LAB 3.2

Introduction To Pandas Module

Here, we are learning data analysis. Pandas is a software library written for the Python programming language that helps with data manipulation and analysis. It offers various data structures which help in the manipulation of data. One of the data structures used is DataFrame. DataFrame is used as a way to store data in a rectangular grid. Similar to how data is stored in Microsoft Excel.

Installation

We can easily install this library using the terminal. Follow the steps given below to install Pandas: Step 1: Go to the Windows PowerShell. Step 2: We can install pandas with the help of pip easily. Head over to the directory where you want to install the pandas and fire the below command: **pip install pandas** This is very simple to install.

```
# !pip install pandas
```

Other way to install

If the above steps do not work for you, here is another way to install: Step 1: Go to your browser. Search for **python unofficial binaries**, or click https://www.lfd.uci.edu/~gohlke/pythonlibs/ and you will be redirected to the binary page. Step 2: To download the library, search for pandas – Ctrl + F will help you. Download pandas library based on your python version. Step 3: After the installation, head over to the directory where you've downloaded the pandas. Step 4: Right-click to open Windows PowerShell. Pip can help us with the installation. Just go to that location in your terminal and run the following command: **pip install pandas** Complete this command by the TAB button so that the location of your folder is mentioned.

import pandas

Once Pandas is installed, import it in your applications by adding the import keyword: **import pandas as pd** Now Pandas is imported and ready to use.

```
import pandas

dict1 = {
    'Name': ["Umar", "Ali", "Usman", "Samra","behzad"],
    'Marks': [80, 72, 77, 65, 15],
    'Class':["1st Year", "2nd Year", "1st Year", "3rd Year", "2nd
Year"]
}

df = pandas.DataFrame(dict1)

df = pandas.DataFrame(dict1)
```

```
df
```

```
     Name  Marks     Class
0    Umar     80  1st Year
1     Ali     72  2nd Year
2   Usman     77  1st Year
3   Samra     65  3rd Year
4  behzad     15  2nd Year
```

Pandas as pd

Pandas is usually imported under the pd alias.

alias:

In Python alias are an alternate name for referring to the same thing. Create an alias with the as keyword while importing: **import pandas as pd** Now the Pandas package can be referred to as pd instead of pandas.

```
import pandas as pd

df = pd.DataFrame(dict1)

df
```

```
     Name  Marks     Class
0    Umar     80  1st Year
1     Ali     72  2nd Year
2   Usman     77  1st Year
3   Samra     65  3rd Year
4  behzad     15  2nd Year
```

Checking Pandas Version

The version string is stored under **version** attribute.

```
pd.__version__
```

```
'2.2.3'
```

Import to CSV

```
df.to_csv('pythonMonday.csv')
```

Remove Index

```
df.to_csv('pythonMondayRemoveIndex.csv', index=False)
```

Show some rows from the Start

```
df.head()
```

```
     Name  Marks     Class
0    Umar     80  1st Year
1     Ali     72  2nd Year
2   Usman     77  1st Year
3   Samra     65  3rd Year
4  behzad     15  2nd Year
```

```
df.head()
```

```
     Name  Marks     Class
0    Umar     80  1st Year
1     Ali     72  2nd Year
2   Usman     77  1st Year
3   Samra     65  3rd Year
4  behzad     15  2nd Year
```

```
df.tail(3)
```

```
     Name  Marks     Class
2   Usman     77  1st Year
3   Samra     65  3rd Year
4  behzad     15  2nd Year
```

```
df.tail()
```

```
     Name  Marks     Class
0    Umar     80  1st Year
1     Ali     72  2nd Year
2   Usman     77  1st Year
3   Samra     65  3rd Year
4  behzad     15  2nd Year
```

```
df.describe()
```

```
           Marks
count   5.000000
mean   61.800000
std    26.771253
min    15.000000
25%    65.000000
50%    72.000000
75%    77.000000
max    80.000000
```

```
classData= pd.read_csv("dataset.csv")
```

```
classData
```

```
   Unnamed: 0    name  Marks     Class  gpa  class
0           0    Umar     80  1st Year  4.0      1
1           1  Behzad     72  2nd Year  3.0      2
2           2     Ali     77  1st Year  3.5      4
3           3   Samra     65  3rd Year  2.4      6
4           4   Usman     15  2nd Year  2.0      5
```

```
classData.drop(0, axis=0)
```

```
   Unnamed: 0    name  Marks     Class  gpa  class
1           1  Behzad     72  2nd Year  3.0      2
2           2     Ali     77  1st Year  3.5      4
3           3   Samra     65  3rd Year  2.4      6
4           4   Usman     15  2nd Year  2.0      5
```

```
classData
```

```
   Unnamed: 0    name  Marks     Class  gpa  class
0           0    Umar     80  1st Year  4.0      1
1           1  Behzad     72  2nd Year  3.0      2
2           2     Ali     77  1st Year  3.5      4
3           3   Samra     65  3rd Year  2.4      6
4           4   Usman     15  2nd Year  2.0      5
```

```
classData['gpa']
```

```
0    4.0
1    3.0
2    3.5
3    2.4
4    2.0
Name: gpa, dtype: float64
```

```
classData['Class']
```

```
0    1st Year
1    2nd Year
2    1st Year
3    3rd Year
4    2nd Year
Name: Class, dtype: object
```

```
classData['class']
```

```
0    1
1    2
2    4
3    6
4    5
Name: class, dtype: int64
```

```
classData['name']
```

```
0      Umar
1    Behzad
2       Ali
3     Samra
4     Usman
Name: name, dtype: object
```

```
type(classData['name'])
```

```
pandas.core.series.Series
```

```
classData['gpa']
```

```
0    4.0
1    3.0
2    3.5
3    2.4
4    2.0
Name: gpa, dtype: float64
```

```
classData['gpa'][0]
```

```
np.float64(4.0)
```

```
classData['gpa'][0]=90
```

```
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\3783213339.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  classData['gpa'][0]=90
```

```
classData
```

	Unnamed: 0	name	Marks	Class	gpa	class
0	0	Umar	80	1st Year	90.0	1
1	1	Behzad	72	2nd Year	3.0	2
2	2	Ali	77	1st Year	3.5	4
3	3	Samra	65	3rd Year	2.4	6
4	4	Usman	15	2nd Year	2.0	5

```
classData['gpa'][4]
```

```
np.float64(2.0)
```

```
classData['gpa'][4]=5
```

```
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\3957951123.py:1:
SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  classData['gpa'][4]=5
```

```
classData["Class"][4]="Second Year"
```

```
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\3056981296.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  classData["Class"][4]="Second Year"
```

```
classData
```

	Unnamed: 0	name	Marks	Class	gpa	class
0	0	Umar	80	1st Year	90.0	1
1	1	Behzad	72	2nd Year	3.0	2
2	2	Ali	77	1st Year	3.5	4
3	3	Samra	65	3rd Year	2.4	6
4	4	Usman	15	Second Year	5.0	5

```
classData["name"][4]
```

```
'Usman'
```

```
classData
```

	Unnamed: 0	name	Marks	Class	gpa	class
0	0	Umar	80	1st Year	90.0	1
1	1	Behzad	72	2nd Year	3.0	2
2	2	Ali	77	1st Year	3.5	4
3	3	Samra	65	3rd Year	2.4	6
4	4	Usman	15	Second Year	5.0	5

```
classData.to_csv('pythonMondayNew.csv')
```

```
classData.to_csv('pythonMondayNewRemoveIndex.csv', index=False)
```

Create Labels

With the index argument, you can name your own labels.

```
classData.index = ["First","Second", "Third", "Fourth", "Fifth"]
```

```
classData
```

```
        Unnamed: 0    name  Marks        Class   gpa  class
First            0    Umar     80     1st Year  90.0      1
Second           1  Behzad     72     2nd Year   3.0      2
Third            2     Ali     77     1st Year   3.5      4
Fourth           3   Samra     65     3rd Year   2.4      6
Fifth            4   Usman     15  Second Year   5.0      5
```

```
classData["name"]["Fifth"]
```

```
'Usman'
```

Pandas

Pandas is an open source data analysis library written in Python. It uses the power and speed of numpy to make data analysis and preprocessing easy for data scientists. **Pandas** is used to analyze data. Python data analysis using pandas is a great start to Exploratory Data Analysis (EDA) for a data scientist.

Pandas Data Structure

Panda has two types of Data Structure: a) Series b) Dataframe

Series

It is a one dimentional array with indexes. It stores a single column or row of data in a Dataframe. It is a one dimensional array capable of holding any type of data.

Dataframe

It is a tabular spreadsheet like structure representing rows each of which contains one or more columns. It is a multi dimensional array capable of holding any type of data. **Series** is like a column, a **DataFrame** is the whole table.

```
classData['gpa']
```

```
First     90.0
Second     3.0
Third      3.5
Fourth     2.4
Fifth      5.0
Name: gpa, dtype: float64
```

```
 type(classData['gpa'])
```

```
pandas.core.series.Series
```

```
classData
```

```
        Unnamed: 0    name  Marks        Class   gpa  class
First            0    Umar     80     1st Year  90.0      1
```

```
Second            1  Behzad     72     2nd Year   3.0      2
Third             2     Ali     77     1st Year   3.5      4
Fourth            3   Samra     65     3rd Year   2.4      6
Fifth             4   Usman     15  Second Year   5.0      5
```

```
type(classData)
```

```
pandas.core.frame.DataFrame
```

```
import numpy as np
```

```
series1= pd.Series(np.random.rand())
```

```
series1
```

```
0    0.863799
dtype: float64
```

```
series1= pd.Series(np.random.rand(30))
```

```
series1
```

```
0     0.490505
1     0.642388
2     0.701803
3     0.427315
4     0.162072
5     0.823410
6     0.007146
7     0.124420
8     0.964552
9     0.691485
10    0.833831
11    0.320095
12    0.537346
13    0.236504
14    0.427526
15    0.726029
16    0.171906
17    0.505061
18    0.558176
19    0.255227
20    0.468784
21    0.236258
22    0.372229
23    0.948222
24    0.285313
25    0.570974
26    0.910419
27    0.877554
28    0.276910
```

```
29    0.403935
dtype: float64
```

```
type(series1)
```

```
pandas.core.series.Series
```

```
newDf = pd.DataFrame(np.random.rand(330,5))
# newDf = pd.DataFrame(np.random.rand(330,5), index=np.arange(330))
```

```
type(newDf)
```

```
pandas.core.frame.DataFrame
```

```
newDf
```

```
            0         1         2         3         4
0    0.428644  0.845682  0.785568  0.887098  0.655208
1    0.771389  0.351744  0.090406  0.778027  0.017863
2    0.563619  0.783972  0.415338  0.971640  0.643814
3    0.975824  0.777449  0.590080  0.534420  0.627735
4    0.525620  0.740227  0.516741  0.533727  0.041541
..        ...       ...       ...       ...       ...
325  0.439123  0.909931  0.741812  0.729477  0.929148
326  0.156356  0.658465  0.396061  0.002193  0.819785
327  0.163087  0.522659  0.529201  0.883081  0.600300
328  0.776183  0.177136  0.876736  0.473567  0.575241
329  0.478343  0.311120  0.342621  0.845378  0.528896

[330 rows x 5 columns]
```

```
newDf.head(2)
```

```
          0         1         2         3         4
0  0.428644  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
```

```
newDf.dtypes
```

```
0    float64
1    float64
2    float64
3    float64
4    float64
dtype: object
```

```
newDf.tail(2)
```

```
            0         1         2         3         4
328  0.776183  0.177136  0.876736  0.473567  0.575241
329  0.478343  0.311120  0.342621  0.845378  0.528896
```

```
newDf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 330 entries, 0 to 329
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       330 non-null    float64
 1   1       330 non-null    float64
 2   2       330 non-null    float64
 3   3       330 non-null    float64
 4   4       330 non-null    float64
dtypes: float64(5)
memory usage: 13.0 KB
```

```
newDf.describe()
```

	0	1	2	3	4
count	330.000000	330.000000	330.000000	330.000000	330.000000
mean	0.508833	0.508141	0.499055	0.496970	0.486661
std	0.292255	0.272950	0.280110	0.289980	0.275773
min	0.001118	0.000351	0.009838	0.000110	0.006137
25%	0.254701	0.289593	0.251430	0.268755	0.259932
50%	0.518867	0.529780	0.497992	0.482537	0.491128
75%	0.762419	0.734844	0.733904	0.771237	0.700446
max	0.999580	0.990092	0.993643	0.999924	0.999810

```
newDf.dtypes
```

```
0    float64
1    float64
2    float64
3    float64
4    float64
dtype: object
```

```
newDf.iloc[0, 0] = "Umar"
```

```
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\392238210.py:1:
FutureWarning: Setting an item of incompatible dtype is deprecated and
will raise an error in a future version of pandas. Value 'Umar' has
dtype incompatible with float64, please explicitly cast to a
compatible dtype first.
  newDf.iloc[0, 0] = "Umar"
```

```
newDf.dtypes
```

```
0     object
1    float64
2    float64
3    float64
```

```
4    float64
dtype: object
```

```
newDf
```

```
            0         1         2         3         4
0        Umar  0.845682  0.785568  0.887098  0.655208
1    0.771389  0.351744  0.090406  0.778027  0.017863
2    0.563619  0.783972  0.415338  0.971640  0.643814
3    0.975824  0.777449  0.590080  0.534420  0.627735
4     0.52562  0.740227  0.516741  0.533727  0.041541
..        ...       ...       ...       ...       ...
325  0.439123  0.909931  0.741812  0.729477  0.929148
326  0.156356  0.658465  0.396061  0.002193  0.819785
327  0.163087  0.522659  0.529201  0.883081  0.600300
328  0.776183  0.177136  0.876736  0.473567  0.575241
329  0.478343  0.311120  0.342621  0.845378  0.528896

[330 rows x 5 columns]
```

```
newDf.index
```

```
RangeIndex(start=0, stop=330, step=1)
```

```
newDf.columns
```

```
RangeIndex(start=0, stop=5, step=1)
```

```
newDf.to_numpy()
```

```
array([['Umar', 0.8456821947437894, 0.7855679557200089,
        0.8870980510759304, 0.6552082421872513],
       [0.7713890033519027, 0.3517441707532226, 0.0904061315213649,
        0.7780271654897173, 0.0178633294894054],
       [0.5636187515945378, 0.7839715463074424, 0.41533796014304836,
        0.9716402421640051, 0.6438144530947115],
       ...,
       [0.16308672038126282, 0.5226594156849904, 0.5292009521323162,
        0.8830814913507018, 0.6002999227158293],
       [0.776183422835709, 0.177136421257725, 0.8767359753400374,
        0.4735673253418551, 0.5752408473426548],
       [0.47834250340156226, 0.31112029439200506, 0.34262138437345857,
        0.8453784051793725, 0.5288955586446525]], dtype=object)
```

```
df.loc[0, 'Name'] = 'Umar'
```

```
newDf.head()
```

```
          0         1         2         3         4
0      Umar  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
```

```
3  0.975824  0.777449  0.590080  0.534420  0.627735
4   0.52562  0.740227  0.516741  0.533727  0.041541

newDf.T

        0         1         2         3         4         5         6
\
0      Umar  0.771389  0.563619  0.975824   0.52562   0.93062
0.008113
1  0.845682  0.351744  0.783972  0.777449  0.740227  0.135287
0.715199
2  0.785568  0.090406  0.415338   0.59008  0.516741  0.478412
0.219151
3  0.887098  0.778027   0.97164   0.53442  0.533727  0.656286
0.452222
4  0.655208  0.017863  0.643814  0.627735  0.041541  0.255964
0.229701

        7         8         9    ...       320       321       322
323  \
0  0.145869  0.351346  0.570656  ...  0.165955  0.621539  0.650114
0.886066
1  0.667175  0.295342  0.738414  ...  0.246964  0.307061  0.523363
0.735252
2  0.616049  0.115213  0.361927  ...  0.575118  0.660315  0.356277
0.193276
3  0.507092  0.313166  0.977326  ...  0.903633  0.546056  0.048854
0.212792
4  0.695681  0.692118  0.203833  ...  0.992035  0.860566  0.097351
0.600351

        324       325       326       327       328       329
0  0.327249  0.439123  0.156356  0.163087  0.776183  0.478343
1    0.8256  0.909931  0.658465  0.522659  0.177136   0.31112
2  0.173012  0.741812  0.396061  0.529201  0.876736  0.342621
3  0.285449  0.729477  0.002193  0.883081  0.473567  0.845378
4  0.456484  0.929148  0.819785    0.6003  0.575241  0.528896

[5 rows x 330 columns]

newDf.sort_index(axis=0)

            0         1         2         3         4
0        Umar  0.845682  0.785568  0.887098  0.655208
1    0.771389  0.351744  0.090406  0.778027  0.017863
2    0.563619  0.783972  0.415338  0.971640  0.643814
3    0.975824  0.777449  0.590080  0.534420  0.627735
4     0.52562  0.740227  0.516741  0.533727  0.041541
..        ...       ...       ...       ...       ...
325  0.439123  0.909931  0.741812  0.729477  0.929148
```

```
326  0.156356  0.658465  0.396061  0.002193  0.819785
327  0.163087  0.522659  0.529201  0.883081  0.600300
328  0.776183  0.177136  0.876736  0.473567  0.575241
329  0.478343  0.311120  0.342621  0.845378  0.528896

[330 rows x 5 columns]
```

```
newDf.sort_index(axis=0, ascending=False)
```

```
            0         1         2         3         4
329  0.478343  0.311120  0.342621  0.845378  0.528896
328  0.776183  0.177136  0.876736  0.473567  0.575241
327  0.163087  0.522659  0.529201  0.883081  0.600300
326  0.156356  0.658465  0.396061  0.002193  0.819785
325  0.439123  0.909931  0.741812  0.729477  0.929148
..        ...       ...       ...       ...       ...
4     0.52562  0.740227  0.516741  0.533727  0.041541
3    0.975824  0.777449  0.590080  0.534420  0.627735
2    0.563619  0.783972  0.415338  0.971640  0.643814
1    0.771389  0.351744  0.090406  0.778027  0.017863
0        Umar  0.845682  0.785568  0.887098  0.655208

[330 rows x 5 columns]
```

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

```
            0         1         2         3         4
0        Umar  0.845682  0.785568  0.887098  0.655208
1    0.771389  0.351744  0.090406  0.778027  0.017863
2    0.563619  0.783972  0.415338  0.971640  0.643814
3    0.975824  0.777449  0.590080  0.534420  0.627735
4     0.52562  0.740227  0.516741  0.533727  0.041541
..        ...       ...       ...       ...       ...
325  0.439123  0.909931  0.741812  0.729477  0.929148
326  0.156356  0.658465  0.396061  0.002193  0.819785
327  0.163087  0.522659  0.529201  0.883081  0.600300
328  0.776183  0.177136  0.876736  0.473567  0.575241
329  0.478343  0.311120  0.342621  0.845378  0.528896

[330 rows x 5 columns]
```

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

```
            4         3         2         1         0
0    0.655208  0.887098  0.785568  0.845682      Umar
1    0.017863  0.778027  0.090406  0.351744  0.771389
2    0.643814  0.971640  0.415338  0.783972  0.563619
3    0.627735  0.534420  0.590080  0.777449  0.975824
4    0.041541  0.533727  0.516741  0.740227   0.52562
..        ...       ...       ...       ...       ...
325  0.929148  0.729477  0.741812  0.909931  0.439123
```

```
326  0.819785  0.002193  0.396061  0.658465  0.156356
327  0.600300  0.883081  0.529201  0.522659  0.163087
328  0.575241  0.473567  0.876736  0.177136  0.776183
329  0.528896  0.845378  0.342621  0.311120  0.478343

[330 rows x 5 columns]
```

```
newDf.head()
```

```
          0         1         2         3         4
0      Umar  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
3  0.975824  0.777449  0.590080  0.534420  0.627735
4   0.52562  0.740227  0.516741  0.533727  0.041541
```

```
newDf.iloc[1, 0]
```

```
0.7713890033519027
```

```
type(newDf.iloc[:, 0])
```

```
pandas.core.series.Series
```

```
newDf
```

```
            0         1         2         3         4
0        Umar  0.845682  0.785568  0.887098  0.655208
1    0.771389  0.351744  0.090406  0.778027  0.017863
2    0.563619  0.783972  0.415338  0.971640  0.643814
3    0.975824  0.777449  0.590080  0.534420  0.627735
4     0.52562  0.740227  0.516741  0.533727  0.041541
..        ...       ...       ...       ...       ...
325  0.439123  0.909931  0.741812  0.729477  0.929148
326  0.156356  0.658465  0.396061  0.002193  0.819785
327  0.163087  0.522659  0.529201  0.883081  0.600300
328  0.776183  0.177136  0.876736  0.473567  0.575241
329  0.478343  0.311120  0.342621  0.845378  0.528896

[330 rows x 5 columns]
```

view

```
newDfView=newDf
```

```
newDfView
```

```
            0         1         2         3         4
0        Umar  0.845682  0.785568  0.887098  0.655208
1    0.771389  0.351744  0.090406  0.778027  0.017863
2    0.563619  0.783972  0.415338  0.971640  0.643814
```

```
3    0.975824  0.777449  0.590080  0.534420  0.627735
4     0.52562  0.740227  0.516741  0.533727  0.041541
..        ...       ...       ...       ...       ...
325  0.439123  0.909931  0.741812  0.729477  0.929148
326  0.156356  0.658465  0.396061  0.002193  0.819785
327  0.163087  0.522659  0.529201  0.883081  0.600300
328  0.776183  0.177136  0.876736  0.473567  0.575241
329  0.478343  0.311120  0.342621  0.845378  0.528896

[330 rows x 5 columns]
```

```
newDfView[0][0]=765
```

```
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\4063526219.py:1:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas
3.0!
You are setting values through chained assignment. Currently this
works in certain cases, but when using Copy-on-Write (which will
become the default behaviour in pandas 3.0) this will never work to
update the original DataFrame or Series, because the intermediate
object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a
DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the
assignment in a single step and ensure this keeps updating the
original `df`.

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy

  newDfView[0][0]=765
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\4063526219.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  newDfView[0][0]=765
```

```
newDf.head()
```

```
          0         1         2         3         4
0       765  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
```

```
3  0.975824  0.777449  0.590080  0.534420  0.627735
4   0.52562  0.740227  0.516741  0.533727  0.041541
```

```
newDfView.head()
```

```
          0         1         2         3         4
0       765  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
3  0.975824  0.777449  0.590080  0.534420  0.627735
4   0.52562  0.740227  0.516741  0.533727  0.041541
```

```
newDfCopy=newDf.copy()
```

```
newDfCopy[0][0]=656
```

```
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\2532441962.py:1:
FutureWarning: ChainedAssignmentError: behaviour will change in pandas
3.0!
You are setting values through chained assignment. Currently this
works in certain cases, but when using Copy-on-Write (which will
become the default behaviour in pandas 3.0) this will never work to
update the original DataFrame or Series, because the intermediate
object on which we are setting values will behave as a copy.
A typical example is when you are setting values in a column of a
DataFrame, like:

df["col"][row_indexer] = value

Use `df.loc[row_indexer, "col"] = values` instead, to perform the
assignment in a single step and ensure this keeps updating the
original `df`.

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy

  newDfCopy[0][0]=656
C:\Users\PMLS\AppData\Local\Temp\ipykernel_100812\2532441962.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  newDfCopy[0][0]=656
```

```
newDf.head()
```

```
          0         1         2         3         4
0       765  0.845682  0.785568  0.887098  0.655208
```

```
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
3  0.975824  0.777449  0.590080  0.534420  0.627735
4   0.52562  0.740227  0.516741  0.533727  0.041541
```

```
newDfCopy.head()
```

```
          0         1         2         3         4
0       656  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
3  0.975824  0.777449  0.590080  0.534420  0.627735
4   0.52562  0.740227  0.516741  0.533727  0.041541
```

loc

```
newDf.head(2)
```

```
          0         1         2         3         4
0       765  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
```

```
newDf.loc[0,0]
```

```
765
```

```
newDf.loc[0,0]=5678
```

```
newDf.head(2)
```

```
          0         1         2         3         4
0      5678  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
```

```
newDf.loc[0,4]
```

```
np.float64(0.6552082421872513)
```

```
newDf.loc[325,1]
```

```
np.float64(0.9099311354265363)
```

```
newDf.loc[0,0]=9875
```

```
newDf.head(2)
```

```
          0         1         2         3         4
0      9875  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
```

```
newDf.columns=list("ABCDE")
```

```
newDf.head(3)
```

```
          A         B         C         D         E
0      9875  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
```

```
newDf.loc[0, "A"]
```

```
9875
```

```
newDf.loc[0,"A"]=65
```

```
newDf.head(3)
```

```
          A         B         C         D         E
0        65  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
```

```
newDf.loc[35,"A"]
```

```
0.45685661234086095
```

```
newDf.drop(0, axis=0).head(3)
```

```
          A         B         C         D         E
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
3  0.975824  0.777449  0.590080  0.534420  0.627735
```

```
newDf.drop("A", axis=1).head(3)
```

```
          B         C         D         E
0  0.845682  0.785568  0.887098  0.655208
1  0.351744  0.090406  0.778027  0.017863
2  0.783972  0.415338  0.971640  0.643814
```

```
newDf.head(3)
```

```
          A         B         C         D         E
0        65  0.845682  0.785568  0.887098  0.655208
1  0.771389  0.351744  0.090406  0.778027  0.017863
2  0.563619  0.783972  0.415338  0.971640  0.643814
```

```
newDf = newDf.drop("A", axis=1)
```

```
newDf.head(3)
```

```
          B         C         D         E
0  0.845682  0.785568  0.887098  0.655208
1  0.351744  0.090406  0.778027  0.017863
2  0.783972  0.415338  0.971640  0.643814
```

```
newDf.loc[[1,2],["C","D"]]
```

```
          C         D
1  0.090406  0.778027
2  0.415338  0.971640
```

```
newDf.head(3)
```

```
          B         C         D         E
0  0.845682  0.785568  0.887098  0.655208
1  0.351744  0.090406  0.778027  0.017863
2  0.783972  0.415338  0.971640  0.643814
```

```
newDf.loc[:,["C","D"]]
```

```
            C         D
0    0.785568  0.887098
1    0.090406  0.778027
2    0.415338  0.971640
3    0.590080  0.534420
4    0.516741  0.533727
..        ...       ...
325  0.741812  0.729477
326  0.396061  0.002193
327  0.529201  0.883081
328  0.876736  0.473567
329  0.342621  0.845378

[330 rows x 2 columns]
```

```
newDf.loc[[1,2],:]
```

```
          B         C         D         E
1  0.351744  0.090406  0.778027  0.017863
2  0.783972  0.415338  0.971640  0.643814
```

```
newDf.loc[:5,["C","D"]]
```

```
          C         D
0  0.785568  0.887098
1  0.090406  0.778027
2  0.415338  0.971640
3  0.590080  0.534420
4  0.516741  0.533727
5  0.478412  0.656286
```

```
newDf.loc[[1,2],:"D"]
```

```
          B         C         D
1  0.351744  0.090406  0.778027
2  0.783972  0.415338  0.971640
```

```
newDf.loc[(newDf["B"]<0.3)]
```

```
            B         C         D         E
5    0.135287  0.478412  0.656286  0.255964
8    0.295342  0.115213  0.313166  0.692118
10   0.036757  0.993643  0.087791  0.962828
19   0.227323  0.255985  0.701267  0.156683
21   0.156348  0.636665  0.390927  0.391794
..        ...       ...       ...       ...
312  0.183028  0.883414  0.775590  0.560528
313  0.011312  0.603112  0.564635  0.743004
315  0.015563  0.031942  0.601035  0.508256
320  0.246964  0.575118  0.903633  0.992035
328  0.177136  0.876736  0.473567  0.575241

[85 rows x 4 columns]
```

```
newDf.loc[(newDf["B"]<0.3) & (newDf["D"]>0.2)]
```

```
            B         C         D         E
5    0.135287  0.478412  0.656286  0.255964
8    0.295342  0.115213  0.313166  0.692118
19   0.227323  0.255985  0.701267  0.156683
21   0.156348  0.636665  0.390927  0.391794
23   0.296591  0.962421  0.584525  0.559112
..        ...       ...       ...       ...
312  0.183028  0.883414  0.775590  0.560528
313  0.011312  0.603112  0.564635  0.743004
315  0.015563  0.031942  0.601035  0.508256
320  0.246964  0.575118  0.903633  0.992035
328  0.177136  0.876736  0.473567  0.575241

[69 rows x 4 columns]
```

```
newDf.head(3)
```

```
          B         C         D         E
0  0.845682  0.785568  0.887098  0.655208
1  0.351744  0.090406  0.778027  0.017863
2  0.783972  0.415338  0.971640  0.643814
```

iloc

```
newDf.iloc[0,0]
```

```
np.float64(0.8456821947437894)
```

```
newDf.iloc[0,0]=555
```

```
newDf.head()
```

```
            B         C         D         E
0  555.000000  0.785568  0.887098  0.655208
```

```
1    0.351744  0.090406  0.778027  0.017863
2    0.783972  0.415338  0.971640  0.643814
3    0.777449  0.590080  0.534420  0.627735
4    0.740227  0.516741  0.533727  0.041541
```

```
newDf.iloc[[0,4],[1,3]]
```

```
          C         E
0  0.785568  0.655208
4  0.516741  0.041541
```

```
newDf.head(4)
```

```
             B         C         D         E
0  555.000000  0.785568  0.887098  0.655208
1    0.351744  0.090406  0.778027  0.017863
2    0.783972  0.415338  0.971640  0.643814
3    0.777449  0.590080  0.534420  0.627735
```

```
newDf.drop(0).head()
```

```
          B         C         D         E
1  0.351744  0.090406  0.778027  0.017863
2  0.783972  0.415338  0.971640  0.643814
3  0.777449  0.590080  0.534420  0.627735
4  0.740227  0.516741  0.533727  0.041541
5  0.135287  0.478412  0.656286  0.255964
```

```
# newDf.drop(0, axis=1)

# newDf.drop("B", axis=1).head()

newDf.drop(["C", "D"], axis=1).head()
```

```
             B         E
0  555.000000  0.655208
1    0.351744  0.017863
2    0.783972  0.643814
3    0.777449  0.627735
4    0.740227  0.041541
```

```
newDf.head()
```

```
             B         C         D         E
0  555.000000  0.785568  0.887098  0.655208
1    0.351744  0.090406  0.778027  0.017863
2    0.783972  0.415338  0.971640  0.643814
3    0.777449  0.590080  0.534420  0.627735
4    0.740227  0.516741  0.533727  0.041541
```

```
newDf.drop(['B', 'D'], axis=1, inplace=True)
```

```
newDf.head()
```

```
          C         E
0  0.785568  0.655208
1  0.090406  0.017863
2  0.415338  0.643814
3  0.590080  0.627735
4  0.516741  0.041541
```

```
newDf.reset_index().head()
```

```
   index         C         E
0      0  0.785568  0.655208
1      1  0.090406  0.017863
2      2  0.415338  0.643814
3      3  0.590080  0.627735
4      4  0.516741  0.041541
```

```
newDf.reset_index(drop=True).head()
```

```
          C         E
0  0.785568  0.655208
1  0.090406  0.017863
2  0.415338  0.643814
3  0.590080  0.627735
4  0.516741  0.041541
```

```
newDf.head()
```

```
          C         E
0  0.785568  0.655208
1  0.090406  0.017863
2  0.415338  0.643814
3  0.590080  0.627735
4  0.516741  0.041541
```

```
newDf.reset_index(drop=True, inplace=True)
```

```
newDf.head()
```

```
          C         E
0  0.785568  0.655208
1  0.090406  0.017863
2  0.415338  0.643814
3  0.590080  0.627735
4  0.516741  0.041541
```

Groupby

The groupby method allows you to group rows of data together and call aggregate functions

```
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
       'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
       'Sales':[200,120,340,124,243,350]}

df = pd.DataFrame(data)

df
```

```
  Company   Person  Sales
0    GOOG      Sam    200
1    GOOG  Charlie    120
2    MSFT      Amy    340
3    MSFT  Vanessa    124
4      FB     Carl    243
5      FB    Sarah    350
```

Now you can use the .groupby() method to group rows together based off of a column name.

For instance let's group based off of Company. This will create a DataFrameGroupBy object:

```
df.groupby('Company')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at
0x000001918A842B30>
```

You can save this object as a new variable:

```
by_comp = df.groupby("Company")
type(by_comp)
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

And then call aggregate methods of the object:

```


by_comp.mean(numeric_only=True)
```

```
         Sales
Company
FB       296.5
GOOG     160.0
MSFT     232.0
```

```
df.groupby('Company').mean(numeric_only=True)
```

```
         Sales
Company
```

```
FB       296.5
GOOG     160.0
MSFT     232.0

by_comp.std(numeric_only=True)

              Sales
Company
FB        75.660426
GOOG      56.568542
MSFT     152.735065

by_comp.min()

          Person  Sales
Company
FB          Carl    243
GOOG     Charlie    120
MSFT         Amy    124

by_comp.max()

          Person  Sales
Company
FB         Sarah    350
GOOG         Sam    200
MSFT     Vanessa    340

by_comp.count()

         Person  Sales
Company
FB            2      2
GOOG          2      2
MSFT          2      2

by_comp.describe()

        Sales
        count   mean         std    min     25%    50%     75%    max
Company
FB        2.0  296.5   75.660426  243.0  269.75  296.5  323.25  350.0
GOOG      2.0  160.0   56.568542  120.0  140.00  160.0  180.00  200.0
MSFT      2.0  232.0  152.735065  124.0  178.00  232.0  286.00  340.0
```