Python Pandas

Agenda

- Introduction to Pandas
- Pandas Series
  - Creating Series
  - Accessing Series
  - Filtering Series
  - Arithmetic Series
  - Ranking and Sorting
  - Null Values
- Pandas Dataframe
  - Creating Dataframe
  - Reading Data from Different Sources
  - Dataframe Manipulations
  - Understanding Data
  - Indexing Dataframe
  - Sorting and Ranking

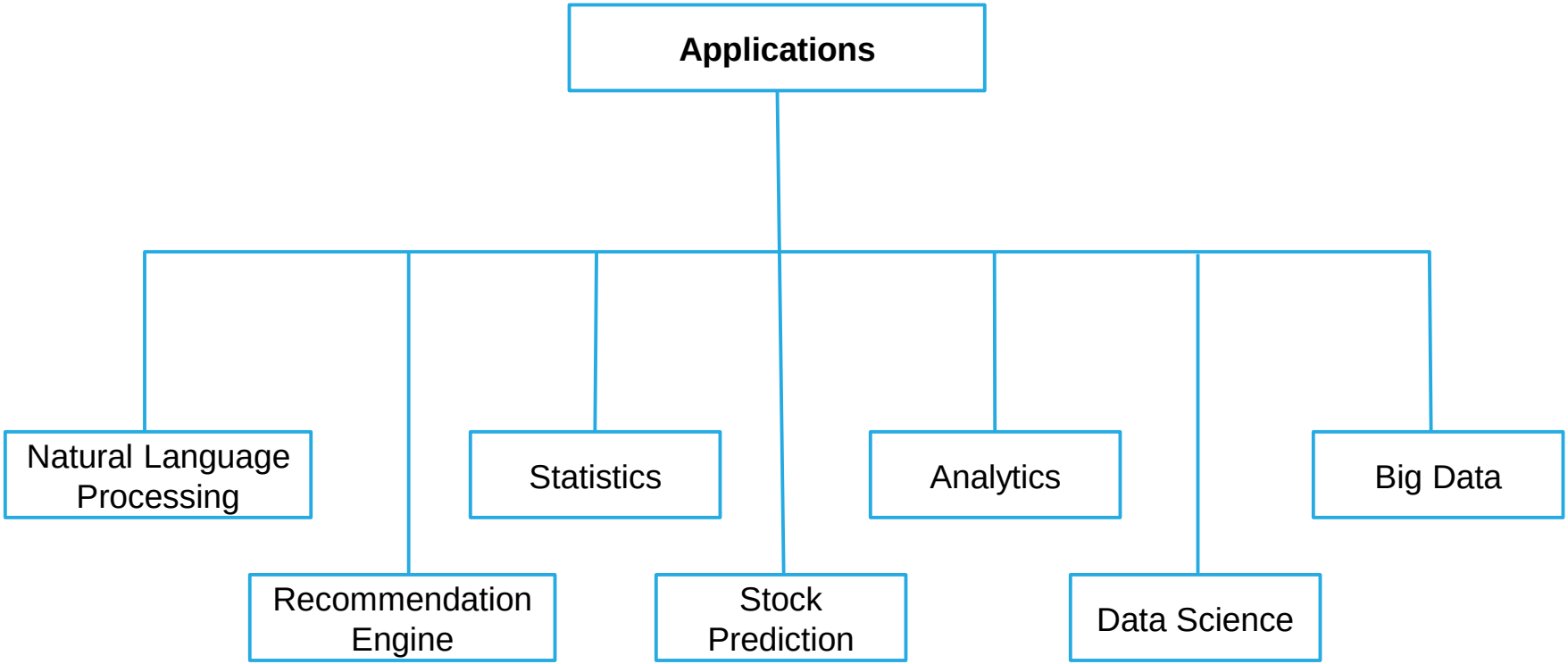
greatlearning
Learning for Life

Introduction

Introduction

- Pandas is a simple yet powerful and expressive tool
- It is an open source library in python
- It is useful in data manipulation and analysis

Applications of pandas



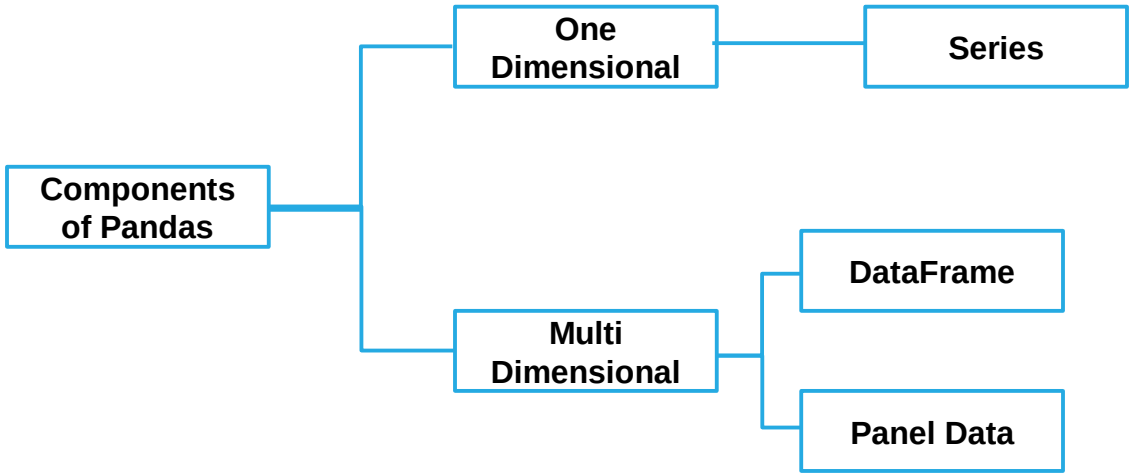
Pandas vs. Numpy

greatlearning
Learning for Life

Pandas	Numpy
High level data structures. It provides in-memory 2D table object called data frame.	Low level data structure. (np.array)
More streamlined handling of tabular data, and rich time series functionality.	Supports large multidimensional arrays and matrices.
Data alignment, handling missing data, groupby, merge, and, join methods.	A wide range of mathematical array operations.

Components of pandas

- Series and dataframe are two primary components of pandas
- Series is a one-dimensional labeled array; typically a column, and the DataFrame is a two-dimensional table made up of a group of Series

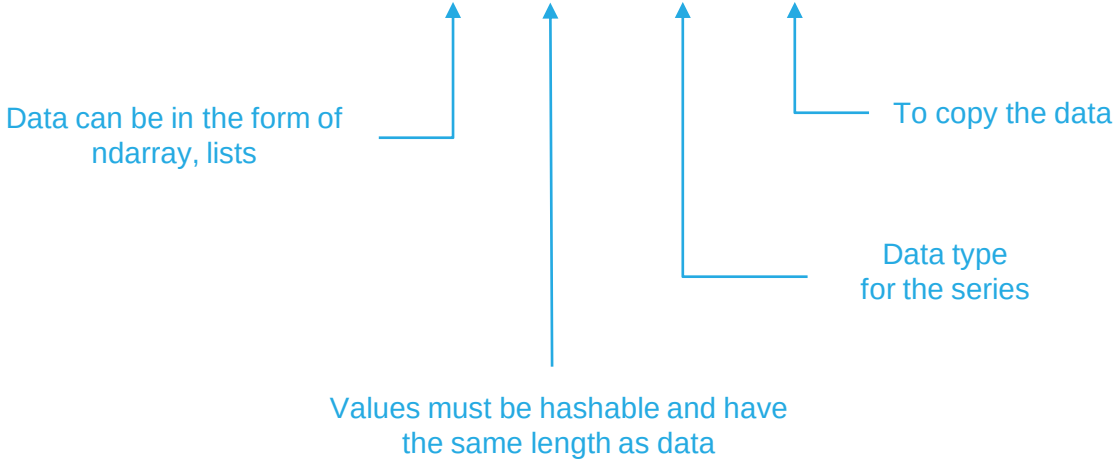


Pandas Series

Pandas series

Series can be created using the following constructor:

```
pandas.Series( data, index, dtype, copy)
```



Creating a Series

Create a series from a list and numpy array

A pandas series can be created from a python list or numpy array

```
series_list = pd.Series([1,2,3,4,5,6])
series_list
```

```
0    1
1    2
2    3
3    4
4    5
5    6
dtype: int64
```

Using list

```
series_np = pd.Series(np.array([10,20,30,40,50,60]))
series_np
```

```
0    10
1    20
2    30
3    40
4    50
5    60
dtype: int32
```

Using numpy array

Set index of a series

Specify numeric values as index while creating a series

```
series_index = pd.Series(
      np.array([11,12,13,14,15,16]),
      index=np.arange(0,12,2)
)
series_index
```

```
0     11
2     12
4     13
6     14
8     15
10    16
dtype: int32
```

Pass index parameter

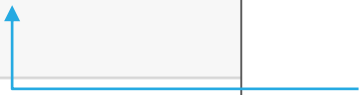
By default, index ranges from 0 to (n-1) for series of length ‘n’

Set index of a series

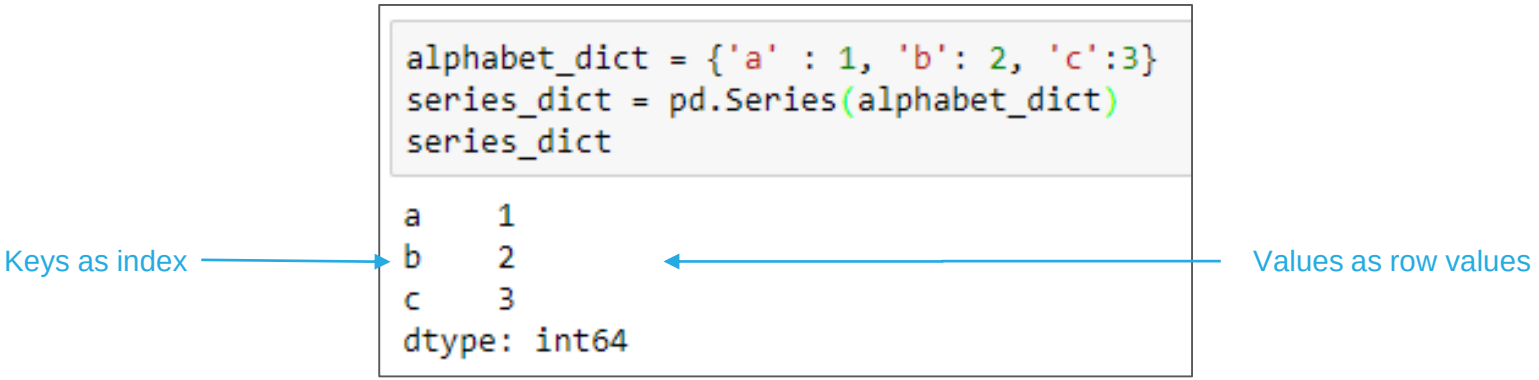
We can also specify the strings as index values

```
series_index = pd.Series(
        np.array([11,12,13,14,15,16]),
        index=['a', 'b', 'c', 'd', 'e', 'f' ]
)
series_index
```

```
a    11
b    12
c    13
d    14
e    15
f    16
dtype: int32
```



Create a series from a dictionary



The key becomes the row index while the value is the value at that row index

Create a series from a dictionary

```
alphabet_dict = {'a' : [1,2,3], 'b': [4,5], 'c':6, 'd': "Hello World"}
series_dict = pd.Series(alphabet_dict)
series_dict
```

```
a      [1, 2, 3]
b         [4, 5]
c              6
d    Hello World
dtype: object
```

If you have multiple values for a single key, those multiple values will take up a single row

Access series index and values

greatlearning
Learning for Life

To display the index names and values of the series use **.index** and **.values** respectively

```
series_dict.index
```

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

```
series_dict.values
```

```
array([list([1, 2, 3]), list([4, 5]), list([6]), 'Hello World'],
      dtype=object)
```

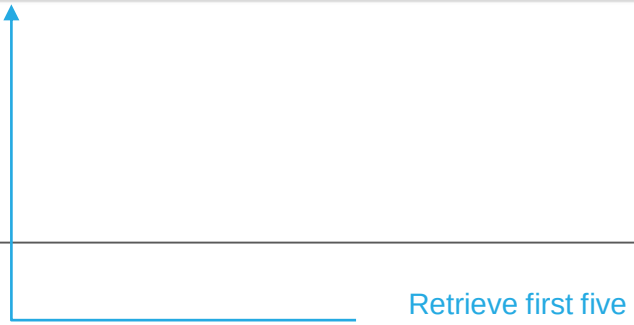
Accessing a Series

Access elements using position

Access the element in a series using the index operator '**[]**'

```
# creating simple array
alpha_array = np.array(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'])
alpha_series = pd.Series(alpha_array)
print(alpha_series[:5])
```

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

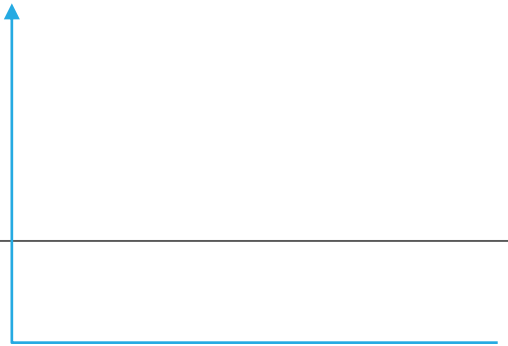


Retrieve first five elements

Access elements using position

```
# creating simple array
alpha_array = np.array(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'])
alpha_series = pd.Series(alpha_array)
print(alpha_series[-5:])
```

```
5    f
6    g
7    h
8    i
9    j
dtype: object
```



Retrieve last five elements

Access elements using index

```
series_index = pd.Series(
        np.array([11,12,13,14,15,16]),
        index=['a', 'b', 'c', 'd', 'e', 'f' ]
)
series_index['c']
```

```
13
```

Use index to access the element

Access elements using index

```
series_index = pd.Series(
        np.array([11,12,13,14,15,16]),
        index=['a', 'b', 'c', 'd', 'e', 'f' ]
)
series_index[['c','a','b']]
```

```
c    13
a    11
b    12
dtype: int32
```

Retrieve multiple elements using a list of indices

Filtering a Series

Filter the values

```
# creating simple array
num_array = np.array([11, 12, 13, 14, 15, 16, 17, 18, 19, 20])
num_series = pd.Series(num_array)
num_series[num_series > 15]
```

```
5    16
6    17
7    18
8    19
9    20
dtype: int32
```

Filter all the values that are greater than 15

Filter the values

```
# create a series from a dictionary
word_series = pd.Series({'C':1, 'C++':3, 'Python':6, 'Data':4})

word_series[word_series > 3]
```

```
Python    6
Data      4
dtype: int64
```

Filter all the words whose length is greater than 3

Arithmetic Operations

Scalar multiplication

```
# creating simple array
num_array = np.array([1, 2, 3])
num_series = pd.Series(num_array)
num_series*2
```

```
0    2
1    4
2    6
dtype: int32
```

Use '*' operator to perform multiplication

One can also use the series .multiply() method to perform the multiplication operation

Multiplication of two series

The series .multiply() method returns the element-wise multiplication of the two series

```
# create two series
odd_series = pd.Series([1,3,5,7])
even_series = pd.Series([2,4,6,8])

# multiply both the series
multi_odd_even = odd_series.multiply(even_series)
print(multi_odd_even)
```

```
0     2
1    12
2    30
3    56
dtype: int64
```

Addition of two series

```
# create two arrays
num_array_1 = np.array([1, 2, 3])
num_series_1 = pd.Series(num_array_1)

num_array_2 = np.array([1,2,3])
num_series_2 = pd.Series(num_array_2)

num_series_1+num_series_2
```

```
0    2
1    4
2    6
dtype: int32
```

Use '+' operator to perform addition

Addition of two series

If the length of the two series are different, then the addition of such series shows the null values (NaN) for the indexes where the values are missing in one of the series

```
num_series_1 = pd.Series([2,4,9])
num_series_2 = pd.Series([1,2])

# add the series
num_series_1 + num_series_2
```

```
0    3.0
1    6.0
2    NaN
dtype: float64
```

Ranking and Sorting of a Series

Rank the series

```
# creating simple array
num_array = np.array([131, 212, 153, 414, 315, 716, 137, 118, 319, 220])
num_series = pd.Series(num_array)
num_series.rank()
```

```
0     2.0
1     5.0
2     4.0
3     9.0
4     7.0
5    10.0
6     3.0
7     1.0
8     8.0
9     6.0
dtype: float64
```

Returns the rank of the underlying data

By default, the rank() returns the ranking in ascending order

Sort the series by values

The sort_values() method sorts the series by values in the series

```
# create a series
num_series = pd.Series([123, 445, np.nan, 411, 223, 334, 155, np.nan, 314, 210])

num_series.sort_values(ascending = True, na_position = 'last')
```

```
0    123.0
6    155.0
9    210.0
4    223.0
8    314.0
5    334.0
3    411.0
1    445.0
2      NaN
7      NaN
dtype: float64
```

Returns the null values in the last position

Sort the series by values

```
# create a series
num_series = pd.Series([123, 445, np.nan, 411, 223, 334, 155, np.nan, 314, 210])

num_series.sort_values(ascending = False, na_position = 'first')
```

```
2      NaN
7      NaN
1    445.0
3    411.0
5    334.0
8    314.0
4    223.0
9    210.0
6    155.0
0    123.0
dtype: float64
```

'ascending = False' sorts the series in descending order

Returns the null values in the first position

Sort the series by index

```
# create a series
num_series = pd.Series([123, 445, np.nan, 411, 223, 334, 155, np.nan, 314, 210])

# sort the series in descending order based on index
num_series.sort_index(ascending = False)
```

```
9    210.0
8    314.0
7      NaN
6    155.0
5    334.0
4    223.0
3    411.0
2      NaN
1    445.0
0    123.0
dtype: float64
```

Sort the series by index

Check for Null Values

Check for null values

- The .isnull() returns the boolean output indicating the presence of null values

- ‘True’ in the output indicates that the corresponding value is null

```
# create a series
num_series = pd.Series([123, 445, np.nan, 411, 223, 334, 155, np.nan, 314, 210])

num_series.isnull()
```

```
0    False
1    False
2     True
3    False
4    False
5    False
6    False
7     True
8    False
9    False
dtype: bool
```

Check for null values

- The .notnull() returns the boolean output indicating the presence of non-null values

- ‘False’ in the output indicates that the corresponding value is null

```
# create a series
num_series = pd.Series([123, 445, np.nan, 411, 223, 334, 155, np.nan, 314, 210])

num_series.notnull()
```

```
0     True
1     True
2    False
3     True
4     True
5     True
6     True
7    False
8     True
9     True
dtype: bool
```

greatlearning
Learning for Life

Pandas DataFrame

Pandas DataFrame

- A DataFrame is two dimensional data structure. i.e., data is aligned in the tabular manner (rows and columns)

- Features of the DataFrame:
  - Columns can be of different types
  - Size is mutable
  - Axes are labeled (rows and columns)
  - Arithmetic operations on rows and columns

Creating a DataFrame

Create a DataFrame from a single list

```
# list of strings
words_list = ['Python', 'For', 'Data', 'Science']
df_words = pd.DataFrame(words_list)
print(df_words)
```

```
         0
0   Python
1      For
2     Data
3  Science
```

As no column name is passed, by default it returns ‘0’ as column name

Pass a list as a column in ‘df’

Create a DataFrame from a list of lists

```
salary_list = [['John', 30000], ['Alia', 50000], ['Mia', 70000], ['Robin', 50000]]
df_salary = pd.DataFrame(salary_list, columns=['Name', 'Salary'])
print(df_salary)
```

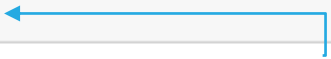
```
    Name  Salary
0   John   30000
1   Alia   50000
2    Mia   70000
3  Robin   50000
```

Pass the list of column names

Create a DataFrame from a dictionary

```
sales_list = {'Month':['Jan', 'Feb', 'March', 'April'],'Sales':[50000,30000,20000,40000]}
df_sales = pd.DataFrame(sales_list)
print(df_sales)
```

```
   Month  Sales
0    Jan  50000
1    Feb  30000
2  March  20000
3  April  40000
```



Keys of the dictionary as column names

Create a DataFrame with index

```
sales_list = {'Month':['Jan', 'Feb', 'March', 'April'],'Sales':[50000,30000,20000, 40000]}
df_sales = pd.DataFrame(sales_list, index=['A', 'B', 'C', 'D'])
print(df_sales)
```

```
   Month  Sales
A    Jan  50000
B    Feb  30000
C  March  20000
D  April  40000
```


Pass a list of index

Create a DataFrame from a list of dictionaries

```
alphabet_dict = [{'A': 101, 'B': 102},{'A': 105, 'B': 110, 'C': 120}]
df_values = pd.DataFrame(alphabet_dict)
print (df_values)
```

```
     A    B      C
0  101  102    NaN
1  105  110  120.0
```

greatlearning
Learning for Life

Reading Data from Different Sources

Read the data from csv file

Use the read_csv() method from pandas to read the data from csv file

```
df = pd.read_csv('Supermarket.csv')
print(df)
```

	Day	Store	Percentage
0	Monday	A	79
1	Monday	B	81
2	Monday	C	74
3	Monday	D	77
4	Monday	E	66
5	Tuesday	A	78
6	Tuesday	B	86
7	Tuesday	C	89
8	Tuesday	D	97

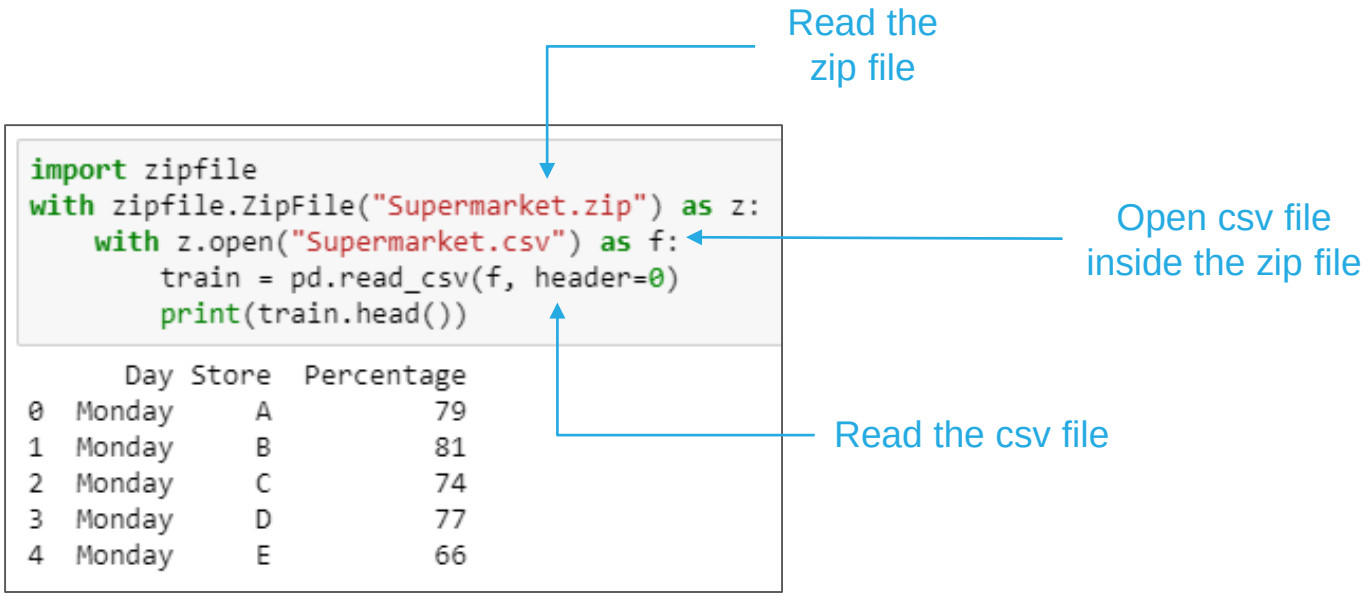
Read the data from xlsx file

Use the read_excel() method from pandas to read the data from xlsx file

```
df_market = pd.read_excel('Supermarket.xlsx')
print(df_market)
```

```
          Day Store  Percentage
0      Monday     A          79
1      Monday     B          81
2      Monday     C          74
3      Monday     D          77
4      Monday     E          66
5     Tuesday     A          78
6     Tuesday     B          86
7     Tuesday     C          89
8     Tuesday     D          97
```

Read the data from zip file



Read the data from text file

Use the read_csv() method from pandas to read the data from text file

```
df = pd.read_csv("Supermarket.txt", sep = "\t")
df
```

	Day	Store	Percentage
0	Monday	A	79
1	Monday	B	81
2	Monday	C	74
3	Monday	D	77
4	Monday	E	66
5	Tuesday	A	78
6	Tuesday	B	86

Read the data from json file

Use the read_json() method from pandas to read the data from json file

```
df = pd.read_json('Supermarket.json')
df
```

	Day	Store	Percentage
0	Monday	A	79
1	Monday	B	81
2	Monday	C	74
3	Monday	D	77
4	Monday	E	66
5	Tuesday	A	78
6	Tuesday	B	86

Read the data from xml file

greatlearning
Learning for Life

Import package to read xml file

```
import xml.etree.ElementTree as ET

tree = ET.parse("sales.xml")
root = tree.getroot()

df_col = ["Day", "Store", "Percentage"]
rows = []

for node in root:
    day = node.attrib.get("day")
    store = node.find("store").text if node is not None else None
    percentage = node.find("percentage").text if node is not None else None

    rows.append({"Day": day, "Store": store,
                 "Percentage": percentage})

xml_df = pd.DataFrame(rows, columns = df_col)
xml_df
```

Extract the xml file

Assign column names of the output DataFrame

Append each observation in the data to ‘rows’

Create a DataFrame ‘xml_df’

Read the data from html file

Use the read_html() method from pandas to read the data from html file

```
df_supermarket = pd.read_html('Supermarket.html')
df_supermarket
```

```
[    Unnamed: 0          A      B           C
 0           1        Day  Store  Percentage
 1           2     Monday      A          79
 2           3     Monday      B          81
 3           4     Monday      C          74
 4           5     Monday      D          77
 5           6     Monday      E          66
 6           7    Tuesday      A          78
 7           8    Tuesday      B          86
 8           9    Tuesday      C          89
 9          10    Tuesday      D          97
 10         11    Tuesday      E          86
```

greatlearning
Learning for Life

DataFrame Manipulations

Read the data from xlsx file

We use the following DataFrame for further explanations

```
df_market = pd.read_excel('Supermarket.xlsx')
print(df_market)
```

```
          Day Store  Percentage
0      Monday     A          79
1      Monday     B          81
2      Monday     C          74
3      Monday     D          77
4      Monday     E          66
5     Tuesday     A          78
6     Tuesday     B          86
7     Tuesday     C          89
8     Tuesday     D          97
```

Display the first five rows

The head() method displays the first five rows of the data

```
df_market.head()
```

	Day	Store	Percentage
0	Monday	A	79
1	Monday	B	81
2	Monday	C	74
3	Monday	D	77
4	Monday	E	66

Display the last five rows

The tail() method displays the last five rows of the data

```
df_market.tail()
```

	Day	Store	Percentage
20	Friday	A	70
21	Friday	B	74
22	Friday	C	77
23	Friday	D	89
24	Friday	E	68

greatlearning
Learning for Life

Understanding the Data

Understand the data

- Check the dimension of the data using the shape attribute

```
df_market.shape
```

```
(25, 3)
```

- Check the data type of each variable in the data using the dtypes attribute

```
df_market.dtypes
```

```
Day            object
Store          object
Percentage      int64
dtype: object
```

Understand the data

- The info() method returns the information about the shape, data type and null values in the data
- Here, ‘df_market’ has 3 variables with 25 non-null observations in each
- There are 2 categorical variables (‘Day’ and ‘Store’) and one numeric variable (Percentage)

```
df_market.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 3 columns):
Day           25 non-null object
Store         25 non-null object
Percentage    25 non-null int64
dtypes: int64(1), object(2)
memory usage: 728.0+ bytes
```

Indexing the DataFrame

Access elements in the DataFrame

- Indexing is frequently required in DataFrame. It may serve the purpose of cross tables or pivot tables

- We can either use the .iloc[], the .loc[] or some conditions to retrieve the elements

- The .iloc[] allows us to retrieve the rows and columns by position, and the .loc[] allows us to retrieve the elements by the column or row name

Access elements in the DataFrame

greatlearning
Learning for Life

Example: Create a DataFrame of six students as shown below

```
# create a DataFrame
data = {'Name': ['Dima', 'James', 'Mia', 'Emity', 'Roben', 'John', 'Jordan'],
        'Score': [12,19,15,10,17,8,17],
        'Attempts' : [3,2,1,3,2,1,2],
        'Qualify' : ['Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']}

df_students = pd.DataFrame(data)
print(df_students)
```

```
     Name  Score  Attempts Qualify
0    Dima     12         3     Yes
1   James     19         2     Yes
2     Mia     15         1     Yes
3   Emity     10         3      No
4   Roben     17         2     Yes
5    John      8         1      No
6  Jordan     17         2     Yes
```

Access elements in the DataFrame

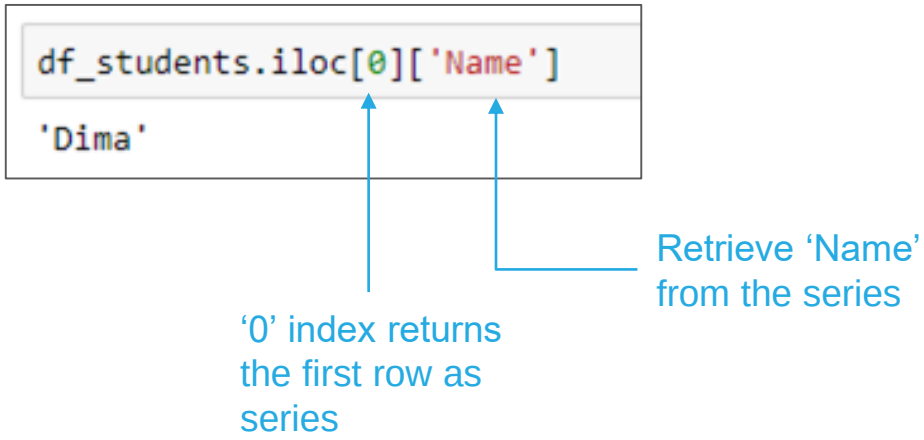
Retrieve the 2nd row by using the .iloc[]

```
df_students.iloc[1]
```

```
Name         James
Score           19
Attempts         2
Qualify        Yes
Name: 1, dtype: object
```

Access elements in the DataFrame

Retrieve the name of the first student using the .iloc[]



Access elements in the DataFrame

Retrieve the 4th, 5th, and 6th row in the DataFrame using the .iloc[]

```
df_students.iloc[3:6]
```

	Name	Score	Attempts	Qualify
3	Emity	10	3	No
4	Roben	17	2	Yes
5	John	8	1	No

Access elements in the DataFrame

Select first three columns by using the position of the columns

```
df_students.iloc[:,:3]
```

	Name	Score	Attempts
0	Dima	12	3
1	James	19	2
2	Mia	15	1
3	Emity	10	3
4	Roben	17	2
5	John	8	1
6	Jordan	17	2

Access elements in the DataFrame

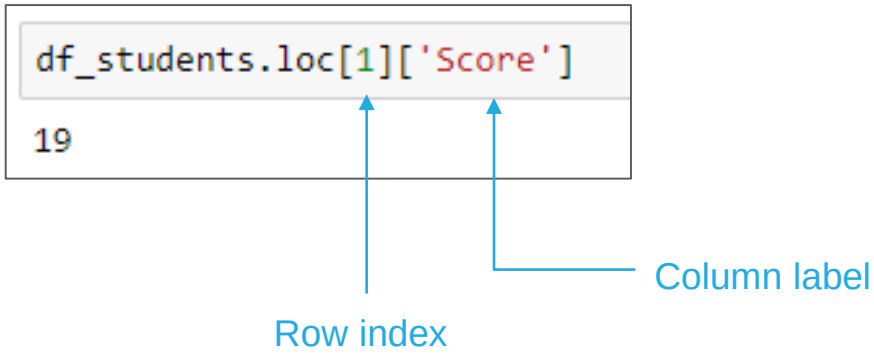
Find the number of attempts corresponding to each student using the .iloc[]

```
df_students.iloc[:,[0,2]]
```

	Name	Attempts
0	Dima	3
1	James	2
2	Mia	1
3	Emity	3
4	Roben	2
5	John	1
6	Jordan	2

Access elements in the DataFrame

- The .loc[] selects the data by the label of the rows and column
- Retrieve the score of the second student using the .loc[]



Access elements in the DataFrame

Retrieve the columns ‘Name’ and ‘Qualify’ for first three students

```
df_students.loc[[0,1,2],['Name','Qualify']]
```

	Name	Qualify
0	Dima	Yes
1	James	Yes
2	Mia	Yes

Access elements in the DataFrame

Retrieve the score for all the students along with their name using the .loc[]

```
df_students[['Name','Score']]
```

	Name	Score
0	Dima	12
1	James	19
2	Mia	15
3	Emity	10
4	Roben	17
5	John	8
6	Jordan	17

Access the elements using different conditions

Retrieve the information of the student whose score is more than 12

```
df_students[df_students.Score > 12]
```

	Name	Score	Attempts	Qualify
1	James	19	2	Yes
2	Mia	15	1	Yes
4	Roben	17	2	Yes
6	Jordan	17	2	Yes

Access the elements using different conditions

Retrieve the students who either have more than two attempts or have qualified the exam

```
df_students[(df_students.Qualify == 'Yes') | (df_students.Attempts > 2)]
```

	Name	Score	Attempts	Qualify
0	Dima	12	3	Yes
1	James	19	2	Yes
2	Mia	15	1	Yes
3	Emity	10	3	No
4	Roben	17	2	Yes
6	Jordan	17	2	Yes

Sorting the DataFrame

Read the data from xlsx file

We use the following dataframe for further manipulations

```
df_market = pd.read_excel('Supermarket.xlsx')
print(df_market)
```

```
          Day Store  Percentage
0      Monday     A          79
1      Monday     B          81
2      Monday     C          74
3      Monday     D          77
4      Monday     E          66
5     Tuesday     A          78
6     Tuesday     B          86
7     Tuesday     C          89
8     Tuesday     D          97
```

Sort the DataFrame

Sort the DataFrame by values in the column ‘Percentage’

```
df_market.sort_values('Percentage')
```

	Day	Store	Percentage
4	Monday	E	66
24	Friday	E	68
20	Friday	A	70
2	Monday	C	74
21	Friday	B	74
22	Friday	C	77

Sort the DataFrame

greatlearning
Learning for Life

Sort the DataFrame by values in the column ‘Percentage’ in the descending order

```
df_market.sort_values('Percentage',ascending=False)
```

	Day	Store	Percentage
8	Tuesday	D	97
13	Wednesday	D	94
23	Friday	D	89
7	Tuesday	C	89
18	Thursday	D	88
11	Wednesday	B	87
6	Tuesday	B	86
9	Tuesday	E	86
12	Wednesday	C	84

Sort the DataFrame

- While sorting the DataFrame by multiple columns, the .sort_values() first sorts the first passed variable and then the next variable

- In this case, the function first sorted the variable ‘percentage’ and then the variable ‘store’

```
df_market.sort_values(['Percentage', 'Store'])
```

	Day	Store	Percentage
4	Monday	E	66
24	Friday	E	68
20	Friday	A	70
21	Friday	B	74
2	Monday	C	74
22	Friday	C	77
3	Monday	D	77
5	Tuesday	A	78

Sort the DataFrame

Sort the DataFrame by values in the columns ‘Store’ and ‘Percentage’

```
df_market.sort_values(['Store','Percentage'])
```

	Day	Store	Percentage
20	Friday	A	70
5	Tuesday	A	78
0	Monday	A	79
15	Thursday	A	80
10	Wednesday	A	81
21	Friday	B	74
1	Monday	B	81
16	Thursday	B	83

Sort the DataFrame

Sort the DataFrame with the condition (Percentage > 85), by index using the sort_index() method

```
1  df_market[df_market.Percentage >85].sort_index(ascending = False)
```

	Day	Store	Percentage
23	Friday	D	89
18	Thursday	D	88
13	Wednesday	D	94
11	Wednesday	B	87
9	Tuesday	E	86
8	Tuesday	D	97
7	Tuesday	C	89
6	Tuesday	B	86

Ranking in the DataFrame

Access elements in the DataFrame

greatlearning
Learning for Life

Example: Create a DataFrame of six students as shown below

```
# create a DataFrame
data = {'Name': ['Dima', 'James', 'Mia', 'Emity', 'Roben', 'John', 'Jordan'],
        'Score': [12,19,15,10,17,8,17],
        'Attempts' : [3,2,1,3,2,1,2],
        'Qualify' : ['Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes']}

df_students = pd.DataFrame(data)
print(df_students)
```

```
     Name  Score  Attempts Qualify
0    Dima     12         3     Yes
1   James     19         2     Yes
2     Mia     15         1     Yes
3   Emity     10         3      No
4   Roben     17         2     Yes
5    John      8         1      No
6  Jordan     17         2     Yes
```

Rank the DataFrame

- Rank the DataFrame by values in the column ‘Score’ using the parameter, method = ‘min’
- If the score is same for two or more observations, then the ‘min’ method assigns the minimum rank to all the equal scores
- Here it assigned the rank ‘5’ to the score = 17

```
df_students['rank'] = df_students.Score.rank(method='min')
df_students
```

	Name	Score	Attempts	Qualify	rank
0	Dima	12	3	Yes	3.0
1	James	19	2	Yes	7.0
2	Mia	15	1	Yes	4.0
3	Emity	10	3	No	2.0
4	Roben	17	2	Yes	5.0
5	John	8	1	No	1.0
6	Jordan	17	2	Yes	5.0

Rank the DataFrame

- Rank the DataFrame by values in the column ‘Score’ using the parameter, method = ‘max’
- If the score is same for two or more observations, then the ‘max’ method assigns the maximum rank to all the equal scores
- Here it assigned the rank ‘6’ to the score = 17

```
df_students['rank'] = df_students.Score.rank(method='max')
df_students
```

	Name	Score	Attempts	Qualify	rank
0	Dima	12	3	Yes	3.0
1	James	19	2	Yes	7.0
2	Mia	15	1	Yes	4.0
3	Emity	10	3	No	2.0
4	Roben	17	2	Yes	6.0
5	John	8	1	No	1.0
6	Jordan	17	2	Yes	6.0

Rank the DataFrame

- Rank the DataFrame by values in the column ‘Score’ using the parameter, method = ‘dense’
- This method does not skip a rank, like the ‘min’ and ‘max’ method
- Here, it assigned the rank ‘5’ to score = 17, and ‘6’ to next greater score = 19

```
df_students['rank'] = df_students.Score.rank(method='dense')
df_students
```

	Name	Score	Attempts	Qualify	rank
0	Dima	12	3	Yes	3.0
1	James	19	2	Yes	6.0
2	Mia	15	1	Yes	4.0
3	Emity	10	3	No	2.0
4	Roben	17	2	Yes	5.0
5	John	8	1	No	1.0
6	Jordan	17	2	Yes	5.0

Rank the DataFrame

- Rank the DataFrame by values in the column ‘Score’ in descending order

- By default, the method is ‘average’ in the .rank(), and it assigns the average rank to the equal values

- Here, it assigned the rank ‘5.5’ to the same score = 17

```
df_students['rank'] = df_students.Score.rank()
df_students
```

	Name	Score	Attempts	Qualify	rank
0	Dima	12	3	Yes	3.0
1	James	19	2	Yes	7.0
2	Mia	15	1	Yes	4.0
3	Emity	10	3	No	2.0
4	Roben	17	2	Yes	5.5
5	John	8	1	No	1.0
6	Jordan	17	2	Yes	5.5

greatlearning
Learning for Life

Thank You