THE UNIVERSITY OF CHICAGO

MASTERS IN COMPUTATIONAL SOCIAL SCIENCE
THE UNIVERSITY OF CHICAGO

MACS 30111

M7: Pandas

Announcements

- Switch to accelerated track: need MPCS exam
  - https://masters.cs.uchicago.edu/page/placement-exams-0
  - Register, take the exam
- Final exam schedule
  - Dec 5th, 10-12pm
- Review
  - Textbook
  - Team tutorial
  - Short exercise
  - Programming assignment
  - Extra exercise

Agenda / misc

- Last assignment due Friday!
- Can submit PA4 OR PA5 to ‘replace’ a prior PA
  - FYI: difficulty level of PA4 similar to PA2 and difficulty level of PA5 is similar to PA3
- Final exam similar to midterm – looking to incorporate choice.

Warm-up question: Programming practice

- **<u>CLOSED NOTE</u>**

- **Suppose you have a dataframe with 20 rows and 10 columns.**
  - **DF name: albums**
  - **Column names: title, num_songs, year, genre, length, billboard, weeks_on_charts, label, avg_bpm, number_sold**
- **From the above dataframe, write the following code on your paper:**
  - **Get the shape of the dataframe**
  - **Print the information for the album in the 10th row**
  - **Print the information for the "genre" column**
  - **(bonus) Select albums that have have been in the top 100 on billboard (e.g. have a value less than or equal to 100)**

Topics:

- Introduction to Pandas
- Creating DataFrames and Series
- Working with DataFrames and Series
- Applying functions to data
- Group, combine, and pivoting
- Visualizing DataFrames and Series (Probably Thurs)

pandas

Data analysis toolkit

Reference: https://pandas.pydata.org/

The "*pandas*" name

pan for "Panel"

da for "Data"

Panel data: multidimensional, structured datasets that include observations of something over a number of different time periods.

Use cases for the *pandas* library

Tabular Data

Time-series Data

Matrix Data

Statistical Datasets

Topics:

- Introduction to Pandas
- **Creating DataFrames and Series**
- Working with DataFrames and Series
- Applying functions to data
- Group, combine, and pivoting
- Visualizing DataFrames and Series

Pandas

Series

```
s
```

```
Susan      90
Joanna     95
Jim        90
dtype: int64
```

DataFrame

	pa1	pa2
Susan	90	88
Joanna	95	90
Jim	90	75

Reference:

- https://pandas.pydata.org/docs/reference/api/pandas.Series.html
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

Importing the Pandas library

Import pandas using the alias “pd”

```
import pandas as pd
```

Series Constructor

Pass in 1-d data (list/np.array), alongside an accompanying (1-d) index:

```
data = [90, 95, 90]
index = ['Susan', 'Joanna', 'Jim']
s = pd.Series(data, index)
```

```
1 s
```

```
Susan      90
Joanna     95
Jim        90
dtype: int64
```

Pass in a **dictionary**, where the keys are inferred to be the indices:

```
data = {'Susan': 90, 'Joanna': 95, 'Jim': 90}
s = pd.Series(data)
```

DataFrame Constructor

Pass in a dictionary of lists/arrays, alongside an accompanying (1-d) index:

```
data = {'pa1': [90, 95, 90], 'pa2': [88, 90, 75]}
index = ['Susan', 'Joanna', 'Jim']
df = pd.DataFrame(data, index = index)
```

	pa1	pa2
Susan	90	88
Joanna	95	90
Jim	90	75

Pass in a **dictionary of dictionaries**

```
data = {'pa1': {'Susan': 90, 'Joanna': 95, 'Jim': 90},
        'pa2': {'Susan': 88, 'Joanna': 90, 'Jim': 75}}
df = pd.DataFrame(data)
```

DataFrame Constructor

Data can also be passed in as an NumPy multidimensional array:

```
import numpy as np
data = np.random.uniform(low = 0, high = 1, size = (3, 2))
data
```

```
array([[0.37751699, 0.20715479],
       [0.61155007, 0.76146711],
       [0.09620871, 0.28342811]])
```

```
index = ['Susan', 'Joanna', 'Jim']
columns = ['pa1', 'pa2']
df = pd.DataFrame(data, index = index, columns = columns)
df
```

	pa1	pa2
Susan	0.837535	0.084144
Joanna	0.798625	0.997886
Jim	0.491516	0.132333

Read from file

CSV

```
trees = pd.read_csv("./2015StreetTreesCensus_TREES.csv")
```

JSON

HTML

Excel Spreadsheets

And much more...

Reference: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Topics:

- Introduction to Pandas
- Creating DataFrames and Series
- **Working with DataFrames and Series**
- Applying functions to data
- Group, combine, and pivoting
- Visualizing DataFrames and Series

Key commands

- Loading: read_csv()
- Summarizing:
  - data.head()
  - Data.tail()
- Selecting:
  - By index: iloc (e.x. `trees.iloc[:,1])`
  - By name: loc (e.x. `trees.loc[:,"block_id"])`

https://www.datacamp.com/cheat-sheet/pandas-cheat-sheet-for-data-science-in-python

Motivating example

- Data exploration for the 2015 New York City Street Tree Survey
- Data: https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/uvpi-gqnh
  - CSV, 683,789 lines

1. How many different species are planted as street trees in New York?
2. What are the five most common street tree species in New York?
3. What is the most common street tree species in Brooklyn?
4. What percentage of the street trees in Queens are dead or in poor health?
5. How does street tree health differ by borough?

https://github.com/computer-science-with-applications/examples/tree/main/working_with_data/pandas

Coding practice: 4.3.1

Series VS DataFrame

- Series:
  - One-dimensional ndarray with axis labels

```
Susan      90
Joanna     95
Jim        90
dtype: int64
```

- DataFrame
  - Two-dimensional tabular data with labeled row and column axes

	pa1	pa2
Susan	90	88
Joanna	95	90
Jim	90	75

Reference:

- Series: https://pandas.pydata.org/docs/reference/api/pandas.Series.html
- DataFrame: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

Approaching the Series / DataFrame

Size, column names and data types: *dtype, shape*

```
s
```

```
Susan      90
Joanna     95
Jim        90
dtype: int64
```

```
s.dtype
```

```
dtype('int64')
```

```
s.shape
```

```
(3,)
```

```
df
```

	pa1	pa2
Susan	0.837535	0.084144
Joanna	0.798625	0.997886
Jim	0.491516	0.132333

```
df.dtypes
```

```
pa1      float64
pa2      float64
dtype: object
```

```
df.shape
```

```
(3, 2)
```

Approaching the Series / DataFrame

What does the data look like? *df.head(), df.tail()*

```
s.head(n=2)
```

```
Susan      90
Joanna     95
dtype: int64
```

```
s.tail(n=1)
```

```
Jim     90
dtype: int64
```

Missing Data

Identifying missing data in Series and DataFrames:

```
s.isna(); s.notna()
df.isna(); df.notna()
```

We can drop rows or columns with missing data:

```
df_drop_byrow = df.dropna()
df_drop_bycol = df.dropna(axis = 1)
```

We can also fill missing values:

```
df.col.fillna(np.mean(df.col))
```

Coding practice: 4.3.2

Selecting rows (indexing)

tree_id	status	health	spc_common	borough	latitude	longitude
180683	Alive	Fair	red maple	Queens	40.723092	-73.844215
200540	Alive	Fair	pin oak	Queens	40.794111	-73.818679

Row label-based indexing

```
trees.loc[180683]
```

```
status            Alive
health             Fair
spc_common    red maple
borough          Queens
latitude      40.723092
longitude    -73.844215
Name: 180683, dtype: object
```

Position-based indexing

```
trees.iloc[0]
```

```
status            Alive
health             Fair
spc_common    red maple
borough          Queens
latitude      40.723092
longitude    -73.844215
Name: 180683, dtype: object
```

Naming columns

- If you want to explore the column names that were imported, you can use: `trees.columns`

```
Index(['created_at', 'block_id', 'the_geom', 'tree_dbh', 'stump_diam',
       'curb_loc', 'status', 'health', 'spc_latin', 'spc_common', 'steward',
       'guards', 'sidewalk', 'user_type', 'problems', 'root_stone',
       'root_grate', 'root_other', 'trnk_wire', 'trnk_light', 'trnk_other',
       'brnch_ligh', 'brnch_shoe', 'brnch_othe', 'address', 'zipcode',
       'zip_city', 'cb_num', 'borocode', 'boroname', 'cncldist', 'st_assem',
       'st_senate', 'nta', 'nta_name', 'boro_ct', 'state', 'Latitude',
       'longitude', 'x_sp', 'y_sp'],
      dtype='object')
```

Selecting columns

Note: depending on data, might have 'boroname'

```
type(trees['health'])
```

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

```
tree_id
180683    Fair
200540    Fair
204026    Good
204337    Good
189565    Good
          ...
155433    Good
183795    Good
166161    Good
184028    Good
200607    Fair
```

```
type(trees[['health']])
```

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

	health
tree_id	
180683	Fair
200540	Fair
204026	Good
204337	Good
189565	Good
...	...
155433	Good
183795	Good
166161	Good
184028	Good
200607	Fair

```
type(trees[['health','borough']])
```

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

	health	borough
tree_id		
180683	Fair	Queens
200540	Fair	Queens
204026	Good	Brooklyn
204337	Good	Brooklyn
189565	Good	Brooklyn
...	...	...
155433	Good	Brooklyn
183795	Good	Queens
166161	Good	Staten Island
184028	Good	Bronx
200607	Fair	Queens

Selecting rows and columns

```
trees.iloc[:5][['health','borough']]
```

	health	borough
tree_id		
180683	Fair	Queens
200540	Fair	Queens
204026	Good	Brooklyn
204337	Good	Brooklyn
189565	Good	Brooklyn

Note: depending on data, might have 'boroname'

```
trees.loc[[180683, 200540, 204026],['health','borough']]
```

	health	borough
tree_id		
180683	Fair	Queens
200540	Fair	Queens
204026	Good	Brooklyn

Filtering

- Boolean masks
- Logical operations

```
filter = (trees.boroname == "Queens") & ((trees.status == "Dead") |
(trees.health == "Poor"))
trees[filter]
```

- Remove rows where trees.status is Dead
- Select rows where trees.status is not Dead

```
trees[trees.status != "Dead"]
```

```
1  trees.status == "Dead"
```

```
tree_id
180683    False
200540    False
204026    False
204337    False
189565    False
          ...
155433    False
183795    False
166161    False
184028    False
200607    False
```

Coding practice: 4.3.4

Practice Task: answer a research question

- **Suppose we have the following question:** how do tree health indicators vary by borough? **GIVEN THE CURRENT SKILLS / TOOLS WE HAVE DISCUSSED SO FAR**
- **In groups:**
  - Determine how you will measure this. (which column)
  - What tools do you currently have to do this?
  - What do you want to be able to do?

Topics:

- Introduction to Pandas
- Creating DataFrames and Series
- Working with DataFrames and Series
- **Applying functions to data**
- Group, combine, and pivoting
- Visualizing DataFrames and Series

Common Summary Statistics

`DataFrame.count`

Count number of non-NA/null observations.

`DataFrame.max`

Maximum of the values in the object.

`DataFrame.min`

Minimum of the values in the object.

`DataFrame.mean`

Mean of the values.

`DataFrame.std`

Standard deviation of the observations.

`DataFrame.select_dtypes`

Subset of a DataFrame including/excluding columns based on their dtype.

```
s_trees.count()
```

```
spc_common    652169
status        683788
health        652172
borough       683788
latitude      683788
longitude     683788
dtype: int64
```

```
s_trees.status.value_counts()
```

```
Alive    652173
Stump     17654
Dead      13961
Name: status, dtype: int64
```

Summary statistics for each column: *df.describe()*

Summary of categorical values: *df.value_counts()*

Pairwise correlation of columns: *df.corr()*

Reference: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html

TASK: develop a research question

- In your groups, develop a research question you can answer with the trees dataset.
- Your question needs to use **ALL of** the following:
  - **Filtering**
  - **Applying a Boolean criterion**
  - **Label-based indexing**
  - **Calculating a numerical value (e.g. mean/min/max/std/corr)**
  - **Exporting the text file (we've discussed prior to break!)**

Warm-up question: Programming practice

- **<u>CLOSED NOTE</u>**

- **Suppose you have a dataframe with 20 rows and 10 columns.**
  - **DF name: albums**
  - **Column names: title, num_songs, year, genre, length, billboard, weeks_on_charts, label, avg_bpm, number_sold**
- **From the above dataframe, write the following code on your paper:**
  - **Get the shape of the dataframe**
  - **Print the information for the album in the 10th row**
  - **Print the information for the "genre" column**
  - **(bonus) Select albums that have have been in the top 100 on billboard (e.g. have a value less than or equal to 100)**

Applying Functions to Data

`Series.apply`

For applying more complex functions on a Series.

`DataFrame.apply`

Apply a function row-/column-wise.

`DataFrame.applymap`

Apply a function elementwise on a whole DataFrame.

Reference: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.apply.html

Topics:

- Introduction to Pandas
- Creating DataFrames and Series
- Working with DataFrames and Series
- Applying functions to data
- **Group, combine, and pivoting**
- Visualizing DataFrames and Series

Grouping

```
s_trees.groupby("borough").size()
```

```
borough
Bronx              85203
Brooklyn          177293
Manhattan          65423
Queens            250551
Staten Island     105318
dtype: int64
```

```
s_trees.groupby(["borough", "status"]).size()
```

```
borough        status
Bronx          Alive      80585
               Dead        2530
               Stump       2088
Brooklyn       Alive     169744
               Dead        3319
               Stump       4230
Manhattan      Alive      62427
               Dead        1802
               Stump       1194
Queens         Alive     237974
               Dead        4440
               Stump       8137
Staten Island  Alive     101443
               Dead        1870
               Stump       2005
dtype: int64
```

Unstack: converts series with hierarchical index into a DataFrame

Coding practice: 4.3.5

Combining DataFrames

- **concat( )**
  - perform concatenation along an axis
  - while performing set logic of the indexes on other axes
  - make a full copy of the data

- **merge( )**
  - Standard database join operations between DataFrame or Series objects

- **join( )**
  - join on index
  - combine columns of two differently indexed DataFrames into a single one

Reference: https://pandas.pydata.org/docs/user_guide/merging.html

Pivoting

Convert a long and thin DataFrame into a short and wide DataFrame

- A column for index
- A column to supply column names for the new DataFrame
- A column to fill the new DataFrame

Coding practice: 4.3.6

Topics:

- Introduction to Pandas
- Creating DataFrames and Series
- Working with DataFrames and Series
- Applying functions to data
- Group, combine, and pivoting
- **Visualizing DataFrames and Series**

Plotting with Pandas

Using the plot method:

df.plot(kind='line') # produces a line plot by default
s.plot()

Reference: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html

Topics:

- Introduction to Pandas
- Creating DataFrames and Series
- Working with DataFrames and Series
- Applying functions to data
- Group, combine, and pivoting
- Visualizing DataFrames and Series

More details: https://pandas.pydata.org/docs/user_guide/