STAT 177, CLASS 3

Richard Waterman

July 2020

OBJECTIVES

OBJECTIVES

- A first look at the pandas library.
- The Series container:
    - Selecting components of a Series:
        - By position.
        - By name.
        - By logical filter.
    - Statistical summaries.

OBJECTIVES (CONT.)

- The DataFrame container:
    - Components of a data frame.
    - Creating a data frame from a dict structure.
    - Selecting components of a data frame (rows and columns):
        - By position.
        - By name.
        - By logical filter.
        - Using the ".loc" and ".iloc" methods.
    - Creating more complex logical filters.

THE PANDAS LIBRARY

THE PANDAS LIBRARY

- This is the most popular Python library for data science.
- It provides tools to simplify many parts of the data science workflow.
- We will look at two special data structures in pandas:
  - Series
  - DataFrame
- The DataFrame is ideally suited for holding statistical data, because it works in a row/column fashion just like a spreadsheet, and can contain different data types. In particular both numeric and categorical data.
- You can think of a data frame schematically like an Excel or Google Sheets spreadsheet, but you manipulate it programatically, rather than through a graphical user interface (GUI).

THE SERIES CONTAINER

THE SERIES CONTAINER

- A series is similar to a Python list structure, but it also has a label for each element, known as the “index”.
- As well as identifying elements by position, you can access them via their labels, or via a logical filter.
- If you don’t specify an index then a default numeric one will be created (starting at 0). Think of it as the row number.

```
# This will be the standard way of importing the pandas library and aliasing it to "pd"
import pandas as pd

# Create some data in place (later we will import from various sources).
# These are median house prices from various locations around Philadelphia.
house_data = pd.Series([66803, 104923, 114233, 114572, 112471, 99843, 74308, 147176, 199065, 130953],
                      index =['Collindale','Downingtown', 'Falls Town', 'Hatboro', 'Lansdale',
                             'Norwood', 'Sharon Hill', 'Springfield', 'Upper Darby', 'Yardley'])
```

REVIEW THE SERIES OBJECT

- Note that the index is printed along with the house prices.
- The data type, (here int64, means a 64 bit integer) is indicated at the bottom (This is a numeric variable).
- Knowing the data type is useful because the relevant statistical summaries and methods will depend on the data type.

```
print(house_data)
```

```
Collindale      66803
Downingtown    104923
Falls Town     114233
Hatboro        114572
Lansdale       112471
Norwood         99843
Sharon Hill     74308
Springfield    147176
Upper Darby    199065
Yardley        130953
dtype: int64
```

OBTAINING JUST THE SERIES VALUES OR JUST THE INDEX

- Use the .values and .index methods.

```
house_data.values # Look at the values in the Series
```

```
array([ 66803, 104923, 114233, 114572, 112471,  99843,  74308, 147176,
       199065, 130953], dtype=int64)
```

```
house_data.index # Look at the index itself
```

```
Index(['Collindale', 'Downingtown', 'Falls Town', 'Hatboro', 'Lansdale',
       'Norwood', 'Sharon Hill', 'Springfield', 'Upper Darby', 'Yardley'],
      dtype='object')
```

ACCESSING ELEMENTS IN THE SERIES

- Using pandas there are three ways of accessing the Series elements:
    1. By position.
    2. By name.
    3. By logical filter.

BY POSITION:

- You can identify arbitrary elements in the Series by position:

```
house_data[[0,7,3]] # Identify elements in positions 0, 7 and 3 (note the [[]] brackets).
```

```
Collindale      66803
Springfield    147176
Hatboro        114572
dtype: int64
```

- If you tried using this notation with a plain list rather than a Series, then …

```
list_a = list(range(10)) # Make a list
list_a[[0,7,3]] # This does not work!
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-6-b982f87e5549> in <module>
      1 list_a = list(range(10)) # Make a list
----> 2 list_a[[0,7,3]] # This does not work!


TypeError: list indices must be integers or slices, not list
```

BY POSITION AS INDICATED BY THE SLICE NOTATION.

```
house_data[2:5] # Recall the slice notation and note the single [] bracket.
```

```
Falls Town    114233
Hatboro       114572
Lansdale      112471
dtype: int64
```

BY INDEX LABEL

- Look at the difference in output between using one square bracket and using two square brackets:

```
house_data['Downingtown'] # Returns just the value of the Downingtown element.
```

```
104923
```

```
house_data[['Downingtown']] # returns a Series containing the Downingtown element.
```

```
Downingtown    104923
dtype: int64
```

GETTING MORE THAN ONE ELEMENT BY NAME

- You can obtain arbitrary elements by name:

```
house_data[['Downingtown', 'Lansdale', 'Upper Darby', 'Falls Town']]
```

```
Downingtown     104923
Lansdale        112471
Upper Darby     199065
Falls Town      114233
dtype: int64
```

BY LOGICAL FILTER

- Rather than using a name or position to extract an element in the Series, you can use a list with logical (True/False) values.
- So long as the list is the same length as the Series, those elements corresponding to a True are selected.
- Think of a logical filter like a sieve, and only those elements lining up with Trues get through.

Rowname	Value	Logical filter	Result
a	31	False	
b	16	True	16
c	12	True	12
d	27	False	
e	18	False	
f	9	True	9

EXAMPLE

```
raw_data = pd.Series([31,16,12,27,28,9],
                     index =['a','b', 'c', 'd', 'e','f'])
logic_filter = [False, True, True, False, False, True]

print(raw_data[logic_filter])
```

```
b    16
c    12
f     9
dtype: int64
```

SELECTION BY FILTER FOR THE HOUSING DATA

```
# A list containing Trues in positions, 0,3,7,8
logical_list = [True, False, False, True, False, False, False, True, True, False]

house_data[logical_list]
```

```
Collindale      66803
Hatboro        114572
Springfield    147176
Upper Darby    199065
dtype: int64
```

CREATE THE LOGICAL FILTER USING COMPARISON OPERATORS

- Find all the locations where the price is greater than $110,000.

```
expensive = house_data > 110000 # A logical comparison returning another Series, but this time of
  logicals.
print(expensive)
```

```
Collindale      False
Downingtown     False
Falls Town       True
Hatboro          True
Lansdale         True
Norwood         False
Sharon Hill     False
Springfield      True
Upper Darby      True
Yardley          True
dtype: bool
```

SELECTING THE EXPENSIVE AREAS

```
type(expensive)
```

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

```
house_data[expensive] # Just those rows where the price is greater than $110,000.
```

```
Falls Town     114233
Hatboro        114572
Lansdale       112471
Springfield    147176
Upper Darby    199065
Yardley        130953
dtype: int64
```

A SECOND EXAMPLE, ALL ON ONE LINE

- Find those areas with prices between 90000 and 110000.
- The logical statement can be created within the [] parenthesis as well.

```
house_data[(house_data > 90000) & (house_data < 110000)]
```

```
Downingtown    104923
Norwood         99843
dtype: int64
```

STATISTICAL SUMMARIES WITH PANDAS

- pandas provides many built in statistical summaries as methods, like mean and median (to be discussed).

```
house_data.mean() # The average of the prices.
```

```
116434.7
```

```
house_data.median() # The median of the prices.
```

```
113352.0
```

```
house_data.quantile([.25, .5, .75]) # The 25th, 50th and 75 percentiles.
```

```
0.25    101113.00
0.50    113352.00
0.75    126857.75
dtype: float64
```

OVERWRITING ELEMENTS OF THE SERIES

- If you can identify parts of a list, then you can edit/overwtite them using the assignment operator.

```
house_data[2] = 123456 # Overwrite a single element.
print(house_data)
```

```
Collindale       66803
Downingtown     104923
Falls Town      123456
Hatboro         114572
Lansdale        112471
Norwood          99843
Sharon Hill      74308
Springfield     147176
Upper Darby     199065
Yardley         130953
dtype: int64
```

OVERWRITING ELEMENTS OF THE SERIES, CTD.

```
house_data[4:6] = [999999, 8888888] # Overwrite using a slice,
print(house_data)
```

```
Collindale       66803
Downingtown     104923
Falls Town      123456
Hatboro         114572
Lansdale        999999
Norwood        8888888
Sharon Hill      74308
Springfield     147176
Upper Darby     199065
Yardley         130953
dtype: int64
```

OVERWRITING ELEMENTS OF THE SERIES, CTD.

```
house_data[house_data < 100000] = 0 # Overwrite using a logical filter to identify, and a repeated
  value to populate.
print(house_data)
```

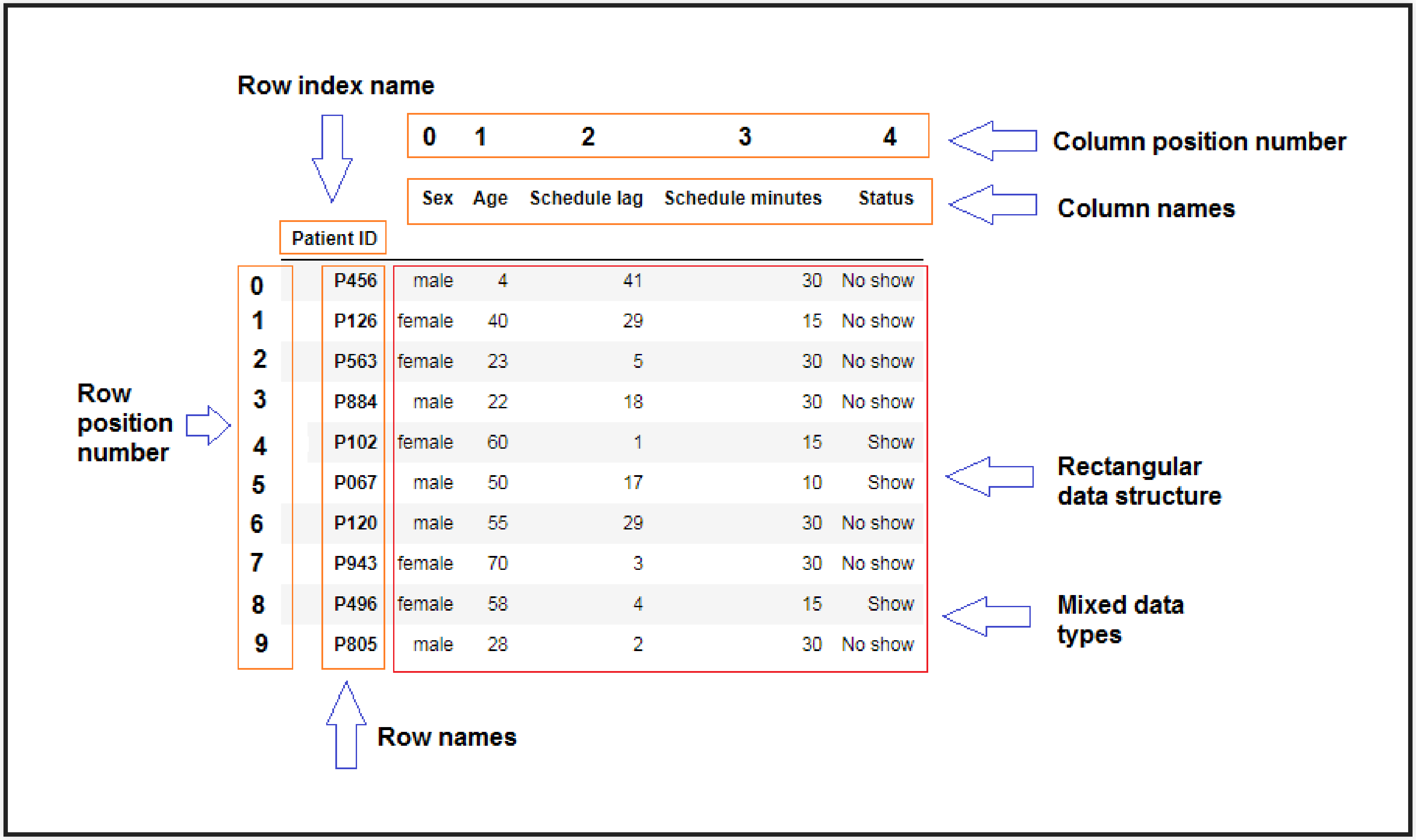
```
Collindale          0
Downingtown    104923
Falls Town     123456
Hatboro        114572
Lansdale       999999
Norwood       8888888
Sharon Hill         0
Springfield    147176
Upper Darby    199065
Yardley        130953
dtype: int64
```

THE DATAFRAME CONTAINER

THE DATAFRAME CONTAINER

- The DataFrame container is a rectangular data structure/container, with rows and columns.
- The columns are usually named, and the rows have an Index.
- Almost all statistical analysis programs use such a structure.
- An important feature of this container is that it can have different data types (numeric, string etc.) in different columns.
- In this way it is able to hold realistic datasets, which are usually of mixed variable types.

THE COMPONENTS OF A DATA FRAME



5.3

CREATING A DATAFRAME

- There are a variety of ways to populate a data frame with data, and we will subsequently learn how to read from an external source like a file or database.
- For our first data frame we will input the data ourselves and create the data frame directly.
- The data will come from a dict of lists.
- The keys in the dict will become the column names and the values in the lists will become the entries for each column.
- The data comes from a hospital outpatient clinic, where each row is a patient, the columns are patient attributes and the key variable of interest is *Status* which indicates whether a patient showed up for their visit.
- Later on we will use a bigger version of this dataset to create a predictive model of whether a patient shows up for their visit.

THE RAW DATA

```
#A dict structure containing the raw data and column names:
raw_data = {'Sex': ['male','female','female','male','female','male','male','female','female','male'],
           'Age': [4,40,23,22,60,50,55,70,58,28],
           'Schedule lag': [41,29,5,18,1,17,29,3,4,2],
           'Schedule minutes':[30,15,30,30,15,10,30,30,15,30],
           'Status': ['No show', 'No show', 'No show', 'No show', 'Show', 'Show', 'No show', 'No show',
  'Show', 'No show']}
```

POPULATING THE DATA FRAME

```
patient_data = pd.DataFrame(data = raw_data) # pd.DataFrame() when passed the raw data will create the
  new data frame.
print(patient_data)
```

```
      Sex  Age  Schedule lag  Schedule minutes   Status
0    male    4            41                30  No show
1  female   40            29                15  No show
2  female   23             5                30  No show
3    male   22            18                30  No show
4  female   60             1                15     Show
5    male   50            17                10     Show
6    male   55            29                30  No show
7  female   70             3                30  No show
8  female   58             4                15     Show
9    male   28             2                30  No show
```

FINDING THE SIZE OF THE DATA FRAME AND THE TYPES OF VARIABLES INCLUDED

```
patient_data.shape # The number of rows and columns (.shape).
```

```
(10, 5)
```

```
patient_data.dtypes # The data types in the data frame (.dtypes).
```

```
Sex                   object
Age                    int64
Schedule lag           int64
Schedule minutes       int64
Status                object
dtype: object
```

- When the data type is listed as “object” this means it is being treated as a string type, so statistically, the column will be viewed as a categorical variable.

REVIEW THE COLUMN NAMES

- We already know the column names of this data frame, but when you read in from an external source that is not always the case.
- The column names can be identified through the .columns attribute.
- Notice the names are in what is called an “Index” object. An index contains information about the rows or columns, for example, their names.

```
## Get just the names of the columns in the data frame with the .columns attribute.
print( patient_data.columns)
```

```
Index(['Sex', 'Age', 'Schedule lag', 'Schedule minutes', 'Status'], dtype='object')
```

SELECTING PIECES OF THE DATA FRAME

- We will be interested in subsetting by rows and subsetting by column, or possibly both.
- The most basic operation is to get at a specific column, and here is a direct way to do it:

```
# Get the Age column by name as we would if the data structure were a dict.
print(patient_data['Age'])
```

```
0     4
1    40
2    23
3    22
4    60
5    50
6    55
7    70
8    58
9    28
Name: Age, dtype: int64
```

GETTING AT THE COLUMN, USING THE NAME AS AN ATTRIBUTE

```
print(patient_data.Age) # This gets at the same column, but by "attribute" name.
```

```
0     4
1    40
2    23
3    22
4    60
5    50
6    55
7    70
8    58
9    28
Name: Age, dtype: int64
```

ADDING AN INDEX FOR THE ROWS

- The data frame was created with default row names, the numbers 0 through 9.

```
print(patient_data.index) # This shows the index: by default here the numbers 0 through 9.
```

```
RangeIndex(start=0, stop=10, step=1)
```

CREATING THE NEW INDEX

- If we had patient identifiers we could use these instead for the row index.
- Below, we create some patient identifiers and add them to the data frame as an index.
- We also give a name “Patient ID” to the new index.

```
patient_ids = ['P456', 'P126','P563', 'P884','P102', 'P067','P120', 'P943','P496', 'P805'] # Patient
  identifiers.
patient_data.index = patient_ids # Assign a new index.
patient_data.index.name = 'Patient ID' # Give the new index a name.
print(patient_data) # Check out the data frame.
```

```
               Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P456          male    4            41                30  No show
P126        female   40            29                15  No show
P563        female   23             5                30  No show
P884          male   22            18                30  No show
P102        female   60             1                15     Show
P067          male   50            17                10     Show
P120          male   55            29                30  No show
P943        female   70             3                30  No show
P496        female   58             4                15     Show
P805          male   28             2                30  No show
```

SELECTING ROWS AND COLUMNS

- There are a variety of ways of selecting rows and columns from the data frame.
- Some are similar to techniques we have seen earlier, but the .loc and .iloc methods are new.

```
print(patient_data[3:6]) # Use the slice operator to identify by row number.
```

```
                   Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P884              male   22            18                30  No show
P102            female   60             1                15     Show
P067              male   50            17                10     Show
```

SELECTING ROWS AND COLUMNS

```
print(patient_data[:6:-1]) # Use the slice operator to identify by row number.
```

```
                Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P805           male   28             2                30  No show
P496         female   58             4                15     Show
P943         female   70             3                30  No show
```

```
print(patient_data[['Sex', 'Status']]) # Identifying a set of columns.
```

```
                Sex   Status
Patient ID
P456           male  No show
P126         female  No show
P563         female  No show
P884           male  No show
P102         female     Show
P067           male     Show
P120           male  No show
P943         female  No show
P496         female     Show
P805           male  No show
```

USING THE LOCATE METHODS

- The first, “.loc” allows to identify by name, and the second “.iloc” by integer location.

```
print(patient_data.loc[['P120', 'P805']])  # Identify the rows by name.
```

```
             Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P120        male   55            29                30  No show
P805        male   28             2                30  No show
```

```
print(patient_data.loc[['P120', 'P805'],  ['Sex', 'Status']]) # Identify rows and columns by name.
```

```
             Sex   Status
Patient ID
P120        male  No show
P805        male  No show
```

IDENTIFYING BY POSITION

- The two statements below look very similar, but one has [] and the other [[]].
- The first returns a Series and the other one, a single column DataFrame.

```
print(patient_data.iloc[1]) # The second row returned as a Series, with index of column names.
```

```
Sex                    female
Age                        40
Schedule lag               29
Schedule minutes           15
Status                No show
Name: P126, dtype: object
```

```
print(patient_data.iloc[[1]]) # The second row returned as a DataFrame.
```

```
               Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P126        female   40            29                15  No show
```

SELECTING THE FIRST AND LAST ROWS OF THE DATA FRAME

```
print(patient_data.iloc[[0,-1]])
```

```
             Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P456        male    4            41                30  No show
P805        male   28             2                30  No show
```

SELECTING MULTIPLE ROWS AND COLUMNS

```
print(patient_data.iloc[[1,3,5],[2,3]]) # Rows 2, 4 and 6, with columns 3 and 4.
```

```
            Schedule lag  Schedule minutes
Patient ID
P126                  29                15
P884                  18                30
P067                  17                10
```

```
print(patient_data.iloc[:2,3:]) # Slice notation works too.
```

```
            Schedule minutes   Status
Patient ID
P456                      30  No show
P126                      15  No show
```

USING LOGICAL FILTERS

- Just like with Series, it can be important to select rows from a data frame with specific attributes.
- For example, just select the males, or just select the females.
- This can be done in the same was we did for the Series data structure.

```
patient_data.Sex == "female" # A series where the trues are for females and the falses for males.
print(patient_data.Sex == "female")
```

```
Patient ID
P456    False
P126     True
P563     True
P884    False
P102     True
P067    False
P120    False
P943     True
P496     True
P805    False
Name: Sex, dtype: bool
```

SELECTING JUST THE FEMALE ROWS FROM THE DATA FRAME

```
print(patient_data[patient_data.Sex == "female"]) # Select only the females
```

```
               Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P126        female   40            29                15  No show
P563        female   23             5                30  No show
P102        female   60             1                15     Show
P943        female   70             3                30  No show
P496        female   58             4                15     Show
```

```
# A compound selection of females who showed up. The "&" here performs the logical "and".
# It works elementwise on the two boolean Series.
print(patient_data[(patient_data.Sex == "female") & (patient_data.Status == "Show")])
```

```
               Sex  Age  Schedule lag  Schedule minutes Status
Patient ID
P102        female   60             1                15   Show
P496        female   58             4                15   Show
```

LOGICAL SELECTION TOGETHER WITH COLUMN EXTRACTION

- We could combine logical selection with column selection, to get at specific columns for which conditions hold true on other columns.

```
# Get the schedule lag for females who showed up.
print(patient_data[(patient_data.Sex == "female") & (patient_data.Status == "Show")].iloc[:, [2]])
```

```
           Schedule lag
Patient ID
P102                  1
P496                  4
```

EDITING/OVERWRITING PARTS OF A DATA FRAME

- As with Series, if you can select a part of a data frame, you can edit through assignment.

```
print(patient_data.iloc[2,0]) # Sex of the third patient.
patient_data.iloc[2,0] = 'male' # Overwrite from female to male.
print(patient_data.iloc[2,0])
```

```
female
male
```

EDITING A ROW SLICE

```
patient_data.iloc[2:4,1] = 19
print(patient_data)
```

```
                Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P456           male    4            41                30  No show
P126         female   40            29                15  No show
P563           male   19             5                30  No show
P884           male   19            18                30  No show
P102         female   60             1                15     Show
P067           male   50            17                10     Show
P120           male   55            29                30  No show
P943         female   70             3                30  No show
P496         female   58             4                15     Show
P805           male   28             2                30  No show
```

EDITING ROWS AND COLUMNS SIMULTANEOUSLY

- Note the nested lists being used for each row.

```
patient_data.loc[['P456', 'P884'],['Sex', 'Age']] = [['NA',99],['NA',99]]
print(patient_data)
```

```
                Sex  Age  Schedule lag  Schedule minutes   Status
Patient ID
P456             NA   99            41                30  No show
P126         female   40            29                15  No show
P563           male   19             5                30  No show
P884             NA   99            18                30  No show
P102         female   60             1                15     Show
P067           male   50            17                10     Show
P120           male   55            29                30  No show
P943         female   70             3                30  No show
P496         female   58             4                15     Show
P805           male   28             2                30  No show
```

CLASS SUMMARY

SUMMARY

- A first look at the pandas library.
- The Series container:
    - Selecting components of a Series.
    - Statistical summaries.
- The DataFrame container:
    - Components of a data frame.
    - Creating a data frame from a dict structure.
    - Selecting components of a data frame (rows and columns).
    - Creating more complex logical filters.

NEXT TIME

NEXT TIME

- Importing data to Python:",
  - Data file types.",
  - CSV.",
  - HTML.",
  - JSON.",
- Data locations:",
  - A local file.",
  - A remote (web) resource.",
  - A database.",
- Joining datasets.",
- Writing basic functions in Python."