

## Experiment No-01

**Title:** Download, install and explore the features of, Jupyter environment.

**Aim:** To study, installation of Anaconda and explore the features of Jupyter environment.

### Theory:

Anaconda is an open-source software that contains Jupyter, spyder, etc that are used for large data processing, data analytics, heavy scientific computing. Anaconda works for R and python programming language.

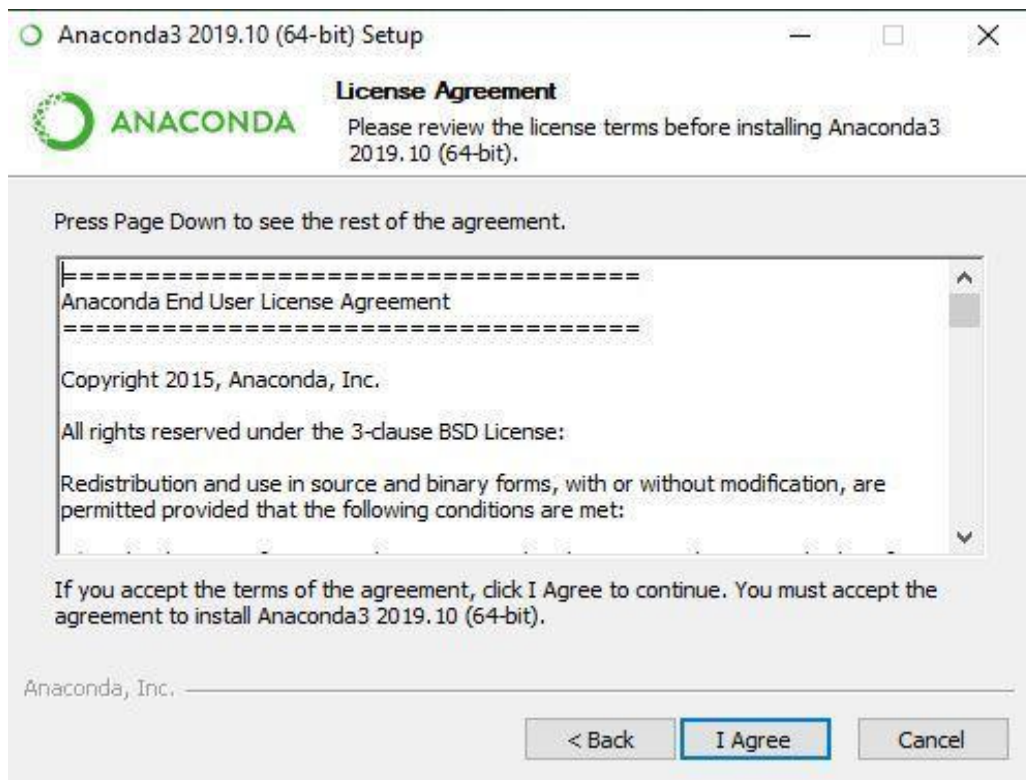
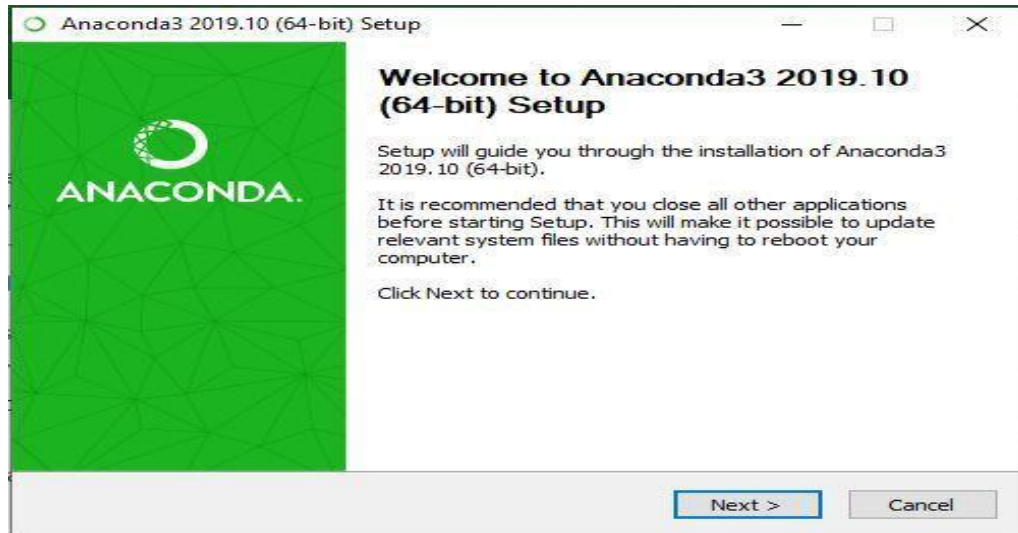
Follow the below instructions to Download and install Anaconda on system:

### Download and install Anaconda:

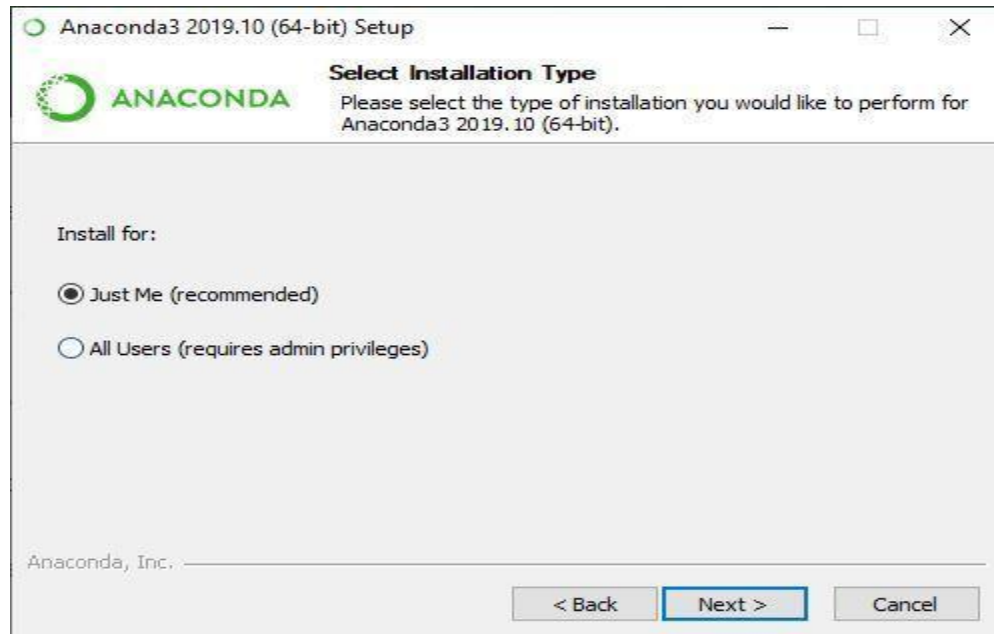
Head over to [anaconda.com](https://anaconda.com) and install the latest version of Anaconda. Make sure to download the “Python 3.7 Version” for the appropriate architecture.



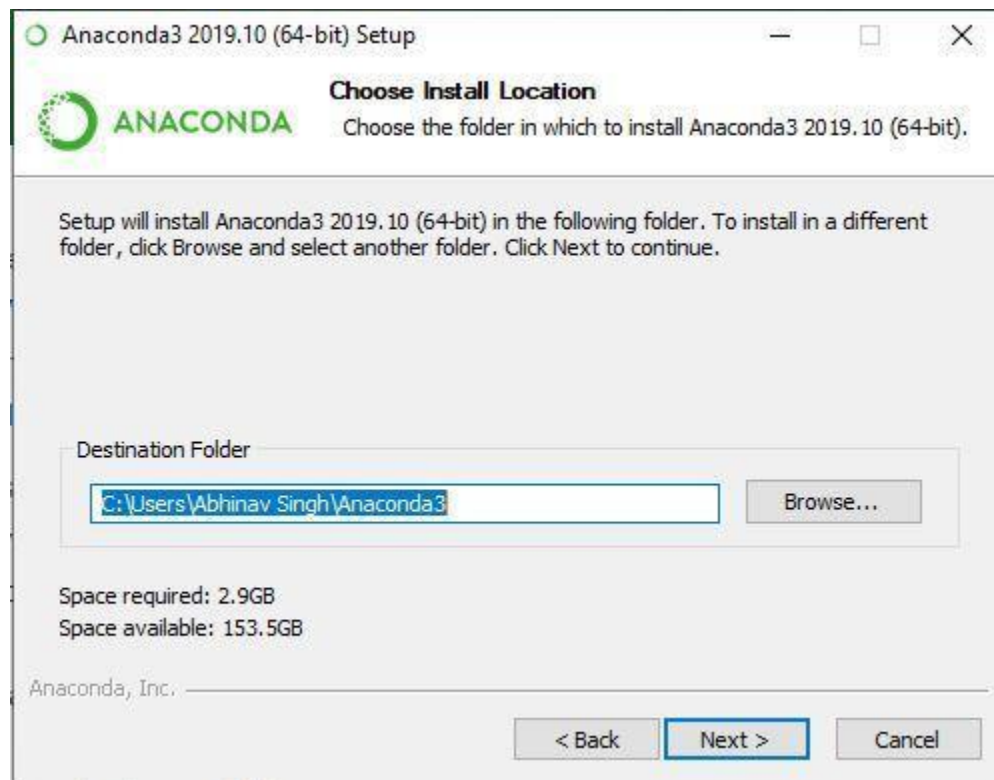
## The installation process:



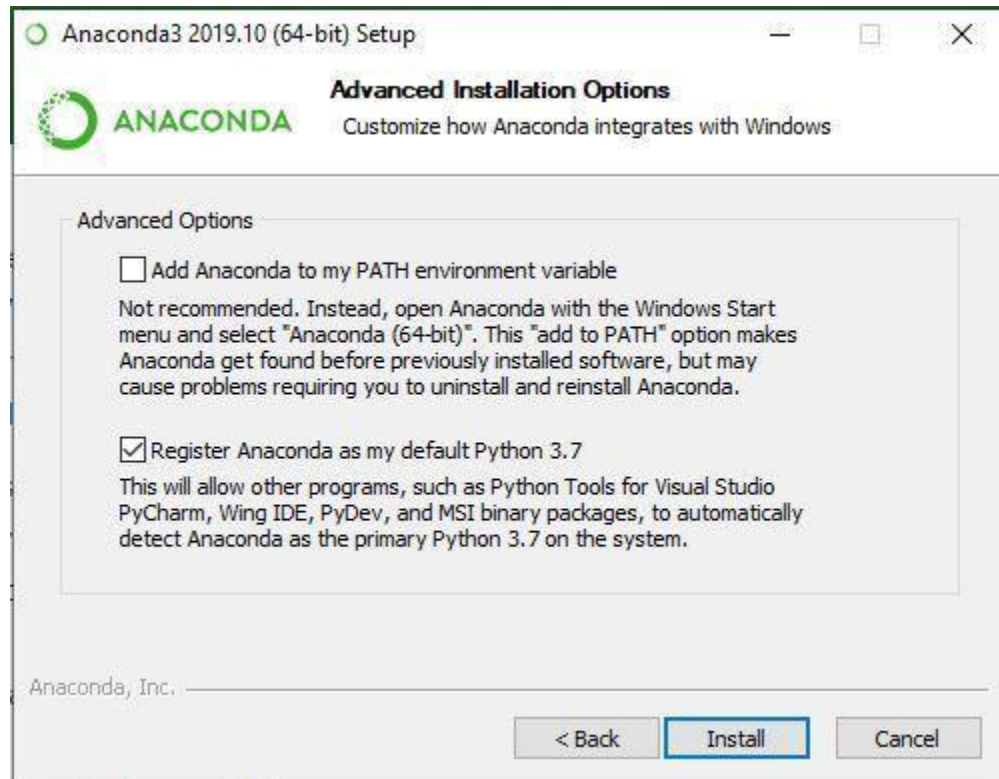
- **Select Installation Type:** Select **Just Me** if we want the software to be used by a single



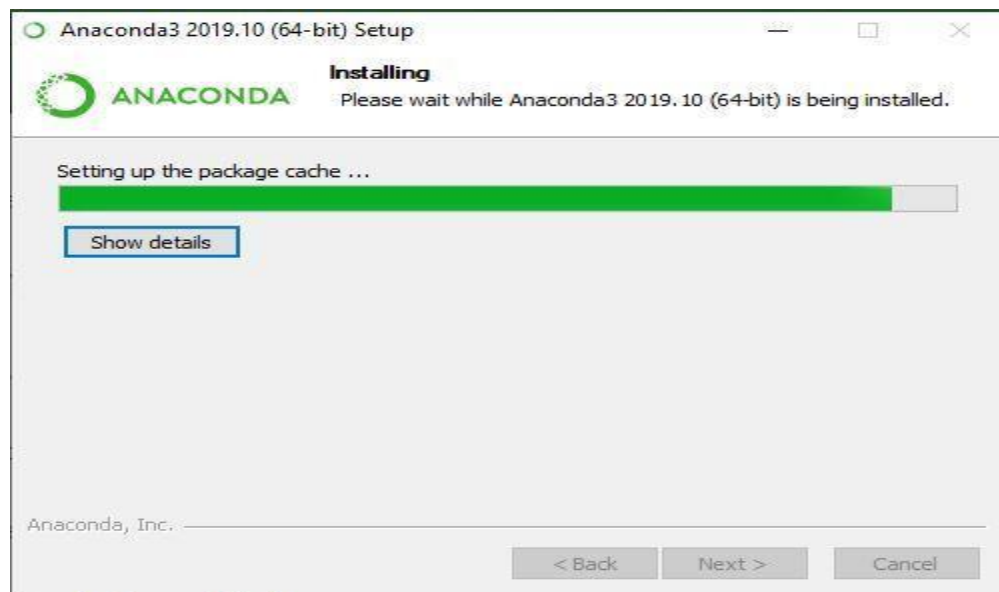
- **Choose Installation Location:**



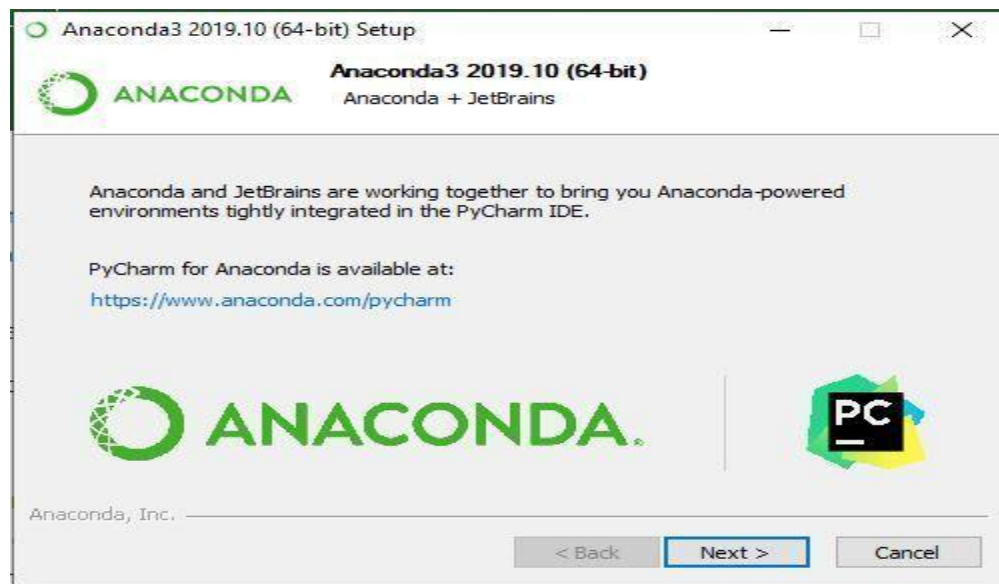
- **Advanced Installation Option:**



- **Getting through the Installation Process:**



- **Recommendation to Install Pycharm:**

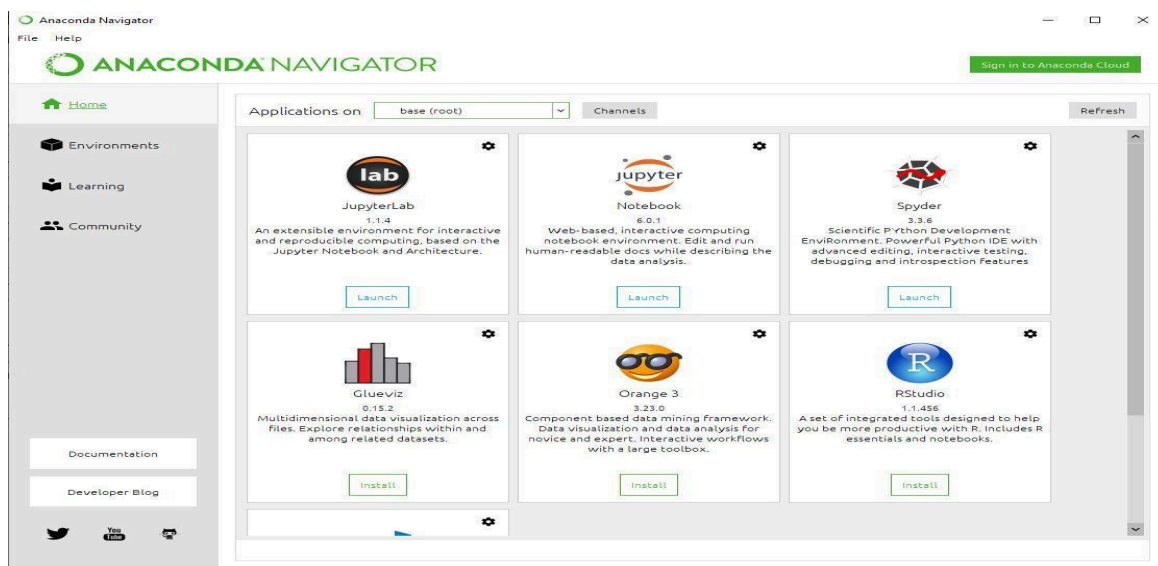
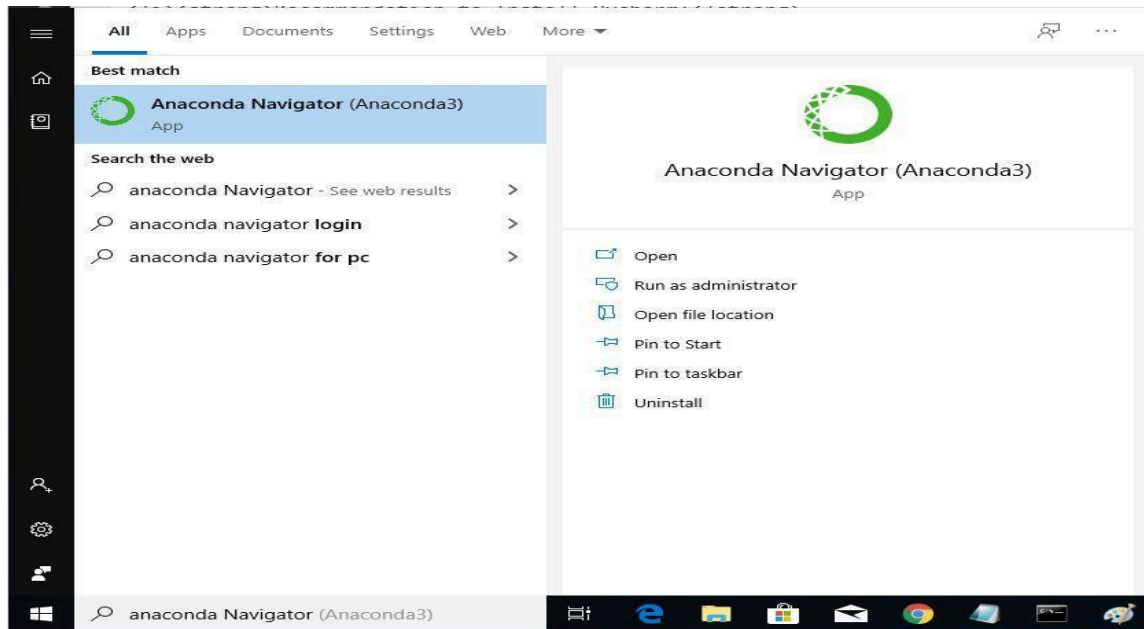


- **Finishing up the Installation:**



- **Working with Anaconda:**

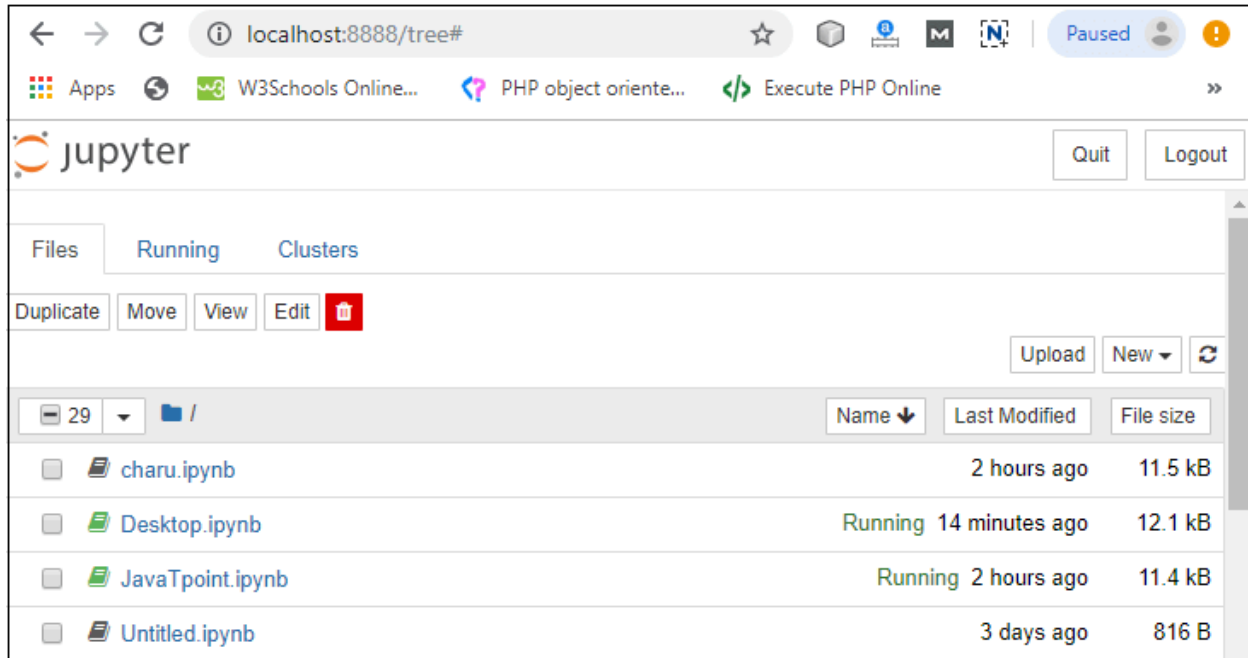
Once the installation process is done, Anaconda can be used to perform multiple operations. To begin using Anaconda, search for Anaconda Navigator from the Start Menu in Windows



Jupyter Notebook is an **open-source, web-based interactive environment**, which allows us to create and share documents that contain **live code, mathematical equations, graphics, maps, plots, visualizations, and narrative text**. It integrates with many programming languages like **Python, PHP, R, C#, etc.**

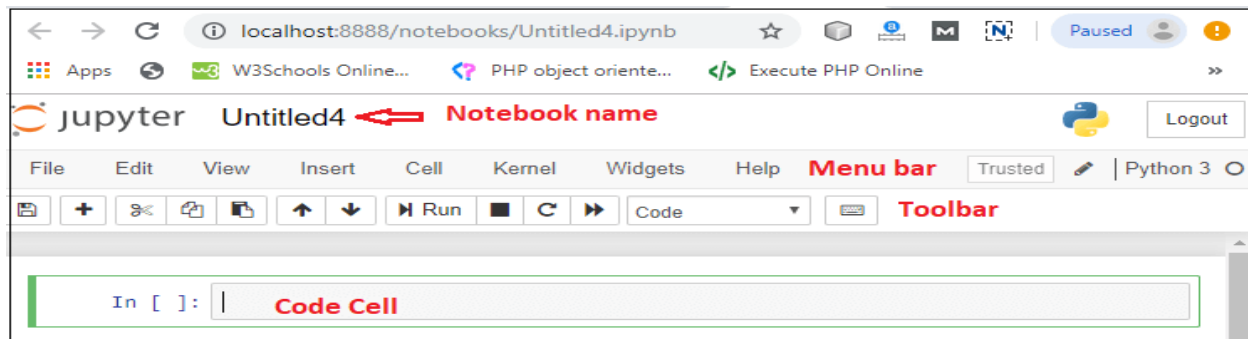


Jupyter notebook starts with the default web browser which shows the list of all python files.



### User interface of Jupyter Notebook

When we create a new notebook, the notebook will be presented with the **notebook name**, **menu bar**, **toolbar**, and an **empty code cell**.



**Notebook name:** Notebook name is displayed at the top of the page, next to the Jupyter logo.

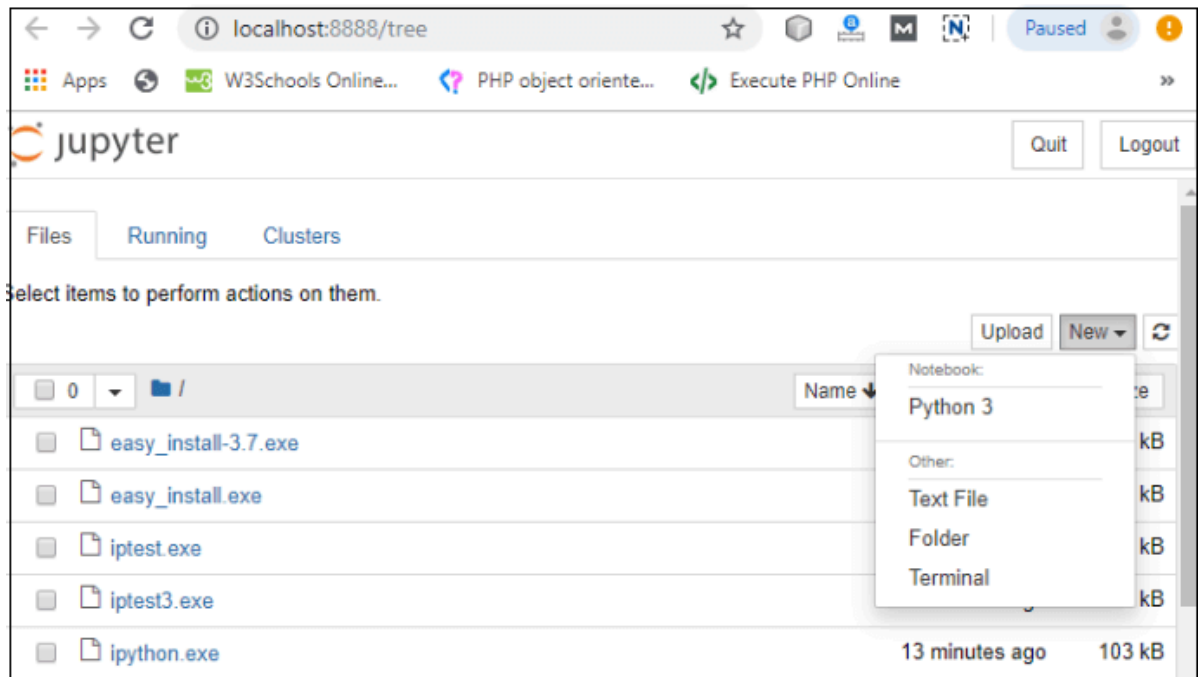
**Menu bar:** The menu bar presents different options that are used to manipulate the notebook functions.

**Toolbar:** The toolbar provides a quick way for performing the most-used operations within the notebook.

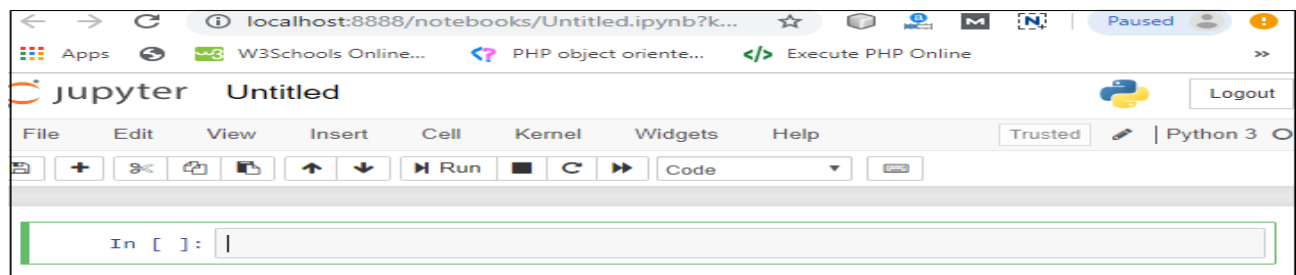
**Code cell:** A code cell allows us to edit and write a new code.

### Creating a Notebook

To create a Notebook in Jupyter, go to **New** and select **Python3**.



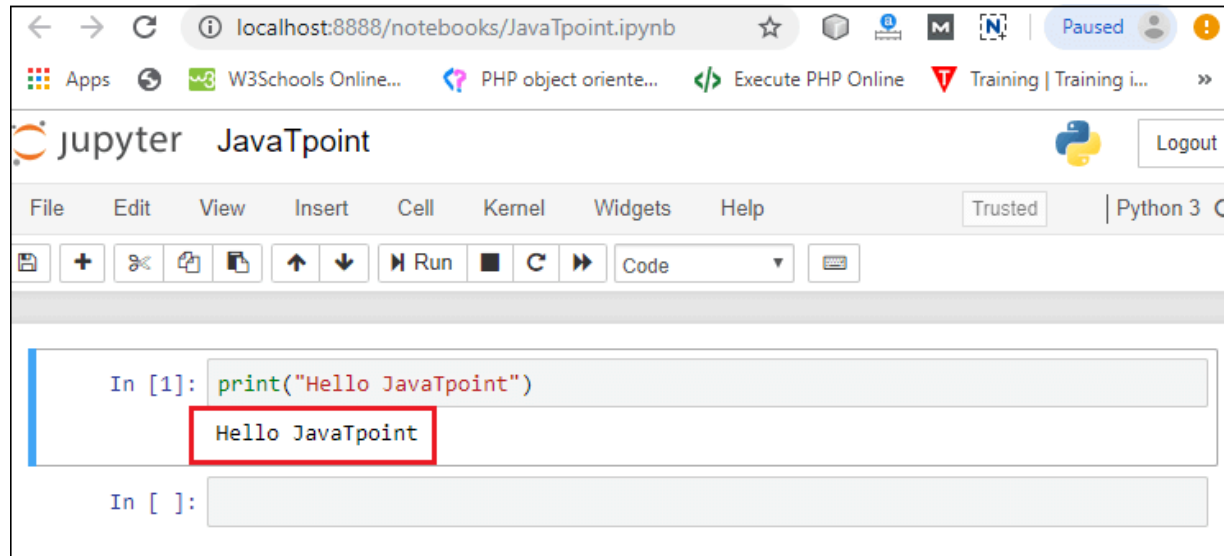
Now, we can see that a new notebook opens in a new tab.



### How to write and run a program in Jupyter

Click on the first cell in the notebook to enter in the edit mode. Now we can write the code in working area. After writing the code, we can run it by pressing the **Shift+ Enter** key or directly click on the **run button** at the top of the screen.





**Conclusion:** Thus, we studied installation process of Anaconda and features of Jupyter.

### Viva Questions

1.	How do you install Anaconda on your system?
2.	What is the purpose of the Anaconda Navigator?
3.	How do you manage packages in an Anaconda environment?
4.	How do you update Anaconda and its packages?
5.	How do you install Jupyter Notebook?
6.	How can you share your Jupyter Notebooks with others?
7.	Explain the concept of kernels in Jupyter Notebook.
8.	How can you install additional packages in a Jupyter Notebook?
9.	Explain how you can debug and profile code in Jupyter Notebook.

## Experiment No-02

**Title:** Download, install and explore the features of, Colab environment, understand the Anaconda

**Aim:** To understand the features of Colab environment.

**Theory:** Google Colab, short for Colaboratory, is a free cloud-based platform provided by Google that allows users to write and execute Python code collaboratively in a Jupyter Notebook environment.

### Benefits of Google Colab

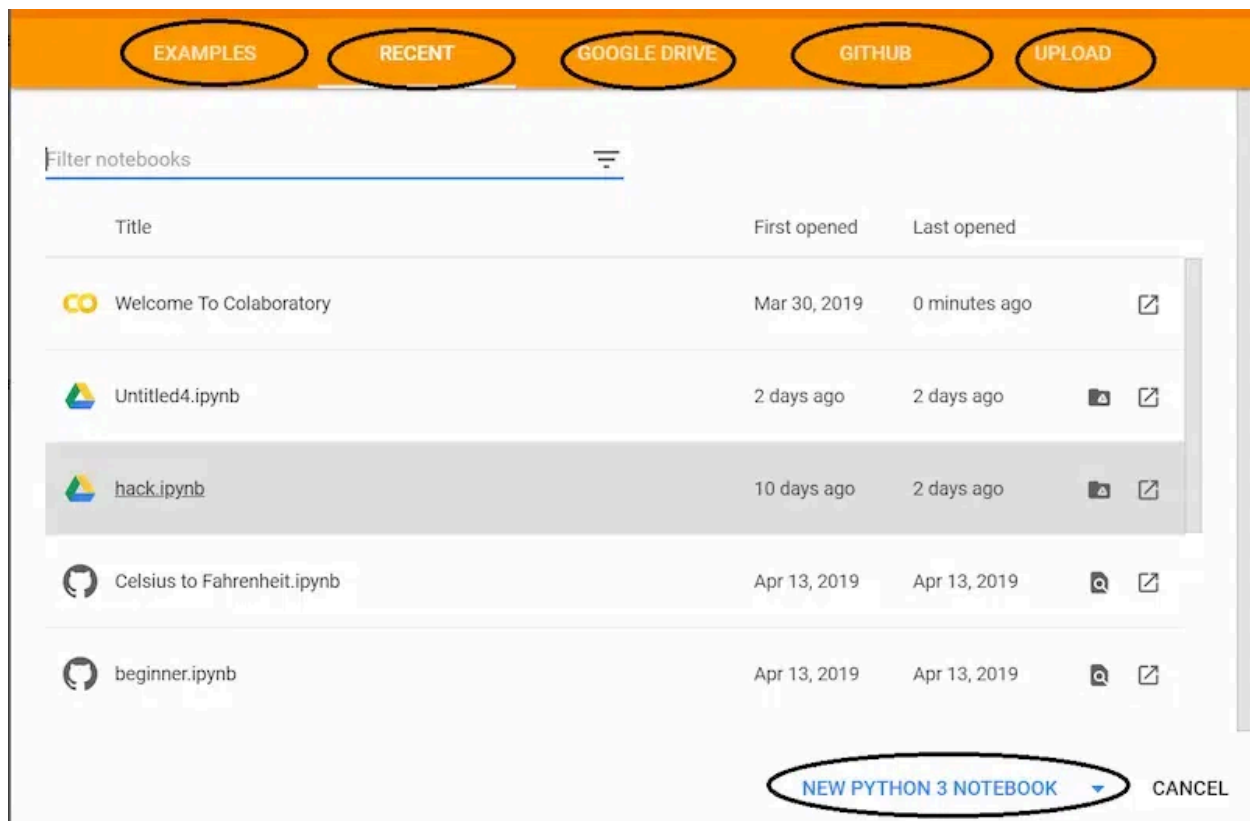
Google Colab offers several benefits that make it a popular choice among data scientists, researchers, and machine learning practitioners. **Key features of Google Collaboratory notebook include:**

- **Free Access to GPUs:** Colab offers free GPU access, which is particularly useful for training machine learning models that require significant computational power.
- **No Setup Required:** Colab runs in the cloud, eliminating the need for users to set up and configure their own development environment. This makes it convenient for quick coding and collaboration.
- **Collaborative Editing:** Multiple users can work on the same Colab notebook simultaneously, making it a useful tool for collaborative projects.
- **Integration with Google Drive:** Colab is integrated with Google Drive, allowing users to save their work directly to their Google Drive account. This enables easy sharing and access to notebooks from different devices.
- **Support for Popular Libraries:** Colab comes pre-installed with many popular Python libraries for machine learning, data analysis, and visualization, such as TensorFlow, PyTorch, Matplotlib, and more.
- **Easy Sharing:** Colab notebooks can be easily shared just like Google Docs or Sheets. Users can provide a link to the notebook, and others can view or edit the code in real-time.

## How to use Colaboratory?

### Open Collaboratory Notebook

On opening the website we will see a pop-up containing the following tabs –

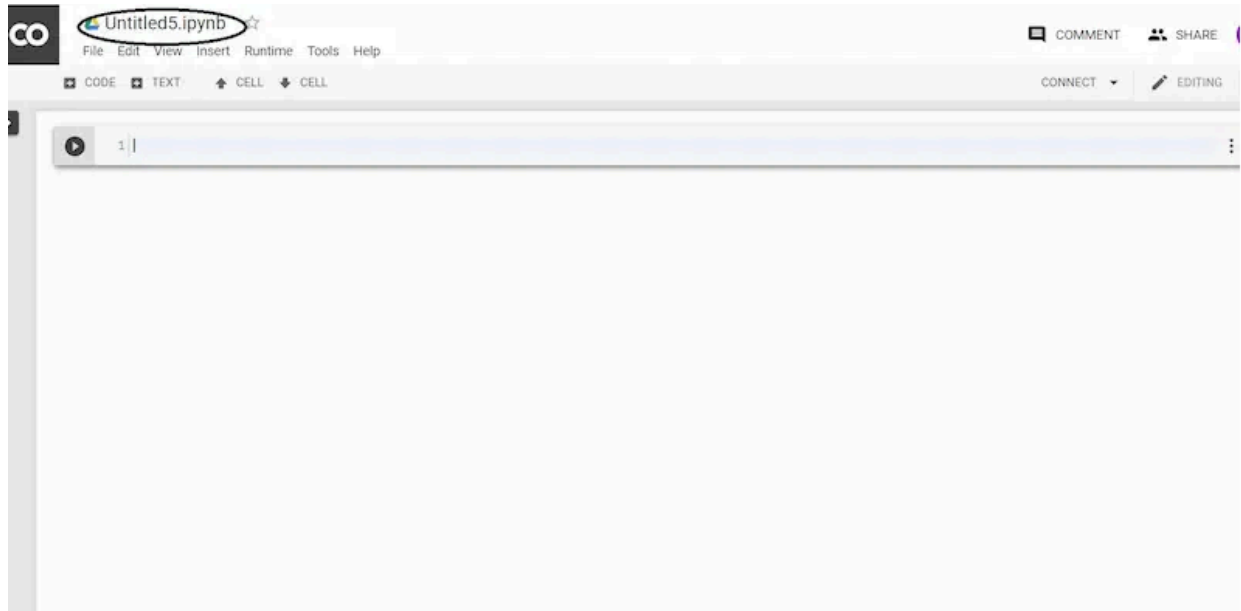


- **EXAMPLES:** Contain a number of Jupyter notebooks of various examples.
- **RECENT:** Jupyter notebook we have recently worked with.
- **GOOGLE DRIVE:** Jupyter notebook in our google drive.
- **GITHUB:** We can add Jupyter notebook from our GitHub but we first need to connect Colab with GitHub.
- **UPLOAD:** Upload from our local directory.

### Create Collaboratory Notebook

We can create a new Jupyter Notebook by clicking New Python3 Notebook or New Python2 Notebook at the bottom right corner.

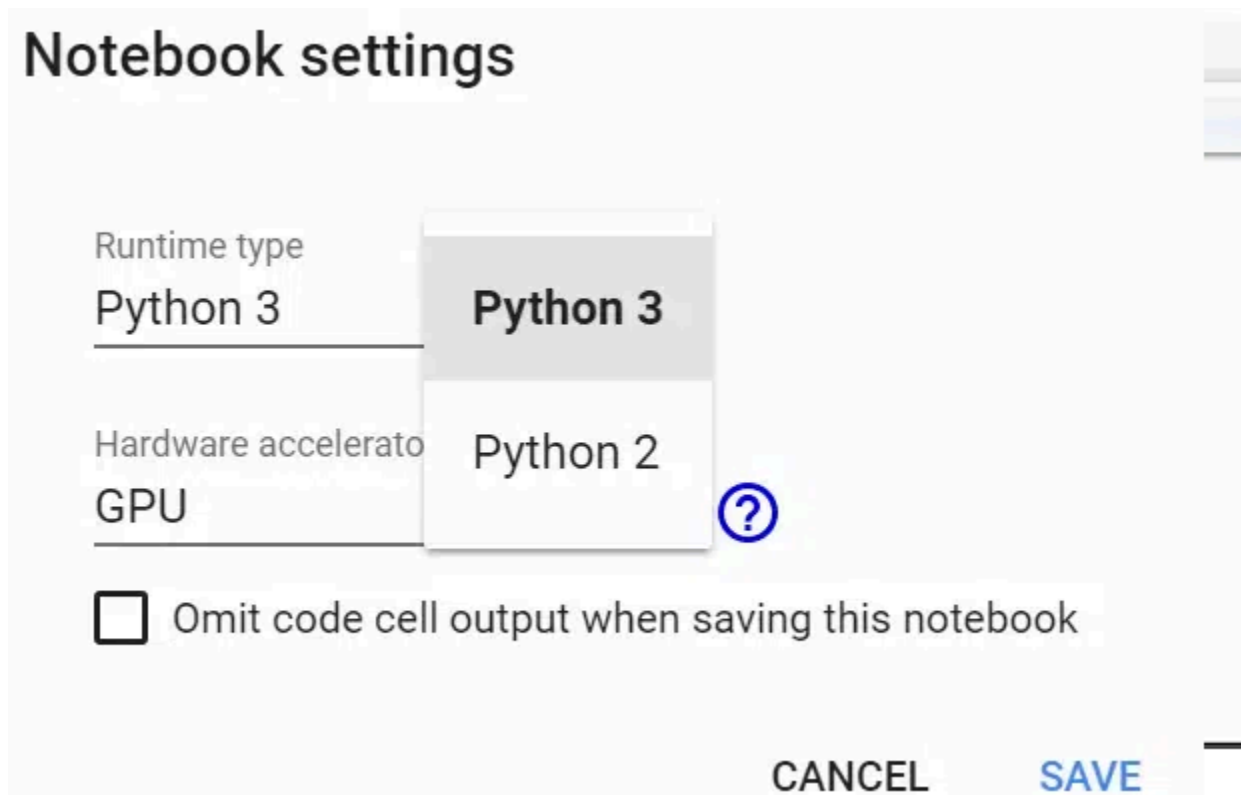
On creating a new notebook, it will create a Jupyter notebook with Untitled0.ipynb and save it to our google drive in a folder named **Colab Notebooks**.



Now as it is essentially a Jupyter Notebook, all commands of Jupyter Notebooks will work here.

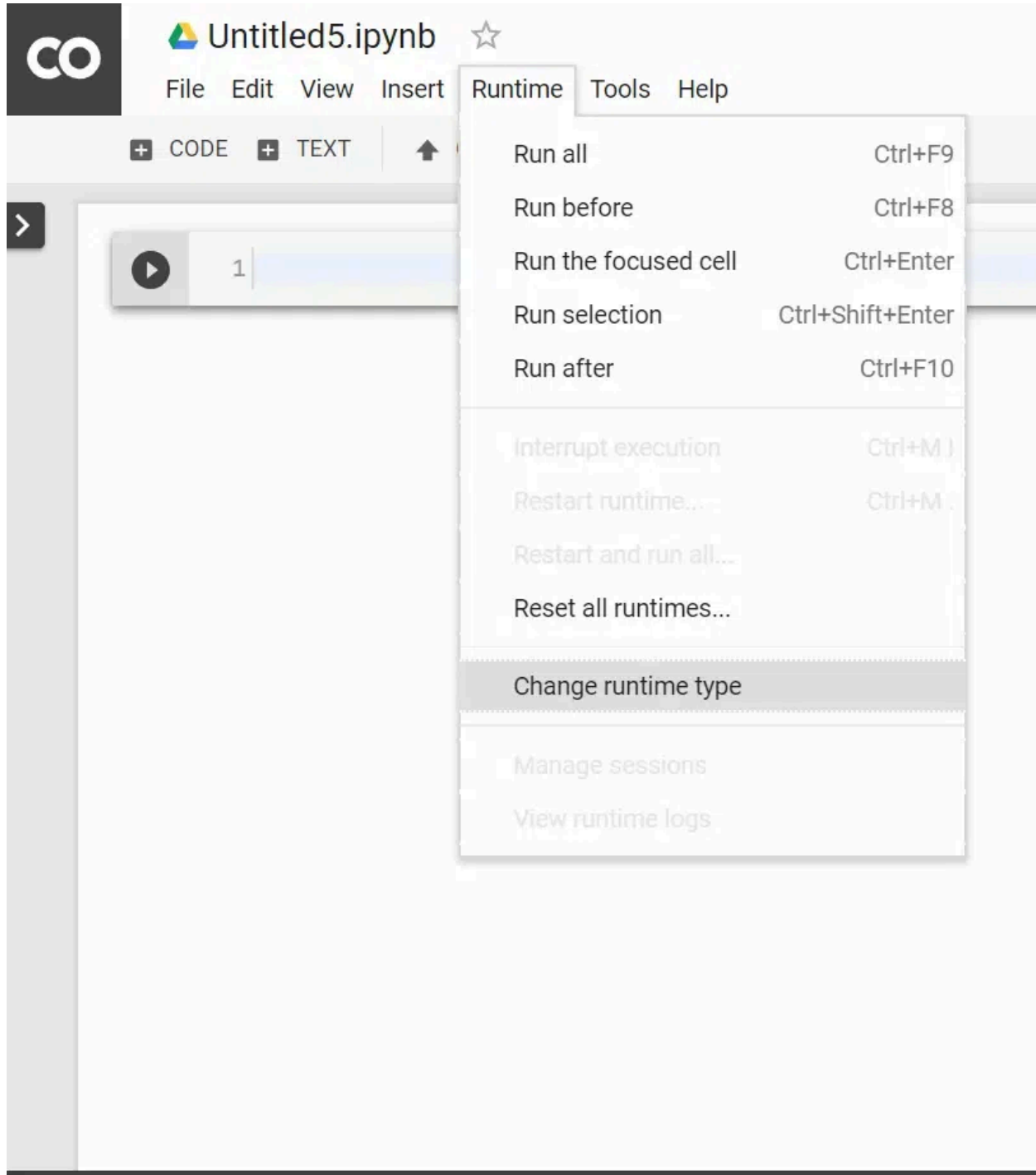
**what is different here:**

**Change Runtime Environment:** Click the “**Runtime**” dropdown menu. Select “**Change runtime type**”. Select python2 or 3 from the “**Runtime type**” dropdown menu.



## Use GPU and TPU

Click the “Runtime” dropdown menu. Select “Change runtime type”. Now select anything (GPU, CPU, None) we want in the “Hardware accelerator” dropdown menu.



## Notebook settings

Runtime type  
Python 3 ▼

Hardware accelerator  
None

☐ Omit code cell output when saving this notebook

None ?  
GPU  
TPU

CANCEL SAVE

### Install Python packages

We can use **pip** to install any package. For example:

**! pip install pandas**

### Clone GitHub repos in Google Colab

Use the git clone command. For example:

**! git clone https://github.com/souvik3333/Testing-and-Debugging-Tools**

### Upload File on Google Colab

```
from google.colab import files
```

```
uploaded = files.upload()
```

### **Upload File By Mounting Google Drive**

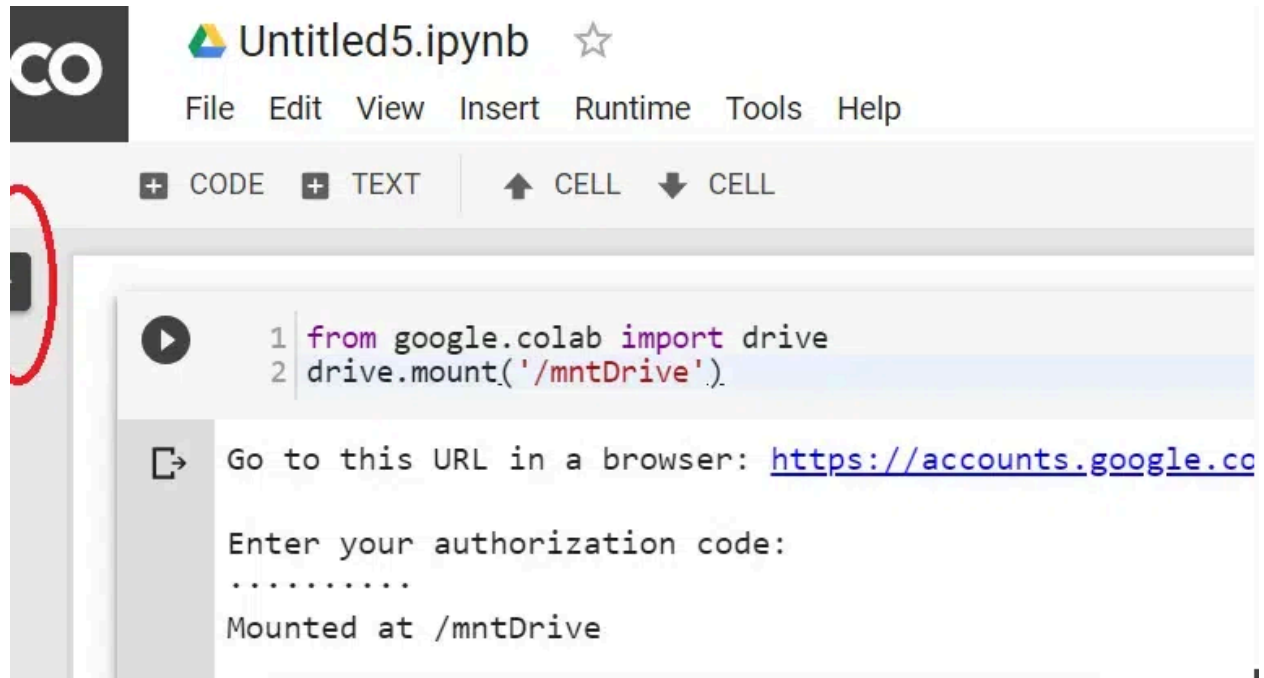
To mount our drive inside the “mntDrive” folder execute the following –

```
from google.colab import drive  
  
drive.mount('/mntDrive')
```

Then we’ll see a link, click on the link, then allow access, copy the code that pops up, and paste it at “Enter our authorization code:”. Now to see all data in our google drive we need to execute the following:

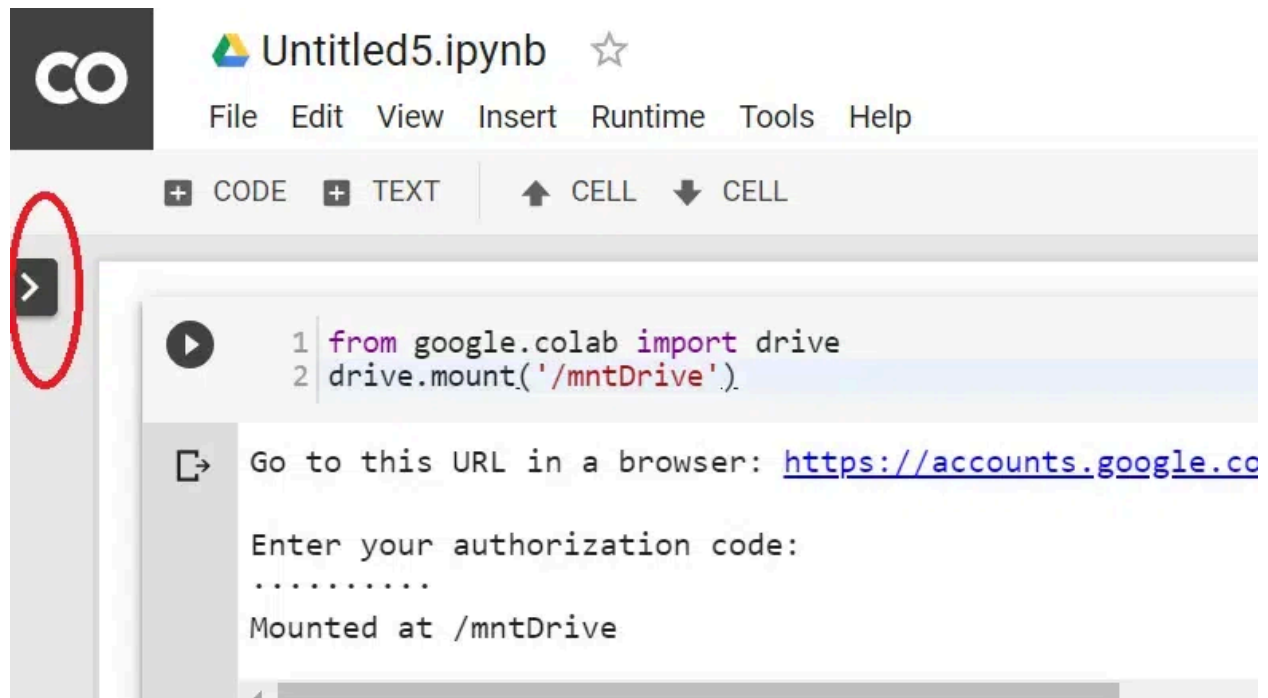
```
!ls '/mntDrive/My Drive'
```

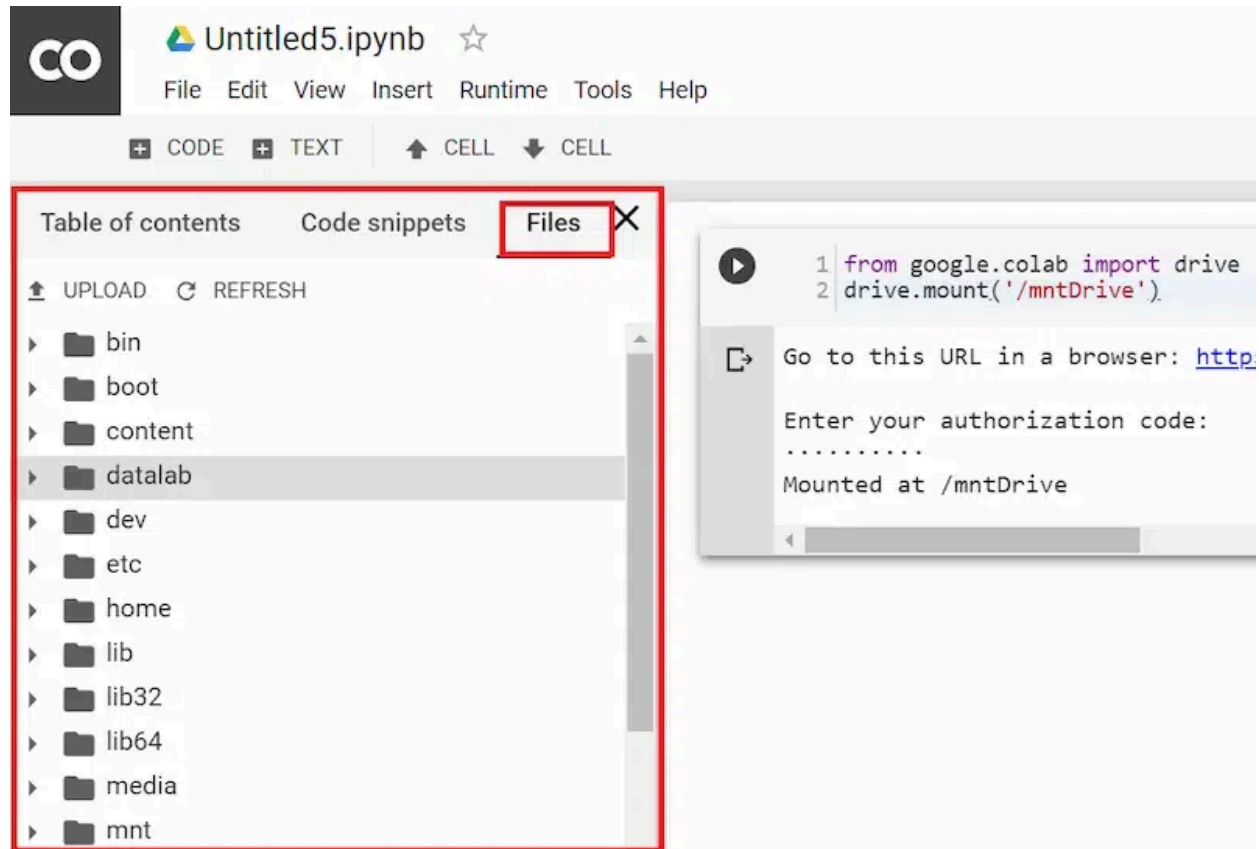




### File Hierarchy In Google Colab

We can also see the file hierarchy by clicking ">" at the top left below the control buttons (CODE, TEXT, CELL).



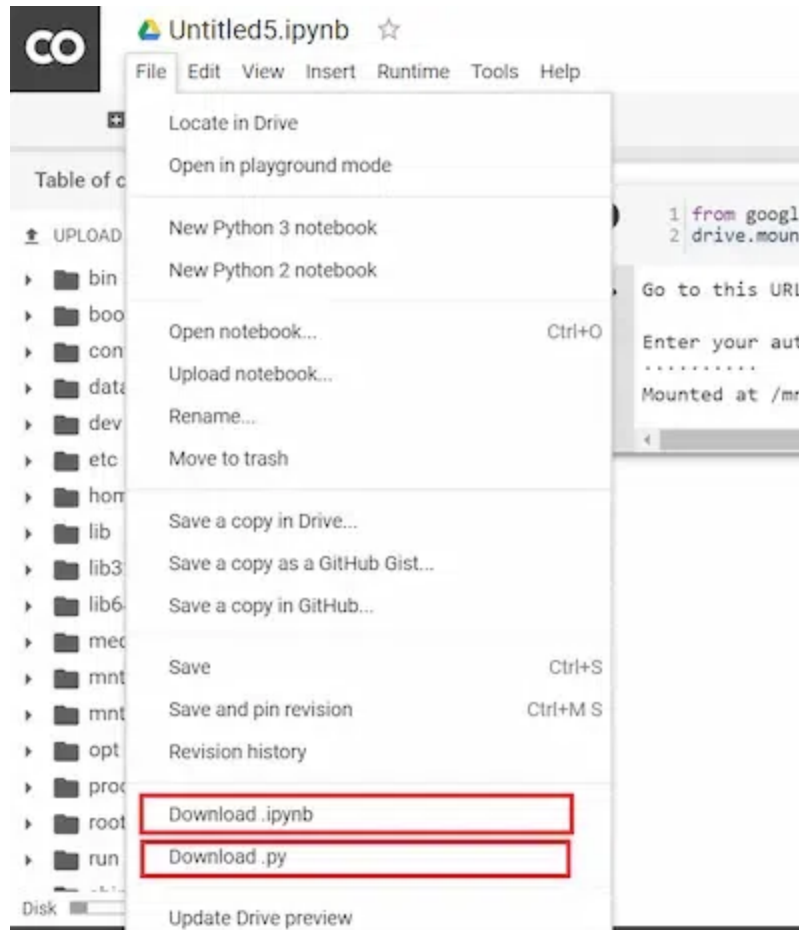


### Download Files from Google Colab

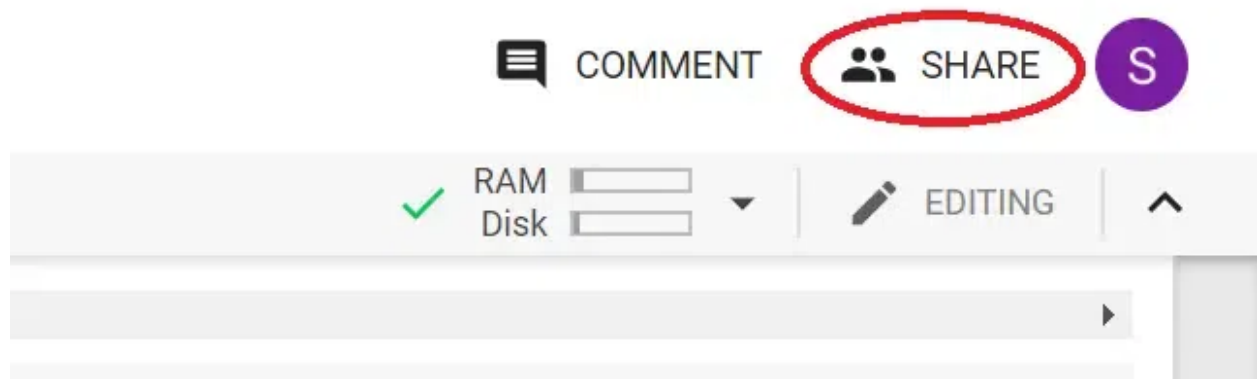
Let's say we want to download "file\_name.csv". We can copy the file to our google drive (In "data" folder, we need to create the "data" folder in google drive) by executing this:

```
cp file_name.csv "/mntDrive/My Drive/data/renamed_file_name.csv"
```


The file will be saved in the "data" folder with the "renamed\_file\_name.csv" name. Now we can directly download from there, Or, we can just open the file hierarchy and right-clicking will give a download option. Download Jupyter Notebook: Click the "File" dropdown menu at the top left corner. Choose "download .ipynb" or "download .py"



**Share Jupyter Notebook:** We can share our notebook by adding others' email addresses or by creating a shareable link.



Share with others

Get shareable link 

Link sharing on [Learn more](#)


Anyone with the link can view ▼

Copy link

https://colab.research.google.com/drive/1CoJj7SOiwDOurVALMMKE-GhOYJTdP8

People

Enter names or email addresses...



Done

Advanced

## Conclusion

In conclusion, Google Colab stands out as a versatile and accessible platform for Python coding. Thus, we explored a features of google Colaboratory.

## Viva Questions

1.	How to Load a Dataset From the Google Drive to Google Colab
2.	Explain Ways to import CSV files in Google Colab
3.	How to Upload Project on GitHub from Google Colab?
4.	How to Delete a locally Uploaded File on Google Colab
5.	How to Import Kaggle Datasets Directly into Google Colab?

### **Experiment No.3**

**Title:** Working With Numpy Arrays.

**Aim:** To study Numpy Arrays

**Theory:** NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and we can use it freely. NumPy stands for Numerical Python.

In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called `ndarray`, it provides a lot of supporting functions that make working with `ndarray` very easy. Arrays are very frequently used in data science, where speed and resources are very important.

### Installation of Numpy Using PIP

Open **command prompt or terminal** and run the following command:

```
pip install numpy.
```

### Import NumPy

Once NumPy is installed, import it in our applications by adding the `import` keyword:

```
import numpy
```

### NumPy as np

NumPy is usually imported under the `np` alias.

**alias:** In Python alias are an alternate name for referring to the same thing.

**Create an alias with the `as` keyword while importing:**

```
import numpy as np
```

For Example:

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
print(arr)
```

Output: [1 2 3 4 5]

### Checking NumPy Version

The version string is stored under `__version__` attribute.

For Example:

```
import numpy as np
```

```
print(np.__version__)
```

```
Output: 1.16.3
```

### Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called **ndarray**.

We can create a NumPy **ndarray** object by using the **array()** function.

For Example:

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
print(arr)
```

```
print(type(arr))
```

```
Output: [1 2 3 4 5]
```

```
<class 'numpy.ndarray'>
```

### Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

#### 0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

For Example:

```
import numpy as np
```

```
arr = np.array(42)
```

```
print(arr)
```

```
Output: 42
```



## 1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array. These are the most common and basic arrays.

Example:

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4, 5])
```

```
print(arr)
```

```
Output: [1 2 3 4 5]
```

## 2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix or 2nd order tensors.

For Example:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
```

```
print(arr)
```

```
Output: [[1 2 3]
```

```
[4 5 6]]
```

## 3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

For Example:

```
import numpy as np
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]])
```

```
print(arr)
```

```
Output: [[1 2 3]
```

```
 [4 5 6]]
```

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

```
 [4 5 6]]]
```

## NumPy Array Indexing

### Access Array Elements

Array indexing is the same as accessing an array element. We can access an array element by referring to its index number. The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

For Example:

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4])
```

```
print(arr[0])
```

```
Output: 1
```

### Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Example

```
import numpy as np
```

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
```

```
print('2nd element on 1st row: ', arr[0, 1])
```

```
Output: 2nd element on 1st dim: 2
```

## Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

For Example:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

print(arr[0, 1, 2])
```

Output: 6

## Negative Indexing

Use negative indexing to access an array from the end.

For Example:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('Last element from 2nd dim: ', arr[1, -1])
```

Output: 10

## Data Types in NumPy

NumPy has some extra data types, and refer to data types with one character, like **i** for integers, **u** for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

- i - integer
- b - boolean
- u - unsigned integer
- f - float
- c - complex float
- m - timedelta

- M - datetime
- O - object
- S - string
- U - unicode string
- V - fixed chunk of memory for other type ( void )

### Checking the Data Type of an Array

For Example:

```
import numpy as np
```

```
arr = np.array([1, 2, 3, 4])
```

```
print(arr.dtype)
```

Output: int64

**Conclusion:** Thus we studied Numpy in python.

### Viva Questions

1.	What is NumPy, and why is it popular in the field of scientific computing?
2.	How does NumPy differ from Python lists, and what advantages does NumPy offer?
3.	What is the purpose of the np.arange() function in NumPy?
4.	How can you find the dimensions and shape of a NumPy array?
5.	How can you concatenate two NumPy arrays horizontally and vertically?
6.	Discuss the purpose of the np.reshape() function in NumPy.
7.	Describe the advantages of using NumPy arrays over Python lists for numerical computations.
8.	What is the purpose of the NumPy dtype attribute? How does it influence array memory allocation?
9.	Discuss the concept of NumPy slicing. How does slicing differ from indexing?
10.	How do you check for an empty (zero Element) array?

## Experiment No.4

**Title:** Working with Pandas data frames

**Aim:** To study working with Pandas data frames

**Theory:** Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

### Installing Pandas

The first step in working with Pandas is to ensure whether it is installed in the system or not. If not, then we need to install it on our system using the **pip command**.

Follow these steps to install Pandas:

**Step 1:** Type 'cmd' in the search box and open it.

**Step 2:** Locate the folder using the cd command where the **python-pip file** has been installed.

**Step 3:** After locating it, type the command:

```
pip install pandas
```

### Importing Pandas

After the Pandas have been installed in the system, we need to import the library. This module is generally imported as follows:

```
import pandas as pd
```

### Checking Pandas Version

The version string is stored under `__version__` attribute.

For Example:

```
import pandas as pd
```

```
print(pd.__version__)
```

```
Output: 1.0.3
```

## Data Structures in Pandas Library

Pandas generally provide two data structures for manipulating data. They are:

- **Series**
- **DataFrame**

### Pandas Series

A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type.

For Example:

```
import pandas as pd
```

```
a = [1, 7, 2]
```

```
myvar = pd.Series(a)
```

```
print(myvar)
```

```
Output: 0    1
```

```
1    7
```

```
2    2
```

```
dtype: int64
```

### Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc. This label can be used to access a specified value. With the **index** argument, we can name our own labels.

For Example:

```
import pandas as pd
```

```
a = [1, 7, 2]
```

```
myvar = pd.Series(a, index = ["x", "y", "z"])
```

```
print(myvar["y"])
```

Output:7

## DataFrames

Data sets in Pandas are usually multi-dimensional tables, called DataFrames. Series is like a column, a DataFrame is the whole table.

For Example:

```
import pandas as pd
```

```
data = {  
    "calories": [420, 380, 390],  
    "duration": [50, 40, 45]  
}
```

```
myvar = pd.DataFrame(data)
```

```
print(myvar)
```

Output: calories duration

0	420	50
1	380	40
2	390	45

## Locate Row

As we can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the `loc` attribute to return one or more specified row(s)

For Example:

```
#refer to the row index:  
print(df.loc[0])
```



Output:

```
calories    420
```

```
duration     50
```

```
Name: 0, dtype: int64
```

## Named Indexes

With the **index** argument, we can name our own indexes.

For Example:

```
import pandas as pd
```

```
data = {  
    "calories": [420, 380, 390],  
    "duration": [50, 40, 45]  
}
```

```
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
```

```
print(df)
```

Output:

```
   calories  duration  
day1     420       50  
day2     380       40  
day3     390       45
```

## Load Files in to a DataFrame

```
import pandas as pd
```

```
df = pd.read_csv('data.csv')
```

```
print(df)
```

## Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files). CSV files contain plain text and is a well known format that can be read by everyone including Pandas.

In our examples we will be using a CSV file called 'data.csv'.

[Download data.csv](#). or [Open data.csv](#)

For Example:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())

max_rows
```

The number of rows returned is defined in Pandas option settings.

We can check our system's maximum rows with the `pd.options.display.max_rows` statement.

For Example:

```
import pandas as pd

print(pd.options.display.max_rows)
```

Example

Increase the maximum number of rows to display the entire DataFrame:

```
import pandas as pd

pd.options.display.max_rows = 9999

df = pd.read_csv('data.csv')

print(df)
```

Conclusion: Thus we studied Pandas in python.

### Viva Questions

1.	Define Python Pandas.
2.	What Are The Different Types Of Data Structures In Pandas?
3.	Explain Series In Pandas.
4.	Define Dataframe In Pandas.
5.	How Can You Create An Empty Dataframe In Pandas?
6.	What Are The Most Important Features Of The Pandas Library?
7.	How Will You Add An Index, Row, Or Column To A Dataframe In Pandas?
8.	What Method Will You Use To Rename The Index Or Columns Of Pandas Dataframe?
9.	Distinguish features between Series and DataFrame.
10.	How Can A Dataframe Be Converted To An Excel File?

## Experiment No.5

**Title:** Perform various types of data cleaning operations on the data collected in previous lab using data exploration, imputation etc.

**Aim:** To study data cleaning operations in python.

**Theory:** Data cleaning is an essential step in the data preprocessing pipeline for any data science or analytics project. Messy, inconsistent, or missing data can lead to inaccurate insights and model predictions.

### Why Data Cleaning is important

1. **Accuracy:** Clean data ensures that your analysis and machine learning models are based on accurate and reliable information.
2. **Consistency:** Inconsistent data can lead to errors, especially when working with categorical variables, date formats, or units of measurement.
3. **Completeness:** Missing data can cause issues in analysis and modeling. Handling missing values is an essential part of data cleaning.

### Python Libraries for Data Cleaning

Python offers several powerful libraries for data cleaning we'll primarily use two popular ones:

- **Pandas:** A versatile library for data manipulation and analysis. It provides tools for cleaning, transforming, and analyzing data.
- **NumPy:** A fundamental package for numerical computations in Python. It's often used alongside Pandas for data cleaning tasks.

### Data Cleaning Techniques with Python

#### 1. Handling Missing Values

Missing data is a common issue in datasets. Pandas provides methods like **isna()**, **fillna()**, and **dropna()** to handle missing values.

```
import pandas as pd
# Load our dataset
data = pd.read_csv('our_dataset.csv')
# Check for missing values
missing_values = data.isna().sum()
# Fill missing values
data.fillna()
# Drop rows with missing values
data.dropna(inplace=True)
```

## 2. Removing Duplicates

Duplicate records can skew our analysis. We can remove duplicates using the `drop_duplicates()` method.

```
# Remove duplicate rows
data.drop_duplicates(inplace=True)
```

## 3. Correcting Data Types

Ensure that columns have the correct data types. Use the `astype()` method to convert data types.

```
# Convert a column to datetime
data['date_column'] = pd.to_datetime(data['date_column'])

# Convert a column to numeric
data['numeric_column'] = data['numeric_column'].astype(float)
```

## 4. Standardizing Text Data

Text data often requires cleaning, including removing extra spaces and converting to lowercase.

```
# Remove leading and trailing spaces
data['text_column'] = data['text_column'].str.strip()

# Convert text to lowercase
data['text_column'] = data['text_column'].str.lower()
```

## 5. Handling Outliers

Outliers can impact statistical analysis. Use NumPy and Pandas to detect and deal with outliers.

```
import numpy as np

# Detect outliers using z-scores
z_scores = np.abs((data['numeric_column'] - data['numeric_column'].mean()) /
data['numeric_column'].std())

# Remove outliers (e.g., values with z-score > 3)
data = data[z_scores <= 3]
```

## 6. Handling Text Data

Cleaning text data often involves removing special characters, punctuation, and unnecessary white spaces.

```
import re

# Remove special characters and punctuation
data['text_column'] = data['text_column'].apply(lambda x: re.sub(r'[^w\s]', '', x))

# Remove extra white spaces
data['text_column'] = data['text_column'].apply(lambda x: ' '.join(x.split()))
```

## 7. Dealing with Inconsistent Data

Sometimes, data entries are inconsistent. We can standardize them using Pandas' `replace()` function.

```
# Standardize country names
data['country'] = data['country'].replace({'USA': 'United States', 'UK': 'United Kingdom'})
```

## 8. Handling Categorical Data

Encoding categorical data into numerical values is essential for machine learning. Use Pandas' `get_dummies()` or `LabelEncoder` for this task.

```
# Convert categorical data to numerical using one-hot encoding
data = pd.get_dummies(data, columns=['category_column'])
```

```
# Alternatively, use LabelEncoder for ordinal data
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
data['ordinal_column'] = label_encoder.fit_transform(data['ordinal_column'])
```

## 9. Data Imputation

Impute missing values using various strategies such as mean, median, mode, or custom values.

```
# Fill missing values with the median
data['numeric_column'].fillna(data['numeric_column'].median(), inplace=True)

# Impute missing categorical values with a custom label
data['category_column'].fillna('Unknown', inplace=True)
```

## 10. Data Validation

Check for outliers, unrealistic values, or data that violates expected constraints.

```
# Detect outliers in numeric data using box plots
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x=data['numeric_column'])
plt.show()

# Filter out unrealistic values
data = data[(data['numeric_column'] >= 0) & (data['numeric_column'] <= 100)]
```

**Conclusion:** Thus we studied data cleaning operations in python.

## Viva Questions

1.	What is data cleaning? How can it be done in python?
2.	Can you explain why data cleansing is important for machine learning models?
3.	Is it possible to detect missing values from a data set without actually going through each row manually? If yes, then how?
4.	What's the difference between categorical data and continuous data?



5.	In which situations should you opt for imputation over deletion?
6.	What are the various ways in which you can address missing data in a data set?
7.	What are the steps involved in data validation?

## Experiment No.6

**Title:** Perform dimensionality reduction on a given dataset and create various visualizations like histograms, scatter-plots, etc.

**Aim:** To study dimensionality reduction in python

### Theory:

Dimensionality reduction is a technique used to reduce the number of features in a dataset while retaining as much of the important information as possible. In other words, it is a process of transforming high-dimensional data into a lower-dimensional space that still preserves the essence of the original data.

In machine learning, high-dimensional data refers to data with a large number of features or variables. The curse of dimensionality is a common problem in machine learning, where the performance of the model deteriorates as the number of features increases. This is because the complexity of the model increases with the number of features, and it becomes more difficult to find a good solution. In addition, high-dimensional data can also lead to overfitting, where the model fits the training data too closely and does not generalize well to new data.

### Components of Dimensionality Reduction

There are two components of dimensionality reduction:

- **Feature selection:** In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
  1. Filter
  2. Wrapper
  3. Embedded
- **Feature extraction:** This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

### Methods of Dimensionality Reduction

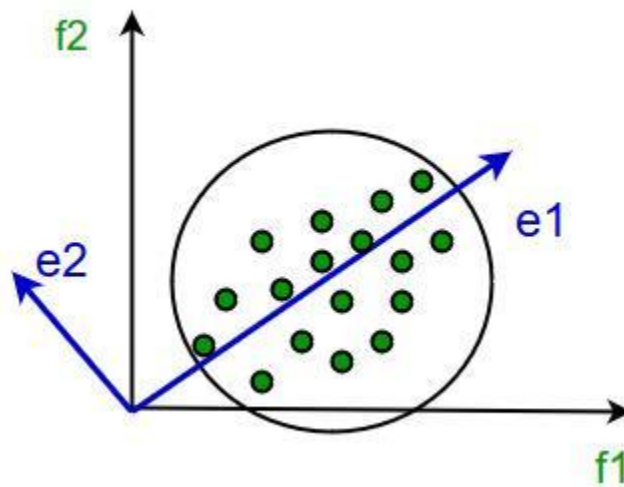
The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

Dimensionality reduction may be both linear and non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

### Principal Component Analysis

This method was introduced by Karl Pearson. It works on the condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.



#### It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

#### Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.
- Improved Visualization: High dimensional data is difficult to visualize, and dimensionality reduction techniques can help in visualizing the data in 2D or 3D, which can help in better understanding and analysis.
- Overfitting Prevention.
- Feature Extraction.
- Data Preprocessing.

- Improved Performance

### Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.
- Overfitting: In some cases, dimensionality reduction may lead to overfitting, especially when the number of components is chosen based on the training data.
- Sensitivity to outliers: Some dimensionality reduction techniques are sensitive to outliers, which can result in a biased representation of the data.
- Computational complexity: Some dimensionality reduction techniques, such as manifold learning, can be computationally intensive, especially when dealing with large datasets.

The python code is as follows (PCA)

```
#import the libraries

from sklearn.datasets import load_iris

from sklearn.decomposition import PCA

import pylab as pl

from itertools import cycle

%matplotlib inline
```

```
#loading the iris data set
iris=load_iris()
```

```
#understanding dataset
numSamples,numFeatures=iris.data.shape
print(numSamples)
print(numFeatures)
print(iris.target_names)
```

**Output:** 150  
4  
['setosa' 'versicolor' 'virginica']

```
x=iris.data
pca=PCA(n_components=2,whiten=True).fit(x)
pca_x=pca.transform(x)
pca_x
```

```
print(pca.components_)
```

**Output:**

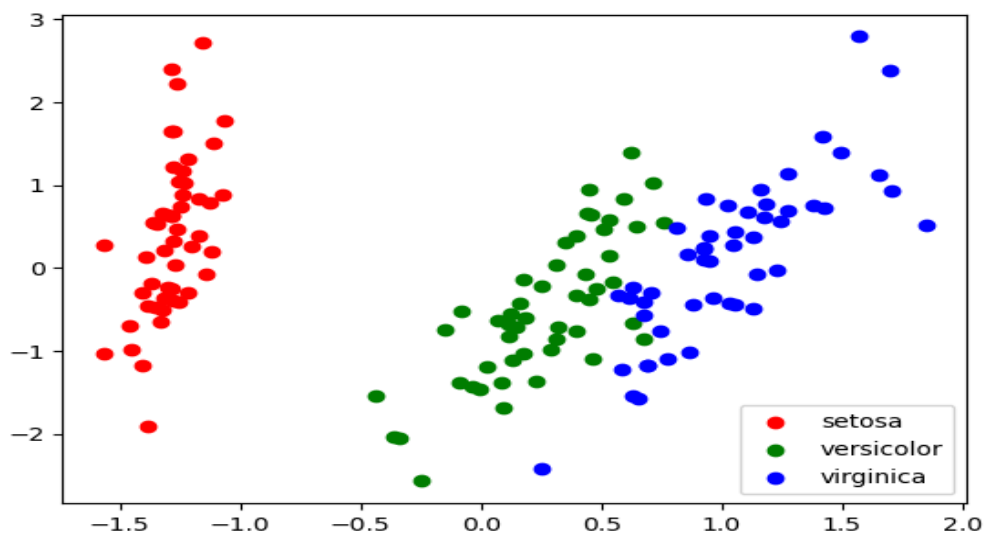
```
[[ 0.36138659 -0.08452251  0.85667061  0.3582892 ]
 [ 0.65658877  0.73016143 -0.17337266 -0.07548102]]
```

```
print(pca.explained_variance_ratio_)
print(sum(pca.explained_variance_ratio_))
```

**Output:**

```
[0.92461872 0.05306648]
0.977685206318795
```

```
#some visualizations here
colors=cycle('rgb')
target_ids=range(len(iris.target_names))
pl.figure()
for i,c,label in zip(target_ids,colors,iris.target_names):
    pl.scatter(pca_x[iris.target == i,0],pca_x[iris.target == i,1],c=c,label=label)
pl.legend()
pl.show()
```



**Conclusion:** Thus we studies one of the dimensionality reduction technique(PCA) in python.

## Experiment No.7

**Title:** Implement Linear and logistic Regression

**Aim:** To study and implement Linear and logistic Regression in python

**Theory:**

**Linear Regression:**

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between the dependent and independent variables, they are considering and the number of independent variables being used.

**Step 1:** Importing all the required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

**Step 2:** Reading the dataset

```
df = pd.read_csv("area.csv")
df
```

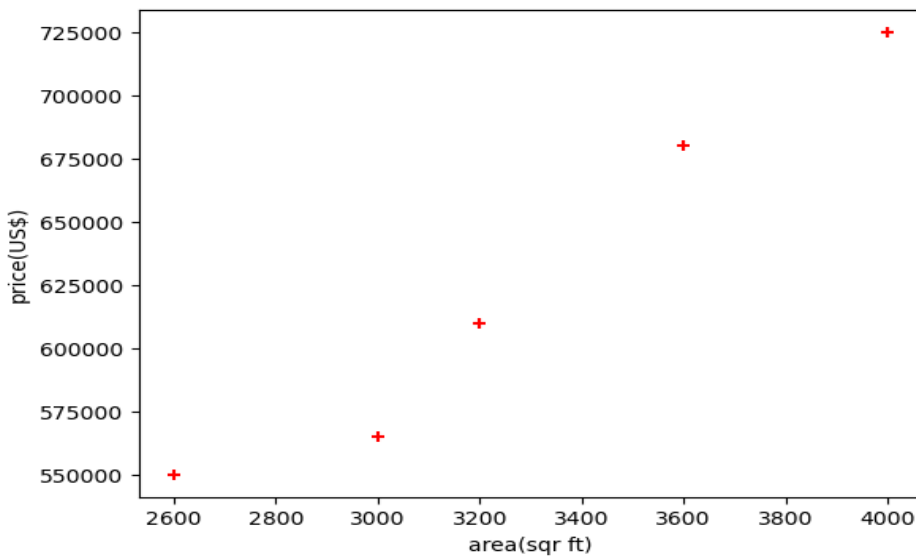
**Output:**

	area	Price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

### Step 3: Draw Scatter Plot for given data

```
%matplotlib inline  
plt.xlabel('area(sqr ft)')  
plt.ylabel('price(US$)')  
plt.scatter(df.area,df.price,color='red',marker='+')
```

**Output:** <matplotlib.collections.PathCollection at 0x8672171890>

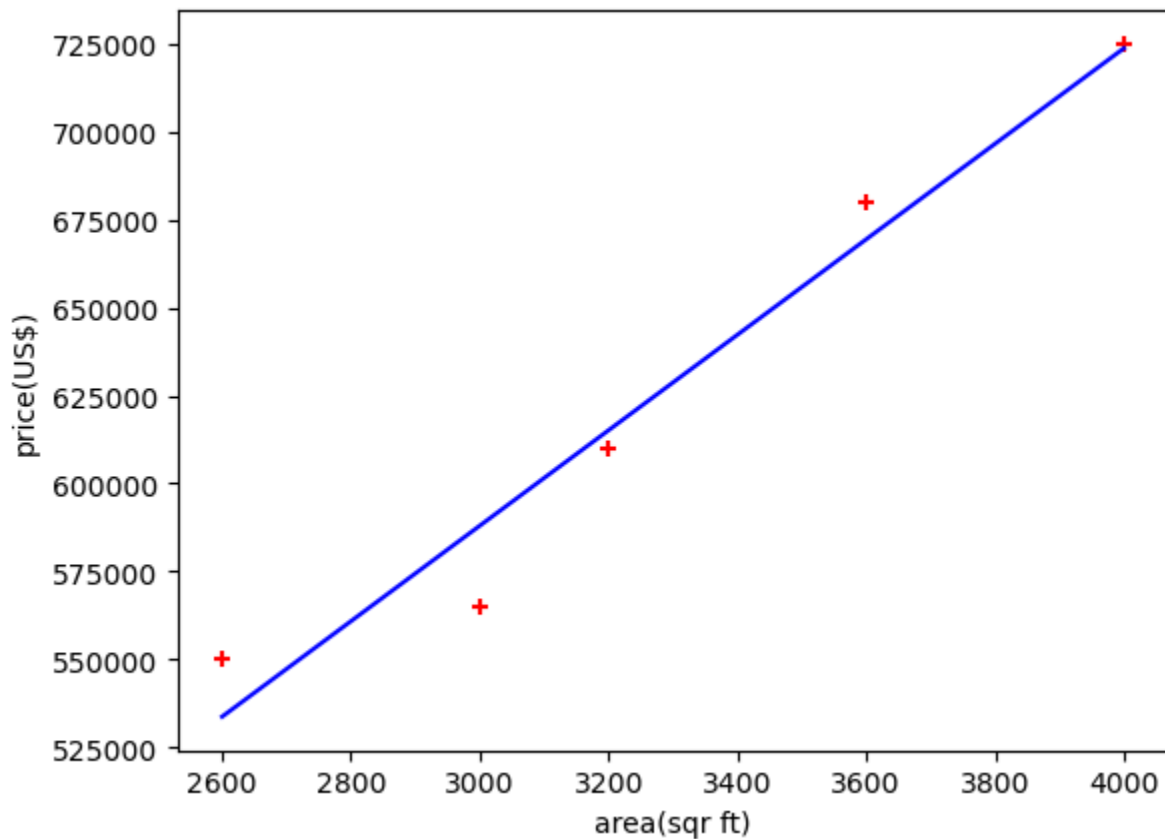


### Step 4: Draw Regression line

```
reg=linear_model.LinearRegression()  
reg.fit(df[['area']],df.price)
```

```
%matplotlib inline
plt.xlabel('area(sqr ft)')
plt.ylabel('price(US$)')
plt.scatter(df.area,df.price,color='red',marker='+')
plt.plot(df.area,reg.predict(df[['area']]),color='blue'
)
```

**Output:** [<matplotlib.lines.Line2D at 0x8672240fd0>]



**Step 5:** Calculate the value of reg.coef and reg.intercept

```
reg.coef_
array([135.78767123])

reg.intercept_
180616.43835616432

reg.predict(np.array(3300).reshape(-1, 1))
array([628715.75342466])
```

A  $y = m \cdot x + b$   
 $y = 135.78767123 \cdot 3300 + 180616.43835616432$

### Step 6: Read only area file

```
d=pd.read_csv("areaonly.csv")
d.head(3)
```

#### Output:

```
   area
0  1000
1  1500
2  2300
```

### Step 7: Predict the value of price by using regression model and store it in CSV File

```
p = reg.predict(d)
d['prices'] = p
d
d.to_csv("prediction.csv",index=False)

d.to_csv("prediction.csv",index=False)
```

### Logistic Regression:

Logistic regression is a supervised machine learning algorithm used for **classification tasks** where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is a statistical algorithm which analyzes the relationship between two data factors.

#### Implementation:

##### Step 1: Import required libraries

```
import pandas as pd

from matplotlib import pyplot as plt

%matplotlib inline
```



### Step 2: Read Dataset

```
df = pd.read_csv("insurance_data.csv")  
  
df.head()
```

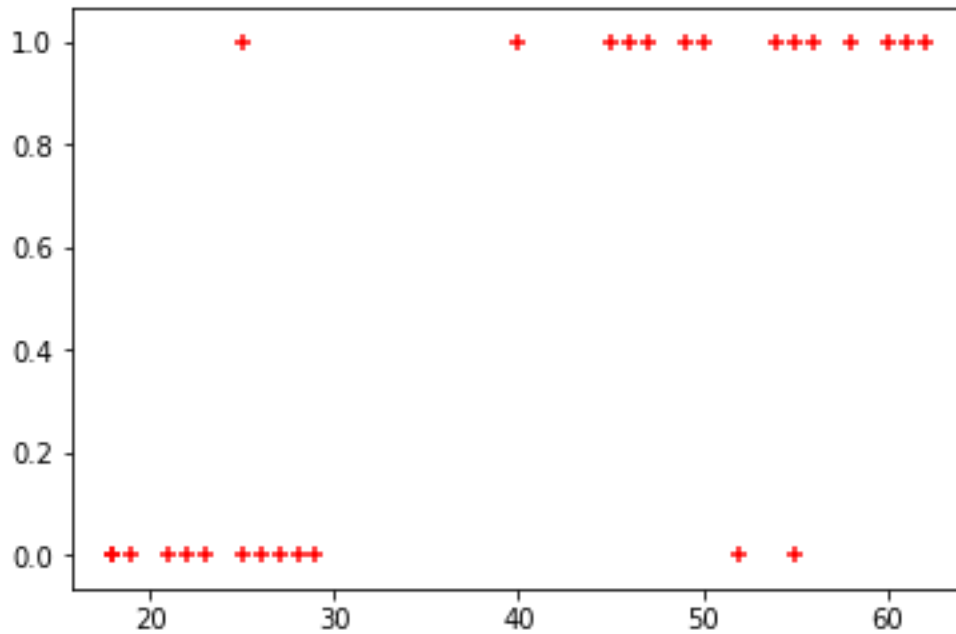
Output:

	age	Bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

### Step 3: Draw Scatter Plot

```
plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

**Output:** <matplotlib.collections.PathCollection at 0x20a8cb15d30>



#### Step 4: Train Data

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(df[['age']],df.bought_insurance,train_size=0.8)

X_test
```

#### Output:

	Age
4	46
8	62
26	23
17	58
24	50
25	54

#### Step 5: logistic regression

```
from sklearn.linear_model import LogisticRegression

AI model = LogisticRegression()

model.fit(X_train, y_train)
```

**Output:**

```
LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn',
n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001,
verbose=0, warm_start=False)
```

```
X_test
```

**Output:**

```
      Age
16    25
21    26
2     47
```

**Step 6:** Predict values for insurance brought or not

```
y_predicted = model.predict(X_test)
model.predict_proba(X_test)
```

**Output:**

```
array([[0.40569485, 0.59430515],
       [0.26002994, 0.73997006],
       [0.63939494, 0.36060506],
       [0.29321765, 0.70678235],
       [0.36637568, 0.63362432],
       [0.32875922, 0.67124078]])
```

```
model.score(X_test,y_test)
```

Output: 1.0

```
y_predicted
```

Output: array([1, 1, 0, 1, 1, 1], dtype=int64)

```
X_test
```

**Output:**

	Age
4	46
8	62
26	23
17	58
24	50
25	54

```
model.coef_  
  
array([[0.04150133]])  
  
model.intercept_  
  
array([-1.52726963])
```

**Define sigmoid function and do the maths with hand**

```
import math  
  
def sigmoid(x):  
    return 1 / (1 + math.exp(-x))  
  
def prediction_function(age):  
    z = 0.042 * age - 1.53 # 0.04150133 ~ 0.042 and -1.52726963 ~ -1.53  
    y = sigmoid(z)  
  
    return y
```

**Predict the insurance brought or not**

```
age = 35  
  
prediction_function(age)  
  
Output: 0.4850044983805899  
0.485 is less than 0.5 which means person with 35 age will not buy insurance  
  
age = 43  
  
prediction_function(age)  
  
Output: 0.568565299077705
```

**Conclusion:** Thus we studied implementation of linear and logistic regression in python.