Experiment No.: 1

Title: Installation of anaconda and implementation of basic python programs using jupyter notebook
Objectives: 1. Install the anaconda software
2. Implement the basic python programs

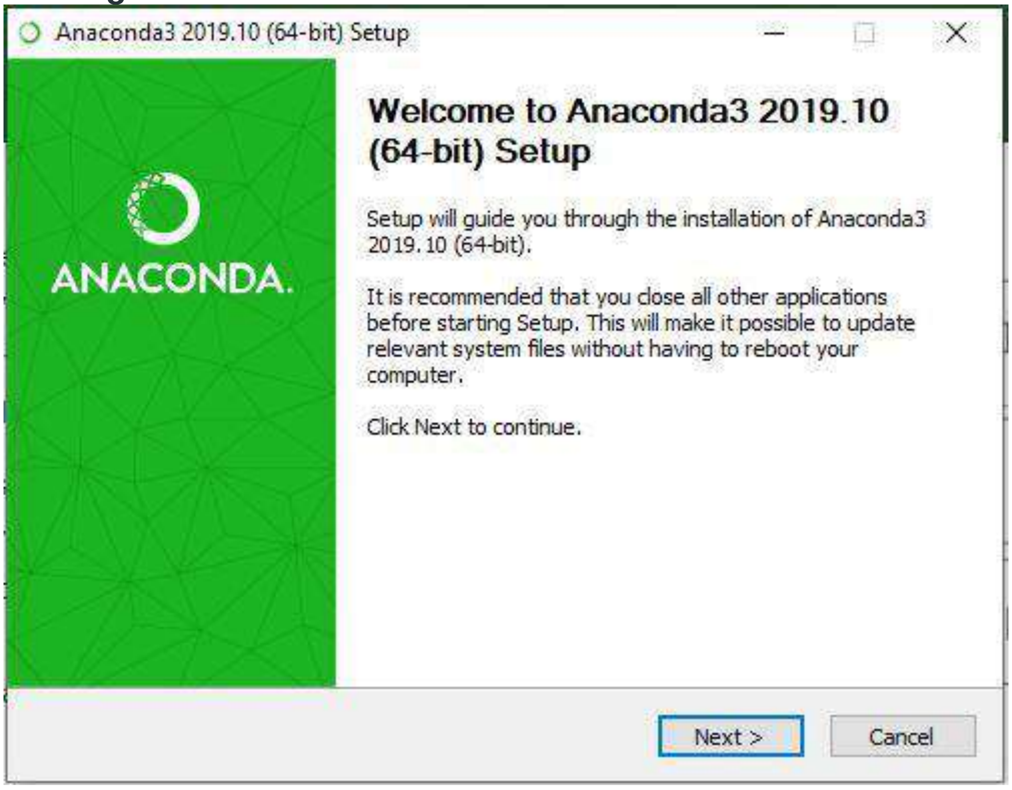
Theory:

Installation of Anaconda:

Head over to anaconda.com and install the latest version of Anaconda

1. Locate your download and double click it. ...

- **Getting Started:**


Anaconda3 2019.10 (64-bit) Setup

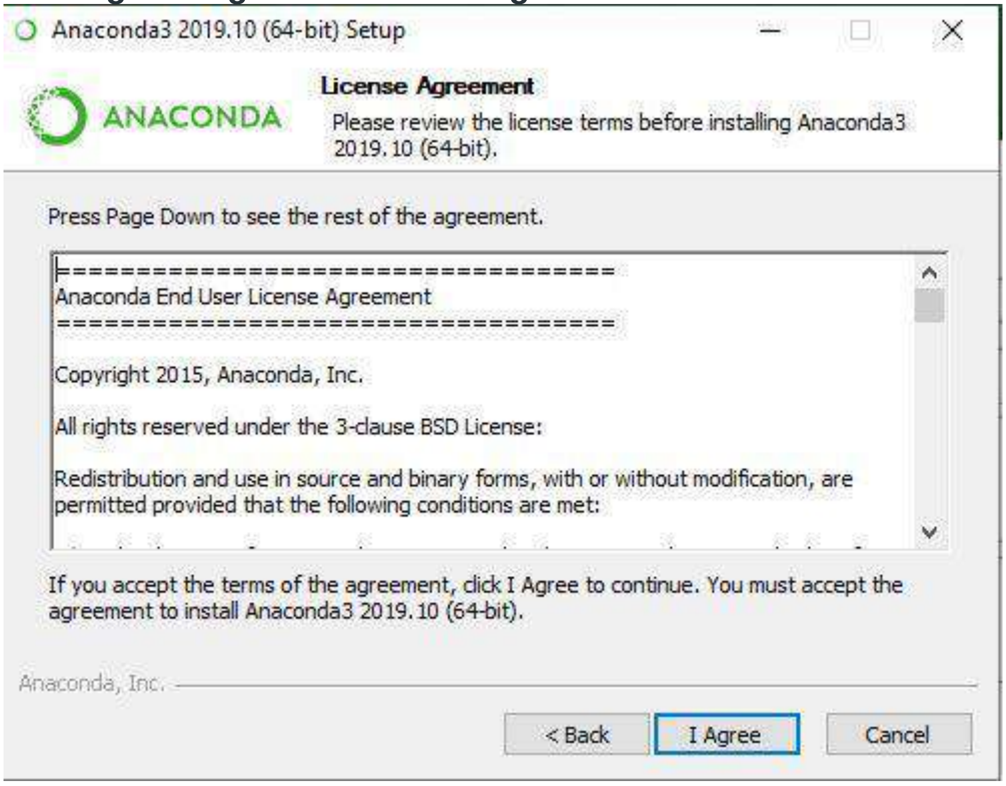
Welcome to Anaconda3 2019.10 (64-bit) Setup

Setup will guide you through the installation of Anaconda3 2019.10 (64-bit).

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

- **Getting through the License Agreement:**



License Agreement

Please review the license terms before installing Anaconda3 2019.10 (64-bit).

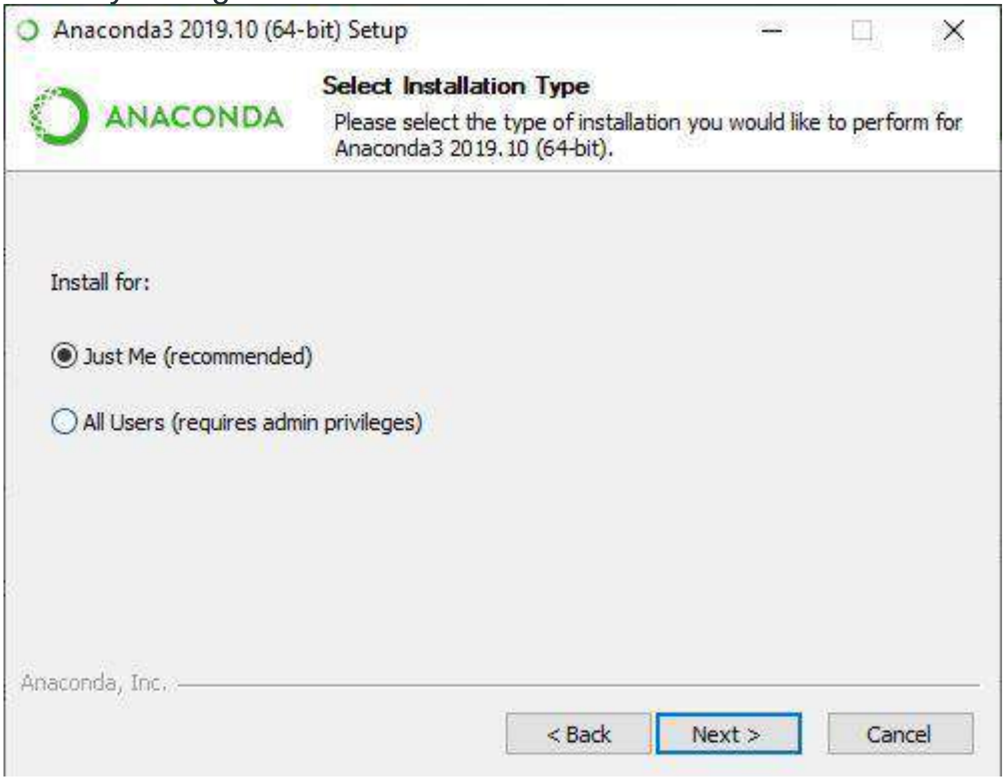
Press Page Down to see the rest of the agreement.

```
===================================
Anaconda End User License Agreement
===================================

Copyright 2015, Anaconda, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without modification, are
permitted provided that the following conditions are met:
```

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2019.10 (64-bit).

Anaconda, Inc.

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



Select Installation Type

Please select the type of installation you would like to perform for Anaconda3 2019.10 (64-bit).

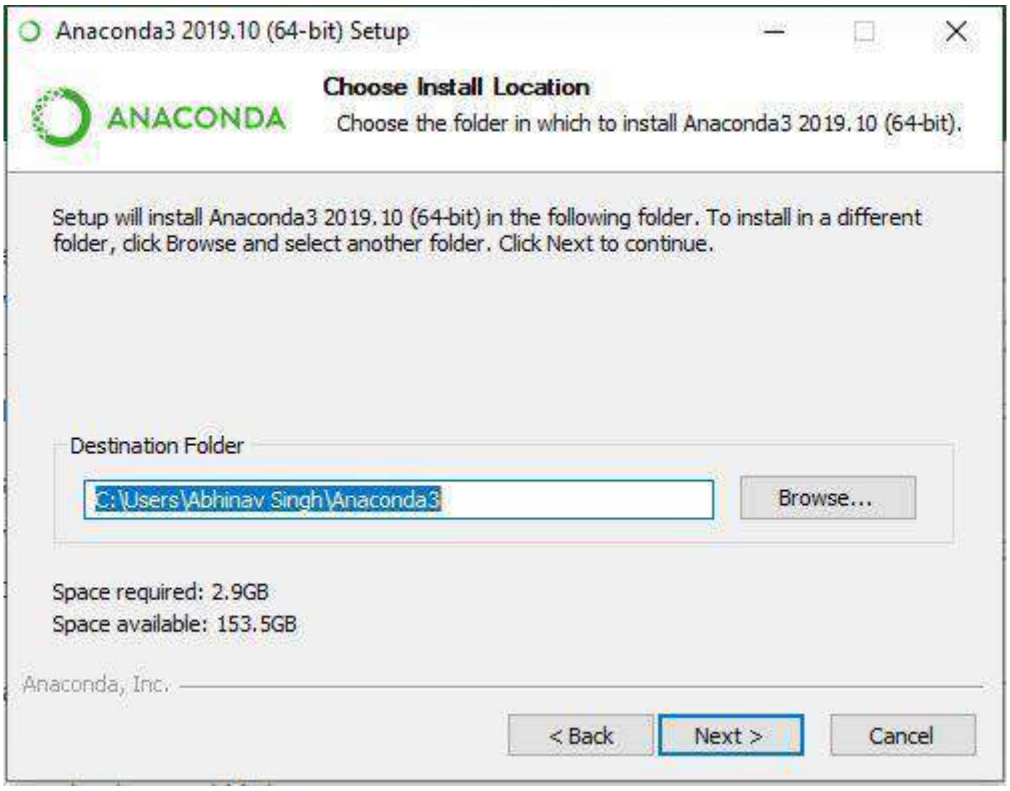
Install for:

[✓ Just Me (recommended)]

All Users (requires admin privileges)

Anaconda, Inc.

- **Choose Installation Location:**



Choose Install Location

Choose the folder in which to install Anaconda3 2019.10 (64-bit).

Setup will install Anaconda3 2019.10 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

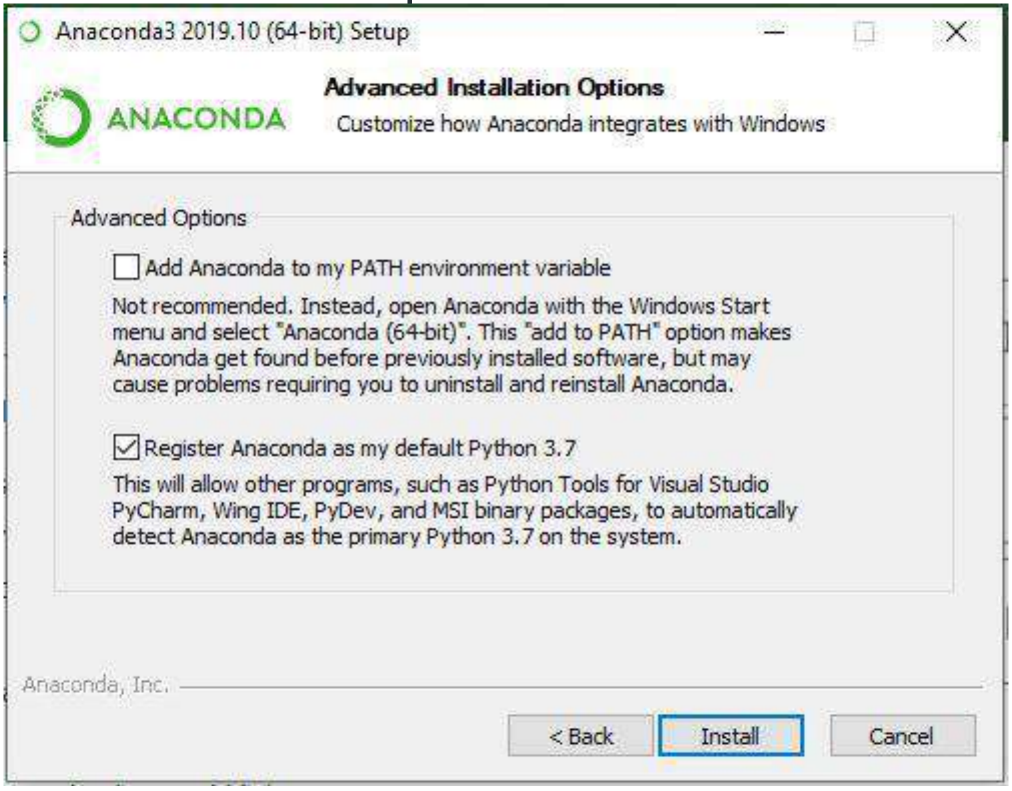
Destination Folder

C:\Users\Abhinav Singh\Anaconda3

Space required: 2.9GB
Space available: 153.5GB

Anaconda, Inc.

- **Advanced Installation Option:**



Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

☐ Add Anaconda to my PATH environment variable

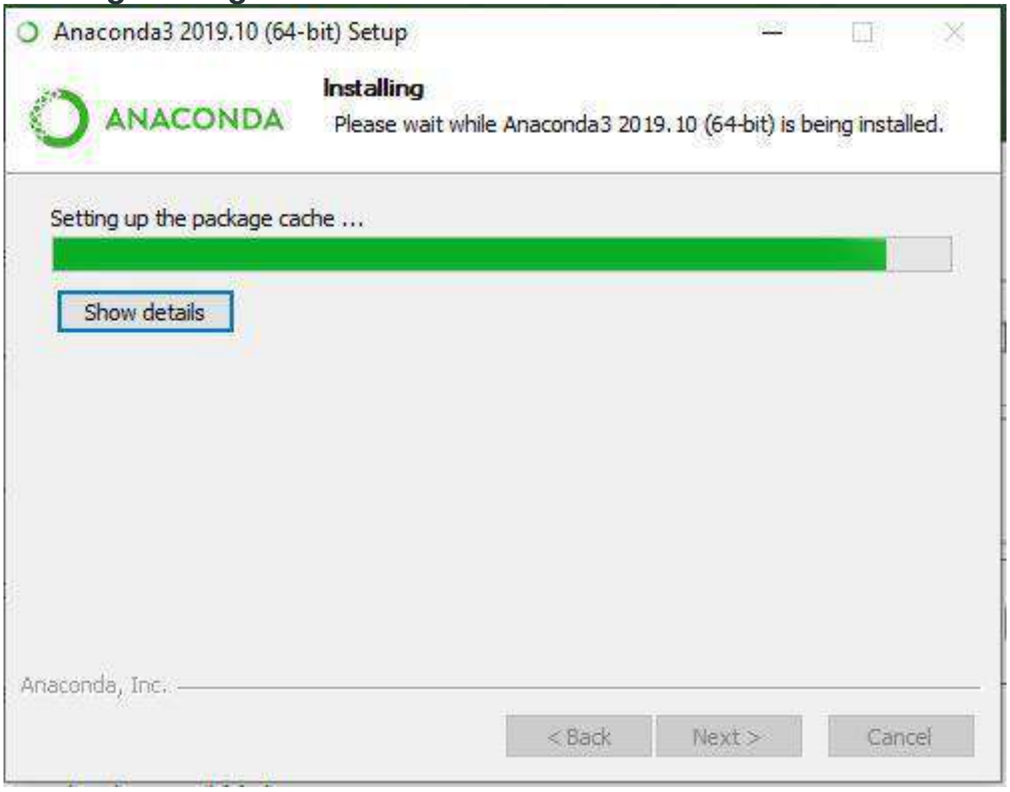
Not recommended. Instead, open Anaconda with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda as my default Python 3.7

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.7 on the system.

Anaconda, Inc.

- **Getting through the Installation Process:**



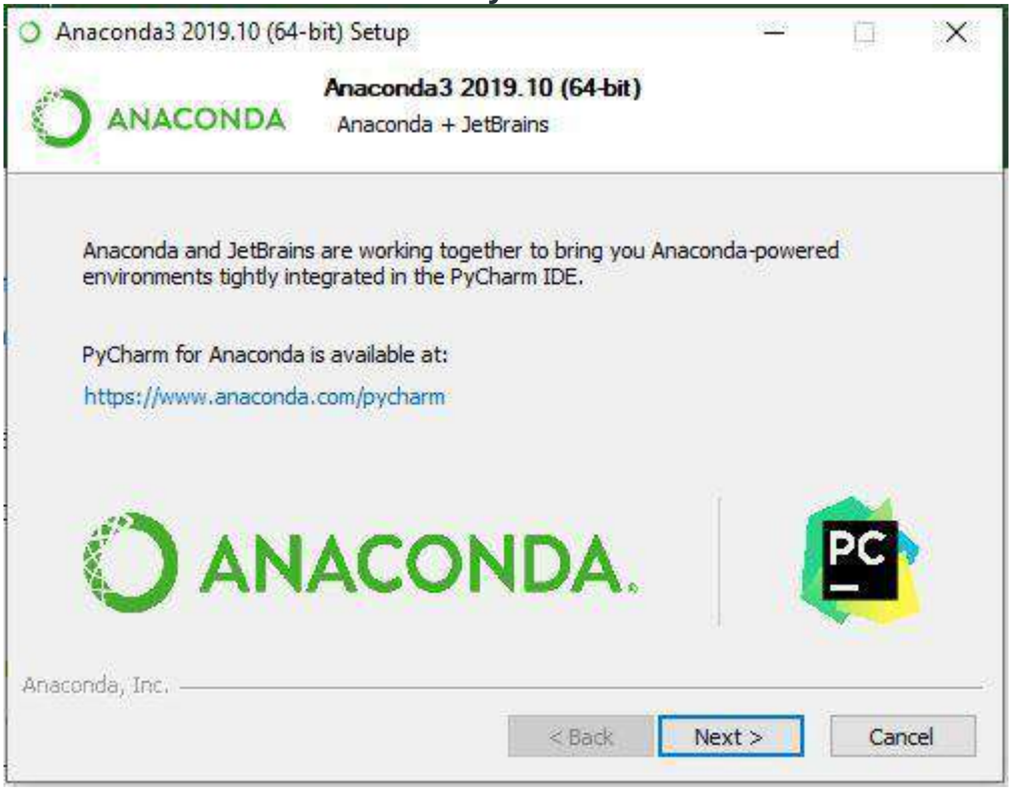
Anaconda3 2019.10 (64-bit) Setup

Installing

Please wait while Anaconda3 2019.10 (64-bit) is being installed.

Setting up the package cache ...

- **Recommendation to Install Pycharm:**



Anaconda3 2019.10 (64-bit) Setup

Anaconda3 2019.10 (64-bit)

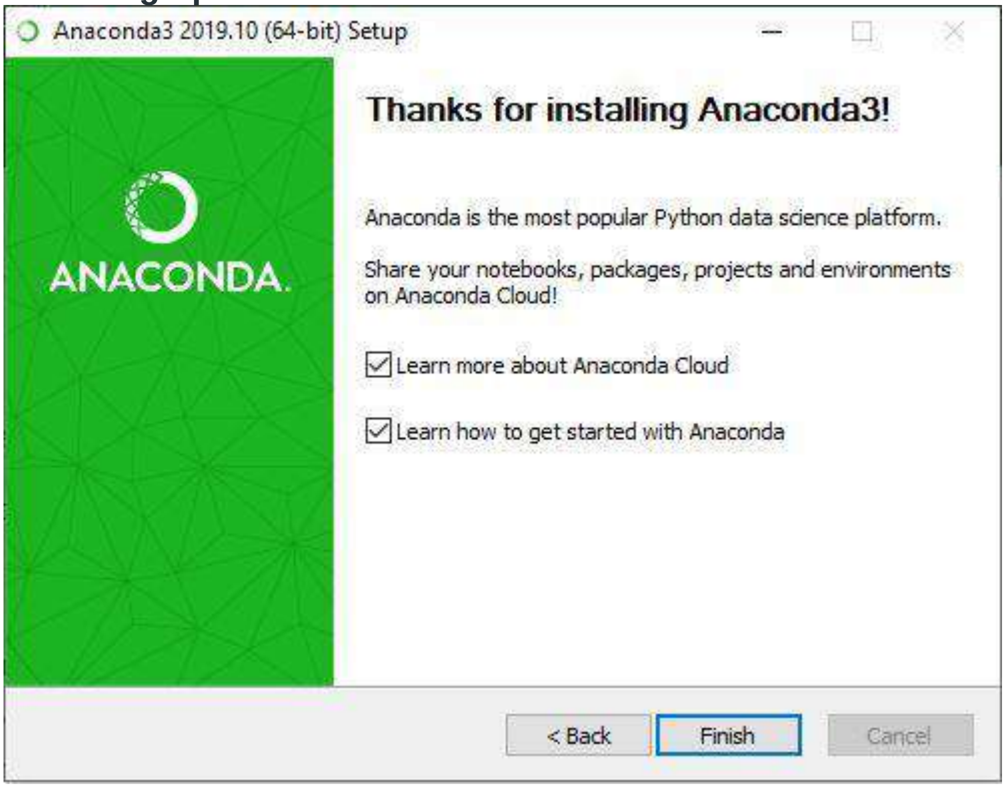
Anaconda + JetBrains

Anaconda and JetBrains are working together to bring you Anaconda-powered environments tightly integrated in the PyCharm IDE.

PyCharm for Anaconda is available at:

https://www.anaconda.com/pycharm

- **Finishing up the Installation:**


Anaconda3 2019.10 (64-bit) Setup

Thanks for installing Anaconda3!

Anaconda is the most popular Python data science platform.

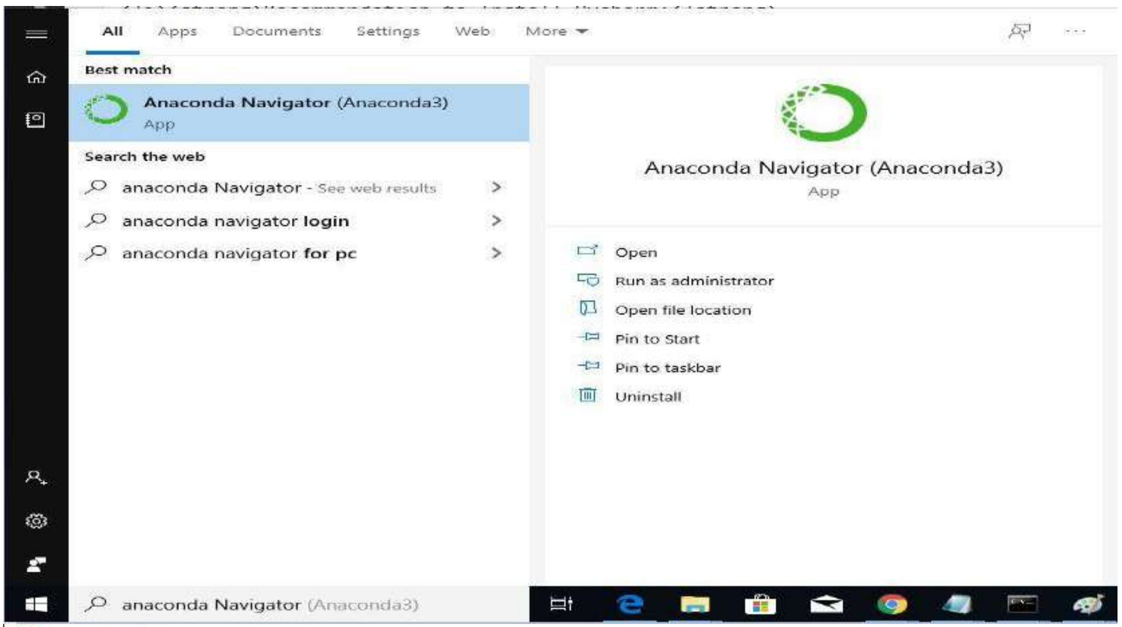
Share your notebooks, packages, projects and environments on Anaconda Cloud!

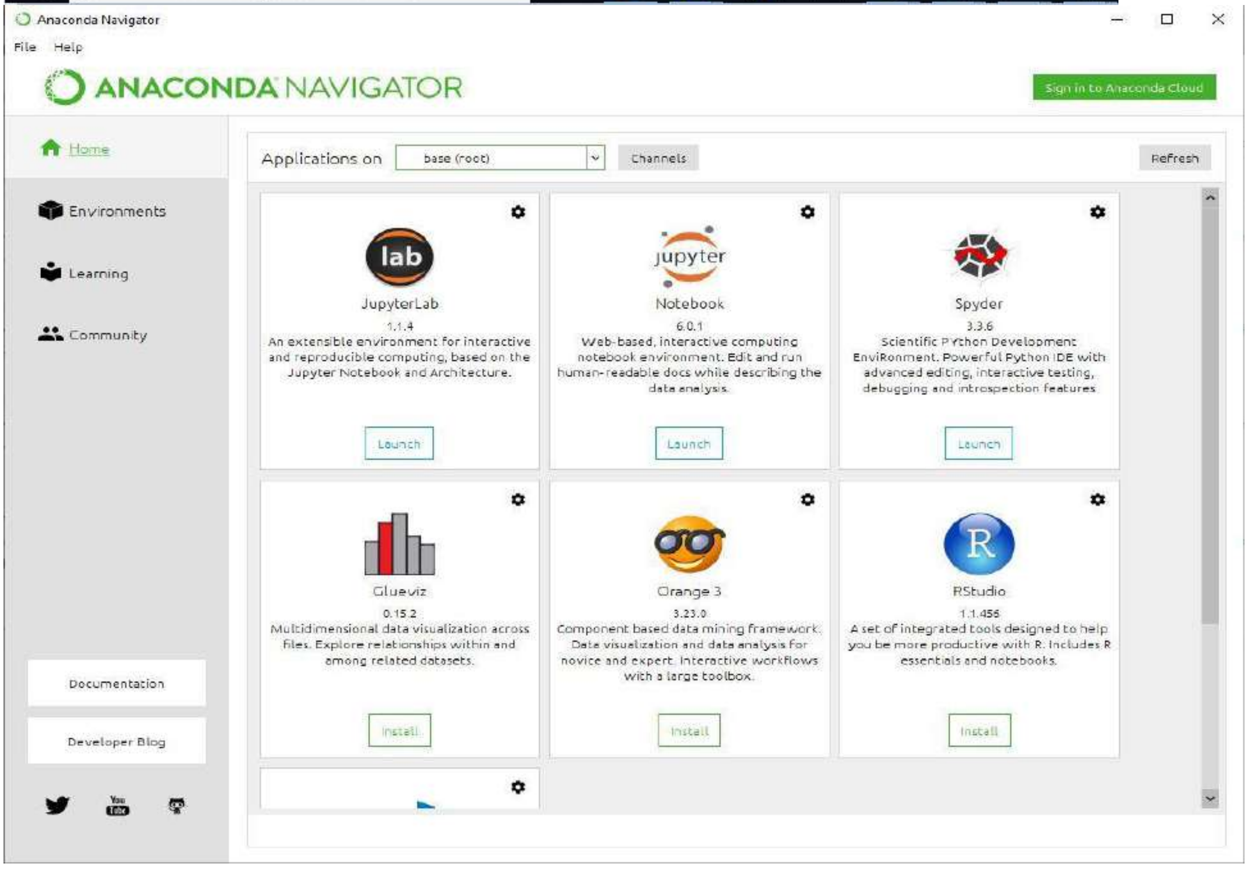
☑ Learn more about Anaconda Cloud

☑ Learn how to get started with Anaconda

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





JupyterLab 1.1.4: An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Notebook 6.0.1: Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Spyder 3.3.6: Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Glueviz 0.15.2: Multidimensional data visualization across files. Explore relationships within and among related datasets.

Orange 3 3.23.0: Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

RStudio 1.1.456: A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

- Do some basic python programming using jupyter notebook
- Write implementation steps

Experiment No.: 2

Title: Implementation of tuples, lists & dictionaries using python.

Objectives: 1. To learn tuples, lists & dictionaries

Theory:

Tuples, lists, and dictionaries:

Lists are what they seem - a list of values. Each one of them is numbered, starting from zero - the first one is numbered zero, the second 1, the third 2, etc. You can remove values from the list, and add new values to the end.

Example: cats' names.

```
cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']
```

Lists are extremely similar to tuples. Lists are modifiable (or 'mutable', as a programmer may say), so their values can be changed. Most of the time we use lists, not tuples, because we want to easily change the values of things if we need to.

Creating lists:	

list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5];
list3 = ["a", "b", "c", "d"];
	# list1, list2,list3 created

Accessing/Updating/Slicing Values in Lists	
print ("list1[0]: ", list1[0]) print ("list2[1:]: ", list2[1:]) list2[2] = 6	# prints **list1[0]: 'physics'** #prints slicedlist2**list2[1:]: [2,3,4,5]** # updates value at index 2 as **6**
Deleting list elements:	
del list[2]	# deletes element at index 2
Basic List Operations:	

len([1, 2, 3])
 	
[1, 2, 3] + [4, 5, 6]
 	
['Hi!'] * 4
 	
3 in [1, 2, 3]
 	
for x in [1,2,3] :
 	
    print (x,end = ' ')
	# Length
	
	# Concatenation
	
	# Repetition
	
	# Membership
	
	# Iteration

Built-in List Functions and Methods:	

Sr.No.	**Function & Description**
1. **cmp(list1, list2)**	No longer available in Python 3.
2. **len(list)**	Gives the total length of the list.
3. **max(list)**	Returns item from the list with max value.
4. **min(list)**	Returns item from the list with min value.
5. **list(seq)**	Converts a tuple into list.
Sr.No.	**Methods & Description**
1. **list.append(obj)**	Appends object obj to list
2. **list.count(obj)**	Returns count of how many times obj occurs in list
3. **list.extend(seq)**	Appends the contents of seq to list
4. **list.index(obj)**	Returns the lowest index in list that obj appears
5. **list.insert(index, obj)**	Inserts object obj into list at offset index
6. **list.pop(obj = list[-1])**	Removes and returns last object or obj from list
7. **list.remove(obj)**	Removes object obj from list
8. **list.reverse()**	Reverses objects of list in place
9. **list.sort([func])**	Sorts objects of list, use compare func if given

Tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The main difference between the tuples and the lists is that the tuples cannot be changed unlike lists. Tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally, you can put these comma-separated values between parentheses also.

Creating Tuples:	

tuple1 = ('physics', 'chemistry', 1997, 2000)
tuple2 = (1, 2, 3, 4, 5)
tuple3 = "a", "b", "c", "d"
tuple4 = ();
tuple5 = (50,)
	# 5 tuples are created
	
	# tuple4 is an **empty** tuple
	#tuple5 : single value **50 ", is must after it"**

Accessing/Slicing Values in Tuples:	
print ("tuple1[0]: ", tuple1[0]) print ("tuple2[1:]: ", tuple2[1:]) tuple2[2] = 6 # Invalid	# prints **tuple1[0]: 'physics'** # prints slicedtuple2 # Invalid since tuples are immutable.
Deleting Tuple elements:	

del tuple1	# deletes tuple1
Basic Tuple Operations:	
len((1, 2, 3)) (1, 2, 3) + (4, 5, 6) ('Hi!',) * 4 3 in (1, 2, 3) for x in (1,2,3) : print (x, end = ' ')	# Length3 # Concatenation(1, 2, 3, 4, 5, 6) # Repetition ('Hi!', 'Hi!', 'Hi!', 'Hi!') # MembershipTrue # Iteration 1 2 3
Built-in Tuple Functions and Methods:	
Sr.No.	**Function & Description**
1. **cmp(tuple1, tuple2)**	Compares elements of both tuples.
2. **len(tuple)**	Gives the total length of the tuple.
3. **max(tuple)**	Returns item from the tuple with max value.
4. **min(tuple)**	Returns item from the tuple with min value.
5. **tuple(seq)**	Converts a list into tuple.

Dictionaries are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - tare similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words, and for each of them a definition. In python, the word is called a 'key', and the definition a 'value'. The values in a dictionary aren't numbered - they aren't in any specific order, either - the key does the same thing. You can add, remove, and modify the values in dictionaries. Example: telephone book.

Creating Dictionaries:	
dict ={'Name':'Zara', 'Age':7,'Name': 'Manni'} print ("dict['Name']: ", dict['Name'])	# Dictionary dict created
Accessing/Updating/Slicing Values in Dictionaries	
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} print ("dict['Name']: ", dict['Name']) dict['Age'] = 8	# prints **dict['Name']: Zara** #updates**dict['Age']: 8**
Deleting Dictionary elements:	

del dict['Name'] dict.clear() del dict	# remove entry with key 'Name' # remove all entries in dict # delete entire dictionary
Keys must be immutable. This means you can use strings, numbers or tuples as dictionary keys but something like **['key']** is not allowed.	example −dict = {['Name']: 'Zara', 'Age': 7} print ("dict['Name']: ", dict['Name']) **Error: list objects are unhashable**
More than one entry per key is not allowed. This means no duplicate key is allowed. When duplicate keys are encountered during assignment, the last assignment wins. For example − dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'} print ("dict['Name']: ", dict['Name'])	# produces dict['Name']: Manni

Built-in List Functions and Methods:

Sr.No.	Function & Description
len(dict)	Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
str(dict)	Produces a printable string representation of a dictionary
type(variable)	Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Sr.No.	Methods & Description
dict.clear()	Removes all elements of dictionary *dict*
dict.copy()	Returns a shallow copy of dictionary *dict*
dict.fromkeys()	Create a new dictionary with keys from seq and values *set* to *value*.
dict.get(key, default=None)	For *key* key, returns value or default if key not in dictionary
dict.has_key(key)	Removed, use the *in* operation instead.
dict.items()	Returns a list of *dict*'s (key, value) tuple pairs
dict.keys()	Returns list of dictionary dict's keys
dict.setdefault(key, default = None)	Similar to get(), but will set dict[key] = default if *key*is not already in dict
dict.update(dict2)	Adds dictionary *dict2*'s key-values pairs to *dict*
dict.values()	Returns list of dictionary *dict*'s values

Experiment No.: 3

Title: Implementation of array processing using NumPy.

Objectives: 1. To learn NumPy arrays.

Theory:

Python is agreat general-purpose programming language on its own, but with the help of a few popularlibraries (NumPy, SciPy, Matplotlib) it becomes a powerful environment for scientfic computing.

NumPy:

Numpy (Numerical Python) is the core library for scientific computing in Python. It provides a high-performancemultidimensional array object, and routines for processing these arrays.NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library for python).

Operations using NumPy:

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

NumPy Arrays:

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

```
import numpy as np
np.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
```

The above constructor takes the following parameters −

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence.
2	**dtype** Desired data type of array, optional

3	**copy** Optional. By default (true), the object is copied
4	**order** C (row major) or F (column major) or A (any) (default)
5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through
6	**Ndmin** Specifies minimum dimensions of resultant array

We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

Examples:

import numpy as np	# import numpy and name it np
a = np.array([1, 2, 3])	# Create a rank 1 array
print(type(a))	# Prints "<class 'numpy.ndarray'>"
print(a.shape)	# Prints "(3,)"
print(a[0], a[1], a[2])	# Prints "1 2 3"
a[0] = 5	# Change an element of the array
print(a)	# Prints "[5, 2, 3]"
b = np.array([[1,2,3],[4,5,6]])	# Create a rank 2 array
print(b.shape)	# Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])	# Prints "1 2 4"
Numpy also provides many functions to create arrays:	
a = np.zeros((2,2))	# Create an array of all zeros
print(a) # [0. 0.]]"	# Prints "[[0. 0.]
b = np.ones((1,2))	# Create an array of all ones
print(b)	# Prints "[[1. 1.]]"
c = np.full((2,2), 7)	# Create a constant array
print(c)	# Prints "[[7. 7.] # [7. 7.]]"

d = np.eye(2)	# Create a 2x2 identity matrix
print(d)	# Prints "[[1. 0.] # [0. 1.]]"
e = np.random.random((2,2))	# Create an array with random values
print(e)	# Might print "[[0.91940167 0.08143941]
	# [0.68744134 0.87236687]]"
Creating an array from sub-classes:	
np.array(np.mat('1 2; 3 4')) [3, 4]])	# Creates array([[1, 2],
np.array(np.mat('1 2; 3 4'), subok=**True**)	# Creates matrix([[1, 2], [3, 4]])

Array indexing:

Numpy offers several ways to index into arrays:fields access, Slicing and advanced indexing.

Slicing: Slicing is the way to choose a range of values in the array. We use a colon (:) in square brackets.

Syntax: **[Start : Stop : Step]**

# slice items between indexes	
a = np.arange(10) print a[2:5]	# prints [2 3 4]
# slice items starting from index	
a = np.array([[1,2,3],[3,4,5],[4,5,6]]) print a[1:] #[4 5 6]]	# prints [[3 4 5]
Slicing can also include ellipsis (…) to make a selection tuple of the same length as the dimension of an array. If ellipsis is used at the row position, it will return an ndarray comprising of items in rows.	
a = np.array([[1,2,3],[3,4,5],[4,5,6]]) print a[...,1]	# this returns array of items in the second column
print a[1,...]	# this returns array of all items from the second row
print a[...,1:]	# this returns array of all items from column 1 onwards
IntegerIndexing: selecting any arbitrary item in an array based on its Ndimensional index	

x = np.array([[1, 2], [3, 4], [5, 6]])
y = x[[0,1,2], [0,1,0]]
print y
	# includes elements at (0,0), (1,1) and (2,0) from the first array
	# [1 4 5]

Boolean Indexing: x = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8],[9, 10, 11]])	
print x[x > 5]	# prints [6 7 8 9 10 11]

Experiment No.: 4

Title: Implementation of exploratory data analysis using Pandas dataframes.

Objectives: 1. To learn pandas.

Theory:

Pandasis an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

Pandas can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data — load, prepare, manipulate, model, and analyze.

Key Features:

Following are the important features of pandas −

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

Pandas Data Structures:

Pandas deals with the following three data structures that are built on top of Numpy array& due to this reason they are fast.

Data Structure	Dimensions	Description
Series	1	1-D labeled homogeneous array, sizeimmutable.
Data Frames	2	General 2-D labeled, size-mutable tabular structure with potentially heterogeneously typed columns.
Panel	3	General 3-D labeled, size-mutable array.

Best way tothink of these data structures is that the higher dimensional data structure is a container of its lower dimensional data structure. For example, DataFrame is a container of Series, Panel is a container of DataFrame.

Using Pandas data structures, the mental effort of the user is reduced.For example, with tabular data (DataFrame) it is more semantically helpful to think of the index (the rows) and the columns rather than axis 0 and axis 1.

Mutability

All Pandas data structures are value mutable (can be changed) and except Series all are size mutable. Series is size immutable.

Series:

Seriesis a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.

The basic method to create a Series is to call:

s = pd.Series(data, index=index)

Here, **data** can be many different things:

- a Python dictionary
- an ndarray
- a scalar value (e.g 5)
- Series acts very similarly to andarray, and is a valid argument to most NumPy functions. However, operations such as slicing will also slice the index.
- A Series is like a fixed-size dict in that you can get and set values by index label.(e.g. s[‘a’])
- Series can also have a name attribute:

```
s = pd.Series(np.random.randn(5), name='something')
```

The passed index is a list of axis labels. Thus, this separates into a few cases depending on what data is.

1. When the data is a dict, and an index is not passed, the Series index will be ordered by the dict’s insertion order, if you’re using Python version >= 3.6 and Pandas version >= 0.23.

```
d = {'b': 1, 'a': 0, 'c': 2}

In [1]: pd.Series(d)
Out[1]:
        b    1
        a    0
        c    2
        dtype: int64
```

2. If **data** is an **ndarray**, index must be the same length as data. If no index is passed, one will be created having values [0, ...,len(data) - 1].

```
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])

In [4]: s
Out[4]:
a    0.469112
b   -0.282863
c   -1.509059
d   -1.135632
e    1.212112dtype: float64
```

3. If **data** is a **scalar** value, an index must be provided. The value will be repeated to match the length of index.

```
In [12]: pd.Series(5., index=['a', 'b', 'c', 'd', 'e'])
Out[12]:
a    5.0
b    5.0
c    5.0
d    5.0
e    5.0
dtype: float64
```

DataFrames

A DataFrame is a two-dimensional array with heterogeneous data.That is, DataFrame accepts many different kinds of input:

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments. If you pass an index and / or columns, you are guaranteeing the index and / or columns of the resulting DataFrame. Thus, a dict of Series plus a specific index will discard all data not matching up to the passed index.

Data is aligned in a tabular fashion in rows and columns. For example,

Name	Age	Gender	Rating
Steve	32	Male	3.45
Lia	28	Female	4.6
Vin	45	Male	3.9
Katie	38	Female	2.78

The table represents the data of a sales team of an organization with their overall performance rating. The data is represented in rows and columns. Each column represents an attribute and each row represents a person.

Key Points

• Homogeneous data
• Size- Mutable
• Data Mutable
• Potentially columns are of different types
• Labeled axes (rows and columns)
• Can Perform Arithmetic operations on rows and columns

The parameters of the constructor are as follows −

Sr.No	Parameter & Description
1	**data** data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**index** For the row labels, the Index to be used for the resulting frame is Optional Default np.arrange(n) if no index is passed.
3	**columns** For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed.
4	**dtype** Data type of each column.
5	**copy** This command (or whatever it is) is used for copying of data, if the default is False.

DataFrameFrom dict of Series or dicts

```
d = {'one': pd.Series([1., 2., 3.], index=['a', 'b', 'c']),  'two': pd.Series([1., 2., 3., 4.],
        index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
```

```
df
Out[1]:
        one  two
        a  1.0  1.0
        b  2.0  2.0
        c  3.0  3.0
        d  NaN  4.0
```

DataFramefrom dict of ndarrays / lists

```
d = {'one': [1., 2., 3., 4.],  'two': [4., 3., 2., 1.]}
pd.DataFrame(d)
Out[1]:
        one  two
        0  1.0  4.0
        1  2.0  3.0
        2  3.0  2.0
        3  4.0  1.0
```

DataFrame fromstructured or record array

```
data = np.zeros((2, ), dtype=[('A', 'i4'), ('B', 'f4'), ('C', 'a10')])

data[:] = [(1, 2., 'Hello'), (2, 3., "World")]

pd.DataFrame(data)
Out[1]:
          A    B        C
        0  1  2.0  b'Hello'
        1    2  3.0  b'World'
```

DataFrame from a Series

The result will be a DataFrame with the same index as the input Series, and with one column whose name is the original name of the Series (only if no other column name provided).

Indexing/Selection:

The basics of indexing are as follows:

Operation	Syntax	Result
Select column	df[col]	Series
Select row by label	df.loc[label]	Series
Select row by integer location	df.iloc[loc]	Series
Slice rows	df[5:10]	DataFrame
Select rows by boolean vector	df[bool_vec]	DataFrame

Experiment No.: 05

Title: Demonstration of plotting the data using python.

Objectives:

1. To learn about plotting the data.
2. To use matplotlib for data visualization/plotting.

Theory:

Data visualization is a very important part of data analysis. We can use it to explore our data. If we understand our data well, we'll have a better chance to find some insights. Finally, when we find any insights, we can use visualizations again to be able to share our findings with other people.

Basic Visualization Rules

Before we look at some kinds of plots, we'll introduce some basic rules. Those rules help us make nice and informative plots instead of confusing ones.

- The first step is to choose the appropriate plot type. If there are various options, we can try to compare them, and choose the one that fits our model the best.
- Second, when we choose your type of plot, one of the most important things is to label your axis. If we don't do this, the plot is not informative enough. When there are no axis labels, we can try to look at the code to see what data is used and if we're lucky we'll understand the plot. But what if we have just the plot as an image? What if we show this plot to your boss who doesn't know how to make plots in Python?
- Third, we can add a title to make our plot more informative.
- Fourth, add labels for different categories when needed.
- Five, optionally we can add a text or an arrow at interesting data points.
- Six, in some cases we can use some sizes and colors of the data to make the plot more informative.

Types of Visualizations and Examples with Matplotlib

There are many types of visualizations. Some of the most famous are: **line plot, scatter plot, histogram, box plot, bar chart, and pie chart**. But among so many options how do we choose the right visualization? First, we need to make some exploratory data analysis. After we know the shape of the data, the data types, and other useful statistical information, it will be easier to pick the right visualization type.

There are many visualization packages in Python. One of the most famous is Matplotlib. It can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, and web application servers.
Some basic functions of the **matplotlib.pyplot** subpackage are included in the examples below. Assume that the **matplotlib.pyplot** subpackage is imported with an alias **plt**.

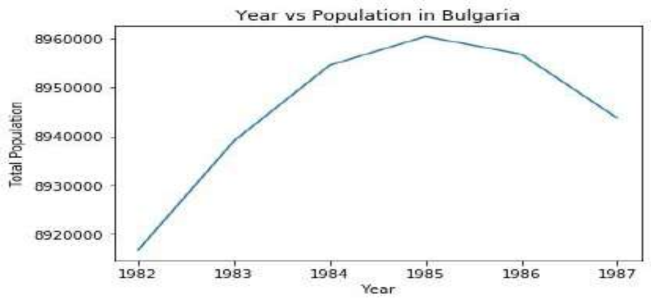
plt.title("My Title") will add a title "My Title" to your plot
plt.xlabel("Year") will add a label "Year" to your x-axis

plt.ylabel(“Population”) will add a label “Population” to your y-axis
plt.xticks([1, 2, 3, 4, 5]) set the numbers on the x-axis to be 1, 2, 3, 4, 5. We can also pass and labels as a second argument. For, example, if we use this code plt.xticks([1, 2, 3, 4, 5], ["1M", "2M", "3M", "4M", "5M"]), it will set the labels 1M, 2M, 3M, 4M, 5M on the x-axis.
plt.yticks() - works the same as plt.xticks(), but for the y-axis.

Line Plot:

It is a type of plot which displays information as a series of data points called “markers” connected by straight lines. In this type of plot, we need the measurement points to be ordered (typically by their x-axis values). This type of plot is often used to visualize a trend in data over intervals of time - a time series.
To make a line plot with Matplotlib, we call plt.plot(). The first argument is used for the data on the horizontal axis, and the second is used for the data on the vertical axis. This function generates your plot, but it doesn’t display it. To display the plot, we need to call the plt.show() function. This is nice because we might want to add some additional customizations to our plot before we display it. For example, we might want to add labels to the axis and title for the plot.

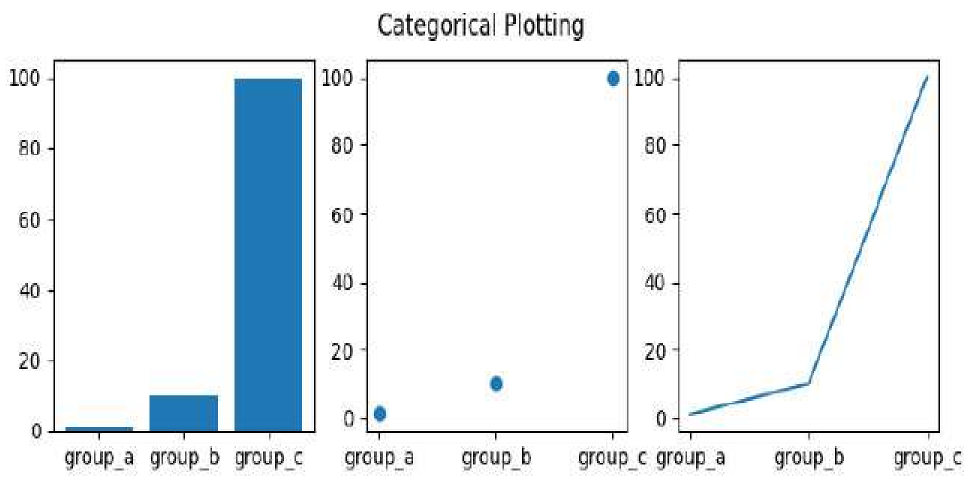
```
import matplotlib.pyplot as plt
years = [1983, 1984, 1985, 1986, 1987]
total_populations = [8939007, 8954518, 8960387, 8956741, 8943721]
plt.plot(years, total_populations)
plt.title("Year vs Population in Bulgaria")
plt.xlabel("Year")
plt.ylabel("Total Population")
plt.show()
```



Subplot

Many plot types can be combined in one figure to create powerful and flexible representations of data.For E.g.

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]
plt.figure(figsize=(9, 3))
plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



The **subplot()** command specifies numrows, numcols, plot_numberwhere plot_number ranges from **1** to **numrows*numcols**. The commas in the subplotcommand are optional if **numrows*numcols<10**. So subplot(211) is identical tosubplot(2, 1, 1).

You can create an arbitrary number of subplots and axes. If you want to place an axes manually, i.e., not on a rectangular grid, use the axes() command, which allows youto specify the location as axes([left, bottom, width, height]) where all valuesare in fractional (0 to 1) coordinates.

You can clear the current figure with **clf()** and the current axes with **cla()**.If you are making lots of figures, you need to be aware of one more thing: the memoryrequired for a figure is not completely released until the figure is explicitly closed with**close()**. Deleting all references to the figure, and/or using the window manager to killthe window in which the figure appears on the screen, is not enough, because pyplotmaintains internal references until close() is called.

Experiment No.: 6

Title: Write a program for implementation of simple linear regression.

Objectives: To learn simple linear regression

Theory:

Regression is a method of modeling a target value based on independent predictors. This method is mostly used for forecasting and finding out cause and effect relationship between variables. There are two main types:**Simple Regression**and **Multivariable Regression**

- Simple linear regression uses traditional slope-intercept form where m and c are the variables our algorithm will try to "learn" to produce the most accurate predictions. x represents our input data and y represents our prediction. y=mx+c.
- Multivariable regression have more complex form

Hypothesis Function

Hypothesis function has the general form:

$$\hat{y} = \mathbf{h}\theta\,(\mathbf{x}) = \theta_0 + \theta_1 x$$

For historical reasons, this function h is called a hypothesis. The job of the hypothesis is a function that takes as input for e.g. the size of a house like maybe the size of the new house your friend's trying to sell so it takes in the value of x and it tries to output the estimated value of y for the corresponding house. So h is a function that maps from x's to y's. When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a **regression problem**.

Note that this is like the equation of a straight line. We give to $\mathbf{h}_\theta(\mathbf{x})$ values for θ_0 and θ_1 to get our estimated output $\hat{y}$. In other words, we are trying to create a function called $\mathbf{h}_\theta$ that is trying to map our input data (the x's) to our output data (the y's).

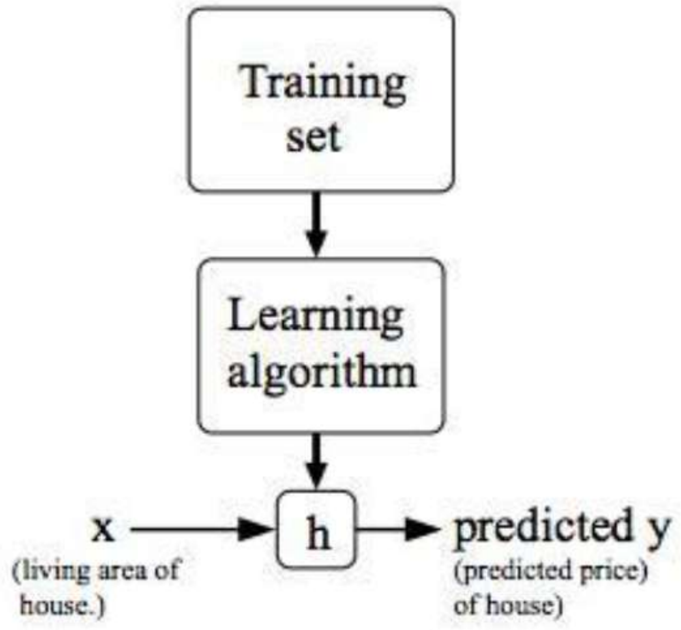
Example:

Suppose we have the following set of training data:

input x	output y
0	4
1	7
2	7
3	8

Now we can make a random guess about our h_θ function: θ_0=2 and θ_1 =2. The hypothesis function becomes $h_\theta(x) = 2 + 2x$.

So for input of 1 to our hypothesis, **y** will be 4. This is off by 3. Note that we will be trying out various values of $\theta_0$and $\theta_1$to try to find values which provide the best possible "fit" or the most representative "straight line" through the data points mapped on the x-y plane.



Cost Function:

We can measure the accuracy of our hypothesis function by using a cost function. This takes an average (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's compared to the actual output y's.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

To break it apart, it is **(½)**$\bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x_i)-y_i$, or the difference between the predicted value and the actual value.
This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved **(½m)** as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the ½ term.
Now we are able to concretely measure the accuracy of our predictor function against the correct results we have so that we can predict new results we don't have.

If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make straight line *(*defined by $h_\theta(x)$*)* which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all the points of our training data set. In such a case the value of **J(*θ₀,θ₁*)** will be **0**.

Gradient Descentfor Linear Regression

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to:

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((h_\theta(x_i) - y_i) x_i)$$

}

where ***α***, which is called the learning rate , **m** is the size of the training set, $\theta_0$is a constant that will be changing simultaneously with θ_1 and x_i , $y_i$are values of the given training set (data).

Experiment No.: 7

Title: Write a program to implement Linear Regression with Multiple Variables.

Objectives: To learn Linear Regression with Multiple Variables

Theory:

Linear regression with multiple variables is also known as "multivariate linear regression".

We now introduce notation for equations where we can have any number of input variables

$$\begin{aligned} x_j^{(i)} &= \text{value of feature } j \text{ in the } i^{th} \text{ training example} \\ x^{(i)} &= \text{the column vector of all the feature inputs of the } i^{th} \text{ training example} \\ m &= \text{the number of training examples} \\ n &= \left|x^{(i)}\right|; \text{(the number of features)} \end{aligned}$$

Now define the multivariable form of the hypothesis function as follows, accommodating these multiple features:

Hypothesis Function

- Hypothesis Function for multiple linear regression is,

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

Using the definition of matrix multiplication, our **multivariable hypothesis function** can be concisely represented as:

$$h_\theta(x) = \theta^T . x$$

This is a **vectorization** of our hypothesis function for one training example;

Remark: Note that for convenience reasons, we assume $x_0^{(i)} = 1$ **for (** $i \in 1,\ldots,m$**).** This allows us to do matrix operations with **theta** and **x**. Hence making the two vectors '**θ**' and '$x^{(i)}$' match each other element-wise (that is, have the same number of elements: n+1).]

Cost Function

Cost Function for multiple linear regression:

For the parameter vector θ (of type $\mathbb{R}^{n+1}$ or in $\mathbb{R}^{(n+1)\times 1}$, the cost function is:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

The vectorized version is:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T(X\theta - \vec{y})$$

Where $\vec{y}$ denotes the vector of all y values.

Gradient Descent for Multiple Variables

The gradient descent equation itself is generally the same form; we just have to repeat it forour 'n' features:

```
repeat until convergence: {
    $\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}$
    $\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)}$
    $\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)}$
    ...
}
```

In other words:

```
repeat until convergence: {
    $\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$    for j := 0..n
}
```

Gradient Descent in Practice I - Feature Scaling

We can speed up gradient descent by having each of our input values in roughly the same range. This is because **θ** will descend **quickly** on **small ranges** and **slowly on large ranges**, and so will **oscillate inefficientlydown to the optimum** when the variables are **very uneven**. The **way to prevent this** is to **modify the ranges** of our input variables so that they are all roughly the same.
Ideally:

$$-1 \le x_{(i)} \le 1$$

or

$$-0.5 \le x_{(i)} \le 0.5$$

These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.

Two techniques to help with this are **feature scaling** and **mean normalization**.

1. **Feature scaling** involves dividing the input values by the **range** (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just **1**.
2. **Mean normalization** involves **subtracting** the **average** value for an **input** variable **from** the values for that input variable resulting in a **new average value** for the input variable of just **zero**.

To implement both of these techniques, adjust your input values as shown in this formula:

$$x_i := \frac{x_i - \mu_i}{S_i}$$

Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.
Note that dividing by the range, or dividing by the standard deviation, give different results. The quizzes in this course use range - the programming exercises use standard deviation.

For example, if x_i represents housing prices with a range of 1000 to 2000 and a mean value of 1000, then, $x_i := \frac{price - 1000}{1900}$

Experiment No.: 08

Title: Write a program for implementation of logistic regression.

Objectives: To learn logistic regression

Theory:

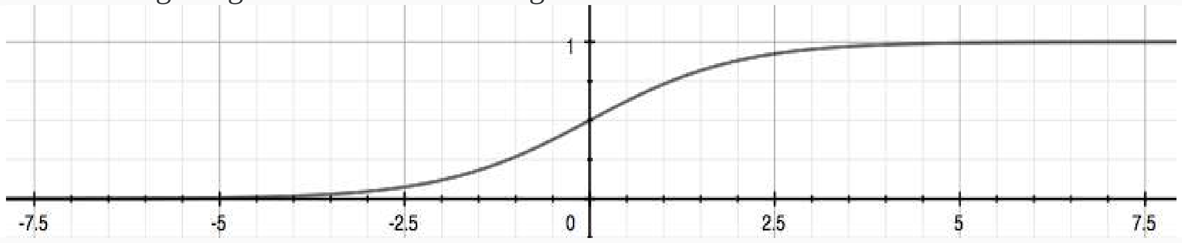
Logistic regression is a method for classifying data into discrete outcomes. For example, we might use logistic regression to classify an email as spam or not spam. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a logit, from logistic unit, hence the alternative names. Analogous models with a different sigmoid function instead of the logistic function can also be used, such as the probit model; the defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at a constant rate, with each independent variable having its own parameter; for a binary dependent variable this generalizes the odds ratio.

Hypothesis Representation

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0, 1\}$. To fix this, let's change the form for our hypotheses $h_\theta(x)$ to satisfy $\mathbf{0 \le h_\theta(x) \le 1}$. This is accomplished by plugging $\boldsymbol{\theta^T.x}$ into the Logistic Function.
Our new form uses the "Sigmoid Function," also called the "Logistic Function":

$$h_\theta(x) = g(\theta^T.x)$$
$$z = \theta^T.x$$
$$g(z) = \frac{1}{1+e^{-z}}$$

The following image shows us what the sigmoid function looks like:



The function g (z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

$h_\theta(x)$ will give us the **probability** that our output is 1. For example $h_\theta(x)$ = 0.7 gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

$$h_\theta(x) = P(y=1 \mid x;\theta) = 1 - P(y=0 \mid x;\theta)$$

$$P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1$$

Hypothesis Function

- Hypothesis Function for Logistic regression is,
- $h_\theta(x) = g((\theta^T x))$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

i.e.

Cost Function

- Cost Function for Logistic regression is

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))]$$

Gradient Descent

- Gradient Descent Function for logistic regression is

```
Repeat {
    $\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
                                  (simultaneously update all $\theta_j$)
}
```

Experiment No.: 09

Title: Implementation of Decision Tree.

Objectives: To learn Decision Tree.

Theory:

A decision tree is a simple representation for classifying examples. Assume that all of the input features have finite discrete domains, and there is a single target feature called the "classification". Each element of the domain of the classification is called a *class*. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes, signifying that the data set has been classified by the tree into either a specific class, or into a particular probability distribution (which, if the decision tree is well-constructed, is skewed towards certain subsets of classes).

A tree is built by splitting the source set, constituting the root node of the tree, into subsets - which constitute the successor children. The splitting is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions.

Assumptions we make while using Decision tree:

- At the beginning, we consider the whole training set as the root.
- Attributes are assumed to be categorical for information gain and for gini index, attributes are assumed to be continuous.
- On the basis of attribute values records are distributed recursively.
- We use statistical methods for ordering attributes as root or internal node.

Pseudocode:

```
1. Find the best attribute and place it on the root node of the tree.
```

2. Now, split the training set of the dataset into subsets. While making the subset make sure that each subset of training dataset should have the same value for an attribute.
3. Find leaf nodes in all branches by repeating 1 and 2 on each subset.

Implementation procedure:

1. Building Phase
    - Preprocess the dataset.
    - Split the dataset from train and test.
    - Train the classifier.
2. Operational Phase
    - Make predictions.
    - Calculate the accuracy.

Experiment No.: 10

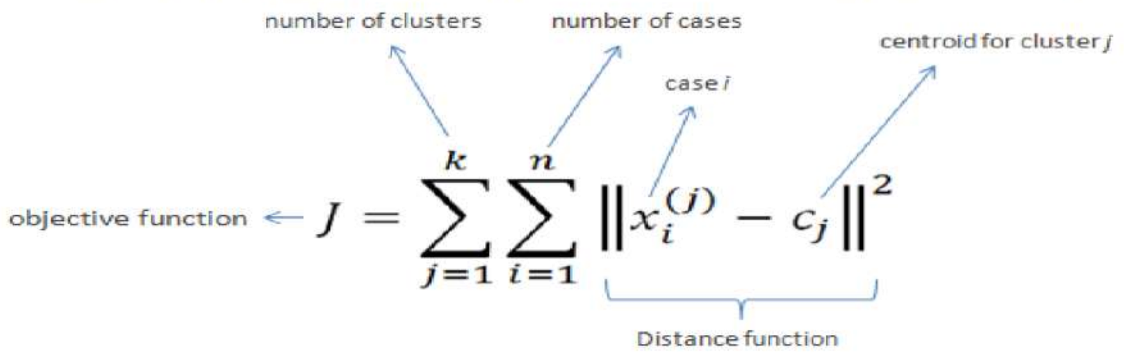
Title: Implementation of clustering using K-means.

Objectives: To learn K-means algorithm.

Theory:

K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. K-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster.

- K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups).
- The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided.
- K-Means clustering intends to partition *n* objects into *k* clusters in which each object belongs to the cluster with the nearest mean.
- This method produces exactly *k* different clusters of greatest possible distinction.
- The best number of clusters *k* leading to the greatest separation (distance) is not known as a priori and must be computed from the data.
- The objective of K-Means clustering is to minimize the squared error function
- K-Means is relatively an efficient method.
- However, we need to specify the number of clusters, in advance and the final results are sensitive to initialization and often terminates at a local optimum.
- Unfortunately, there is no global theoretical method to find the optimal number of clusters.
- A practical approach is to compare the outcomes of multiple runs with different *k* and choose the best one based on a predefined criterion.
- In general, a large *k* probably decreases the error but increases the risk of overfitting.

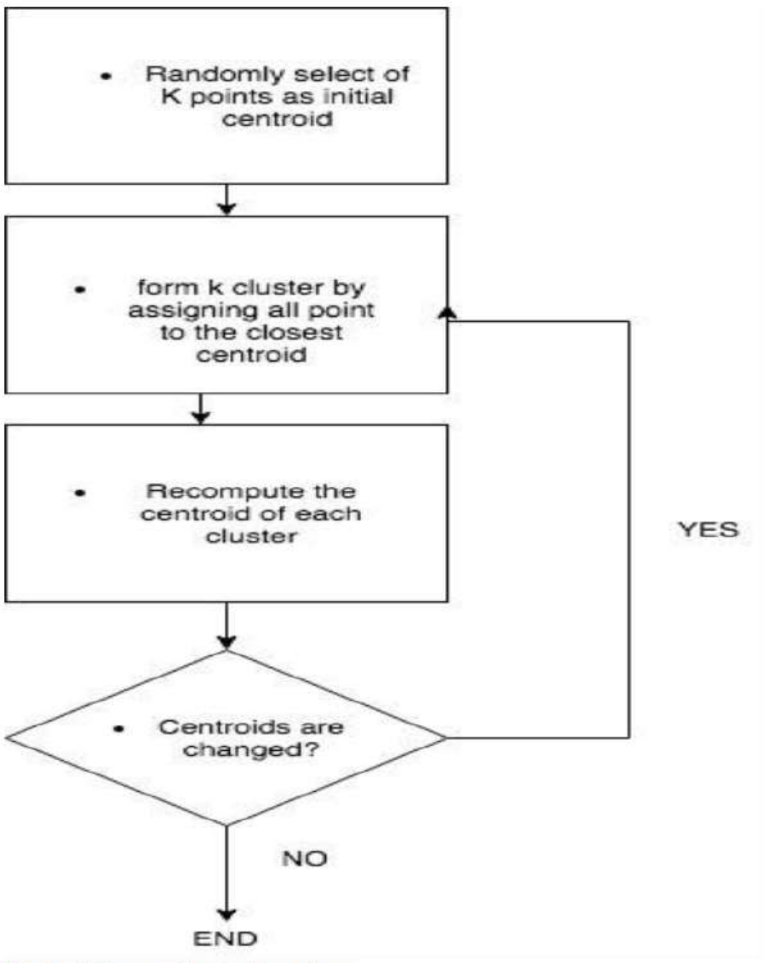


$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2$$

Steps:

1. It starts with K as the input which is how many clusters you want to find. Place K centroids in random locations in your space.
2. Now, using the Euclidean distance between data points and centroids, assign each data point to the cluster which is close to it.

3. Recalculate the cluster centers as a mean of data points assigned to it.
4. Repeat 2 and 3 until no further changes occur.



Randomly select of K points as initial centroid
form k cluster by assigning all point to the closest centroid
Recompute the centroid of each cluster
Centroids are changed?
YES
NO
END

Fig 1: K Means Clustering steps

Algorithm:

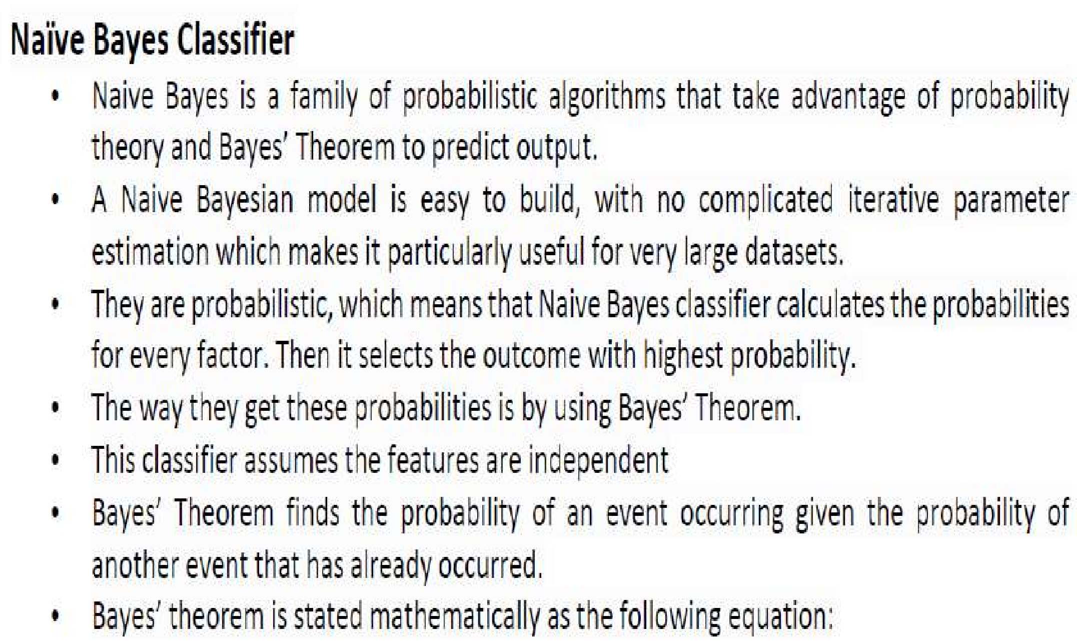
1. Suppose we want to group the visitors to a website using just their age (one-dimensional space) as follows *n* = 19 15,15,16,19,19,20,20,21,22,28,35,40,41,42,43,44,60,61,65
2. Consider initial cluster k=2.
3. Let C1=16 and C2=22
4. Calculate the Euclidean distance d1 and d2 using data points and centroids (C1 and C2).
5. Assign each data point to the respective cluster which is having minimum distance function value. For example, if first data point having d1<d2 then first data point is belonging to first cluster.
6. Calculate new centroid of each cluster by considering the mean of data points present in that cluster
7. Repeat step 4 to 6 till same centroids are generated.
8. Display the clusters

Experiment No.: 11

Title: Implementation of Naïve Bays Classifier.

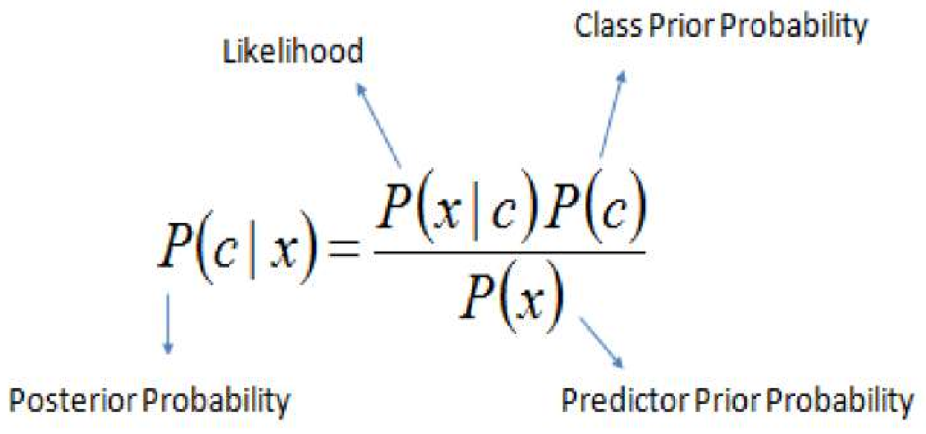
Objectives: To learn Naïve Bayes Classifier.

Theory:



Naïve Bayes Classifier

- Naive Bayes is a family of probabilistic algorithms that take advantage of probability theory and Bayes' Theorem to predict output.
- A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets.
- They are probabilistic, which means that Naive Bayes classifier calculates the probabilities for every factor. Then it selects the outcome with highest probability.
- The way they get these probabilities is by using Bayes' Theorem.
- This classifier assumes the features are independent
- Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred.
- Bayes' theorem is stated mathematically as the following equation:

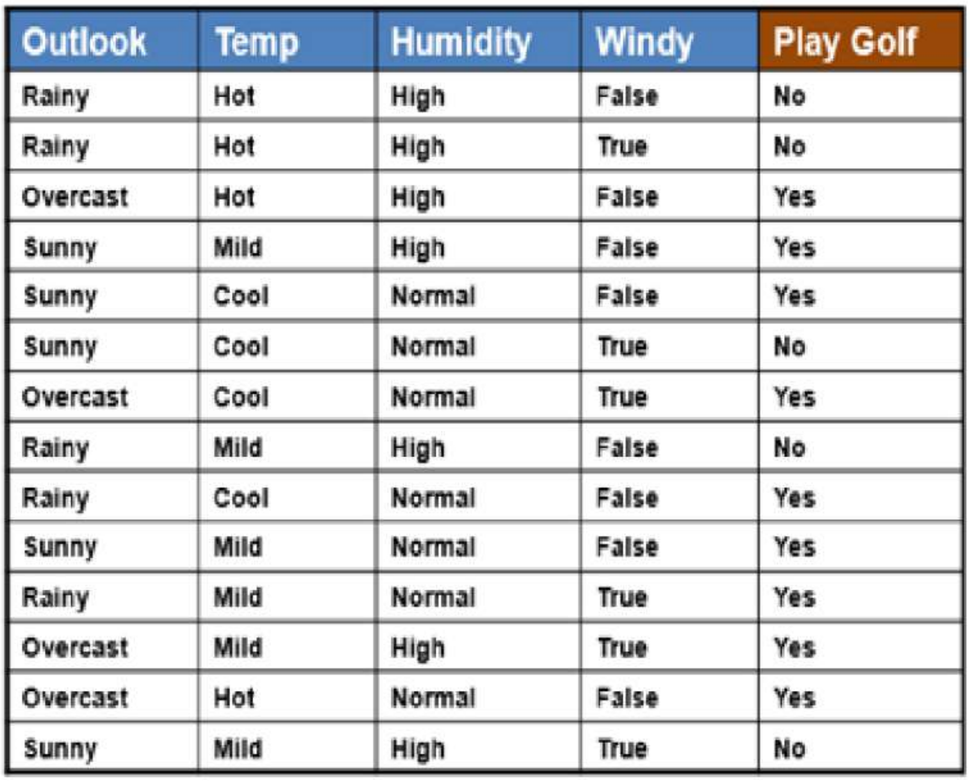


$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

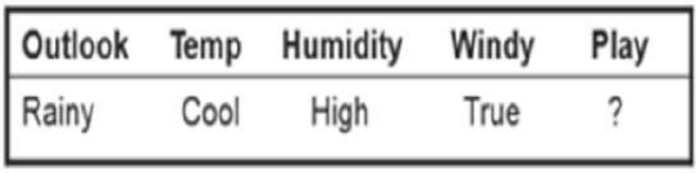
Implementation Procedure:

1. Take input from below table



Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

2. Construct frequency table for each attribute against the target.
3. Then, transforming the frequency tables to likelihood tables.
4. Consider below input



Outlook	Temp	Humidity	Windy	Play
Rainy	Cool	High	True	?

5. Use the Naive Bayesian equation to calculate the posterior probability for each class given above.
6. The class with the highest posterior probability is the outcome of prediction.
7. Display result