DANIELS
COLLEGE OF BUSINESS
UNIVERSITY of DENVER

Python for Data Analytics

STAT Module: Lesson 3

Training Manual

STAT: Lesson 3

Lecture for STAT Lesson 3

Mini Assignment STAT L3

Objectives Lesson 3

Statistical Analysis (STAT)

- Data Aggregation
  - Pandas and User Functions (apply vs agg)
  - Column-wise and multiple function application
  - Pivot Tables and Cross-Tabulation
- Presenting Data in an iPython Notebook

DANIELS COLLEGE OF BUSINESS UNIVERSITY of DENVER

Data Aggregation

Data aggregation refers to producing summary values from arrays. We have used mean, median, describe, sum and size previously with **groupby**.

```
np.random.seed(seed=1234) #run next statements for repeatable results
df = DataFrame({'state' : ['Ohio','California','California',
                           'Ohio','Ohio', 'California','Ohio',
                           'Ohio','California','California'],
                'year' : [2014,2014,2015,2014,2015,2014,
                          2015,2015,2014,2014],
                'data1' : np.random.rand(10),
                'data2' : np.random.randn(10)+10})
```

```
      data1      data2       state  year
0  0.191519  10.015696        Ohio  2014
1  0.622109   7.757315  California  2014
2  0.437728  11.150036  California  2015
3  0.785359  10.991946        Ohio  2014
4  0.779976  10.953324        Ohio  2015
```

Here is a slightly different version of the syntax with the column after **groupby.**

```
df.groupby('state').data1.mean()
```

```
state
California    0.633300
Ohio          0.567038
Name: data1, dtype: float64
```

Pandas Basic Statistics Functions

count	Number of non-null observations
sum	Sum of values
mean	Mean of values
mad	Mean absolute deviation
median	Arithmetic median of values
min	Minimum
max	Maximum
mode	Mode
abs	Absolute Value
prod	Product of values
std	Unbiased standard deviation
var	Unbiased variance
sem	Unbiased standard error of the mean
skew	Unbiased skewness (3rd moment)
kurt	Unbiased kurtosis (4th moment)
quantile	Sample quantile (value at %)
cumsum	Cumulative sum
cumprod	Cumulative product
cummax	Cumulative maximum
cummin	Cumulative minimum

Data Aggregation

We can use any of these basic statistics functions with the **`groupby`** object. For example, quantile or standard deviation.

```
grpState = df.groupby('state')
grpState.data1.quantile(0.9)
```

```
state
California    0.925257
Ohio          0.795267
Name: data1, dtype: float64
```

```
grpState.data2.std()
```

```
state
California    0.713797
Ohio          0.559076
Name: data2, dtype: float64
```

And we can use this syntax for multiple data columns.

```
grpState[['data1','data2']].std()
```

```
               data1     data2
state
California  0.244859  0.713797
Ohio        0.270238  0.559076
```

Which if these is NOT correct syntax to calculate the mean of data1 grouped by state?

```
1.df.groupby('state').data1.mean()
2.df.data1.groupby(df.state).mean()
3.df['data1'].groupby(df.state).mean()
4.df['data1'].groupby('state').mean()
```


Data Aggregation – apply vs agg

In Lesson 2 we had an example using **`apply`** to apply a user created function to each group. **`agg`** is similar to apply.

- **`apply`** applies the function to each group
- **`agg`** aggregates each column *for each group* down to a single value.

For simple function returns, there won't be a difference. The following both produce the same output. The use of **`agg`** is more common.

`agg` is a shortcut for **`aggregate`** which works the same way

```
def myRange1(group):
    return group.max() - group.min()
grpState['data1','data2'].apply(myRange1)

def myRange1(group):
    return group.max() - group.min()
grpState['data1','data2'].agg(myRange1)
```

```
              data1     data2
state
California  0.575997  1.456809
Ohio        0.568254  1.230687
```

Both produce same output

Data Aggregation – apply vs agg

But when the return is more complicated than single values, the **`apply`** output is easier to work with.

```
def myRange2(group):
    return {'range': group.max() - group.min()}
grpState.data1.apply(myRange2)
```

```
state
California  range    0.575997
Ohio        range    0.568254
Name: data1, dtype: float64
```

We can easily use .unstack( ) on this output but not on the agg output.

```
def myRange2(group):
    return {'range': group.max() - group.min()}
grpState.data1.agg(myRange2)
```

```
state
California    {u'range': 0.575996901584}
Ohio          {u'range': 0.568254118081}
Name: data1, dtype: object
```

Column-wise and multiple function application

Let's get a tips dataset from the R reshape2 package (originally from Bryant & Smith, 1995 textbook)

```
tips = pd.read_csv('tips.csv')
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

We'll add a column with the tip percentage of the total bill.

```
tips['tip_pct'] = tips.tip / tips.total_bill
tips.head()
```

	total_bill	tip	sex	smoker	day	time	size	tip_pct
0	16.99	1.01	Female	No	Sun	Dinner	2	0.059447
1	10.34	1.66	Male	No	Sun	Dinner	3	0.160542
2	21.01	3.50	Male	No	Sun	Dinner	3	0.166587
3	23.68	3.31	Male	No	Sun	Dinner	2	0.139780
4	24.59	3.61	Female	No	Sun	Dinner	4	0.146808

Note when you create a new column, you must use [] syntax for the column name

Column-wise and multiple function application

Now let's look at some aggregation using a different function depending on the column or using multiple functions at once.

First we'll group the tips by sex and smoker. Then specify the tip_pct column. And finally, note for many descriptive statistics you can pass the name of the function as a string or call the numpy function.

```
grpSexSmok = tips.groupby(['sex', 'smoker'])
grpSexSmok_pct = grpSexSmok['tip_pct']
grpSexSmok_pct.agg('mean')
```

```
sex     smoker
Female  No        0.156921
        Yes       0.182150
Male    No        0.160669
        Yes       0.152771
Name: tip_pct, dtype: float64
```

equivalent:
`grpSexSmok_pct.agg(np.mean)`

To return aggregated data in "unindexed" form, pass **`as_index=False`**.

```
tips.groupby(['sex', 'smoker'], as_index=False)['tip_pct'].agg('mean')
```

	sex	smoker	total_bill	tip	size	tip_pct
0	Female	No	18.105185	2.773519	2.592593	0.156921
1	Female	Yes	17.977879	2.931515	2.242424	0.182150
2	Male	No	19.791237	3.113402	2.711340	0.160669
3	Male	Yes	22.284500	3.051167	2.500000	0.152771

Column-wise and multiple function application

If you pass a list of function names instead, you get back a DataFrame with column names taken from the functions.

```
grpSexSmok_pct.agg(['mean', 'std', myRange1])
```

		mean	std	myRange1
sex	smoker			
Female	No	0.156921	0.036421	0.195876
	Yes	0.182150	0.071595	0.360233
Male	No	0.160669	0.041849	0.220186
	Yes	0.152771	0.090588	0.674707

You can update the name to be something instead of the function by passing (name, function) tuples:

```
grpSexSmok_pct.agg([('Average', 'mean'), ('Standard Deviation', 'std')])
```

		Average	Standard Deviation
sex	smoker		
Female	No	0.156921	0.036421
	Yes	0.182150	0.071595
Male	No	0.160669	0.041849
	Yes	0.152771	0.090588

Column-wise and multiple function application

With a DataFrame you can specify a list of functions to apply to all columns or different functions per column. Suppose we wanted to compute the same three statistics for the tip_pct and total_bill columns. We can save the functions in a list. Again if you want to change the name you can use a name, function tuple. The resulting DataFrame has hierarchical columns

```
functions = [('N','count'), 'mean', ('maximum','max')]
result = grpSexSmok['tip_pct', 'total_bill'].agg(functions)
result
```

		tip_pct			total_bill		
		N	mean	maximum	N	mean	maximum
sex	smoker						
Female	No	54	0.156921	0.252672	54	18.105185	35.83
	Yes	33	0.182150	0.416667	33	17.977879	44.30
Male	No	97	0.160669	0.291990	97	19.791237	48.33
	Yes	60	0.152771	0.710345	60	22.284500	50.81

```
result['tip_pct'] #get only the tip_pct column
```

		N	mean	maximum
sex	smoker			
Female	No	54	0.156921	0.252672
	Yes	33	0.182150	0.416667
Male	No	97	0.160669	0.291990
	Yes	60	0.152771	0.710345

Column-wise and multiple function application

Now, if you want to apply different functions to one or more of the columns, the trick is to pass a dict to **agg** that contains a mapping of column names to any of the function specifications listed so far.

```
grpSexSmok.agg({'tip' : np.max, 'size' : 'sum'})
```

Note you can use the numpy function or the name in quotes

		tip	size
sex	smoker		
Female	No	5.2	140
	Yes	6.5	74
Male	No	9.0	263
	Yes	10.0	150

```
grpSexSmok.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
                'size' : 'sum'})
```

		tip_pct				size
		min	max	mean	std	sum
sex	smoker					
Female	No	0.056797	0.252672	0.156921	0.036421	140
	Yes	0.056433	0.416667	0.182150	0.071595	74
Male	No	0.071804	0.291990	0.160669	0.041849	263
	Yes	0.035638	0.710345	0.152771	0.090588	150

Which code could have produced this output?

		tip	size
sex	smoker		
Female	No	5.2	2
	Yes	6.5	2
Male	No	9.0	2
	Yes	10.0	2

```
1. grpSexSmok.agg({'tip' : 'max', 'size' : 'median'})
2. grpSexSmok['tip'].agg([('N','tip'), ('maximum','size')])
```

DANIELS COLLEGE OF BUSINESS UNIVERSITY of DENVER

Pivot Tables

A pivot table is a data summarization tool that aggregates a table of data by one or more keys, arranging the data in a rectangle with some of the group keys along rows and some along the columns. Pivot tables in Python are created with the **`groupby`** facility combined with the **`reshape`** operation. DataFrame has a **`pivot_table`** method and there is also a top-level **`pandas.pivot_table`** function.

Here is the basic syntax:

```
df.pivot_table(data, rows (index=), columns)
```

DANIELS COLLEGE OF BUSINESS UNIVERSITY of DENVER

Pivot Tables

Lets compute a table of group means (the default **pivot_table** aggregation type) arranged by sex and smoker on the rows. Note we could have gotten the same output with groupby.

```
tips.pivot_table(index=['sex', 'smoker'])
```

sex	smoker	size	tip	tip_pct	total_bill
Female	No	2.592593	2.773519	0.156921	18.105185
	Yes	2.242424	2.931515	0.182150	17.977879
Male	No	2.711340	3.113402	0.160669	19.791237
	Yes	2.500000	3.051167	0.152771	22.284500

```
grpSexSmok= tips.groupby(['sex', 'smoker'])
grpSexSmok.agg('mean')
```

This groupby code
Produces the same results

Pivot Tables

Now suppose we want to aggregate only tip_pct and size and additionally group by day. We can put smoker in the columns and day in the rows.

```
tips.pivot_table(['tip_pct', 'size'], index=['sex', 'day'],
                 columns='smoker')
```

		tip_pct		size	
smoker		No	Yes	No	Yes
sex	day				
Female	Fri	0.165296	0.209129	2.500000	2.000000
	Sat	0.147993	0.163817	2.307692	2.200000
	Sun	0.165710	0.237075	3.071429	2.500000
	Thur	0.155971	0.163073	2.480000	2.428571
Male	Fri	0.138005	0.144730	2.000000	2.125000
	Sat	0.162132	0.139067	2.656250	2.629630
	Sun	0.158291	0.173964	2.883721	2.600000
	Thur	0.165706	0.164417	2.500000	2.300000

Pivot Tables

We can add partial totals by passing **`margins=True`**. This has the effect of adding all row and column labels, with corresponding values being the group statistics for **all** the data within a single tier. In this example the **All** values are means without taking into account smoker vs. non-smoker or any of the two levels of grouping on the rows.

```
tips.pivot_table(['tip_pct', 'size'], index=['sex', 'day'],
                 columns='smoker', margins=True)
```

		tip_pct			size		
smoker		No	Yes	All	No	Yes	All
sex	day						
Female	Fri	0.165296	0.209129	0.199388	2.500000	2.000000	2.111111
	Sat	0.147993	0.163817	0.156470	2.307692	2.200000	2.250000
	Sun	0.165710	0.237075	0.181569	3.071429	2.500000	2.944444
	Thur	0.155971	0.163073	0.157525	2.480000	2.428571	2.468750
Male	Fri	0.138005	0.144730	0.143385	2.000000	2.125000	2.100000
	Sat	0.162132	0.139067	0.151577	2.656250	2.629630	2.644068
	Sun	0.158291	0.173964	0.162344	2.883721	2.600000	2.810345
	Thur	0.165706	0.164417	0.165276	2.500000	2.300000	2.433333
All		0.159328	0.163196	0.160803	2.668874	2.408602	2.569672

Pivot Tables

To use a different aggregation function besides the default **`mean`**, pass it to **`aggfunc`**. For example, **`'count'`** will give you a cross-tabulation (count or frequency) of group sizes.

```
tips.pivot_table('tip_pct', index=['sex', 'smoker'],
                 columns='day', aggfunc='count', margins=True)
```

day		Fri	Sat	Sun	Thur	All
sex	smoker					
Female	No	2.0	13.0	14.0	25.0	54.0
	Yes	7.0	15.0	4.0	7.0	33.0
Male	No	2.0	32.0	43.0	20.0	97.0
	Yes	8.0	27.0	15.0	10.0	60.0
All		19.0	87.0	76.0	62.0	244.0

If some combinations are empty, you can pass a **fill_value**.

```
tips.pivot_table('size', index=['time', 'sex', 'smoker'],
                 columns='day', aggfunc='sum', fill_value=0)
```

day			Fri	Sat	Sun	Thur
time	sex	smoker				
Dinner	Female	No	2.0	30.0	43.0	2.0
		Yes	8.0	33.0	10.0	NaN
	Male	No	4.0	85.0	124.0	NaN
		Yes	12.0	71.0	39.0	NaN
Lunch	Female	No	3.0	NaN	NaN	60.0
		Yes	6.0	NaN	NaN	17.0
	Male	No	NaN	NaN	NaN	50.0
		Yes	5.0	NaN	NaN	23.0

day			Fri	Sat	Sun	Thur
time	sex	smoker				
Dinner	Female	No	2	30	43	2
		Yes	8	33	10	0
	Male	No	4	85	124	0
		Yes	12	71	39	0
Lunch	Female	No	3	0	0	60
		Yes	6	0	0	17
	Male	No	0	0	0	50
		Yes	5	0	0	23

Cross-tabulations: pd.crosstab

A cross-tabulation (or crosstab for short) is a special case of a pivot table that computes group *frequencies* from pandas.

The syntax is:

`pd.crosstab(rows, columns, margins=True)`

```
pd.crosstab(tips.sex, tips.day)
```

day	Fri	Sat	Sun	Thur
sex				
Female	9	28	18	32
Male	10	59	58	30

```
pd.crosstab(tips.sex,tips.day,margins=True)
```

day	Fri	Sat	Sun	Thur	All
sex					
Female	9	28	18	32	87
Male	10	59	58	30	157
All	19	87	76	62	244

Cross-tabulations: pd.crosstab

Just like with **`groupby`** and **`pivot_table`**, we can have multiple row groups or column groups.

```
pd.crosstab([tips.sex,tips.smoker],[tips.time, tips.day],
            margins=True)
```

time		Dinner				Lunch		All
day		Fri	Sat	Sun	Thur	Fri	Thur	
sex	smoker							
Female	No	1	13	14	1	1	24	54
	Yes	4	15	4	0	3	7	33
Male	No	2	32	43	0	0	20	97
	Yes	5	27	15	0	3	10	60
All		12	87	76	1	7	61	244

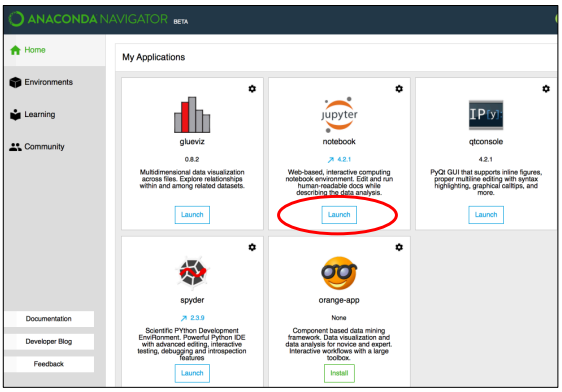
Which code could have produced this output?

sex	Female	Male
day		
Fri	9	10
Sat	28	59
Sun	18	58
Thur	32	30

1. `pd.crosstab(tips.sex, tips.day)`
2. `pd.crosstab(tips.day, tips.sex)`
3. `pd.crosstab([tips.day, tips.size], tips.sex)`

iPython Notebooks

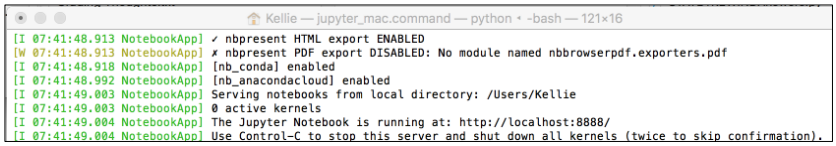
We can create and edit iPython notebooks using jupyter from Anaconda Navigator.



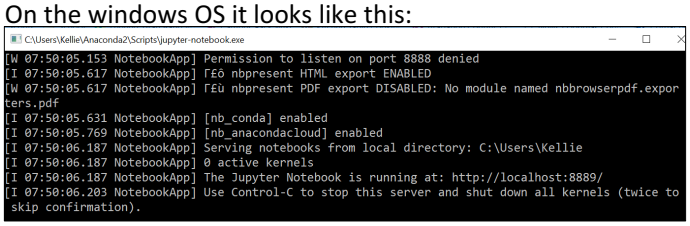
iPython Notebooks

When the environment is setting up, your computer will use its command prompt or terminal to define a local host. Keep these windows open or it will close the local server. The local server allows you to open .ipynb files in your browser and they are connected to a python kernel so you can run python code and get the output.

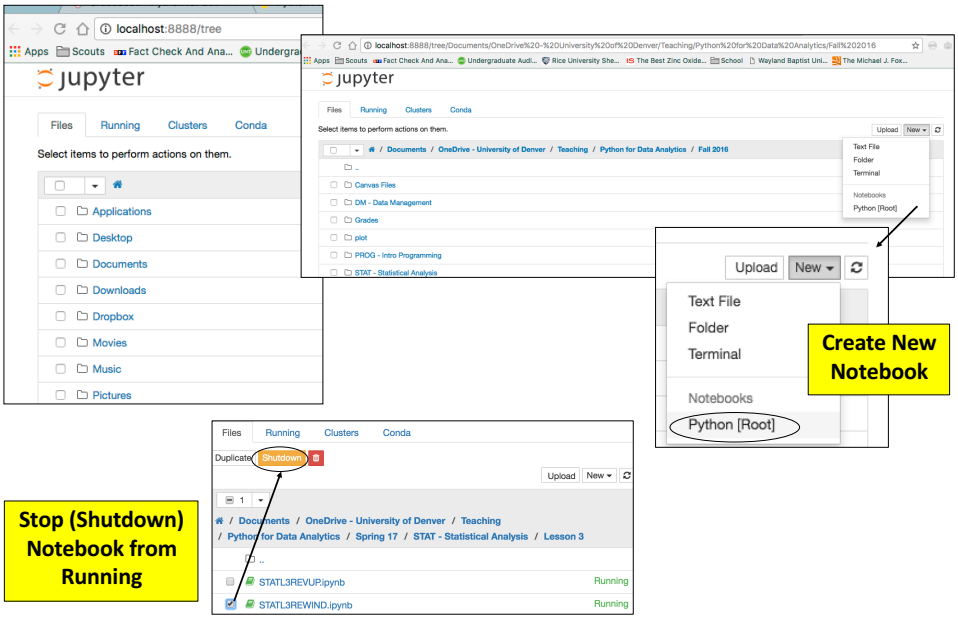
On the mac OS it looks like this:



On the windows OS it looks like this:

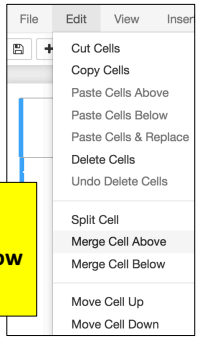


iPython Notebooks

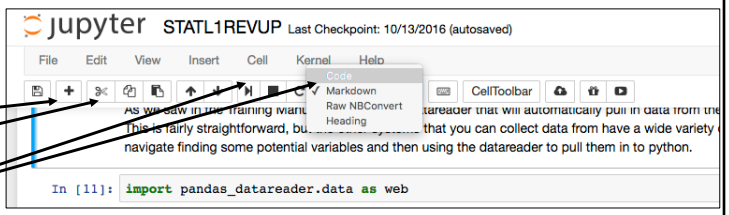


iPython Notebook

- Types of cells
- Code
  - normal python code
- Markdown
  - Formatted text
  - ## Heading # Bigger Heading ### Smaller Heading
  - More Syntax: https://guides.github.com/features/mastering-markdown/

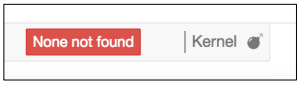


Other Editing Tools:
Split Cell
Merge Cell Above/Below
Move Cell Up/Down



Add a cell
Cut (Delete) a cell
Change the cell type
Run a cell

Tips for working with iPython



- What to do if jupyter loses its Kernel?
  - Set to Python Root (Kernel, Change kernel, Python[Root]
  - Save and Reopen
- Getting graphics to show
  - `#Needed to view graphics inline`
  - `%matplotlib inline`
- Reading in data
  - Make sure your data is in the same location as the .ipynb file
- Inserting pictures from file
  - Make sure your picture is in the same location as .ipynb
  - `#This is just code to display an image in iPython`
  - `from IPython.display import Image`
  - `Image(filename='FREDFindingVariableName.png')`

DANIELS COLLEGE OF BUSINESS UNIVERSITY of DENVER

Saving your iPython Notebook

Normal Saving

- File, Save and Checkpoint

Export to another format

- File, Download As, HTML (.html)
- This will contain all your markdown, code, and output

Which is the correct cell type to put a command to show a scatterplot in your Notebook?

1. Code

2. Markdown

3. Text

DANIELS COLLEGE OF BUSINESS UNIVERSITY of DENVER

Which is the correct cell type to put some text to describe content in your Notebook?

1. Code

2. Markdown

3. Text

DANIELS COLLEGE OF BUSINESS UNIVERSITY of DENVER

REWIND and REV UP (optional)

REWIND

- Additional Practice Problems + Extra Credit

REV UP

- Reading in numeric data with % at the end
  - Using strip()
  - Using a function and apply
  - Reading in data with a function specified as the converter
- Reading in numeric data with embedded commas
- Comparing Methods of Filtering/Selecting Data
  - Using numbers to slice/filter certain rows or columns
  - Using index labels to filter certain rows
  - Using criteria to select certain rows
  - Using multiple criteria to select certain rows
  - Returning Series instead of DataFrame

DANIELS COLLEGE OF BUSINESS UNIVERSITY of DENVER