Introduction to Python

Tom Paskhalis

RECSM Summer School 2021, Pandas and Data I/O, Part 3, Day 1

Data in Python

- Python can hold and manipulate > 1 dataset at the same time
- Python stores objects in memory
- The limit on the size of data is determined by your computer memory
- Most functionality for dealing with data is provided by external libraries

Pandas

- Standard Python library does not have data type for tabular data
- However, `pandas` library has become the de facto standard for data manipulation
- pandas is built upon (and often used in conjuction with) other computational libraries
- E.g. `numpy` (array data type), `scipy` (linear algebra) and `scikit-learn` (machine learning)

Pandas

- Standard Python library does not have data type for tabular data
- However, `pandas` library has become the de facto standard for data manipulation
- pandas is built upon (and often used in conjuction with) other computational libraries
- E.g. `numpy` (array data type), `scipy` (linear algebra) and `scikit-learn` (machine learning)

```
In [1]: import pandas as pd # Using 'as' allows to avoid typing full name each time the mod
```

Series

- *Series* is a one-dimensional array-like object

Series

- *Series* is a one-dimensional array-like object

In [2]:
```
sr1 = pd.Series([150.0, 120.0, 3000.0])
sr1
```

Out[2]:
```
0     150.0
1     120.0
2    3000.0
dtype: float64
```

Series

- *Series* is a one-dimensional array-like object

In [2]:
```
sr1 = pd.Series([150.0, 120.0, 3000.0])
sr1
```

Out[2]:
```
0     150.0
1     120.0
2    3000.0
dtype: float64
```

In [3]:
```
sr1[0] # Slicing is simiar to standard Python objects
```

Out[3]:
```
150.0
```

Series

- *Series* is a one-dimensional array-like object

In [2]:
```
sr1 = pd.Series([150.0, 120.0, 3000.0])
sr1
```

Out[2]:
```
0     150.0
1     120.0
2    3000.0
dtype: float64
```

In [3]:
```
sr1[0] # Slicing is simiar to standard Python objects
```

Out[3]:
```
150.0
```

In [4]:
```
sr1[sr1 > 200]
```

Out[4]:
```
2    3000.0
dtype: float64
```

Indexing in Series

- Another way to think about Series is as a ordered dictionary

Indexing in Series

- Another way to think about Series is as a ordered dictionary

In [5]:
```
d = {'apple': 150.0, 'banana': 120.0, 'watermelon': 3000.0}
```

Indexing in Series

- Another way to think about Series is as a ordered dictionary

In [5]:
```
d = {'apple': 150.0, 'banana': 120.0, 'watermelon': 3000.0}
```

In [6]:
```
sr2 = pd.Series(d)
sr2
```

Out[6]:
```
apple          150.0
banana         120.0
watermelon    3000.0
dtype: float64
```

Indexing in Series

- Another way to think about Series is as a ordered dictionary

In [5]:
```
d = {'apple': 150.0, 'banana': 120.0, 'watermelon': 3000.0}
```

In [6]:
```
sr2 = pd.Series(d)
sr2
```

Out[6]:
```
apple          150.0
banana         120.0
watermelon    3000.0
dtype: float64
```

In [7]:
```
sr2[0] # Recall that this slicing would be impossible for standard dictionary
```

Out[7]:
```
150.0
```

Indexing in Series

- Another way to think about Series is as a ordered dictionary

In [5]:
```
d = {'apple': 150.0, 'banana': 120.0, 'watermelon': 3000.0}
```

In [6]:
```
sr2 = pd.Series(d)
sr2
```

Out[6]:
```
apple          150.0
banana         120.0
watermelon    3000.0
dtype: float64
```

In [7]:
```
sr2[0] # Recall that this slicing would be impossible for standard dictionary
```

Out[7]:
```
150.0
```

In [8]:
```
sr2.index
```

DataFrame - the workhorse of data analysis

- *DataFrame* is a rectangular table of data

DataFrame - the workhorse of data analysis

- *DataFrame* is a rectangular table of data

In [9]:
```
data = {'fruit': ['apple', 'banana', 'watermelon'], # DataFrame can be constructed
        'weight': [150.0, 120.0, 3000.0],           # a dict of equal-length lists/
        'berry': [False, True, True]}
df = pd.DataFrame(data)
df
```

Out[9]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True
2	watermelon	3000.0	True

Indexing in DataFrame

- DataFrame has both row and column indices
- `DataFrame.loc()` provides method for *label* location
- `DataFrame.iloc()` provides method for *index* location

Indexing in DataFrame

- DataFrame has both row and column indices
- `DataFrame.loc()` provides method for *label* location
- `DataFrame.iloc()` provides method for *index* location

```
In [10]: df.iloc[0] # First row

Out[10]: fruit      apple
         weight     150.0
         berry      False
         Name: 0, dtype: object
```

Indexing in DataFrame

- DataFrame has both row and column indices
- `DataFrame.loc()` provides method for *label* location
- `DataFrame.iloc()` provides method for *index* location

```
In [10]: df.iloc[0] # First row
```

```
Out[10]: fruit      apple
         weight     150.0
         berry      False
         Name: 0, dtype: object
```

```
In [11]: df.iloc[:,0] # First column
```

```
Out[11]: 0         apple
         1        banana
         2    watermelon
         Name: fruit, dtype: object
```

Summary of indexing in DataFrame

Expression	Selection Operation
`df[val]`	Column or sequence of columns +convenience (e.g. slice)
`df.loc[lab_i]`	Row or subset of rows by label
`df.loc[:, lab_j]`	Column or subset of columns by label
`df.loc[lab_i, lab_j]`	Both rows and columns by label
`df.iloc[i]`	Row or subset of rows by integer position
`df.iloc[:, j]`	Column or subset of columns by integer position
`df.iloc[i, j]`	Both rows and columns by integer position
`df.at[lab_i, lab_j]`	Single scalar value by row and column label
`df.iat[i, j]`	Single scalar value by row and column integer position

Subsetting in DataFrame

Subsetting in DataFrame

In [12]:
```
df.iloc[:2] # Select the first two rows (with convenience shortcut for slicing)
```

Out[12]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True

Subsetting in DataFrame

In [12]:
```
df.iloc[:2] # Select the first two rows (with convenience shortcut for slicing)
```

Out[12]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True

In [13]:
```
df[:2]  # Shortcut
```

Out[13]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True

Subsetting in DataFrame

In [12]:
```
df.iloc[:2] # Select the first two rows (with convenience shortcut for slicing)
```

Out[12]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True

In [13]:
```
df[:2]  # Shortcut
```

Out[13]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True

In [14]:
```
df.loc[:, ['fruit', 'berry']] # Select the columns 'fruit' and 'berry'
```

Out[14]:

	fruit	berry
0	apple	False
1	banana	True

Subsetting in DataFrame

In [12]:
```
df.iloc[:2] # Select the first two rows (with convenience shortcut for slicing)
```

Out[12]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True

In [13]:
```
df[:2]  # Shortcut
```

Out[13]:

	fruit	weight	berry
0	apple	150.0	False
1	banana	120.0	True

In [14]:
```
df.loc[:, ['fruit', 'berry']] # Select the columns 'fruit' and 'berry'
```

Out[14]:

	fruit	berry
0	apple	False
1	banana	True

Columns in DataFrame

Columns in DataFrame

In [16]:
```
df.columns # Retrieve the names of all columns
```

Out[16]:
```
Index(['fruit', 'weight', 'berry'], dtype='object')
```

Columns in DataFrame

In [16]:
```
df.columns # Retrieve the names of all columns
```

Out[16]:
```
Index(['fruit', 'weight', 'berry'], dtype='object')
```

In [17]:
```
df.columns[0] # This Index object is subsettable
```

Out[17]:
```
'fruit'
```

Columns in DataFrame

In [16]:
```
df.columns # Retrieve the names of all columns
```

Out[16]:
```
Index(['fruit', 'weight', 'berry'], dtype='object')
```

In [17]:
```
df.columns[0] # This Index object is subsettable
```

Out[17]:
```
'fruit'
```

In [18]:
```
df.columns.str.startswith('fr') # As column names are strings, we can apply str met
```

Out[18]:
```
array([ True, False, False])
```

Columns in DataFrame

In [16]:
```
df.columns # Retrieve the names of all columns
```

Out[16]:
```
Index(['fruit', 'weight', 'berry'], dtype='object')
```

In [17]:
```
df.columns[0] # This Index object is subsettable
```

Out[17]:
```
'fruit'
```

In [18]:
```
df.columns.str.startswith('fr') # As column names are strings, we can apply str met
```

Out[18]:
```
array([ True, False, False])
```

In [19]:
```
df.iloc[:,df.columns.str.startswith('fr')] # This is helpful with more complicated
```

Out[19]:

	fruit
0	apple
1	banana
2	watermelon

Filtering in DataFrame

Filtering in DataFrame

In [20]:
```
df[df.loc[:,'berry'] == False] # Select rows where fruits are not berries
```

Out[20]:

	fruit	weight	berry
0	apple	150.0	False

Filtering in DataFrame

In [20]:
```
df[df.loc[:,'berry'] == False] # Select rows where fruits are not berries
```

Out[20]:

	fruit	weight	berry
0	apple	150.0	False

In [21]:
```
df[df['berry'] == False] # The same can be achieved with more concise syntax
```

Out[21]:

	fruit	weight	berry
0	apple	150.0	False

Filtering in DataFrame

In [20]:
```
df[df.loc[:,'berry'] == False] # Select rows where fruits are not berries
```

Out[20]:

	fruit	weight	berry
0	apple	150.0	False

In [21]:
```
df[df['berry'] == False] # The same can be achieved with more concise syntax
```

Out[21]:

	fruit	weight	berry
0	apple	150.0	False

In [22]:
```
weight200 = df[df['weight'] > 200] # Create new dataset with rows where weight is h
weight200
```

Out[22]:

	fruit	weight	berry
2	watermelon	3000.0	True

Reading data in Python

- We will use the data from Kaggle 2020 Machine Learning and Data Science Survey
- For more information you can read the executive summary
- Or explore the winning Python Jupyter Notebooks

Reading data in Python

- We will use the data from Kaggle 2020 Machine Learning and Data Science Survey
- For more information you can read the executive summary
- Or explore the winning Python Jupyter Notebooks

In [23]:
```
# We specify that we want to combine first two rows as a header
kaggle2020 = pd.read_csv('../data/kaggle_survey_2020_responses.csv', header = [0,1]
```

In [24]:
```
kaggle2020.head() # Returns the top n (n=5 default) rows
```

Out[24]:

	Time from Start to Finish (seconds)	Q1	Q2	Q3	Q4	Q5	Q6	Q7_Part_1	Q7_Part_2	Q7_Par
	Duration (in seconds)	What is your age (# years)?	What is your gender? - Selected Choice	In which country do you currently reside?	What is the highest level of formal education that you have attained or plan to attain within the next 2 years?	Select the title most similar to your current role (or most recent title if retired): - Selected Choice	For how many years have you been writing code and/or programming?	What programming languages do you use on a regular basis? (Select all that apply) - Selected Choice - Python	What programming languages do you use on a regular basis? (Select all that apply) - Selected Choice - R	What progra langua you use regular (Select that ap Selecte Choice
0	1838	35-39	Man	Colombia	Doctoral degree	Student	5-10 years	Python	R	SQL
1	289287	30-34	Man	United States of America	Master's degree	Data Engineer	5-10 years	Python	R	SQL
2	860	35-39	Man	Argentina	Bachelor's degree	Software Engineer	10-20 years	NaN	NaN	NaN

Reading in other (non-`.csv`) data files

- Pandas can read in file other than `.csv` (comma-separated value)
- Common cases include STATA `.dta`, SPSS `.sav` and SAS `.sas`
- Use `pd.read_stata(path)`, `pd.read_spss(path)` and `pd.read_sas(path)`
- Check here for more examples

Writing data out in Python

- Note that when writing data out we start with the object name storing the dataset
- I.e. `df.to_csv(path)` as opposed to `df = pd.read_csv(path)`
- Pandas can also write out into other data formats
- E.g. `df.to_excel(path)`, `df.to_stata(path)`

Writing data out in Python

- Note that when writing data out we start with the object name storing the dataset
- I.e. `df.to_csv(path)` as opposed to `df = pd.read_csv(path)`
- Pandas can also write out into other data formats
- E.g. `df.to_excel(path)`, `df.to_stata(path)`

In [25]:
```
kaggle2020.to_csv('../temp/kaggle2020.csv')
```

Additional pandas materials

Books:

- McKinney, Wes. 2017. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 2nd ed. Sebastopol, CA: O'Reilly Media

  From the original author of the library!

Online:

Tomorrow

- Exploratory data analysis
- Data visualization