MAHA BARATHI ENGINEERING COLLEGE

NH-79, SALEM-CHENNAI HIGHWAY, A.VASUDEVANUR, CHINNASALEM TK, KALLAKURICHI DT – 606 201.
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai
2(f) & 12(B) status of UGC, New Delhi,
www.mbec.ac.in | 04151-256333, 257333 | mbec123@gmail.com

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3361 – DATA SCIENCE LAB MANUAL

II Year/III Semester B.E CSE

Regulation 2021
(As Per Anna University, Chennai syllabus)

Prepared By,
P.AKILA
(AP/CSE)

Verified By,
N.KHADIRKUMAR
(HOD/CSE)

CS3361 **DATA SCIENCE LABORATORY** **L T P C**
0 0 4 2

OBJECTIVES:

- To understand the python libraries for data science
- To understand the basic Statistical and Probability measures for data science.
- To learn descriptive analytics on the benchmark data sets.
- To apply correlation and regression analytics on standard data sets.
- To present and interpret data using visualization packages in Python .

LIST OF EXPERIMENTS:

1. Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages.

2. Working with Numpy arrays

3. Working with Pandas data frames

4. Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.

5. Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following:

a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.

b. Bivariate analysis: Linear and logistic regression modeling

c. Multiple Regression analysis

d. Also compare the results of the above analysis for the two data sets.

6. Apply and explore various plotting functions on UCI data sets.

a. Normal curves

b. Density and contour plots

c. Correlation and scatter plots

d. Histograms

e. Three dimensional plotting

7. Visualizing Geographic Data with Basemap

TOTAL : 60 PERIODS

OUTCOMES:

On completion of this course, the students will be able to:

CO1: Make use of the python libraries for data science

CO2: Make use of the basic Statistical and Probability measures for data science.

CO3: Perform descriptive analytics on the benchmark data sets.

CO4: Perform correlation and regression analytics on standard data sets

CO5: Present and interpret data using visualization packages in Python.

INSTALLING ANACONDA ON WINDOWS

Anaconda distribution of Python is the best option for problem solvers who want to use Python. Anaconda is free (although the download is large which can take time) and can be installed. Anaconda comes bundled with about 600 packages pre-installed including NumPy, Matplotlib and SymPy. These three packages are very useful for problem solvers and will be discussed in subsequent chapters.

Follow the steps below to install the Anaconda distribution of Python on Windows.

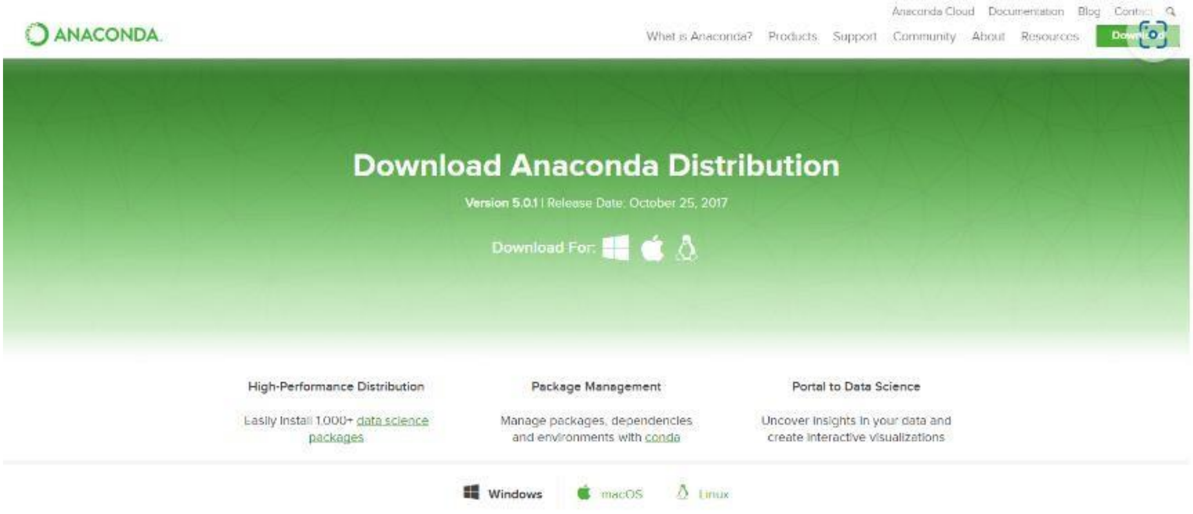
Steps:

1. Visit Anaconda.com/downloads
2. Select Windows
3. Download the .exe installer
4. Open and run the .exe installer
5. Open the Anaconda Prompt and run some Python code

1. Visit the Anaconda downloads page

Go to the following link: Anaconda.com/downloads

The Anaconda Downloads Page will look something like this:



Download Anaconda Distribution

Version 5.0.1 | Release Date: October 25, 2017

High-Performance Distribution

Easily install 1,000+ data science packages

Package Management

Manage packages, dependencies and environments with conda

Portal to Data Science

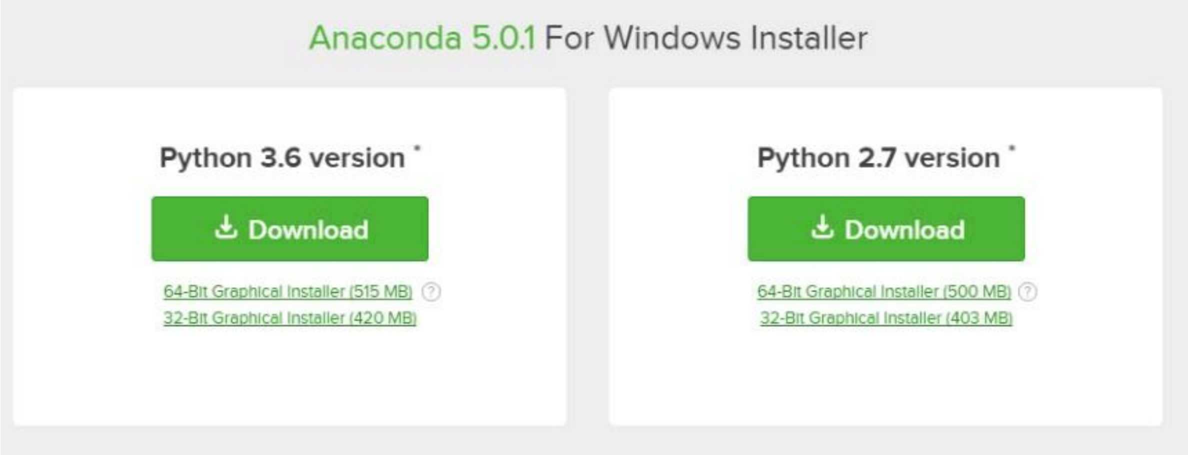
Uncover insights in your data and create interactive visualizations

2. Select Windows

Select Windows where the three operating systems are listed.

3.Download

Download the most recent Python 3 release. At the time of writing, the most recent release was the Python Version. Python 2.7 is legacy Python. For problem solvers, select the Python 3.6 version. If you are unsure if your computer is running a 64-bit or 32-bit version of Windows, select 64-bit as 64-bit Windowsis most common.



Anaconda 5.0.1 For Windows Installer

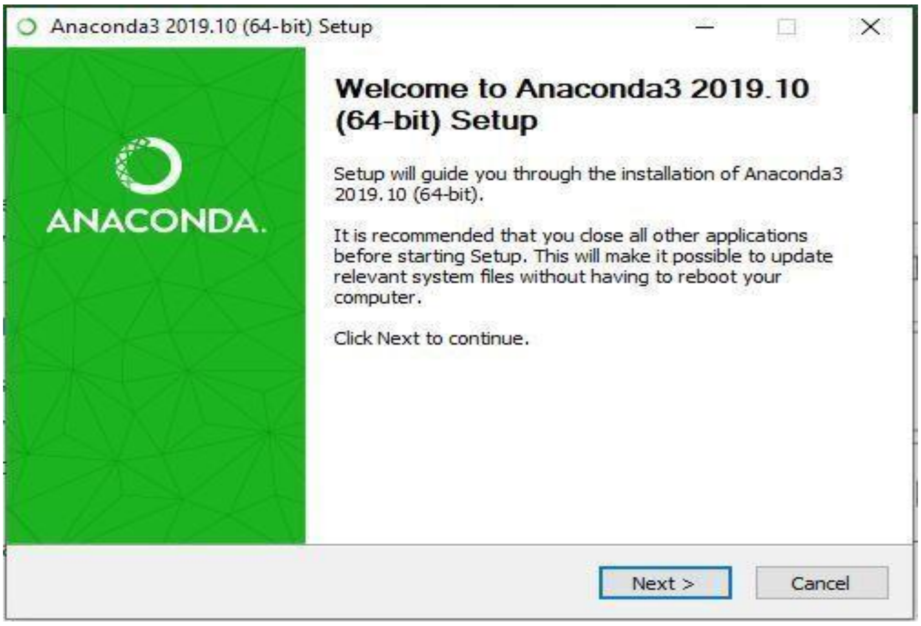
Python 3.6 version *

64-Bit Graphical Installer (515 MB)
32-Bit Graphical Installer (420 MB)

Python 2.7 version *

64-Bit Graphical Installer (500 MB)
32-Bit Graphical Installer (403 MB)

Begin with the installation process:
Getting Started:



Anaconda3 2019.10 (64-bit) Setup

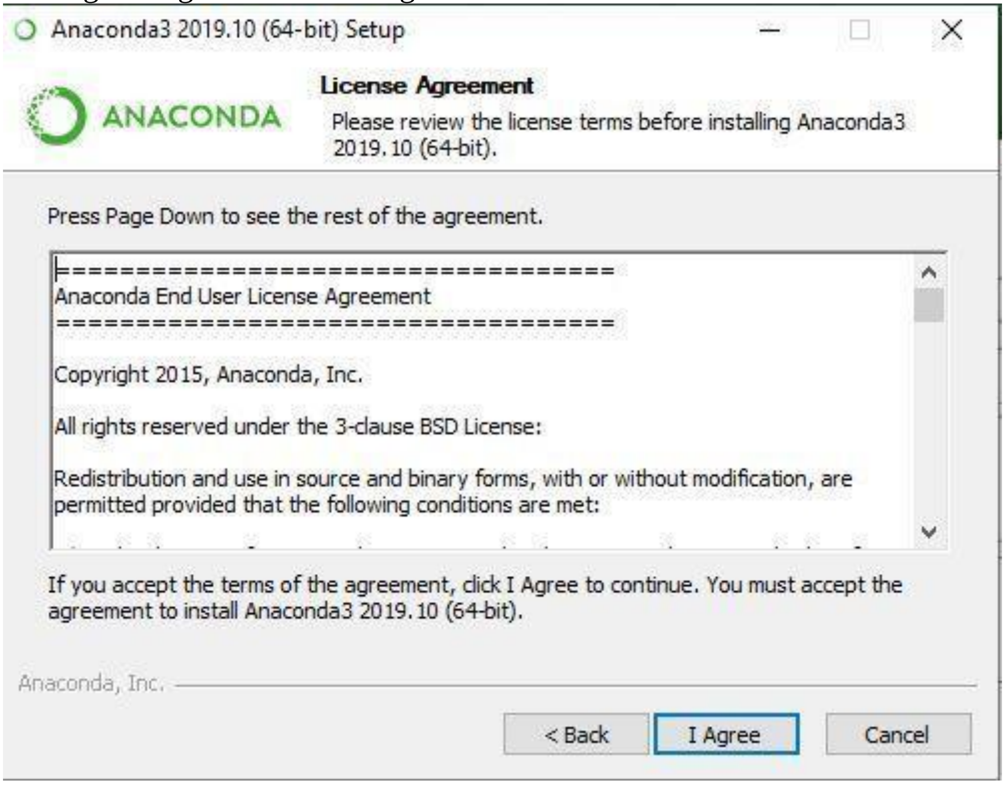
Welcome to Anaconda3 2019.10 (64-bit) Setup

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:



Anaconda3 2019.10 (64-bit) Setup

License Agreement

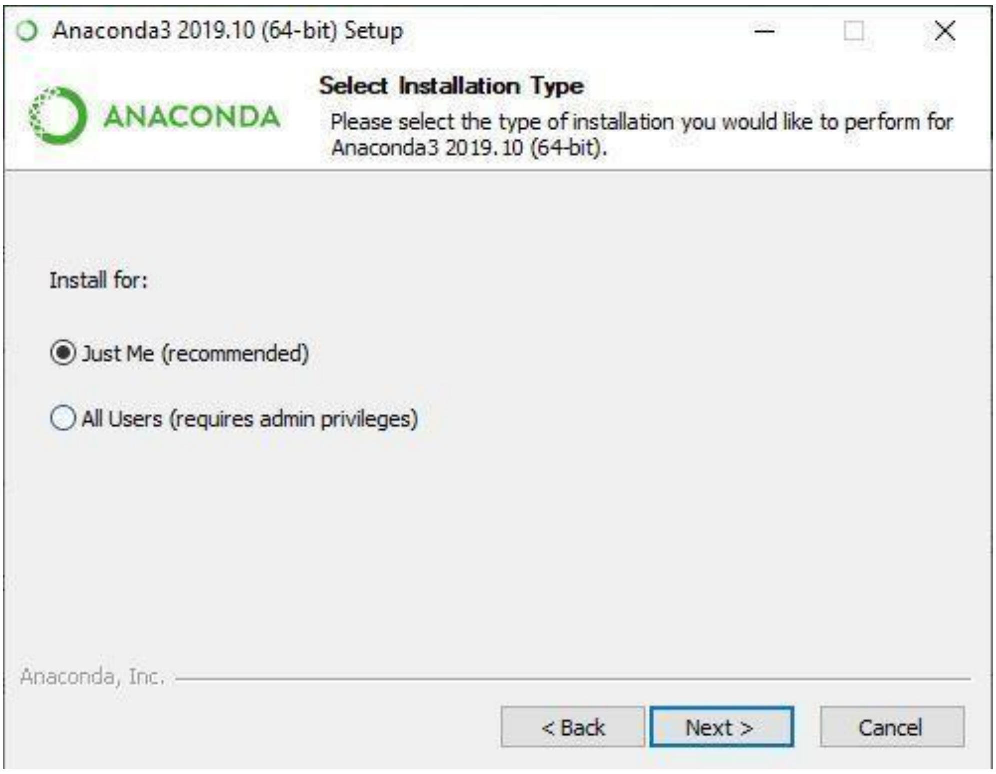
Please review the license terms before installing Anaconda3 2019.10 (64-bit).

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).

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



Anaconda3 2019.10 (64-bit) Setup

Select Installation Type

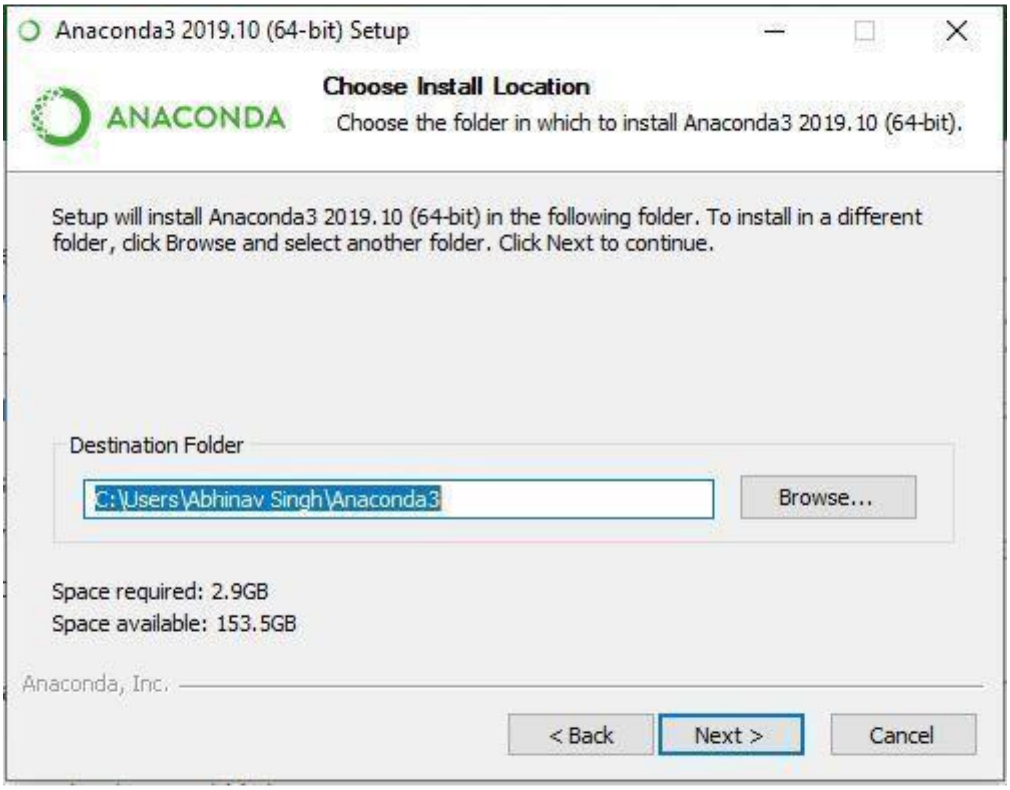
Please select the type of installation you would like to perform for Anaconda3 2019.10 (64-bit).

Install for:

● Just Me (recommended)

○ All Users (requires admin privileges)

Choose Installation Location:



Anaconda3 2019.10 (64-bit) Setup

ANACONDA

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

Browse...

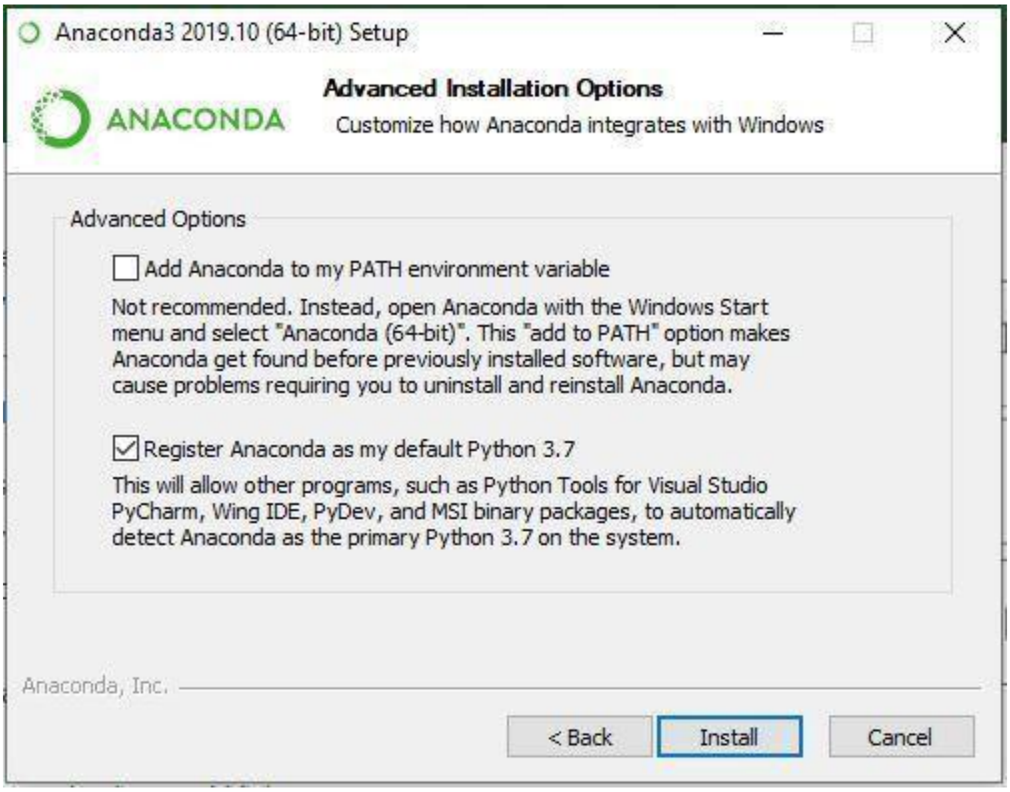
Space required: 2.9GB

Space available: 153.5GB

Anaconda, Inc.

< Back | Next > | Cancel

- Advanced Installation Option:



Anaconda3 2019.10 (64-bit) Setup

ANACONDA

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.

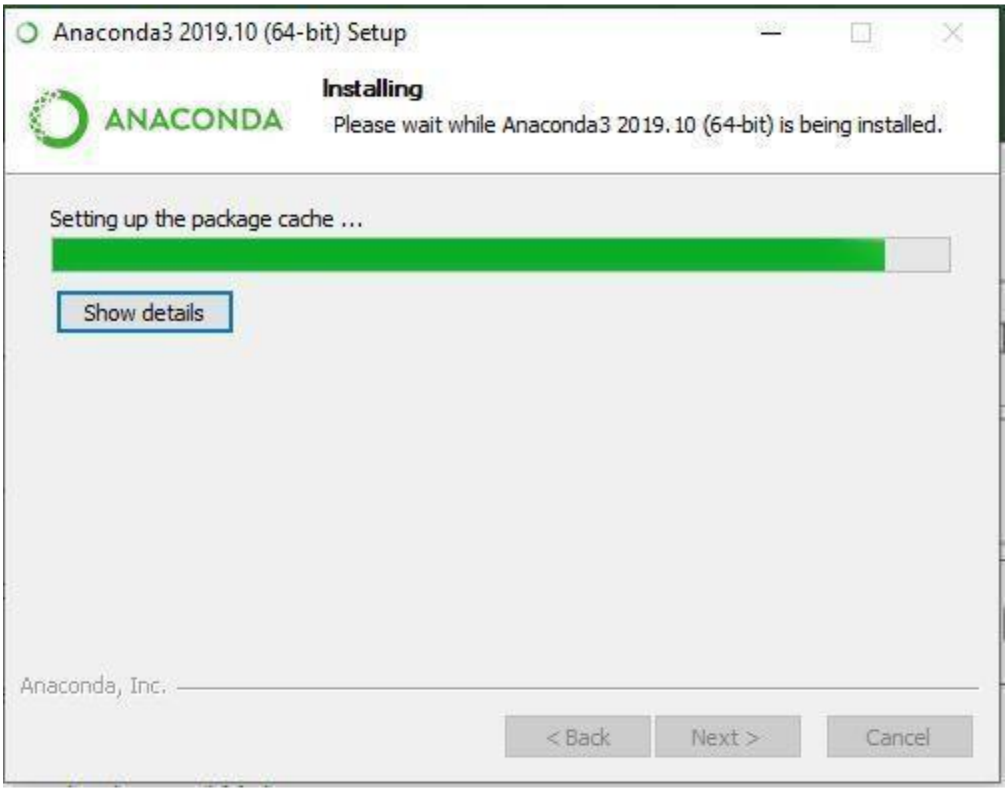
☑ Register Anaconda as my default Python 3.7

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.

< Back | Install | Cancel

- Getting through the Installation Process:



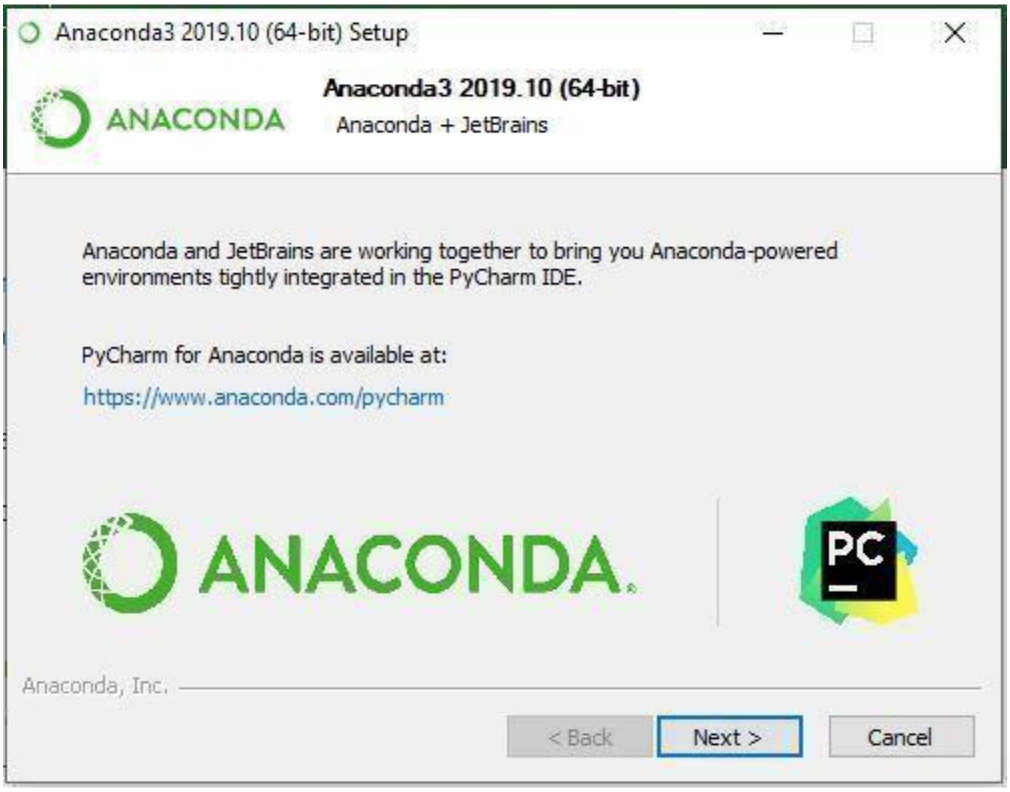
Anaconda3 2019.10 (64-bit) Setup

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)

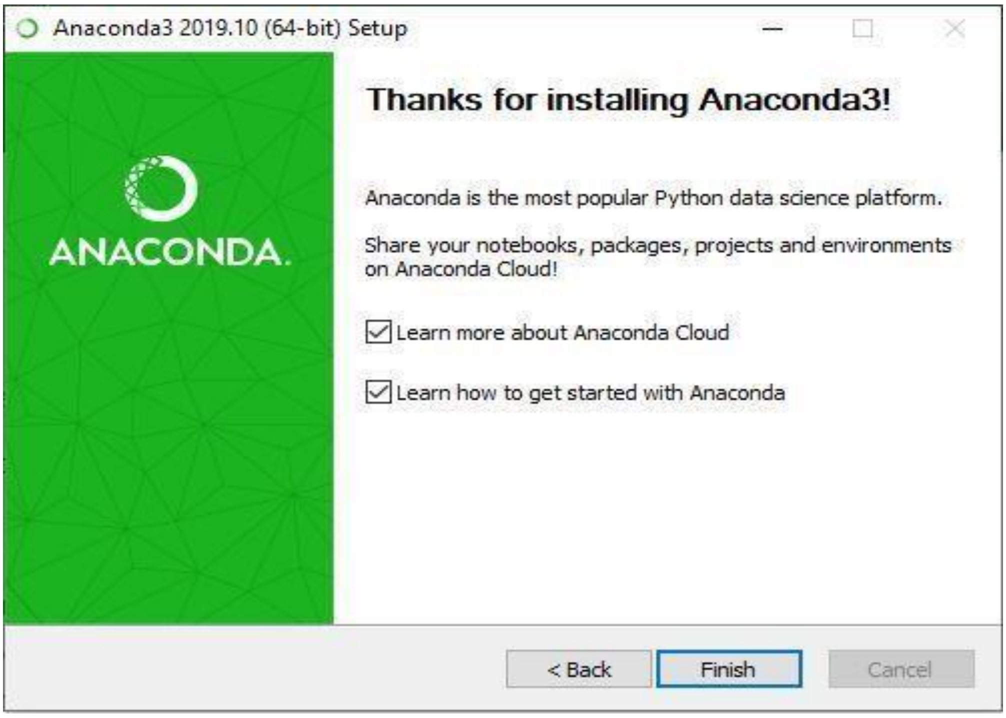
Anaconda + JetBrains

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:



Thanks for installing Anaconda3!

Anaconda is the most popular Python data science platform.

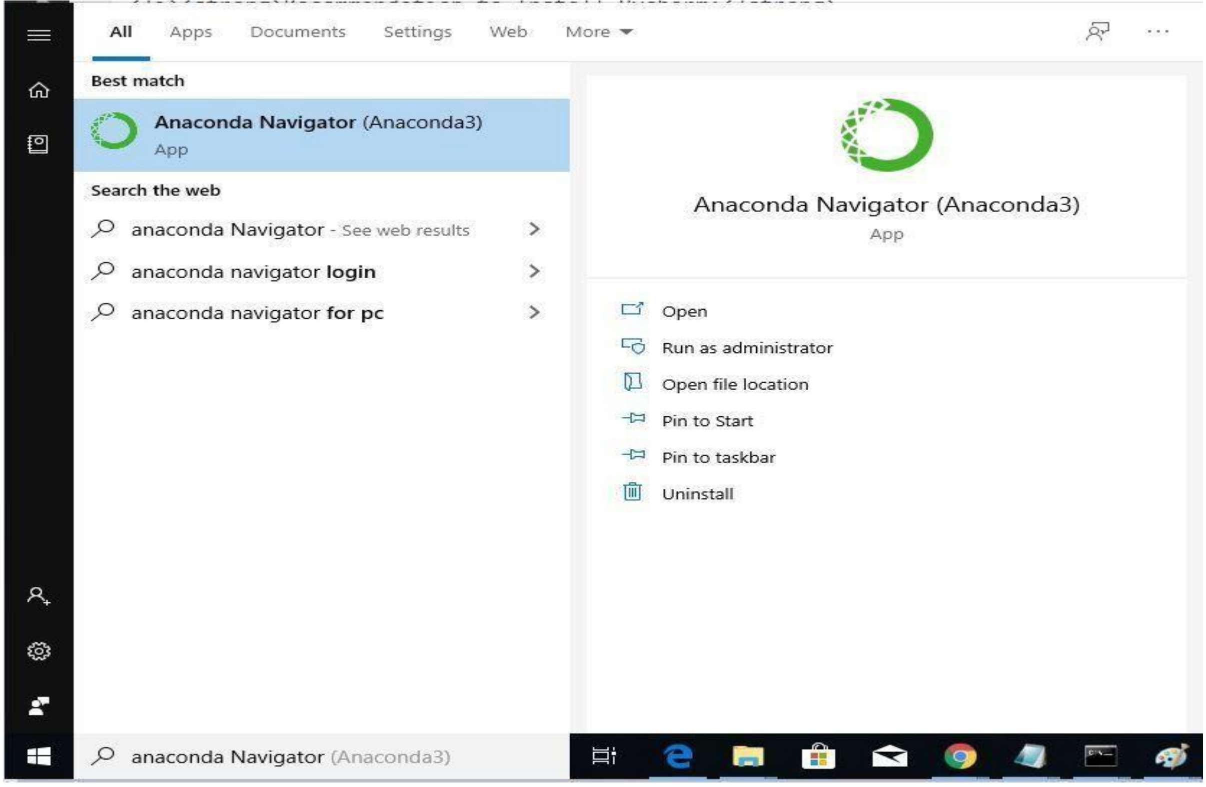
Share your notebooks, packages, projects and environments on Anaconda Cloud!

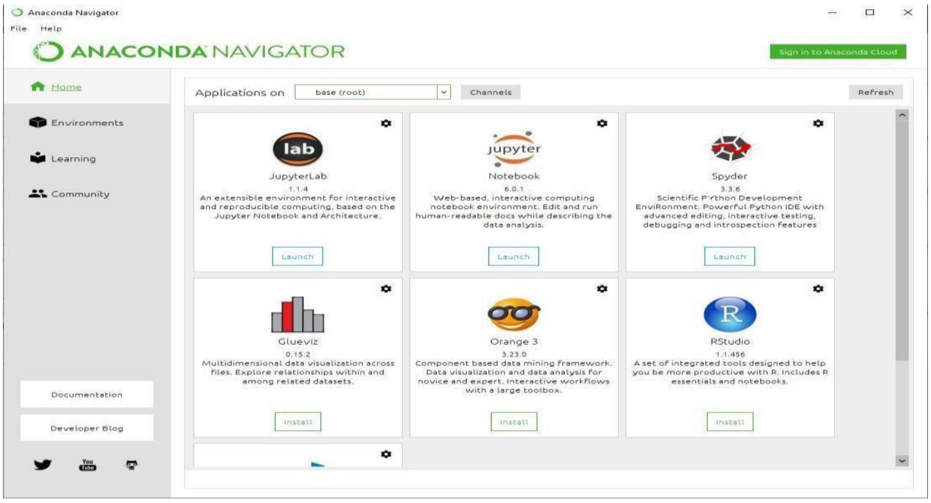
☑ Learn more about Anaconda Cloud

☑ 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.

Exploring NumPy Packages:

NumPy is a Python package used for numerical computation. NumPy is one of the foundational packages for scientific computing with Python. NumPy's core data type is the array and NumPy functions operate on arrays.

Installing NumPy

Before NumPy's functions and methods can be used, NumPy must be installed. Depending on which distribution of Python you use, the installation method is slightly different.

Install NumPy on Anaconda

If you installed the Anaconda distribution of Python, NumPy comes pre-installed and no further installation steps are necessary.

If you use a version of Python from python.org or a version of Python that came with your operating system, the Anaconda Prompt and conda or pip can be used to install NumPy.

Install NumPy with the Anaconda Prompt

To install NumPy, open the Anaconda Prompt and type:

```
> conda install numpy
```

Type y for yes when prompted.

Verify NumPy installation

To verify NumPy is installed, invoke NumPy's version using the Python REPL. Import NumPy and call the . version attribute common to most Python packages.

```
In [1]:
import numpy as np
np.version
```

```
Out[1]:'1.16.4'
```

A version number like '1.16.4' indicates a successful NumPy installation.

Exploring SciPy Packages:

Installing With Pip

You can install SciPy from PyPI with pip:

python -m pip install scipy

Installing Via Conda

You can install SciPy from the defaults or conda-forge channels with conda:

conda install scipy

Exploring Juypter Packages:

Installing Juypter

The simplest way to install Jupyter notebooks is to download and install the Anaconda distribution of Python. The Anaconda distribution of Python comes with Jupyter notebook included and no further installation steps are necessary.

Installing Jupyter on Windows using the Anaconda Prompt

To install Jupyter on Windows, open the Anaconda Prompt and type:

> conda install jupyter

Type y for yes when prompted. Once Jupyter is installed, type the command below into the Anaconda Prompt to open the Jupyter notebook file browser and start using Jupyter notebooks.
> jupyter notebook

Exploring Stats models Packages:

The easiest way to install stats models is to install it as part of the Anaconda distribution, a cross-platform distribution for data analysis and scientific computing. This is the recommended installation method for most users.

Instructions for installing from PyPI, source or a development version are also provided.

Python Support

Stats models supports Python 3.8, 3.9, and 3.10.

Anaconda

Stats models is available through conda provided by Anaconda. The latest release can be installed using:

conda install -c conda-forge stats models

PyPI (pip)

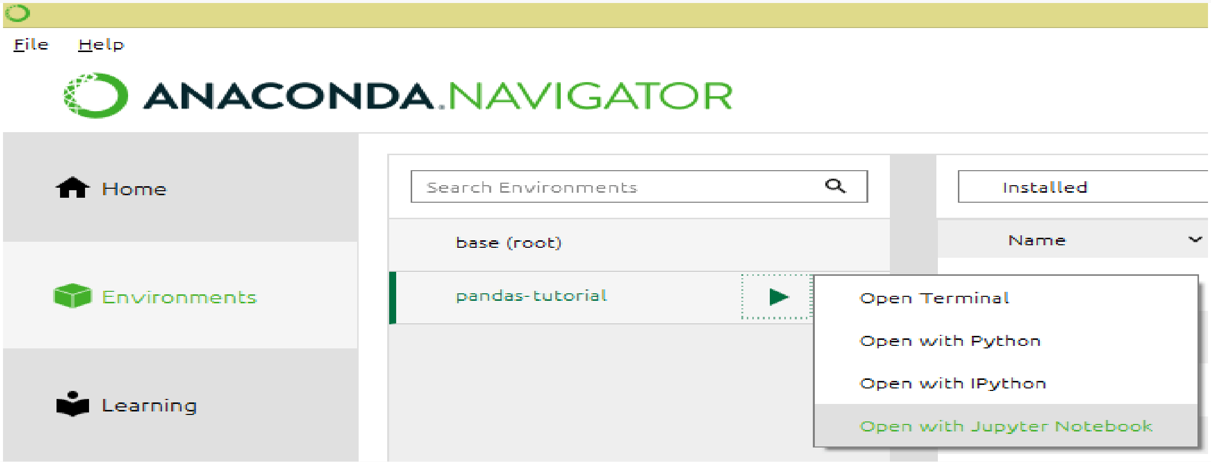
To obtain the latest released version of stats models using pip:

python -m pip install stats model.

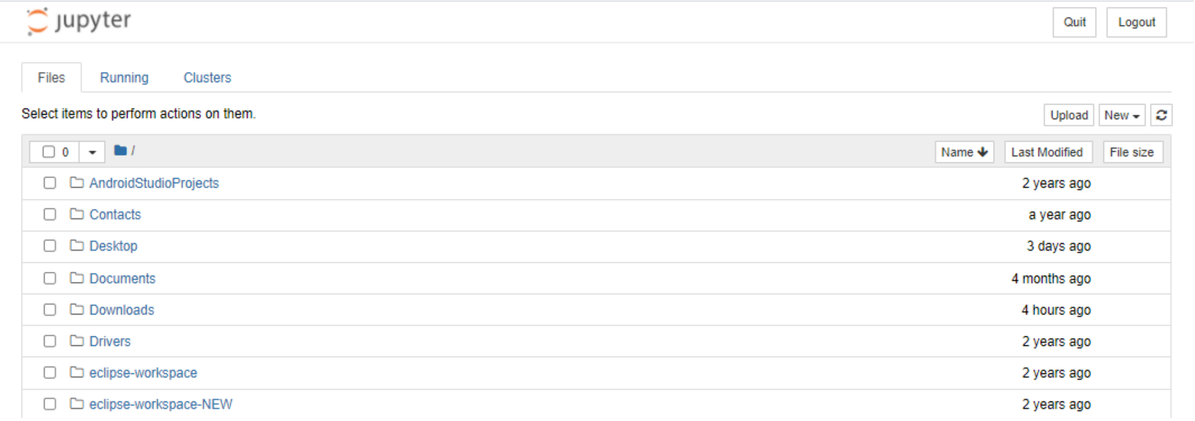
Follow this link to our PyPI page to directly download wheels or source.

Exploring Pandas packages

Go to **Anaconda Navigator** -> **Environments** -> **your environment** (mine pandas-tutorial) -> select **Open with Jupyter Notebook**



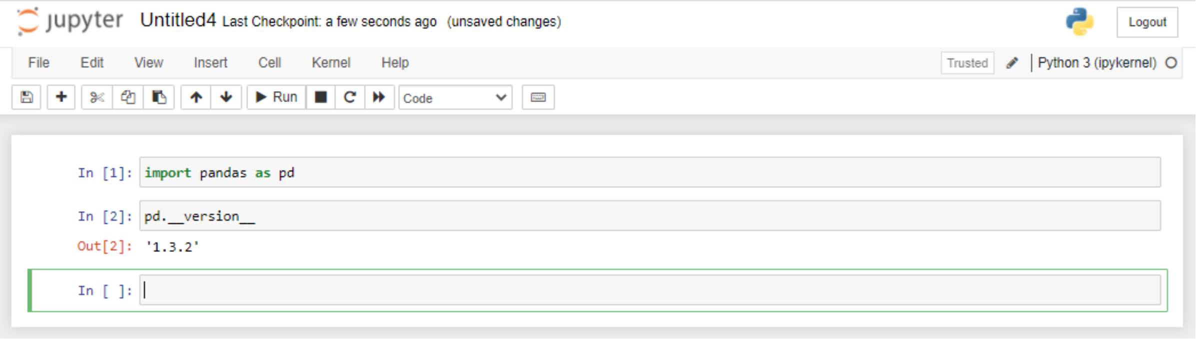
This opens up Jupyter Notebook in the default browser.



Select items to perform actions on them.

Name	Last Modified
AndroidStudioProjects	2 years ago
Contacts	a year ago
Desktop	3 days ago
Documents	4 months ago
Downloads	4 hours ago
Drivers	2 years ago
eclipse-workspace	2 years ago
eclipse-workspace-NEW	2 years ago

Now select New -> Python X and enter the below lines and select Run.


jupyter Untitled4 Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code

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

In [2]: pd.__version__
Out[2]: '1.3.2'

In [ ]:
```

Result:

This completes installing Anaconda and running pandas on Jupyter Notebook.

EX.NO:2 **DATE:**	**ARRAY INDEXING using NUMPY**

AIM:

To write a python program to implement array indexing using numpy

ALGORITHM:

```
Step1: Start
Step2:Import necessary libraries-numpy
Step3: Using random module, seed for reproducibility
Step4: Create one dimensional, two dimensional array using randint
Step5: Access the elements by using the index for the different dimensional array.Step6:
Stop the Program
```

PROGRAM:

```
import numpy as np

np.random.seed(0)  # seed for reproducibility


x1 = np.random.randint(10, size=6)  # One-dimensional array

x2 = np.random.randint(10, size=(3, 4))  # Two-dimensional array

x3 = np.random.randint(10, size=(3, 4, 5)) # Three-dimensional arrayprint(x1)

print(x1[0])

print(x1[4])

#To index from the end of the array, negative indices are usedprint( x1[-

1])

print(x1[-2])


#In a multidimensional array, items are accessed using a comma-separated tuple
```

```
#of indices:

print(x2) print(x2[0,

0])

print(x2[2, 0])

print(x2[2, -1])



#modifying values using index notation:

x2[0, 0] = 12

print(x2)

x1[0] = 3.14159 # this will be truncated!

print(x1)
```

OUTPUT

```
 [5 0 3 3 7 9]

5

7

9

7

[[3 5 2 4]

 [7 6 8 8]

 [1 6 7 7]]

3

1

7

[[12  5  2  4]

 [ 7  6  8  8]

 [ 1  6  7  7]]
```

```
[3 0 3 3 7 9]
```

INFERENCE:

Array indexing is required for accessing the elements in that array. In the above program I havelearnt to implement array indexing using numpy for a three dimensional array.

RESULT: This program was successfully executed using NUMPY.

EX.NO: 3	ARRAY SLICING using NUMPY
DATE:	

AIM:

To write a python program to implement array slicing using numpy

ALGORITHM:

```
Step1: START
Step2: Import necessary libraries -numpy
Step 3:Using arrange function, print n elements
Step 4:By using the slice method, [x:n] , array slicing can be done
Step 5: Similarly, array slicing for the two dimensional array can be doneStep 6:
STOP
```

PROGRAM:

```
import numpy as np

np.random.seed(0)  # seed for reproducibility


x1 = np.random.randint(10, size=6)  # One-dimensional array

x2 = np.random.randint(10, size=(3, 4))  # Two-dimensional array

x3 = np.random.randint(10, size=(3, 4, 5)) # Three-dimensional arrayprint(x1)
print(x1[0])

print(x1[4])

#To index from the end of the array, negative indices are usedprint( x1[-
1])
print(x1[-2])


#In a multidimensional array, items are accessed using a comma-separated tuple#of
indices:
```

```
print(x2) print(x2[0,
0])
print(x2[2, 0])
print(x2[2, -1])



#modifying values using index notation:
x2[0, 0] = 12
print(x2)
x1[0] = 3.14159 # this will be truncated!
print(x1)
```

OUTPUT:

```
[5 0 3 3 7 9]
5
7
9
7
[[3 5 2 4]
 [7 6 8 8]
 [1 6 7 7]]
3
1
7
[[12  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
[3 0 3 3 7 9]
```

INFERENCE:

Array slicing is required for accessing certain the elements in that array. In the above program Ihave learnt to implement array slicing using numpy.

RESULT: This program was successfully executed using NUMPY.

EX.NO:4	**SUBARRAYS using NUMPY**
DATE:	

AIM:

To write a python program to implement subarrays using numpy

ALGORITHM:

```
Step 1:START
Step 2:Import the necessary libraries – numpy
Step 3:Usingrandint and random module create a two dimensional arrayStep
4:Extract  a n*n subarray from main array
Step 5:print the elements in sub array
Step 6:STOP
```

PROGRAM:

```
import numpy as np
x = np.arange(10)
print(x)
print(x[:5]) # first five elements
print(x[5:]) # elements after index 5
print(x[4:7]) # middle subarray
print(x[::2] )# every other element
print(x[1::2])# every other element, starting at index 1
print(x[::-1]) # all elements, reverse
print(x[5::-2]) # reversed every other from index 5
x2 = np.random.randint(10, size=(3, 4)) # Two-dimensional array
print(x2)
print(x2[:2, :3]) # two rows, three columns
print(x2[:3, ::2]) # all rows, every other column
print(x2[::-1, ::-1])#subarray dimensions reversed together
print(x2[:, 0]) # first column of x2
print(x2[0, :]) # first row of x2
print(x2[0]) # equivalent to x2[0, :]
```

OUTPUT:

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

[0 1 2 3 4]

[5 6 7 8 9]
```

```
[4 5 6]

[0 2 4 6 8]

[1 3 5 7 9]

[9 8 7 6 5 4 3 2 1 0]

[5 3 1]

[[6 4 3 4]

 [8 6 0 6]

 [4 7 2 1]]

[[6 4 3]

 [8 6 0]]

[[6 3]

 [8 0]

 [4 2]]

[[1 2 7 4]

 [6 0 6 8]

 [4 3 4 6]]

[6 8 4]

[6 4 3 4]

[6 4 3 4]
```

INFERENCE:

Sub arrays are required for further processing. From this program, we learnt to extract a subarray from the main two dimensional array and print the elements in sub array

RESULT: This program was successfully executed using NUMPY.

EX.NO:5	**DATA INDEXING AND SELECTION USING PANDAS**
DATE:	

AIM:

To write a python program to implement data indexing and selection using pandas

ALGORITHM:

Step 1:START
Step 2:Import the necessary libraries – pandas
Step 3:Create a series using series module from pandas
Step 4:Creat a rows and columns (i.e) index and values respectively using pandas series functionStep
5:print the one dimensional array within a range using string slicing
Step 6:STOP

PROGRAM:

```
#Subarrays as no-copy views
import numpy as np
x2 = np.random.randint(10, size=(3, 4)) # Two-dimensional array

#extract a 2×2 subarray from this
x2_sub = x2[:2, :2]
print(x2_sub)

#if we modify this subarray, we'll see that the original array is changed!
x2_sub[0, 0] = 99
print(x2_sub)
print(x2)
#when we work with large datasets, we can access and process pieces of these datasets without the
need to copy the underlying data buffer.
```

OUTPUT:

```
[[0 1]

 [8 4]]

[[99  1]

 [ 8  4]]

[[99  1  2  2]
 [ 8  4  5  9]

 [ 9  3  6  5]]
```

INFERENCE:

Pandas are packages that can be added to python for doing the data analysis. From this program,we learnt to construct series as objects using pandas libraries.

RESULT: This program was successfully executed using PANDAS.

EX.NO:6	**OBJECT as Series using PANDAS**
DATE:	

AIM:

To write a python program to implement object as series using pandas

ALGORITHM:

Step 1:START
Step 2:Import the necessary libraries-numpy,pandas.
Step 3:Create a series using numpy array
Step 4:Create a specialized dictionary and build a series.Step
5:print the series by using the pandas
Step 6:STOP

PROGRAM:

```
#PANDAS SERIES AS OBJECT
importnumpyas np
import pandas as pd
data = pd.Series([0.25, 0.5, 0.75, 1.0])
print(data)
print(data.values)
print(data.index)
print(data[1])
print(data[1:3])
#series as numpy array
data = pd.Series([0.25, 0.5, 0.75, 1.0],index=['a', 'b', 'c', 'd'])
print(data)
print(data['b'])
data = pd.Series([0.25, 0.5, 0.75, 1.0],index=[2, 5, 3, 7])
print(data)
print(data[5])
#series as specilized dictionary
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
population = pd.Series(population_dict)
print(population)
print(population['California'])
print(population['California':'Florida'])
#constructing series objects
a=pd.Series([2, 4, 6])
print(a)
b=pd.Series(5, index=[100, 200, 300])
print(b)
c=pd.Series({2:'a', 1:'b', 3:'c'})
print(c);
#after indexing
c=pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2])
print(c)
```

OUTPUT:

```
0   0.25
1   0.50
2   0.75
3   1.00
dtype: float64
[0.25 0.5  0.75 1.
]
RangeIndex(start=0, stop=4,
step=1)0.5
1   0.50
2   0.75
dtype:
float64a
    0.25
b   0.50
c   0.75
d   1.00
dtype:
float640.5
2   0.25
5   0.50
3   0.75
7   1.00
dtype:
float640.5
California   38332521
Texas      26448193
New York    19651127
Florida    19552860
Illinois    12882135
dtype:
int64
38332521
California  38332521
Texas      26448193
New York    19651127
Florida    19552860
dtype:
int640
    2
    1   4
    2   6
dtype: int64
    100   5
    200   5
    300   5
    dtype:
    int64
    2   a
    1   b
    3   c
    dtype: object
    3   c
    2   a
    dtype: object
```

INFERENCE:

Pandas are packages that can be added to python for doing the data analysis. From this problem,we learnt to create a dataframe as specialized dictionary using pandas library functions

RESULT: This program was successfully executed using PANDAS.

EX.NO:7	**DATAFRAME OBJECT SERIES AS SPECILIZED DICTIONARY USING PANDAS**
DATE:	

AIM:

To write a python program to implement dataframe object series as specilized dictionary usingpandas

ALGORITHM:

Step 1:START
Step 2:Import the necessary libraries-pandas
Step 3:Create a dictionary named population_dict.
Step 4:create a series by using the pandas libraries
Step 5:print the results.
Step 6:STOP

PROGRAM:

```
import pandas as pd
#PANDAS DATAFRAME OBJECT
#series as specilized dictionary
population_dict = {'California': 38332521,
                   'Texas': 26448193,
                   'New York': 19651127,
                   'Florida': 19552860,
                   'Illinois': 12882135}
population = pd.Series(population_dict)
area_dict = {'California': 423967, 'Texas': 695662, 'New York': 141297,'Florida':
170312, 'Illinois': 149995}
area = pd.Series(area_dict)
print(area)
print()
states = pd.DataFrame({'population': population,'area': area})
print(states)
print()
print(states.index)
print()
print(states.columns)
print()
#dataframe as specilized dictionary
```

```
print(states['area'])
print()
a= pd.DataFrame(population, columns=['population'])
print(a)
print()
```

OUTPUT:

```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995


dtype: int64

population      area

California    38332521  423967
Texas         26448193  695662
New York      19651127  141297
Florida       19552860  170312
Illinois      12882135  149995

Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'], dtype='object')

Index(['population', 'area'], dtype='object')


California       423967
Texas            695662
New York         141297
Florida          170312
Illinois         149995


Name: area, dtype: int64

                population

California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
```

INFERENCE:

Pandas are packages that can be added to python for doing the data analysis. From this problem,we learnt to have to create dataframe object series as specialized dictionary using pandas.

RESULT: This program was successfully executed using PANDAS.

EX.NO:8	**KNN CLASSIFICATION FOR USE OF IRIS DATASET**
DATE:	

AIM:

To write a python program to implement knn classification for use of iris dataset

ALGORITHM:

Step 1: Load and Train the IRIS data
Step 2: Initialize K to your chosen number of neighbours.
Step 3: For each example in the data

i. Calculate the distance between the query example and the current example from the data.
ii. Add the distance and the index of the example to an ordered collection.
iii. Sort the ordered collection of distances and indices from smallest to largest (in ascendingorder) by the distances
iv. Pick the first K entries from the sorted collection
v. Get the labels of the selected K entries
vi. Classify the new category as the mode of the K labels and return type

PROGRAM:

```
# Make Predictions with k-nearest neighbors on the Iris Flowers Dataset from csvimport

reader
from math import sqrt

# Load a CSV file
def load_csv(filename): dataset = list()
with open(filename, 'r') as file: csv_reader = reader(file) for row in csv_reader:if not
row:
continue dataset.append(row)
return dataset

# Convert string column to float
def str_column_to_float(dataset, column):for
row in dataset:
row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
class_values = [row[column]
for row in dataset] unique = set(class_values)
 lookup = dict()
 for i, value in enumerate(unique): lookup[value] = iprint('[%s] =>
 %d' % (value, i)) for row in dataset: row[column] =
 lookup[row[column]] return lookup
 # Find the min and max values for each column def
 dataset_minmax(dataset):
 minmax = list()
 for i in range(len(dataset[0])):
 col_values = [row[i] for row in dataset] value_min = min(col_values)value_max =
 max(col_values)
 minmax.append([value_min, value_max]) return minmax

 # Rescale dataset columns to the range 0-1 def
 normalize_dataset(dataset, minmax):
 for row in dataset:
 for i in range(len(row)):
 row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

 # Calculate the Euclidean distance between two vectors def
 euclidean_distance(row1, row2):
 distance = 0.0
 for i in range(len(row1)-1):
 distance += (row1[i] - row2[i])**2 return sqrt(distance)

 # Locate the most similar neighbors
 def get_neighbors(train, test_row, num_neighbors): distances = list()for train_row
 in train:
 dist = euclidean_distance(test_row, train_row)
 distances.append((train_row, dist))
 distances.sort(key=lambda tup: tup[1]) neighbors = list()
 for i in range(num_neighbors): neighbors.append(distances[i][0])return neighbors

 # Make a prediction with neighbors
 def predict_classification(train, test_row, num_neighbors): neighbors =
 get_neighbors(train, test_row, num_neighbors) output_values = [row[-1] for row in
 neighbors]
 prediction = max(set(output_values), key=output_values.count) return prediction

 # Make a prediction with KNN on Iris Dataset filename = 'iris.csv' dataset =
 load_csv(filename) for i in range(len(dataset[0])-1): str_column_to_float(dataset, i)
 # convert class column to integers
```

```
str_column_to_int(dataset, len(dataset[0])-1) # define model parameternum_neighbors = 5
# define a new record row = [5.1,3.7,1.5,0.4]# predict
the label
label = predict_classification(dataset, row, num_neighbors)
print('Data=%s, Predicted: %s' % (row, label))
```

OUTPUT:

```
[Setosa] => 0

[Versicolor] => 1

[Virginica] => 2

Data=[5.1, 3.7, 1.5, 0.4],

Predicted: 0
```

INFERENCE:

Classification is used to classify the given data into known groups. In this program we classifythe IRIS data.

RESULT: This program was successfully executed.

EX.NO:9	**CLASSIFICATION USING LINEAR REGRESSION**
DATE:	

AIM:

To write a python program to implement classification using linear regression

ALGORITHM:

Step1: Consider a set of values x, y.
Step2: Take the linear set of equation y = a+bx.
Step3: Computer value of a, b with respect to the given values, b = n∑xy − (∑x) (∑y) / n∑x2−(∑x)2, a = ∑y−b (∑x)n.

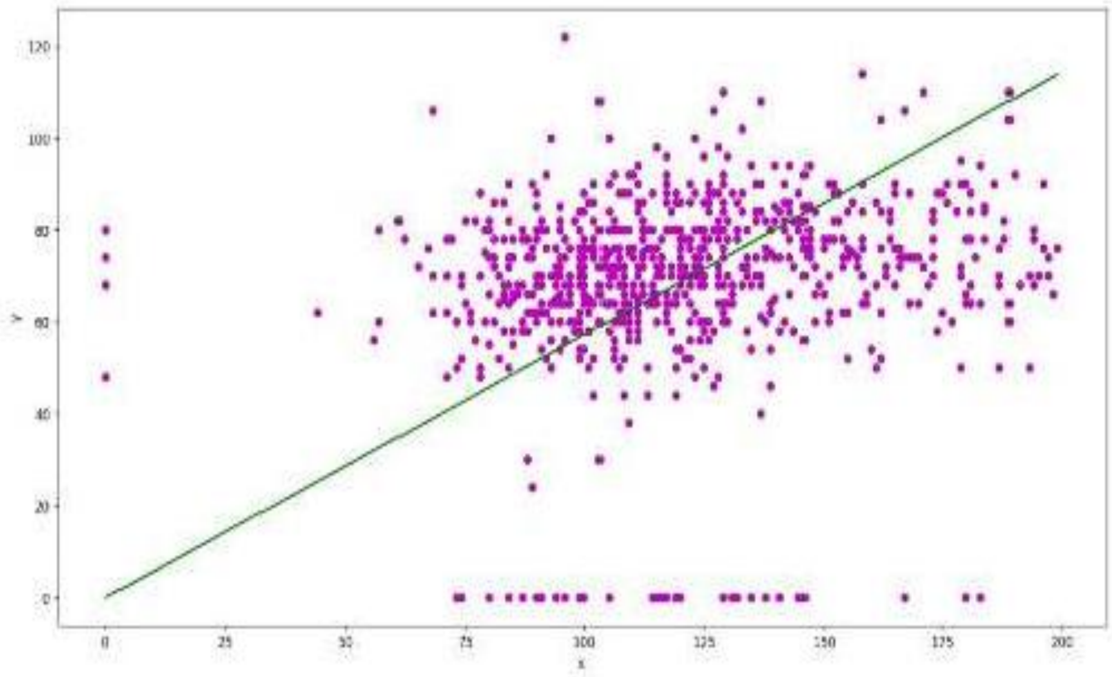
Step4: Implement the value of a, b in the equation y = a+ bx.
Step5: Regress the value of y for any x.

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from csv import DictReader def
estimate_coef(x, y):
   # number of observations/pointsn
   = np.size(x)
   # mean of x and y vector
m_x, m_y = np.mean(x), np.mean(y)
   # calculating cross-deviation and deviation about x
SS_xy = np.sum(y*x - n*m_y*m_x)
SS_xx  = np.sum(x*x  - n*m_x*m_x) #
   calculating regression coefficientsb_1
   = SS_xy / SS_xx
   b_0 = m_y - b_1*m_x
return(b_0, b_1)
def plot_regression_line(x, y, b):
   # plotting the actual points as scatter plot
plt.scatter(x, y, color = "m",
   marker = "o", s = 30)
   # predicted response vector
y_pred = b[0] + b[1]*x
   # plotting the regression line
plt.plot(x, y_pred, color = "g")
```

```
    # putting labels
plt.xlabel('x')
plt.ylabel('y')
    # function to show plot
plt.show()
def main():
    # observations
    Data = []
X,Y=[],[]
    # opening csv file
    with open('diabetes.csv','r') as file:
        reader = DictReader(file)
        for row in reader:
Data.append(row)
    for i in Data: X.append(int(i['Glucose']))
Y.append(int(i['BloodPressure']))
    x = np.array(X)y
    = np.array(Y)
    # estimating coefficientsb
    = estimate_coef(x, y)
print("Estimated coefficients:\nb_0 = {} nb_1 = {}".format(b[0], b[1]))# plotting
    regression line
plot_regression_line(x, y, b) if
name_______== "_main_":
main()
```

OUTPUT:



INFERENCE:

Linear regression is knowing the relationship between two values .From this program we learnt about the how to implement linear regression using python

RESULT: This program was successfully executed.

EX.NO:10	**CLASSIFICATION USING LOGISTIC REGRESSION**
DATE:	

AIM:

To write a python program to implement classification using logistic regression

ALGORITHM:

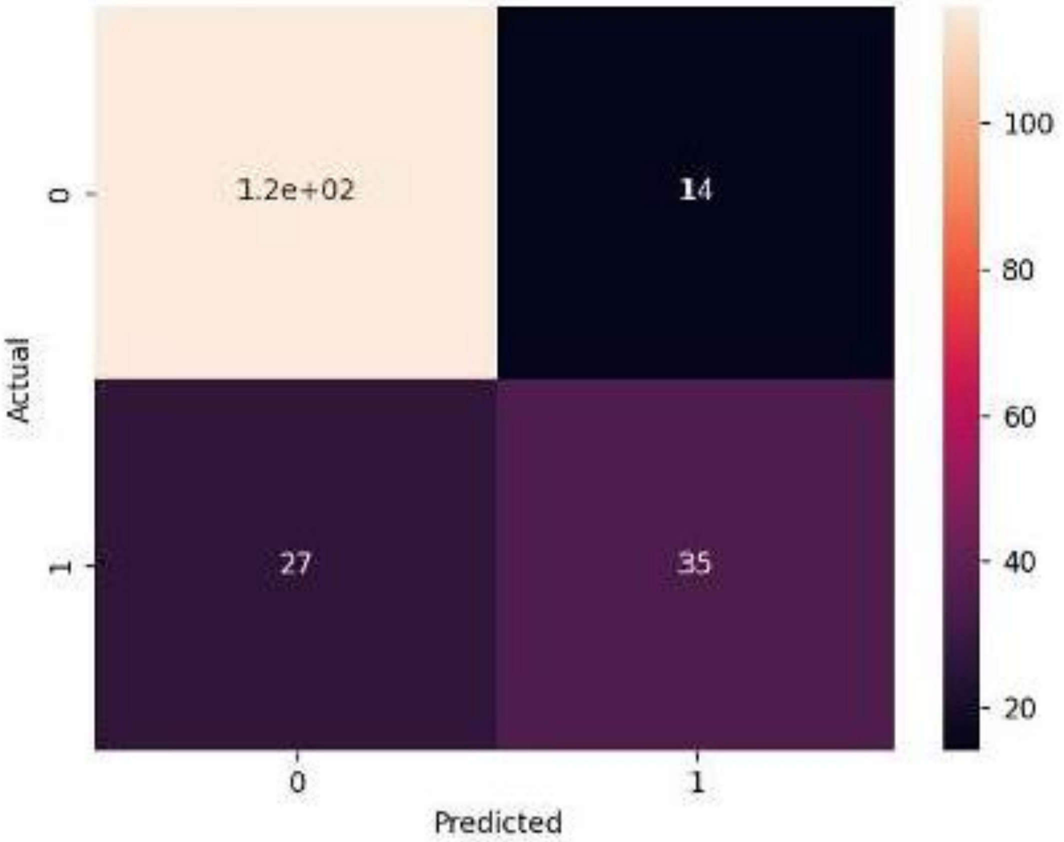
Step1: Initialize the variables
Step2: Set the Data frame
Step3: Spilt data set into training and testing. Step4: Fit the data into logistic regression function.Step5: Predict the test data set.
Step6: Print the results.

PROGRAM:

```
importpandasaspd
fromsklearn.model_selectionimporttrain_test_split
fromsklearn.linear_modelimportLogisticRegression
fromsklearnimport metrics
import seaborn assn
importmatplotlib.pyplotasplt
fromcsvimportDictReader

Data = []
Glucose,BloodPressure,BMI,Outcome=[],[],[],[]
# opening csv file
withopen('diabetes.csv','r') asfile:
    reader = DictReader(file)
    forrowinreader:
        Data.append(row)
foriinData:
    Glucose.append(int(i['Glucose']))
    BloodPressure.append(int(i['BloodPressure']))
    BMI.append(float(i["BMI"]))
    Outcome.append(int(i["Outcome"]))

candidates =
{'Glucose':Glucose,'BMI':BMI,'BloodPressure':BloodPressure,'Outcome': Outcome}
```

```
df = pd.DataFrame(candidates,columns= ['Glucose',
'BMI','BloodPressure','Outcome'])
print (df)
print("Df printed\n")
X = df[['Glucose', 'BMI','BloodPressure']]
y = df['Outcome']
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.25,random_state=0)
print (X_train)
print (y_train)
print("Train\n")
logistic_regression= LogisticRegression()
logistic_regression.fit(X_train,y_train)
y_pred=logistic_regression.predict(X_test)
confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'],
colnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
print('Accuracy: ',metrics.accuracy_score(y_test, y_pred))
print (X_test) #test dataset
print (y_pred) #predicted values
print('confusion_matrix:', confusion_matrix, sep='\n', end='\n\n')
plt.show()
```

OUTPUT:



INFERENCE:

Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring. From this program we learnt to draw the logistics regressions using python

RESULT: This program was successfully executed.

EX.NO:12	MULTIPLE REGRESSION ANALYSIS
DATE:	

AIM:

To write a python program to implement multiple regression analysis

ALGORITHM:

Step1: Get the multi-attribute dataset using the Scikit-learn data source.Step
2: Create a regression object.
Step 3: Train the dataset with the regression model fit.
Step 4: Get and print the regression coefficients and variance.Step 5.
Plot the residual error.

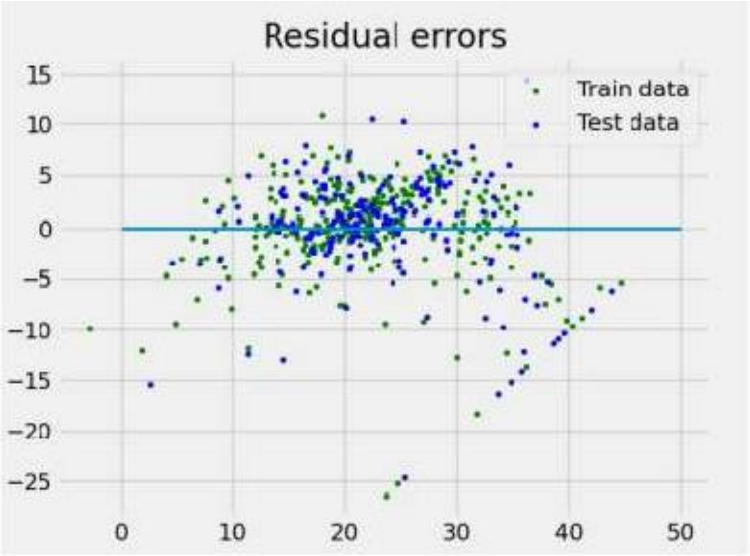
PROGRAM:

```
import matplotlib.pyplot as plt import numpy as np from

sklearn import datasets, linear_model, metrics # load the

boston dataset

boston = datasets.load_boston(return_X_y=False) # defining

feature matrix(X) and response vector(y)

X = boston.data y = boston.target

# splitting X and y into training and testing sets from
sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,

random_state=1)

# create linear regression object

reg = linear_model.LinearRegression()

# train the model using the training sets reg.fit(X_train, y_train)# regression

coefficients print('Coefficients: ', reg.coef_)
```

```
# variance score: 1 means perfect prediction print('Variance score:
{}'.format(reg.score(X_test, y_test))) # plot
for residual error
## setting plot style plt.style.use('fivethirtyeight')
## plotting residual errors in training data plt.scatter(reg.predict(X_train),reg.predict(X_train) - y_train,
color = "green", s = 10, label = 'Train data')
## plotting residual errors in test data plt.scatter(reg.predict(X_test),reg.predict(X_test) - y_test,
color = "blue", s = 10, label = 'Test data')##
plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)##
plotting legend plt.legend(loc = 'upper right') ## plot title
plt.title("Residual errors")
## method call for showing the plot plt.show()
```

OUTPUT:

```
Coefficients:
[-8.95714048e-02 6.73132853e-02
5.04649248e-02 2.18579583e+00
 -1.72053975e+01 3.63606995e+00
2.05579939e-03 -1.36602886e+00
 2.89576718e-01 -1.22700072e-02 -
8.34881849e-01 9.40360790e-03
 -5.04008320e-01]
Variance score: 0.720905667266178
```



INFERENCE:

Multiple regression is a statistical technique that can be used to analyze the relationship between a single dependent variable and several independent variables. The objective of multiple regression analysis is to use the independent variables whose values are known to predict the value of the single dependent value. From this program we learnt to draw the multiple linear regression.

RESULT: This program was successfully executed.

EX.NO: 13	**NORMAL CURVES**
DATE:	

AIM:

To write a python program to implement normal curves

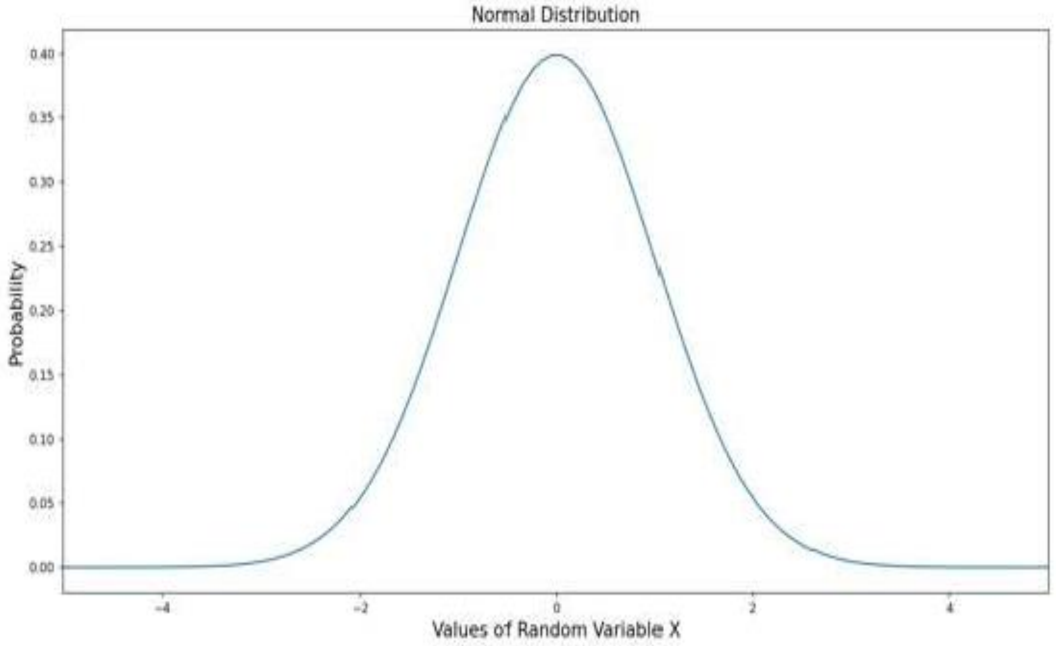
ALGORITHM:

Step1: Set the Mean as 0 and Standard Deviation as 1.
Step2: Generate the set x of 100 random numbers in the range of -5 to 5.
Step3: Define the probability density function using x.
Step4: Plot the Normal Distribution.

PROGRAM:

```
importnumpyasnp
importmatplotlib.pyplotasplt
fromscipyimport stats

# Create a standard normal distribution with mean as 0 and standard deviation as
1
#
mu = 0
std = 1
snd = stats.norm(mu, std)
#
# Generate 100 random values between -5, 5
#
x = np.linspace(-5, 5, 100)
#
# Plot the standard normal distribution for different values of random variable
# falling in the range -5, 5
#
plt.figure(figsize=(7.5,7.5))
plt.plot(x, snd.pdf(x))
plt.xlim(-5, 5)
plt.title('Normal Distribution', fontsize='15')
plt.xlabel('Values of Random Variable X', fontsize='15')
plt.ylabel('Probability', fontsize='15')
plt.show()
```

OUTPUT:



INFERENCE:

Normal distribution, also known as the Gaussian distribution, is a probability distribution that is symmetric about the mean, showing that data near the mean are more frequent in occurrence thandata far from the mean. In graphical form, the normal distribution appears as a "bell curve". From this program we learnt to draw a curve for normal distribution using matplotlib and numpy functions

RESULT: This program was successfully executed.

EX.NO: 14	**CORRELATION ANALYSIS**
DATE:	

AIM:

To write a python program to implement correlation analysis

ALGORITHM:

Step1: Compute the value of x̄ & ȳ.

Step 2: Compute $\sum_{n=1}^{n}(X - \bar{x})(Y - \bar{y})$

Step 3: Compute

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{\Sigma_j=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{\Sigma_j=1}^{n}(y_i - \bar{y})^2}}$$

Step 4: Find it is highly correlated or low correlated and display the result.

PROGRAM:

```
# Python Program to find correlation coefficient. import math

# function that returns correlation coefficient. def correlationCoefficient(X, Y, n) :sum_X = 0

    sum_Y = 0

    sum_XY = 0

    squareSum_X = 0

    squareSum_Y = 0

    i = 0

    while i<n :

    # sum of elements of array X. sum_X = sum_X + X[i]
# sum of elements of array Y. sum_Y = sum_Y + Y[i]


# sum of X[i] * Y[i].

sum_XY = sum_XY + X[i] * Y[i]


# sum of square of array elements. squareSum_X = squareSum_X + X[i] *X[i]
squareSum_Y = squareSum_Y + Y[i] * Y[i]

i = i + 1

# use formula for calculating correlation # coefficient.

corr =  (float)(n *  sum_XY -  sum_X *  sum_Y)/  ( float )(math.sqrt((n *squareSum_X

-sum_X * sum_X)* (n * squareSum_Y - sum_Y * sum_Y))) return corr


# Driver function

X = [15, 18, 21, 24, 27]

Y = [25, 25, 27, 31, 32]


print(X) print(Y)

# Find the size of array. n = len(X)


# Function call to correlationCoefficient. z = correlationCoefficient(X, Y, n)if(abs(z) > 0.5):

print ('{0:.6f}'.format(z), "Highly COrrelated") else:print('{0:.6f}'.format(z),"Low

Correlated")
```

OUTPUT:

```
[15, 18, 21, 24, 27]

[25, 25, 27, 31, 32]

0.953463 Highly Correlated
```

INFERENCE:

Correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). It's a common tool for describing simple relationships without making a statement about cause and effect. From this program we learnt about the correlation analysis technique using python

RESULT: This program was successfully executed.

EX.NO: 15	**MEAN,MEDIAN, MODE, STANDARD DEVIATION**
DATE:	

AIM:

To write a python program to implement mean, median, mode and standard deviation.

ALGORITHM:

Step1: Take a list of 8 Numbers.
Step2: Compute the Mean value by simple Computation and print it.Step3:
Compute the Mean value using numpy method and print it.
Step4: Compute the Median value by simple Computation and print it.Step5:
Compute the Mode value by simple Computation and print it. Step6:
Compute the Mode value using numpy method and print it.
Step7: Compute the Standard Deviation by simple Computation and print it.Step8:
Compute the Standard Deviation using Numpy and print it.

PROGRAM:

```
# Write a program to compute mean, median, mode and Standard Deviationimport
numpy as np
from collections import Counter
from scipy import stats
# Finding Mean by simple Computationa=
[11, 21, 34, 22, 27, 11, 23, 21]
mean = sum(a)/len(a)
print("Finding Mean by simple Computation")print
(mean)
# Finding Mean using numpy method
mean = np.mean(a)
print("Finding Mean using numpy method ")print
(mean)
#Finding Median by simple Computation.def
median(nums):
    nums.sort()
    if len(nums)%2 == 0:
        return int(nums[len(nums)//2-1]+nums[len(nums)//2])/2else:
        return nums[len(nums)//2]
print("Finding Median by simple Computation")print
(median(a))
print("Finding Median by numpy method")
print(np.median(a))
# Finding Mode by simple Computationdata
= dict(Counter(a))
mode = [k for k, v in data.items() if v == max(list(data.values()))]print("Finding
Mode by simple Computation ")
print (mode)
# Finding Mode using numpy method print("Finding
Mode using numpy method") print
(stats.mode(a,axis=None,keepdims=True)) # Find
Standard deviation by simple computationn=len(a)
std=(sum(map(lambda x: (x-sum(a)/n)**2,a))/n )**0.5
print(std)
# Find Standard deviation using numpy methodprint
(np.std(a))
```

OUTPUT:

```
Finding Mean by simple Computation
21.25
Finding Mean using numpy method
21.25
Finding Median by simple Computation21.5
Finding Median by numpy method
21.5
Finding Mode by simple Computation[11,
21]
Finding Mode using numpy method
ModeResult(mode=array([11]), count=array([2]))
7.1545440106270926
7.1545440106270926
```

INFERENCE:

Mean, median, mode and standard deviation are used for data analysis in data science. From this program we have learnt how to calculate Mean, median, mode and standard deviation using simple method and numpy method.

RESULT: This program was successfully executed

EX.NO: 14	**DATA VISUALIZATION**
DATE:	

AIM:

To write a python program to implement data visualization

ALGORITHM:

Step1: Load the IRIS Dataset and Wine Review DatasetStep
2: Create the Color Scatter Plot of IRIS Dataset.
Step 3: Create the Line chart for each attributes of IRIS Dataset.Step 4:
Create the Histogram for Wine Review Scores.
Step 5: Create the Bar Chart for Wine Review Scores.
Step 6: Create the multiple histogram for attributes of IRIS Dataset.
Step 7: Create the vertical bar chart for Wine Review Scores using plot.bar(). Step 8:
Create the horizontal bar chart for Wine Review Scores using plot.bar().
Step 9: Create the bar chart for Wine Review with highest cost five different Counties.

PROGRAM:

```
import pandas as pd import numpy as npimport
matplotlib.pyplot as plt
iris = pd.read_csv('iris.csv', names=['sepal_length', 'sepal_width', 'petal_length','petal_width', 'class'])
print(iris.head())
wine_reviews        =      pd.read_csv('winemag-data-130k-v2.csv',            index_col=0)
wine_reviews.head()


# Create Color Scatter Plotting
colors = {'Iris-setosa':'r', 'Iris-versicolor':'g', 'Iris-virginica':'b'} # create a figureand axis
fig, ax = plt.subplots() # plot each data-pointfor i in
range(len(iris['sepal_length'])):
```

```
ax.scatter(iris['sepal_length'][i], iris['sepal_width'][i],color=colors[iris['class'][i]])# set a title and
labels
ax.set_title('Iris Dataset') ax.set_xlabel('sepal_length')
ax.set_ylabel('sepal_width')plt.show()
```

```
# Create Line Chart Plotting columns = iris.columns.drop(['class']) # create x datax_data = range(0,
iris.shape[0]) # create figure and axis
fig, ax = plt.subplots() # plot each columnfor
column in columns:
ax.plot(x_data, iris[column], label=column) # set title and legendax.set_title('Iris
Dataset') ax.legend()
plt.show()
```

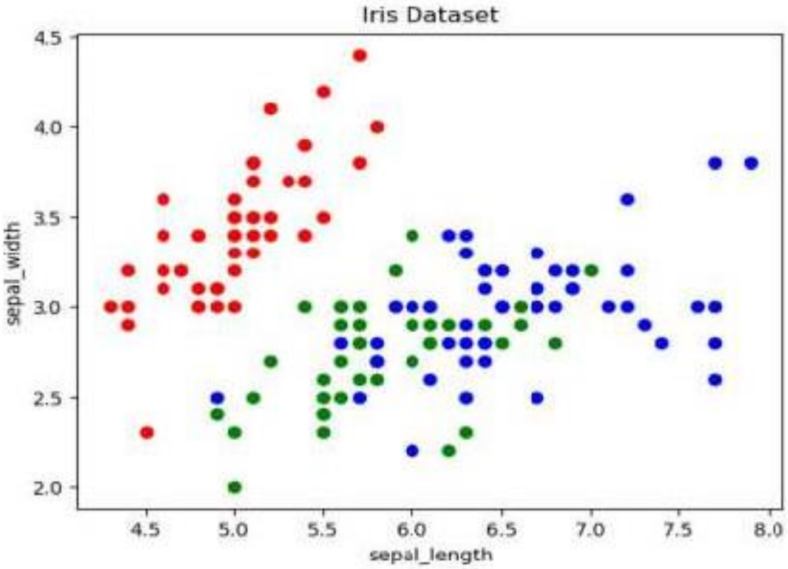
```
# create figure and axis fig, ax = plt.subplots() # plot histogramax.hist(wine_reviews['points']) # set title
and labels
ax.set_title('Wine Review Scores') ax.set_xlabel('Points')
ax.set_ylabel('Frequency') plt.show()
```

```
# create a figure and axis fig, ax = plt.subplots()# count
the occurrence of each class
data = wine_reviews['points'].value_counts() # get x and y data
points = data.index frequency = data.values # create bar chart ax.bar(points,frequency) # set title
and labels
ax.set_title('Wine Review Scores') ax.set_xlabel('Points')
ax.set_ylabel('Frequency') plt.show()
```

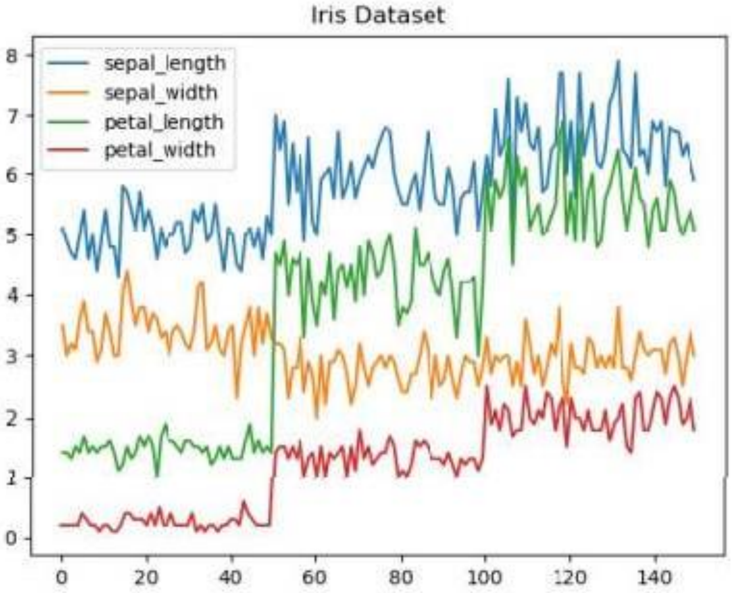
```
iris.plot.hist(subplots=True, layout=(2,2), figsize=(10, 10), bins=20) plt.show()

wine_reviews['points'].value_counts().sort_index().plot.bar() plt.show()
wine_reviews['points'].value_counts().sort_index().plot.barh() plt.show()

wine_reviews.groupby("country").price.mean().sort_values(ascending=False)[:5
].plo t.bar()
plt.show()

# Correlation Matrix corr = iris.corr() fig, ax =
plt.subplots() # create heatmapim =
ax.imshow(corr.values)

# set labels ax.set_xticks(np.arange(len(corr.columns)))
ax.set_yticks(np.arange(len(corr.columns))) ax.set_xticklabels(corr.columns)
ax.set_yticklabels(corr.columns)

# Rotate the tick labels and set their alignment. plt.setp(ax.get_xticklabels(),rotation=45,
ha="right",
rotation_mode="anchor")

# Loop over data dimensions and create text annotations. for i in
range(len(corr.columns)):
for j in range(len(corr.columns)):
text = ax.text(j, i, np.around(corr.iloc[i, j], decimals=2), ha="center",
va="center", color="black")
plt.show()
```

OUTPUT:

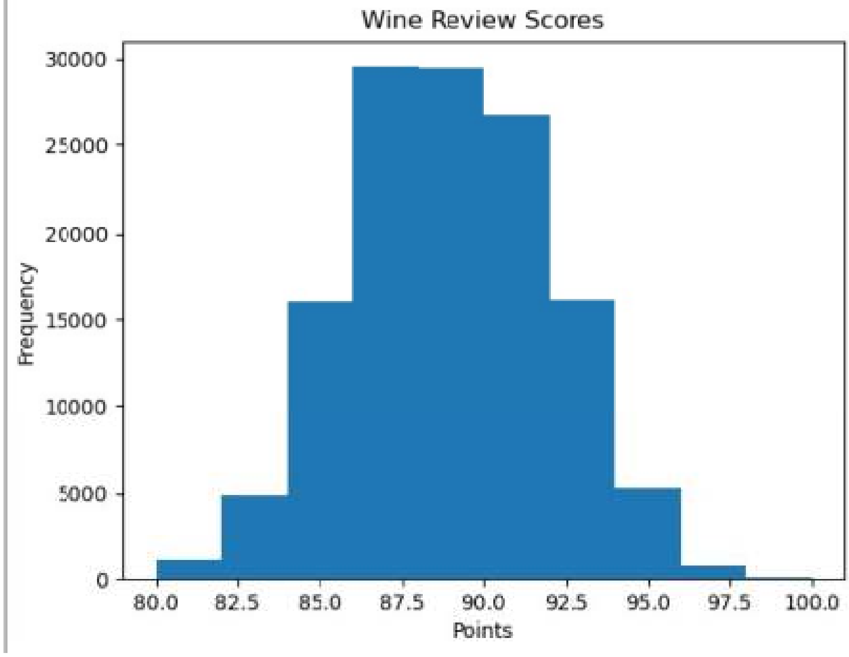
Scatter Plot of IRIS Dataset



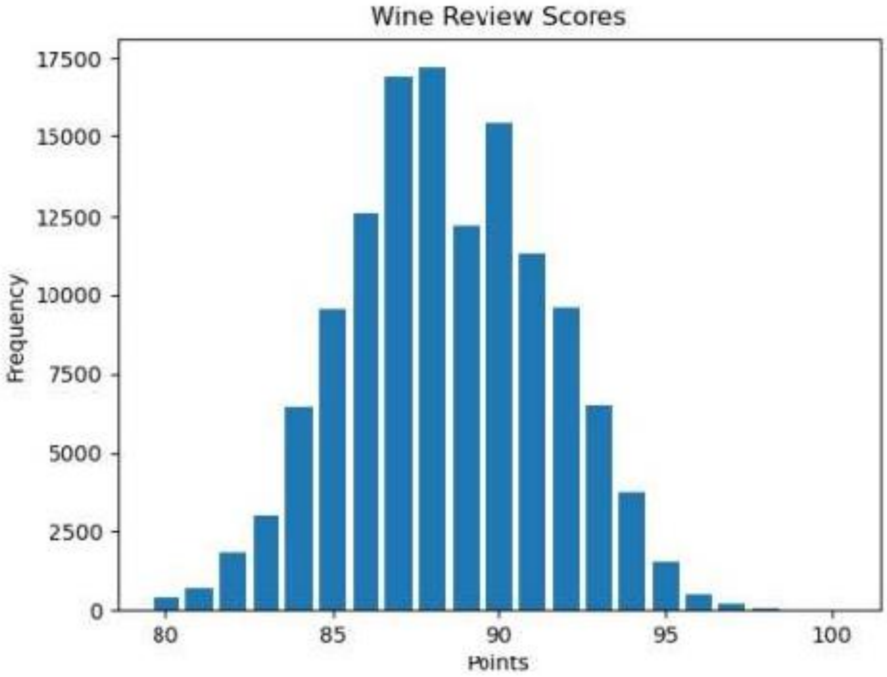
Line chart for each attribute of IRIS Dataset



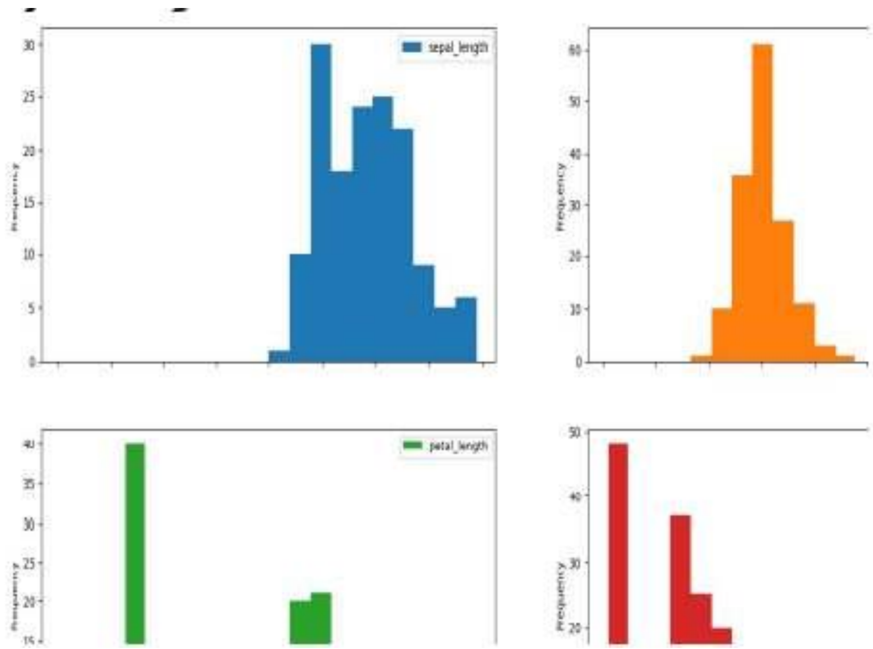
Histogram for Wine Review Scores.



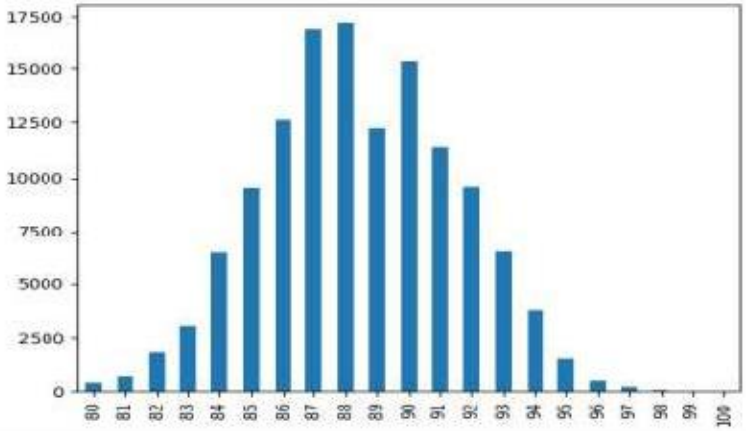
Bar Chart for Wine Review Scores



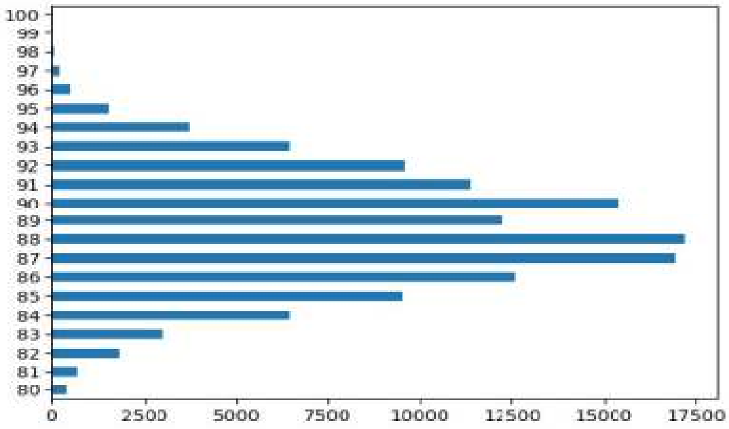
Multiple histogram for attributes of IRIS Dataset



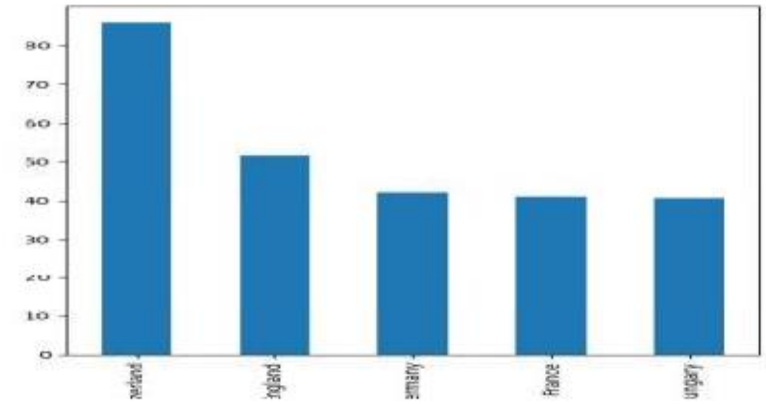
Vertical bar chart for Wine Review Scores



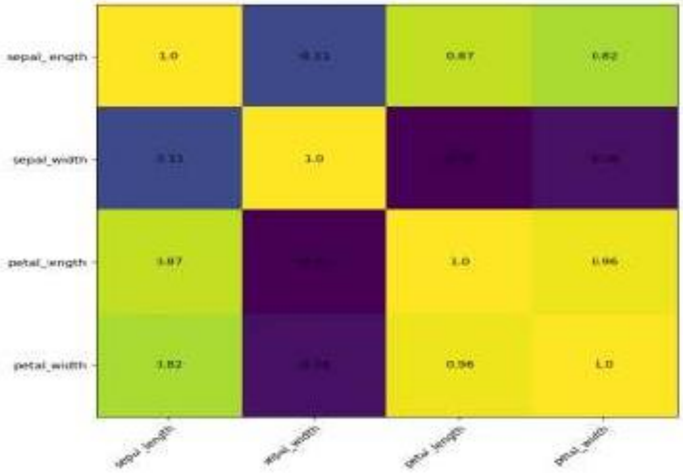
Horizontal bar chart for Wine Review Score



Bar chart for Wine Review with highest cost five different Counties.



Correlation Matrix



INFERENCE:

Data visualization is a way to represent information graphically, highlighting patterns and trends in data and helping the reader to achieve quick insights. From this program we learnt how to visualize data using python.

RESULT: This program was successfully executed.

CONTENT BEYOND SYLLABUS

EX.NO: 15	**PRINCIPAL COMPONENT ANALYSIS**
DATE:	

AIM:

To write a python Application Program to demonstrate the Principal Component Analysis.

ALGORITHM:

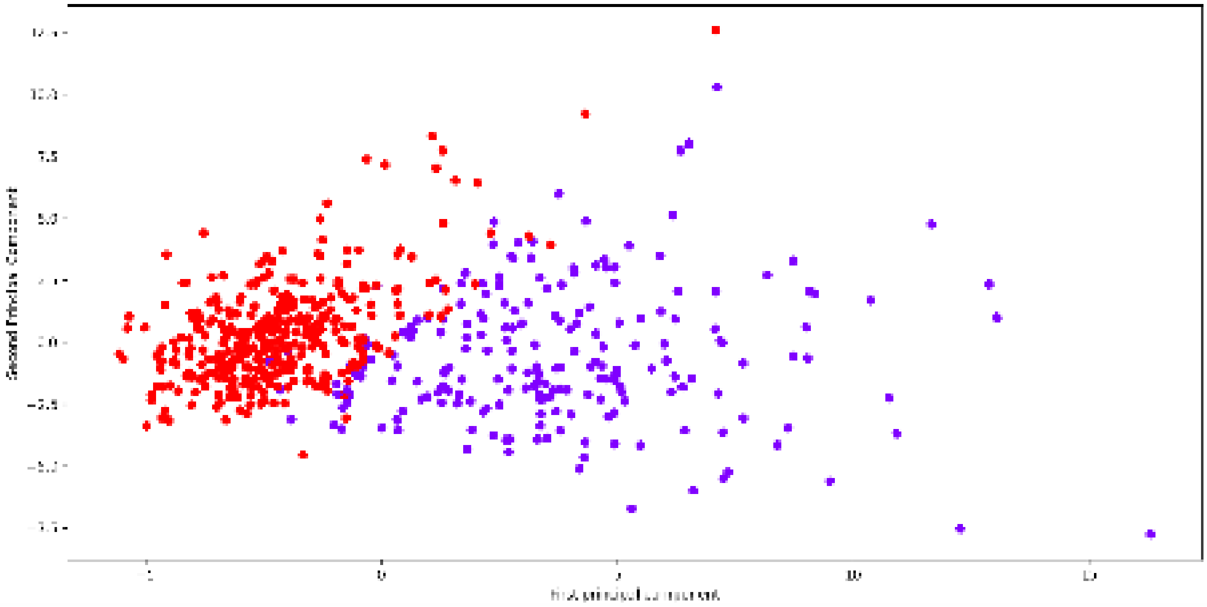
Step 1: Get data.
Step 2: Compute the mean vector (μ). Step 3:
Subtract mean from the given data.Step 4:
Calculate the covariance matrix.
Step 5: Calculate the eigen vectors and eigen values of the covariance matrix.Step
6: Choosing components and forming a feature vector.
Step 7: Deriving the new data set.

PROGRAM:

```
import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sns

from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()

cancer.keys()

df = pd.DataFrame(cancer['data'],columns=cancer['feature_names'])

df.head()

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
scaler.fit(df)

scaled_data = scaler.transform(df)

from sklearn.decomposition import PCA

pca = PCA(n_components=2)

pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
print(“Actual size”,scaled_data.shape)
print(“After PCA”,x_pca.shape)
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1],c=cancer['target'],cmap='rainbow') plt.xlabel('First
principal component')
plt.ylabel('Second Principal Component')
plt.show()
map= pd.DataFrame(pca.components_,columns=cancer['feature_names'])
plt.figure(figsize=(12,6))
sns.heatmap(map,cmap='twilight')plt.show()
```

OUTPUT:



INFERENCE:

Principal components analysis (PCA) is a dimensionality reduction technique that enables you toidentify correlations and patterns in a data set so that it can be transformed into a data set of significantly lower dimension without loss of any important information. From this program we learnt how to implement a PCA using python.

RESULT: This program was successfully executed.