CS 111: Program Design I
Lecture 15: More Pandas, Misc., Legal Analytics

Robert H. Sloan & Richard Warner
University of Illinois at Chicago

October 17, 2019

UIC

Pandas: Brief demo in Spyder

- Showing a couple of the graphs from the previous lecture

PANDAS: USER MANUAL STYLE

Resource

- Python for Data Science Pandas Cheat Sheet
- https://www.datacamp.com/community/blog/python-pandas-cheat-sheet

Pandas data types

- Most important: dataframe, which we are getting from pandas.read_csv()
  - 2-d array, with column headers
- Series: 1-d array, e.g., one column of a dataframe, second most important

Dataframe Indexing: General idea overview

Sample 3 x 3 dataframe df:

	A	B	C
0	1	2	3
1	4	5	6
2	7	8	9

- Idea is [row][col]
- .iloc with (only) numbers ("integer location")
  - To get the (red) 1:
    - df.iloc[0][0]
- .loc with labels/column headers, possibly mixed with numbers
  - To get the 1:
    - df.loc[0]['A']

Dataframe indexing: Columns

- frame[columnname] returns series from *column* with *name* columnname
- Giving the []s *list* of names selects those columns *in list's order.* E.g.,
  - `scdb[['justiceName','chief','docketID']]`
- Other indexing: .iloc, .loc
  - (also others we won't cover)
    - Special case: specifically a slice index to whole frame will slice by *rows* for convenience because it's a very common operation, but inconsistent with overall Pandas syntax

Dataframe positional slicing: iloc

- .iloc for 100% *positional* indexing and slicing with usual Python 0 to length – 1 numbering (stands for "integer location")
- Arguments for both dimensions separated by comma [rows, cols]:
  - `frame.iloc[:3, :4]` upper left 3 rows/4 cols
  - `frame.iloc[:, :3]` all rows, first 3 cols
- One argument: *rows* *(possibly counterintuitive)*
  - `frame.iloc[3:6]` second 3 rows
  - `frame.iloc[41]` 42nd row

Dataframe label indexing: .loc

- Use .loc to access by *labels*, or *mix of labels and ints*
  - selection list will put columns in list's order; selection *set* in {}s keeps original dataframe order
  - `scdb.loc[3:6, {'docketId', 'chief', 'justiceName'}]`
    - Rows 3 through 6 inclusive, columns in scdb's order
  - `scdb.loc[3:6, ['docketId', 'chief', 'justiceName']]`
    - Rows 3 through 6 inclusive, columns in order ['docketId', 'chief', 'justiceName']
- Notice loc uses slices inclusive of *both* ends, *unlike all rest of Python & Pandas (!)*
- .loc with only numerical slices: error (e.g., foo.loc[3:6, 2:4])

Dataframe and series methods

- head(): returns sub-dataframe (top rows)
  - or for series, first entries
- tail(): same, bottom rows
  - With no argument they default to 5 rows; can give positive integer argument for number of rows
- count(): For series, returns number of values (excluding missing, NaN, etc.), *does include repetitions*
  - For dataframe, returns series, with count of each column, labeled by column
- .nunique(): For series: does *not* include repetitions

Dataframe and series *methods* (cont.)

- abs, max, mean, median, min, mode, sum
  - All behave like count, *except* will give errors if data types don't support the operation
  - E.g., a series of strings *does* return good answer with .max() method (based on alphabetical order), but cannot take .median()

plotting

- Both DataFrame and Series have a plot() method (as do many other Pandas types)
- Must have loaded Python's plotting module, because Pandas is making use of it:

```
import matplotlib.pyplot as plt
```

- Default is Series makes a line graph; DataFrame makes one line graph per column, and labels each line by column labels

100% Optional: Aside for graph geeks

- Optional for fun: To change style of your plot:

```
import matplotlib
matplotlib.style.use('fivethirtyeight') # OR
matplotlib.style.use('ggplot')   # R style
```

- Out of the box, it's Matlab style, which some folks like a lot

pandas dataframe .plot() method

- Needs no arguments
- Has optional arguments including *kind*:
  - .plot(kind='bar') for bar graphs
  - Many others including
    - 'hist' for histogram
    - 'box' for box with whiskers
    - 'area' for stacked area plots
    - 'scatter' for scatter plots
    - 'pie' for pie charts

.plot() x and y arguments

- If you have dataframe but want one column as x values and one as y values, can use optional argument(s)
- df.plot(x='Year')
  - Plot all columns except 'Year' as line graphs against x being the Year column

Brief demo: Chi murder rate by year

```
import matplotlib.pyplot as plt
import pandas

f = open('Chicago murders to 2012HeaderRows.csv', 'r')
df = pandas.read_csv(f)
```

#Note to self: Look in this semester CS 111 Law SCDB

.groupby(label) method

- Idea: split dataframe into groups that all have same value in column named label. E.g.,
  - `grouped = scdb.groupby('justiceName')`
  - grouped has many of same methods, indexing options as a dataframe
  - grouped.count() → dataframe with 60 columns (all but justice name) and 1 row per justiceName
  - grouped['docketID'] selects out that column
  - we plotted grouped['docketId'].count()
    - groupby type objects have a plot() method

A series and series groupby method

- nunique() is a method of true series, where it returns number of distinct values in the series (a number)
- nunique() is also a method of series-like groupby objects, where it returns actual series: How many were in each group.

POTPOURRI: FILES, LOOPS

File Open: Not needed for pandas!

```
fileref = open('SCDB_2019_01_justiceCentered_Citation.csv',
                      encoding='ISO-8859-1')
# Read from file with pandas preparing to exploit csv format
scdb = pandas.read_csv(fileref)
```

Could instead be simply:

```
scdb pandas.read_csv('SCDB_2019_01_justiceCentered_Citation.csv',
                      encoding='ISO-8859-1')
# But wanted to get file opening idea across; more files soon
```

while vs. for

Which of the following is true?

A. We can always rewrite any *for* loop using *while* construct
B. We can always rewrite any *while* loop using *for* construct
C. if *for* and *while* would both work, *for* usually cleaner/clearer
D. A and B
E. A and C

for to while conversion: range of numbers

```
for i in range(a, b, c):
    do stuff with i
```

```
i = a
while i < b:
    do stuff with i
    i += c
```

# wonder if this is true
# if a > b …

for to while conversion: general sequnce

```
for i in seq:
    do stuff with i
```

```
index = 0
while index < len(seq):
    i = seq[index]
    do stuff with i
    index += 1
```

```
# seq could be any string
# seq could be any list
```

for for's a jolly good construct

- Generally prefer for in cases where it will do the job
  - *Easier for human reader to understand*
  - Don't have to do work of initializing loop variable
  - *Don't have to remember to correctly increment loop variable*
- *In particular, use for for:*
  - Doing things fixed number of iterations when you can figure out that fixed number before start of loop
  - To access each element of a container (e.g., list) or of a string