Pandas (Python Module)

Seyyed Ali Shohadaalhosseini

Introduction

What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

What Can Pandas Do?

Pandas gives you answers about the data. Like:

- **Is there a correlation between two or more columns?**
- **What is average value?**
- **Max value?**
- **Min value?**

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Where is the Pandas Codebase?

The source code for Pandas is located at this GitHub repository
https://github.com/pandas-dev/pandas

Let’s get into coding

Look at the front example:

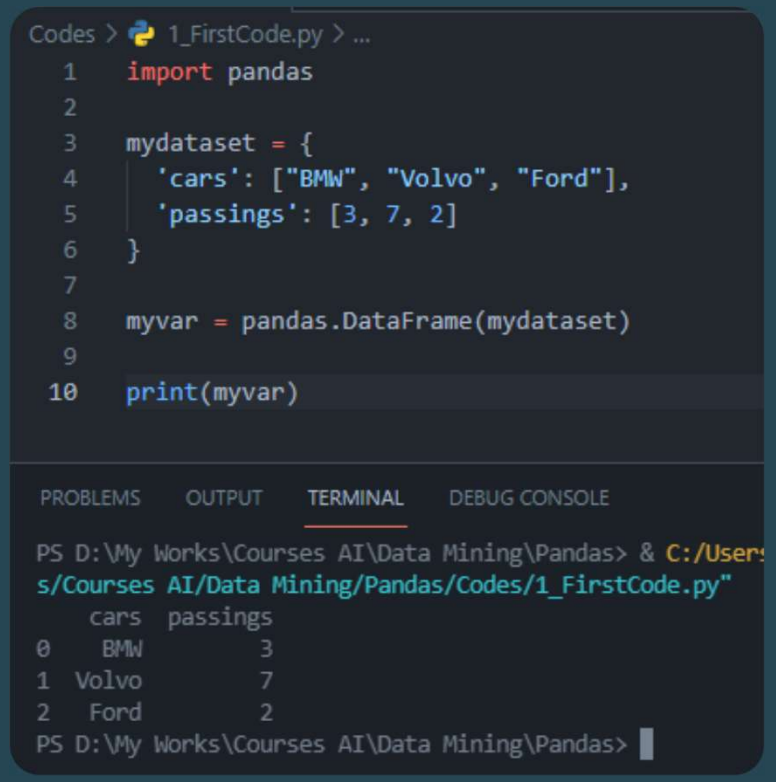
alias: In Python alias are an alternate name for referring to the same thing.

```
1    import pandas as pd
```

Now the Pandas package can be referred to as pd instead of pandas.

The version string is stored under __version__ attribute, So

```
1    import pandas as pd
2
3    print(pd.__version__)
```



Pandas Series

What is a Series?

- A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type.

3_Series.py > ...

```
1   import pandas as pd
2
3   a = [1, 7, 2]
4   myvar = pd.Series(a)
5   print(myvar)
6
```

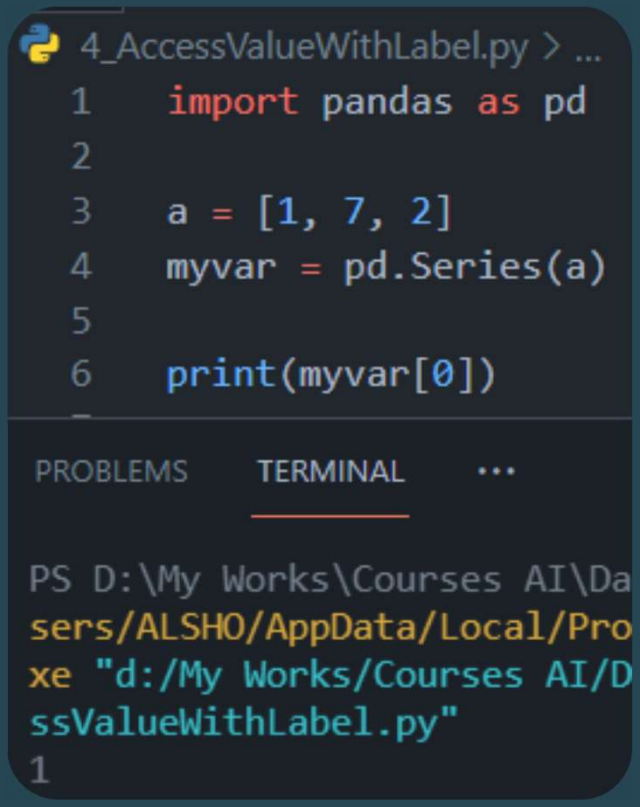
PROBLEMS OUTPUT TERMINAL DEBUG CON

```
PS D:\My Works\Courses AI\Data Mining
a/Local/Programs/Python/Python39/pyth
ning/Pandas/Codes/3_Series.py"
0    1
1    7
2    2
dtype: int64
```

Pandas Series

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc. This label can be used to access a specified value.



Pandas Series

Create Labels

With the index argument, you can name your own labels.
When you have created labels, you can access an item by referring to the label.

5_CreateLabels.py > ...

```
1  import pandas as pd
2
3  a = [1, 7, 2]
4  myvar = pd.Series(a, index = ["x", "y", "z"])
5  print(myvar)
6  print(myvar["y"])
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserv

Install the latest PowerShell for new features and imp

PS D:\My Works\Courses AI\Data Mining\Pandas\Codes> &
thon/Python39/python.exe "d:/My Works/Courses AI/Data
x    1
y    7
z    2
dtype: int64
7
```

Pandas Series

Key/Value Objects as Series
You can also use a key/value object, like a dictionary, when creating a Series.

- The keys of the dictionary become the labels.

6_KeyValueObject.py > ...

```
1  import pandas as pd
2
3  calories = {"day1": 420, "day2": 380, "day3": 390}
4  myvar = pd.Series(calories)
5  print(myvar)
6
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvem

PS D:\My Works\Courses AI\Data Mining\Pandas\Codes> & C:/Us
thon/Python39/python.exe "d:/My Works/Courses AI/Data Minin

day1    420
day2    380
day3    390
dtype: int64
```

Pandas Series

Key/Value Objects as Series

To select only some of the items in the dictionary, use the index argument and specify only the items you want to include in the Series.

7_CreateSeariesWithSpecialKey.py > myvar

```
1  import pandas as pd
2
3  calories = {"day1": 420, "day2": 380, "day3": 390}
4  myvar = pd.Series(calories, index = ["day1", "day2"])
5  print(myvar)
```

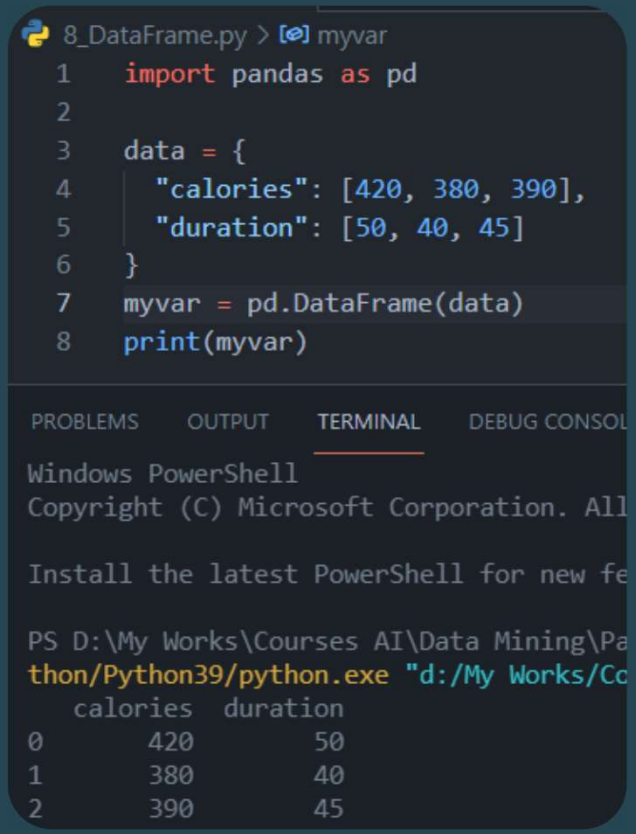
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvement

PS D:\My Works\Courses AI\Data Mining\Pandas\Codes> & C:/Users
thon/Python39/python.exe "d:/My Works/Courses AI/Data Mining/P
pecialKey.py"
day1    420
day2    380
dtype: int64
```

Pandas Series

DataFrames
Data sets in Pandas are usually multi-dimensional tables, called DataFrames. Series is like a column, a DataFrame is the whole table.

What is a DataFrame?
A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.



Pandas DataFrames

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns. Pandas use the *loc* attribute to return one or more specified row(s).

Note: This example returns a Pandas **Series**.

```
9_LocateRow.py > df
1   import pandas as pd
2
3   data = {
4     "calories": [420, 380, 390],
5     "duration": [50, 40, 45]
6   }
7   df = pd.DataFrame(data)
8   print(df.loc[0])
9
```

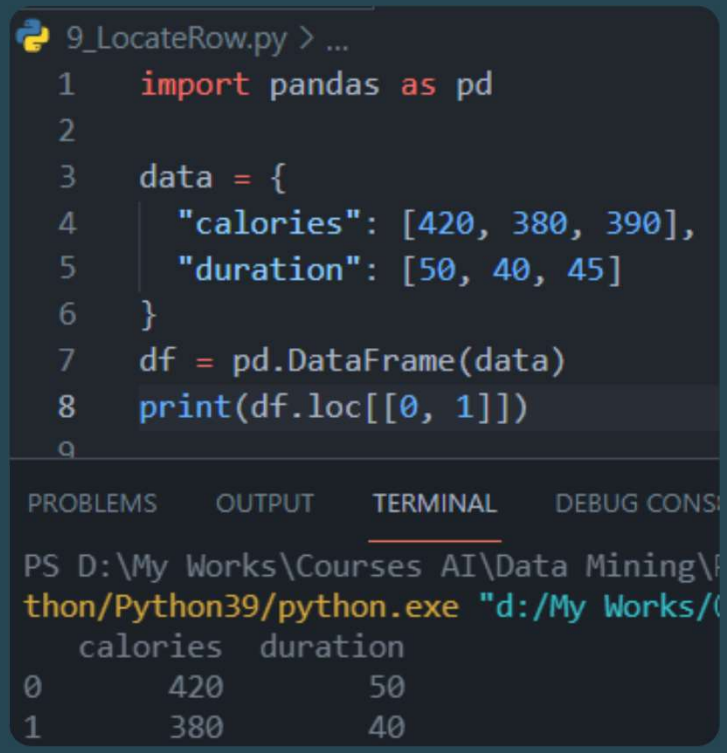
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSO
PS D:\My Works\Courses AI\Data Mining\P
thon/Python39/python.exe "d:/My Works/C
calories    420
duration     50
Name: 0, dtype: int64
```

loc[:] helps to access a group of rows and columns in a dataset, a slice of the dataset, as per our requirement. For instance, if we only want the last 2 rows and the first 3 columns of a dataset, we can access them with the help of loc[:]. We can also access rows and columns based on labels instead of row and column number.

iloc[:] works in a similar manner, just that iloc[:] is not inclusive on both values. So iloc[0:4] would return rows with index 0, 1, 2, and 3, while loc[0:4] would return rows with index 0, 1, 2, 3, and 4. The documentation for iloc[:] can be found here.

Pandas DataFrames

Note: When using [ ], the result is a Pandas DataFrame.



Pandas DataFrames

Named Indexes

With the *index* argument, you can name your own indexes.

Example

Add a list of names to give each row a name:

NOTE: Use the named index in the loc attribute to return the specified row(s).

10_NamedIndexes.py > ...

```
1  import pandas as pd
2
3  data = {
4    "calories": [420, 380, 390],
5    "duration": [50, 40, 45]
6  }
7  df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
8  print(df)
9  print(df.loc["day2"])
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS D:\My Works\Courses AI\Data Mining\Pandas\Codes> & C:/Users/ALS
thon/Python39/python.exe "d:/My Works/Courses AI/Data Mining/Panda
      calories  duration
day1       420        50
day2       380        40
day3       390        45
calories    380
duration     40
Name: day2, dtype: int64
```

Pandas DataFrames

Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame with *read_csv()* method.

Example

In our example we are importing BrutForce.csv. Load a comma separated file (CSV file) into a DataFrame:

11_LoadaDataSetWithPandas.py > ...

```
1  import pandas as pd
2
3  df = pd.read_csv('D:\\My Works\\Projects\\Python Projects\\Project AI BruteFor
4  print(df)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Python

```
PS D:\My Works\Courses AI\Data Mining\Pandas\Codes> & C:/Users/ALSHO/AppData/Local/Programs/Python/Python39/python.exe "d:/My Works/Courses AI/Data Mining/Pandas/Codes/11_LoadaDataSetWithPandas.py"
        Unnamed: 0  'Dst Port'  Protocol  ...  'Idle Max'  'Idle Min'           Label
0               94     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
1               95     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
2               96     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
3               97     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
4               98     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
...            ...         ...       ...  ...         ...         ...             ...
580967      1048570     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
580968      1048571     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
580969      1048572     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
580970      1048573     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
580971      1048574     0.00032  0.352941  ...         0.0         0.0  FTP-BruteForce
```

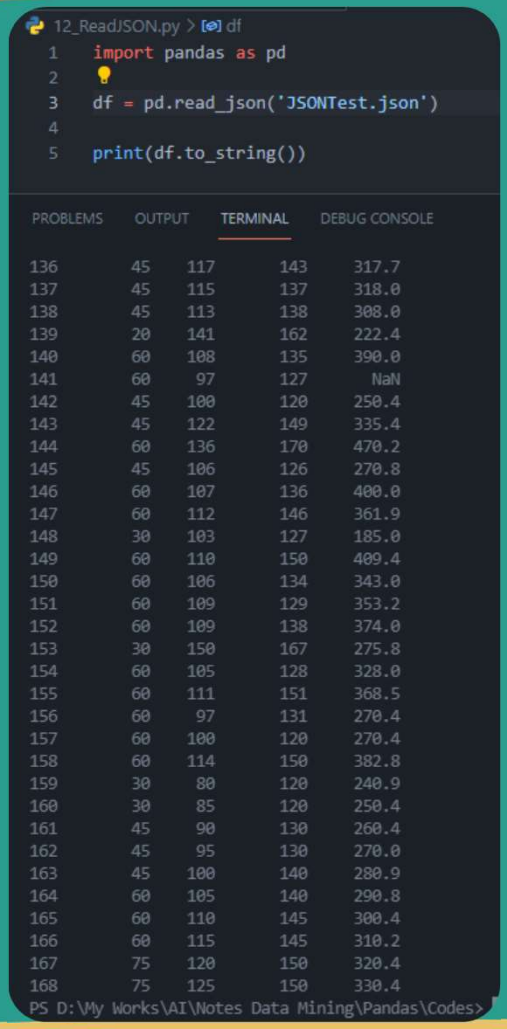
Pandas Read JSON

Big data sets are often stored, or extracted as JSON.

JSON is plain text, but has the format of an object, and is well known in the world of programming, including Pandas.

In our examples we will be using a JSON file called 'JSONTest.json'.

Note: use *to_string()* to print the entire DataFrame.



JSON

JSON = Python Dictionary ☺

JSON objects have the same format as Python dictionaries, So If your JSON code is not in a file, but in a Python Dictionary, you can load it into a DataFrame directly:

Pandas - Analyzing

Viewing the Data

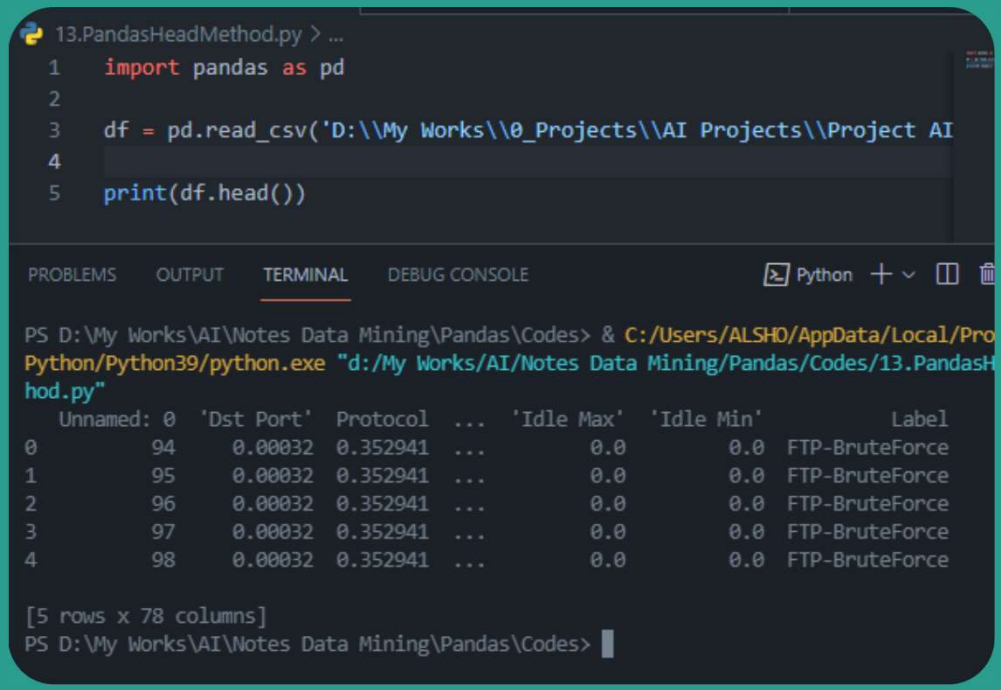
- ***head(n)***

  One of the most used method for getting a quick overview of the DataFrame, is the ***head()*** method. The ***head(n)*** method returns the headers and a specified **n** number of rows, starting from the top.

Note: if the number of rows is not specified, the ***head()*** method will return the top 5 rows.

- ***tail()***

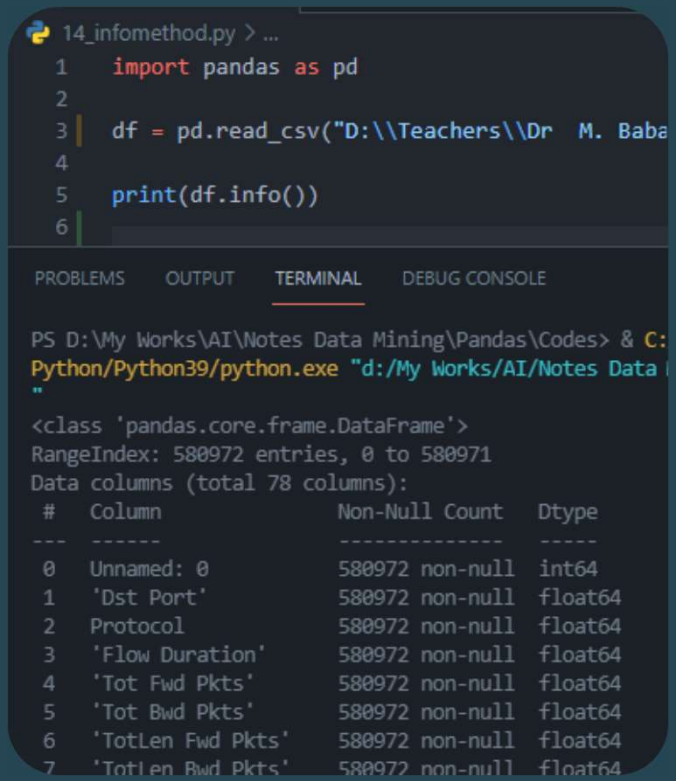
  There is also a ***tail()*** method for viewing the last rows of the DataFrame. The ***tail()*** method returns the headers and a specified number of rows, starting from the bottom.



Pandas - Analyzing

- ***info()***

  The DataFrames object has a method called *info()*, that gives you more information about the data set.



Pandas - Analyzing

- ***describe()***

  describe() is used to generate descriptive statistics of the data in a Pandas DataFrame or Series. It summarizes central tendency and dispersion of the dataset. *describe()* helps in getting a quick overview of the dataset. More details about describe() can be found here.

describe() lists out different descriptive statistical measures for all numerical columns in our dataset. By assigning the include attribute the value 'all', we can get the description to include all columns, including those containing categorical information.

15_describeMethod.py > ...

```
1  import pandas as pd
2
3  df = pd.read_csv("D:\\Teachers\\Dr  M. Babagoli\\Data Mining
4
5  print(df.describe())
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Python

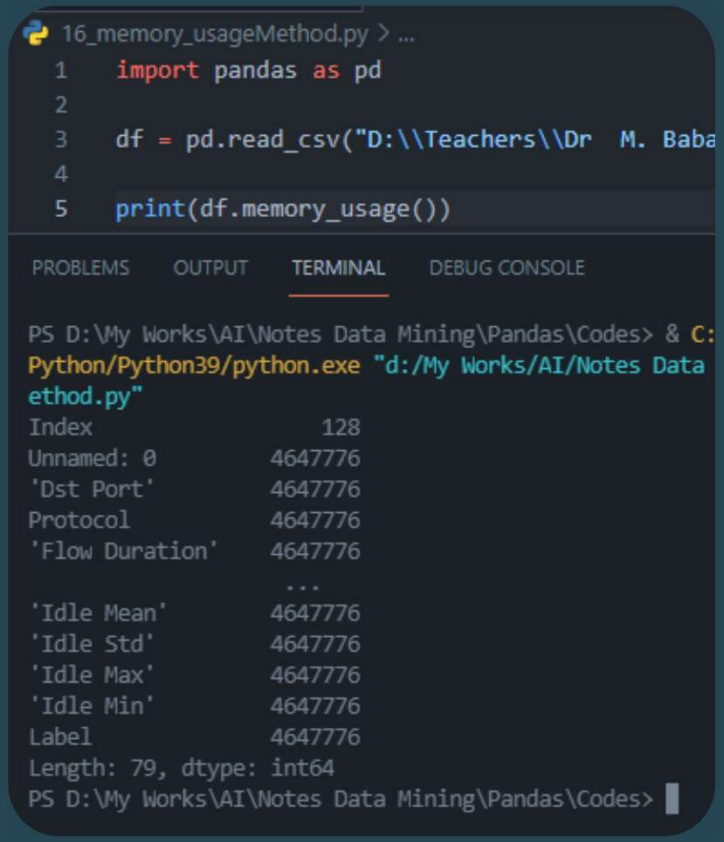
```
PS D:\My Works\AI\Notes Data Mining\Pandas\Codes> & C:/Users/ALSHO/AppDat
Python/Python39/python.exe "d:/My Works/AI/Notes Data Mining/Pandas/Codes
d.py"
        Unnamed: 0     'Dst Port'  ...     'Idle Max'     'Idle Min'
count  5.809720e+05  580972.000000  ...  580972.000000  580972.000000
mean   4.517814e+05       0.042099  ...       0.000001       0.000088
std    3.724537e+05       0.171969  ...       0.000008       0.000573
min    9.400000e+01       0.000000  ...       0.000000       0.000000
25%    1.453678e+05       0.000320  ...       0.000000       0.000000
50%    2.912295e+05       0.000336  ...       0.000000       0.000000
75%    9.033312e+05       0.000809  ...       0.000000       0.000000
max    1.048574e+06       1.000000  ...       0.000122       0.009521

[8 rows x 77 columns]
PS D:\My Works\AI\Notes Data Mining\Pandas\Codes>
```

Pandas - Analyzing

- ***memory_usage()***

  memory_usage() returns a Pandas Series having the memory usage of each column (in bytes) in a Pandas DataFrame. By specifying the deep attribute as True, we can get to know the actual space being taken by each column. More details on *memory_usage()* can be found here.

The memory usage of each column has been given as output in a Pandas Series. It is important to know the memory usage of a DataFrame, so that you can tackle errors like MemoryError in Python.



Pandas - Analyzing

- ***astype()***

  astype() is used to cast a Python object to a particular data type. It can be a very helpful function in case your data is not stored in the correct format (data type). For instance, if floating point numbers have somehow been misinterpreted by Python as strings, you can convert them back to floating point numbers with astype(). Or if you want to convert an object datatype to category, you can use astype().

Pandas - Analyzing

value_counts() returns a Pandas Series containing the counts of unique values. Consider a dataset that contains customer information about 5,000 customers of a company. value_counts() will help us in identifying the number of occurrences of each unique value in a Series. It can be applied to columns containing data like State, Industry of employment, or age of customers.

Resources

W3Schools

Thanks for your attention.