MSDS 7333: Quantifying the World

Week 7: Python

Python: A house divided is sort of getting its act together

- Python 2 is slowly fading. Python 3 is rising
  - Python 2.7 is current
  - Python 3.6 is current
- Python 2.7 may be around for a while
  - There is no Python 2.8. Ever.
  - Many Unix and Unix Like systems use Python 2.7 as part of the OS
    - This includes Mac
- Learn to write code compatible for both version 2 and version 3.
  - A/k/a use Python 3 syntax!

Which to choose

- Python 3
- Python 3
- Python 3

- ONLY USE PYTHON 2.7 IF YOU ARE NOT IN CONTROL OF YOUR ENVIRONMENT!
- Python 2.6 also out there. Make your org upgrade to at least 2.7 - no excuses. (2.7 came out 10 years ago).

- All major libraries exist in 3.X

Back to the __future__

- Use the future module

```
from __future__ import print_statement, division
```

  - Allows Python 2 to behave like Python 3 (and backwards compatible)
  - MUST BE THE FIRST LINE
- Things to watch for
  - Print Statements
  - Errors (Catching)
  - Division
  - Integers
  - Unicode

MARK IT ZERO

- Python has some unique ‘features’ that are tough for new users
- If you specify a range, the left side is included, the right side is excluded:

  range(0,5)==0,1,2,3,4

  range(3,9)==3,4,5,6,7,8

  Awkward at first (I hated it). It becomes natural
- If you come from R, counting starts at 0.
  - It’s a computer thing. Computer science counts from 0.
  - The first row is row ZERO, not row 1
- List Comprehensions (more later)
- Lambda Expressions (no, developers were not high)

Resources

- http://python-future.org/compatible_idioms.html
- https://wiki.python.org/moin/Python2orPython3
- http://pandas.pydata.org/pandas-docs/version/0.21/
  (previous versions also exist—get at least into the high teens)
- https://docs.scipy.org/doc/numpy/reference/index.html
- https://www.anaconda.com/download/
  (includes Jupyter Notebook and Spyder)
- https://docs.python.org/3.6/
  (previous versions also available at that link)

With that unpleasantness behind us…

- Pandas is your data science workhorse
- Numpy is your other workhorse (behind the scenes)
- Think of Pandas as ‘tables’ in Python
    - SQL thinking works
    - Excel Spreadsheet thinking works
    - Know it, love it, use it.
- What makes Python equal to R
- Use Pandas. Get good at Pandas
- Dataframes make Data Science go

Getting Started

Conda	Description
conda create –n <name>	Create environment
source <name> activate	Switch to environment
source deactivate	Deactivate to environment
conda create –n <name> python=2.7	Create a python 2.7 environment

Virtual Environment	
python -m venv <name>	Create environment
source <name>/bin/activate	Switch to environment
source deactivate	Deactivate to environment
python -m venv <name> -p /path/to/other/python	Create a python 2.7 environment

Working on *nix (this includes you Mac kids)

- On most unix and linux systems (Mac OS is Unix) Python 2.7 is part of the OS. It cannot be removed or changed without significant impact.
- Installing a Python 3 creates a new install that is utilized by the ‘python3’ command. Installing packages is done by the ’pip3’ command.
- Inside your virtual environment the environment python is used
- When in doubt, type ‘python’ and see which version pops up (there are a lot of variables, including paths, bashrc, etc.…)

Using pip

Pip command	Description
pip install pandas	Install pandas
pip install pandas==0.18	Install a specific version
pip install –r requirements.txt	Install a list of packages (recursively) from a file called requirements.txt
pip freeze	List all packages in the current envirnment
pip freeze > output.txt	List all the packages and put them in a file called ‘output.txt’

Series, Dataframes, Panels

- A 1-dimensional object is a SERIES
  - If you return a single column from a dataframe, it is a series.
- A 2-Dimensional object is a DATAFRAME
  - Many ways to think of this, but it is a list of series
- A 3-Dimensional Object is a PANEL (and getting deprecated—use a multi-index)

data.iloc[1:5,2:4] returns a dataframe

data.iloc[1:5,2] returns a series

N x 0 -> series!

N x 1 -> dataframe!

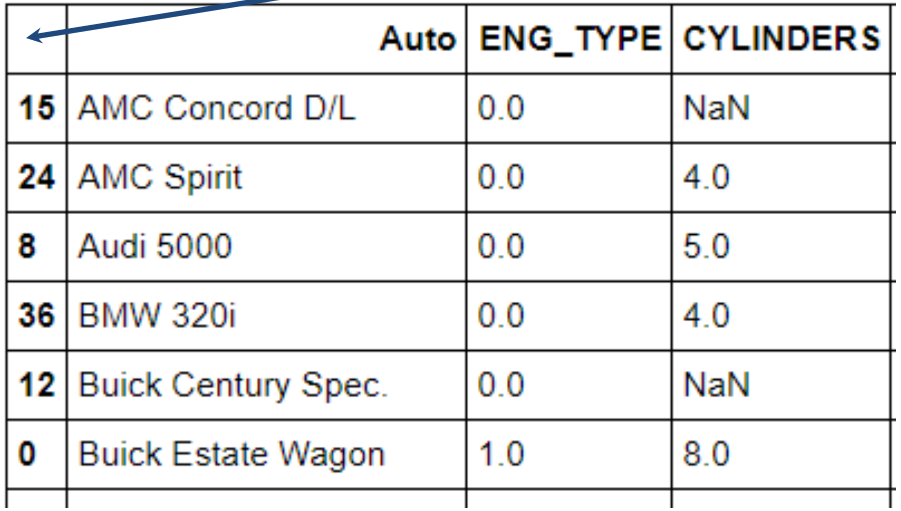
Numpy arrays also have this feature/issue.

Pandas is flexible with series vs dataframes (vs panels).

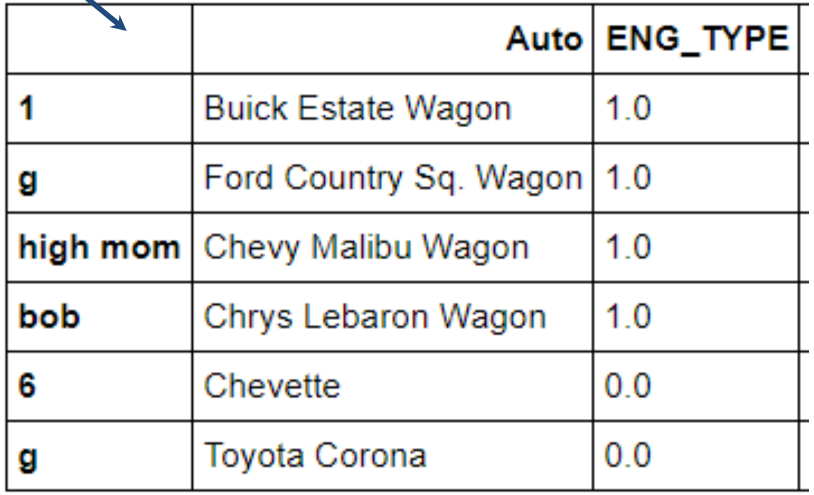
Watch out for Numpy array size checks!!

Series and Dataframe vs Array

- Both Series and Dataframe have a ‘label’ or index that can be modified
- An array just numeric location indices only
- Dataframe/Series index can be ANYTHING and UNORDERED



	Auto	ENG_TYPE	CYLINDERS
15	AMC Concord D/L	0.0	NaN
24	AMC Spirit	0.0	4.0
8	Audi 5000	0.0	5.0
36	BMW 320i	0.0	4.0
12	Buick Century Spec.	0.0	NaN
0	Buick Estate Wagon	1.0	8.0



	Auto	ENG_TYPE
1	Buick Estate Wagon	1.0
g	Ford Country Sq. Wagon	1.0
high mom	Chevy Malibu Wagon	1.0
bob	Chrys Lebaron Wagon	1.0
6	Chevette	0.0
g	Toyota Corona	0.0

Basic Pandas Utilities

- df.loc[<loc1>,<loc2]
  - Identifies data by columns, indexes
  - Can be used with qualifiers, column names, conditions
  - Example—give me the data from column ‘Make’ where column ‘MPG’ is greater than 30

```
df.loc[df[“MPG”]>30,”Make”]
```

- df.iloc[<iloc1>,<iloc2>]
  - Identifies columns and rows by location (NOT INDEX!!)

```
df.iloc[0,1]
```

Gives the first column, second row.

Great for loops, functions, etc.

More handy utilities

- Stat functions (dataframes/series):

```
df.max(), df.min(), df.median()
```

- Summary statistics for each column

```
df.describe()
```

- Array of unique values in the column (aka series)

```
df[‘<colname>’].unique()
```

In Class Exercise

Q1: What is the maximum and minimum mpg for Ford and Chevy cars with more than 150 HP?

Q2: What is the median weight of cars in produced in 1970, 1972, and 1974?

Q3: What are all the non-American cars with displacement >100 but not more than 200?

Q4: What is the average MPG for each year for 6 cylinder cars? Do not include 1974. Or Fords.

Remember last week?

- df.fillna(<val>)
- pd.get_dummies(<series>)
  - You will have to append to your dataframe!

Work these together

```
df[‘cylinders’].fillna(df[‘cylinders’].mean())
```

A test

```
[1]: data['CYLINDERS'].fillna(data['CYLINDERS'].mean())
[2]: data
```

Question: are the missing values filled or not?

NO!

```
data['CYLINDERS'].fillna(data['CYLINDERS'].mean())
```

Returns a filled in dataframe, but that dataframe wasn’t assigned to anything!

```
data['CYLINDERS'].fillna(data['CYLINDERS'].mean(),inplace=True)
```

or

```
data['CYLINDERS']=data['CYLINDERS'].fillna(data['CYLINDERS'].mean())
```

Lambda Expressions

- Seem redundant
- Get used many times with dataframes/apply/map
    - Adds speed
    - Avoids for loops
    - Breaks python 2/3 compatibility.

```
data.loc[:,'CYLINDERS':].apply(lambda x: x.max()-x.min())
```

List Comprehension

- Nothing to do with pandas
- Essential for python code
- YOU WILL ENCOUNTER THESE
- Optimized ‘For’ loops

```
doubled_odds = []
for n in numbers:
  if n % 2 == 1:
      doubled_odds.append(n * 2)

doubled_odds = [n * 2 for n in numbers if n % 2 == 1]
```

http://treyhunner.com/2015/12/python-list-comprehensions-now-in-color/

Jupyter Notebooks

- Using LaTex/Markdown
- Using Code
- Plotting Inline

Preparation for Unit 4 Case

- Changes to pandas.data.io
  - Deprecated (aka removed)
  - Use :

    ```
    from pandas_datareader import data as web
    ```

  - Also start and end dates must be ‘datetime’ objects:

    ```
    import datetime
    start=datetime.datetime(2018, 1, 25)
    ```