Object Oriented Programming (Using Python)

UNIT- III

Python Libraries:

- Pandas
  - Introduction
  - Pandas features
  - Installing Pandas
  - Object Creation
  - Viewing Data

Prof. R. MADANA MOHANA
Professor, Artificial Intelligence & Data Science
http://rmadanamohana.com/

Pandas (Python Data Analysis)

- Pandas is powerful Python data analysis toolkit
- Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- Pandas is a Python package (library) designed to make working with "relational" or "labeled" data both easy and in-built.
- It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.
- Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language.

Pandas - Main Features

- Easy handling of missing data (represented as `NaN`, `NA`, or `NaT`) in floating point as well as non-floating point data.
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects.
- Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let `Series`, `DataFrame`, etc. automatically align the data for you in computations.
- Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data.

Pandas - Main Features

- Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects.
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets.
- Intuitive merging and joining data sets.
- Flexible reshaping and pivoting (rotating) of data sets.
- Hierarchical labeling of axes (possible to have multiple labels per tick).

Pandas - Main Features

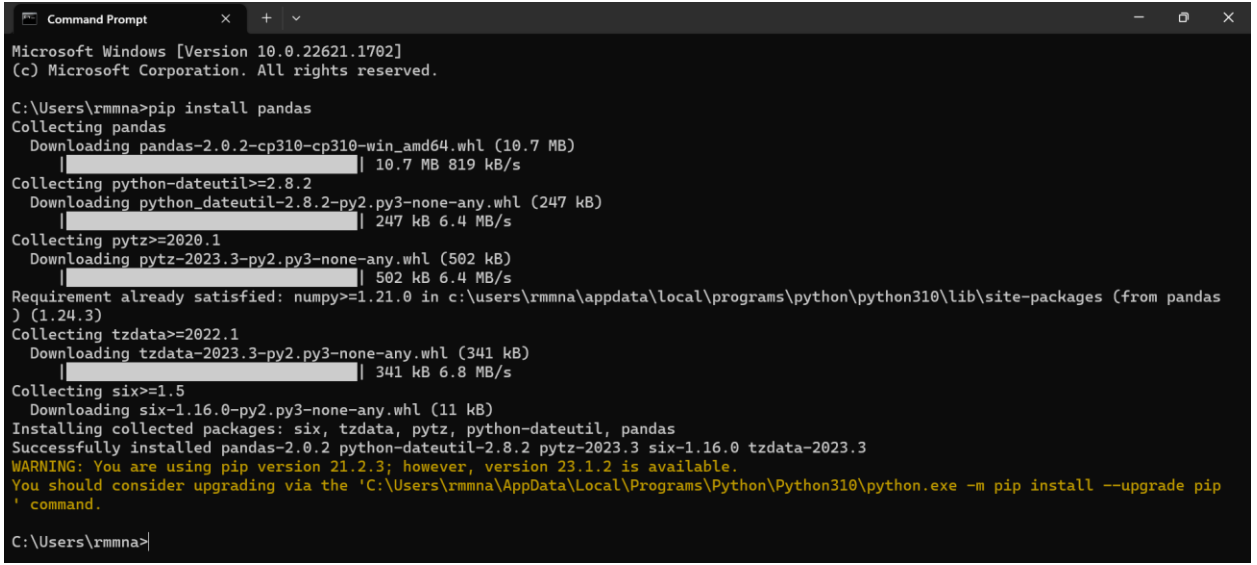
- Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving/loading data from the ultrafast HDF5 format (Hierarchical Data Format).
- Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging.

Installing Pandas

```
# conda
conda install pandas
# PyPI  (Python Package Index )
pip install pandas
```

How to install Pandas using official python IDLE?

- Press the Windows key on your keyboard.
- Type CMD and open Command Prompt. A black terminal should open up.
- Type pip install pandas and hit enter.
- It should start the installation. After you see the "Successfully Installed" message, go back to your IDLE and try importing pandas, it should work.



How to import Pandas

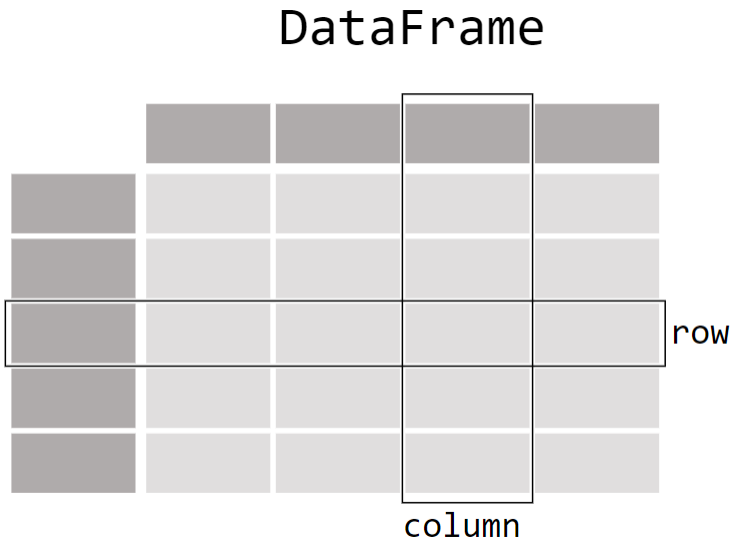
- To access pandas and its functions import it in your Python code like this:

```
import pandas as pd
```

- We shorten the imported name to pd for better readability of code using pandas. This is a widely adopted convention that you should follow so that anyone working with your code can easily understand it.
- https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html#min
- https://pandas.pydata.org/docs/getting_started/index.html

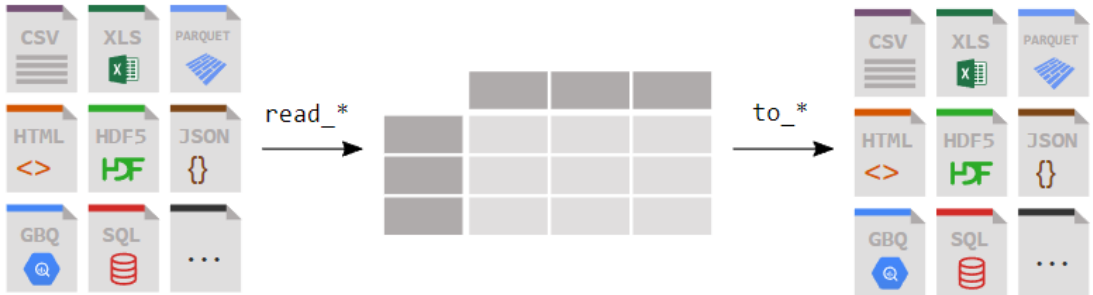
What kind of data does pandas handle?

- When working with tabular data, such as data stored in spreadsheets or databases, pandas is the right tool for us.
- Pandas will help us to explore, clean, and process your data.
- In pandas, a data table is called a DataFrame.



How do we read and write tabular data?

- Pandas supports the integration with many file formats or data sources out of the box (csv, excel, sql, json - JavaScript Object Notation , parquet-open source, column-oriented data file format,…).
- Importing data from each of these data sources is provided by function with the prefix `read_*`. Similarly, the `to_*` methods are used to store data.

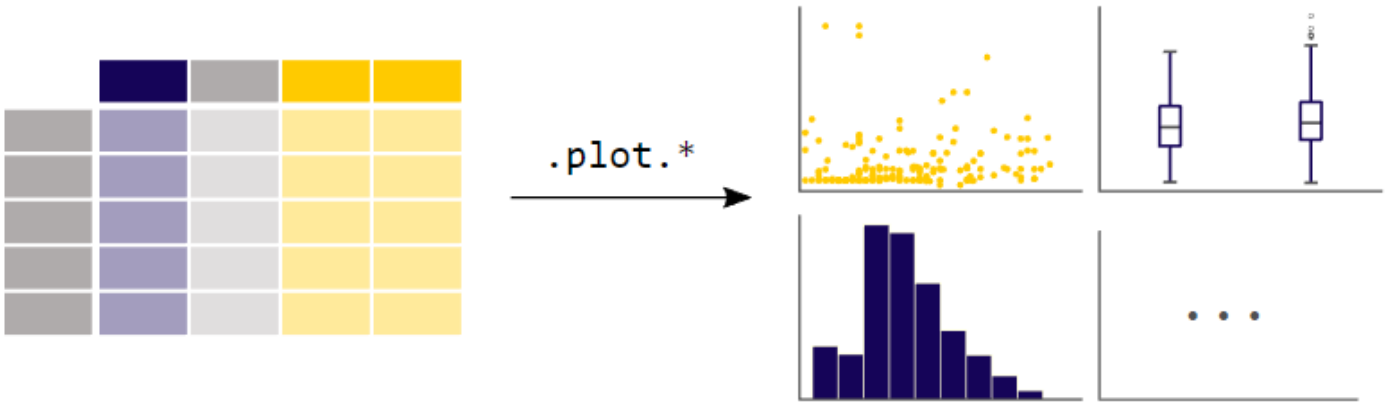


How do we select a subset of a table?

- Selecting or filtering specific rows and/or columns?
- Filtering the data on a condition?
- Methods for slicing, selecting, and extracting the data we need are available in pandas.

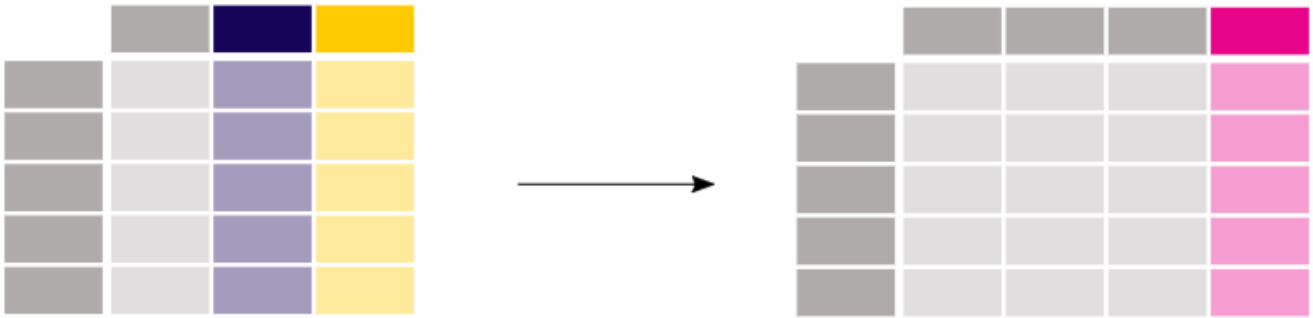
How to create plots in pandas?

- Pandas provides plotting our data out of the box, using the power of Matplotlib.
- We can pick the plot type (scatter, bar, boxplot,…) corresponding to our data.



How to create new columns derived from existing columns?

- There is no need to loop over all rows of our data table to do calculations.
- Data manipulations on a column work elementwise.
- Adding a column to a DataFrame based on existing data in other columns is straightforward.

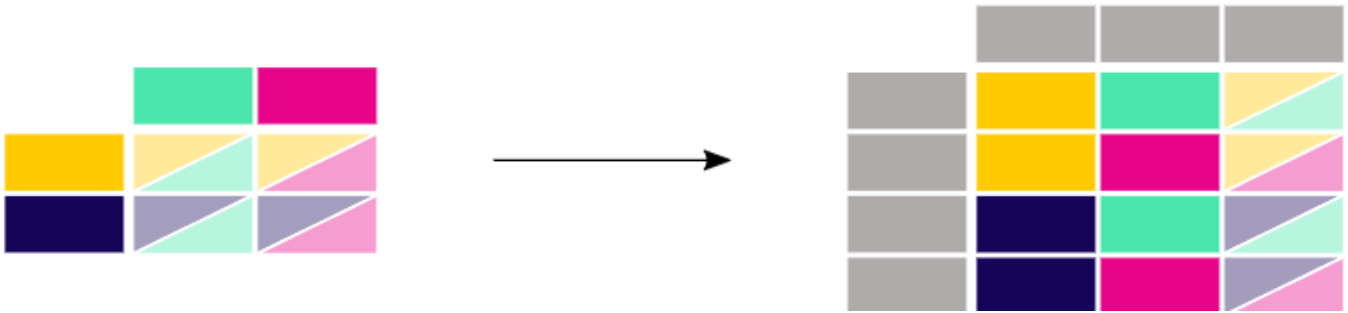


How to calculate summary statistics?

- Basic statistics (mean, median, min, max, counts…) are easily calculable.
- These or custom aggregations can be applied on the entire data set, a sliding window of the data, or grouped by categories.
- The latter is also known as the split-apply-combine approach.

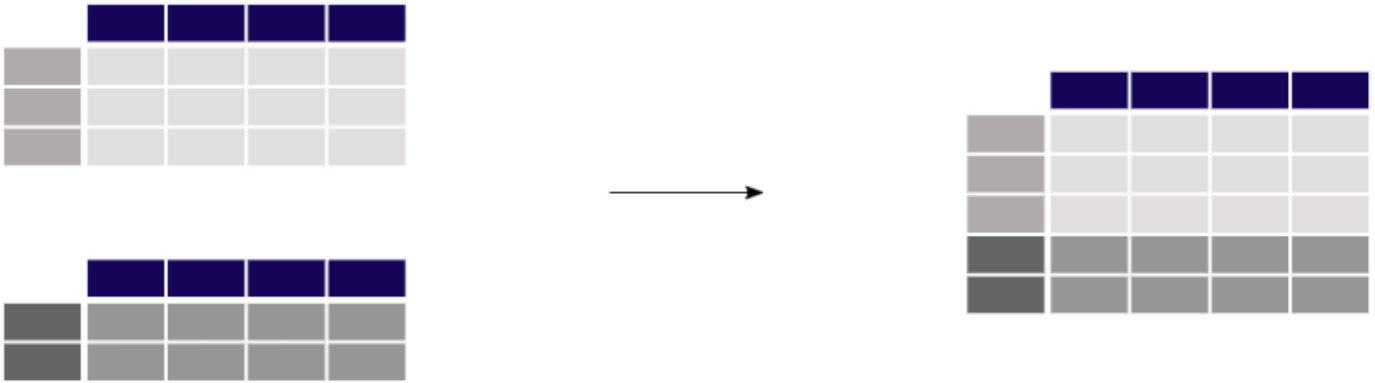
How to reshape the layout of tables?

- Change the structure of our data table in multiple ways.
- We can `melt()` our data table from wide to long/tidy form or `pivot()` from long to wide format.
- With aggregations built-in, a pivot table is created with a single command.



How to combine data from multiple tables?

- Multiple tables can be concatenated both column wise and row wise as database-like join/merge operations are provided to combine multiple tables of data.



How to handle time series data?

- Pandas has great support for time series and has an extensive set of tools for working with dates, times, and time-indexed data.

How to manipulate textual data?

- Data sets do not only contain numerical data.
- Pandas provides a wide range of functions to clean textual data and extract useful information from it.

Quick overview of pandas?

- https://pandas.pydata.org/docs/user_guide/10min.html#min

Object Creation

Creating a Series by passing a list of values, letting pandas create a default integer index:

```
>>> import numpy as np
>>> import pandas as pd
>>> s = pd.Series([1, 3, 5, np.nan, 6, 8])# nan - not a number
>>> s
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

Object Creation

Creating a DataFrame by passing a NumPy array, with a datetime index using date_range() and labeled columns:

```
>>> import numpy as np
>>> import pandas as pd
>>> dates = pd.date_range("20130101", periods=6)
>>> dates
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-
03', '2013-01-04', '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')
```

Object Creation

Creating a DataFrame by passing a NumPy array, with a datetime index using date_range() and labeled columns: cont'd.

```
>>> df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
>>> df
```

```
                   A         B         C         D
2013-01-01 -0.472821  1.541007  1.390524 -0.404981
2013-01-02 -0.229180  1.769937 -0.119053 -0.794888
2013-01-03  3.368224  0.708559 -0.251711  0.702893
2013-01-04 -0.397939  0.539684  1.556475 -0.326841
2013-01-05  1.853638  0.400270  1.168184 -0.363158
2013-01-06 -0.997194 -0.503115  0.190057  0.547042
```

Object Creation

Creating a DataFrame by passing a dictionary of objects that can be converted into a series-like structure:

```
>>> import numpy as np
>>> import pandas as pd
>>> df2 = pd.DataFrame(
    {
        "A": 1.0,
        "B": pd.Timestamp("20130102"),
        "C": pd.Series(1, index=list(range(4)), dtype="float32"),
        "D": np.array([3] * 4, dtype="int32"),
        "E": pd.Categorical(["test", "train", "test", "train"]),
        "F": "foo",
    }
)
```

Object Creation

Creating a DataFrame by passing a dictionary of objects that can be converted into a series-like structure:

```
>>> df2
     A          B    C    D      E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

Object Creation

Creating a DataFrame by passing a dictionary of objects that can be converted into a series-like structure:

- The columns of the resulting DataFrame have different dtypes:

```
>>> df2.dtypes
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

Viewing Data

Use DataFrame.head() and DataFrame.tail() to view the top and bottom rows of the frame respectively:

```
>>> import numpy as np
>>> import pandas as pd
>>> dates = pd.date_range("20230615", periods=6)
>>> dates
DatetimeIndex(['2023-06-15', '2023-06-16', '2023-06-17',
'2023-06-18', '2023-06-19', '2023-06-20'],
dtype='datetime64[ns]', freq='D')
```

Viewing Data

Use DataFrame.head() and DataFrame.tail() to view the top and bottom rows of the frame respectively:

```
>>> df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
>>> df
                   A         B         C         D
2023-06-15  1.647283 -0.659818  0.058287  0.522661
2023-06-16 -0.699142 -0.078138 -1.802049 -0.753840
2023-06-17 -0.652668  0.755713 -1.039440 -0.678967
2023-06-18  0.625232  0.112527 -1.274081  0.724884
2023-06-19 -1.719933  1.858491  0.713219 -0.283876
2023-06-20  1.419357  0.952211 -0.146506 -1.337652
```

Viewing Data

Use DataFrame.head() and DataFrame.tail() to view the top and bottom rows of the frame respectively:

```
>>> df.head()
                   A         B         C         D
2023-06-15  1.647283 -0.659818  0.058287  0.522661
2023-06-16 -0.699142 -0.078138 -1.802049 -0.753840
2023-06-17 -0.652668  0.755713 -1.039440 -0.678967
2023-06-18  0.625232  0.112527 -1.274081  0.724884
2023-06-19 -1.719933  1.858491  0.713219 -0.283876
```

Viewing Data

Use DataFrame.head() and DataFrame.tail() to view the top and bottom rows of the frame respectively:

```
>>> df.tail(3)
                   A         B         C         D
2023-06-18  0.625232  0.112527 -1.274081  0.724884
2023-06-19 -1.719933  1.858491  0.713219 -0.283876
2023-06-20  1.419357  0.952211 -0.146506 -1.337652
```

Viewing Data

Display the DataFrame.index or DataFrame.columns:

```
>>> df.index
DatetimeIndex(['2023-06-15',  '2023-06-16',  '2023-06-17',
'2023-06-18', '2023-06-19', '2023-06-20'],
              dtype='datetime64[ns]', freq='D')
>>> df.columns
Index(['A', 'B', 'C', 'D'], dtype='object')
```

Viewing Data

DataFrame.to_numpy() gives a NumPy representation of the underlying data.

Note that this can be an expensive operation when your DataFrame has columns with different data types, which comes down to a fundamental difference between pandas and NumPy: **NumPy arrays have one dtype for the entire array, while pandas DataFrames have one dtype per column**. When you call DataFrame.to_numpy(), **pandas** will find the NumPy dtype that can hold *all* of the dtypes in the DataFrame. This may end up being object, which requires casting every value to a Python object.

Viewing Data

For df, our DataFrame of all floating-point values, and DataFrame.to_numpy() is fast and doesn't require copying data:

```
>>> df.to_numpy()
array([[ 1.64728338, -0.65981759,  0.05828652,  0.52266126],
       [-0.69914212, -0.07813804, -1.80204948, -0.7538402 ],
       [-0.65266814,  0.7557127 , -1.03944031, -0.67896746],
       [ 0.6252324 ,  0.11252683, -1.27408074,  0.7248838 ],
       [-1.71993319,  1.85849098,  0.7132188 , -0.28387634],
       [ 1.41935705,  0.95221056, -0.14650593, -1.33765214]])
```

Viewing Data

For df2, the DataFrame with multiple dtypes, DataFrame.to_numpy() is relatively expensive:

```
>>> df2
    A      B         C   D    E    F
0  1.0 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
>>> df2.to_numpy()
array([[1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo']],
      dtype=object)
```

Viewing Data

Note

DataFrame.to_numpy() does *not* include the index or column labels in the output.

describe() shows a quick statistic summary of your data:

```
>>> df.describe()
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean   0.073711 -0.431125 -0.687758 -0.233103
std    0.843157  0.922818  0.779887  0.973118
min   -0.861849 -2.104569 -1.509059 -1.135632
25%   -0.611510 -0.600794 -1.368714 -1.076610
50%    0.022070 -0.228039 -0.767252 -0.386188
75%    0.658444  0.041933 -0.034326  0.461706
max    1.212112  0.567020  0.276232  1.071804
```

Viewing Data

Transposing your data:

```
>>> df.T
   2023-06-15  2023-06-16  2023-06-17  2023-06-18  2023-06-19  2023-06-20
A    1.647283   -0.699142   -0.652668    0.625232   -1.719933    1.419357
B   -0.659818   -0.078138    0.755713    0.112527    1.858491    0.952211
C    0.058287   -1.802049   -1.039440   -1.274081    0.713219   -0.146506
D    0.522661   -0.753840   -0.678967    0.724884   -0.283876   -1.337652
```

Viewing Data

DataFrame.sort_index() sorts by an axis:

```
>>> df.sort_index(axis=1, ascending=False)
                   D         C         B         A
2023-06-15  0.522661  0.058287 -0.659818  1.647283
2023-06-16 -0.753840 -1.802049 -0.078138 -0.699142
2023-06-17 -0.678967 -1.039440  0.755713 -0.652668
2023-06-18  0.724884 -1.274081  0.112527  0.625232
2023-06-19 -0.283876  0.713219  1.858491 -1.719933
2023-06-20 -1.337652 -0.146506  0.952211  1.419357
```

Viewing Data

DataFrame.sort_values() sorts by values:

```
>>> df.sort_values(by="B")
                   A         B         C         D
2023-06-15  1.647283 -0.659818  0.058287  0.522661
2023-06-16 -0.699142 -0.078138 -1.802049 -0.753840
2023-06-18  0.625232  0.112527 -1.274081  0.724884
2023-06-17 -0.652668  0.755713 -1.039440 -0.678967
2023-06-20  1.419357  0.952211 -0.146506 -1.337652
2023-06-19 -1.719933  1.858491  0.713219 -0.283876
```

Write a Pandas program to convert a Panda Module Series to Python list and its type.

```
>>> import pandas as pd
# Define a Pandas Series
>>> data = pd.Series([1, 2, 3, 4, 5])
# Convert the Series to a Python list
>>> data_list = data.tolist()
# Print the list and its type
>>> print("List:", data_list)
List: [1, 2, 3, 4, 5]
>>> print("Type:", type(data_list))
Type: <class 'list'>
```

Write a Pandas program to convert a Panda Module Series to Python list and its type.

- In this PROGRAM, we first import the Pandas library.
- Then, we define a Pandas Series ‘data’ with some sample data.
- We then use the ‘tolist()’ function to convert the Series to a Python list, and store the result in the ‘data_list’ variable.
- Finally, we print the list and its type using the ‘print()’ function.