pandasNotes

December 19, 2022

1 Pandas

- is Python library used to analyze data
- has functions for analyzing, cleaning, exploring, and manipulating data
- `Pandas` refer to “Python Data Analysis”

Why Pandas?

- allows us to analyze big data and make conclusions based on statistical theories.
- can clean messy data sets, and make them readable and relevant.

1.1 Series

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

Labels (index) - If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.

[ ]:

```
import pandas as pd

data = pd.Series([0.25, 0.5, 0.75, 1.0])

print(data)
print('---------------------')
print(type(data))
print('---------------------')
print(data.values)
print('---------------------')
print(data.index)
print('---------------------')
print(data.keys)
print('---------------------')
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
---------------------
```

```
<class 'pandas.core.series.Series'>
---------------------
[0.25 0.5  0.75 1.  ]
---------------------
RangeIndex(start=0, stop=4, step=1)
---------------------
<bound method Series.keys of 0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64>
---------------------
```

describe()

- method returns description of the data in the DataFrame or Series.

```
count - The number of not-empty values.
mean - The average (mean) value.
std - The standard deviation.
min - the minimum value.
25% - The 25% percentile*.
50% - The 50% percentile*.
75% - The 75% percentile*.
max - the maximum value.
```

[ ]:

```
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

data = pd.Series(data_list)

data_description = data.describe()

print(type(data_description))
print('---------------------')
print(data_description)
print('---------------------')
```

```
<class 'pandas.core.series.Series'>
---------------------
count    11.000000
mean      5.363636
std       2.292280
min       2.000000
25%       3.500000
50%       5.000000
75%       7.000000
max       9.000000
```

```
dtype: float64
---------------------
```

agg()

- the same as describe but it doesn't show all description just the information passed for it

[ ]:
```python
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

data = pd.Series(data_list)

data_description = data.agg(['min', 'max', 'mean', 'std'])

print(data_description)
print('-------------------------')
```

```
min     2.000000
max     9.000000
mean    5.363636
std     2.292280
dtype: float64
-------------------------
```

Accessing & Slicing

- here we can access data with index and also slice it as `lists`

[ ]:
```python
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

print(data[1:3])
print('----------------------')
print(data[1:6:2])
print('----------------------')
print(data[5])
print('----------------------')
```

```
1    6
2    9
dtype: int64
----------------------
1    6
3    8
5    4
dtype: int64
----------------------
4
```

```
----------------------
```

create index “labels”

- we can set indexs as we want not just use default indices

[ ]:

```
import pandas as pd

data_list = [3, 6, 9, 8]
indexs = ['a', 'b', 'c', 'd']
new_data_list = dict(zip(indexs, data_list))

data1 = pd.Series(data_list, index={'a':3, 'b':6, 'c':9, 'd':8})
data2 = pd.Series(data_list, index=indexs)
data3 = pd.Series(new_data_list)
data4 = pd.Series({'a':3, 'b':6, 'c':9, 'd':8})

print(data1)
print('------------------------')
print(data2)
print('------------------------')
print(data3)
print('------------------------')
print(data4)
print('------------------------')
```

```
a    3
b    6
c    9
d    8
dtype: int64
------------------------
a    3
b    6
c    9
d    8
dtype: int64
------------------------
a    3
b    6
c    9
d    8
dtype: int64
------------------------
a    3
b    6
c    9
d    8
dtype: int64
```

Operators (&, |, ^)

- we use them to filter data
  - &: to get intersection between 2 lists
  - |: union
  - ^: values not repeated in just one of lists and not in another

[ ]:

```
import pandas as pd

a = pd.Index([0, 1, 3, 5, 7, 9])
b = pd.Index([2, 3, 5, 6, 9])

print(a)
print('---------------------------')
print(b)
print('---------------------------')
print(a&b)                 #intersection
print('---------------------------')
print(a|b)                 #union
print('---------------------------')
print(a^b)        #means get numbers not repeated ('just in a or b not in both')
print('---------------------------')
```

```
Int64Index([0, 1, 3, 5, 7, 9], dtype='int64')
---------------------------
Int64Index([2, 3, 5, 6, 9], dtype='int64')
---------------------------
Int64Index([3, 5, 9], dtype='int64')
---------------------------
Int64Index([0, 1, 2, 3, 5, 6, 7, 9], dtype='int64')
---------------------------
Int64Index([0, 1, 2, 6, 7], dtype='int64')
---------------------------

/tmp/ipykernel_48679/734037960.py:10: FutureWarning: Index.__and__ operating as
a set operation is deprecated, in the future this will be a logical operation
matching Series.__and__.  Use index.intersection(other) instead.
  print(a&b)                 #intersection
/tmp/ipykernel_48679/734037960.py:12: FutureWarning: Index.__or__ operating as a
set operation is deprecated, in the future this will be a logical operation
matching Series.__or__.  Use index.union(other) instead.
  print(a|b)                 #union
/tmp/ipykernel_48679/734037960.py:14: FutureWarning: Index.__xor__ operating as
a set operation is deprecated, in the future this will be a logical operation
matching Series.__xor__.  Use index.symmetric_difference(other) instead.
  print(a^b)        #means get numbers not repeated ('just in a or b not in both')
```

1.2 Plotting

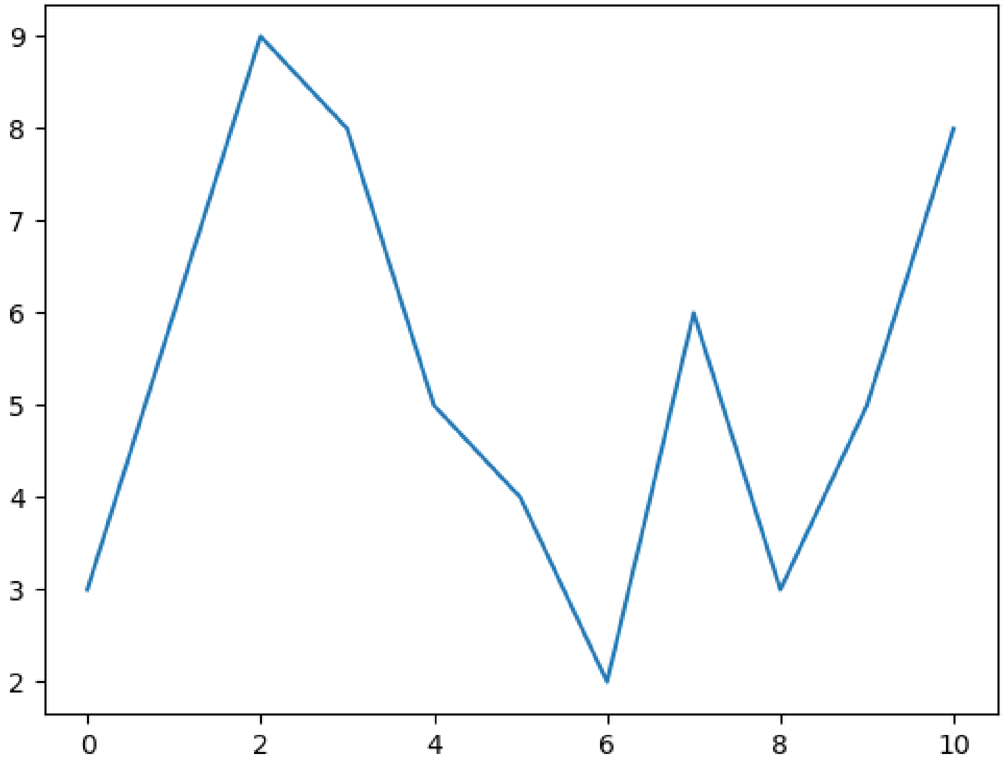
plot()

- uses to create diagrams.
- We can use `Pyplot`, a submodule of the `Matplotlib` library to visualize the diagram on the screen.

`kind` - `line` : defualt - `pie` - `bar`, `barh` - `hist` - `box` - `kde` - `area`

[ ]:
```
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

data = pd.Series(data_list)

data.plot()
#data.plot(kind='line')
```

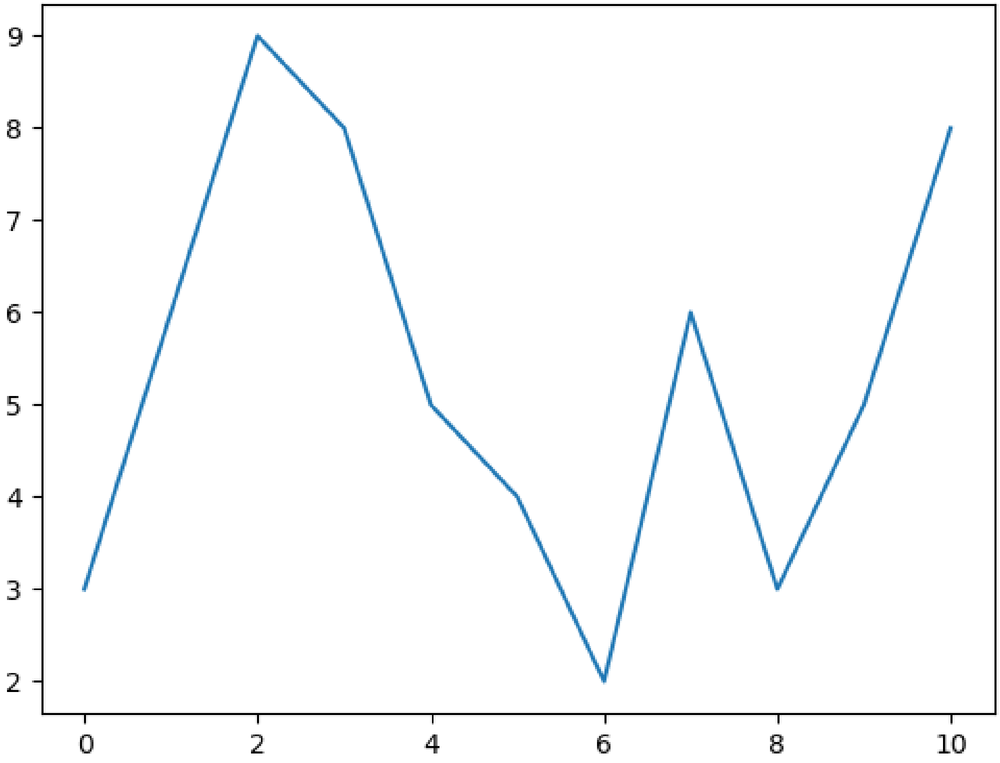
[ ]:
```
<AxesSubplot:>
```



[ ]:
```
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]
```

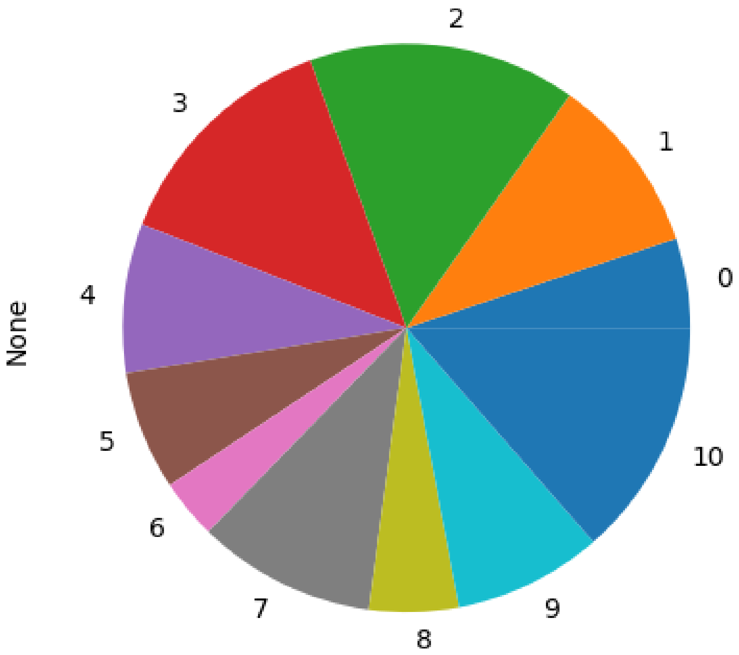
```
data = pd.Series(data_list)

data.plot(kind='line')
```

```
[ ]: <AxesSubplot:>
```



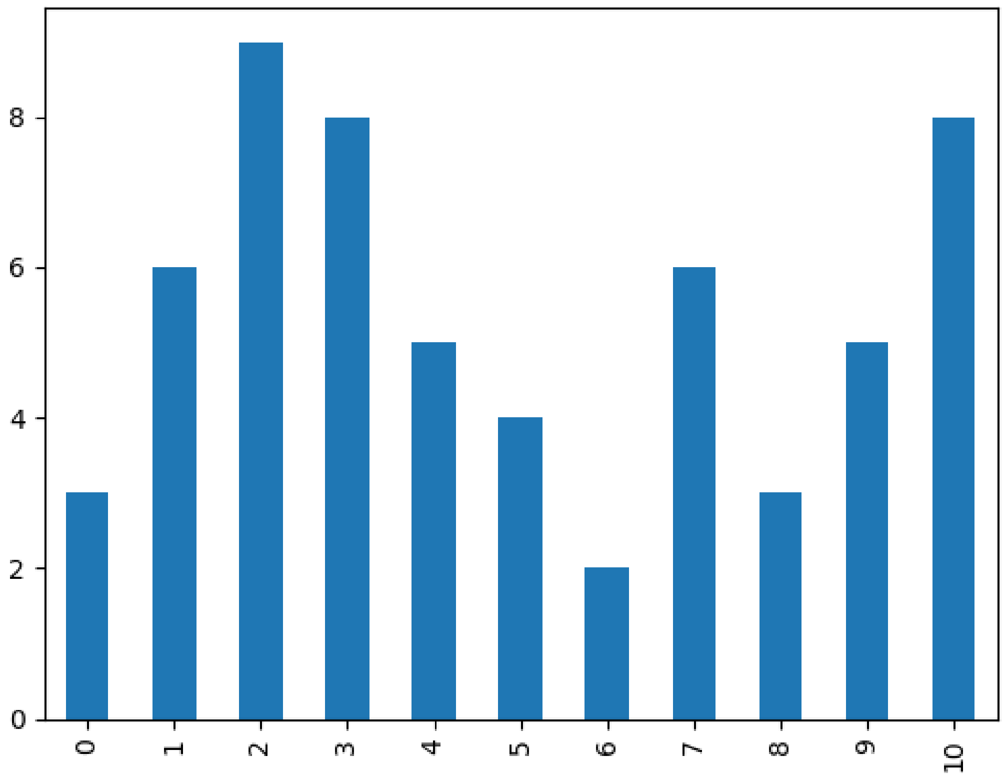
```
[ ]: import pandas as pd

     data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

     data = pd.Series(data_list)

     data.plot(kind='pie')
```

```
[ ]: <AxesSubplot:ylabel='None'>
```



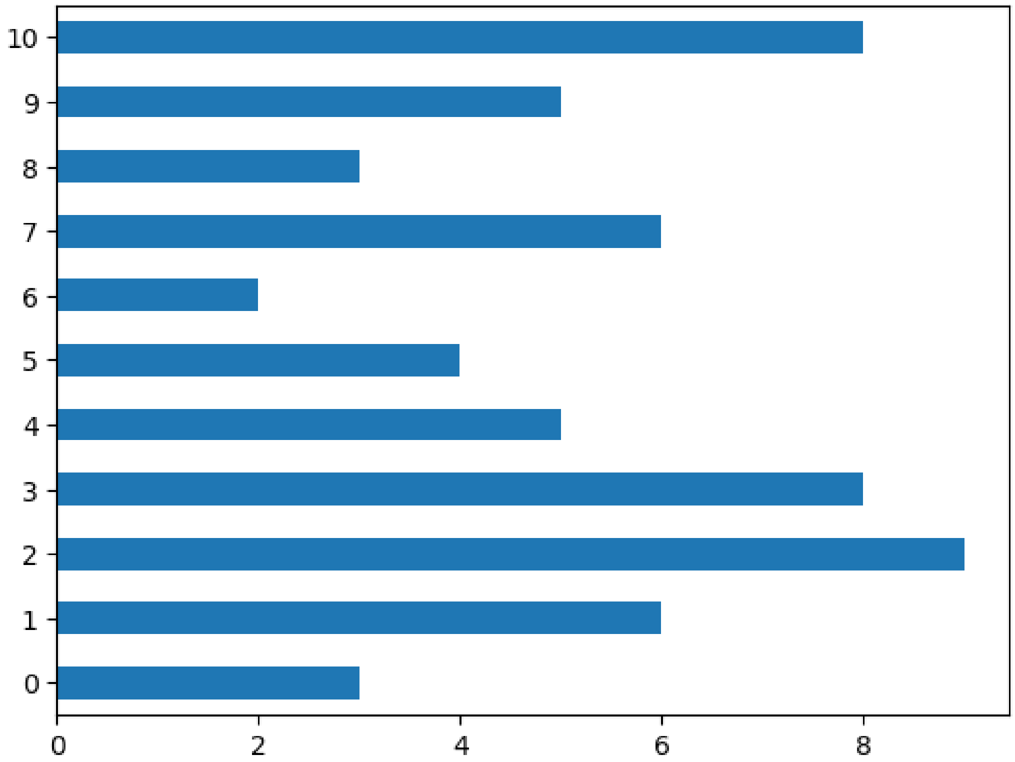
[ ]:
```
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

data = pd.Series(data_list)

data.plot(kind='bar')
```

[ ]:
```
<AxesSubplot:>
```



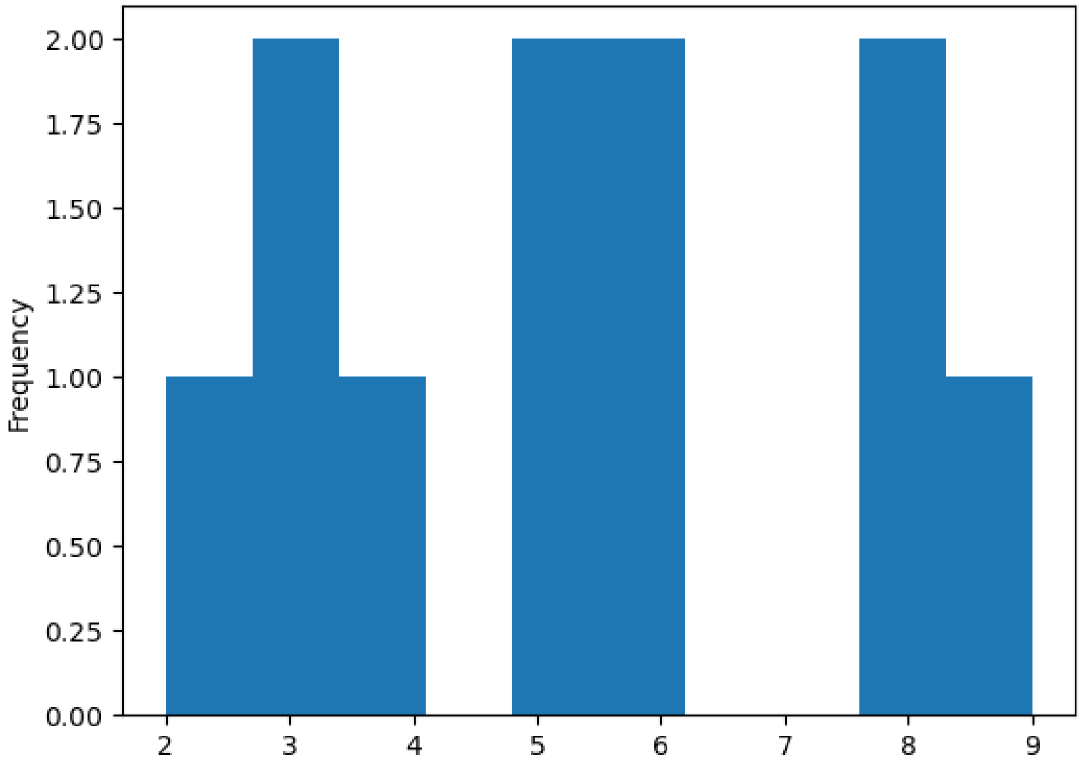
```
[ ]: import pandas as pd

     data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

     data = pd.Series(data_list)

     data.plot(kind='barh')
```

```
[ ]: <AxesSubplot:>
```



```
[ ]: import pandas as pd

     data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

     data = pd.Series(data_list)

     data.plot(kind='hist')
```

```
[ ]: <AxesSubplot:ylabel='Frequency'>
```

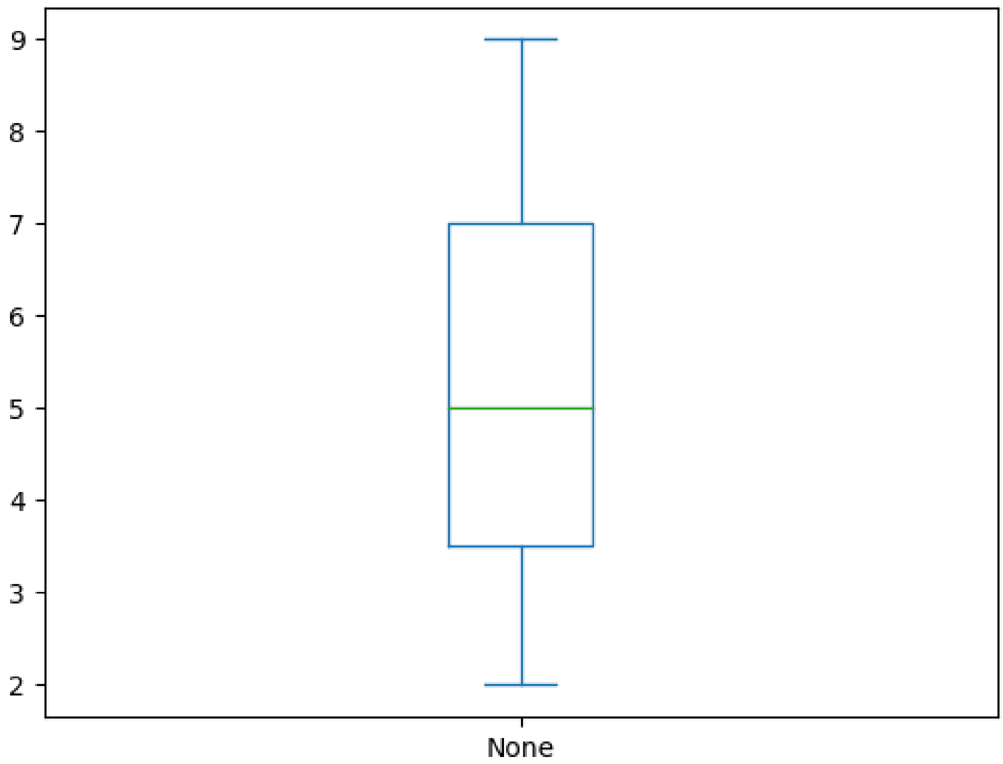


[ ]:

```python
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

data = pd.Series(data_list)

data.plot(kind='box')
```

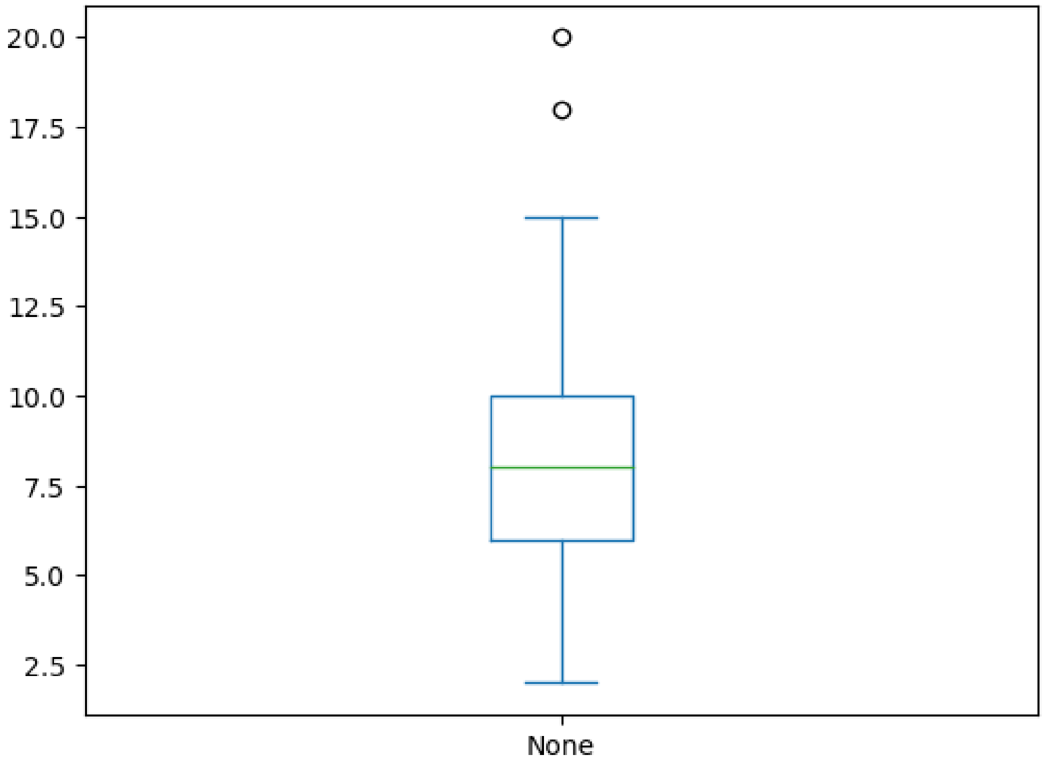
[ ]:

```
<AxesSubplot:>
```



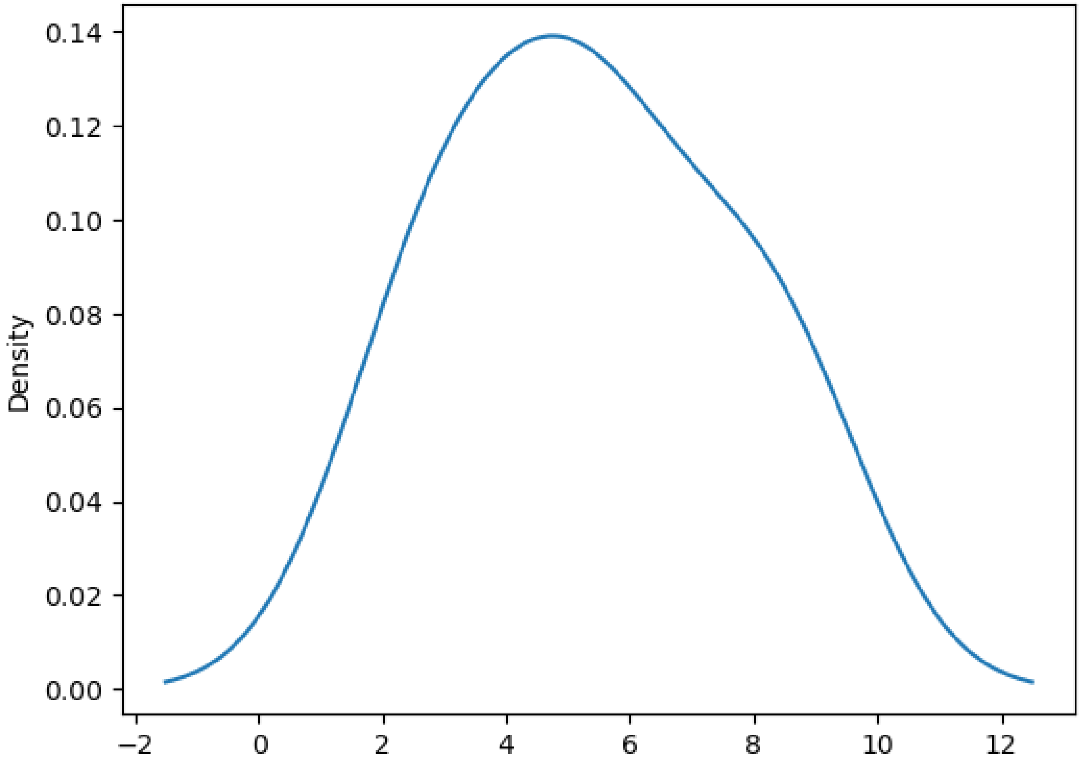
```
[ ]: import pandas as pd

     data_list = [3, 6, 9, 8, 10, 4, 2, 6, 10, 20, 15, 18, 8]

     data = pd.Series(data_list)

     data.plot(kind='box')
```

```
[ ]: <AxesSubplot:>
```



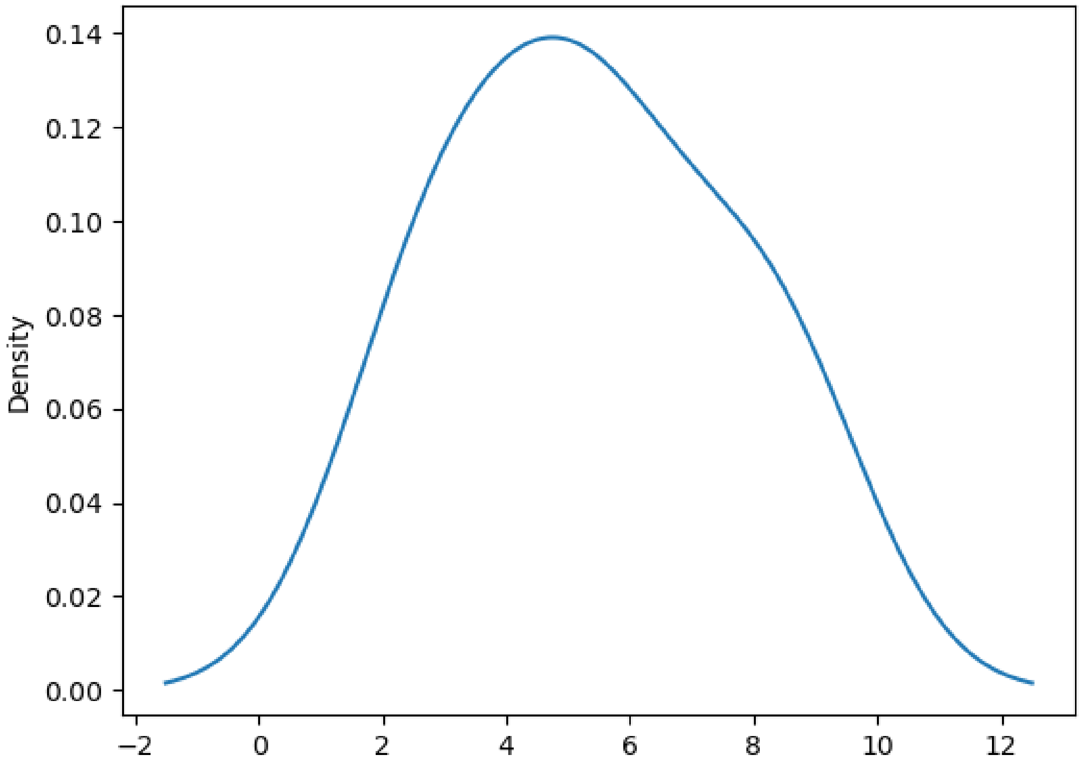
```
[ ]: import pandas as pd

     data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

     data = pd.Series(data_list)

     data.plot(kind="kde")
```

```
[ ]: <AxesSubplot:ylabel='Density'>
```



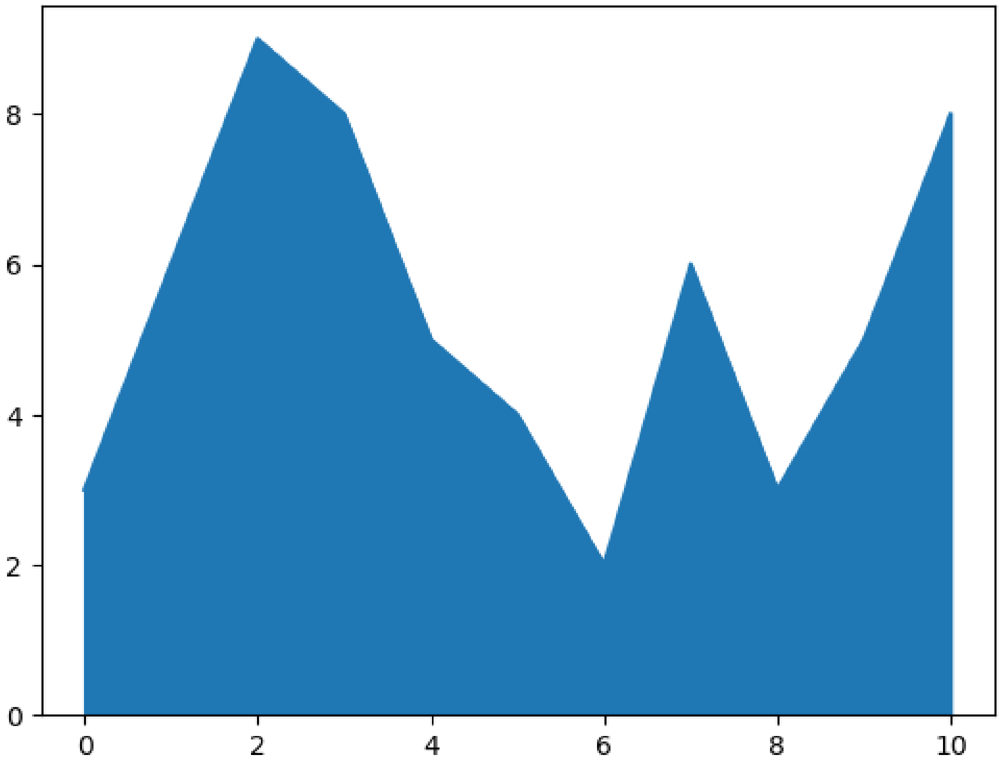
[ ]:
```
import pandas as pd

data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

data = pd.Series(data_list)

data.plot(kind='density')
```

[ ]:
```
<AxesSubplot:ylabel='Density'>
```



```
[ ]: import pandas as pd

     data_list = [3, 6, 9, 8, 5, 4, 2, 6, 3, 5, 8]

     data = pd.Series(data_list)

     data.plot(kind='area')
```

```
[ ]: <AxesSubplot:>
```



1.3 DataFrame

- is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

[ ]:

```
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=15).reshape(3, 5)

rows_names = ['a', 'b', 'c']
columns_names = ['I', 'II', 'III', 'IV', 'V']

data_frame = pd.DataFrame(data=data, index=rows_names, columns=columns_names)

print(data_frame)
print('------------------------------')
```

```
    I  II  III  IV   V
a   6  51   87  13  52
b  11  58   36  15  67
c   9  79   14  94  15
------------------------------
```

[ ]:

```python
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(4, 5)
rows_names = ['a', 'b', 'c', 'd', 'e']
w , x, y, z = [dict(zip(rows_names, row)) for row in data]

df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})

print('---------------data--------------------')
print(data)
print('----------------w----------------------')
print(w)
print('-----------------x----------------------')
print(x)
print('-----------------y------------------')
print(y)
print('-----------------z-------------------')
print(z)
print('----------------------df----------------------')
print(df)
print('----------------------------------------------')
```

```
---------------data--------------------
[[24 35 98 92 73]
 [19 95 22  4 22]
 [55 53 20 90 27]
 [88  8 29 21 76]]
----------------w----------------------
{'a': 24, 'b': 35, 'c': 98, 'd': 92, 'e': 73}
-----------------x----------------------
{'a': 19, 'b': 95, 'c': 22, 'd': 4, 'e': 22}
-----------------y------------------
{'a': 55, 'b': 53, 'c': 20, 'd': 90, 'e': 27}
-----------------z-------------------
{'a': 88, 'b': 8, 'c': 29, 'd': 21, 'e': 76}
----------------------df----------------------
   Math  Physics  French  Chemistry
a    24       19      55         88
b    35       95      53          8
c    98       22      20         29
d    92        4      90         21
e    73       22      27         76
----------------------------------------------
```

Acessing Columns

[ ]:

```
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(4, 5)
rows_names = ['a', 'b', 'c', 'd', 'e']
w , x, y, z = [dict(zip(rows_names, row)) for row in data]

df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})

print(df['Math'])
print('-----------------------------')
```

```
a    53
b    22
c     2
d     3
e    22
Name: Math, dtype: int64
-----------------------------
```

Transpose

- T property is used to transpose index and columns of the data frame. The property T is somehow related to method transpose().
- The main function of this property is to create a reflection of the data frame overs the main diagonal by making rows as columns and vice versa.

[ ]:

```
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(4, 5)
rows_names = ['a', 'b', 'c', 'd', 'e']
w , x, y, z = [dict(zip(rows_names, row)) for row in data]

df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})

print(df)
print('---------------------------------------------')
print(df.T)
print('---------------------------------------------')
```

```
   Math  Physics  French  Chemistry
a    88       78      24         96
b    99       54      28          6
c    87       19      74         90
d    19       16      36         13
e    37       94      57         41
---------------------------------------------
            a   b   c   d   e
```

```
Math        88  99  87  19  37
Physics     78  54  19  16  94
French      24  28  74  36  57
Chemistry   96   6  90  13  41
---------------------------------------------
```

keys() & values

- `keys()`: use to get column names
- `values`: return row values
- `index`: return row names

```
[ ]: import numpy as np
     import pandas as pd

     data = np.random.randint(1, 100, size=20).reshape(4, 5)
     rows_names = ['a', 'b', 'c', 'd', 'e']
     w , x, y, z = [dict(zip(rows_names, row)) for row in data]

     df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})

     print(df.keys())
     print('----------------------------------------')
     print(df.values)
     print('----------------------------------------')
     print(df.index)
     print('----------------------------------------')
```

```
Index(['Math', 'Physics', 'French', 'Chemistry'], dtype='object')
----------------------------------------
[[15 25 37 87]
 [ 6 95 10 65]
 [85 22 14 98]
 [90 59 17 42]
 [98 33 51 94]]
----------------------------------------
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
----------------------------------------
```

check

- we can check of an element is in `DataFrame` or not with using `in`
  - check in `keys()`
  - check in `values`

```
[ ]: import numpy as np
     import pandas as pd

     data = np.random.randint(1, 100, size=20).reshape(4, 5)
     rows_names = ['a', 'b', 'c', 'd', 'e']
```

```
w , x, y, z = [dict(zip(rows_names, row)) for row in data]

df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})

print('Math' in df.keys())
print('---------------------------------')
print('math' in df.keys())
print('---------------------------------')
print(12 in df.values)
print('--------------------------------')
print(55 in df.values)
print('---------------------------------')
```

```
True
---------------------------------
False
---------------------------------
True
--------------------------------
False
---------------------------------
```

stack & stack

- stack: used to stack the prescribed level(s) from columns to index. Return a reshaped DataFrame or Series having a multi-level index with one or more new inner-most levels compared to the current DataFrame.
- unstack: unstack the prescribed level(s) from row to column.

[ ]:

```
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(4, 5)
rows_names = ['a', 'b', 'c', 'd', 'e']
w , x, y, z = [dict(zip(rows_names, row)) for row in data]

df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})

print(df)
print('-----------------')
new_df = df.stack()
print(new_df)
print('-----------------')
print(new_df.unstack())
print('-----------------')
```

```
   Math  Physics  French  Chemistry
a    25       56       3         47
b    43       80      37         20
```

```
c    27       57      56          77
d    96        7      80          74
e    87       80      48          44
-----------------
a  Math         25
   Physics      56
   French        3
   Chemistry    47
b  Math         43
   Physics      80
   French       37
   Chemistry    20
c  Math         27
   Physics      57
   French       56
   Chemistry    77
d  Math         96
   Physics       7
   French       80
   Chemistry    74
e  Math         87
   Physics      80
   French       48
   Chemistry    44
dtype: int64
-----------------
   Math  Physics  French  Chemistry
a    25       56       3         47
b    43       80      37         20
c    27       57      56         77
d    96        7      80         74
e    87       80      48         44
-----------------
```

locate

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

- Pandas use the `loc` attribute to return one or more specified row(s)
- we can use `loci` to locate with indcies

[ ]:

```python
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(4, 5)
rows_names = ['a', 'b', 'c', 'd', 'e']
w , x, y, z = [dict(zip(rows_names, row)) for row in data]

df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})
```

```python
print(df.iloc[:3, :2])
print('----------------------------------')
print(df.loc['b':'c', 'Math':])
print('----------------------------------')
```

```
   Math  Physics
a    72       14
b    45       54
c    15       19
----------------------------------
   Math  Physics  French  Chemistry
b    45       54      34         63
c    15       19      35         49
----------------------------------
```

Filtering

[ ]:

```python
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(4, 5)
rows_names = ['a', 'b', 'c', 'd', 'e']
w , x, y, z = [dict(zip(rows_names, row)) for row in data]

df = pd.DataFrame({'Math':w, 'Physics':x, 'French':y, 'Chemistry':z})

print(df.loc[df.Math > 50])
print('-------------------------------')
print(df.loc[df.Math > 50, ['Math', 'French']])
print('-------------------------------')
print(df.loc[(df.Math > 50) & (df.French > 65)])
print('--------------------------------')
```

```
   Math  Physics  French  Chemistry
c    56       62      88         32
d    89       70      44         32
e    57       79      61         17
-------------------------------
   Math  French
c    56      88
d    89      44
e    57      61
-------------------------------
   Math  Physics  French  Chemistry
c    56       62      88         32
--------------------------------
```

Retrieve with Sorting Values

[ ]:

```
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(5, 4)
rows_names = ['a', 'b', 'c', 'd', 'e']
columns_names = ['Math', 'Physcis', 'French', 'Chemistry']
cols = [dict(zip(columns_names, row)) for row in data]
table = dict(zip(rows_names, cols))

grades = pd.DataFrame(table).T
print(data)
print('----------------------------------')
print(grades)
print('----------------------------------')
print(grades.sort_values(['Math'], ascending=True))
print('----------------------------------')
print(grades.sort_values(['French'], ascending=False))
print('----------------------------------')
```

```
[[33 23 45 62]
 [66 58 23 78]
 [92 58 83  9]
 [37 33 90 97]
 [41 75  8 75]]
----------------------------------
   Math  Physcis  French  Chemistry
a    33       23      45         62
b    66       58      23         78
c    92       58      83          9
d    37       33      90         97
e    41       75       8         75
----------------------------------
   Math  Physcis  French  Chemistry
a    33       23      45         62
d    37       33      90         97
e    41       75       8         75
b    66       58      23         78
c    92       58      83          9
----------------------------------
   Math  Physcis  French  Chemistry
d    37       33      90         97
c    92       58      83          9
a    33       23      45         62
b    66       58      23         78
e    41       75       8         75
----------------------------------
```

1.3.1 Statistics

- max()
- min()
- mean()
- std()

```
[ ]: import numpy as np
import pandas as pd

arr = np.arange(1, 21).reshape(5, 4)
rows_names = ['a', 'b', 'c', 'd', 'e']
columns_names = ['Math', 'Physcis', 'French', 'Chemistry']
cols = [dict(zip(columns_names, row)) for row in arr]
table = dict(zip(rows_names, cols))

grades = pd.DataFrame(table).T

print(grades)
print('--------------------------------------')
print(grades.max())
print('--------------------------------------')
print(grades.min())
print('--------------------------------------')
print(grades.mean())
print('--------------------------------------')
print(grades.std())
print('--------------------------------------')
```

```
   Math  Physcis  French  Chemistry
a     1        2       3          4
b     5        6       7          8
c     9       10      11         12
d    13       14      15         16
e    17       18      19         20
--------------------------------------
Math         17
Physcis      18
French       19
Chemistry    20
dtype: int64
--------------------------------------
Math         1
Physcis      2
French       3
Chemistry    4
dtype: int64
--------------------------------------
```

```
Math          9.0
Physcis      10.0
French       11.0
Chemistry    12.0
dtype: float64
--------------------------------------
Math         6.324555
Physcis      6.324555
French       6.324555
Chemistry    6.324555
dtype: float64
--------------------------------------
```

[ ]:

```
import numpy as np
import pandas as pd

arr = np.arange(1, 21).reshape(5, 4)
rows_names = ['a', 'b', 'c', 'd', 'e']
columns_names = ['Math', 'Physcis', 'French', 'Chemistry']

grades = pd.DataFrame(arr, index=rows_names, columns=columns_names)

print(grades)
print('---------------------')
print(grades.Math.max())
print('---------------------')
print(grades.Physcis.std())
print('---------------------')
print(grades.French.mean())
print('---------------------')
```

```
   Math  Physcis  French  Chemistry
a     1        2       3          4
b     5        6       7          8
c     9       10      11         12
d    13       14      15         16
e    17       18      19         20
---------------------
17
---------------------
6.324555320336759
---------------------
11.0
---------------------
```

[ ]:

```
import numpy as np
import pandas as pd
```

```
data = np.random.randint(1, 100, size=20).reshape(10, 2)

df = pd.DataFrame(data, columns=['A', 'B'])

print(df)
print('-----------------------')
print(df.sum())
print('-----------------------')
print(df.prod())
print('-----------------------')
print(df.mean())
print('-----------------------')
print(df['A'].sum())
print('-----------------------')
print(df['B'].mean())
print('-----------------------')
print(df.mean(axis='columns'))
print('-----------------------')
print(df.sum(axis='index'))
print('-----------------------')
```

```
    A   B
0  98  34
1  97  44
2  43  22
3  98  30
4  39  26
5  83  28
6  51  49
7  94  14
8  94  99
9   4  82
-----------------------
A    701
B    428
dtype: int64
-----------------------
A    233733474638943552
B      4002949130019840
dtype: int64
-----------------------
A    70.1
B    42.8
dtype: float64
-----------------------
701
```

```
-----------------------
42.8
-----------------------
0    66.0
1    70.5
2    32.5
3    64.0
4    32.5
5    55.5
6    50.0
7    54.0
8    96.5
9    43.0
dtype: float64
-----------------------
A    701
B    428
dtype: int64
-----------------------
```

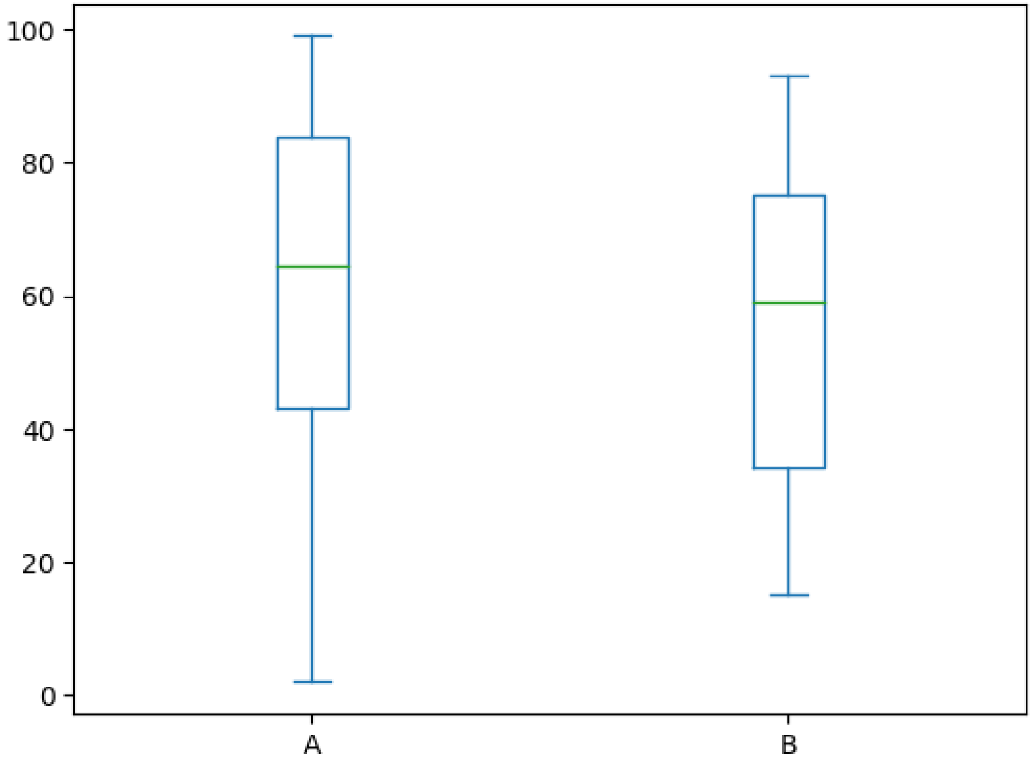
[ ]:

```
import numpy as np
import pandas as pd

data = np.random.randint(1, 100, size=20).reshape(10, 2)

df = pd.DataFrame(data, columns=['A', 'B'])

print(df)
print('-----------------------')
print(df.describe())
print('-----------------------')
df.plot(kind='box')
```

```
    A   B
0  92  93
1   2  18
2  11  78
3  80  30
4  56  89
5  73  67
6  85  65
7  99  15
8  47  53
9  42  47
-----------------------
               A          B
count  10.000000  10.000000
mean   58.700000  55.500000
```

```
std    33.326833  27.953334
min     2.000000  15.000000
25%    43.250000  34.250000
50%    64.500000  59.000000
75%    83.750000  75.250000
max    99.000000  93.000000
----------------------
```

```
[ ]: <AxesSubplot:>
```



Correlation

used to find the pairwise correlation of all columns in the Pandas Dataframe in Python.

```
[ ]: import numpy as np
     import pandas as pd

     df = pd.DataFrame(np.random.rand(5, 3), columns=['A', 'B', 'C'])

     print(df)
     print('----------------------')
     print(df.corr())
     print('----------------------')
```

```
          A         B         C
0  0.413500  0.874991  0.762693
1  0.776048  0.690615  0.224443
2  0.126178  0.118267  0.159303
3  0.023866  0.975316  0.073367
4  0.250222  0.891413  0.992557
---------------------
          A         B         C
A  1.000000  0.073569  0.116899
B  0.073569  1.000000  0.399207
C  0.116899  0.399207  1.000000
---------------------
```

Skew

The skew() method calculates the skew for each column.

[ ]:

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.rand(5, 3), columns=['A', 'B', 'C'])

print(df)
print('--------------------------------')
print(df.skew())
print('--------------------------------')
```

```
          A         B         C
0  0.404252  0.314501  0.282954
1  0.662645  0.553560  0.783029
2  0.193921  0.003088  0.039726
3  0.677650  0.766873  0.721927
4  0.578256  0.734138  0.786718
-------------------------------
A   -1.018482
B   -0.844611
C   -0.861548
dtype: float64
-------------------------------
```

1.3.2 Operations on DataFrames

`dataframe with list comprehension`

[ ]:

```
import pandas as pd

data = [{'square': i**2} for i in range(10)]
df = pd.DataFrame(data)
```

```
print(df)
print('------------------------------')
```

```
   square
0       0
1       1
2       4
3       9
4      16
5      25
6      36
7      49
8      64
9      81
------------------------------
```

[ ]:
```
import pandas as pd

data = [{'square': i**2, 'cube': i**3, 'root': i**0.5} for i in range(10)]

df = pd.DataFrame(data)

print(df)
print('----------------------------------')
```

```
   square  cube      root
0       0     0  0.000000
1       1     1  1.000000
2       4     8  1.414214
3       9    27  1.732051
4      16    64  2.000000
5      25   125  2.236068
6      36   216  2.449490
7      49   343  2.645751
8      64   512  2.828427
9      81   729  3.000000
----------------------------------
```

NaN

NaN stands for Not A Number and is one of the common ways to represent the missing value in the data

[ ]:
```
import pandas as pd

df = pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}, {'c': 5, 'd': 6}])

print(df)
```

```
     a    b    c    d
0  1.0  2.0  NaN  NaN
1  NaN  3.0  4.0  NaN
2  NaN  NaN  5.0  6.0
```

Create New Column

```
[ ]: import numpy as np
     import pandas as pd

     data = np.random.randint(1, 100,size=20).reshape(5, 4)

     grades = pd.DataFrame(data, index=['a', 'b', 'c', 'd', 'e'], columns=['Math',␣
      ↪'Physics', 'French', 'Chemistry'])

     print(grades)
     print('----------------------------------------')

     grades['Total'] = grades.sum(axis=1) / 4
     print(grades)
     print('----------------------------------------')
     print(grades.loc[grades.Total > 80])
     print('----------------------------------------')
```

```
   Math  Physics  French  Chemistry
a    52       12       6         72
b    51        1      33          1
c    44       98      18         39
d    52       84      94         84
e    47       66      80         94
----------------------------------------
   Math  Physics  French  Chemistry  Total
a    52       12       6         72  35.50
b    51        1      33          1  21.50
c    44       98      18         39  49.75
d    52       84      94         84  78.50
e    47       66      80         94  71.75
----------------------------------------
Empty DataFrame
Columns: [Math, Physics, French, Chemistry, Total]
Index: []
----------------------------------------
```

```
[ ]: import numpy as np
     import pandas as pd

     df = pd.DataFrame(np.random.randint(1, 100, size=15).reshape(5, 3),␣
      ↪columns=['A', 'B', 'C'])
```

```
result = (df['A'] + df['B']) / (df['C'] - 1)

print(df)
print('----------------------------------')
print(result)
print('----------------------------------')
```

```
    A   B   C
0   9  67  24
1  10  56  74
2  48  97  76
3  74  97  40
4  93  82  28
----------------------------------
0    3.304348
1    0.904110
2    1.933333
3    4.384615
4    6.481481
dtype: float64
----------------------------------
```

eval()

function evaluates a string describing operations on DataFrame columns. Operates on columns only, not specific rows or elements. This allows eval to run arbitrary code, which can make you vulnerable to code injection if you pass user input to this function.

[ ]:
```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.randint(1, 100, size=15).reshape(5, 3),␣
 ↪columns=['A', 'B', 'C'])
result = pd.eval('(df.A + df.B) / (df.C - 1)')

print(df)
print('--------------------------------')
print(result)
print('--------------------------------')
```

```
    A   B   C
0  82  65  62
1   8  91  64
2   2  99  62
3  46  11  47
4  83  17  16
--------------------------------
0    2.409836
```

```
1    1.571429
2    1.655738
3    1.239130
4    6.666667
dtype: float64
--------------------------------
```

query()

method is used to query the rows based on the expression (single or multiple column conditions) provided and returns a new DataFrame.

[ ]:

```python
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.rand(5, 3), columns=['A', 'B', 'C'])

print(df)
print('----------------------------')
print(df.query('A > 0.5 and B < 0.5'))
print('----------------------------')
```

```
          A         B         C
0  0.071751  0.813642  0.710513
1  0.289516  0.714604  0.396607
2  0.247046  0.033635  0.295897
3  0.798599  0.565521  0.812307
4  0.631097  0.136117  0.962577
----------------------------
          A         B         C
4  0.631097  0.136117  0.962577
----------------------------
```

[ ]:

```python
import numpy as np
import pandas as pd


df = pd.DataFrame(np.random.rand(5, 3), columns=['A', 'B', 'C'])

tmp1 = df.A < 0.5
tmp2 = df.C > 0.5
tmp3 = tmp1 & tmp2

print(df)
print('-----------------------------------')
print(tmp3)
print('-----------------------------------')
print(df[tmp3])
print('-----------------------------------')
```

```
          A         B         C
0  0.055417  0.545461  0.437569
1  0.608685  0.978338  0.701264
2  0.178670  0.764195  0.635076
3  0.338034  0.344962  0.094271
4  0.091445  0.367199  0.223734
----------------------------------
0    False
1    False
2     True
3    False
4    False
dtype: bool
----------------------------------
         A         B         C
2  0.17867  0.764195  0.635076
----------------------------------
```

[ ]:

```python
import numpy as np
import pandas as pd

def make_df(rows, cols):
    data = {col: [str(col)+str(row) for row in rows] for col in cols}
    return pd.DataFrame(data, rows)


rows = range(3)
cols = 'ABC'
df = make_df(rows, cols)
print(df)
```

```
    A   B   C
0  A0  B0  C0
1  A1  B1  C1
2  A2  B2  C2
```

merge()

- function is used to merge two DataFrame objects with a database-style join operation.
- The joining is performed on columns or indexes.
- If the joining is done on columns, indexes are ignored.
- This function returns a new DataFrame and the source DataFrame objects are unchanged.

[ ]:

```python
import pandas as pd

df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering',␣
 ↪'HR']})
```

```
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})

df3 = pd.merge(df1, df2)
print(df3)
print('---------------------------------------')

df4 = pd.DataFrame({'group': ['Accountin', 'Engineering', 'HR'],
                    'supervisor': ['Carly', 'Guido', 'Steve']})

df5 = pd.merge(df3, df4)
print(df5)
print('----------------------------------------')
```

```
  employee        group  hire_date
0      Bob   Accounting       2008
1     Jake  Engineering       2012
2     Lisa  Engineering       2004
3      Sue           HR       2014
---------------------------------------
  employee        group  hire_date supervisor
0     Jake  Engineering       2012      Guido
1     Lisa  Engineering       2004      Guido
2      Sue           HR       2014      Steve
----------------------------------------
```

Merge Options - merge has some options by passing value to `how` arg - `'inner'`: just merge intersection between them *defualt option* - `'outer'`: merge 2 dataframes and create new one that contains all itmes - `'right'`: merge 2 dataframes and create new one that contains all itmes of 1st dataframe and only intersected items from second - `'left'`: merge 2 dataframes and create new one that contains all itmes of 2nd dataframe and only intersected items from first

[ ]:

```
import pandas as pd

food_orders = pd.DataFrame({
    'name': ['Peter', 'Paul', 'Mary'],
    'food': ['fish', 'beans', 'bread']
})

drink_orders = pd.DataFrame({
    'name': ['Mary', 'Joseph'],
    'drink': ['cola', 'orange juice']
})

print(food_orders)
print('---------------------------------')
print(drink_orders)
print('---------------------------------')
```

```python
print(pd.merge(food_orders, drink_orders))
print('---------------------------------')
print(pd.merge(food_orders, drink_orders, how='inner'))
print('---------------------------------')
print(pd.merge(food_orders, drink_orders, how='outer'))
print('---------------------------------')
print(pd.merge(food_orders, drink_orders, how='right'))
print('---------------------------------')
print(pd.merge(food_orders, drink_orders, how='left'))
print('---------------------------------')
```

```
    name   food
0  Peter   fish
1   Paul  beans
2   Mary  bread
---------------------------------
     name         drink
0    Mary          cola
1  Joseph  orange juice
---------------------------------
   name   food drink
0  Mary  bread  cola
---------------------------------
   name   food drink
0  Mary  bread  cola
---------------------------------
     name   food         drink
0   Peter   fish           NaN
1    Paul  beans           NaN
2    Mary  bread          cola
3  Joseph    NaN  orange juice
---------------------------------
     name   food         drink
0    Mary  bread          cola
1  Joseph    NaN  orange juice
---------------------------------
    name   food drink
0  Peter   fish   NaN
1   Paul  beans   NaN
2   Mary  bread  cola
---------------------------------
```

drop()

method removes the specified row or column. By specifying the column axis (axis=‘columns’), the `drop()` method removes the specified column. By specifying the row axis (`axis='index'`), the `drop()` method removes the specified row.

[ ]:

```python
import pandas as pd

df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering',␣
 ↪'HR']})

df2 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'salary': [70000, 80000, 120000, 90000]})

print(df1)
print('---------------------------------')
print(df2)
print('---------------------------------')

df3 = pd.merge(df1, df2, left_on='employee', right_on='name')
print(df3)
print('---------------------------------')
print(df3.drop('name', axis=1))
print('----------------------------------')
```

```
  employee        group
0      Bob   Accounting
1     Jake  Engineering
2     Lisa  Engineering
3      Sue           HR
---------------------------------
   name  salary
0   Bob   70000
1  Jake   80000
2  Lisa  120000
3   Sue   90000
---------------------------------
  employee        group  name  salary
0      Bob   Accounting   Bob   70000
1     Jake  Engineering  Jake   80000
2     Lisa  Engineering  Lisa  120000
3      Sue           HR   Sue   90000
---------------------------------
  employee        group  salary
0      Bob   Accounting   70000
1     Jake  Engineering   80000
2     Lisa  Engineering  120000
3      Sue           HR   90000
----------------------------------
```

set_index()

method allows one or more column values become the row index.

[ ]:

```python
import pandas as pd

data = {
    'name': ['Mark', 'John', 'Lisa', 'Monica'],
    'age': [50, 40, 30, 40],
    'qualified': [True, False, False, True]
}

df = pd.DataFrame(data)

print(df)
print('-----------------------------------------')
new_df = df.set_index('name')
print(new_df)
print('-----------------------------------------')
```

```
     name  age  qualified
0    Mark   50       True
1    John   40      False
2    Lisa   30      False
3  Monica   40       True
-----------------------------------------
        age  qualified
name
Mark     50       True
John     40      False
Lisa     30      False
Monica   40       True
-----------------------------------------
```

groupby()

- is used for grouping the data according to the categories and apply a function to the categories.
- It also helps to aggregate data efficiently. Pandas `dataframe.groupby()` function is used to split the data into groups based on some criteria.

[ ]:

```python
import pandas as pd

data = {
    'key': ['A', 'B', 'C', 'A', 'B', 'C'],
    'data': range(6)
}

df = pd.DataFrame(data)

print(df)
print('-----------------')
gdf = df.groupby('key')
```

```python
print(gdf.first())          #get first value of each group
print('-----------------')
print(gdf.sum())            #sum of each group
print('-----------------')
print(gdf.describe())
print('-----------------')
print(gdf.describe().unstack())
print('-----------------')
```

```
  key  data
0   A     0
1   B     1
2   C     2
3   A     3
4   B     4
5   C     5
-----------------
     data
key
A       0
B       1
C       2
-----------------
     data
key
A       3
B       5
C       7
-----------------
     data
    count mean      std  min   25%  50%   75%  max
key
A     2.0  1.5  2.12132  0.0  0.75  1.5  2.25  3.0
B     2.0  2.5  2.12132  1.0  1.75  2.5  3.25  4.0
C     2.0  3.5  2.12132  2.0  2.75  3.5  4.25  5.0
-----------------
             key
data  count  A      2.00000
             B      2.00000
             C      2.00000
      mean   A      1.50000
             B      2.50000
             C      3.50000
      std    A      2.12132
             B      2.12132
             C      2.12132
      min    A      0.00000
```

```
             B      1.00000
             C      2.00000
      25%    A      0.75000
             B      1.75000
             C      2.75000
      50%    A      1.50000
             B      2.50000
             C      3.50000
      75%    A      2.25000
             B      3.25000
             C      4.25000
      max    A      3.00000
             B      4.00000
             C      5.00000
dtype: float64
-----------------
```

aggregate()

method allows you to apply a function or a list of function names to be executed along one of the axis of the DataFrame

[ ]:

```python
import numpy as np
import pandas as pd

df = pd.DataFrame({
    'key': ['A', 'B', 'C', 'A', 'B', 'C'],
    'data1': range(6),
    'data2': np.random.randint(0, 10, size=6)
})

print(df)
print('------------------')
new_df = df.groupby('key').aggregate({'data1': 'min', 'data2': 'max'})
print(new_df)
```

```
  key  data1  data2
0   A      0      0
1   B      1      9
2   C      2      2
3   A      3      9
4   B      4      4
5   C      5      2
------------------
     data1  data2
key
A        0      9
B        1      9
C        2      2
```

filter()

- function filters the DataFame for rows and columns.
- The returned DataFrame contains only rows and columns that are specified with the function.
- It doesn't update the existing DataFrame instead it always returns a new one.

[ ]:

```
import numpy as np
import pandas as pd

df = pd.DataFrame({
    'key': ['A', 'B', 'C', 'A', 'B', 'C'],
    'data1': range(6),
    'data2': np.random.randint(0, 10, size=6)
})

print(df)
print('------------------')

def  filter_func(x):
    return x['data2'].std() > 4

print(df.groupby('key').filter(filter_func))
print('---------------------')
```

```
  key  data1  data2
0   A      0      5
1   B      1      9
2   C      2      7
3   A      3      5
4   B      4      1
5   C      5      8
------------------
  key  data1  data2
1   B      1      9
4   B      4      1
---------------------
```

trasnform()

function call func on self producing a DataFrame with transformed values and that has the same axis length as self.

[ ]:

```
import numpy as np
import pandas as pd

df = pd.DataFrame({
    'key': ['A', 'B', 'C', 'A', 'B', 'C'],
    'data1': range(6),
    'data2': np.random.randint(0, 10, size=6)
})

print(df)
print('------------------')
new_df = df.groupby('key').transform(lambda x: x**2)
print(new_df)
print('------------------')
```

```
  key  data1  data2
0   A      0      3
1   B      1      8
2   C      2      5
3   A      3      4
4   B      4      4
5   C      5      8
------------------
   data1  data2
0      0      9
1      1     64
2      4     25
3      9     16
4     16     16
5     25     64
------------------
```