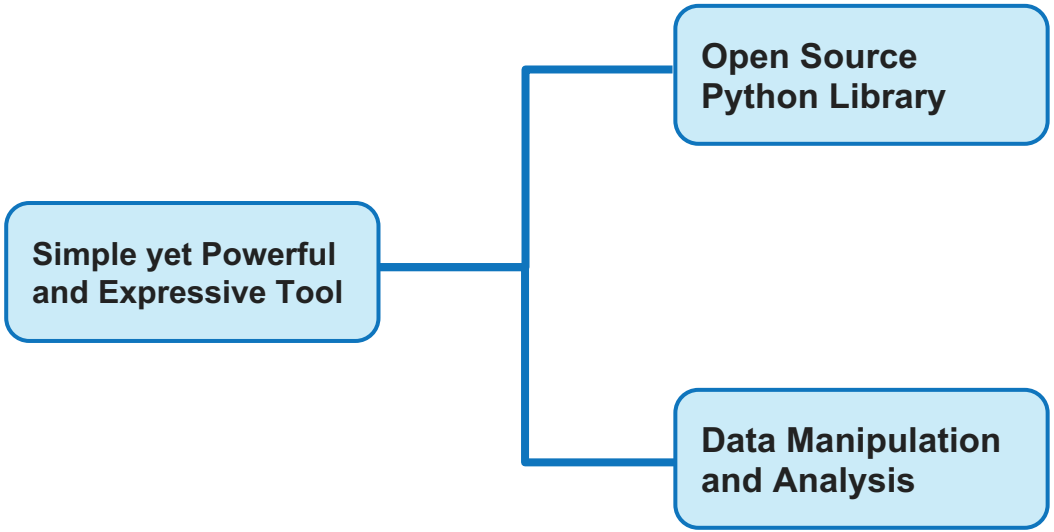
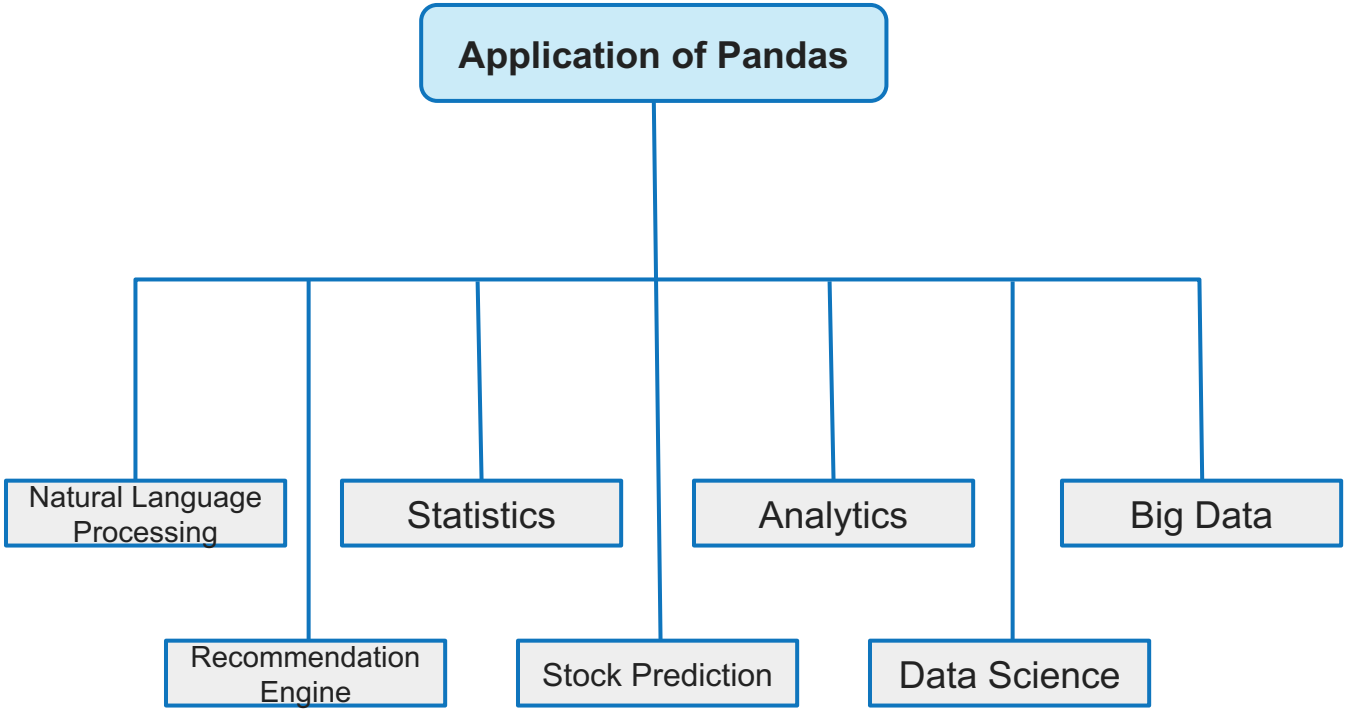
Python Pandas

Python pandas



Application of Pandas



Pandas Vs. Numpy

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

Installation

Open terminal program(for Mac user) or command line(for Windows) and install it using following command:

```
conda install pandas
```

Or

```
pip install pandas
```

Installation

- Alternatively, you can install pandas in a jupyter notebook using below code:

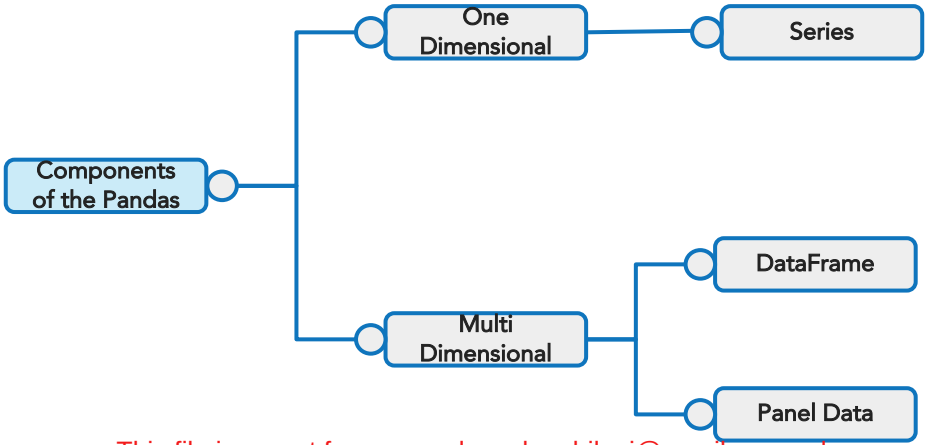
```
!pip install pandas
```

- To import pandas we import it with a shorter name:

```
import pandas as pd
```

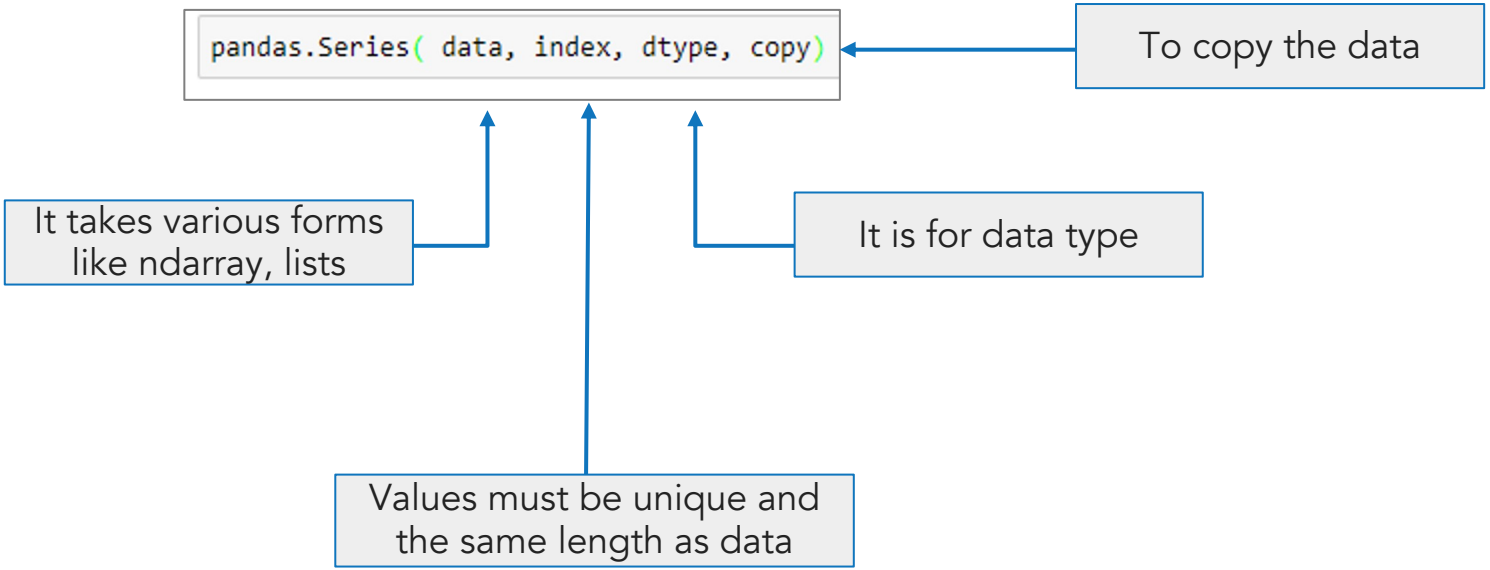
Components of the pandas

- Series and dataframe are two primary components of the pandas
- A series is typically a column, and a data frame is a multi-dimensional table made up of a group of Series



Pandas series

Series can be created using the following constructor:



Pandas series

A pandas series can be created out of 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

Pandas series

We can create our index values 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

Pandas series

```
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
```

String as a row index

Pandas series

- Create a series from python dictionary

```
my_dict = {'a' : 1, 'b': 2, 'c':3}
series_dict = pd.Series(my_dict)
series_dict
```

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

- The **key** becomes the row index while the **value** becomes the value at that row index

Pandas series

Here, the list items remain part of a single row index

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

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

Accessing elements of series

Use the index operator [] to access element in a series

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

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

Retrieve first five elements

Accessing elements of series

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

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

Retrieve last five elements

Access element 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 element to access element

Access element 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 element to access element

Access element 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 index

Filter the values

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

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

Filter all the values that are greater than 15

Multiply each element in the series by 2

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

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

Use '*' operator to perform multiplication

Add two series

```
# create two arrays
array1 = np.array([1, 2, 3])
series1 = pd.Series(array1)

array2 = np.array([1,2,3])
series2 = pd.Series(array2)

series1+series2
```

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

Use '+' operator to perform addition

Ranking in the series

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

It returns the rank of the underlying data

Sort series in ascending order

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

```
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
```

Sort series in descending order

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

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

Sort series based on index

```
# sort in descending order based on index
my_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
```

Check null values using .isnull()

False indicates that the value is not null

```
my_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 null values using .notnull()

False indicates that the value is not null

```
my_series.notnull()
```

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

Pandas dataframe

A data frame is two dimensional data structure, i.e., data aligned in tabular manner(rows and column)

Features of DataFrame:

- **Potentially Columns are of Different Types**
- **Size - Mutable**
- **Labeled axes(rows and column)**
- **Can Perform Arithmetic Operations on Rows and Column**

Reading data from csv file

Use ‘read_csv()’ function from pandas to read 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

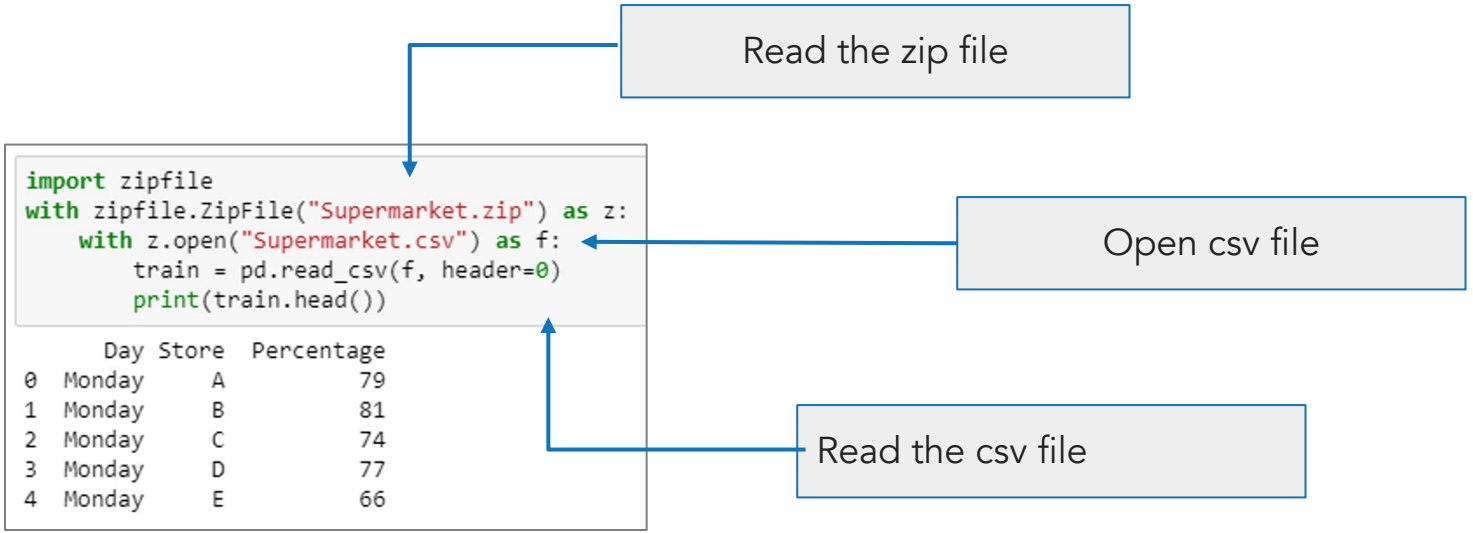
Reading data from xlsx file

Use ‘read_excel()’ function from pandas to read 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
```

Reading data from zip file



Reading data from text file

Use ‘read_csv()’ function from pandas to read 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

Reading data from json file

Use ‘read_json()’ function from pandas to read 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

Reading data from xml file

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
```

Parse or extract the xml file

Assign the column names of output dataframe

Use for loop to extract all the data

Append each observation in data to 'rows'

Create a dataframe 'xml_df'

Reading data from html file

Use ‘read_html()’ function from pandas to read 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
```

Read first five rows of the data

DataFrame.head() will display 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

Read last five rows of the data

DataFrame.tail() will display 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

Know more about data

- Check the dimension of the data

```
df_market.shape
```

```
(25, 3)
```

- Check the data type

```
df_market.dtypes
```

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

Know more about data

- Check the information of the data

```
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
```

Creating data frame using single list

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

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

Creating dataframe using list of list

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

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

Creating dataframe from dictionary of ndarrays

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

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

Creating data frame using arrays

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

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

Create list of index

Creating data frame using list of dictionaries

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

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

Dealing with rows and column

- Use DataFrame.iloc[] method to retrieve rows from a data frame
- iloc[] function allows us to retrieve rows and columns by position

Example: Access the value that is at index 0 in column 'Name'

```
# create a dataframe
data = {'Names':['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 = pd.DataFrame(data, index=[0,1,2,3,4,5,6])
print(df)
# access the value that is at index 0 in column 'Name'
df.iloc[0][0]
```

```
    Names  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

'Dima'
```

Dealing with rows and column

Great Learning

Select row by iloc[] method

```
df.iloc[1]
```

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

Dealing with rows and column

Select 4th and 6th rows

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

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

Dealing with rows and column

Great Learning

Select first three columns by using column number

```
# select first 3 columns
df.iloc[:,:3]
```

	Names	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

Dealing with rows and column

Great Learning

Select first and third column

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

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

Dealing with rows and column

- loc[] function selects data by the label of the rows and column
- Access the value that is at index 1 in column ‘Score’ using loc method

```
print(df.loc[1]['Score'])
```

```
19
```

Dealing with rows and column

Great Learning

Select multiple value by row label and column label using loc

```
df.loc[[1,2,3],['Names','Qualify']]
```

	Names	Qualify
1	James	Yes
2	Mia	Yes
3	Emity	No

Dealing with rows and column

Great Learning

Select two columns from the data frame

```
df[['Names', 'Score']]
```

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

Sort data frame

Sort data frame based on the values of the column

```
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 data frame

Sort data frame based on the values of the column in 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 data frame

Great Learning

Sort data frame based on the values of the multiple columns

```
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 data frame

Note that, while sorting dataframe by multiple columns, pandas sort_value() uses the first variable first and second variable next

```
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

Rank the data frame

Great Learning

Rank the dataframe in pandas on ascending order

```
df_market['Percent_Ranked']=df_market['Percentage'].rank(ascending=1)
df_market
```

	Day	Store	Percentage	Percent_Ranked
0	Monday	A	79	9.0
1	Monday	B	81	12.0
2	Monday	C	74	4.5
3	Monday	D	77	6.5
4	Monday	E	66	1.0
5	Tuesday	A	78	8.0
6	Tuesday	B	86	18.5
7	Tuesday	C	89	22.5
8	Tuesday	D	97	25.0

Rank the data frame

Great Learning

Rank the dataframe in pandas on descending order

```
df_market['Percent_Descending_Ranked']=df_market['Percentage'].rank(ascending=0)
df_market
```

	Day	Store	Percentage	Percent_Ranked	Percent_Descending_Ranked
0	Monday	A	79	9.0	17.0
1	Monday	B	81	12.0	14.0
2	Monday	C	74	4.5	21.5
3	Monday	D	77	6.5	19.5
4	Monday	E	66	1.0	25.0
5	Tuesday	A	78	8.0	18.0
6	Tuesday	B	86	18.5	7.5
7	Tuesday	C	89	22.5	3.5
8	Tuesday	D	97	25.0	1.0
9	Tuesday	E	86	18.5	7.5

Rank the data frame

- Rank the dataframe in pandas by minimum value of the rank
- Rank the data frame in descending order of percentage and if found two percentage are same then assign the minimum rank to both the percentage

```
df_market['Percent_Min_Ranked']=df_market['Percentage'].rank(ascending=0, method='min')
df_market
```

	Day	Store	Percentage	Percent_Ranked	Percent_Descending_Ranked	Percent_Min_Ranked
0	Monday	A	79	9.0	17.0	17.0
1	Monday	B	81	12.0	14.0	13.0
2	Monday	C	74	4.5	21.5	21.0
3	Monday	D	77	6.5	19.5	19.0
4	Monday	E	66	1.0	25.0	25.0
5	Tuesday	A	78	8.0	18.0	18.0
6	Tuesday	B	86	18.5	7.5	7.0

Rank the data frame

- Rank the dataframe in pandas by maximum value of the rank
- Rank the data frame in descending order of percentage and if found two percentage are same then assign the maximum rank to both the percentage

```
df_market['Percent_Max_Ranked']=df_market['Percentage'].rank(ascending=0, method='max')
df_market
```

	Day	Store	Percentage	Percent_Ranked	Percent_Descending_Ranked	Percent_Min_Ranked	Percent_Max_Ranked
0	Monday	A	79	9.0	17.0	17.0	17.0
1	Monday	B	81	12.0	14.0	13.0	15.0
2	Monday	C	74	4.5	21.5	21.0	22.0
3	Monday	D	77	6.5	19.5	19.0	20.0
4	Monday	E	66	1.0	25.0	25.0	25.0
5	Tuesday	A	78	8.0	18.0	18.0	18.0
6	Tuesday	B	86	18.5	7.5	7.0	8.0

Rank the data frame

- Rank the dataframe in pandas by dense rank
- Rank the data frame in descending order of score and if found two scores are same then assign the same rank . Dense rank does not skip any rank (in min and max ranks are skipped)

```
df_market['Percent_Dense_Ranked']=df_market['Percentage'].rank(ascending=0, method='dense')
df_market
```

	Day	Store	Percentage	Percent_Dense_Ranked
0	Monday	A	79	12.0
1	Monday	B	81	10.0
2	Monday	C	74	15.0
3	Monday	D	77	14.0
4	Monday	E	66	18.0
5	Tuesday	A	78	13.0
6	Tuesday	B	86	6.0
7	Tuesday	C	89	3.0

Great Learning

Thank You