to its corresponding worksheet. So when you modify the worksheet, the dashboard is updated and when you modify the view in the dashboard, the worksheet is updated.

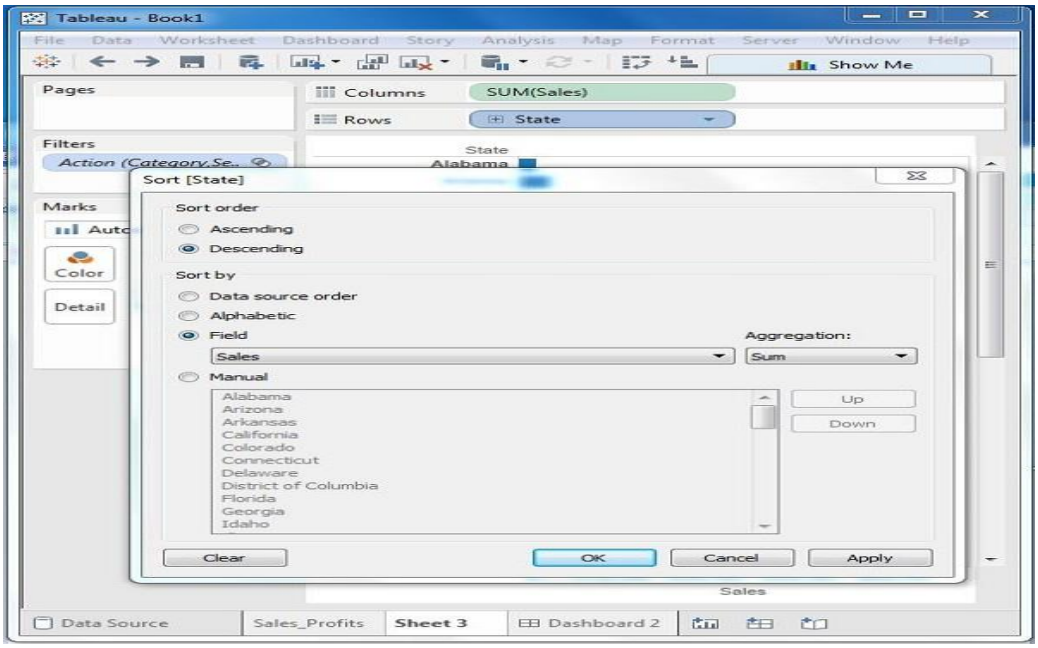
Creating a Dashboard

Using the Sample-superstore, plan to create a dashboard showing the sales and profits for different segments and Sub-Category of products across all the states. To achieve this objective, following are the steps.

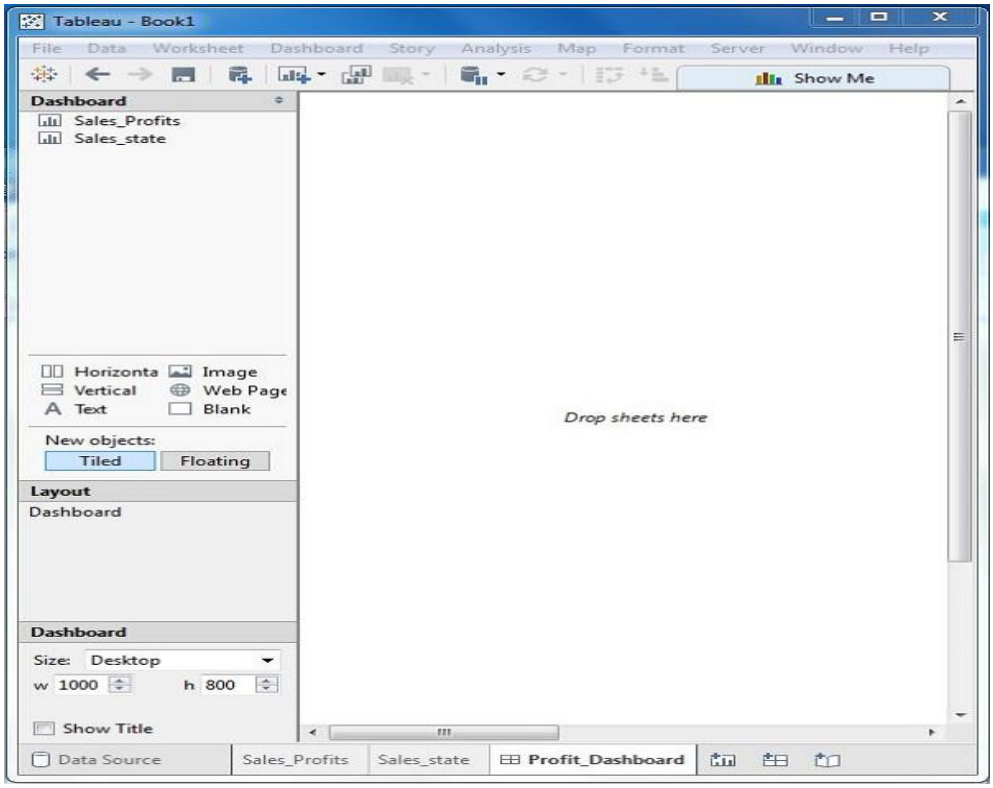
Step 1 − Create a blank worksheet by using the add worksheet icon located at the bottom of the workbook. Drag the dimension Segment to the columns shelf and the dimension Sub-Category to the Rows Shelf. Drag and drop the measure Sales to the Color shelf and the measure Profit to the Size shelf. This worksheet is referred as the Master worksheet. Right-click and rename this worksheet as **Sales_Profits**. The following chart appears.



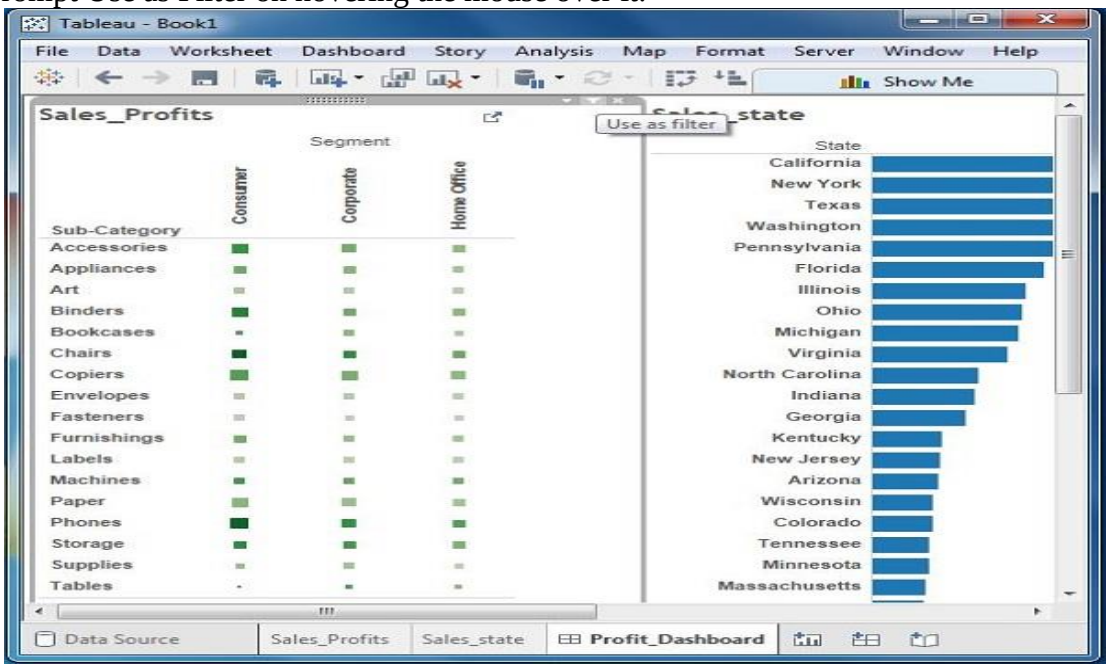
Step 2 − Create another sheet to hold the details of the Sales across the States. For this, drag the dimension State to the Rows shelf and the measure Sales to the Columns shelf as shown in the following screenshot. Next, apply a filter to the State field to arrange the Sales in a descending order. Right-click and rename this worksheet as **Sales_state**.



Step 3 − Next, create a blank dashboard by clicking the Create New Dashboard link at the bottom of the workbook. Right-click and rename the dashboard as Profit_Dashboard.



Step 4 − Drag the two worksheets to the dashboard. Near the top border line of Sales Profit worksheet, you can see three small icons. Click the middle one, which shows the prompt Use as Filter on hovering the mouse over it.



Step 5 − Now in the dashboard, click the box representing Sub-Category named Machines and segment named Consumer.

You can notice that only the states where the sales happened for this amount of profit are filtered out in the right pane named **Sales_state**. This illustrates how the sheets are linked in a dashboard.

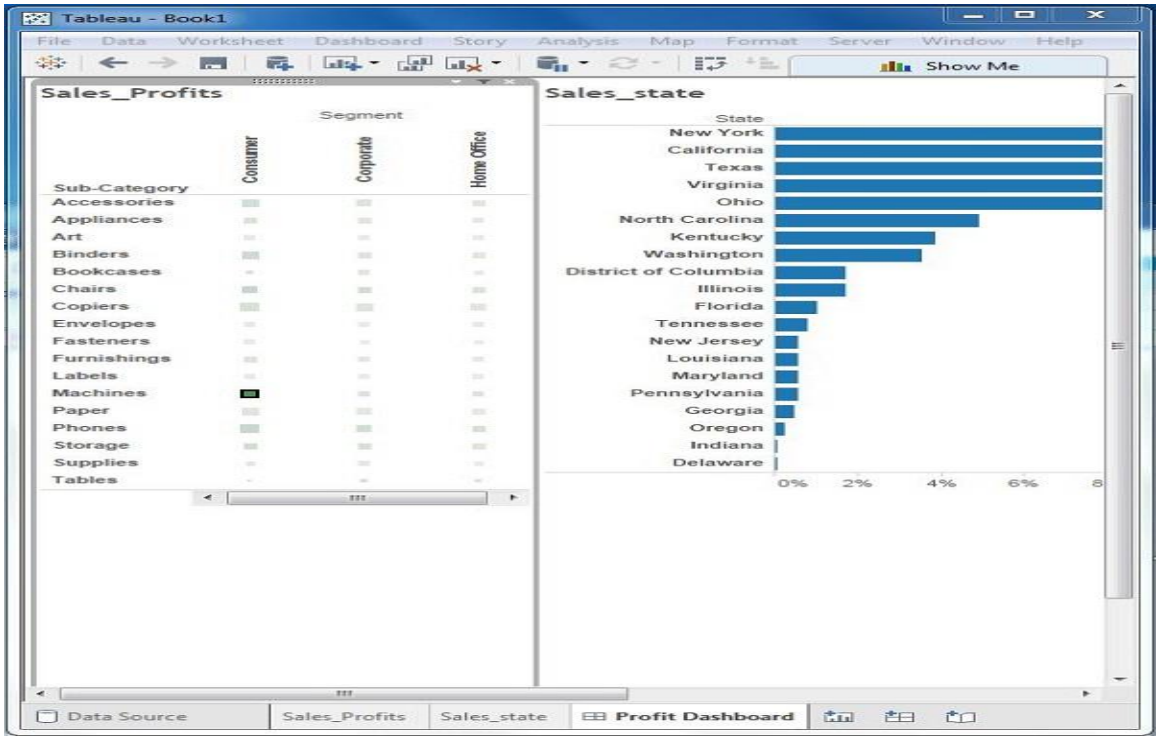
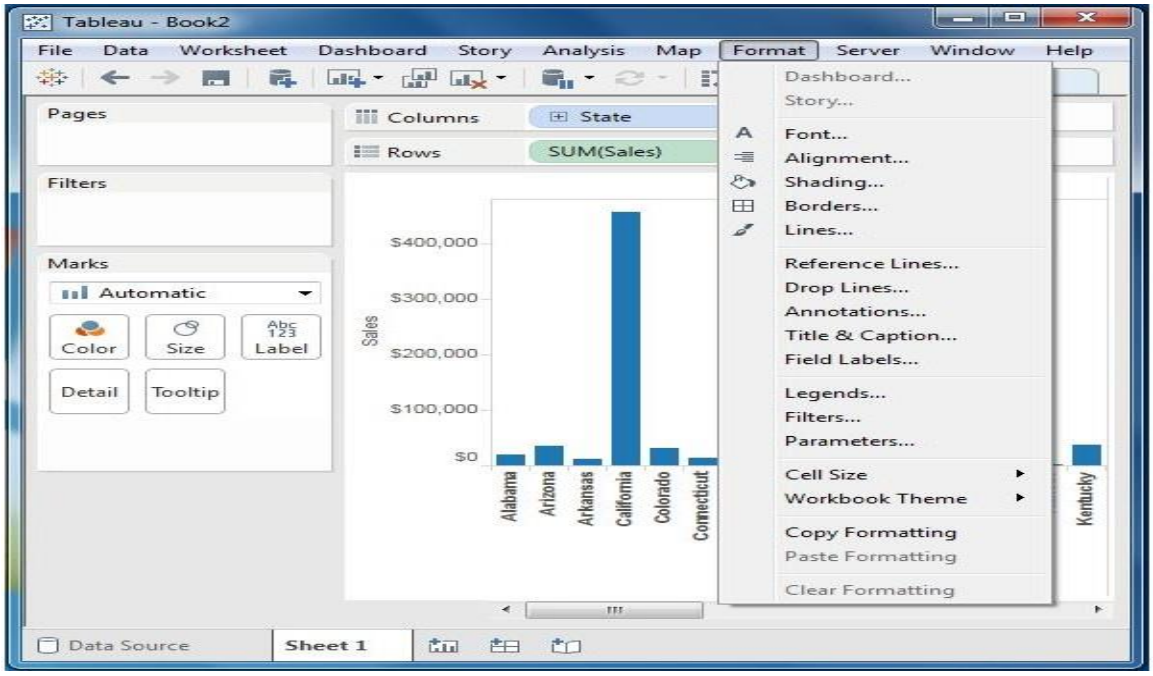


Tableau – Formatting

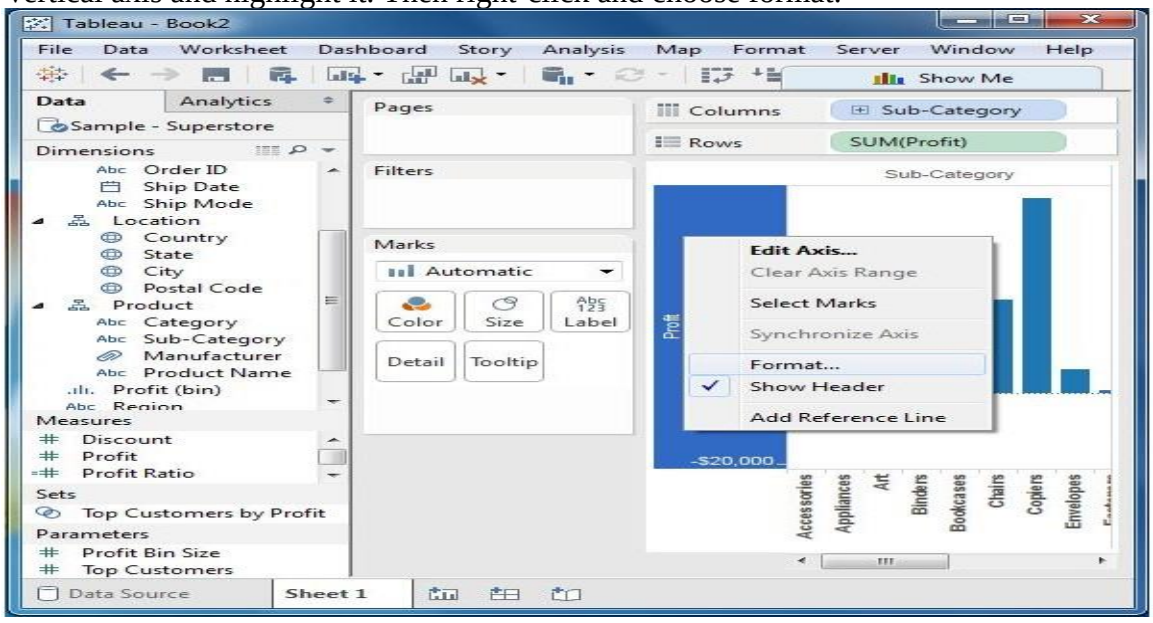
Tableau has a very wide variety of formatting options to change the appearance of the visualizations created. You can modify nearly every aspect such as font, color, size, layout, etc. You can format both the content and containers like tables, labels of axes, and workbook theme, etc.

The following diagram shows the Format Menu which lists the options. In this chapter, you will touch upon some of the frequently used formatting options.



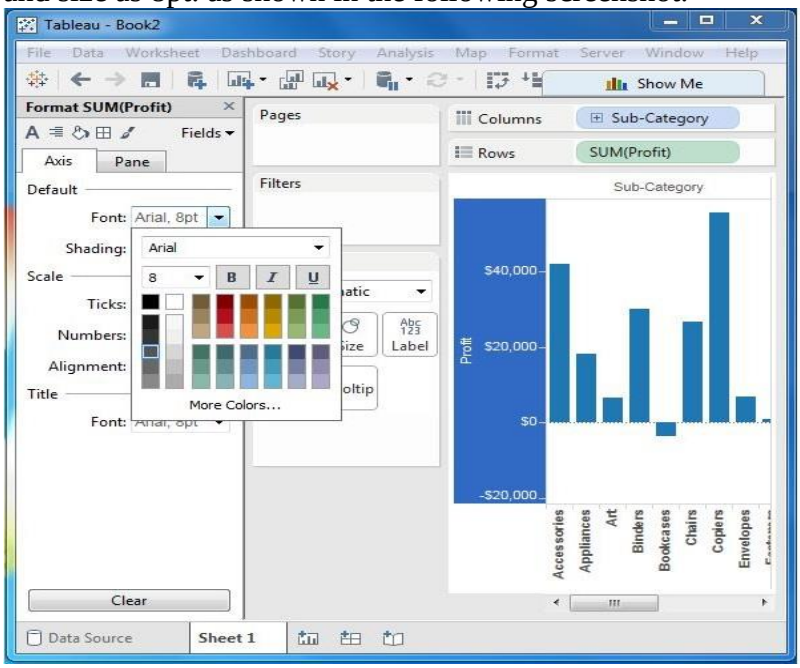
Formatting the Axes

You can create a simple bar chart by dragging and dropping the dimension Sub-Category into the Columns Shelf and the measure Profit into the Rows shelf. Click the vertical axis and highlight it. Then right-click and choose format.



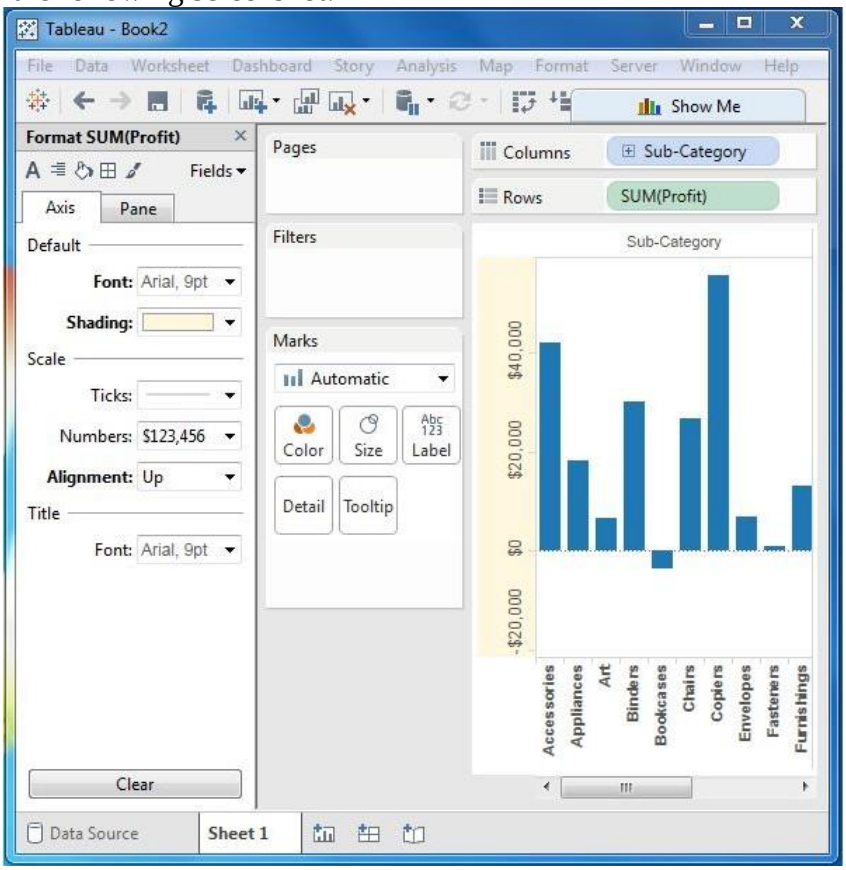
Change the Font

Click the font drop-down in the Format bar, which appears on the left. Choose the font type as Arial and size as 8pt. as shown in the following screenshot.



Change the Shade and Alignment

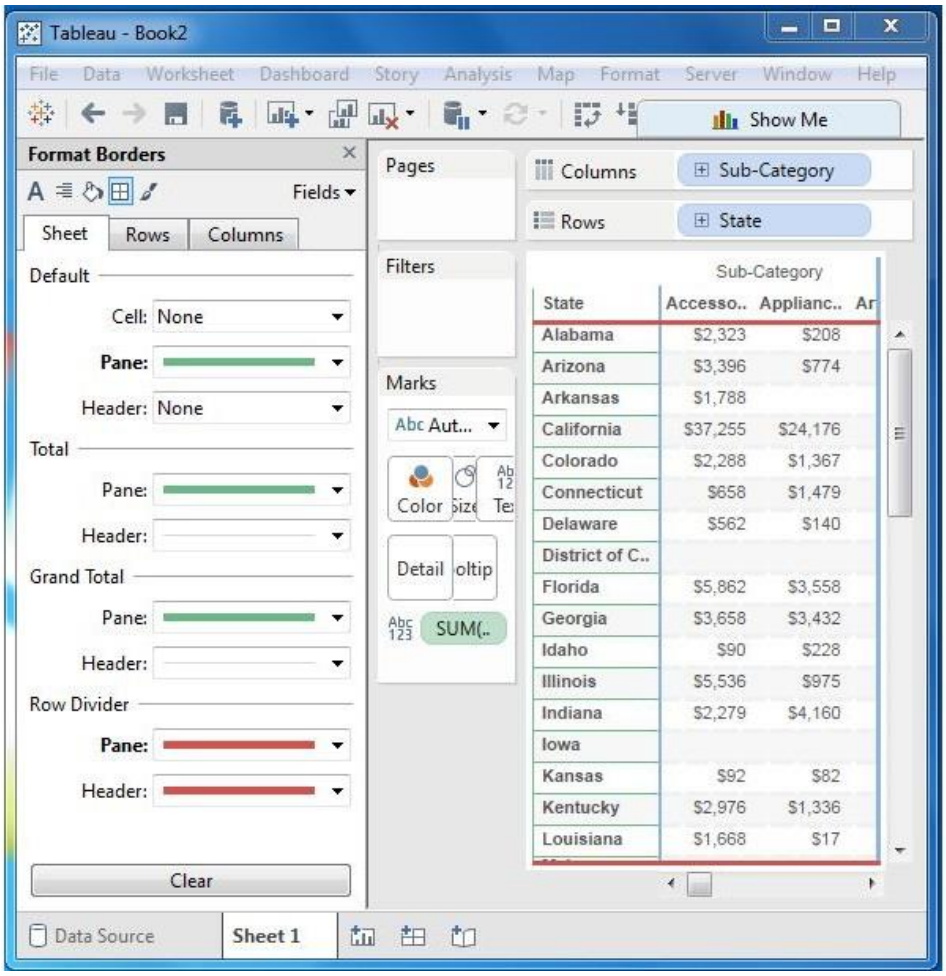
You can also change the orientation of the values in the axes as well as the shading color as shown in the following screenshot.



Format Borders

Consider a crosstab chart with Sub-Category in the Columns shelf and State in the Rows shelf. Now, you can change the borders of the crosstab table created by using the formatting options. Right-click on crosstab chart and choose Format.

The Format Borders appear in the left pane. Choose the options as shown in the following screenshot.



Questions:

1. How to create Dashboard in Tableau?
2. Explain different filters in Tableau?

Conclusion : Thus, I have created dashboard in Tableau by Adding Sheets with defining Global Filters and Layout Design.