Importing libraries

```
# Importing libraries

import pandas as pd
import numpy as np
from pandas import Series, DataFrame
```

Data Structures - Series

```
# Creating Series. Series is a one-dimensional arrays

obj = pd.Series([4, 7, -5, 3])


print(obj)
```

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

```
# Gives only the values present in the series

obj.values
```

```
array([ 4,  7, -5,  3], dtype=int64)
```

```
# Gives the indexed reference of the series

obj.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

```
# Series with an index identifying each data point with a label

obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])


obj2
```

```
d    4
b    7
a   -5
c    3
dtype: int64
```

```
obj2.index
```

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```

```
# use labels in the index when selecting single values or a set of values

obj2['a']
```

```
-5
```

```
# Changing values

obj2['d'] = 6


# use labels in the index when selecting single values or a set of values

obj2[['c', 'a', 'd']]
```

```
c    3
a   -5
d    6
dtype: int64
```

```
# Filtering based on certain conditions

obj2[obj2 > 0]
```

```
d    6
b    7
c    3
dtype: int64
```

```
obj2 * 2
```

```
d    12
b    14
a   -10
c     6
dtype: int64
```

```
np.exp(obj2)
```

```
d     403.428793
b    1096.633158
a       0.006738
c      20.085537
dtype: float64
```

```
# Creating dictionaries

sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}
obj3 = pd.Series(sdata)
obj3
```

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

```
# Additing indexes

states = ['California', 'Ohio', 'Oregon', 'Texas']
obj4 = pd.Series(sdata, index=states)
obj4
```

```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

```
# Finding missing values

obj4.isnull()
```

```
California     True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

```
# A useful Series feature for many applications is that it automatically aligns by index label in arithmetic operations

obj3
```

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

```
obj4
```

```
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
dtype: float64
```

```
obj3 + obj4
```

```
California         NaN
Ohio           70000.0
```

```
Oregon         32000.0
Texas         142000.0
Utah               NaN
dtype: float64
```

```
# Assigning name of the Series and then assigning name to the index column

obj4.name = 'population'
obj4.index.name = 'state'
```

```
obj4
```

```
state
California        NaN
Ohio          35000.0
Oregon        16000.0
Texas         71000.0
Name: population, dtype: float64
```

```
# A Series’s index can be altered in-place by assignment

obj
```

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

```
# Changing the index

obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
obj
```

```
Bob      4
Steve    7
Jeff    -5
Ryan     3
dtype: int64
```

Data Structures - DataFrame

A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.). The DataFrame has both a row and column index; it can be thought of as a dict of Series all sharing the same index. Under the hood, the data is stored as one or more two-dimensional blocks rather than a list, dict, or some other collection of one-dimensional arrays.

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)
```

```
frame
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

```
# Gives the first 5 rows

frame.head()
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7

```
# If you specify a sequence of columns, the DataFrame’s columns will be arranged in that order

pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

	year	state	pop
0	2000	Ohio	1.5
1	2001	Ohio	1.7
2	2002	Ohio	3.6
3	2001	Nevada	2.4
4	2002	Nevada	2.9
5	2003	Nevada	3.2

```
# If you pass a column that isn’t contained in the dict, it will appear with missing values in the result

frame2 = pd.DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
                      index=['one', 'two', 'three', 'four','five', 'six'])


frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

```
# Displays the columns that are present in your dataset

frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

```
frame2['state']
```

```
one        Ohio
two        Ohio
three      Ohio
four     Nevada
five     Nevada
six      Nevada
Name: state, dtype: object
```

```
frame2.year
```

```
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```

```
# Rows can also be retrieved by position or name with the special loc attribute

frame2.loc['three']
```

```
year     2002
state    Ohio
pop       3.6
debt      NaN
Name: three, dtype: object
```

```
# Columns can be modified by assignment. For example, the empty 'debt' column could be assigned a scalar value or an array of

frame2['debt'] = 16.5
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
# Substituting different values for a column

val = pd.Series([-1.2, -1.5, -1.7], index=['two', 'four', 'five'])
frame2['debt'] = val
frame2
```

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	-1.2
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	-1.5
five	2002	Nevada	2.9	-1.7
six	2003	Nevada	3.2	NaN

```
# Adding a column

frame2['eastern'] = frame2.state == 'Ohio'
frame2
```

	year	state	pop	debt	eastern
one	2000	Ohio	1.5	NaN	True
two	2001	Ohio	1.7	-1.2	True
three	2002	Ohio	3.6	NaN	True
four	2001	Nevada	2.4	-1.5	False
five	2002	Nevada	2.9	-1.7	False
six	2003	Nevada	3.2	NaN	False

```
# Deleting a column

del frame2['eastern']
frame2.columns
```

```
Index(['year', 'state', 'pop', 'debt'], dtype='object')
```

Essentional functionality - Reindexing

```
obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
obj
```

```
d    4.5
b    7.2
a   -5.3
c    3.6
dtype: float64
```

```
# Calling reindex on this Series rearranges the data according to the new index,
#introducing missing values if any index values were not already present

obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
obj2
```

```
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN
dtype: float64
```

```
obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
obj3
```

```
0      blue
2    purple
4    yellow
dtype: object
```

```
obj3.reindex(range(6), method='ffill')
```

```
0      blue
1      blue
2    purple
3    purple
4    yellow
5    yellow
dtype: object
```

```
# With DataFrame, reindex can alter either the (row) index, columns, or both.
# When passed only a sequence, it reindexes the rows in the result

frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                     index=['a', 'c', 'd'],
                     columns=['Ohio', 'Texas', 'California'])
frame
```

	Ohio	Texas	California
a	0	1	2
c	3	4	5
d	6	7	8

```
frame2 = frame.reindex(['a', 'b', 'c', 'd'])
```

```
frame2
```

	Ohio	Texas	California
a	0.0	1.0	2.0
b	NaN	NaN	NaN
c	3.0	4.0	5.0
d	6.0	7.0	8.0

```
# The columns can be reindexed with the columns keyword

states = ['Texas', 'Utah', 'California']
frame.reindex(columns=states)
```

	Texas	Utah	California
a	1	NaN	2
c	4	NaN	5
d	7	NaN	8

Dropping Entries from an Axis

```
obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
obj
```

```
a    0.0
b    1.0
c    2.0
d    3.0
```

```
e    4.0
dtype: float64
```

```
new_obj = obj.drop('c')
new_obj
```

```
a    0.0
b    1.0
d    3.0
e    4.0
dtype: float64
```

```
obj.drop(['d', 'c'])
```

```
a    0.0
b    1.0
e    4.0
dtype: float64
```

```
# With DataFrame, index values can be deleted from either axis

data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data.drop(['Colorado', 'Ohio'])
```

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

```
# You can drop values from the columns by passing axis=1 or axis='columns'

data.drop('two', axis=1)
```

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

```
data.drop(['two', 'four'], axis='columns')
```

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

```
# When inplace = True is used, it performs operation on data and nothing is returned

obj.drop('c', inplace=True)
```

Indexing, Selection, and Filtering

```
obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
obj
```

```
a    0.0
b    1.0
c    2.0
d    3.0
dtype: float64
```

```
obj['b']
```

```
1.0
```

```
obj[2:4]
```

```
c    2.0
d    3.0
dtype: float64
```

```
obj[['b', 'a', 'd']]
```

```
b    1.0
a    0.0
d    3.0
dtype: float64
```

```
obj[[1, 3]]
```

```
b    1.0
d    3.0
dtype: float64
```

```
obj[obj < 2]
```

```
a    0.0
b    1.0
dtype: float64
```

```
obj['b':'c'] = 5
obj
```

```
a    0.0
b    5.0
c    5.0
d    3.0
dtype: float64
```

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data['two']
```

```
Ohio         1
Colorado     5
Utah         9
New York    13
Name: two, dtype: int32
```

```
data[['three', 'one']]
```

	three	one

```
data[:2]
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

```
data[data['three'] > 5]
```

	one	two	three	four
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
data < 5
```

	one	two	three	four
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

```
data[data < 5] = 0
data
```

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
# Selection with loc and iloc

data.loc['Colorado', ['two', 'three']]
```

```
two      5
three    6
Name: Colorado, dtype: int32
```

```
data.iloc[2, [3, 0, 1]]
```

```
four    11
one      8
two      9
Name: Utah, dtype: int32
```

```
data.iloc[2]
```

```
one       8
two       9
three    10
four     11
Name: Utah, dtype: int32
```

```
data.iloc[[1, 2], [3, 0, 1]]
```

	four	one	two
Colorado	7	0	5
Utah	11	8	9

Aritmetic and data alignment

```
s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],index=['a', 'c', 'e', 'f', 'g'])
```

```
s1
```

```
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
```

```
s2
```

```
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

```
s1 + s2
```

```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

```
df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)), columns=list('bcd'),index=['Ohio', 'Texas', 'Colorado'])
df1
```

	b	c	d
Ohio	0.0	1.0	2.0
Texas	3.0	4.0	5.0
Colorado	6.0	7.0	8.0

```
df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)), columns=list('bde'),index=['Utah', 'Ohio', 'Texas', 'Oregon'])
df2
```

	b	d	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

```
df1 + df2
```

	b	c	d	e
Colorado	NaN	NaN	NaN	NaN
Ohio	3.0	NaN	6.0	NaN
Oregon	NaN	NaN	NaN	NaN
Texas	9.0	NaN	12.0	NaN
Utah	NaN	NaN	NaN	NaN

Sorting

```
obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])
obj
```

```
d    0
a    1
b    2
c    3
dtype: int64
```

```
obj.sort_index()
```

```
a    1
b    2
c    3
d    0
dtype: int64
```

```
frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
                     index=['three', 'one'],
                     columns=['d', 'a', 'b', 'c'])
frame
```

	d	a	b	c
three	0	1	2	3
one	4	5	6	7

```
frame.sort_index()
```

	d	a	b	c
one	4	5	6	7
three	0	1	2	3

```
frame.sort_index(axis=1)
```

	a	b	c	d
three	1	2	3	0
one	5	6	7	4

```
frame.sort_index(axis=1, ascending=False)
```

	d	c	b	a
three	0	3	2	1
one	4	7	6	5

```
obj = pd.Series([4, 7, -3, 2])
obj
```

```
0    4
1    7
2   -3
3    2
dtype: int64
```

```
obj.sort_values()
```

```
2   -3
3    2
0    4
1    7
dtype: int64
```

```
# Any missing values are sorted to the end of the Series by default

obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
obj
```

```
0    4.0
1    NaN
2    7.0
3    NaN
4   -3.0
5    2.0
dtype: float64
```

```
obj.sort_values()
```

```
4   -3.0
5    2.0
0    4.0
2    7.0
```

```
1    NaN
3    NaN
dtype: float64
```

```
frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})
frame
```

	b	a
0	4	0
1	7	1
2	-3	0
3	2	1

```
frame.sort_values(by='b')
```

	b	a
2	-3	0
3	2	1
0	4	0
1	7	1

```
# To sort by multiple columns, pass a list of names

frame.sort_values(by=['a', 'b'])
```

	b	a
2	-3	0
0	4	0
3	2	1
1	7	1

Axis Indexes with Duplicate Labels

```
obj = pd.Series(range(5), index=['a', 'a', 'b', 'b', 'c'])
obj
```

```
a    0
a    1
b    2
b    3
c    4
dtype: int64
```

```
obj.index.is_unique
```

```
False
```

```
obj['a']
```

```
a    0
a    1
dtype: int64
```

```
obj['c']
```

```
4
```

```
df = pd.DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])
df
```

	0	1	2
a	2.058845	0.788215	2.207426

```
df.loc['b']
```

	0	1	2
b	1.316775	-1.137726	0.239879
b	0.096749	-0.067170	1.708222