Program 1: Installing Anaconda on Windows

1. Download the Anaconda installer.
2. RECOMMENDED: Verify data integrity with SHA-256. For more information on hashes, see what about cryptographic hash verification?
3. Double click the installer to launch.

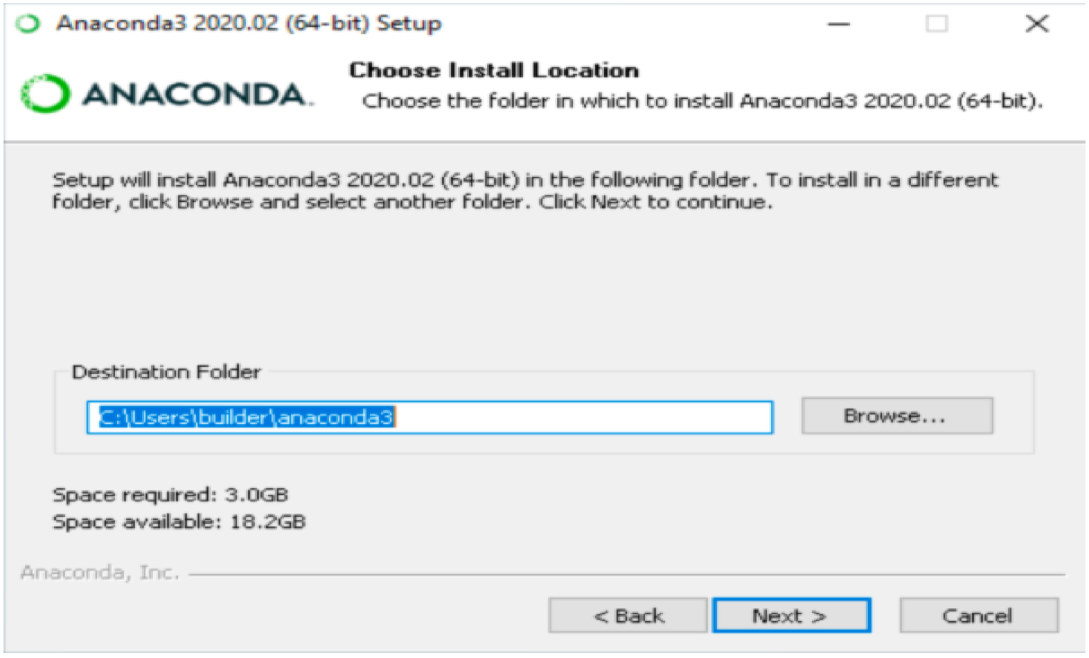
Note

a. To prevent permission errors, do not launch the installer from the Favorites folder.
b. If you encounter issues during installation, temporarily disable your anti-virus software during install, and then re-enable it after the installation concludes. If you installed for all users, uninstall Anaconda and re-install it for your user only and try again.

4. Click Next.
5. Read the licensing terms and click "I Agree".
6. Select an install for "Just Me" unless you're installing for all users (which require Windows Administrator privileges) and click Next.
7. Select a destination folder to install Anaconda and click the Next button

Note

c. Install Anaconda to a directory path that does not contain spaces or unicode characters.
d. Do not install as Administrator unless admin privileges are required.


Anaconda3 2020.02 (64-bit) Setup

Choose Install Location

Choose the folder in which to install Anaconda3 2020.02 (64-bit).

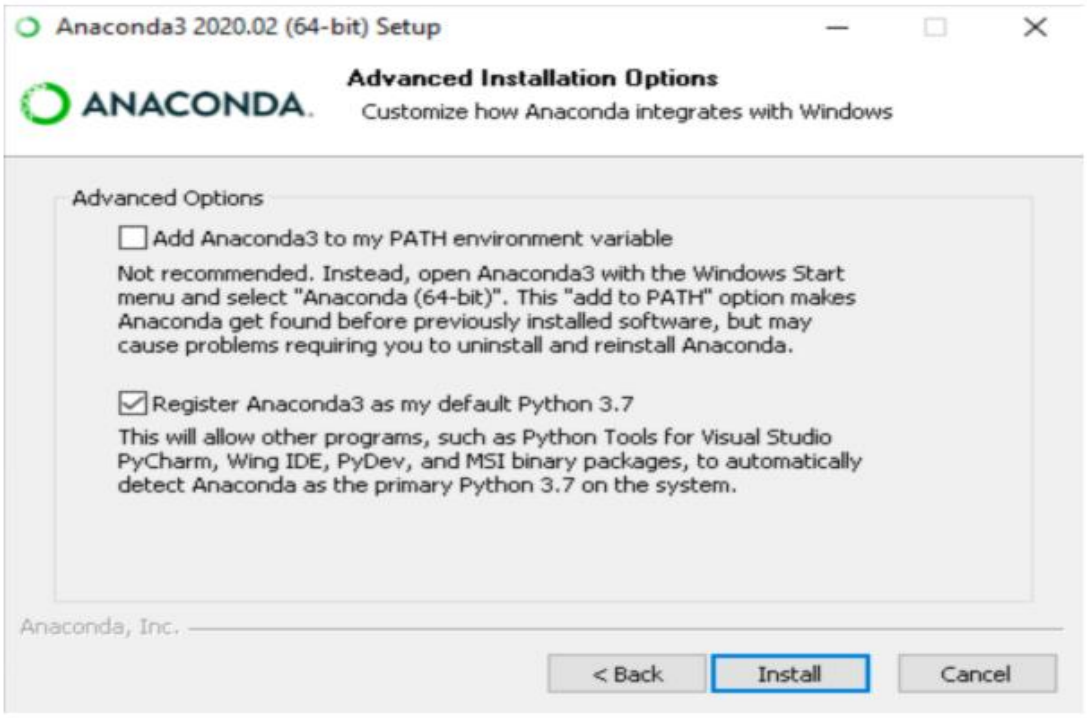
Setup will install Anaconda3 2020.02 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Users\builder\anaconda3

Space required: 3.0GB
Space available: 18.2GB

Anaconda, Inc.

8. Choose whether to add Anaconda to your PATH environment variable. We recommend not adding Anaconda to the PATH environment variable, since this can interfere with other software. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu.



Advanced Installation Options
Customize how Anaconda integrates with Windows

Advanced Options

☐ Add Anaconda3 to my PATH environment variable

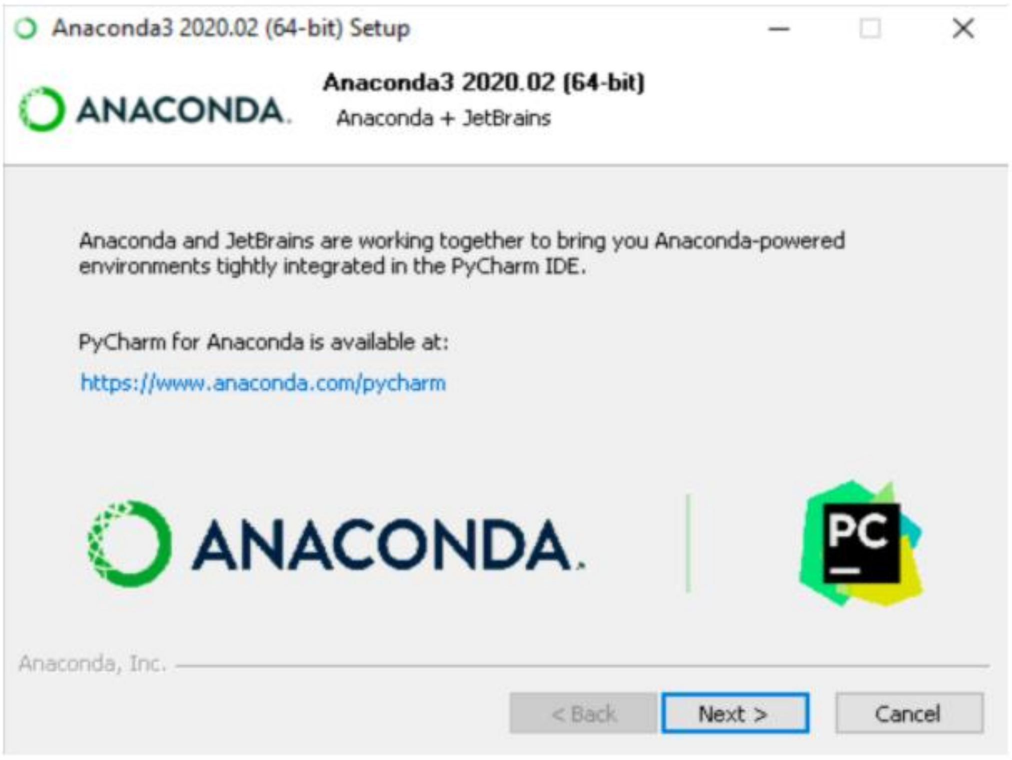
Not recommended. Instead, open Anaconda3 with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.

☑ Register Anaconda3 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.

9. Choose whether to register Anaconda as your default Python. Unless you plan on installing and running multiple versions of Anaconda or multiple versions of Python, accept the default and leave this box checked.
10. Click the Install button. If you want to watch the packages Anaconda is installing, click Show Details.
11. Click the Next button.
12. Optional: To install PyCharm for Anaconda, click on the link to https://www.anaconda.com/pycharm.



Anaconda3 2020.02 (64-bit)
Anaconda + JetBrains

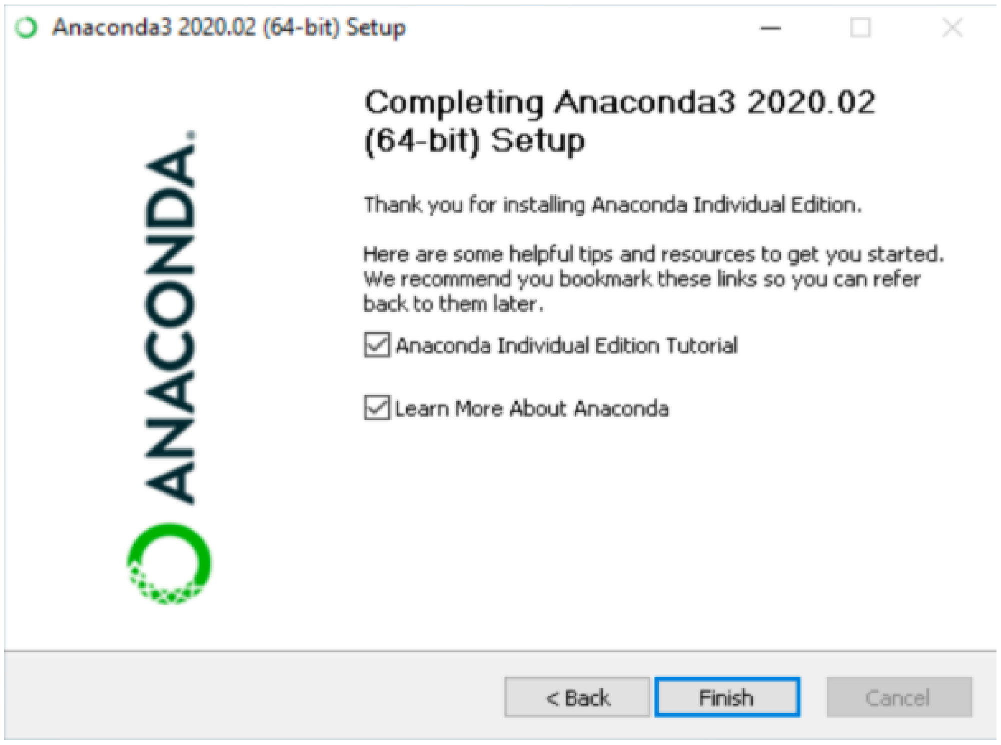
Anaconda and JetBrains are working together to bring you Anaconda-powered environments tightly integrated in the PyCharm IDE.

PyCharm for Anaconda is available at:

https://www.anaconda.com/pycharm

Anaconda, Inc.

13. After a successful installation you will see the “Thanks for installing Anaconda” dialog box:


Anaconda3 2020.02 (64-bit) Setup

Completing Anaconda3 2020.02 (64-bit) Setup

Thank you for installing Anaconda Individual Edition.

Here are some helpful tips and resources to get you started. We recommend you bookmark these links so you can refer back to them later.

☑ Anaconda Individual Edition Tutorial

☑ Learn More About Anaconda

14. If you wish to read more about Anaconda.org and how to get started with Anaconda, check the boxes “Anaconda Distribution Tutorial” and “Learn more about Anaconda”.

15. Click the Finish button.

Program 2: Introduction to usage of python 3 packages –
(i) numpy
Numpy-Ndarray:
NumPy is the fundamental package for scientific computing in Python. It stands for numerical python It is a Python library that provides a multidimensional array object, various derived objects and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. **Travis Oliphant** created NumPy package in 2005 by injecting the features of the ancestor module Numeric into another module Numarray.

It provides supports for large multi-dimensional array object and various tools to work with them. Various other libraries like pandas, matplotlib, scikit-learn are built top of this amazing library.
Arrays are the collection of elements/values that can have one (or) more dimensions. An array of one dimension is called vector and array of two dimensions is called a matrix. Numpy arrays are called ndarray (or) n-dimensional array. They store the elements of the same type and provide efficient storage.

The need of numpy:
Numpy provide a easy and efficient way to handle huge amount of data. Numpy is also very convenient with Matrix multiplication and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.

Advantages of numpy:
1. **Numpy performs array oriented programs.**
2. **It is efficient to implement the multi-dimensional arrays.**
3. **It performs scientific calculation.**
4. NumPy provides the in-built functions for linear algebra and random number generation.

NumPy in combination with SciPy and Matplotlib is used as the replacement to MATLAB as Python is more complete and easier programming language than MATLAB.

Creating of numpy array:
We can create a NumPy ndarray object by using the “array ()” function. To create an ndarray, we can pass a list, tuple or any array-like object into the array () method and it will be converted into an ndarray.

Basic ndarray:
```
import numpy as np
a=np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])
print (a)
```
Output:
```
[[1 2 3 4 5]
[6 7 8 9 10]
 [11 12 13 14 15]]
```
Array of zeros:
This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 0.
Syntax: numpy. zeros (shape)
```
import numpy as np
b=np.zeros(5)
print(b)
```
Output:
```
0 0 0 0 0]
```

Array of ones:
It is used to create the numpy array with the specified shape where each numpy array item is initialized to 1.
Syntax: np.ones(shape)

```
import numpy as np
d=np.ones(6)
print(d)
```

<u>Output:</u>
[1 1 1 1 1 1]

Array of random values:
The random is a module present in the NumPy library. This module contains the functions which are used for generating random numbers. This module contains some simple random data generation methods, some permutation and distribution functions, and random generator functions.
Syntax: np.random.rand(shape)

```
import numpy as np
e=np.random.rand(6)
print(e)
```

<u>Output:</u>

```
 [0.02905376  0.59423152  0.25030791  0.60751057  0.52254074  0.80428618]
```

Array of your choice:
To print the array elements of your choice we use full().

```
import numpy as np
f=np.full((3,3),7)
print(f)
```

<u>Output:</u>

```
[7 7 7 ]
[7 7 7]
[7 7 7]
```

Imatrix with numpy:
For printing identity matrix we use eye().
Syntax :np.eye(size)

```
import numpy as np
g=np.eye(4)
print(g)
```

<u>Output:</u>

```
 [1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```

Evenly spaced ndarray:
It creates an array by using the evenly spaced values over the given interval. The syntax to use the function is given below.
Syntax: np.arrange(start, stop, step, dtype)
It accepts the following parameters.

1. **start:** The starting of an interval. The default is 0.
2. **stop:** represents the value at which the interval ends excluding this value.
3. **step:** The number by which the interval values change.
4. **dtype:** the data type of the numpy array items.

```
import numpy as np
h = np.arange(0,10,2,float)
print(h)
```

Output:

```
[0 2 4 6 8]
```

dtype: Every ndarray has an associated data type (dtype) object. This data type object (dtype) informs us about the layout of the array.

```
import numpy as np
a=np.array([1,2,3])
print(a.dtype)
```

Output:

```
int 16
```

Shape of the given ndarray:

The shape of an array can be defined as the number of elements in each dimension. Dimension is the number of indices or subscripts that we require in order to specify an individual element of an array.

Syntax: *numpy.shape(array_name)*

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a.shape)
```

Output:

```
(2,3)
```

Dimension of the given array:

To find the dimensions of the given array we use ndim function. It will return the dimension.

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print (a.ndim)
```

Output:

```
2
```

(ii) pandas

Introduction for Pandas:

- "Pandas" is developed by **WES MCKINNEY** in 2008 and is used for data analysis. As for data analysis requires lots of processing like restructing, cleaning, etc. So we use pandas.
- In python, pandas are defined as an open source library which is used for high performance data manipulation and high level data structure.
- It contains high level data structures and manipulation tools, designed for fast and easy data analysis in python.
- Pandas were built on top of numpy and make it easy and more effective use for numpycentri application.

Introduction to Pandas Data Structure:

To get started with pandas, you need to be comfortable with two data structure.

1. Series
2. Data Frames

Series:

It is a one dimensional array, like object containing an "array of data" and it's associated with "array of data labels". The data labels are also called index. The given example is best way to create series.

Ex:
```
import pandas as pd
s=pd.Series([-2,5,7,9,23])
print(s)
```
Output:
```
0   -2
1    5
2    7
3    9
4   23
dtype: int64
```

We can also return the index and values of the series. While using the series to create the series we should give initial "s" as upper case.

```
Print(s.index)
```
Output:
```
RangeIndex(start=0, stop=5, step=1)
print (s.values)
```
Output:
```
array([-2,  5,  7,  9, 23], dtype=int64)
```

Often it will be desirable to create a Series with an index identifying each data point:

```
import pandas as pd
s=pd.Series ([-2, 5, 7, 9, 23], index= ['a','b','c','d','e'])
print(s)
```

Output:
```
a   -2
b    5
c    7
d    9
e   23
dtype: int64
```
Now we can the value of any index value:

```
print(s['d'])
print(s['a','b','c'])
```

Output:
```
9
a   -2
b    5
c    7
dtype: int64
```

Operator on pandas:
We can also perform all the arithmetic and comparison operation in this array. We can perform operation like scalar multiplication, or applying math functions.

Ex:
```
import pandas as pd
k=pd.Series([-7,6,5,-21,5,65])
print(k)
```

Output:

```
0    -7
1     6
2     5
3   -21
4     5
5    65
dtype: int64
```

```
print(k*k)
```

Output:

```
0      49
1      36
2      25
3     441
4      25
5    4225
dtype: int64
```

```
print(k*2)
```

Output:

```
0    -14
1     12
2     10
3    -42
4     10
5    130
dtype: int64
```

```
print(4 in k)
```

Output:

```
True
```

```
Print(7 in k)
```

Output:

```
False
```

Working with directories:

Pandas support the directories directly without converting (or) rewriting in series.

Ex:

```
data= {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
Obj=pd.Series(data)
Print(Obj)
```

Output:

```
Ohio      35000
Texas     71000
Oregon    16000
Utah       5000
dtype: int64
```

We can give your own indexing to the series. In pandas can check if the given values are null or not null

Ex:

```
data={'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
place=['a','Ohio','Texas','Oregon','Utah','b']
obj=pd.Series(data,index=place)
print(obj)
```

Output:

```
aNaN
Ohio      35000.0
Texas     71000.0
Oregon    16000.0
Utah       5000.0
bNaN
dtype: float64
```

```
a=pd.isnull(obj)
print(a)
```

Output:

```
a          True
Ohio      False
Texas     False
Oregon    False
Utah      False
b          True
dtype: bool
```

```
b=pd.notnull(obj)
print(b)
```

Output:

```
a         False
Ohio       True
Texas      True
Oregon     True
Utah       True
b         False
dtype: bool
```

Data Frames:

- A Data Frame represents a tabular, spreadsheet-like data structure containing an collection of columns, each of which can be a different value type (numeric, string, Boolean, etc.). The Data Frame has both a row and column index; it can be thought of as a dict of Series.
- The data frame has both rows and column index. The data is stored as two or more dimensional blocks rather than list, directories, ndarrays or some other collections of one dimensional arrays.
- Data Frames stores data internally in two dimensional format and we can easily represent much high dimensional data in tabular format with hierarchical indexing.

  ```
  import pandas as pd
  data={‘Year’:[2021,2020,2019,2018],’Month’:[‘Jan’,’Feb’,’Mar’,’Apr’]}
  df=pd.DataFrame(data)
  print(df)
  ```

Output:

```
   Year   Month
0  2021   Jan
1  2020   Feb
2  2019   Mar
3  2018   Apr
```

- There are number of ways to construct data frames, one of the most common form is dictionary.

```
data={'Year':[2021,2020,2019,2018],'Month':['Jan','Feb','Mar','Apr'],'day':['Mon','Tue','Wed','Thu']}
df=pd.DataFrame (data,columns=['Month','Day','Year'])
print(df)
```

Output:

```
   Month   Day   Year
0  Jan     NaN   2021
1  Feb     NaN   2020
2  Mar     NaN   2019
3  Apr     NaN   2018
```

- Whatever the output order it may be we can get our specified order of output for column.
- Same as series we can change the default index by adding index in the DataFrame().

```
data={'Year':[2021,2020,2019,2018],'Month':['Jan','Feb','Mar','Apr'],'day':['Mon','Tue','Wed','Thu']}
df=pd.DataFrame(data,columns=['Month','Day','Year'],index=['a','b','c','d'])
print(df)
```

Output:

```
   Month   Day   Year
a  Jan     NaN   2021
b  Feb     NaN   2020
c  Mar     NaN   2019
d  Apr     NaN   2018
```

- If you want to alter all the values of particular column with a unique value to the following set.

```
import pandas as pd
k=pd.DataFrame({'R.no':[4401,4402,4403,4404],'Sname':['ABC','DEF','GHI','JKL'],'Rank':[7,
6,3,9])
k[‘Rank’]=2
print(k)
```

<u>Output:</u>

	R.no	**Sname**	**Rank**
0	4401	ABC	7
1	4402	DEF	6
2	4403	GHI	3
3	4404	JKL	9

```
import pandas as pd
k=pd.DataFrame({'R.no':[4401,4402,4403,4404],'Sname':['ABC','DEF','GHI','JKL'],'Rank':[7,6,3,9]})
k['Rank']=2
print(k)
```

Output:

	R.no	**Sname**	**Rank**
0	4401	ABC	2
1	4402	DEF	2
2	4403	GHI	2
3	4404	JKL	2

- If you want give arange of values to all the record of a particular column. To the above set.

```
import numpy as np
import pandas as pd
k=pd.DataFrame({'R.no':[4401,4402,4403,4404],'Sname':['ABC','DEF','GHI','JKL'],'Rank':[7,
6,3,9]}
k['Rank']=np.arange(4)
print(k)
```

Output:

	R.no	**Sname**	**Rank**
0	4401	ABC	0
1	4402	DEF	1
2	4403	GHI	2
3	4404	JKL	3

- Pandas support working of data frames with series data structure.

```
import numpy as np
import pandas as pd
k=pd.DataFrame({'R.no':[4401,4402,4403,4404],'Sname':['ABC','DEF','GHI','JKL'],'Rank':[7,6,3,9]})
val=pd.Series([11,22],index=[0,3])
k['Rank']=val
print(k)
```

Output:

	R.no	**Sname**	**Rank**
0	4401	ABC	11.0
1	4402	DEF	NaN
2	4403	GHI	NaN
3	4404	JKL	22.0

- Possible data input to data frame constructor.
    1. 2D ndarray
    2. List,Tuple, Dictionary of arrays
    3. Numpy structured array
    4. dict(Series)
    5. dict(dicts)
    6. List(dict)
    7. List(Series)
    8. List(Lists)
    9. List(Tuple)
    10. Another data frame
    11. Numpy array

Working with altering index:
We can alter the index as whatever we want.

```
import pandas as pd
a=pd.Series([1,2,3],index=['a','b','c'])
print(a)
```

Output:

```
a    1
b    2
c    3
dtype: int64
```

```
import pandas as pd
a=pd.Series([1,2,3],index=['a','b','c'])
a.index=['x','y','z']
print(a)
```

Output:

```
x    1
y    2
z    3
dtype: int64
```

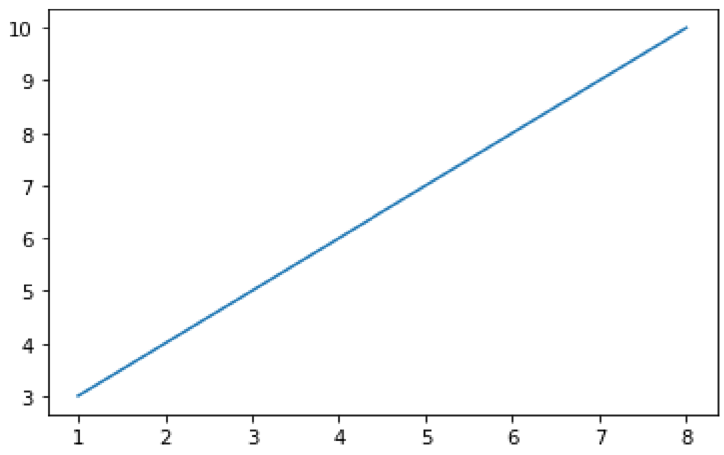
In this concept while working with altering index we cannot change the vales in the series.
a.index['x']=5

<u>Output:</u>
Index does not support mutable operations.

<u>(iii) matplotlib & (iv) plotly</u>

Introduction to Matplotlib:
 Matploplib is a low-level library of Python which is used for data visualization. It is easy to use and emulates MATLAB like graphs and visualization. This library is built on the top of NumPy arrays and consists of several plots like line chart, bar chart, histogram, etc.
 Matplotlib is originally written by Dr. John D Hunter.
 We need to install matplotlib in command prompt by using "pip install matplotlib" Command
 Pyplot is a subpackage of matplotlib.

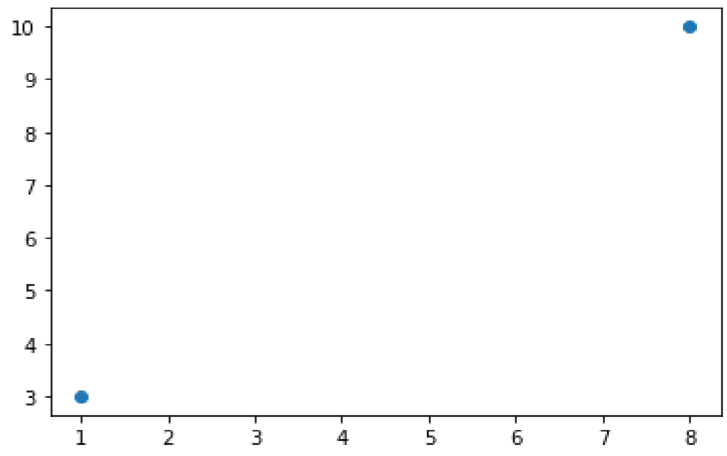
Example: 1
Draw a line in a diagram from position (1, 3) to position (8, 10):
```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints)
plt.show()
```

<u>OUTPUT:</u>


Plotting Without Line
To plot only the markers, you can use shortcut string notation parameter 'o', which means 'rings'.

Example: 2
Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])
plt.plot(xpoints, ypoints, 'o')
plt.show()
```

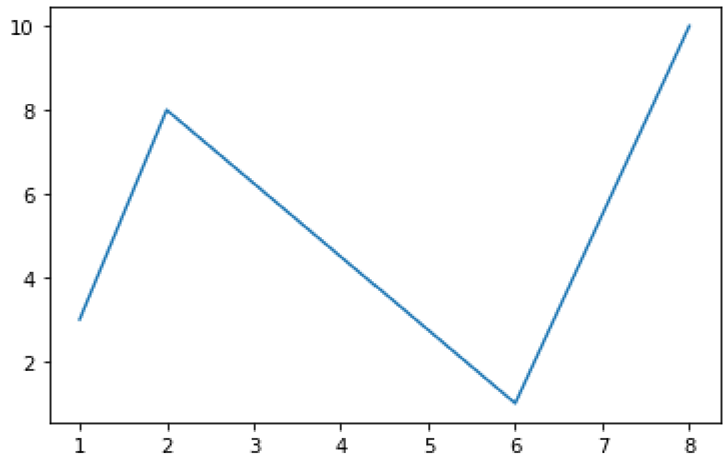
OUTPUT:



Example: 3
Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10)

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

xpoints = np.array([1, 2, 6, 8])
ypoints = np.array([3, 8, 1, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

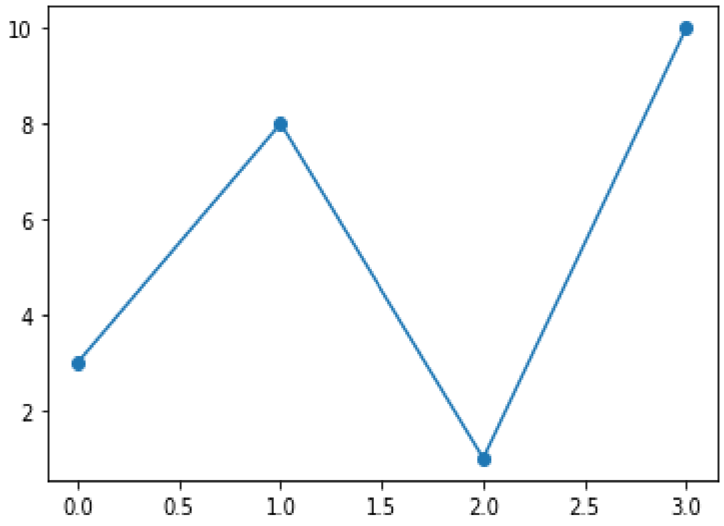
OUTPUT:



Example: 4
Mark each point with a circle:

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, marker = 'o')
plt.show()
```

OUTPUT:



(v) ggplot, ggplot2

Python has a number of powerful plotting libraries to choose from. One of the oldest and most popular is matplotlib - it forms the foundation for many other Python plotting libraries. For this exercise we are going to use plotnine which is a Python implementation of the The Grammar of Graphics, inspired by the interface of the ggplot2 package from R. plotnine (and it's R cousin ggplot2) is a very nice way to create publication quality plots.

Using ggplot in Python allows you to build visualizations incrementally, first focusing on your data and then adding and tuning components to improve its graphical representation.

The Grammar of Graphics
Statistical graphics is a mapping from data to aesthetic attributes (colour, shape, size) of geometric objects (points, lines, bars)
Faceting can be used to generate the same plot for different subsets of the dataset

These are basic building blocks according to the grammar of graphics:
data: The data + a set of aesthetic mappings that describing variables mapping
geom: Geometric objects, represent what you actually see on the plot: points, lines, polygons, etc.
stats: Statistical transformations, summarise data in many useful ways.
scale: The scales map values in the data space to values in an aesthetic space
coord: A coordinate system, describes how data coordinates are mapped to the plane of the graphic.
facet: A faceting specification describes how to break up the data into subsets for plotting individual set

First, install the pandas and plotnine packages to ensure they are available.
!pip install pandas plotnine

# We run this to suppress various deprecation warnings from plotnine - keeps our notebook cleaner
import warnings
warnings.filterwarnings('ignore')

Plotting in ggplot style
Let's set up our working environment with necessary libraries and also load our csv file into data frame called survs_df,

Program:

```
import numpy as np
import pandas as pd
from plotnine import *

%matplotlib inline
survs_df = pd.read_csv('surveys.csv').dropna()
ggplot(survs_df, aes(x='weight', y='hindfoot_length', size = 'year', color = 'species_id')) +
geom_point()
```

For reference: https://towardsdatascience.com/matplotlib-vs-ggplot2-c86dd35a9378.

(vi) seaborn

Seaborn is a library in Python predominantly used for making statistical graphics. Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data.

One has to be familiar with Numpy and Matplotlib and Pandas to learn about Seaborn.

Seaborn offers the following functionalities:

1. Dataset oriented API to determine the relationship between variables.
2. Automatic estimation and plotting of linear regression plots.
3. It supports high-level abstractions for multi-plot grids.
4. Visualizing univariate and bivariate distribution.

These are only some of the functionalities offered by Seaborn, there are many more

Using Seaborn we can plot wide varieties of plots like:

1. Distribution Plots
2. Pie Chart & Bar Chart
3. Scatter Plots
4. Pair Plots
5. Heat maps

Program 3: Demonstrate the usage of data sources, custom data view and extracting data fields operations

Python become a beloved language for text and files because of its simple syntax for interaction with files, data structure and it convenient features like list,tuples, series, data frames, packing and unpacking.

Here we are going to taking the support of CSV, TEXT, EXCEL, JSON files to directly creating the data set instead of creating the data sets by using data structures.

CSV FILE: CSV file is delimited text file that uses commas to separate the values

 Each line of the file is data record, each data record contains one or more fields separated by commas.
 CSV stores the data in tabular format as a plane text ,various software programs and applications can be import or export from the CSV file.
 To directly import the CSV file into python environment, we need to use a method called read_csv(). Similarly we can also use a method called read_table() method.
 Firstly we have to create a MS EXCEL file in csv file format to create a data set.
Examples:
(a) import pandas as pd
data = pd.read_csv(r'C:\Users\user\Desktop\kkpy\test.csv')
print(data)

OUTPUT:
```
CSE    DS  AI    IT know
0  501  4401   1  1201    a
1  502  4402   3  1202    b
2  503  4403   2  1203    c
3  504  4401   1  1204    d
4  505  4402   3  1205    e
5  506  4403   2  1206    f
6  507  4401   2  1207    g
7  508  4402   3  1208    h
8  509  4402   3  1209    i
9  510  4401   2  1210    j
```

- csv file format will never allow any beautifications to the current working file.

Now we will work with read_table.

(b) import pandas as pd
data = pd.read_table(r'F:\KK\enterprise-survey-2020.csv')
print(data)
OUTPUT:
```
CSE,DS,AI,IT,know
0  501,4401,1,1201,a
1  502,4402,3,1202,b
2  503,4403,2,1203,c
3  504,4401,1,1204,d
4  505,4402,3,1205,e
5  506,4403,2,1206,f
6  507,4401,2,1207,g
7  508,4402,3,1208,h
8  509,4402,3,1209,i
9  510,4401,2,1210,j
```

Program 4: Experimenting with data editing metadata, data joining and data blending

(a) Concatenating ndarrays(Data joining using numpy)

While stacking arrays is one way of combining old arrays to get a new one, you could also use the concatenate() method where the passed arrays are joined along an existing axis:

```
import numpy as np
a = np.arange(0,5).reshape(1,5)
b = np.arange(5,10).reshape(1,5)
print('Array 1 :','\n',a)
print('Array 2 :','\n',b)
print('Concatenate along rows :','\n',np.concatenate((a,b),axis=0))
print('Concatenate along columns :','\n',np.concatenate((a,b),axis=1))
```

<u>Output:</u>

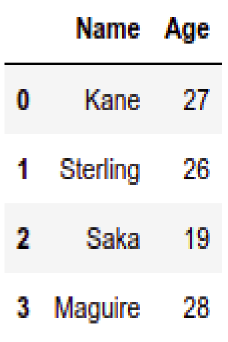
```
Array 1 :
 [[0 1 2 3 4]]
Array 2 :
 [[5 6 7 8 9]]
Concatenate along rows:
 [[0 1 2 3 4]
 [5 6 7 8 9]]
Concatenate along columns:
 [[0 1 2 3 4 5 6 7 8 9]]
```

The drawback of this method is that the original array must have the axis along which you want to combine. Otherwise, get ready to be greeted by an error.

(b) Concat Series and DataFrame (Data joining using pandas)

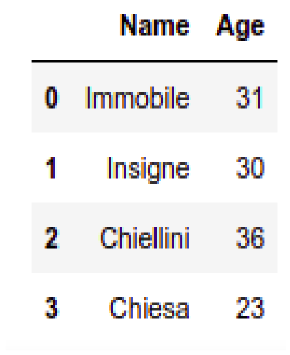
```
import pandas as pd
# Create team data
data_england = {'Name': ['Kane', 'Sterling', 'Saka', 'Maguire'], 'Age': [27, 26, 19, 28]}
data_italy = {'Name': ['Immobile', 'Insigne', 'Chiellini', 'Chiesa'], 'Age': [31, 30, 36, 23]}

# Create Dataframe
df_england = pd.DataFrame(data_england)
df_italy = pd.DataFrame(data_italy)
```

The England data frame looks something like this



	Name	Age
0	Kane	27
1	Sterling	26
2	Saka	19
3	Maguire	28

The Italy data frame looks something like this



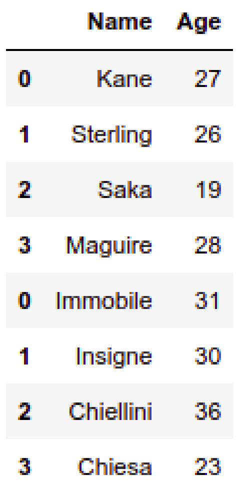
	Name	Age
0	Immobile	31
1	Insigne	30
2	Chiellini	36
3	Chiesa	23

Let's start by concatenating our two data frames. The word "concatenate" means to "link together in series". Now that we have created two data frames, let's try and "concat" them.

We do this by implementing the concat() function.

```
frames = [df_england, df_italy]
both_teams = pd.concat(frames)
both_teams
```

The result looks something like this:



	Name	Age
0	Kane	27
1	Sterling	26
2	Saka	19
3	Maguire	28
0	Immobile	31
1	Insigne	30
2	Chiellini	36
3	Chiesa	23

NOTE:

Data blending is a method for combining data from multiple sources. Data blending brings in additional information from a secondary data source and displays it with data from the primary data source directly in the view.

Data Blending allows a combination of data from different data sources to be linked. Whereas, Data Joining works only with data from one and the same source.

Program 5: Implementation of calculations with operator's, functions, and numeric calculations

Arithmetic Operators:

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)

print('First array:')
print (a )
print ('\n')

print ('Second array:' )
b = np.array([10,10,10])
print (b )
print ('\n')

print ('Add the two arrays:' )
print (np.add(a,b) )
print ('\n' )

print( 'Subtract the two arrays:' )
print (np.subtract(a,b) )
print ('\n' )

print ('Multiply the two arrays:' )
print( np.multiply(a,b))
print ('\n')

print ('Divide the two arrays:')
print (np.divide(a,b))
print ('\n')

print ('Applying power function again:')
print (np.power(a,b))
print ('\n')
```

OUTPUT:

```
First array:
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]


Second array:
[10 10 10]

Add the two arrays:
[[10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]

Subtract the two arrays:
[[-10.  -9.  -8.]
 [ -7.  -6.  -5.]
 [ -4.  -3.  -2.]]
```

```
Multiply the two arrays:
[[ 0. 10. 20.]
 [30. 40. 50.]
 [60. 70. 80.]]

Divide the two arrays:
[[0.  0.1 0.2]
 [0.3 0.4 0.5]
 [0.6 0.7 0.8]]

Applying power function again:
[[0.00000000e+00 1.00000000e+00 1.02400000e+03]
 [5.90490000e+04 1.04857600e+06 9.76562500e+06]
 [6.04661760e+07 2.82475249e+08 1.07374182e+09]]
```

Logical Operators:

```
print((True and True))
print((True and False))
print((False and True))
print((False and False))
print((True or True))
print((True or False))
print((False or True))
print((False or False))
```

OUTPUT:

```
True
False
False
False
True
True
True
False
```

Comparison Operators:

```
import numpy as np

x = np.array([0, 2, 3, 0, 1, 6, 5, 2])
print('Original Array = ', x)
print('\nGreater Than 0 = ', np.greater(x, 0))
print()

print('Greater Than 1 = ', np.greater(x, 1))
print()

print('Greater Than 2 = ', np.greater(x, 2))
```

OUTPUT:

```
Original Array = [0 2 3 0 1 6 5 2]
Greater Than 0 = [False  True  True False  True  True  True  True]
Greater Than 1 = [False  True  True False False  True  True  True]
Greater Than 2 = [False False  True False False  True  True False]
```

FUNCTIONS:

```
import numpy
 arr1 = numpy.arange(4)
print('Elements of an array1:\n',arr1)
arr2 = numpy.arange(4,8)
print('Elements of an array2:\n',arr2)
res1 = arr1.reshape(2,2)
print('Reshaped array with 2x2 dimensions:\n',res1)

res2 = arr2.reshape(2,2)
print('Reshaped array with 2x2 dimensions:\n',res2)

print("Concatenation two arrays:\n")
concat = numpy.concatenate((arr1,arr2),axis=1)
print(concat)
```

OUTPUT:

```
Elements of an array1:
 [0 1 2 3]
Elements of an array2:
 [4 5 6 7]
Reshaped array with 2x2 dimensions:
 [[0 1]
 [2 3]]
Reshaped array with 2x2 dimensions:
 [[4 5]
 [6 7]]
Concatenation two arrays:

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

FOR PRACTICE

(a) NumPy Arithmetic functions

The below mentioned NumPy functions are used to perform the basic arithmetic operations on the data values of an array–

numpy.add() function : It adds two arrays and returns the result.
numpy.subtract() function : Subtracts elements of array2 from array1 and returns the result.
numpy.multiply() function : Multiplies the elements of both the arrays and returns the product.
numpy.divide() function : Divides the array1 by array2 and returns the quotient of array values.
numpy.mod() function: Performs modulus operation and returns the remainder array.
numpy.power() function: Returns the exponential value of array1 ^ array2.

(b) NumPy Statistical functions

NumPy Statistical functions are very helpful in the domain of data mining and analysis of the huge amount of traits in the data.
numpy.median() : Calculates the median value of the passed array.
numpy.mean() : Returns the mean of the data values of the array.
numpy.average() : It returns the average of all the data values of the passed array.
numpy.std() : Calculates and returns the standard deviation of the data values of the array.

Program 6: Implementation of calculations with operations on string, date
(a) NumPy String functions
With NumPy String functions, we can manipulate the string values contained in an array. Some of the most frequently used String functions are mentioned below:

numpy.char.add() function: Concatenates data values of two arrays, merges them and represents a new array as a result.
numpy.char.capitalize() function: It capitalizes the first character of the entire word/string.
numpy.char.lower() function: Converts the case of the string characters to lower string.
numpy.char.upper() function: Converts the case of the string characters to upper string.
numpy.char.replace() function: Replaces a string or a portion of string with another string value.

(b) Date functions
```
import numpy as np

todays_date = np.datetime64('today', 'D')
print("Today's Date: ", todays_date)

yesterdays_date = np.datetime64('today', 'D') - np.timedelta64(1, 'D')
print("Yesterday's Date: ", yesterdays_date)

tomorrows_date = np.datetime64('today', 'D') + np.timedelta64(1, 'D')
print("Tomorrow's Date: ", tomorrows_date)
```

<u>OUTPUT:</u>
```
Today's Date: 2022-04-02
Yesterday's Date: 2022-04-01
Tomorrow's Date: 2022-04-03
```

For Reference:

	Here are the date units:			
S.No	**Code**	**Meaning**	**Time span (relative)**	**Time span (absolute)**
1	Y	year	+/- 9.2e18 years	[9.2e18 BC, 9.2e18 AD]
2	M	month	+/- 7.6e17 years	[7.6e17 BC, 7.6e17 AD]
3	W	week	+/- 1.7e17 years	[1.7e17 BC, 1.7e17 AD]
4	D	day	+/- 2.5e16 years	[2.5e16 BC, 2.5e16 AD]
	Here are the time units:			
S.No	**Code**	**Meaning**	**Time span (relative)**	**Time span (absolute)**
1	h	hour	+/- 1.0e15 years	[1.0e15 BC, 1.0e15 AD]
2	m	minute	+/- 1.7e13 years	[1.7e13 BC, 1.7e13 AD]
3	s	second	+/- 2.9e11 years	[2.9e11 BC, 2.9e11 AD]
4	ms	millisecond	+/- 2.9e8 years	[2.9e8 BC, 2.9e8 AD]
5	us / µs	microsecond	+/- 2.9e5 years	[290301 BC, 294241 AD]
6	ns	nanosecond	+/- 292 years	[1678 AD, 2262 AD]
7	ps	picosecond	+/- 106 days	[1969 AD, 1970 AD]
8	fs	femtosecond	+/- 2.6 hours	[1969 AD, 1970 AD]
9	as	attosecond	+/- 9.2 seconds	[1969 AD, 1970 AD]

Program 7: Experiment to working with sorting and filtering operations

Numpy Sort

Sorting means putting elements in an ordered sequence.
Ordered sequence means sequencing in any order corresponding to elements like numeric or alpha; ascending or descending.
Numpy has a function called sort(), that will sort a specified array.

(a)
```
import numpy as np
a=np.array([3,2,0,1])
print("Original array:", a)
print("Sorted array:", np.sort(a))
```

OUTPUT:
```
Original array: [3 2 0 1]
Sorted array: [0 1 2 3]
```

This method returns the copy, leaving the original unchanged.
We can also sort array that has string or any other datatype.

(b)
```
a=np.array(['Banana', 'Cherry', 'Apple'])
print("Original array:", a)
print("Sorted array:", np.sort(a))
```

OUTPUT:
```
Original array: ['Banana'  'Cherry'  'Apple']
Sorted array: ['Apple' 'Banana'  'Cherry']
```

(c)
```
a=np.array(['False', 'True', 'False'])
print("Original array:", a)
print("Sorted array:", np.sort(a))
```

OUTPUT:
```
Original array: ['False'  'True'  'False']
Sorted array: ['False' 'False'  'True']
```

(d)
```
import numpy as np

# sort along the first axis
a = np.array([[12, 15], [10, 1]])
arr1 = np.sort(a, axis = 0)
print ("Along first axis : \n", arr1)


# sort along the last axis
a = np.array([[10, 15], [12, 1]])
arr2 = np.sort(a, axis = -1)
print ("\nAlong first axis : \n", arr2)


a = np.array([[12, 15], [10, 1]])
arr1 = np.sort(a, axis = None)
print ("\nAlong none axis : \n", arr1)
```

OUTPUT:

```
Along first axis:
 [[10  1]
 [12 15]]

Along first axis:
 [[10 15]
 [ 1 12]]

Along none axis:
 [ 1 10 12 15]
```

Filtering Arrays

Getting some elements out of an existing array and creating a new array out of them is called filtering. In NumPy, you filter an array using a boolean index list.

A boolean index list is a list of booleans corresponding to indexes in the array.
If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.

(a)
```
import numpy as np
arr = np.array([41, 42, 43, 44])
x = [True, False, True, False]
newarr = arr[x]
print(newarr)
```

OUTPUT:

```
[41 43]
```

(b)
```
import numpy as np

arr = np.array([41, 42, 43, 44])
# Create an empty list
filter_arr = []
# go through each element in arr

for element in arr:
  # if the element is higher than 42, set the value to True, otherwise False:
  if element > 42:
    filter_arr.append(True)
  else:
    filter_arr.append(False)

newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

OUTPUT:

```
[False, False, True, True]

[43 44]
```

(c) import numpy as np

```
arr = np.array([41, 42, 43, 44])
filter_arr = arr > 42
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

OUTPUT:

```
[False, False, True, True]
[43 44]
```

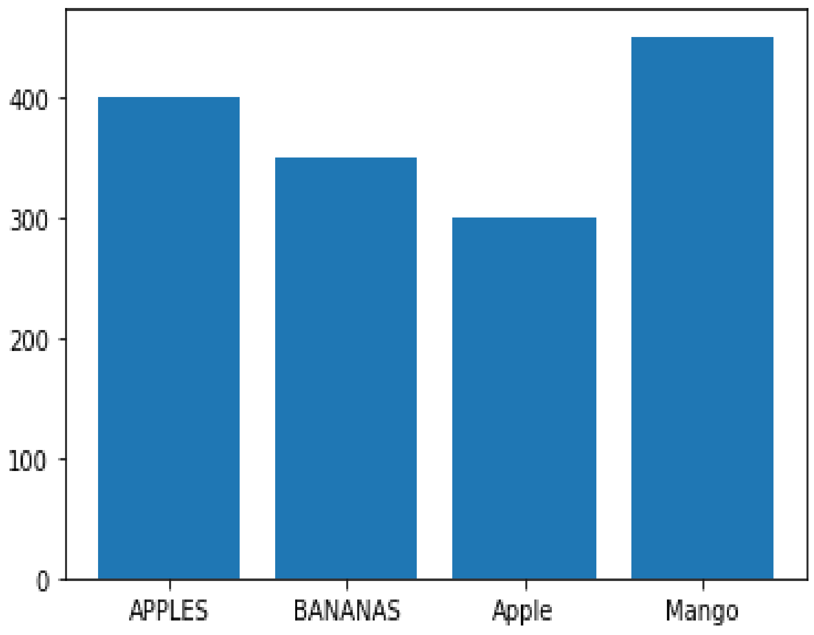
Program 8: Experiment to demonstrate data visualization with charts: bar chart, line chart and pie chart

Creating Bars Chart:

With Pyplot, you can use the bar() function to draw bar graphs:

(a)
```
import matplotlib.pyplot as plt
import numpy as np

x = ["APPLES", "BANANAS","Apple", "Mango"]
y = [400, 350, 300, 450]

plt.bar(x, y, color = "red", width = 0.2)
plt.show()
```

OUTPUT:

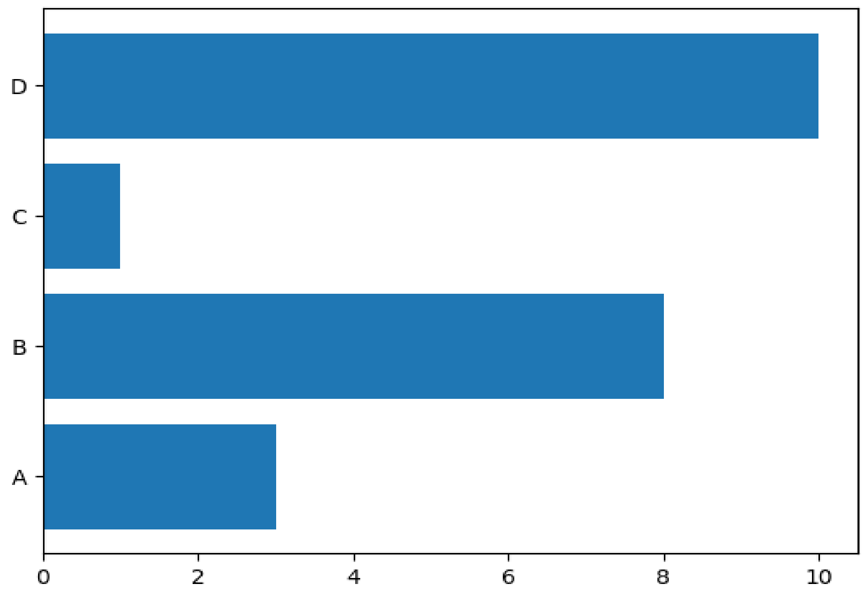


(b) **Horizontal Bars**

If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, color = "green", width = 0.2)
plt.show()
```

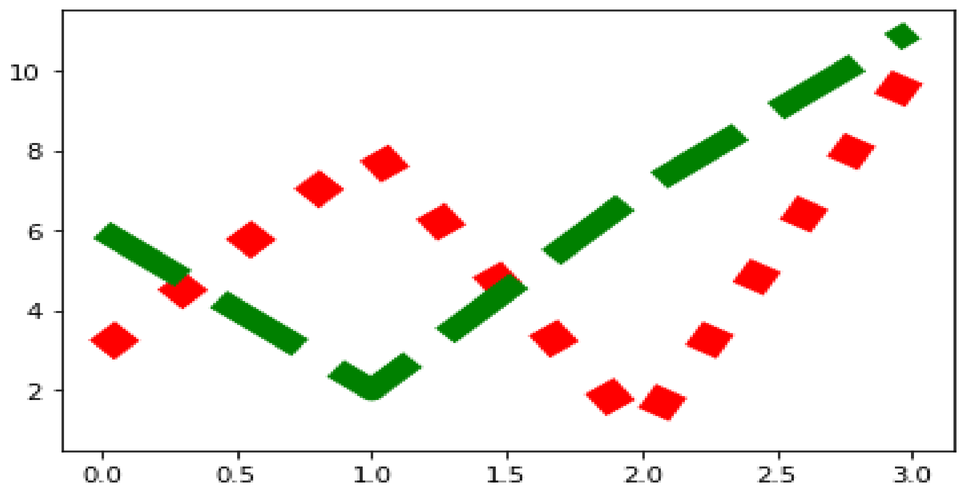
OUTPUT:



Creating Line Chart:
Linestyle
You can use the keyword argument linestyle, or shorter ls, to change the style of the plotted line:

```
(a) import matplotlib.pyplot as plt
import numpy as np
x1 =np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
x2 = np.array([0, 1, 2, 3])
y2 = np.array([6, 2, 7, 11])

plt.plot(x1, y1, linestyle = 'dotted', color='red', linewidth = '13.5')
plt.plot(x2, y2, linestyle = '--' , color='green', linewidth = '10.5')
plt.show()
```

OUTPUT:

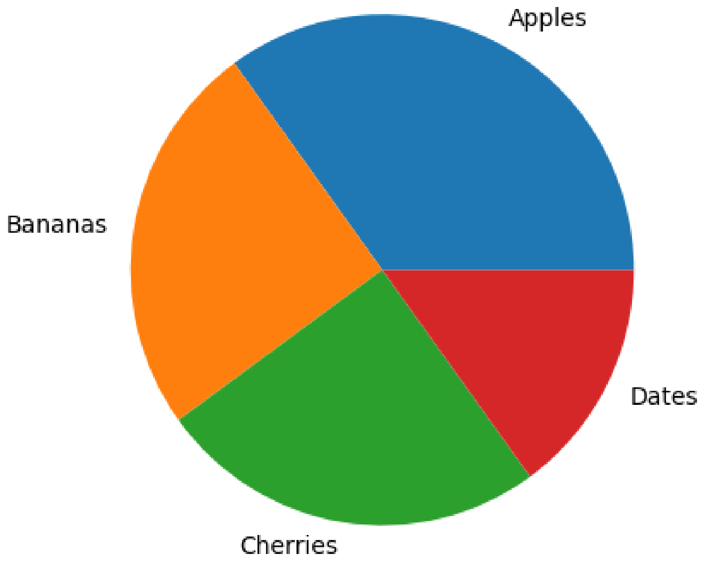


Creating Pie Chart:
With Pyplot, you can use the pie() function to draw pie charts:
Add labels to the pie chart with the label parameter.

The label parameter must be an array with one label for each wedge:

(a) import matplotlib.pyplot as plt
```
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

OUTPUT:

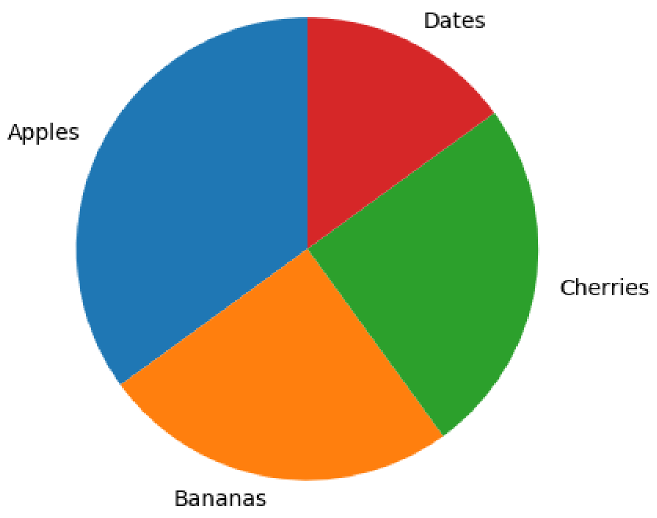


(b) Start Angle
As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.

The startangle parameter is defined with an angle in degrees, default angle is 0:

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```

OUTPUT:



Program 9: Experiment to demonstrate data visualization with charts: crosstab, scatter plot and bubble chart.

(a) Creating Crosstab:

Using pandas crosstab to create a bar plot

First Lets us know more about the crosstab, It is a simple cross-tabulation of two or more variables.

What is cross-tabulation?

It is a simple cross-tabulation that help us to understand the relationship between two or more variable. It will give a clear understanding of the data and makes analysis easier.

Let us take an example if we take a data set of Handedness of people which includes peoples nationality, sex, age, and name. Suppose if we want to analyze the relationship between nationality and handedness of the peoples. Crosstab gives you the relationship between them.

Crosstab using pandas

Before creating the barplot we should create cross-tabulation using pandas.

Syntax: pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, margins_name='All', dropna=True, normalize=False

(a) import pandas as pd
```
# Reading the csv file and storing it in a variable
df = pd.read_csv(r 'C:\Users\user\Desktop\kkpy\test.csv')

# Crosstab function is called 2 parameters are passed
# The table is stored in a variable
crosstb = pd.crosstab(df.DS, df.AI)
print(crosstb)
```

OUTPUT:

```
AI     1  2  3
DS
4401   2  2  0
4402   0  0  4
4403   0  2  0
```

Creating bar plots using crosstab:

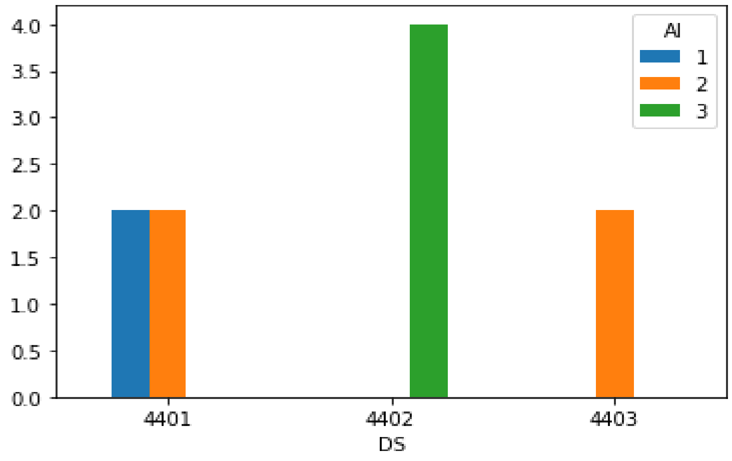
Bar graphs are most used to compare between different groups or to track changes over time. Using bar plots to the crosstab is one of the efficient ways to conclude the crosstab and analyze them even better.

Syntax: DataFrame.plot.bar(x=None, y=None, **kwargs)

(b) import pandas as pd
```
# Reading the csv file and storing it in a variable
df = pd.read_csv(r 'C:\Users\user\Desktop\kkpy\test.csv')

# Crosstab function is called 2 parameters are passed
# The table is stored in a variable
crosstb = pd.crosstab(df.DS, df.AI)
print(crosstb)
barplot = crosstb.plot.bar(rot=0)
```

OUTPUT:

```
AI     1   2   3
DS
4401   2   2   0
4402   0   0   4
4403   0   2   0
```

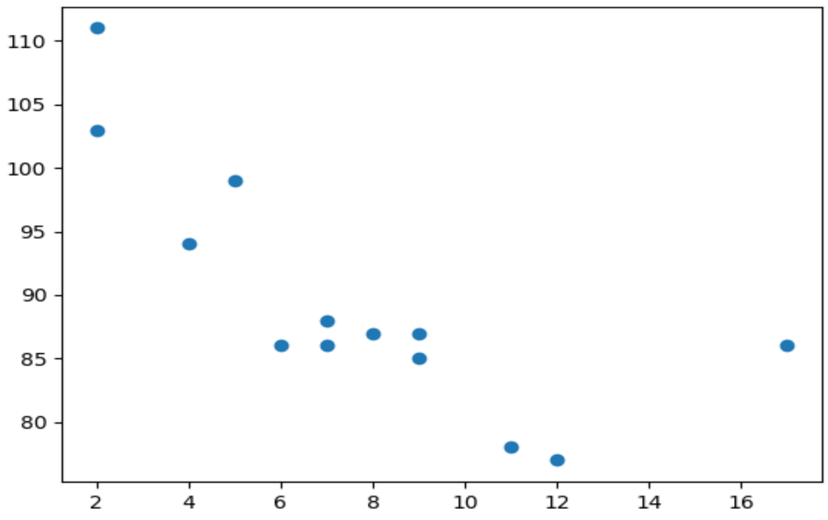


(b) Creating scatter plot
With Pyplot, you can use the scatter() function to draw a scatter plot.

The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

(a) import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.show()

OUTPUT:



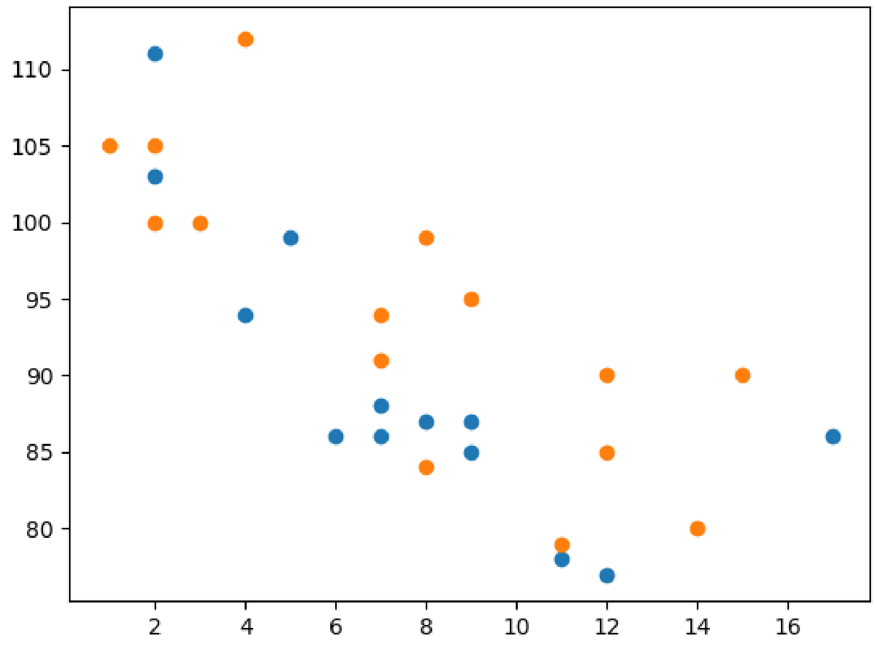
The observation in the example above is the result of 13 cars passing by.

The X-axis shows how old the car is.
The Y-axis shows the speed of the car when it passes.
Are there any relationships between the observations?
It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Compare Plots:
In the example above, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

```
(b) import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])

plt.scatter(x, y)
plt.show()
```

<u>OUTPUT:</u>



(c) Creating bubble chart.

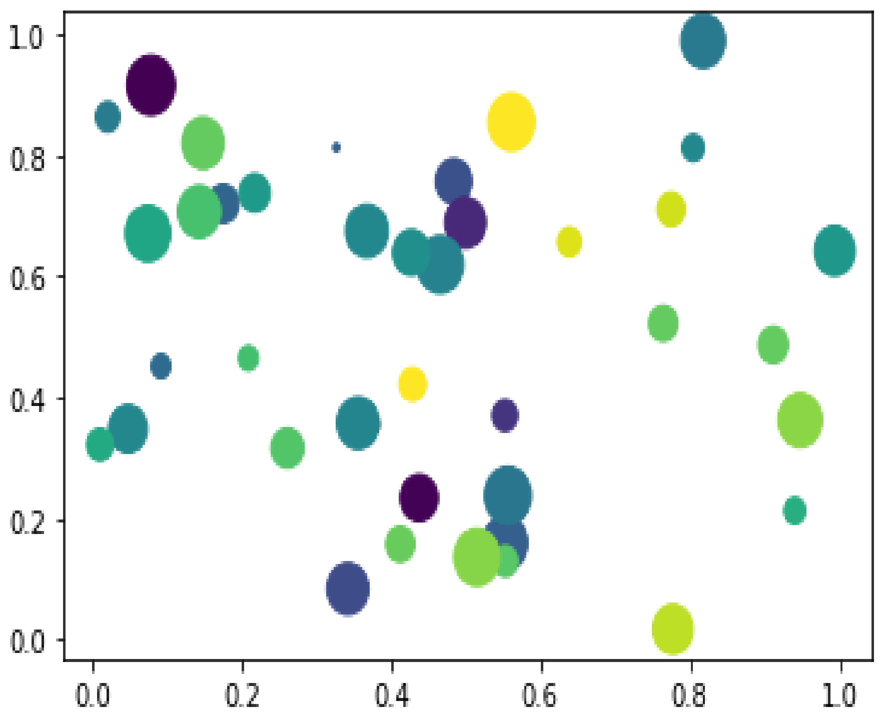
The bubble chart in Plotly is created using the scatter plot. It can be created using the scatter() method of plotly.express. A bubble chart is a data visualization which helps to displays multiple circles (bubbles) in a two-dimensional plot as same in scatter plot. .

A bubble chart is primarily used to depict and show relationships between numeric variables.

NOTE: Bubble chart can be created using the DataFrame.plot.scatter() methods.

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

# create data
x = np.random.rand(40)
y = np.random.rand(40)
z = np.random.rand(40)
colors = np.random.rand(40)
# use the scatter function
plt.scatter(x, y, s=z*400,c=colors)
plt.show()
```

<u>OUTPUT:</u>



Program 10: Experiment to demonstrate data visualization with charts: bullet graph, box plot and tree map / heat map.

(a) bullet graph
Plotly library of Python can be very useful for data visualization and understanding the data simply and easily.

plotly.figure_factory.create_bullet
This method is used to create bullet charts. This function can take both dataframes or a sequence of dictionaries.
Syntax: plotly.figure_factory.create_bullet(data, markers=None, measures=None, ranges=None, subtitles=None, titles=None, orientation='h', **layout_options)

Parameters:

data: either a list/tuple of dictionaries or a pandas DataFrame.

markers: the column name or dictionary key for the markers in each subplot.

measures: This bar usually represents the quantitative measure of performance, usually a list of two values [a, b] and are the blue bars in the foreground of each subplot by default.

ranges: This parameter is usually a 3-item list [bad, okay, good]. They correspond to the grey bars in the background of each chart.

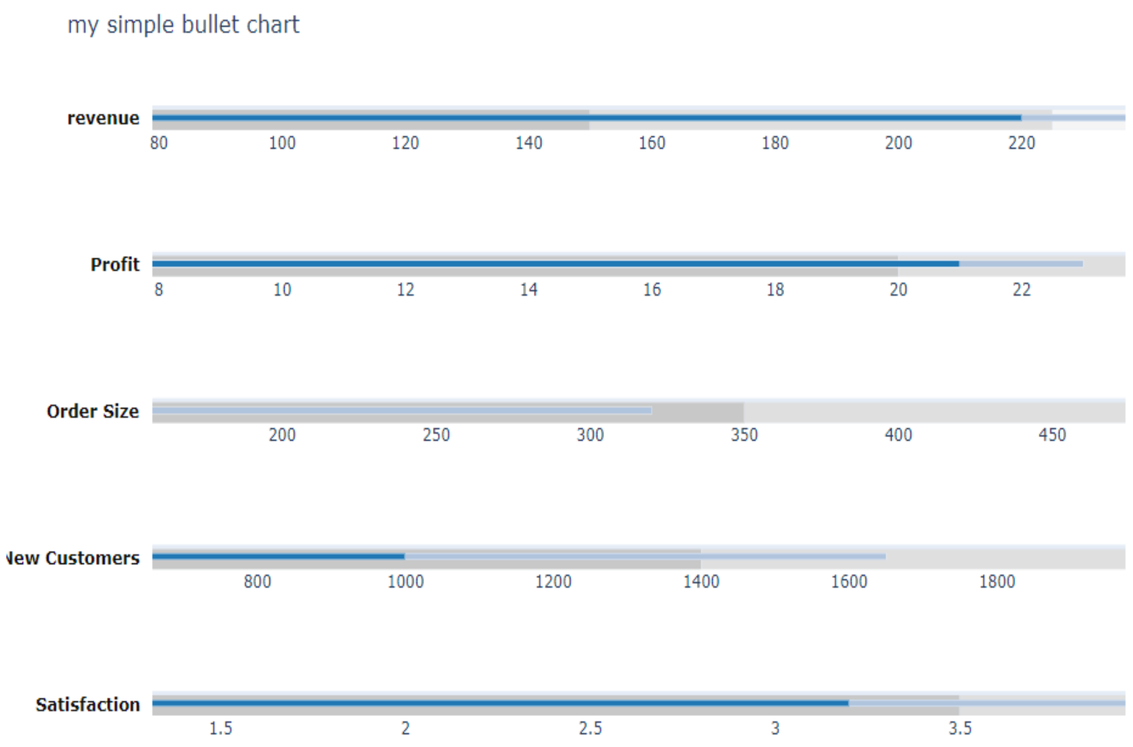
subtitles: the column name or dictionary key for the subtitle of each subplot chart.

titles ((str)) – the column name or dictionary key for the main label of each subplot chart.

<u>Program:</u>

```
import plotly.figure_factory as ff

data = [

  {"label": "revenue",
   "sublabel": "us$, in thousands",
   "range": [150, 225, 300],
   "performance": [220,270],
   "point": [250]},

  {"label": "Profit",
   "sublabel": "%",
   "range": [20, 25, 30],
   "performance": [21, 23],
   "point": [26]},


  {"label": "Order Size",
   "sublabel":"US$, average",
   "range": [350, 500, 600],
   "performance": [100,320],
   "point": [550]},
  {"label": "New Customers",
   "sublabel": "count",
   "range": [1400, 2000, 2500],
   "performance": [1000, 1650],
   "point": [2100]},


  {"label": "Satisfaction",
   "sublabel": "out of 5",
   "range": [3.5, 4.25, 5],
   "performance": [3.2, 4.7],
   "point": [4.4]}
]

fig = ff.create_bullet(
    data, titles='label',
    subtitles='sublabel',
    markers='point',
    measures='performance',
    ranges='range',
    orientation='h',
    title='my simple bullet chart'
)

fig.show()
```

OUTPUT:



(b) Box plot

A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.

The matplotlib.pyplot module of matplotlib library provides boxplot() function with the help of which we can create box plots.

Syntax: matplotlib.pyplot.boxplot(data, notch=None, vert=None, patch_artist=None, widths=None)

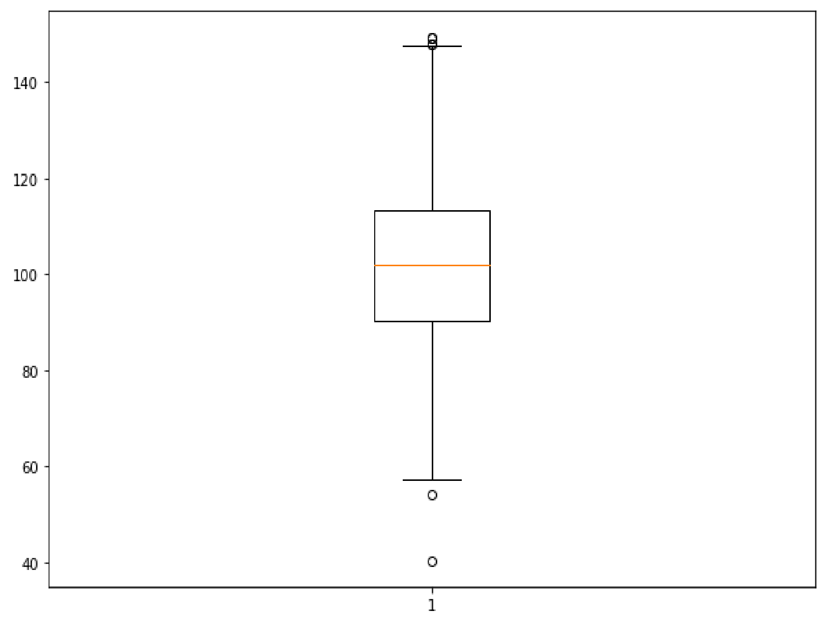
Parameters:

Attribute	Value
data	array or sequence of array to be plotted
notch	optional parameter accepts boolean values
vert	optional parameter accepts boolean values false and true for horizontal and vertical plot respectively
bootstrap	optional parameter accepts int specifies intervals around notched boxplots
usermedians	optional parameter accepts array or sequence of array dimension compatible with data
positions	optional parameter accepts array and sets the position of boxes
widths	optional parameter accepts array and sets the width of boxes
patch_artist	optional parameter having boolean values
labels	sequence of strings sets label for each dataset
meanline	optional having boolean value try to render meanline as full width of box
order	optional parameter sets the order of the boxplot

The data values given to the ax.boxplot() method can be a Numpy array or Python list or Tuple of arrays. Let us create the box plot by using numpy.random.normal() to create some random data, it takes mean, standard deviation, and the desired number of values as arguments.

```
(a) import matplotlib.pyplot as plt
import numpy as np


# Creating dataset
np.random.seed(10)
data = np.random.normal(100, 20, 200)

fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(data)

# show plot
plt.show()
```

OUTPUT:

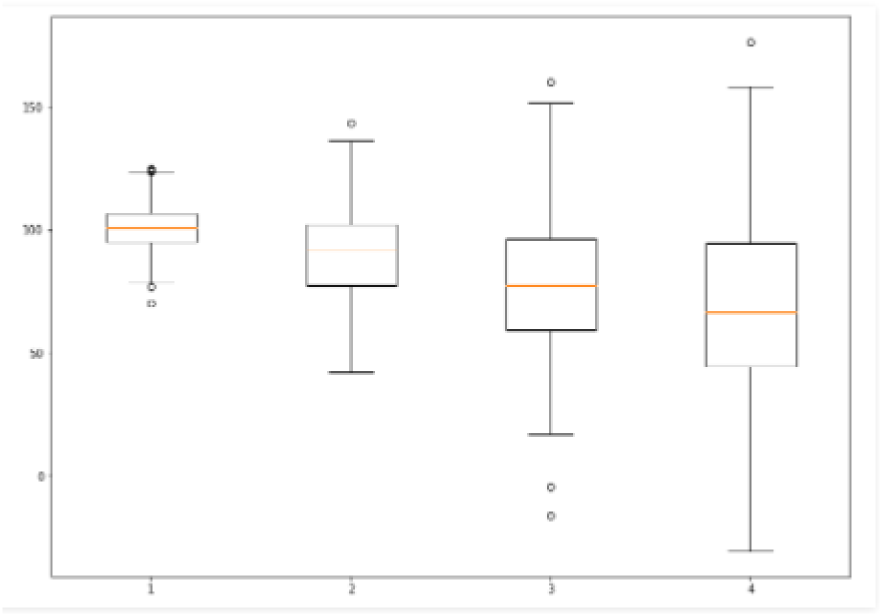


Customizing Box Plot

The matplotlib.pyplot.boxplot() provides endless customization possibilities to the box plot. The notch = True attribute creates the notch format to the box plot, patch_artist = True fills the boxplot with colors, we can set different colors to different boxes.The vert = 0 attribute creates horizontal box plot. labels takes same dimensions as the number data sets.

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


# Creating dataset
np.random.seed(10)

data_1 = np.random.normal(100, 10, 200)
data_2 = np.random.normal(90, 20, 200)
data_3 = np.random.normal(80, 30, 200)
data_4 = np.random.normal(70, 40, 200)
data = [data_1, data_2, data_3, data_4]

fig = plt.figure(figsize =(10, 7))

# Creating axes instance
ax = fig.add_axes([0, 0, 1, 1])

# Creating plot
bp = ax.boxplot(data)

# show plot
plt.show()
```

OUTPUT:



(c) Tree map / heat map.

Treemap in plotly.express is convenient to use, high_ranking terminal to plotly, which accomplishes a variety of types of data and produces easy-to-style figures. A treemap provides a stratified view of data and makes it easy to blot the patterns. The tree branches are characterized by rectangles and each of the sub_branches is shown in a smaller rectangle.

Syntax: plotly.express.treemap(data_frame=None, names=None, values=None, parents=None, ids=None, path=None, color=None, color_continuous_scale=None, range_color=None, color_continuous_midpoint=None, color_discrete_sequence=None, color_discrete_map={}, hover_name=None, hover_data=None, custom_data=None, labels={}, title=None, template=None, width=None, height=None, branchvalues=None, maxdepth=None)

Parameters:

data_frame: This argument needs to be passed for column names (and not keyword names) to be used. Array-like and dict are transformed internally to a pandas DataFrame.

names: Either a name of a column in data_frame, or a pandas Series or array_like object. Values from this column or array_like are used as labels for sectors.

values: Either a name of a column in data_frame, or a pandas Series or array_like object. Values from this column or array_like are used to set values associated to sectors.

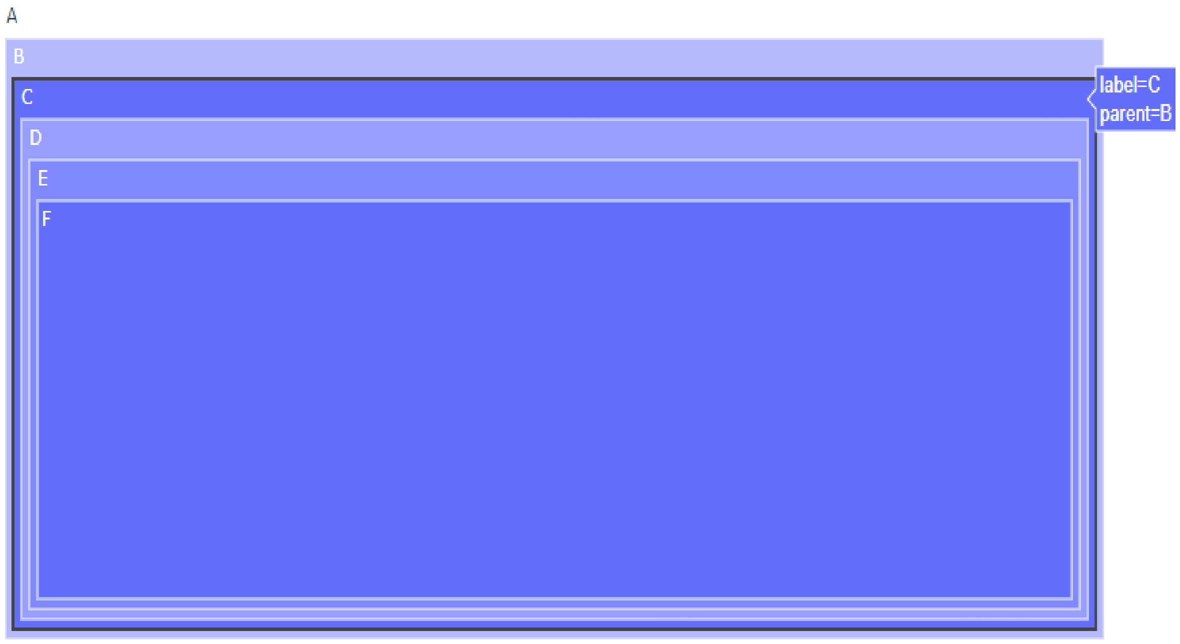
path: Either names of columns in data_frame, or pandas Series, or array_like objects List of columns names or columns of a rectangular dataframe defining the hierarchy of sectors, from root to leaves.

color: Either a name of a column in data_frame, or a pandas Series or array_like object. Values from this column or array_like are used to assign color to marks.

Program:

```
import plotly.express as px
 fig = px.treemap(
    names = ["A","B", "C", "D", "E","F"],
    parents = ["", "A", "B", "C", "D","E"]
)
 fig.show()
```

OUTPUT:



Program 11: Experiment to demonstrate data visualization with charts: bump chart, gantt chart and histograms.

(a) Gantt chart

A Gantt chart is a type of bar chart that illustrates a project schedule. The chart lists the tasks to be performed on the vertical axis, and time intervals on the horizontal axis. The width of the horizontal bars in the graph shows the duration of each activity.

Gantt Charts and Timelines with plotly.express

Plotly Express is the easy-to-use, high-level interface to Plotly, which operates on a variety of types of data and produces easy-to-style figures. With px.timeline (introduced in version 4.9) each data point is represented as a horizontal bar with a start and end point specified as dates.

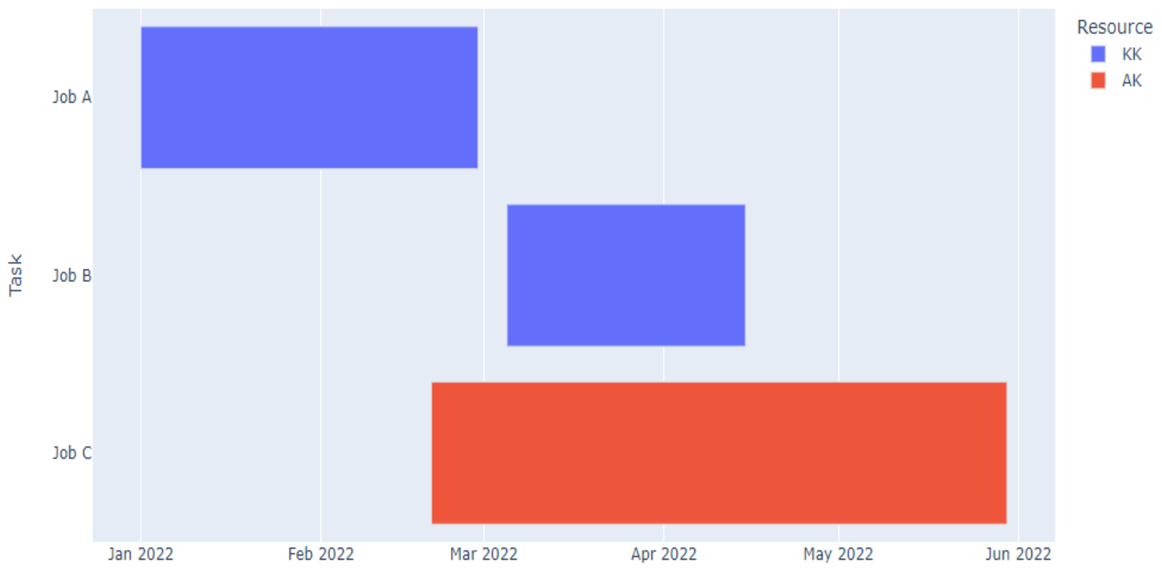
The px.timeline function by default sets the X-axis to be of type=date, so it can be configured like any time-series chart.

Plotly Express also supports a general-purpose px.bar function for bar charts.

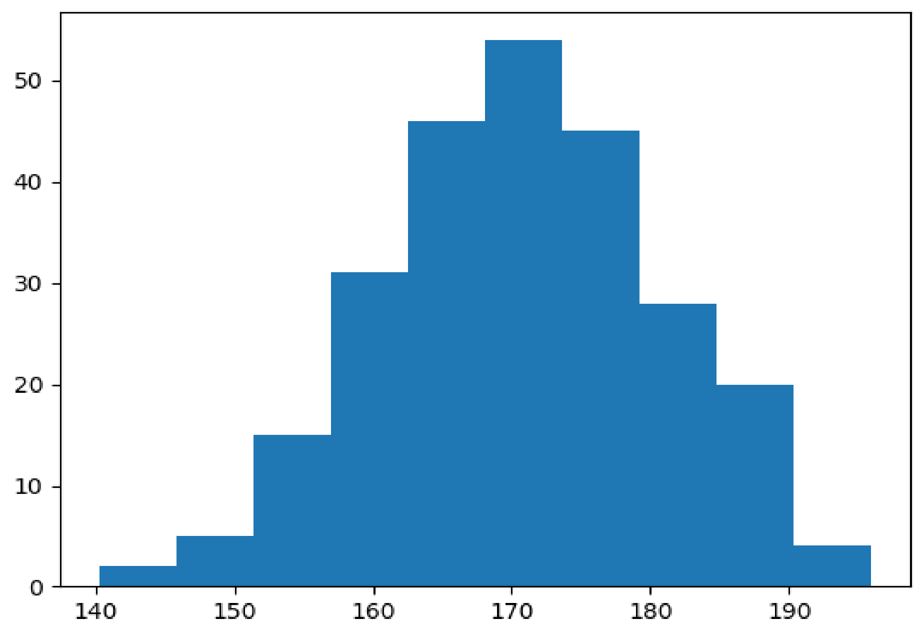
Program:

```
import plotly.express as px
import pandas as pd

df = pd.DataFrame([
    dict(Task="Job A", Start='2022-01-01', Finish='2022-02-28', Resource="KK"),
    dict(Task="Job B", Start='2022-03-05', Finish='2022-04-15', Resource="KK"),
    dict(Task="Job C", Start='2022-02-20', Finish='2022-05-30', Resource="AK")
])

fig = px.timeline(df, x_start="Start", x_end="Finish", y="Task", color="Resource")
fig.update_yaxes(autorange="reversed")
fig.show()
```

OUTPUT:



(b) Histograms.
A histogram is a graph showing frequency distributions.
It is a graph showing the number of observations within each given interval.
Example: Say you ask for the height of 250 people, you might end up with a histogram like this:



You can read from the histogram that there are approximately:

2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm
31 people from 157 to 162cm
46 people from 163 to 168cm
53 people from 168 to 173cm
45 people from 173 to 178cm
28 people from 179 to 184cm
21 people from 185 to 190cm
4 people from 190 to 195cm

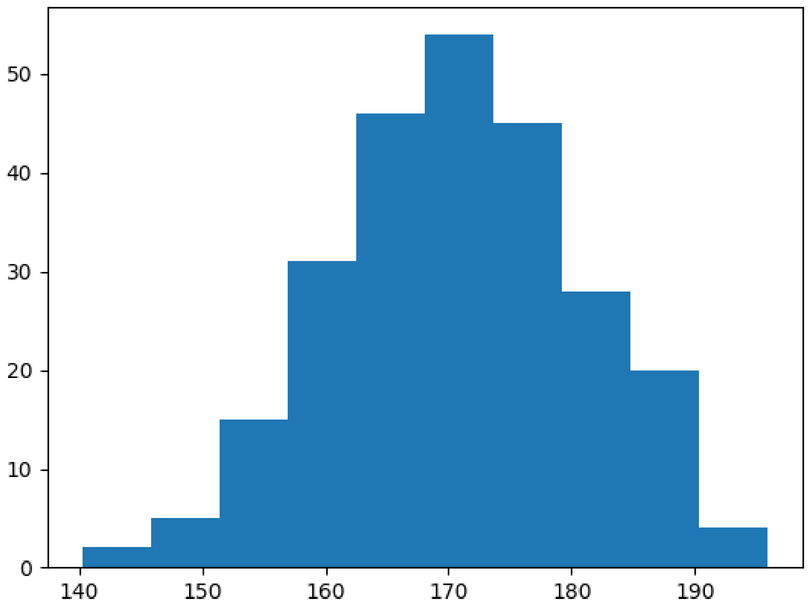
Create Histogram
In Matplotlib, we use the hist() function to create histograms.

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10. Learn more about Normal Data Distribution in our Machine Learning Tutorial.

Program:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```

OUTPUT:



Program 12: Experiment to demonstrate data visualization with charts: motion charts and waterfall charts.

(a) Motion charts

Animations are a great way to make Visualizations more attractive and user appealing. It helps us to demonstrate Data Visualization in a Meaningful Way. Python helps us to create Create Animation Visualization using existing powerful Python libraries. Matplotlib is a very popular Data Visualisation Library and is used commonly used for graphical representation of data and also for animations using inbuilt functions.

There are two ways of Creating Animation using Matplotlib:

Using pause() function

Using FuncAnimation() function

<u>**Method 1: Using pause() function**</u>

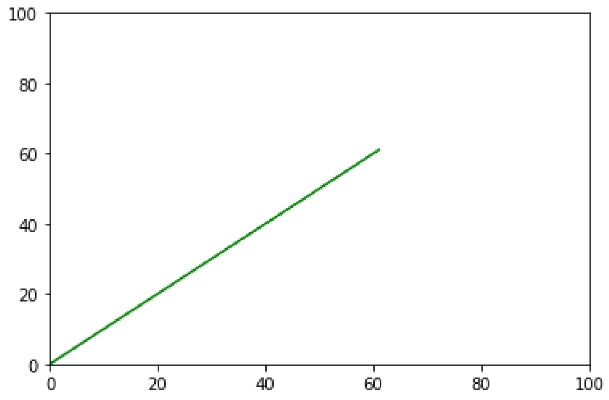
The pause() function in the pyplot module of the matplotlib library is used to pause for interval seconds mentioned in the argument. Consider the below example in which we will create a simple linear graph using matplotlib and show Animation in it:

Create 2 arrays, X and Y, and store values from 1 to 100.
Plot X and Y using plot() function.
Add pause() function with suitable time interval
Run the program and you'll see the animation.

Program:

```
from matplotlib import pyplot as plt
x = []
y = []

for i in range(100):
    x.append(i)
    y.append(i)
      # Mention x and y limits to define their range
    plt.xlim(0, 100)
    plt.ylim(0, 100)

    # Ploting graph
    plt.plot(x, y, color = 'green')
    plt.pause(0.01)
 plt.show()
```

<u>**OUTPUT:**</u>



For reference: https://www.geeksforgeeks.org/how-to-create-animations-in-python/

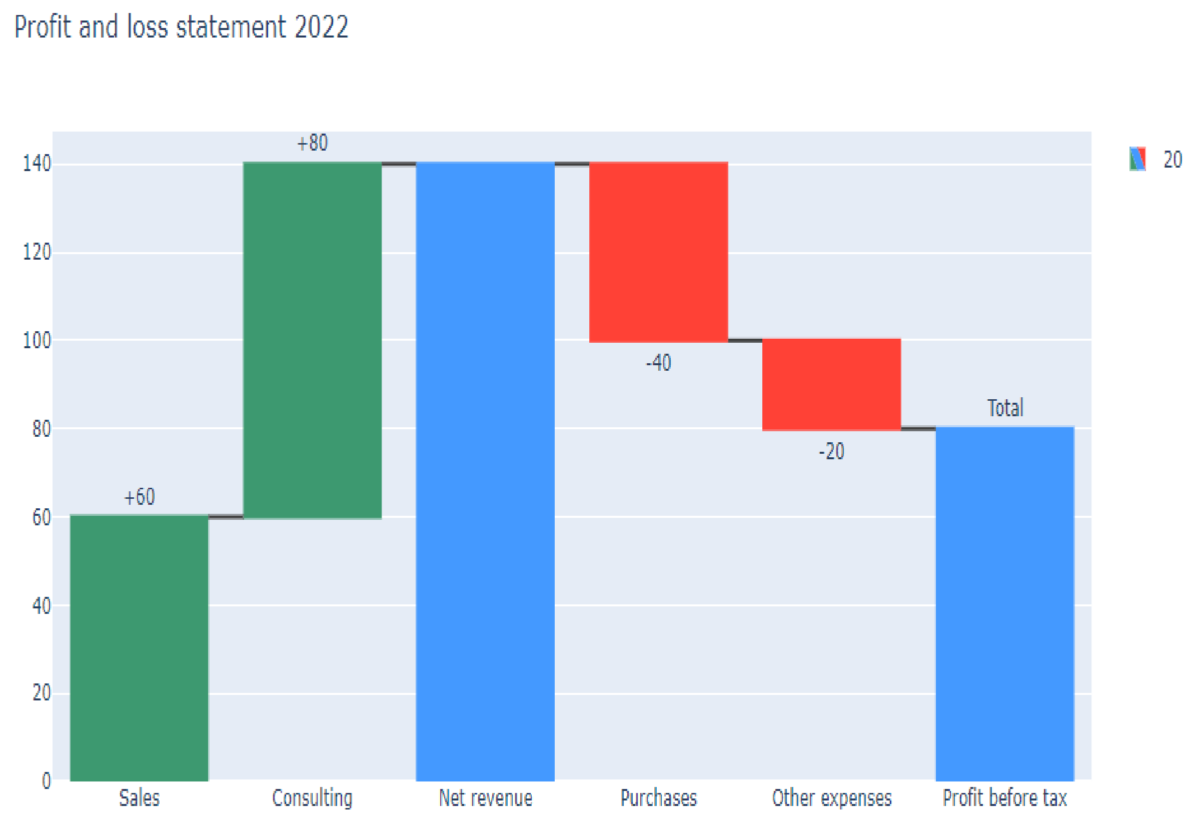
(b) Waterfall charts

A waterfall chart (also known as a cascade chart or a bridge chart) is a special kind of chart that illustrates how positive or negative values in a data series contribute to the total.

Program:

```
import plotly.graph_objects as go

fig = go.Figure(go.Waterfall(
    name = "20", orientation = "v",
    measure = ["relative", "relative", "total", "relative", "relative", "total"],
    x = ["Sales", "Consulting", "Net revenue", "Purchases", "Other expenses", "Profit before tax"],
    textposition = "outside",
    text = ["+60", "+80", "", "-40", "-20", "Total"],
    y = [60, 80, 0, -40, -20, 0],
    connector = {"line":{"color":"rgb(63, 63, 63)"}},
))

fig.update_layout(
        title = "Profit and loss statement 2022",
        showlegend = True
)
fig.show()
```

<u>OUTPUT:</u>



Program 13: Construction of advanced visualization with waffle charts.

Steps:

1.import the necessary packages:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
```

2. Create a DataFrame. For this tutorial, I will make a DataFrame that will fakely represent the number of immigrants from Argentina, Brazil, Cuba, and Peru to the US. The numbers are just imaginary.

```
df = pd.DataFrame({
    'country': ['Argentina', 'Brazil', 'Cuba', 'Peru'],
    'number': [212, 334, 1500, 130]
})
```

3. As we will present the contribution to each country's immigrants number to the total, we need to know the total. Calculate the total:

```
total = sum(df['number'])
```

4. Calculate the proportion of each country

```
proportions = [(float(value) / total) for value in df['number']]
```

```
#Output:
[0.0974264705882353,
 0.15349264705882354,
 0.6893382352941176,
 0.05974264705882353]
```

5. Specify the width and height of the chart. Then calculate the total number of tiles in the waffle chart

```
width = 40
height=10
total= width * height
```

6. Determine, how many tiles will be allotted to each country

```
tiles_per_category = [round(proportion * total) for proportion in proportions]
```

```
#Output:
[39, 61, 276, 24]
```

All the data preparation is done.

7. All the easier work is done. Now the tricky part. Generate the waffle. For that, we will generate a matrix of all zeros. And then replace each zero with the corresponding category. This matrix will resemble the waffle chart.

```
waffle = np.zeros((height, width))
category_index = 0
tile_index = 0
for col in range(width):
    for row in range(height):
        tile_index += 1
        if tile_index > sum(tiles_per_category[0:category_index]):
            category_index += 1
        waffle[row, col] = category_index
```

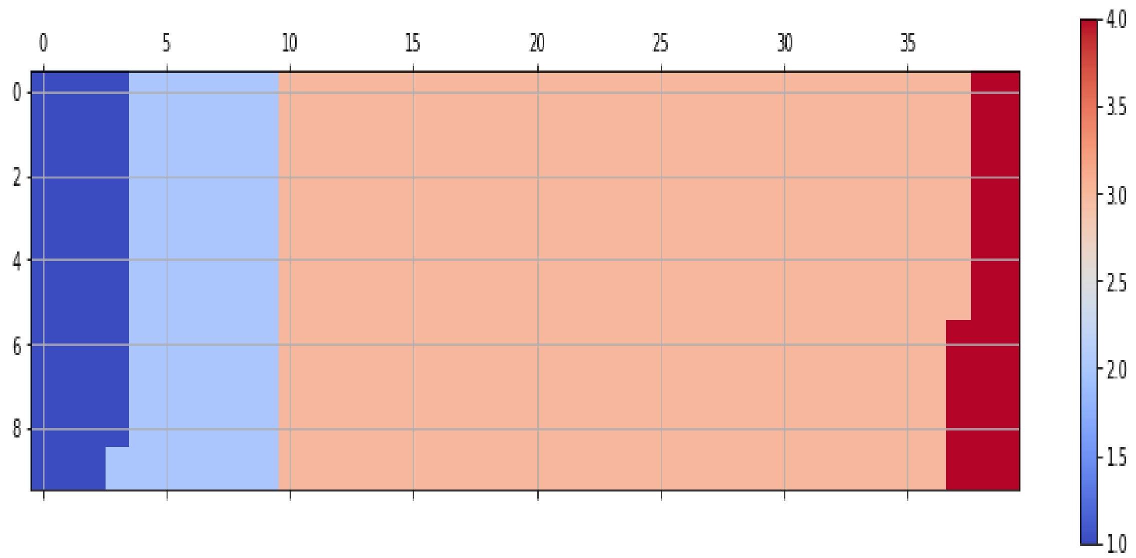
8. Choose a colormap and use matshow to display the matrix

```
fig = plt.figure()
colormap = plt.cm.coolwarm
plt.matshow(waffle, cmap=colormap)
plt.colorbar()
```

Program:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

df = pd.DataFrame({
    'country': ['Argentina', 'Brazil', 'Cuba', 'Peru'],
    'number': [212, 334, 1500, 130]
})

total = sum(df['number'])
proportions = [(float(value) / total) for value in df['number']]
width = 40
height=10
total= width * height
tiles_per_category = [round(proportion * total) for proportion in proportions]

waffle = np.zeros((height, width))
category_index = 0
tile_index = 0
for col in range(width):
    for row in range(height):
        tile_index += 1
        if tile_index > sum(tiles_per_category[0:category_index]):
            category_index += 1
        waffle[row, col] = category_index

fig = plt.figure()
colormap = plt.cm.coolwarm

plt.matshow(waffle, cmap=colormap)
plt.colorbar()
```

OUTPUT:



Program 14: Construction of advanced visualization with word clouds.

Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. Significant textual data points can be highlighted using a word cloud. Word clouds are widely used for analyzing data from social network websites.

For generating word cloud in Python, modules needed are – matplotlib, pandas and wordcloud. To install these packages, run the following commands :

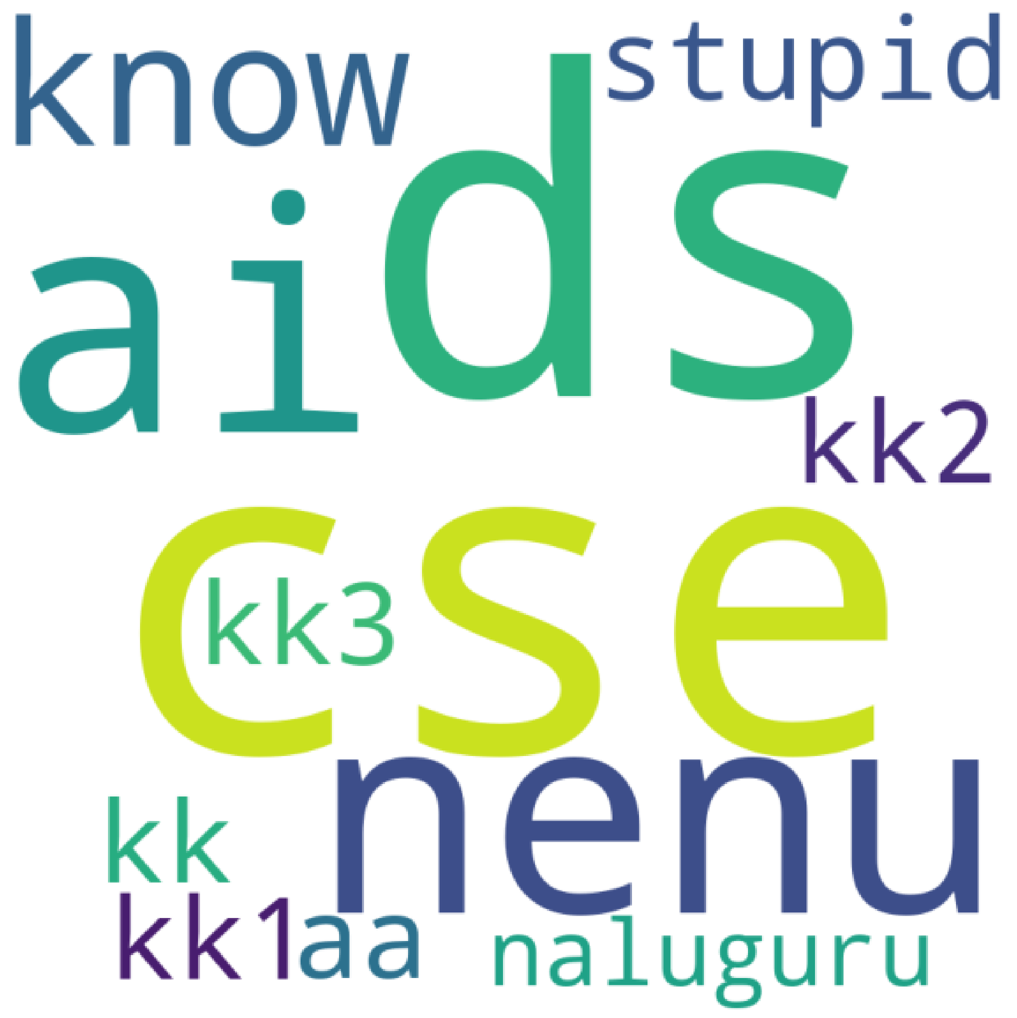
```
pip install matplotlib
pip install pandas
pip install wordcloud
```

The dataset used for generating word cloud is collected from UCI Machine Learning Repository

Program:

```
# importing all necessary modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

# Reads 'Youtube04-Eminem.csv' file
df = pd.read_csv(r'C:\Users\user\Desktop\kkpy\test.csv', encoding ="latin-1")

comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in df:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

OUTPUT:



Program 15: Construction of advanced visualization sea born and regression plots.

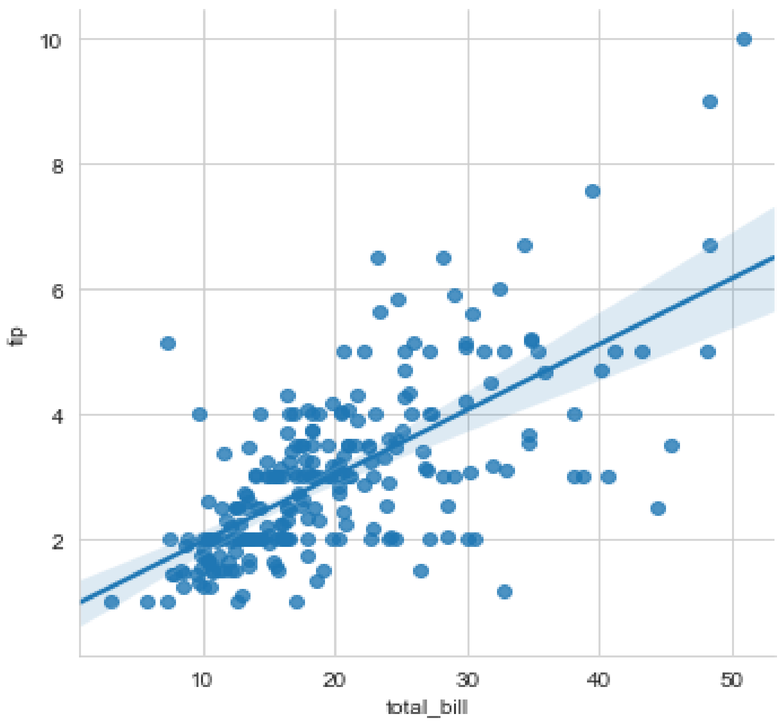
The regression plots in seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses. Regression plots as the name suggests creates a regression line between 2 parameters and helps to visualize their linear relationships. This article deals with those kinds of plots in seaborn and shows the ways that can be adapted to change the size, aspect, ratio etc. of such plots.

Seaborn is not only a visualization library but also a provider of built-in datasets. Here, we will be working with one of such datasets in seaborn named 'tips'. The tips dataset contains information about the people who probably had food at the restaurant and whether or not they left a tip. It also provides information about the gender of the people, whether they smoke, day, time and so on.

Program:

```
import seaborn as sns
# load the dataset
dataset = sns.load_dataset('tips')

# the first five entries of the dataset
dataset.head()
sns.set_style('whitegrid')
sns.lmplot(x ='total_bill', y ='tip', data = dataset)
```

<u>OUTPUT:</u>



Explanation

x and y parameters are specified to provide values for the x and y axes. sns.set_style() is used to have a grid in the background instead of a default white background. The data parameter is used to specify the source of information for drawing the plots.

Program 16: Creating maps and visualizing geospatial data with folium and map styles.

One of the most important tasks for someone working on datasets with countries, cities, etc. is to understand the relationships between their data's physical location and their geographical context. And one such way to visualize the data is using Folium.

Folium is a powerful data visualization library in Python that was built primarily to help people visualize geospatial data. With Folium, one can create a map of any location in the world. Folium is actually a python wrapper for leaflet.js which is a javascript library for plotting interactive maps.

We shall now see a simple way to plot and visualize geospatial data. We will use a dataset consisting of unemployment rates in the US

Installation
If folium is not installed, one can simply install it using any one of the following commands:

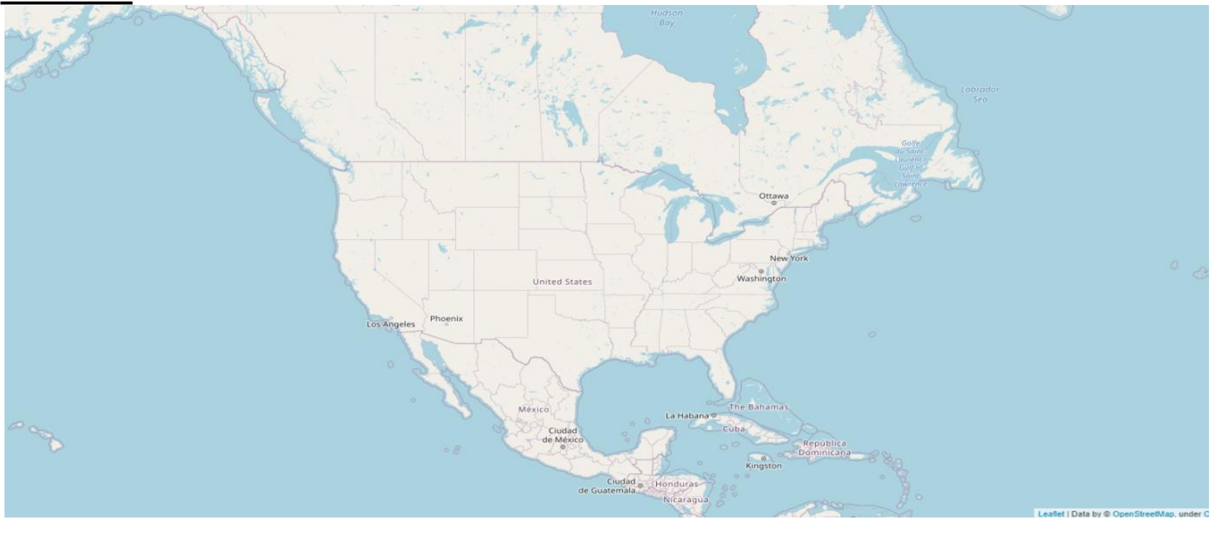
```
$ pip install folium
OR
$ conda install -c conda-forge folium
```

Using folium.Map(), we will create a base map and store it in an object. This function takes location coordinates and zoom values as arguments.
Syntax: folium.Map(location,tiles= "OpenStreetMap" zoom_start=4)
Parameters:
location: list of location coordinates
tiles: default is OpenStreetMap. Other options: tamen Terrain, Stamen Toner, Mapbox Bright etc.
zoom_start: int

Program:

```
# import the folium, pandas libraries
import folium
import pandas as pd

# initialize the map and store it in a m object
m = folium.Map(location = [40, -95],zoom_start = 4)
# show the map
m.save('my_map.html')
```

<u>OUTPUT:</u>



Program 17: Creating maps and visualizing geospatial data using maps with markers.

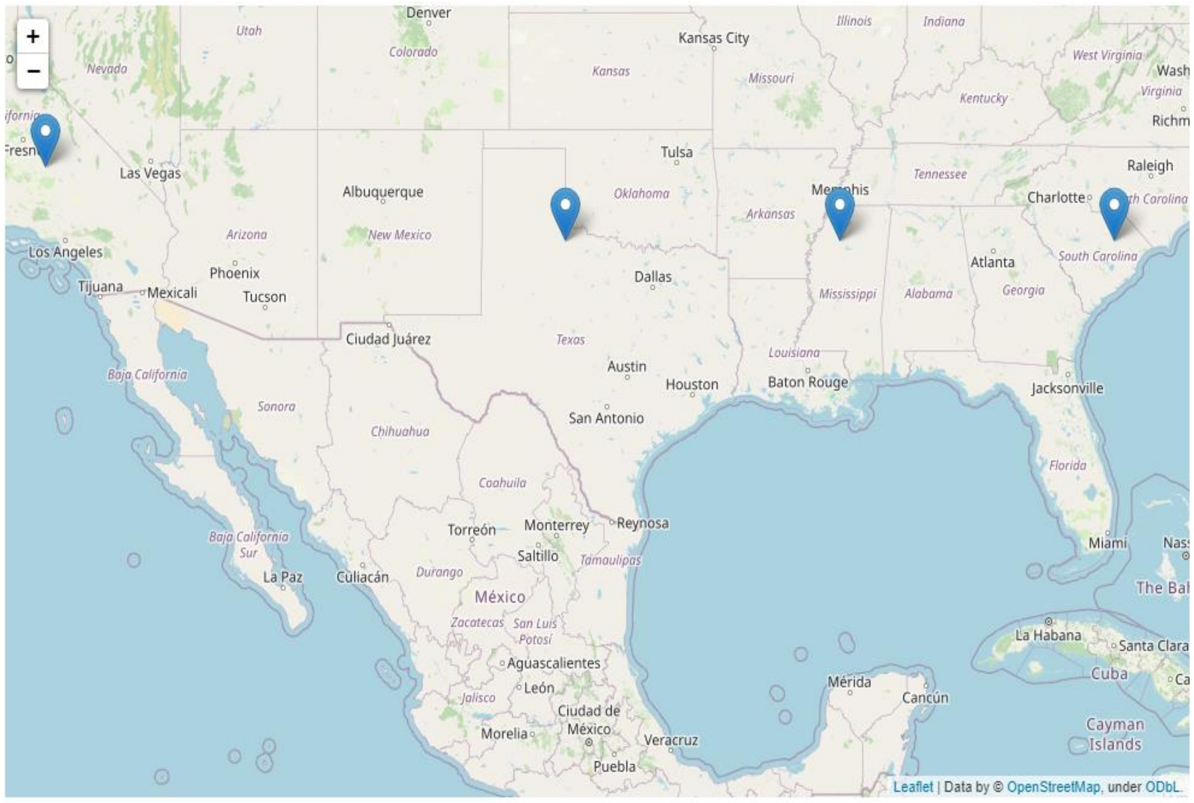
Markers are important in an interactive map to specify a location. Folium provides folium.Marker class to create a marker in a given location that can be added to a map.

Basic Markers
It's possible to plot all the points from our data by passing that process as a lambda function to the apply method on our dataframe.

Program:

```
import folium
import pandas as pd

hosp_df = pd.read_csv(r'C:\Users\user\Desktop\kkpy\map.csv')
WORKING_COLS = ["ADDRESS", "STATE", "TYPE", "STATUS", "POPULATION",
"LATITUDE", "LONGITUDE"]

hosp_df = hosp_df.loc[hosp_df["STATE"] == STATE, WORKING_COLS]

hosp_df = hosp_df[hosp_df["POPULATION"] >= 0]

m=folium.Map(location=[hosp_df["LATITUDE"].mean(),hosp_df["LONGITUDE"].mean()],
zoom_start=8)

hosp_df.apply(lambda row: folium.Marker(
        location=[row['LATITUDE'], row['LONGITUDE']]).add_to(m), axis=1)
m
```

<u>OUTPUT:</u>



https://www.geeksforgeeks.org/seaborn-regression-plots/

https://www.geeksforgeeks.org/visualizing-geospatial-data-using-folium-in-python/

https://www.safegraph.com/guides/visualizing-geospatial-data
(may helpful)

Program 13: Experiment to demonstrate building a dashboard with tables and charts for any business applications.

Program 14: Experiment to demonstrate data visualization for prediction and forecasting with trend lines.

Program 20: Creating maps and visualizing geospatial data using choropleth maps.