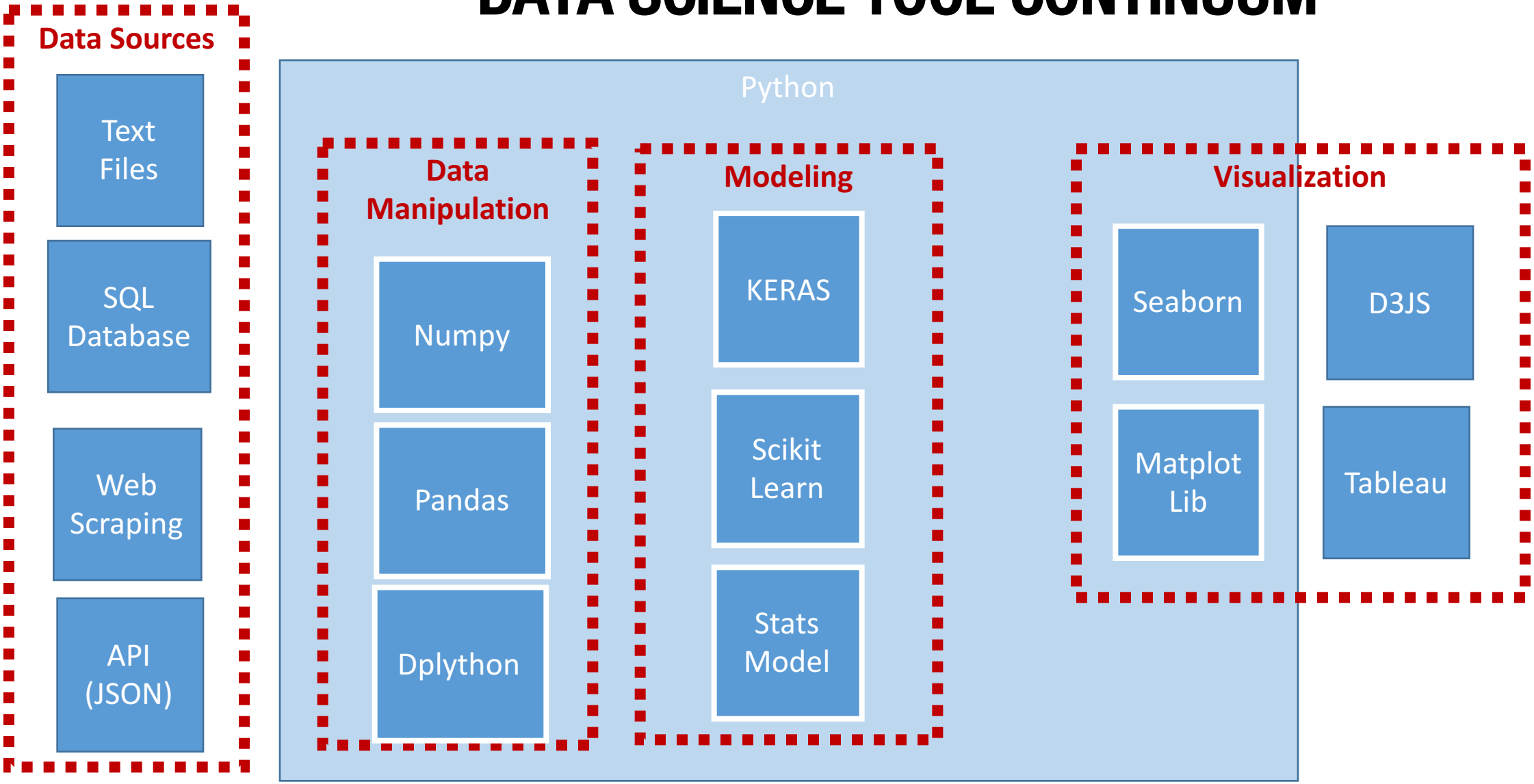
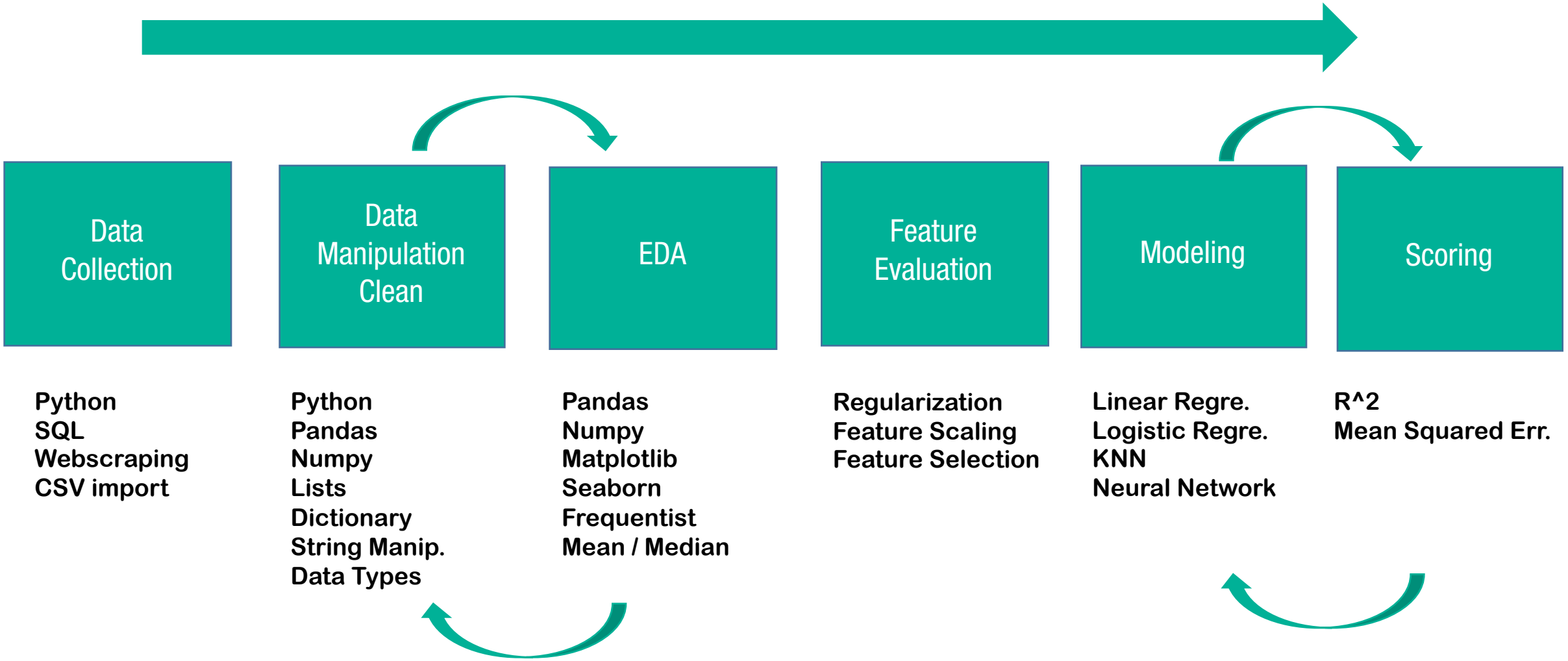
DATA SCIENCE NOTES

Because I forget things all the time

DATA SCIENCE TOOL CONTINUUM



DATA SCIENCE PROCESSING



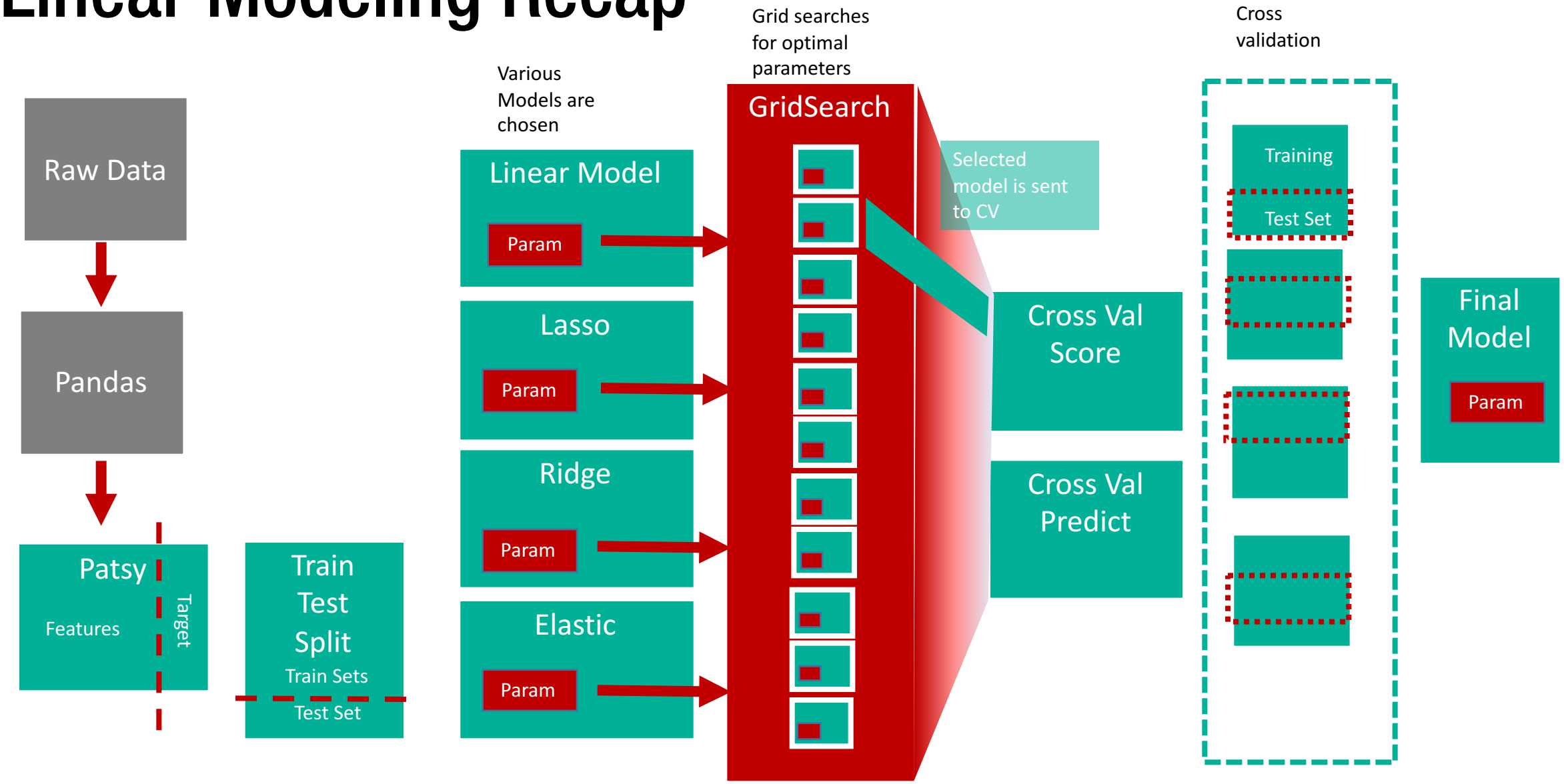
DATA SCIENCE STUDIES

	Unsupervised	**Supervised** **(have answers / history)**

Continuous	• Clustering & Dimensionality Reduction
	  • SVD
	  • PCA
	  • K-means
		• Regression
		  • Linear
		  • Ridge
		  • Lasso
		  • Polynomial
		• Decision Trees
		• Random Forests

Categorical
(Fixed options)
	• Association Analysis	• Classification
		  • KNN
		  • Trees
		  • Logistic
		  • Naïve-Bayes
		  • SVM

Linear Modeling Recap



Schedule 12 Weeks

Schedule Week1

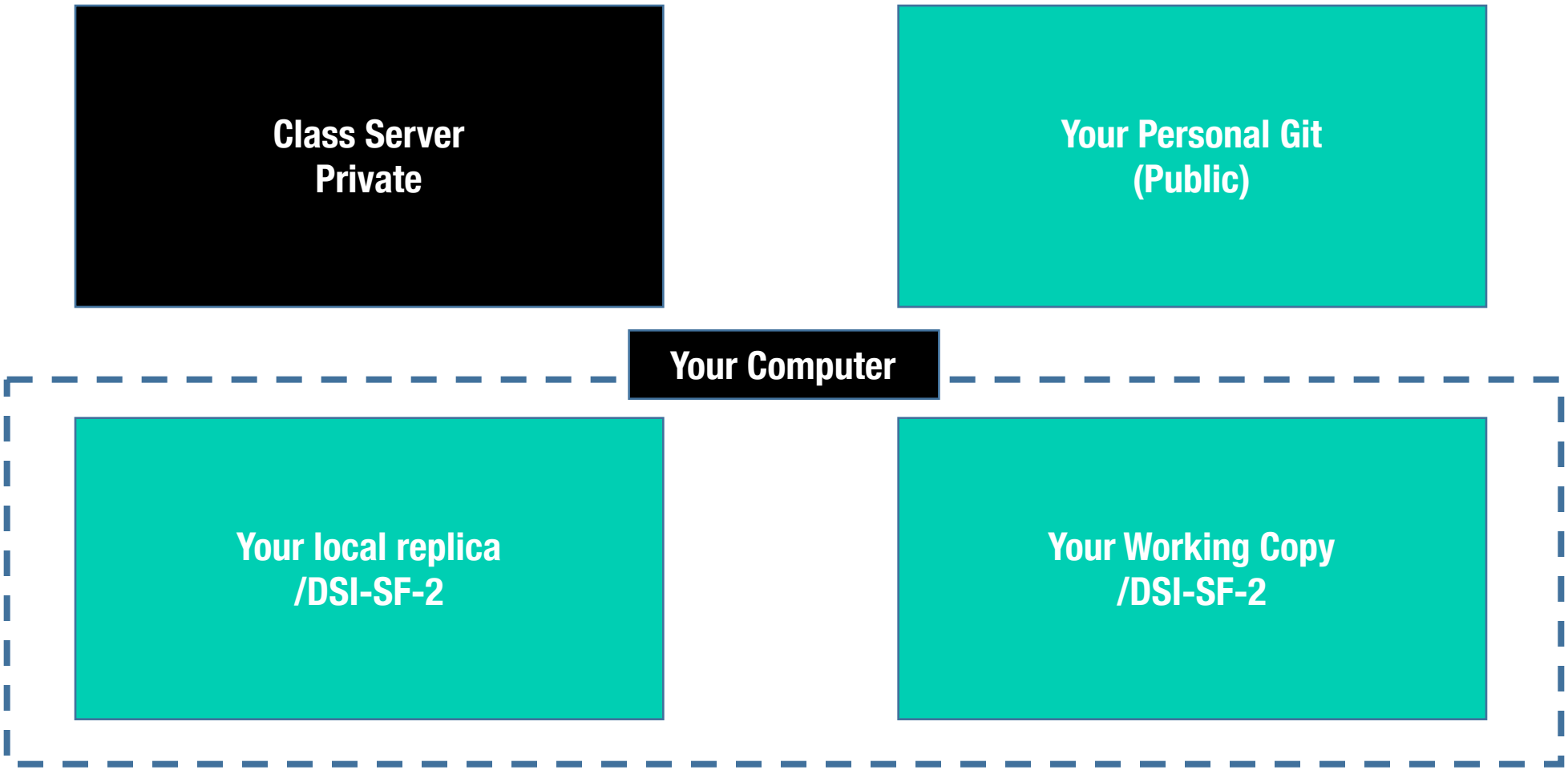
- Day 1
  - 1.1 Command Line
  - 1.2 Intro to Git
  - 1.3 Data Types
  - 1.4 Lists & Dictionaries
- Day 2
  - 2.1 Control Flows
  - 2.2 Python Functions Lab
  - 2.3 Iteration
  - 2.4 Python Movies Lab
- Day 3
  - 3.1 Outcomes: Intro
  - 3.2 List Comprehensions
  - 3.3 Numpy and Distributions
  - 3.4 [Lab]
- Day 4
  - 4.1 Math Primer
  - 4.2 Numpy Lab
  - 4.3 Data Vizualization Pt. 14.4 [Lab]

Week 2

- Day 1
  - 1.1 Intro to Study Design
  - 1.2 Basic Statistics
  - 1.3 Pandas DataFrames
  - 1.4 Numpy & Pandas Lab
- Day 2
  - 2.1 Pandas Indexing
  - 2.2 Pandas Computation Lab
  - 2.3 Intro to Data Cleaning
  - 2.4 Data Cleaning Lab
- Day 3
  - 3.1 Outcomes: Job Search Workshop
  - 3.2 Missing Data Lab
  - 3.3 Pandas Pivot Tables
  - 3.4 Pivot Tables Lab
- Day 4
  - 4.1 Pandas Grouping
  - 4.2 Split-Apply-Combine Lab
  - 4.3 Pandas Joins
  - 4.4 Joins Lab

Day 1: GIT and GITHUB

GIT SETUP


Class Server
Private
Your Personal Git
(Public)
Your Computer
Your local replica
/DSI-SF-2
Your Working Copy
/DSI-SF-2

What's in your directory?

- GET IT FROM THE SERVER
  - Git pull
- WHATS THE STATUS?
  - Git status
- PUT INTO STAGING
  - Git add [file name]
- CONFIRM ITS READY
  - Git commit
- WHOOPS, TIME TO START OVER
  - Git fetch all
  - Git reset HEAD origin/master
- I NEED TO GET RID OF STUFF
  - Git rm [filename]

Day 1: Command Line

Command Line Part 1

- USE TERMINAL
  - ⌘ (Command) + Space
  - "Terminal"
  - Enter
- ABSOLUTE PATH:
  - /Users/[your username]/mydir
- RELATIVE PATH:
  - ../mydir
- HOME DIRECTORY:
  - cd ~\
- CHANGE DIRECTORY:
  - Cd [directory]
  - Cd mydirectory/mysubdirectory
- DESKTOP:
  - cd ~/desktop
- WHERE AM I?
  - pwd – current working directory

What's in your directory?

- LIST IT OUT:
  - Ls
  - Ls /applications
- MAKE DIRECTORY
  - Mkdir
- CREATE EMPTY FILE:
  - Touch file
- REMOVE A FILE:
  - Rm file

- SORTA SELECTING:
  - * = wildcard
  - Ls *i*
  - Rm *d*

Python: Datatypes

What's your Type?

- INTEGER - int
  - 1
  - 235897
- STRING - str
  - 'this is a string'
  - "this is a string with dbl quotes"
- TUPLE - tuple
  - (1,2,4)
  - ('1','string',4.0, 5)
- LIST - list
  - [1, 3, 4, 5, 6]
  - ['curry','klay','durant', 9999]
- DICTIONARY – dict
  - { 'name': 'mcwolfsy', 'age': 45, 'species': 'cat'}
- FLOAT – float
  - 1.0
  - 33.2405898

Notes with Strings

- STRINGS CAN BE INDEXED
  - a = ‘Hello there’
  - a[3] = ‘l’
- SLICING
  - a = ‘redbluegreen’
  - a[4:8] = ‘blue’
  - a[-1] = ‘n’

- STRINGS CAN BE MULTIPLIED
  - “are we there yet” * 3 =
  - “are we there yet are we there yet are we there yet”
- STRINGS CAN BE SPLIT
  - b = “my/name/is/what?”
  - c = b.split(‘/’)å
    - c = [’my’,’name’,’is’,’what’]
- LEADING SPACES
  - d = ‘ uhoh ‘
  - d.strip() = ‘uhoh’

LISTS ON LISTS ON LISTS

- HOW TO MAKE
- Need [ ]
- Can mix values
- a = [1, 2, ‘string’, true]

- SORT A LIST (IN PLACE)
  - newlist.sort()
- REVERSE THE ORDER
  - Newlist.reverse()

LISTS ON LISTS

- MAKE A BLANK
  - my_new_list = [ ]
- ADD AN ITEM
  - newlist.append(3)
- ADD A NESTED LIST
  - newlist.append([3,3])
- EXTEND A LIST
  - newlist.extend([3,3])
- INSERT NEW VALUE
  - Newlist.insert(0 , ‘newitem’); 0 is index
- COUNT ITEMS IN LIST:
  - Newlist.count(‘thisstring’)
- WHAT INDEX IS ___
  - Newlist.index(‘frank’)
- REMOVE A VALUE
  - Newlist.remove(‘uglyitem’)
- REMOVE LAST ELEMENT
  - Newlist.pop()

DICTIONARIES

- CREATE A BLANK
  - Newdict = {}
- CREATE WITH VALUES
  - Newdict = { 'a' : 1, 'b' : 2}
- ONCE MADE, ADD A NEW VALUE
  - Newlist['newkey'] = 'some value'
- I HAVE THE KEY, WHATS THE VALUE?
  - Newlist['thiskey']
  - Newlist.get('thiskey')
- WHAT KEYS ARE THERE?
  - Newdict.keys()
- DOES IT HAVE 'THIS' KEY?
  - Newdict.has_key('this')
- WHAT VALUES ARE THERE?
  - Newdict.values()
- WHAT KEYS AND VALUES ARE THERE?
  - Newdict.items()

DICTIONARIES

- JOIN TWO DICTIONARIES
  - Newdict = { ’a’ : 1, ‘b’ : 2}
  - Seconddict = {‘c’:3}
  - Newdict.update(seconddict)
- CLEAR OUT THE DICTIONARY
  - Newlist.clear()

- WHAT KEYS ARE THERE?
  - Newdict.keys()
- WHAT VALUES ARE THERE?
  - Newdict.values()
- WHAT KEYS AND VALUES ARE THERE?
  - Newdict.items()

Python: IF need logic THEN check this section

IF

- IF
  - If condition :
    - Action
  - Else:
    - Action

- IF ELSE IF
  - If condition:
    - Action
  - Elif:
    - Action
  - Elif:
    - Action
  - Else:
    - action

Python: Functions

Function Anatomy

- def my_function (input1, input2=defaultvalue):
  - Code is indented ….
- In some other python script somewhere:
  - a = 3
  - b = 4
  - my_function(a, b)

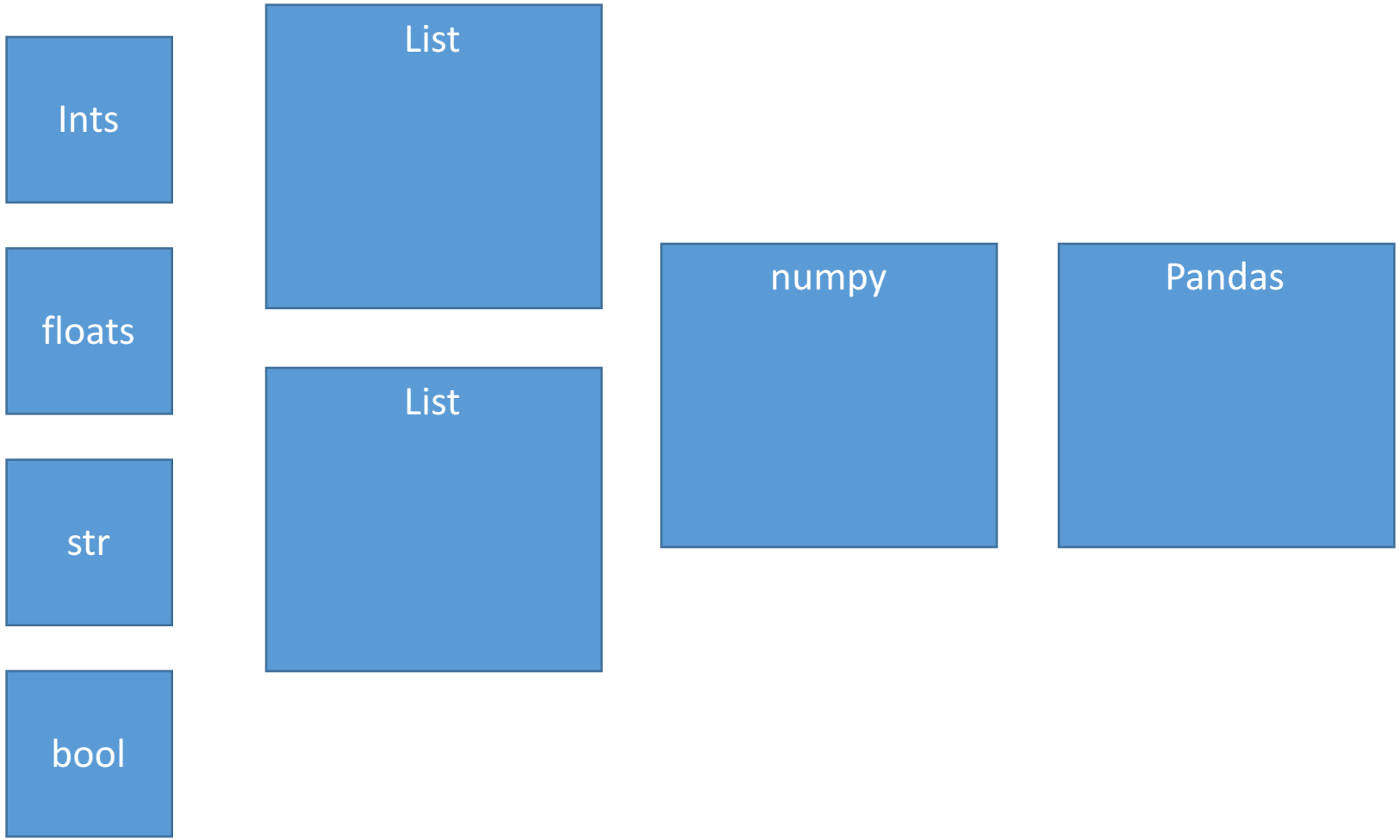
- def my_function (input1, input2):
  - Code is indented ….
  - Return out3 (optional)
- In some other python script somewhere:
  - a = 3
  - b = 4
  - var = my_function(a,b)
  - Print var

Function Anatomy

- **def my_function (input1, input2=defaultvalue):**
  - Code is indented ….
- In some other python script somewhere:
  - a = 3
  - b = 4
  - **my_function(a, b)**

- **def my_function (input1, input2):**
  - Code is indented ….
  - **Return out3 (optional)**
- In some other python script somewhere:
  - a = 3
  - b = 4
  - **var = my_function(a,b)**
  - **Print var**

Python Overview


Ints
floats
str
bool
List
List
numpy
Pandas

Day 3 Intro to Stats

MEAN = Average

- List of **Even** numbers
- **a = [1,2,3,4,5,6]**
- **mean = sum(a) / len(a)**
  - Without numpy
- **a = np.array([1,2,3,4,5,6])**
- **a.mean()**
  - With numpy

- List of **Odd** numbers
- **a = [1,2,3,4,5,6,7]**
- **mean = sum(a) / len(a)**
  - Without numpy
- **a = np.array([1,2,3,4,5,6,7])**
- **a.mean()**
  - With numpy

MEDIAN = Middle Numbers

- List of **Even** numbers
- **a = [1,2,3,4,5,6]**
- **Median_index = int(len(a)/2)**
- **a = [median_index]**
  - Without numpy
- **a = np.array([1,2,3,4,5,6])**
- **a.median**
  - With numpy

- List of **Odd** numbers
- **a = [1,2,3,4,5,6,7]**
- **Median_index = int(len(a)/2)**
- **a = [median_index]**
  - Without numpy
- **a = np.array([1,2,3,4,5,6,7])**
- **a.median()**
  - With numpy

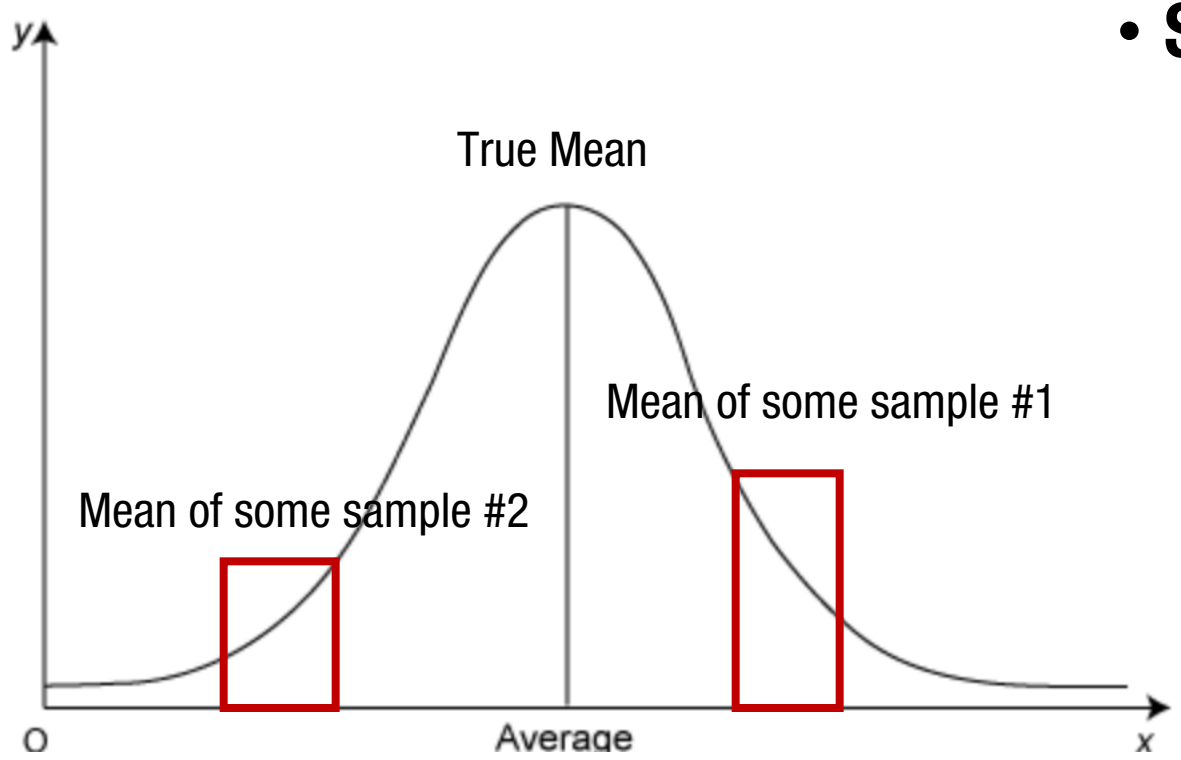
Day 5: Central Limit Theorem

Central Limit Theorem

- In probability theory, the central limit theorem (CLT) states that, given certain conditions, the arithmetic mean of a sufficiently large number of iterates of independent random variables, each with a well-defined (finite) expected value and finite variance, will be approximately normally distributed, regardless of the underlying distribution.

- Short Version:
  - An entire dataset has a normal distribution
  - As I take subsets and calculate means(avg), as I continually sample and sample, the mean(avg) will approach the standard distribution and the true mean

Central Limit Theorem



- **Short Version:**
  - An entire dataset has a normal distribution
  - As I take subsets and calculate means(avg), as I continually sample and sample, the mean(avg) will approach the standard distribution and the true mean

Central Limit Theorem

- **What does this mean?**
- It means that if you compute a mean of samples from ANY *true population distribution*, that mean will be normally distributed with
- **mean = mean of true population mean** and
- **standard deviation = standard error of the population mean**
- where a **standard error = (standard deviation) / sqrt(N)**
- check out:
- http://blog.vctr.me/posts/central-limit-theorem.html

Day 5: Experimental Design

Hypothesis

- **Hypotheses in industry**
- Hypothesis testing is not limited to the academic setting. It happens all the time in the business world and properly carrying it out is critical to the success of most companies.
- Consider the following hypotheses:

Testing hypotheses

- **Testing hypotheses**
- Hypotheses have corresponding, broader questions. The first hypothesis from the previous slide, for example, corresponds to the question "Does gender affect the likelihood of purchasing our product?"
- This may seem trivial, but it is critical to correctly frame and specify your hypothesis and the question that you are going to answer.
- If the hypothesis is too broad an un-specific, it is less possible to come up with a feasible test and get a result.

Testing hypotheses

- **Framing and setting expectations for a test**
- A big difference between the academic world and the business world is that in a professional setting hypothesis testing will be **much** more time-constrained.
- There is a convenient backronym called **SMART**:, while not originally devised for experimental design, it is useful for remembering good practices:
  - **Specific**
  - **Measurable**
  - **Achievable**
  - **Relevant**
  - **Time/Cost Limited**

Be Specific

- Consider the following hypothesis again:
- **Hypothesis**: If our product was smaller, more women would pay more to purchase it.
- **Revised hypothesis**: Does a 20% smaller product size increase percieved value of the product among women, as measured by survey responses and increased revenue among that demographic?
  - The more specific the hypothesis the better. When a hypothesis is specific:
  - Tests are easier to construct
  - Tests are more likely to be viable/feasible
  - A test is more likely to answer the question
  - There will be fewer pitfalls in the statistical analysis of the test

Ensure the hypothesis is measurable

- Without specific and **pre-determined metrics** to evaluate a test, you will not be able to draw conclusions or answer your question.
- Take the revised hypothesis from before:
- **"Does a 20% smaller product size increase percieved value of the product among women, as measured by survey responses and increased revenue among that demographic?"**
- Imagine we have the specifically smaller version of the product ready. How could we effectively measure this to answer the question?

Ensure a test is achievable

- You could have the best idea for an experiment in the world, but if it is going to take 5 years to carry out the test it is unlikely anyone will sign off on it.
- Say, for example, you believed that the smaller version of your product would result in women buying the product for $5 more, but in this scenario the smaller version does not exist. The engineering team says it would take at least a year and significant manpower for them to prototype the device. This test is not reasonably achieveable.

Ensure the hypothesis is relevant to the goals of the business

- Imagine now that women currently comprise only 1% of people currently purchasing your product. Is answering the question of increased product value for women still relevant to the success of the company?

Time-box and limit cost ahead of time

- If you do not specify at the beginning exactly how much time and resources you are willing to spend on your experiment, then it is easy to end up wasting time and money when things don't go smoothly.
  - Have due dates
  - Your salary is the cost to the company
  - Problems can be deep with many layers

Important experimental concepts

- **Reproducibility**
- **Randomization**
- **Control Conditions**

Reproducibility

- **Reproducibility** is necessary for the validity of an experiment. A reproducable experiement essentially means that the steps carried out by the experimenters can be repeated and achieve the same results. Obviously, this is not always possible when one is iterating on a product over time, but it is important to prioritize clarity of the process and documentation.

Randomization

- **Randomization** is important for avoiding biased results. If you were running a split test, for example, you would randomly assign people into arm A and arm B rather than assigning females into arm A and males into arm B.

Control conditions

- **Control conditions** ensure that you are comparing the "experimental condition" in your experiment to an existing baseline. From the example above, you would want to compare the smaller product to the original sized product, not the smaller product to a new bigger version of the product.

Day 5: Frequentist Statistics

Control conditions

- **Control conditions** ensure that you are comparing the "experimental condition" in your experiment to an existing baseline. From the example above, you would want to compare the smaller product to the original sized product, not the smaller product to a new bigger version of the product.

Day 5: Pandas Data Syntax

Getting it out of the DataFrame

- My_dframe.iloc [ ]
  - Takes absolute numbers
- My_dframe.loc[ ]
  - Takes lables
- My_dframe.ix ( )
  - Can take either label or number
- My_dframe.sales
  - Will give you a panda.series
- My_dframe.sales.values
  - Will give you a python list

- My_dframe[‘Sales’]
  - Take one column of data
- My_dframe[[‘sales’,’total’]]
  - More than one column

Complex Pandas Referencing, mixing rows and columns

- My_dframe.iloc [123:126, [1,2]]
  - Gives 4 rows from 123-126
    - 123
    - 124
    - 125
    - 126
  - Only two fields, columns 1 and 2
- My_dframe.iloc [[123,126], [1,2]]
  - Gives 2 rows 123 and 126 individually
    - 123
    - 126

Changing Values in the columss

- **My_dframe.my_field = a column of values or 1 value for all**
  - Doesn't reliably work
  - Only really works without filters

- **My_dframe.iloc [rows,columns] = a column of values or 1 value for all**
- My_dframe.iloc [[123,126], [1,2]]
  - Gives 2 rows 123 and 126 individually
    - 123
    - 126

Adding Columns

- My_dframe
  - A: [1 2 3 4]
  - B: [5 6 7 8]
- If we put My_dframe[‘C’] == 12
  - A: [1 2 3 4]
  - B: [5 6 7 8]
  - C: [12 12 12 12]

- boston.r

Day 6: EDA – Exploratory Data Analysis

Look at it before it you summarize

Data Dictionary

- Describe your data for your own reference
- Sample
  - CRIM: per capita crime rate by town
  - ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
  - INDUS: proportion of non-retail business acres per town
  - CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
  - NOX: nitric oxides concentration (parts per 10 million)
  - RM: average number of rooms per dwelling
  - AGE: proportion of owner-occupied units built prior to 1940
  - DIS: weighted distances to five Boston employment centres

About that data…

- Df = pd.Dataframe(…some data...)
- Df.head() – top rows
- Df.tail() – bottom rows
- Df.info() – field names and data types
- Df.describe() - provides general stats mean, max, min
- Df.shape – size of the matrix (property not method)
- Df.index
- Df[‘field’].unique

Unwanted Columns

- My_data.columns – find all the column/field names in the data frame
- My_data.drop('name of col', axis = 1, inplace = True)
- Can also remove rows
  - Will actually interrupt numbering 1,2,4,5,6
  - .iloc(3) – integer indexing will do absolute position iloc (3) is the 3rd element no matter the index
  - .loc(3) – will not be found. Looking for row by name (3 doesnt exist anymore)
- Df.reset_index

Masking the Data

- Make Diagram here

Masking the Data

- boston.head()
- Sample of Masks:
  - charles_river_only = (boston.CHAS == 1) & (boston.TAX<400)
  - column_mask = [col for col in boston.columns if len(col) == 3]
- Apply the mask
  - tmp = boston.loc[charles_river_only, column_mask].head(3)

Pandas Reminder

- Panda_var[rows, columns]
- Axis = 0 : rows
- Axis = 1 : columns

- Dataframe.apply()
  - Applies to

Bad Column names, Bad

- My_data.rename(columns={ ‘original’ :’new col name’
- ….}
- , inplace = True)

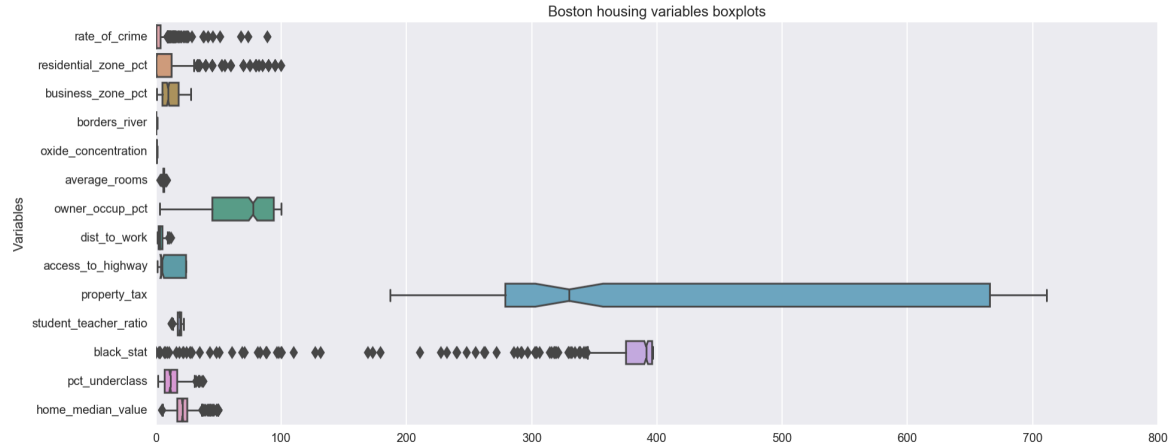
- boston.rename(columns={
  'CRIM':'rate_of_crime',
  'ZN':'residential_zone_pct',
  'INDUS':'business_zone_pct',
  'CHAS':'borders_river',
  'NOX':'oxide_concentration',
  'RM':'average_rooms',},
  inplace=True)

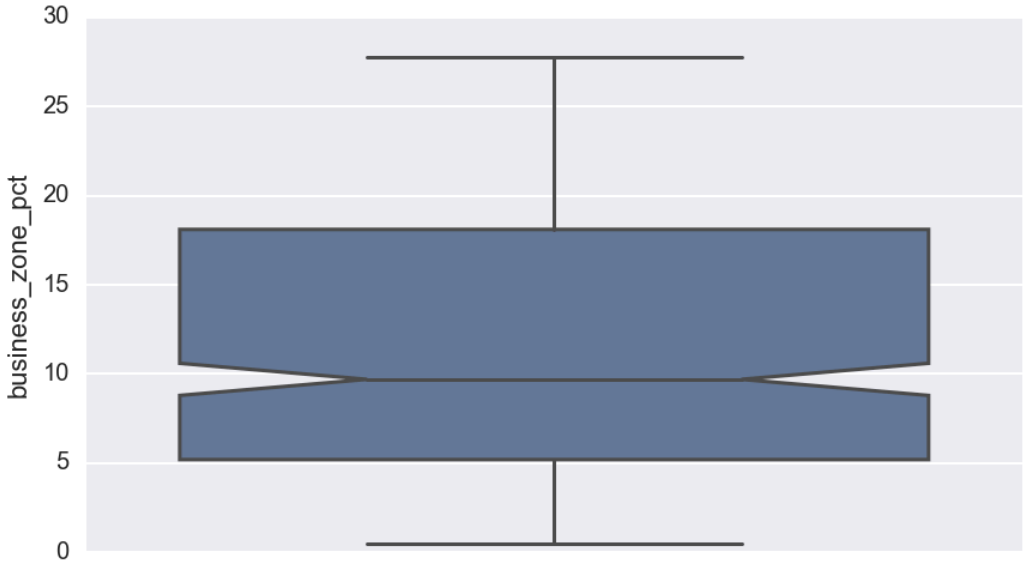
Seaborne Sample 1

- Step 1: fig = plt.figure(figsize = (7,4))
  - Sets the overall container size
- Step 2: ax = fig.gca()
  - Pulls out a area to write for a plot
- Step 3: ax = sns.boxplot(my_dataframe.my_field, orient='v', fliersize =8, linewidth = 1.5, notch=True, saturation = 0.5, ax=ax)
  - Will do the actual box plot
- Step 4: ax.set_ylabel('rate of crime')
  - Will add Y labels to the graph
- Step 5: plt.show()

Sample of the Boxplot

- Lines at top and bottom are the upper and lower bounds of the range
- Box starts at Quartile sections
- Notch represents confidence level
- Middle line represents the median



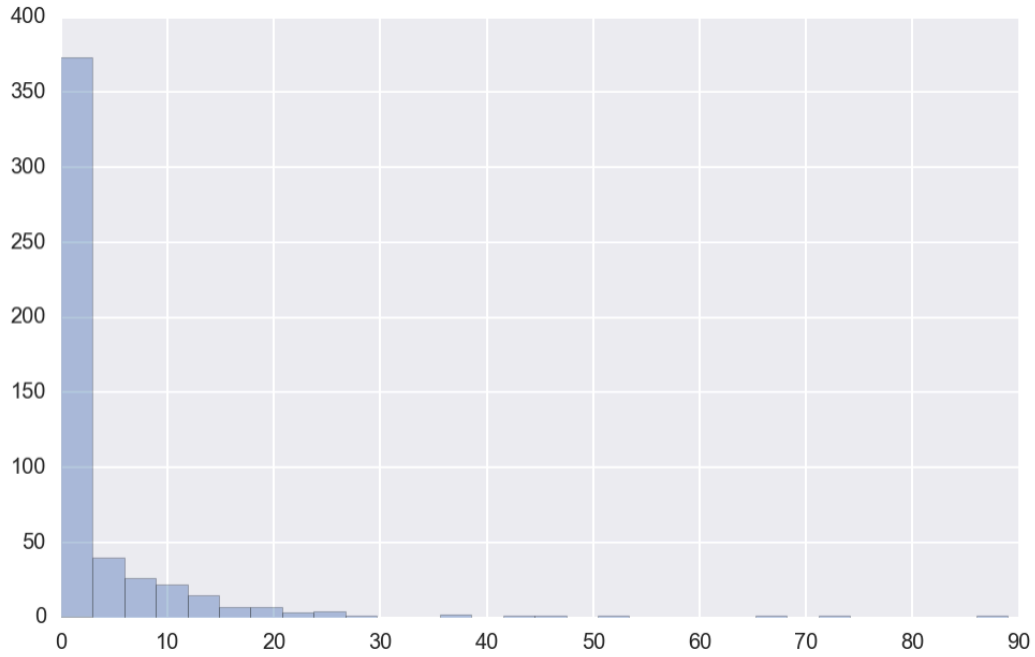


Seaborne Sample 2 – Dist Plot

- fig = plt.figure(figsize = (8,5))
- ax = fig.gca()
- ax = sns.distplot(roc, bins=30, kde=False)
- plt.show( )

Sample of the Distplot

- Shows distributions of datasets
- The “bins” or how the counts are grouped can be set in the options. Bins = 5 would be fewer, and Bins = 40 would be a lot of vertical columns



Covariance

- Given sample size N, and two datasets X and Y

$$\text{covariance}(X, Y) = \sum_{i=1}^{N} \frac{(X - \bar{X})(Y - \bar{Y})}{N}$$

- The covariance is a measure of the “relatedness” between variables. It is literally the sum of deviations from the mean of X times deviations from the mean of Y adjusted by the sample size N.
- **np.cov(pct_under, med_val, bias =True)**
  - Bias =False will make the formula divide by N-1. This will assume we are using a sample of a larger subset and need to compensate for the uncertainty

Correlation of Variables

- Given sample size N. and two datasets X and Y

$$\text{pearson correlation } r = cor(X, Y) = \frac{cov(X, Y)}{std(X)std(Y)}$$

- What the pearson correlation does is in fact directly related to what we did above during the normalization procedure. We are taking the covariance and dividing it by the product of the standard deviations of X and Y. This adjusts the value we get out by the variance of the variables so that r must fall between -1 and 1

Correlation Matrix

- Given sample size N, and two datasets X and Y
- My_dframe.corr()
- Will create matrix of all fields with the r values:

	rate_of_crime	residential_zone_pct	business_zone_pct
rate_of_crime	1.000000	-0.200469	0.406583
residential_zone_pct	-0.200469	1.000000	-0.533828
business_zone_pct	0.406583	-0.533828	1.000000
borders_river	-0.055892	-0.042697	0.062938

- np.corrcoef(pct_under, med_val, bias = True)
  - Bias = false divides by n-1 instead of n

Day 6: Data Cleaning in Pandas

Dataframes.apply

Pandas Dtype

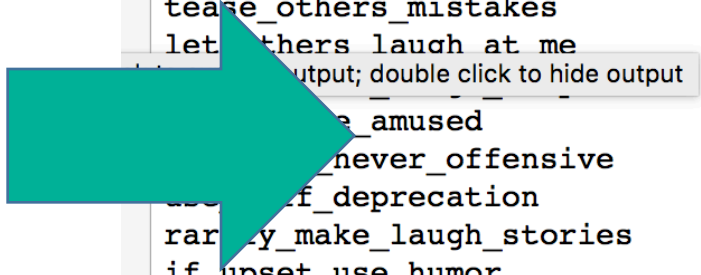
Pandas Series

Columns Renaming with a Dictionary

Change these headings

```
print hsq_clean.describe().loc['std',]
```

```
Q1                      1.075782
Q2                      1.112898
Q3                      1.167877
Q4                      1.160252
Q5                      1.061281
Q6                      0.979315
Q7                      1.099974
Q8                      1.231380
Q9                      1.224530
Q10                     1.205013
Q11                     1.249987
```



To these headings

```
hsq_wide.columns
hsq_wide.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1071 entries, 0 to 1070
Data columns (total 40 columns):
usually_dont_laugh                  1071 non-null int64
if_depressed_use_humor              1071 non-null int64
tease_others_mistakes               1071 non-null int64
let  thers laugh at me              1071 non-null int64
utput; double click to hide output  1071 non-null int64
 amused                             1071 non-null int64
 never_offensive                    1071 non-null int64
 f_deprecation                      1071 non-null int64
rar  y_make_laugh_stories           1071 non-null int64
if_upset_use_humor                  1071 non-null int64
dont_care_impact_jokes              1071 non-null int64
self_deprecate_to_befriend          1071 non-null int64
laugh_alot_with_friends             1071 non-null int64
humorous_outlook_improves_mood      1071 non-null int64
```

Columns Renaming with a Dictionary

- column_change = {
  'Q1':'usually_dont_laugh',
  'Q2':'if_depressed_use_humor',
  'Q3':'tease_others_mistakes',
  'Q4':'let_others_laugh_at_me',
  'Q5':'make_others_laugh_easy',
  'Q6':'when_alone_amused'…}

- hsq_wide = hsq_clean.rename(columns=**column_change)**

Pandas Reminder

- Panda_var[row units, column units]
- When using options in functions:
  - Axis = 0 : Columns (selecting all row units in a column)
  - Axis = 1 : rows (selecting all column units in a row)
- To fill later

Pandas Data Types (Dtype)

- Float
- Int
- Bool
- Datetime64
- Timedelta
- Category
- Object** - most common and default dtype, not a lot of operations can be applied

- Dataframe.apply()
  - Applies something to every element of a dataset
- Pandas.series.value_counts
  - Series – a new data type
  - Vector

A sample of all Pandas Dtypes

Create our DataFrame

In [5]:

```
dft = pd.DataFrame(test_data)
dft
```

Out[5]:

	A	B	C	D	E	F	G
0	0.729845	1	foo	2001-01-02	1.0	False	1
1	0.815146	1	foo	2001-01-02	1.0	False	1
2	0.547269	1	foo	2001-01-02	1.0	False	1

In [14]:

```
dft.dtypes
```

Out[14]:

```
A            float64
B              int64
C             object
D    datetime64[ns]
E            float32
F               bool
G               int8
dtype: object
```

- Sample table with all the different data types to the left

What data types would these be?

- [2,3,4,5,6,7,8.9]
  - Pandas Chooses: FLOAT
  - [2.0,3.0,4.0,5.0,6.0,7.0,8.9]
- [0,1,2,3,'foo']
  - Panda Chooses: OBJECT
  - The most generic – won't lose any data
- [1,3,9,.33, false, '03-20-197', np.arange(22)]
  - Last item will be Array[1….22]

- In pandas – the most common datatype to accommodate all data types
- For more goodness:
  - http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.dtypes.html

Df.Apply() – do all things unto all elements

- Apply()
  - If you give axis = 0, will apply by column
  - If you give for axis =1, will apply by row
- By a single field
  - Df[‘targetfield’].map(my_func)
  - All map functions require RETURN
  - Very similar to a vectorized operation

Sample: Apply Sqrt to all values

```
# Create some more test data
df = pd.DataFrame(np.random.randn(5, 4), columns=['a', 'b', 'c', 'd'])
df
```

	a	b	c	d
0	-1.369438	0.080447	-1.224570	0.422078
1	-0.765415	0.407583	0.385544	0.834827
2	1.758356	0.449190	-1.435722	-0.234399
3	1.172705	-1.596128	-0.821374	0.892523
4	0.026303	-0.749009	-0.940387	0.117611

```
# square root ALL CELLS (NaN == Not a Number)
df.apply(np.sqrt)
```

	a	b	c	d
0	NaN	0.283631	NaN	0.649675
1	NaN	0.638422	0.620922	0.913689
2	1.326030	0.670216	NaN	NaN
3	1.082915	NaN	NaN	0.944734
4	0.162181	NaN	NaN	0.342944

Sample: Apply Mean

```
# Create some more test data
df = pd.DataFrame(np.random.randn(5, 4), columns=['a', 'b', 'c', 'd'])
df
```

	a	b	c	d
0	-1.369438	0.080447	-1.224570	0.422078
1	-0.765415	0.407583	0.385544	0.834827
2	1.758356	0.449190	-1.435722	-0.234399
3	1.172705	-1.596128	-0.821374	0.892523
4	0.026303	-0.749009	-0.940387	0.117611

```
df.apply(np.mean, axis=0)
```

```
a    0.164502
b   -0.281584
c   -0.807302
d    0.406528
dtype: float64
```

Value Counts (similar to group by)

- Value counts for a single field can be either called:
  - Df['fieldname'].value_counts()
  - Pd.value_counts(input_series_data)
- Only useful with repeatable values. For a calendar month data set, value_counts(days) would result in a count of 1 for every day of the month

```
data = np.random.randint(0, 7, size = 50)
data
```

```
array([5, 3, 5, 4, 5, 1, 1, 2, 0, 4, 3, 5, 2
       6, 4, 3, 5, 2, 4, 6, 5, 2, 3, 2, 1, 6
       5, 0, 5, 4])
```

```
# The counts of eac
pd.value_counts(s)
```

```
5    14
2    10
4     9
3     7
0     4
6     3
1     3
dtype: int64
```

Day 8: Pivots : the Long and Wide

Wide vs. Long Format

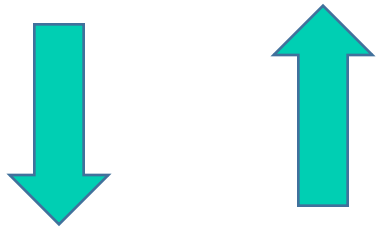
- Two different types of data formats. Used for different things
- "long" = like databases (SQL)
- "wide" = mainly for the scikit learn. Modeling will only be in this format

- Will need to use each throughout the data science process

- **"wide"**

Area	rent	sqft	Person
SOMA	4000	100	A
FIDI	3000	120	B
SOMA	5000	150	C
FIDI	3500	130	D

- **.melt()**



- **.pivot_table()**

- **"long"**

Person	Variable	Value
SOMA	rent	4000
FIDI	rent	3000
SOMA	rent	5000
FIDI	rent	3500
SOMA	sqft	100
FIDI	sqft	120
SOMA	sqft	150
FIDI	sqft	130
SOMA	person	A
FIDI	person	B
SOMA	person	C
FIDI	person	D

Long format, wide format, pivot tables, and melting

- This lesson is all about data transformation in pandas. Data transformation is in essense reorganizing the rows and columns of your dataset to be a different shape and format.
- The benefits to transforming your data are primarily for easier access and manipulation of data, whether it be through easier masking/conditional statements or because you would prefer to operate across columns or down rows.
- Over time you will get a feel for which data formats are better for different tasks. This lesson, however, is focused in large part on the *functional application* of data transformation. How do you do this to a dataset?

Wide Format Data

Area	rent	sqft	Person
SOMA	4000	100	A
FIDI	3000	120	B
SOMA	5000	150	C
FIDI	3500	130	D

- **Wide** format data is the more common format of data for .csv type files. You are already familiar with wide format data: I believe all of the datasets we have been using thus far have been in wide format.
- Wide format data is formatted with criteria:
- There are multiple ID *and* value columns. In other words, there is a column for every "variable" with its own unique values.
- The format has both the conceptual simplicity of a single column of values per variable and a more compact matrix.

- Is not useful for SQL-style operations: it can make it much harder or even impossible to join tables together on a value.
- Can be more useful in pandas when you need to preform operations on variables **across columns**. For example, multiplying columns together.
- It is the most commonly the format that you will put the data in when you are ready to perform modeling (with some exceptions). When we get into modeling next week I will explain why.

Long Format Data

- Now we can load the same data in but in what's commonly referred to as **"long format".**
- Long data is formatted with criteria:
- Potentially multiple "id" (identification) columns.
- **Variable:** value column pairs that match a variable key to a value (in the simple case, a single variable column and a single value column).
- The "variable" column corresponds to the multiple variable columns in your wide format data. Now, instead of a column for each variable, you have a row for each variable:value pair, per id.
- This is a standard format in SQL databases because it is appropriate for joining different tables together by keys.

Person	Variable	Value
SOMA	rent	4000
FIDI	rent	3000
SOMA	rent	5000
FIDI	rent	3500
SOMA	sqft	100
FIDI	sqft	120
SOMA	sqft	150
FIDI	sqft	130
SOMA	person	A
FIDI	person	B
SOMA	person	C
FIDI	person	D

How do you find missing data?

- **My_df.head**()
  - Look at the first rows
- **My_df.describe**()
  - Look at the overall statistics
- **pd.isnull**(My_df)
  - Will provide all the overall blanks

- Should you replace?
  - If #Missing data > #populated values, the missing values outweigh the populated
  - If you use the mean, it may dilute the real data that was collected
- Should you drop?
  - If you drop too many rows, your source dataset may become too small

How do you replace nan in Pandas?

- **wide_nonan = new_wide.dropna()**
  - Drop the entire row that may have nan’s
- **New_wide.fillna()**
  - Fill in the na with replacement values
- **New_wide[new_wide[‘field’]==np.nan] = new value**
  - Manually do the same as before

Pandas : Pivot Tables .pivot_table()

- **new_wide = pd.pivot_table(input_data, columns=['variable'], values = 'value' , index=['subject_id'], aggfunc=select_item_or_nan, fill_value = np.nan)**
  - **input_data –** input data fed in
  - **select_item_or_nan –** aggregating function – custom written
- After transforming the data, may be worth reindexing the data so the row count doesn't repeat:
  - New_wide.**reset_index**

Area	rent	sqft	Person
SOMA	4000	100	A
FIDI	3000	120	B
SOMA	5000	150	C
FIDI	3500	130	D

Pandas : Pivot Tables .pivot_table()

- **columns**: this is the list of columns in the wide format data to transform back to columns in wide format, with each unique value in the long format column becoming a header for the wide format
- **values**: a single column indicating the values to use when pivoting and filling in the new wide format columns
- **index**: columns in the long format data that are index variables – this means that these will be left as single columns, not spread out across columns by unique value such as in the columns parameter
- **aggfunc**: often pivot_table() is used to perform a summary of the data. aggfunc stands for "aggregation function". It is required and defaults to np.mean. You can put your own function in, which I do below.
- **fill_value**: if a cell is missing for the wide format data, the value to fill in

Pandas .melt() -> wide to long format

- subset_long = pd.melt(**my_input**, id_vars=['subject_id','major'])
  - Need to specify the id_value, otherwise the other columns won't make sense. The id value should be the main index of the dataset.
  - Can limit the dataset before the transformation
- subset_long = pd.melt(**my_input**, id_vars=['subject_id','major'], **value_vars['sales','items']**)
  - Can also limit what fields are transformed with the **value_vars**

Pandas .melt() -> wide to long format

- **melt()** is a function that essentially performs the inverse operation of pivot_table on dataframes.
- Melt takes a dataframe as its first argument. Additional arguments typically used in the melt function are:
- **id_vars**: the column or columns that will be id variables. id variables contain datapoints specified by the variable and value columns
- **value_vars**: a list that specifies which columns should be converted into a single value column and variable column.
- **var_name**: the header name of the variable column (default='variable')
- **value_name**: the header name of the value column (default='value')

Pandas .melt() -> wide to long format

Original

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	...	Q30	Q31	Q32	affiliative	selfenhancing	agressive	selfdefeating	ag
					4	5	4	3	4	3	...	4	2	2	4.0	3.5	3.0	2.3	25
					4	4	4	3	4	3	...	4	3	1	3.3	3.5	3.3	2.4	44

Melt

```
hsq_long = pd.melt(hsq_wide, id_vars=['subject_id','age','gender']
                   ,var_name = 'variable', value_name = 'value')
```

Long

	subject_id	age	gender	variable	value
0	0	25	2	usually_dont_laugh	2.0
1	1	44	2	usually_dont_laugh	2.0
2	2	50	1	usually_dont_laugh	3.0
3	3	30	2	usually_dont_laugh	3.0
	4	52	1	usually_dont_laugh	1.0

(39627, 5)

Pandas – changing column types

- **My_dframe.dtypes**
  - Check the current datatypes of the existing data
- **My_dframe.value = My_dframe.value.astype(float)**
  - May result in Nans or errors if data is not checked beforehand

Pandas .pivot () wide <- long format

- subset_summary = pd.pivot_table(**input_data**, columns =['variable'], values = 'value', index=['major'], **aggfunc = [np.mean, np.median]**)

Pandas .pivot () wide <- long format

	mean			median		
variable	anxious	bookish	calm	anxious	bookish	calm
major						
None yet	7.0	3.0	3.0	7.0	3.0	3.0
+ACI-+ACIAIg-hotel and restaurant management+ACIAIg-+ACI-	2.0	2.0	7.0	2.0	2.0	7.0
Aerospace Engineer	2.0	2.0	7.0	2.0	2.0	7.0
Aerospace Engineering	3.0	4.0	3.0	3.0	4.0	3.0
Agricultural Economics	2.0	2.0	6.0	2.0	2.0	6.0

Navigating / indexing a Panda Pivot - Hardway

	mean			median		
variable	anxious	bookish	calm	anxious	bookish	calm
major						
None yet	7.0	3.0	3.0	7.0	3.0	3.0
+ACI-+ACIAIg-hotel and restaurant management+ACIAIg-+ACI-	2.0	2.0	7.0	2.0	2.0	7.0
Aerospace Engineer	2.0	2.0	7.0	2.0	2.0	7.0
Aerospace Engineering	3.0	4.0	3.0	3.0	4.0	3.0
Agricultural Economics	2.0	2.0	6.0	2.0	2.0	6.0

- Pivot_summary.loc[:,'mean'].loc['Art,'anxious']
  - Under mean – pull "Art" and "Anxious

Navigating / indexing a Panda Pivot - Easier

- Pivot_summary.to_records()
  - Gives in a flat format

	mean			median		
variable	anxious	bookish	calm	anxious	bookish	calm
major						
None yet	7.0	3.0	3.0	7.0	3.0	3.0
+ACI-+ACIAIg-hotel and restaurant management+ACIAIg-+ACI-	2.0	2.0	7.0	2.0	2.0	7.0
Aerospace Engineer	2.0	2.0	7.0	2.0	2.0	7.0
Aerospace Engineering	3.0	4.0	3.0	3.0	4.0	3.0
Agricultural Economics	2.0	2.0	6.0	2.0	2.0	6.0

```
In [65]: subset_summary.to_records()

Out[65]: rec.array([(' None yet', 7.0, 3.0, 3.0, 7.0, 3.0, 3.0),
          ('+ACI-+ACIAIg-hotel and restaurant management+ACIAIg-+ACI-', 2.0, 2.0, 7.0, 2.0, 2.0, 7.0),
          ('Aerospace Engineer ', 2.0, 2.0, 7.0, 2.0, 2.0, 7.0),
          ('Aerospace Engineering', 3.0, 4.0, 3.0, 3.0, 4.0, 3.0),
          ('Agricultural Economics', 2.0, 2.0, 6.0, 2.0, 2.0, 6.0),
          ('Anthropology', 5.333333333333333, 3.666666666666665, 4.333333333333333, 5.0, 4.0, 4.0),
          ('Anthropology ', 5.0, 4.0, 3.0, 5.0, 4.0, 3.0),
          ('Architecture', 3.0, 4.0, 5.666666666666667, 4.0, 4.0, 6.0),
```

To_records → Pandas

- Pd.Dataframe (Pivot_summary.to_records())
  - Taking the flat format and making a similar dataframe from it
  - Concatenates the headings together

	mean			median		
variable	anxious	bookish	calm	anxious	bookish	calm
major						
None yet	7.0	3.0	3.0	7.0	3.0	3.0
+ACI-+ACIAIg-hotel and restaurant management+ACIAIg-+ACI-	2.0	2.0	7.0	2.0	2.0	7.0
Aerospace Engineer						
Aerospace Engineering						
Agricultural Economics						

	major	('mean', 'anxious')	('mean', 'bookish')	('mean', 'calm')	('median', 'anxious')	('median', 'bookish')
0	None yet	7.0	3.0	3.0	7.0	3.0
1	+ACI-+ACIAIg-hotel and restaurant management+A...	2.0	2.0	7.0	2.0	2.0
2	Aerospace Engineer	2.0	2.0	7.0	2.0	2.0
3	Aerospace Engineering	3.0	4.0	3.0	3.0	4.0
4	Agricultural Economics	2.0	2.0	6.0	2.0	2.0

Convert the weird tuple headings into 1

- **new_cols = ['major']+['_'.join(eval(col)) for col in subset_summary_flat.columns[1:]]**
  - For all columns, join the subsets with '_' ('mean','anxious') => mean_anxious
- **subset_summary_flat.columns = new_cols**
  - Assign the new column names

	major	('mean', 'anxious')	('mean', 'bookish')	('mean', 'calm')	('a

	major	mean_anxious	mean_bookish	mean_calm	n

Example: Beers

- Lets's say you have some beer preferences:

	class	name	price	rating
0	crap	coors	1.5	3.251746
1	crap	bud	1.7	7.053655
2	crap	natural light	1.2	5.711367
3	crap	keystone ice	1.2	7.420232
4	mid	sierra nevada	2.0	4.666245
5	mid	sam adams	1.9	3.780572
6	mid	rolling rock	2.1	4.930330
7	notabeer	odouls	3.0	6.075566
8	pretentious	pbr	0.5	6.820391
9	pretentious	stella	3.5	5.617917
10	pretentious	chimay	10.0	8.583392
11	pretentious	magnolia	15.0	3.828258
12	pretentious	21amendment	2.0	8.168663

Example: Beers melt→ long

- beers_long = pd.melt(beers, id_vars = ['name'])
- beers_long.sort_values('name',axis=0)

	name	variable	value
38	21amendment	rating	8.16866
25	21amendment	price	2
12	21amendment	class	pretentious
1	bud	class	crap
27	bud	rating	7.05365
14	bud	price	1.7
36	chimay	rating	8.58339
23	chimay	price	10
10	chimay	class	pretentious
26	coors	rating	3.25175
13	coors	price	1.5
0	coors	class	crap
16	keystone ice	price	1.2

Example: Beers long→ pivot

- beer_class_summary =
  pd.pivot_table(beers
  , index='class'
  , values =['rating','price']
  , aggfunc = [len, np.mean,np.std]
)

- beer_class_summary

	len		mean		std	
	price	rating	price	rating	price	rating
class						
crap	4.0	4.0	1.4	5.859250	0.244949	1.887169
mid	3.0	3.0	2.0	4.459049	0.100000	0.602232
notabeer	1.0	1.0	3.0	6.075566	NaN	NaN
pretentious	5.0	5.0	6.2	6.603724	6.109419	1.942298

Day 9: Groupby – meetups for data

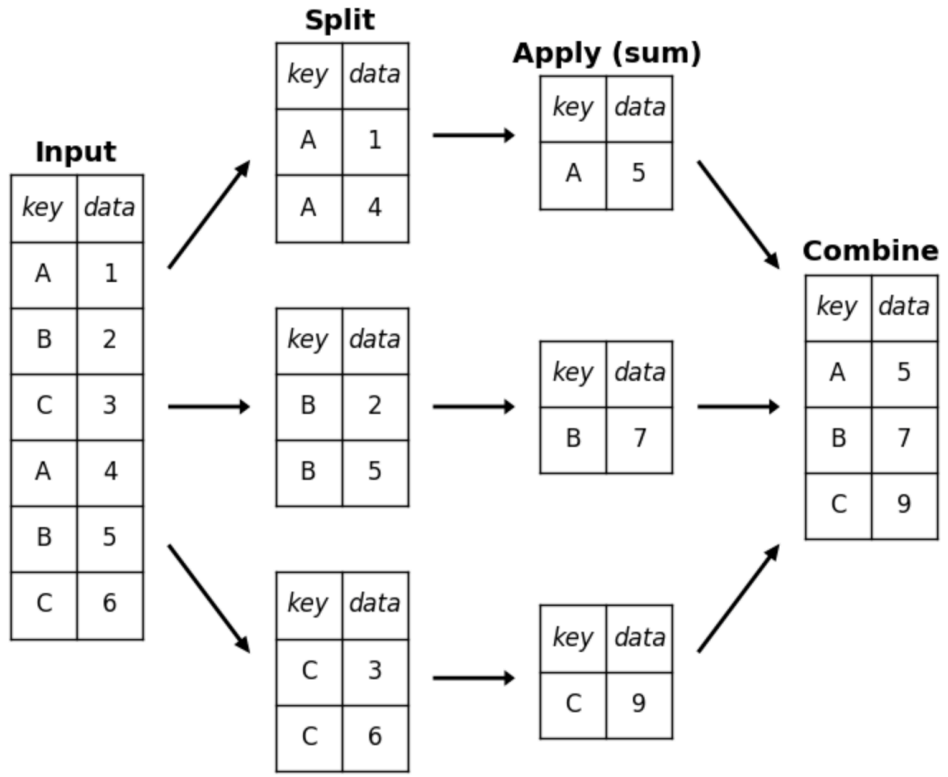
Generally: Multi-Dimensional Data Analysis - Powerful - Underused

- Describe segments of your data based on unique values
- Helps understand hidden(latent) charactaristics of your data
- Apply aggregate functions to subsets, based on variable aspects of our datasets
- Summary statistics of subsets
- Discover patterns that exist in subsets

How does it work? Hadly-Wickham

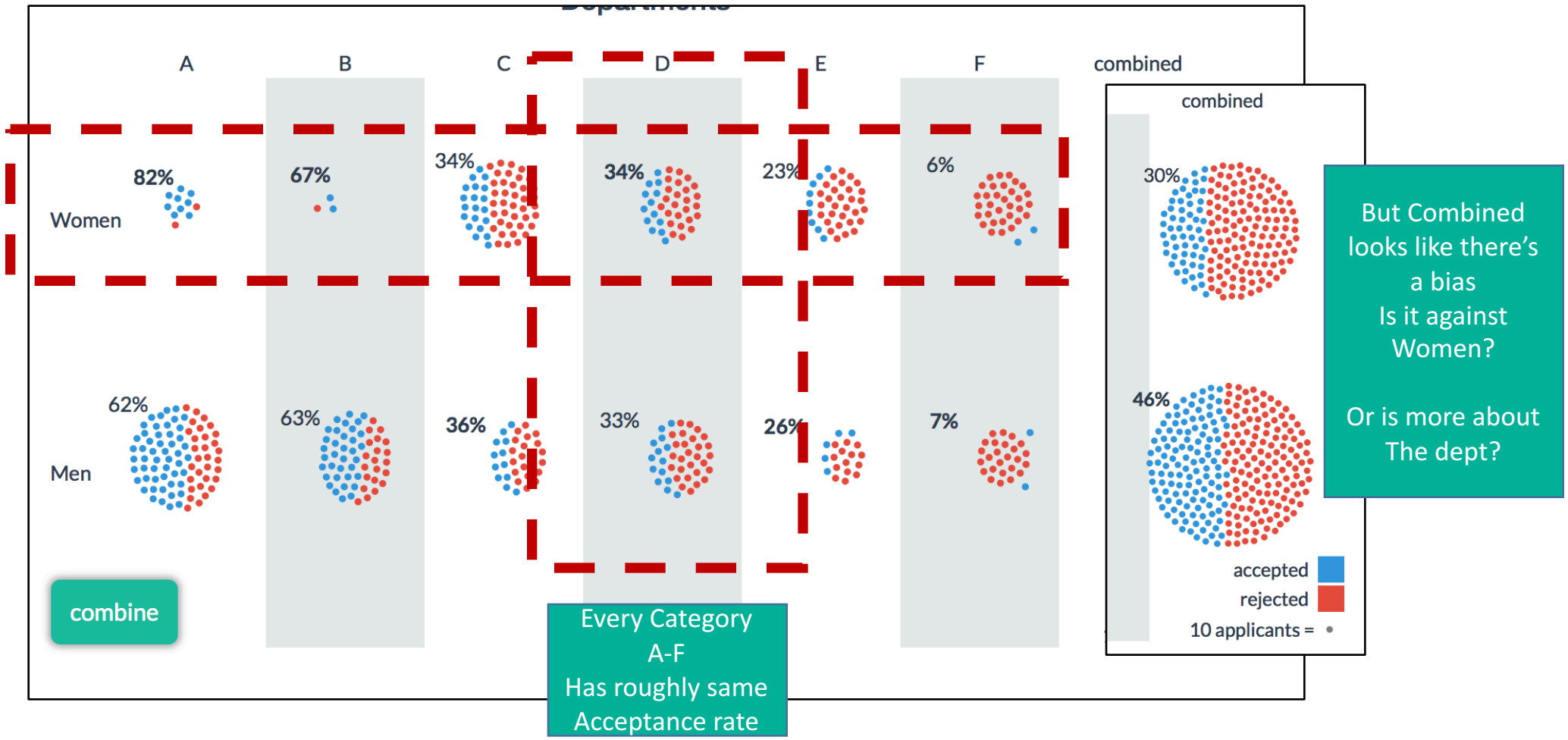
How does it work?

- Functionality draws from the Split-Apply-Combine Strategy Hadley Wickham



The level of aggregation tell different stories:

But the distribution of applications to each dept is very difference



But Combined looks like there's a bias
Is it against Women?

Or is more about The dept?

Introducing Group by

- **Groupby([var …..]) .[[what i want to know]]. What metric i want ()**

- Groupby returns another panda datatype:
  - Pandas.core.groupby.dataframeGroupby

Samples

- **Groupby([‘sex’,’age’]) .[[ ]]. mean()**

<class 'pandas.core.groupby.DataFrameGroupBy'>

Sex	Age	PassengerId	Survived	Pclass	SibSp	Parch	Fare
female	0.75	557.500000	1.000000	3.000000	2.000000	1.000000	19.258300
	1.00	277.500000	1.000000	3.000000	0.500000	1.500000	13.437500
	2.00	379.666667	0.333333	2.500000	1.500000	1.500000	43.245833
	3.00	209.500000	0.500000	2.500000	2.000000	1.500000	31.327100
	4.00	451.600000	1.000000	2.600000	0.800000	1.200000	22.828340
	5.00	380.000000	1.000000	2.750000	1.750000	1.250000	22.717700
	6.00	767.500000	0.500000	2.500000	2.000000	1.500000	32.137500
	7.00	536.000000	1.000000	2.000000	0.000000	2.000000	26.250000
	8.00	131.500000	0.500000	2.500000	1.500000	1.500000	23.662500
	9.00	544.500000	0.000000	3.000000	2.500000	1.750000	27.198950
	10.00	420.000000	0.000000	3.000000	0.000000	2.000000	24.150000
	11.00	543.000000	0.000000	3.000000	4.000000	2.000000	31.275000

- **Groupby([‘sex’]) .[[‘age’]]. mean()**
  - ‘sex’ = [‘sex’] for single values

[144 rows x 6 columns]

Sex	Age
female	27.745174
male	30.726645

A note on Aggregate functions

- **A quick note about two popular aggregates that seem the same, but are not!**

```
        PassengerId  Survived  Name  Sex  Age  SibSp  Parch  Fare  Embarked
Pclass
1               184       184   184  184  184    184    184   184       184
2               173       173   173  173  173    173    173   173       173
3               355       355   355  355  355    355    355   355       355

===

Pclass
1    184
2    173
3    355
dtype: int64
```

.count() vs .size()

.count() gives us counts of values per fields

.size() gives us a count of the number of series (some series may have null values, but as long as a value is in there, it will be counted

One Answer – but Two Methods

- **print titanic.groupby(['Sex','Survived'])['Age'].mean()**
  - Group by first (implied that these fields will be added to the output
  - then pull the data fields to be manipulated

- **print titanic[['Age','Sex','Survived']].groupby(['Sex','Survived']).mean()**
  - Pre-Filter all fields you might need
  - If sex, survived in the first block. Then group by will only see Age
  - Then group by the target ones
  - Will average the remainder

Sample of what reset_index( ) does

Restores a proper index to a integer index. See picture.

The reason why is this table may be used for additional operations and the index will assist with joining and filtering operations

```
print titanic.groupby(['Sex','Survived'])[['Age']].mean()
```

```
                Age
Sex    Survived
female 0        25.046875
       1        28.630769
male   0        31.618056
       1        27.276022
```

```
print titanic.groupby(['Sex','Survived'])[['Age']].mean().reset_index()
```

```
      Sex  Survived        Age
0  female         0  25.046875
1  female         1  28.630769
2    male         0  31.618056
3    male         1  27.276022
```

Get group: Sample Data Part 1 of 3

	order_id	quantity	item_name	choice_description	item_price	sub_order_id	clean_price	category	order_total_price	adjusted_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39	0	2.39	chips	11.56	2.39
1	1	1	Izze	[Clementine]	$3.39	1	3.39	drink	11.56	3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39	2	3.39	drink	11.56	3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39	3	2.39	chips	11.56	2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans	$16.98	0	16.98	burrito	16.98	33.96

Get group: Sample Data Part 2 of 3

- First group by fields. IN this example, grouped by Category and Item name.
- But what if you only wanted ”carnitas bowls”?

```
chip6.groupby(['category','item_name']).size()
```

category	item_name	
burrito	Barbacoa Bowl	66
	Barbacoa Burrito	91
	Barbacoa Salad Bowl	10
	Bowl	2
	Burrito	6
	Carnitas Bowl	68
	Carnitas Burrito	59
	Carnitas Salad Bowl	6
	Chicken Bowl	726
	Chicken Burrito	553
	Chicken Salad Bowl	110
	Steak Bowl	211
	Steak Burrito	368
	Steak Salad Bowl	29
	Veggie Bowl	85
	Veggie Burrito	95
	Veggie Salad Bowl	18
chips	Chips	211
	Chips and Fresh Tomato Salsa	110
	Chips and Guacamole	479
	Chips and Mild Fresh Tomato Salsa	1
	Chips and Roasted Chili Corn Salsa	22
	Chips and Roasted Chili-Corn Salsa	18
	Chips and Tomatillo Green Chili Salsa	43
	Chips and Tomatillo Red Chili Salsa	48
	Chips and Tomatillo-Green Chili Salsa	31
	Chips and Tomatillo-Red Chili Salsa	20

Get group: Sample Data 3 of 3

- In a grouped data frame, you can select a particular “row” of the summarized table

```
chip6.groupby(['category','item_name']).get_group(('burrito','Carnitas Bowl'))
```

	order_id	quantity	item_name	choice_description	item_price	sub_order_id	clean_price	category	order_total_price	adjusted_price
33	17	1	Carnitas Bowl	Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$8.99	0	8.99	burrito	10.08	8.99
97	43	1	Carnitas Bowl	[Fresh Tomato Salsa, [Fajita Vegetables, Rice,...	$11.75	0	11.75	burrito	20.50	11.75
145	65	1	Carnitas Bowl	[Roasted Chili Corn Salsa, Rice, Fajita Veget...	$9.25	1	9.25	burrito	25.45	9.25
163	74	1	Carnitas	[Roasted Chili Corn Salsa	$11.48	0	11.48	burrito	22.36	11.48

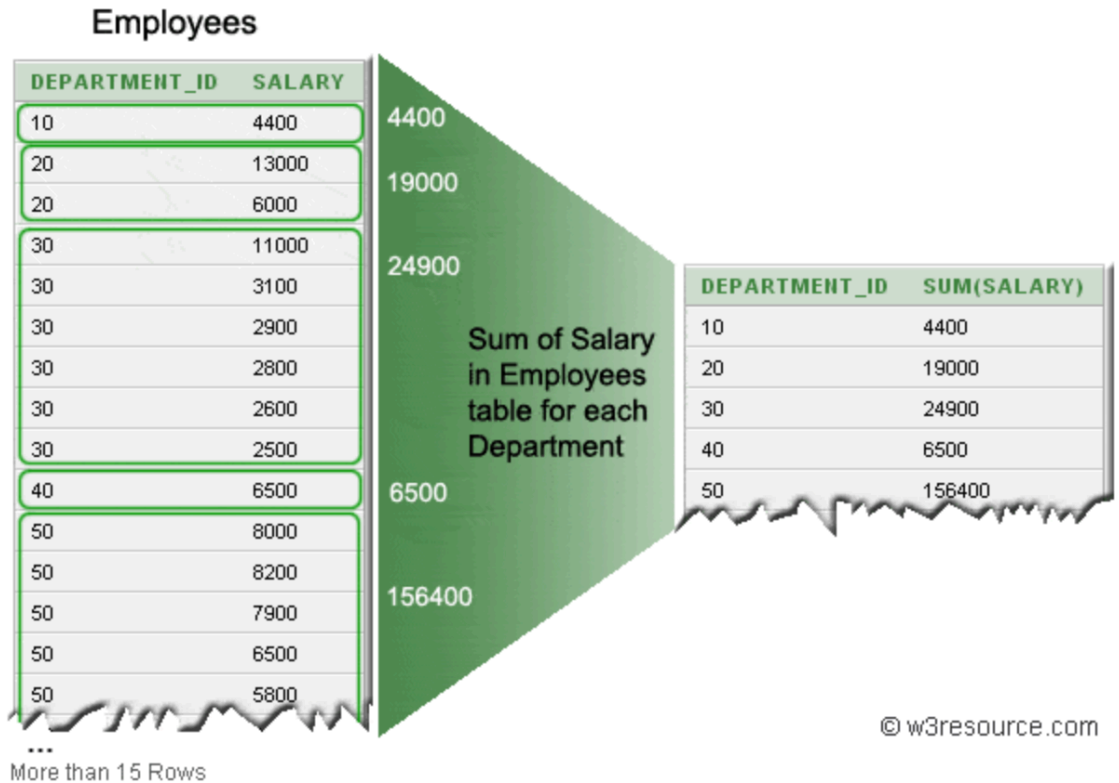
.add_prefix()

```
print titanic.groupby(['Sex','Survived'])[['Age']].apply(np.mean).reset_index()

print titanic.groupby(['Sex','Survived'])[['Age']].apply(np.mean).add_prefix('hi_').reset_index()
```

```
      Sex  Survived        Age
0  female         0  25.046875
1  female         1  28.630769
2    male         0  31.618056
3    male         1  27.276022
      Sex  Survived     hi_Age
0  female         0  25.046875
1  female         1  28.630769
2    male         0  31.618056
3    male         1  27.276022
```

Subsets

Let's dive into the dataset, exploring/creating subsets



Lambda – inline functions

- Regular function
  - Def addone(x)
    - Return x +1
  - Addone(x) for x in [1,2,3]
- Lambda version:
  - Lambda x: x+1
  - [Lambda x: x+1 for x in [1,2,3]]

Graph Cheat Sheet

- *List of options for graph*
- ‘bar’ or ‘barh’ for bar plots
- ‘hist’ for histogram
- ‘box’ for boxplot
- ‘kde’ or 'density' for density plots
- ‘area’ for area plots
- ‘scatter’ for scatter plots
- ‘hexbin’ for hexagonal bin plots
- ‘pie’ for pie plots

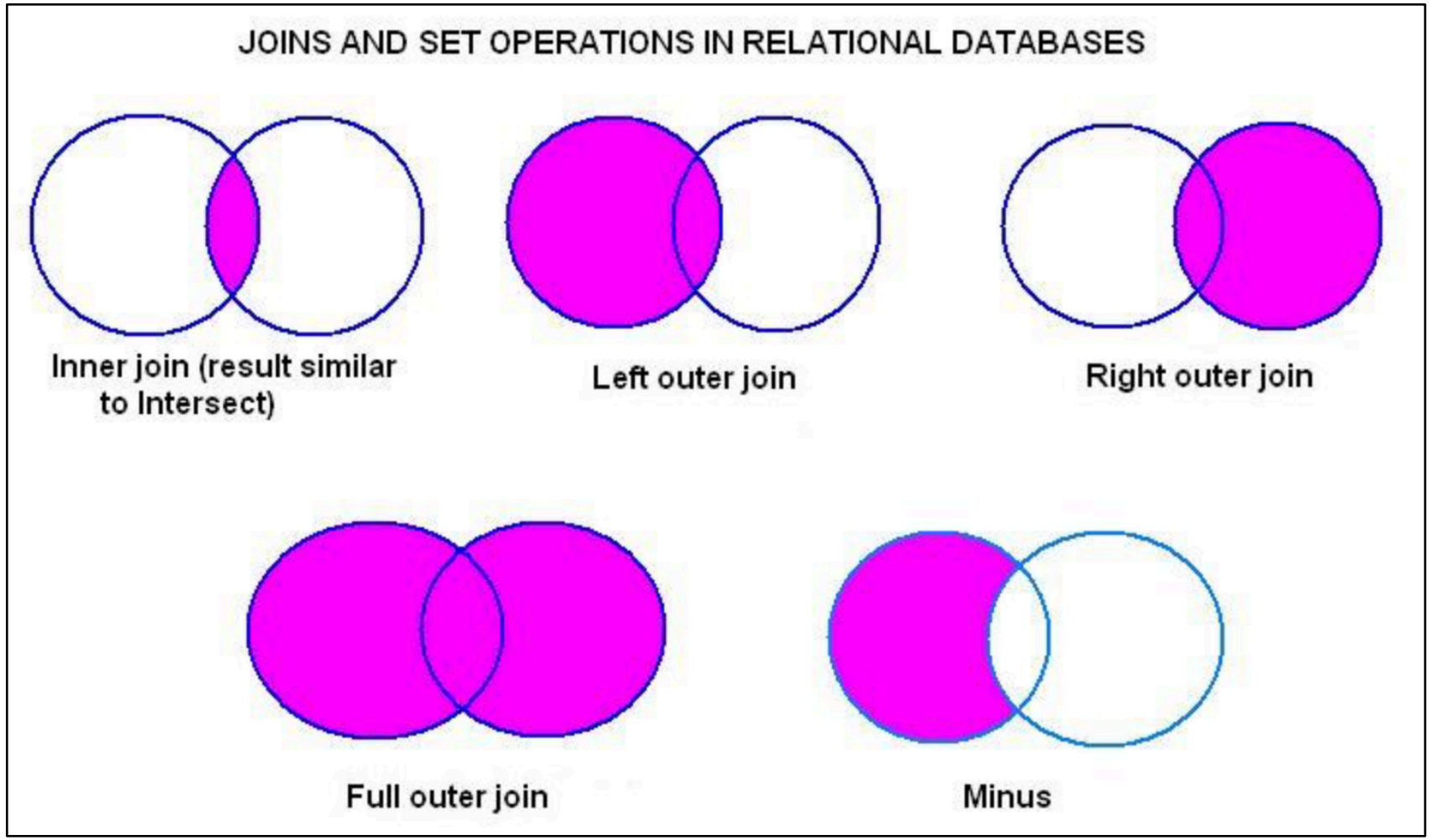
One other note for APPLY, if you need AXIS

- If there's no Group by – put in apply
  - Var = var.apply(function , **axis = 1**)
- If there's a group by put the axis in the group by
  - Var = var.groupby('somefield' , **axis = 1**).apply(function)
- i_df_clean2 = i_df_clean2.groupby('YEAR',axis=1).apply(map_Q1)

Day 9: Merge and Joins

Joinz

- Inner Join
- Outer join
- Left join
- Right Join



Concat Sample: two datasets

```
# df1

df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])
df1
```

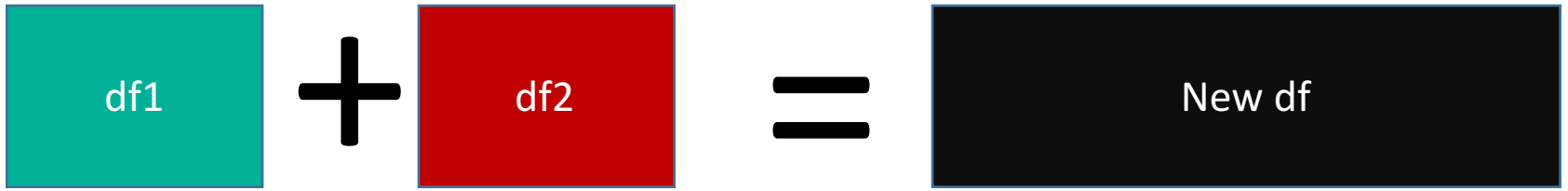
	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
# df2

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                   index=[4, 5, 6, 7])
df2
```

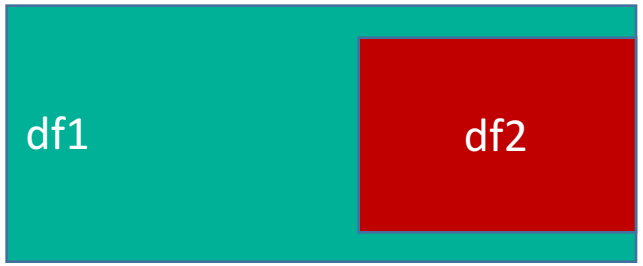
	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

Concat – Options

- **pd.concat([df1, df2])**
  - Will create a new dataframe that will be returned

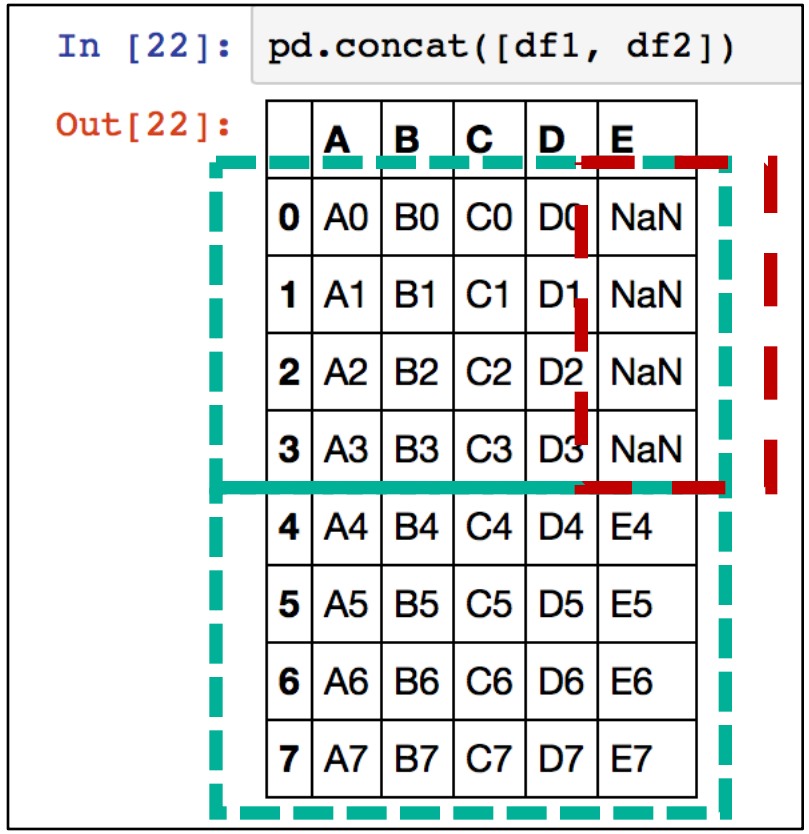


- **Df1.append(df2)**
  - Will not create a new variable, but will “extend” the first datafram to capture the 2nd one.



Concat Stacking line the tower

- **Concat** two datasets ontop of each other
- If the two targets are two different sizes, will fill in null values for any missing cells, noted in red



	A	B	C	D	E
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3	NaN
4	A4	B4	C4	D4	E4
5	A5	B5	C5	D5	E5
6	A6	B6	C6	D6	E6
7	A7	B7	C7	D7	E7

Merge and Join Sample

First Dataframe

	subject_id	first_name	last_name
0	1	Alex	Anderson
1	2	Amy	Ackerman
2	3	Allen	Ali
3	4	Alice	Aoni
4	5	Ayoung	Atiches

Second Dataframe

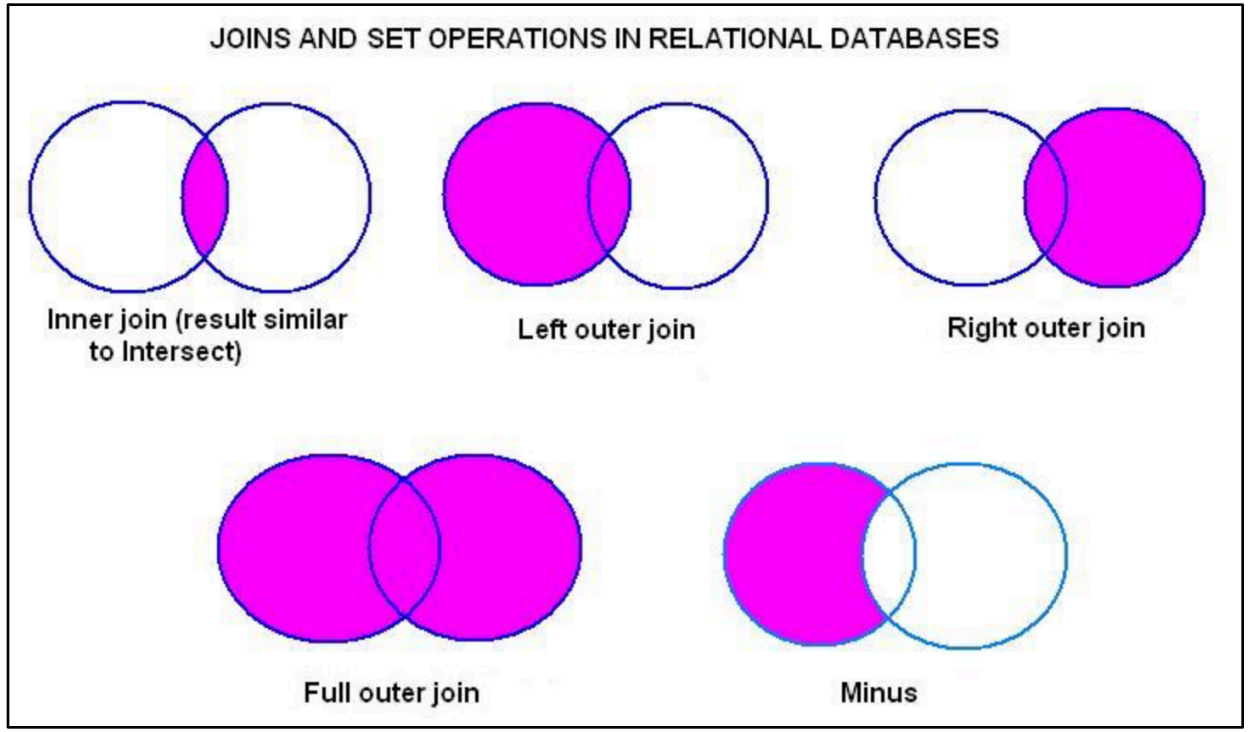
	subject_id	first_name	last_name
0	4	Billy	Bonder
1	5	Brian	Black
2	6	Bran	Balwner
3	7	Bryce	Brice
4	8	Betty	Btisan

Third Dataframe

	subject_id	test_id
0	1	51
1	2	15
2	3	15
3	4	61
4	5	16
5	7	14
6	8	15
7	9	1
8	10	61
9	11	16

Merge Syntax

- **pd.merge(df_a, df_b, on='subject_id', how='left')**
  - First dataset
  - Second dataset
  - Join key or field
  - How = type of join.
    - LEFT , RIGHT , OUTER , INNER
- **More pictures!**
- http://pandas.pydata.org/pandas-docs/stable/merging.html



Merge: (Left join), note the nulls (non–match)

```
pd.merge(df_a, df_b, on='subject_id', how='left')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black

Merge: (Right join), note the nulls (non–match)

```
In [29]: pd.merge(df_a, df_b, on='subject_id', how='right')
```

Out[29]:

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black
2	6	NaN	NaN	Bran	Balwner
3	7	NaN	NaN	Bryce	Brice
4	8	NaN	NaN	Betty	Btisan

Merge: Outer join

```
pd.merge(df_a, df_b, on='subject_id', how='outer')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	1	Alex	Anderson	NaN	NaN
1	2	Amy	Ackerman	NaN	NaN
2	3	Allen	Ali	NaN	NaN
3	4	Alice	Aoni	Billy	Bonder
4	5	Ayoung	Atiches	Brian	Black
5	6	NaN	NaN	Bran	Balwner
6	7	NaN	NaN	Bryce	Brice
7	8	NaN	NaN	Betty	Btisan

Merge: Inner join

```
pd.merge(df_a, df_b, on='subject_id', how='inner')
```

	subject_id	first_name_x	last_name_x	first_name_y	last_name_y
0	4	Alice	Aoni	Billy	Bonder
1	5	Ayoung	Atiches	Brian	Black

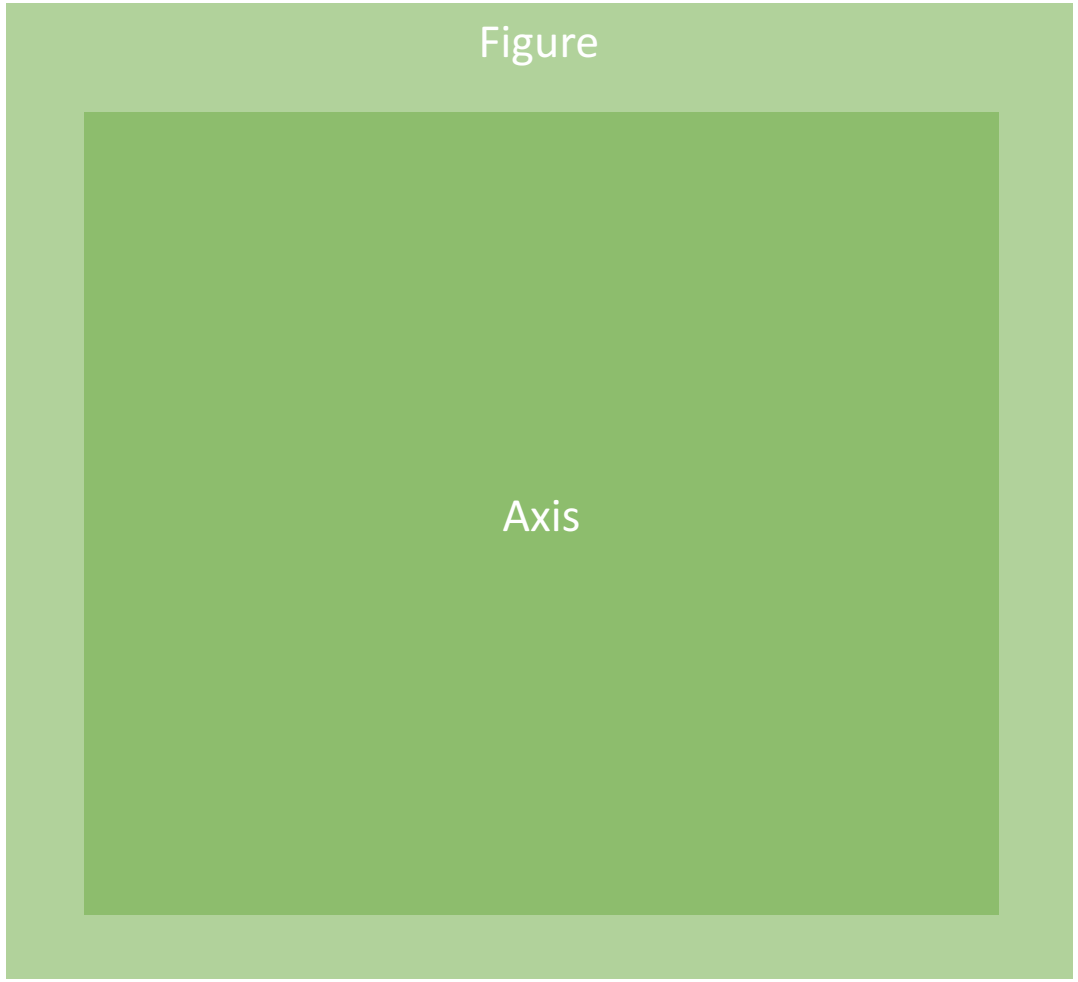
Day 10: Graphing Side Quest

Graph Cheat Sheet (revisited)

- *List of options for graph (as plotting functions of Matplot lib, e.g. ax.scatter(…) , or ax.hist(...) (not sure about seaborn)*
  - ‘bar’ or ‘barh’ for bar plots
  - ‘hist’ for histogram
  - ‘box’ for boxplot
  - ‘kde’ or 'density' for density plots
  - ‘area’ for area plots
  - ‘scatter’ for scatter plots
  - ‘hexbin’ for hexagonal bin plots
  - ‘pie’ for pie plots

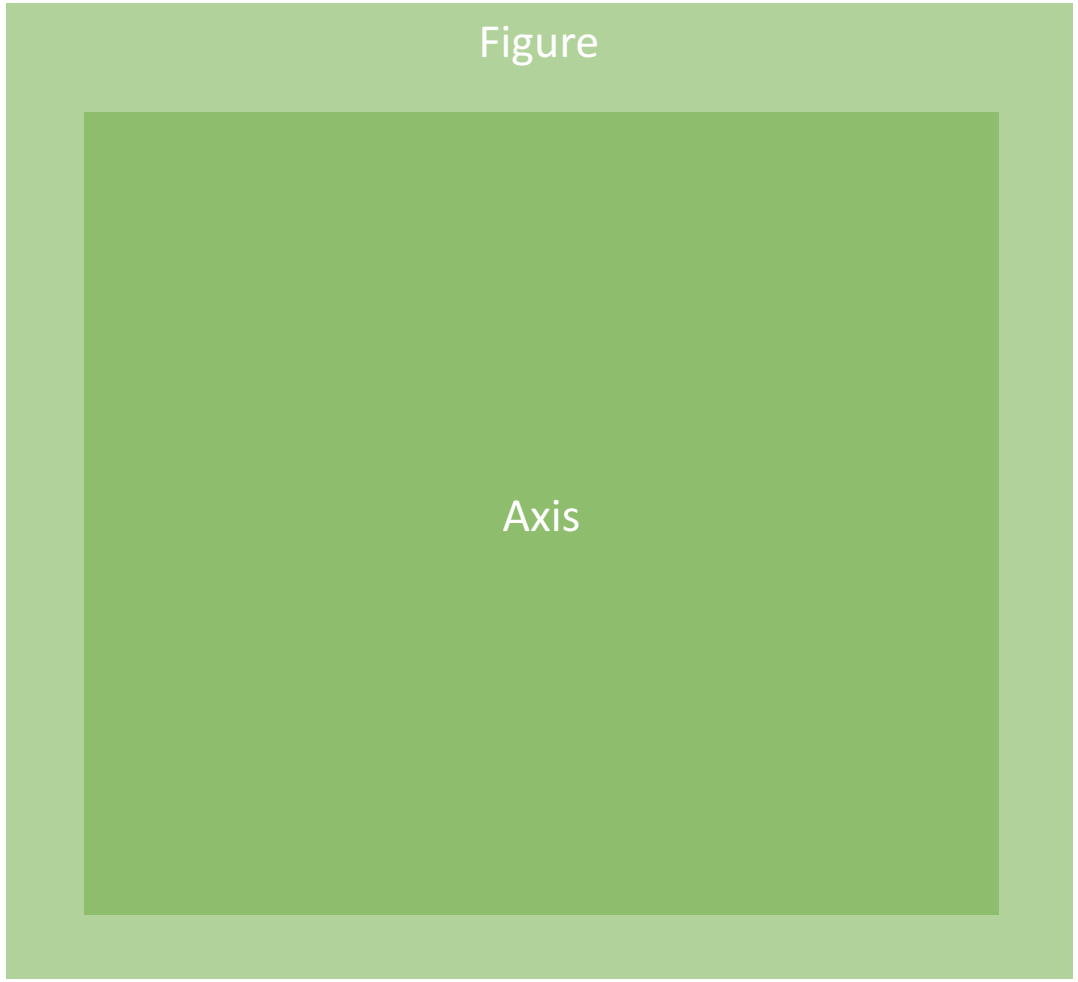
Matplotlib

- fig_var = plt.figure(figsize = (5,3))
  - Creates a figure object with size 5,3 like a poster board
- Ax_var = fig.gca()
  - Creating an axis object, with X / y Positioning and a size
- Use the Ax_var object to plot as follows:
  - Ax.hist()
  - Ax.plot()



Seaborn Plotting Sample

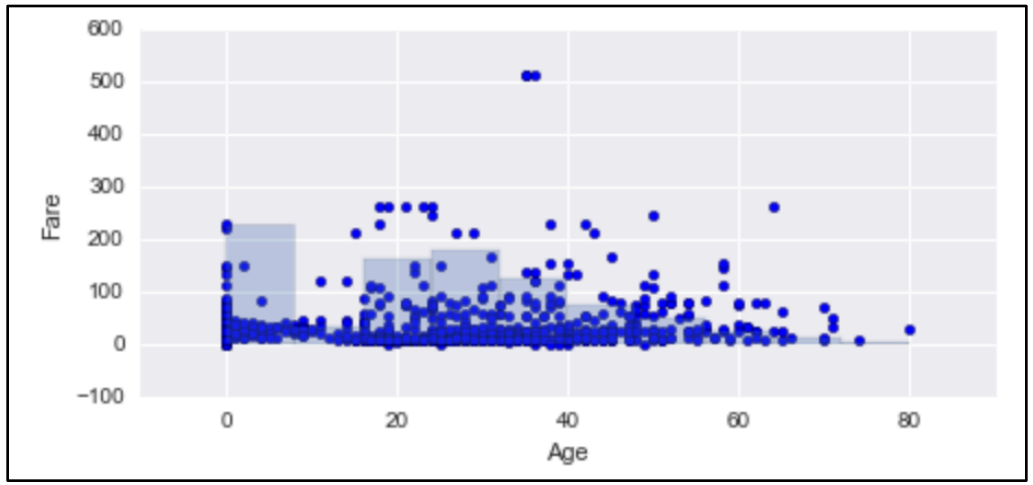
- Import seaborn as sns
- Sns.set(rc={”figure.figsize”:(5,3)}}
- Sns.set_style(“white”)
- If not set above, seaborn will make its own figures and axis
- sns.distplot(td.loc[:,'Age'], kde=False, bins = 10)

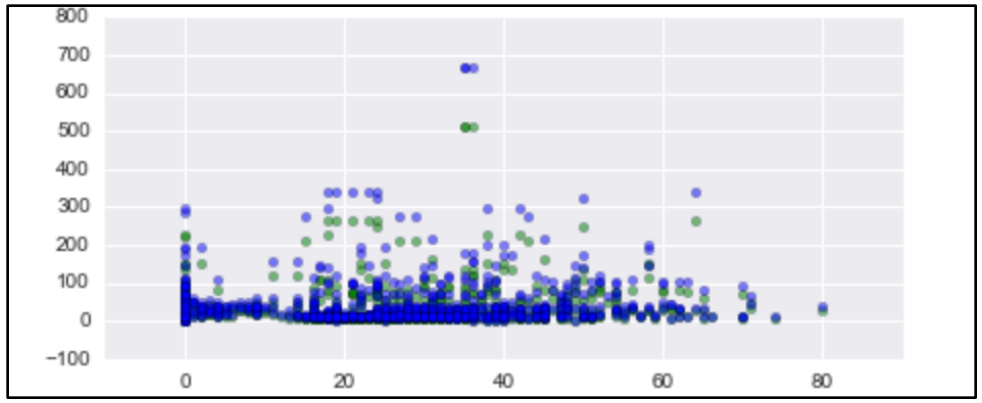


Overlay Multiple Plots

- Simply call two plot functions before show() after a figure command to **overlay** the two different data sets. Use **alpha** to change the transparency
- Example 1:
  - Ax.scatter(…)
  - Ax.hist(…)
- Example 2:
  - Ax.scatter(...)
  - Ax.scatter(...)
- Then plt.show( )

- Samples

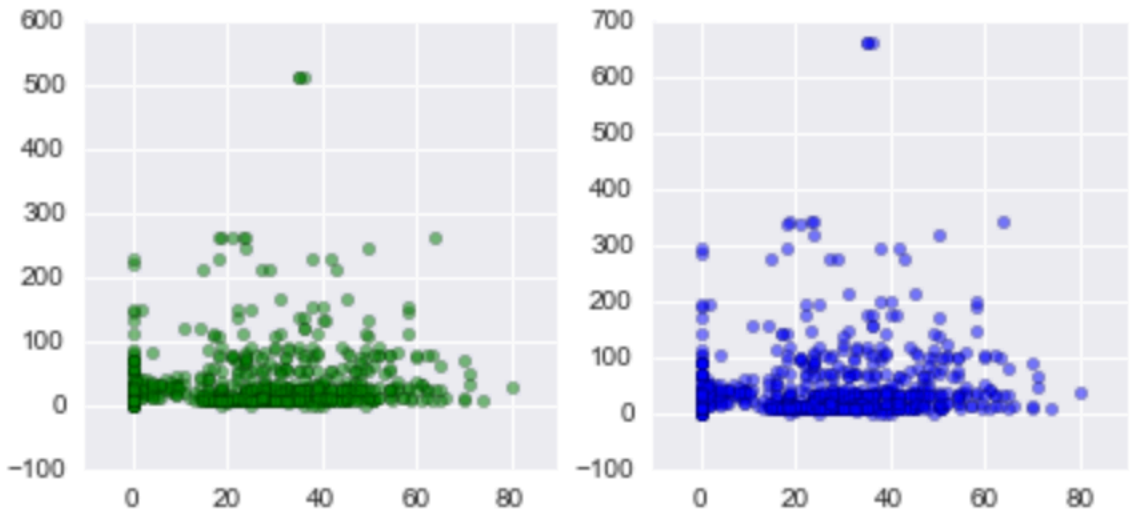




Multiple Side by Side Plots

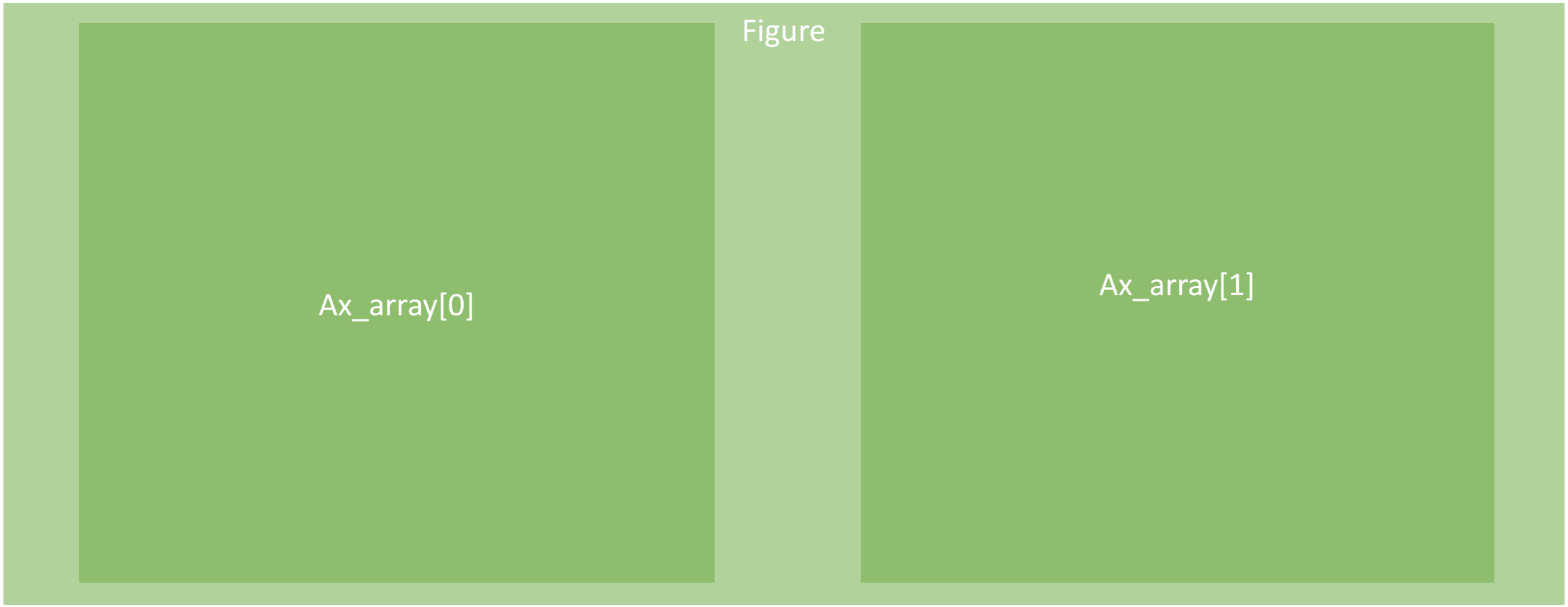
- Fig, Ax_array = plt.figure(1,2,figsize=(5,5))
  - Now we have an array that we can assign different plots to different parts of the array
- ax_array[0] = scatter ()
- Ax_array[1] = scatter ()

```
fig, ax_array = plt.subplots(1,2,figsize=(7,3))
ax_array[0].scatter(td['Age'],td['Fare'], c= 'Green',
ax_array[1].scatter(td['Age'],td['something'], c= 'Blu
plt.show()
```



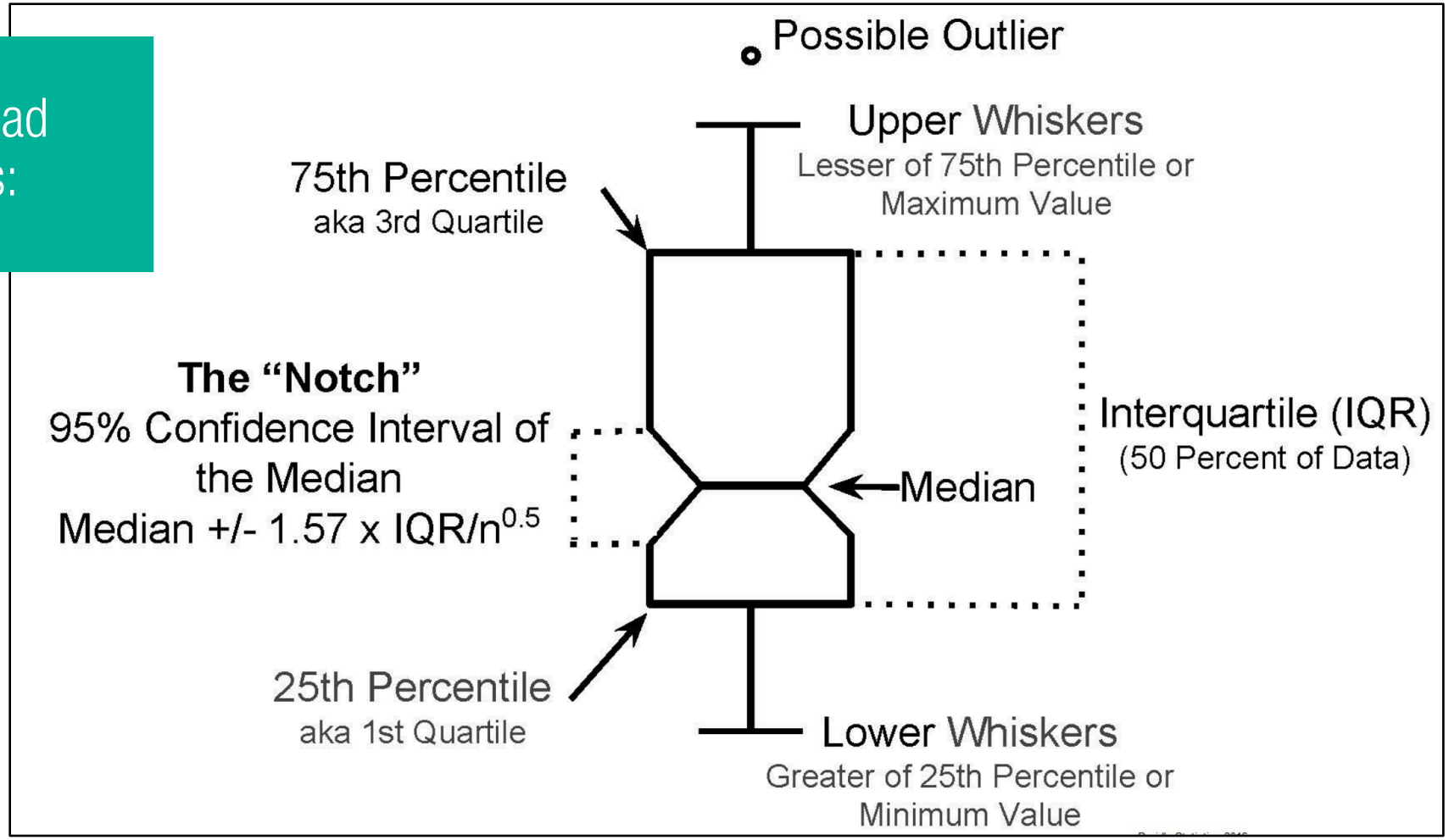
Ax_array[ ] object

- Fig, Ax_array = plt.figure(1,2,figsize=(5,5))



Boxplots

- How to read boxplots:



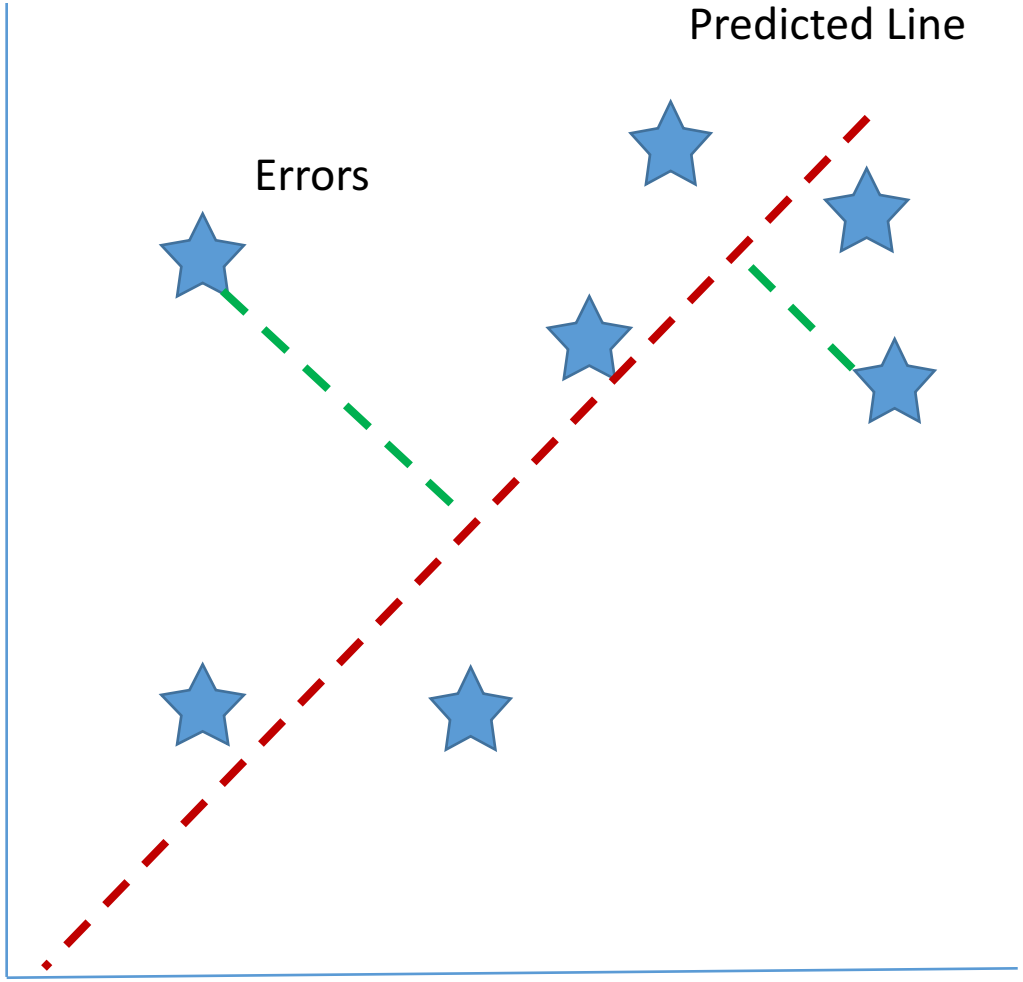
Day 11: Intro to regression

Types of

	Continuous	Categorical
Supervised	**“Regression”** Prediction of a value, such as revenue or quantity.	**“classification”** Prediction of a Y/N question, or % chance falling into a fixed grouping
Unsupervised	**“Dimensionality reduction”** (better set of variables than before… a 100Q get reduced down to 5 key metrics)	**Clustering** (no outcome, but you think there are sub-categories in an ambiguous group.)

Linear Regression

- Linear regression for two variables X, Y,
- There is some constant relationship between the two:
  - y = x (constant) + offset
- Linear regression predicts a relationship(red line) between the two metrics. This line is iterated to by the error.
- The method approximates the error by error^2



Linear Regression - Continued

- BLOOP
- If assumptions are met, no outliers. Best linear unbiased predictor. It is the “best”
- Normality of errors.
  - If all the errors were plotted, and all the errors are in a standard distribution
  - Then it cannot be outperformed

- Popular
- Simple structure (see equation below)

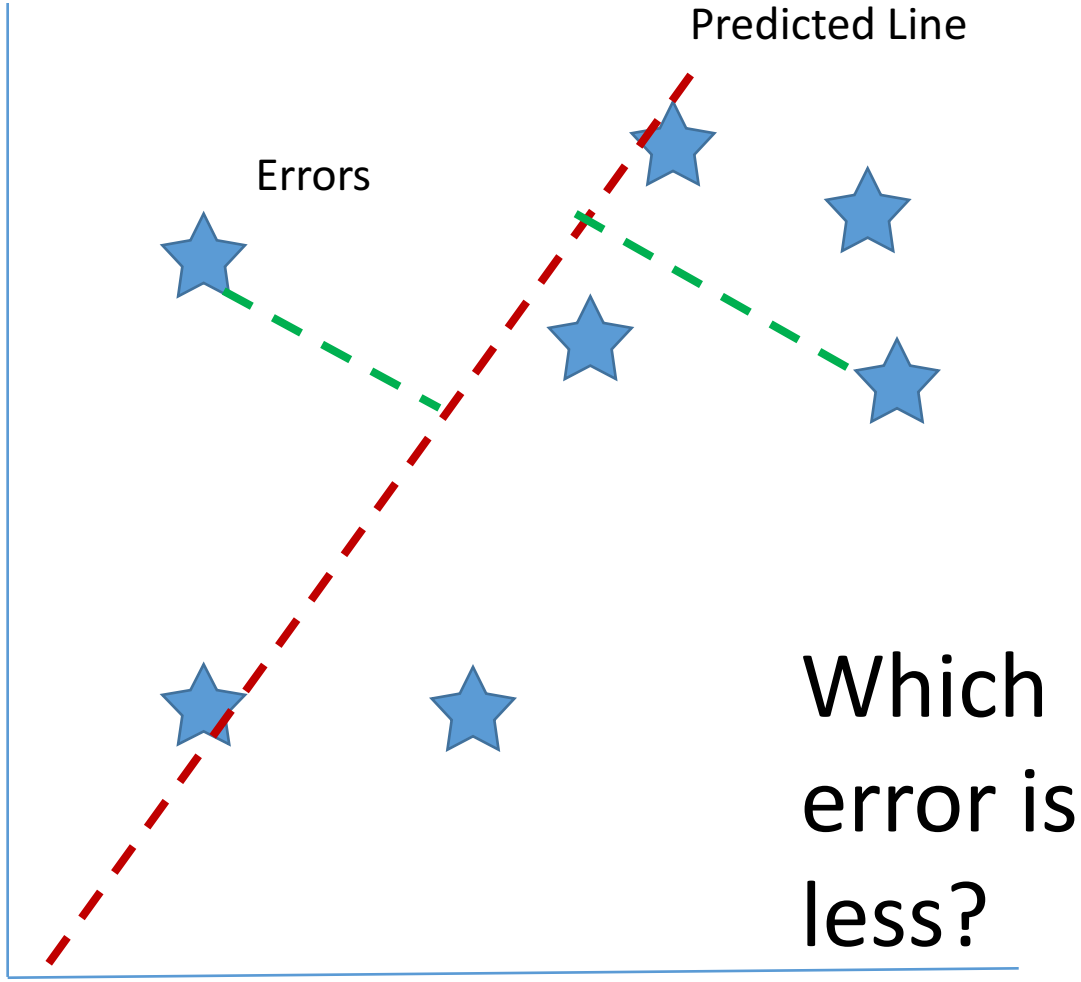
$$Y\sim b_0 + b_1 x_1$$

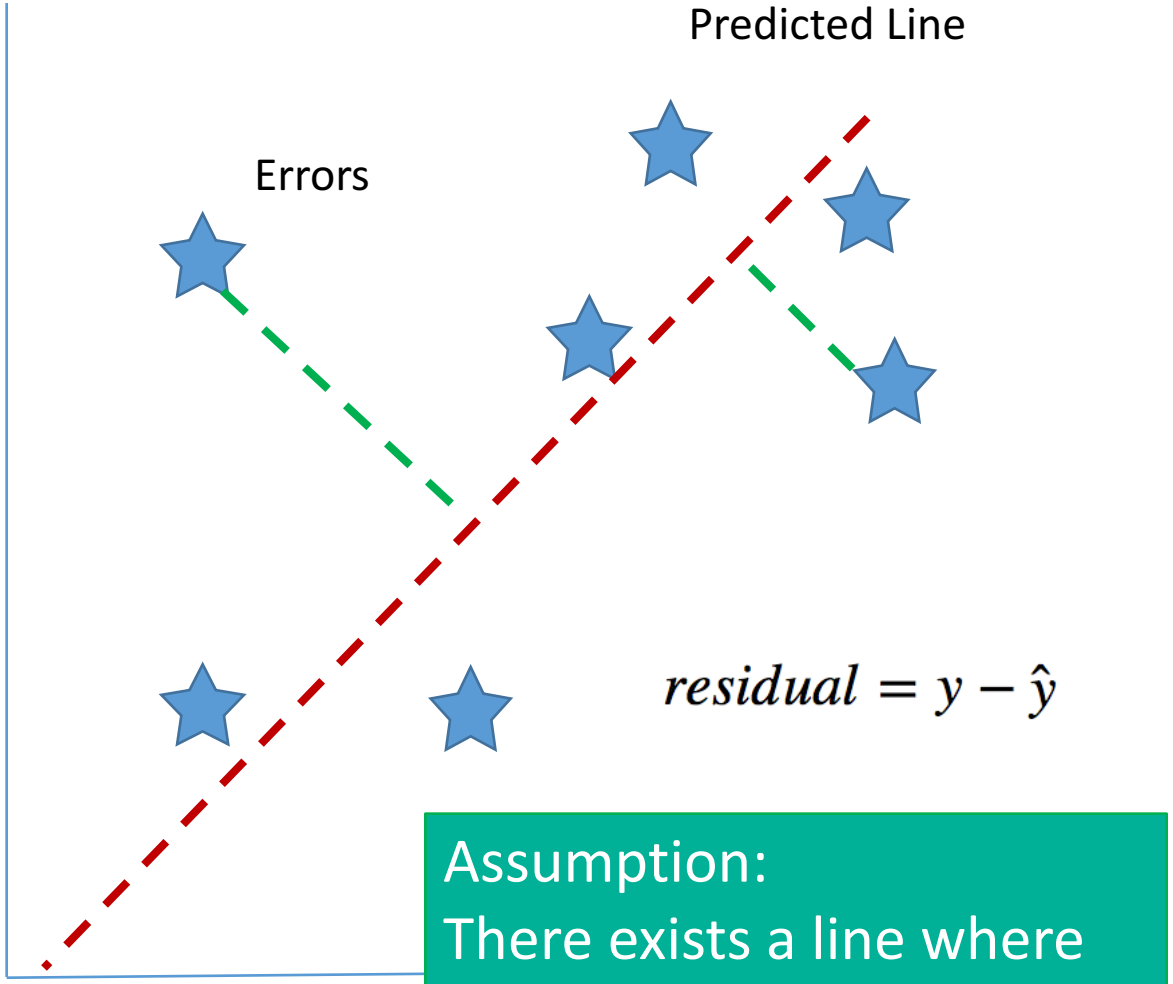
~ = predict

B_0 = intercept

B_1 = slope

Linear Regression – minimizing Errors





Assumption:
There exists a line where error is minimal

Another way to calculate slope

$$\beta_1 = \frac{cov(x, y)}{var(x)}$$

- If x varies a lot
  - Small number / big number
  - Which would be a small slope (close to a horizontal line
  - For all x, y = 5
  - If x doesn't vary a lot
  - Big number / smaller number
  - Would would be a large slope (almost a vertical line
  - Which is by definition a vertical line: for all Y, x = 3

Another way to calculate slope

$$\beta_1 = \frac{cov(x, y)}{var(x)}$$

- If x varies a lot
  - Small number / big number
  - Which would be a small slope (close to a horizontal line
  - For all x, y = 5
  - If x doesn't vary a lot
  - Big number / smaller number
  - Would would be a large slope (almost a vertical line
  - Which is by definition a vertical line: for all Y, x = 3

Sum of the deviations from mean = 0

- 1,2,3,6
- Mean = 3
- Difference from mean:
  - -2, -1, 0 ,3
- Sum of all the differences = 0

- Note on degrees of freedom

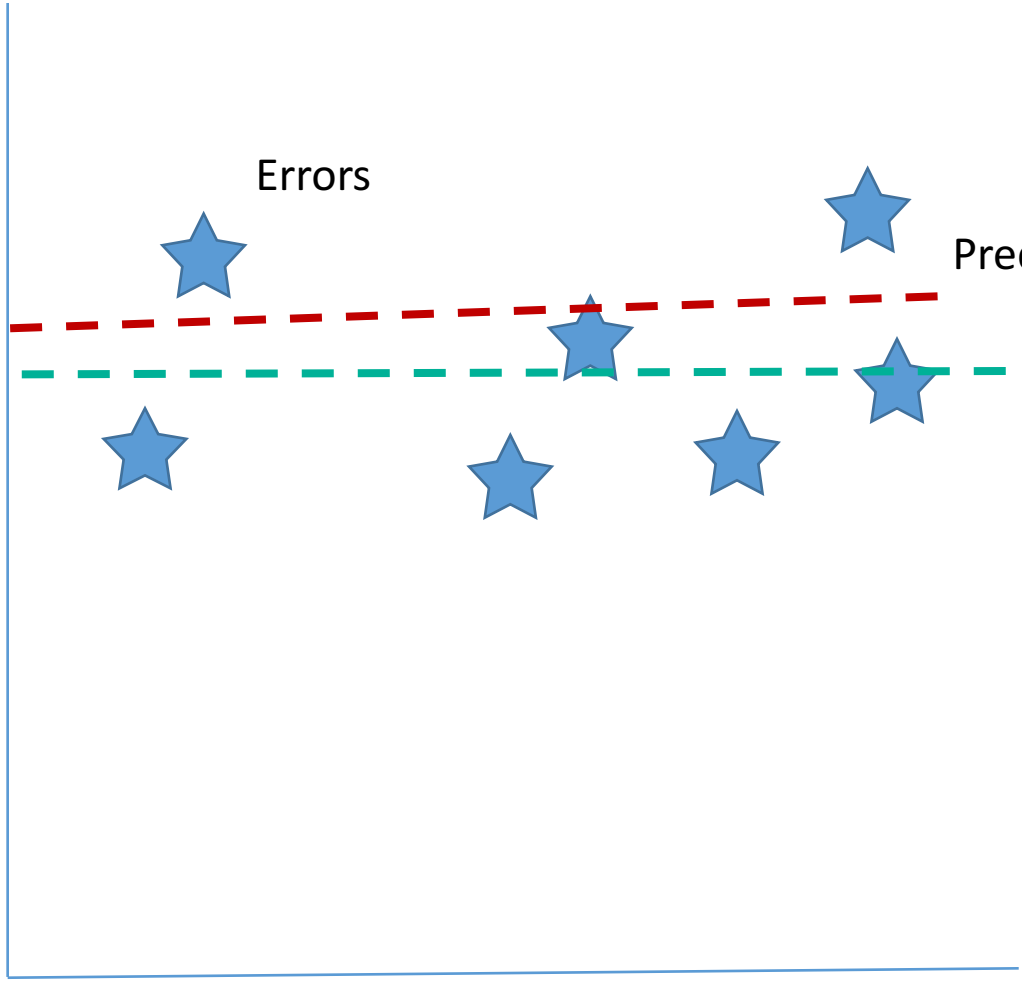
R^2 the coefficient of determination

- The amount of variance above baseline in your target y by predictor
- How much better your model is with the predictor.
- It could be that mean is the driving factor could be the nature of the dataset.
- Goes between 0 and 1. 1 means a significant effect and 0 means that your predictive efforts had no effect

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

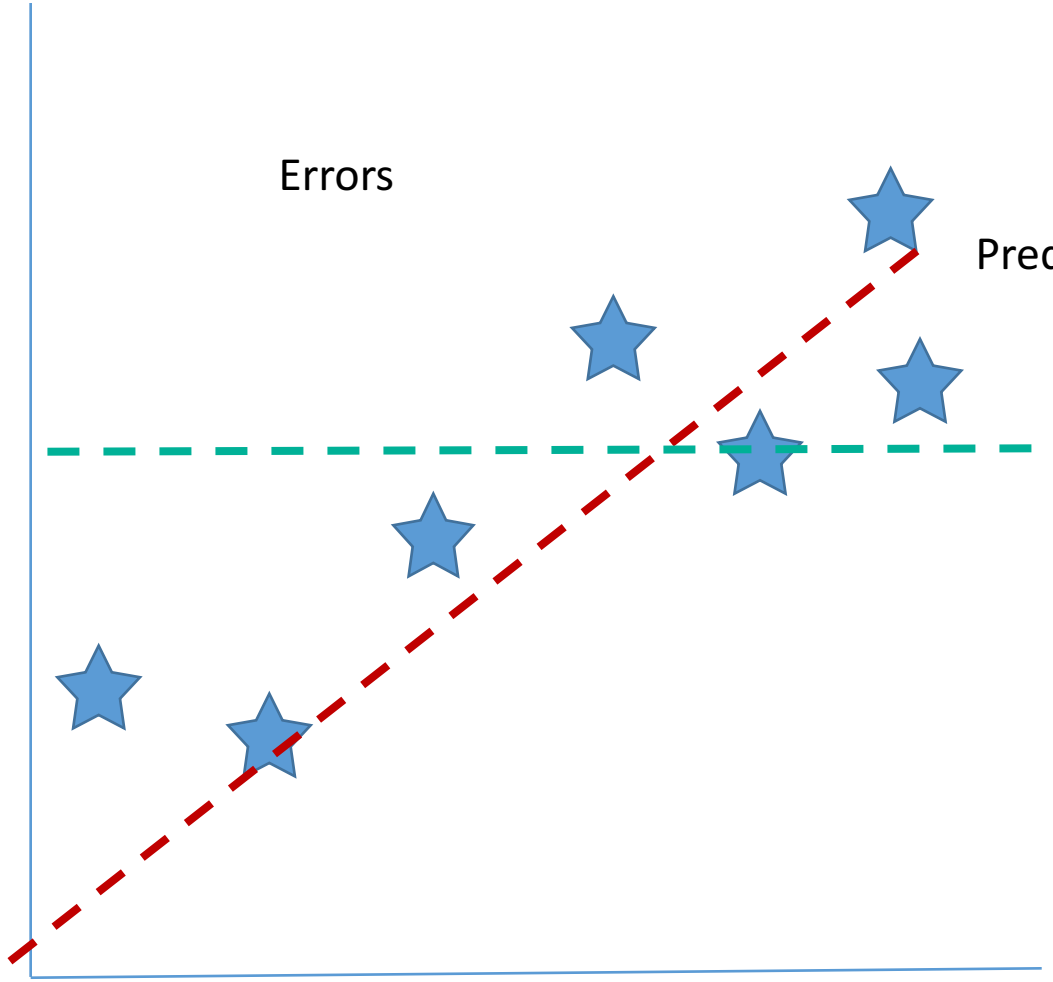
- 1 = your prediction has explained all the variance
- 0 = your prediction has explained non of the variance

R2 – sample: when R2 is low



The mean is very close to the idea target. So all the work done to get the more accurate Predicted line is incremental

R2 – sample: when R2 is low



In this case, the accurate prediction is much better htan the generics

Need another explainer slide here

What about multiple features?

- Instead of
  - $Y = b_0 + b_1x_1$
- You may have the following:
  - $Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$
- So instead of say just SQFT ~ Price
  - You may now incorporate the following
  - Bed
  - Age
  - Location
  - Etc.

Previous Linear Regression comparison

- For these:
  - $Y = b_0 + b_1x_1$
- For existing data, you may have the following:
  - $\$100 = b_0 + b_1 10$
  - $\$200 = b_0 + b_1 15$
  - $\$400 = b_0 + b_1 40$
  - $\$500 = b_0 + b_1 30$
- An from the data we fit a line for b values b_0, b_1

- For this:
  - $Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3$
- For existing data you may have:
  - $\$100 = b_0 + b_1 10 + b_2 20 + b_3 5$
  - $\$200 = b_0 + b_1 15 + b_2 10 + b_3 3$
  - $\$300 = b_0 + b_1 40 + b_2 5 + b_3 2$
  - $\$400 = b_0 + b_1 30 + b_2 7 + b_3 1$
  - ….
- An from the data we fit a line for b values b_0, b_1 , b_2 , b_3 , …

Multiple features = Matrix Math

- Instead of
  - $Y = b_0 + b_1x_1$
- You may have the following:
  - $Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4$
- So instead of say just SQFT ~ Price
  - You may now incorporate the following
  - Bed
  - Age
  - Location
  - Etc.
- As a result

Day 11: Using Scikit Learn

Scikit - Overview

- Scikit-learn provides a wide variety of machine learning opportunities including:
- Linear regression
- Logistic regression
- Support Vector Machines
- Classification And Regression Tree Models
- Naive Bayes
- Clustering Models (K-Means, Hierarchical, DBScan)

Scikit - Overview

- Also handling typical machine learning pipeline utilities for:
  - Model evaluation
  - Model selection
  - Preprocessing
  - Natural Language Processing
  - Dimensionality Reduction

- Lastly, Scikit-learn comes with a ton of datasets that are formatted nicely to work with models provided within their library.
  - Boston Housing
  - Iris Flowers
  - Diabetes Diagnostics
  - Various sample images (for classification)
  - Faces
  - MINIST (handwriting examples)
  - Random data generators
  - Spam examples
  - Newsgroup classfication

Scikit - Overview

- **Under the Hood**
  - **Numpy**: The base data structure used for data and model parameters. Input data is presented as numpy arrays, thus integrating seamlessly with other scientific Python libraries. Numpy's viewbased memory model limits copies, even when binding with compiled code. It also provides basic arithmetic operations.
  - **Scipy**: Efficient algorithms for linear algebra, sparse matrix representation, special functions and basic statistical functions.
  - **Cython**: A language for combining C in Python. Cython makes it easy to reach the performance of compiled languages with Python-like syntax and high-level operations.

Scikit – bundled datasets

- Scikit learn has a number of bundled datasest with explanations and data dictionaries
- See code on the right:
- From sklearn import datasets, linear_model

```
]: from matplotlib import pyplot as plt

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error

data = datasets.load_boston()
data = datasets.
```

```
datasets.base
datasets.california_housing
datasets.clear_data_home
datasets.covtype
datasets.dump_svmlight_file
datasets.fetch_20newsgroups
datasets.fetch_20newsgroups_vectorized
datasets.fetch_california_housing
datasets.fetch_covtype
datasets.fetch_lfw_pairs
```

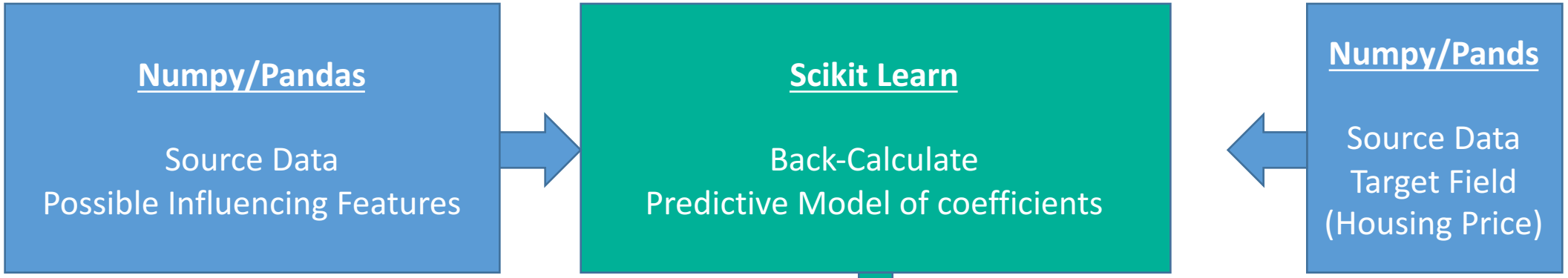
```
print                                                    ary!

# Don                                                    he notebook
%matpl

Boston

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lo
```

Process Overview

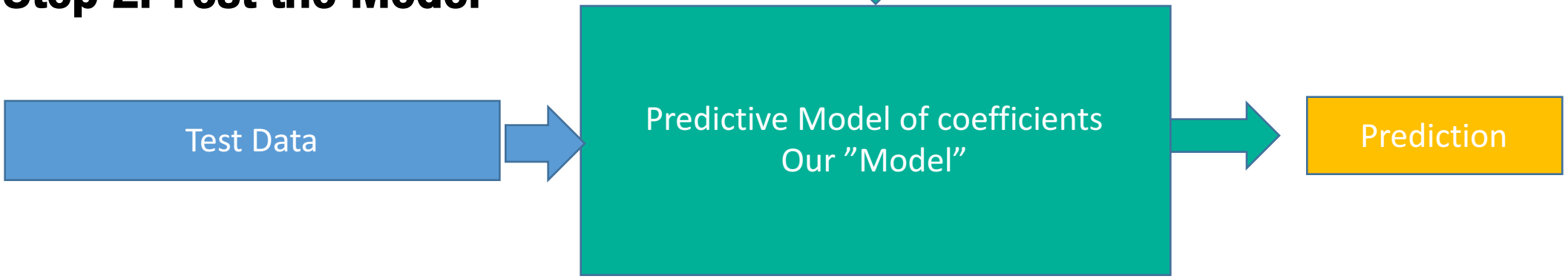
- **Step 0: Gather / Clean the data**
- **Step 1: Train the model**
- **Step 2: re-created the training data, with predictive model**
- **Step 3: Score the model (R^2 score)**
- **Step 4: use the model on future data (no answers available)**

Overview - Supervised

- **Step 1: Train the model**



- **Step 2: Test the Model**



Overview – in Table Pictures – make the model

Sample of Features that could be used

Target Metric you are looking at

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRAT	O	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3		396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8		396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8		392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7		394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7		396.90	5.33

Scikit Learn will fit these two datasets

Calculates: b_0, b_1, b_2, b_3.....

Overview with Pictures Step 2- use the model to predict new values

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0

Calculates: b_0, b_1, b_2, b_3,....

B
391.99
396.90
396.90
393.45
396.90

Overview - With Code (instead of pictures

- **Step 0 Get data**
  - X = df (… source data, which ever fields)
  - Y = df (.... TARGET FIELD ...)
- **Step 1 Train the model with existing data**
  - Lm = scikit.linear_model.linearRegression()
  - Model = lm.fit (x,y)
- **Step 2: use the model for prediction**
  - New_predict = model.predict(some new x)
- **Step 3: score the model**
  - Score = model.score(X,y)

Note: a great model won't output a perfect copy of the results:

- **Left:** Linear model fit to some scatter data
- Because the input data is a cloud and has scatter, no linear model can ever perfectly re-create the original training model. Because: a line can't be a cloud.
- Similarly, models by their nature will "fit" the original data, but inherently are approximations

Day 12: More on Loss Functions

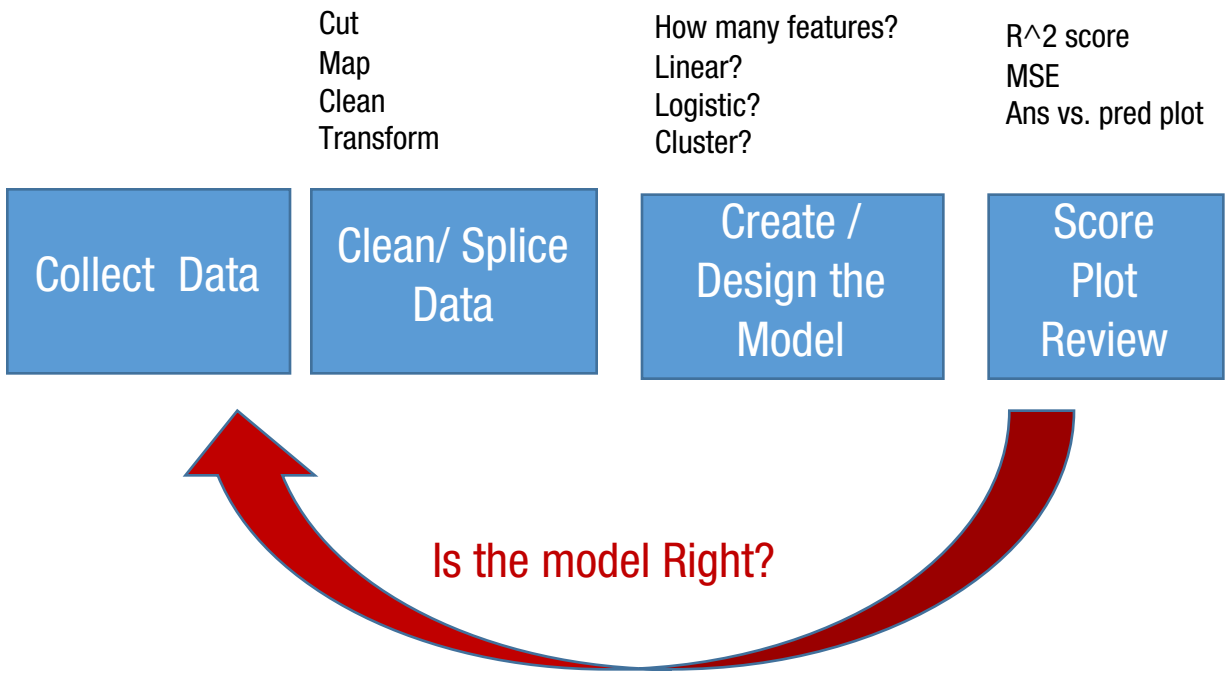
Most of the morning was doing LABS – reviewing the scikit learn model and the stats model

Day 12: Cross Validation

How to split your test data vs. your source data

Pulling data out for modeling

The standard supervised modeling process:



This is the standard modeling process below.

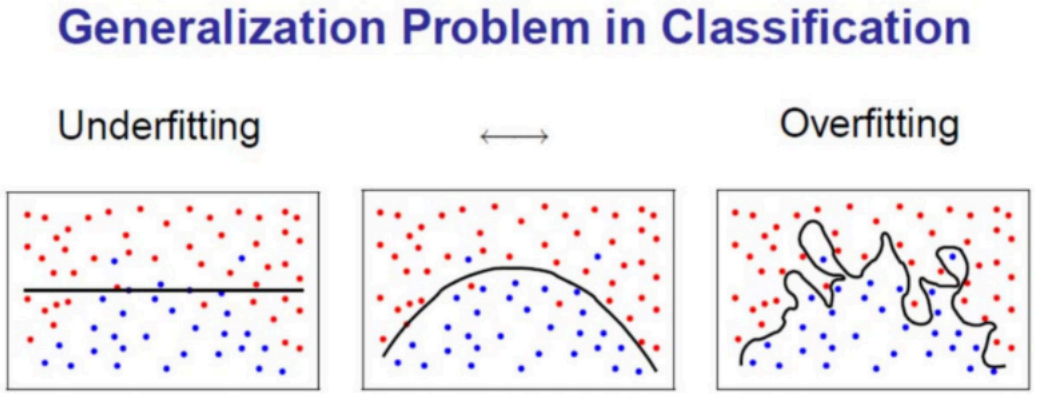
From a higher level, examination, what can go wrong from a modeling standpoint?

How do you know that the model that has been generated is correct?

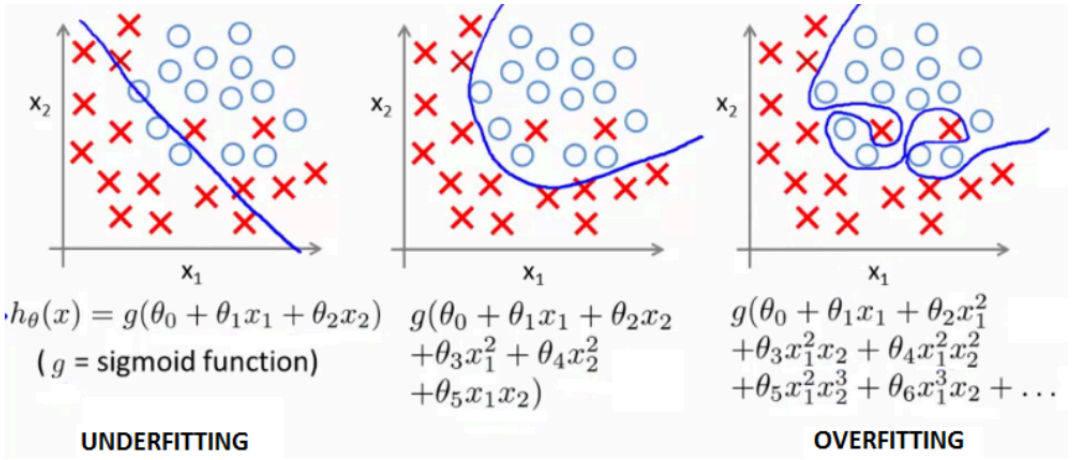
How do you **validate? How do you know your model is good?**
(especially if you are working of a small sample?)

When Predictions goes wrong

- Even after prediction, there may be issues. (see the plots on the right).



Another Example

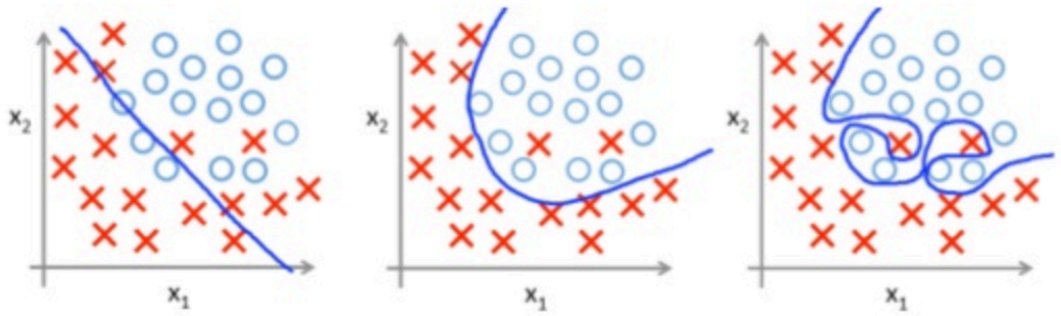


Overfitting

Overfitting

Learn the “data”and not the underlying function

Overfitting may occur when learned function performs well on the data used during the training and poorly with new data.



Introduction

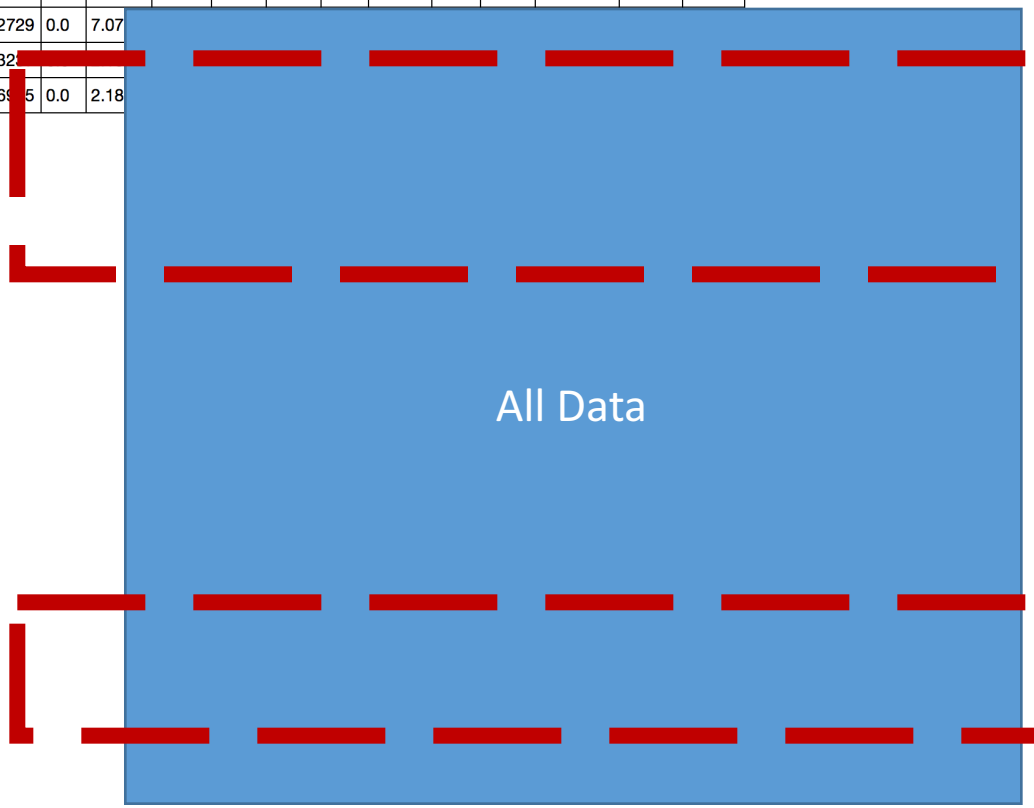
- So far we've focused on fitting the **best model** to our data. In practice, we need to also make and **validate predictions** with our models.
- By **splitting our data** set into a **subset to train our model** on and a subset to make and test predictions on, we can **validate the effectiveness** of our model. This is called a *train/test split* and we'll explore a number of ways to effectively carry out the split. It's also a good way to avoid overfitting on your dataset (but not always).
- **Test/train split benefits:**
  - **Save a subset of data to make predictions**
  - **Can verify predictions without having to collect new data (which may be difficult or expensive)**
  - Can help avoid overfitting
  - Improve the quality of our predictions

About Cross-Validation

- We use cross-validation to sample our data in order to understand how it may perform in a variety of cases, given a set of parameters. It also helps us understand how predictions react to data. Largely, we are "testing" how our model stands up to basic assumptions, given a model.
- Why Validate?
  - Test the model
  - Avoid overfitting
  - How well a model generalizes to an independet dataset
- *The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like overfitting, give an insight on how the model will generalize to an independent dataset (i.e., an unknown dataset, for instance from a real problem), etc.*

Pulling data out for modeling

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07										
3	0.032												
4	0.069	0.0	2.18										



Take data samples to test the model

How big of a sample size? How big should the base data be? How big should the test data be?

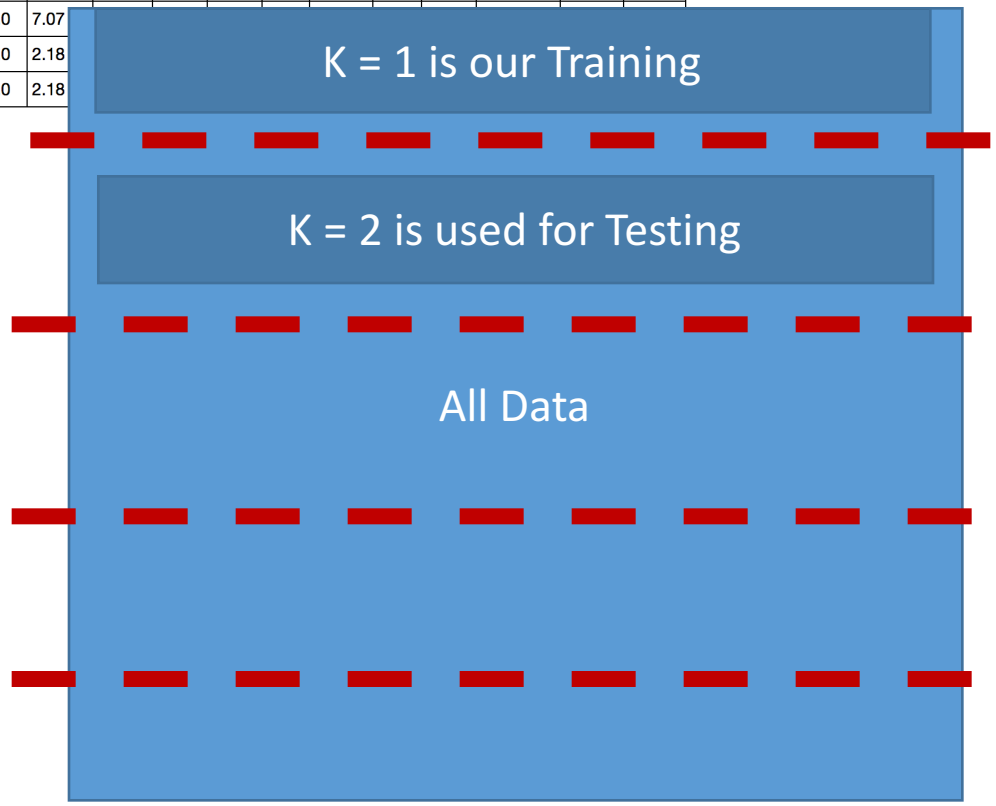
Take different data samples to test the model

K-Fold Validation

- Essentially, K-Fold splits our dataset up into K-Folds, and uses one segment to test.
  - Split data into K folds (subsets)
  - Use K-1 for training
  - Use 1 for testing
  - Repeat K times
  - Mean results

K-Folds Subsectioning

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07										
3	0.03237	0.0	2.18										
4	0.06905	0.0	2.18										



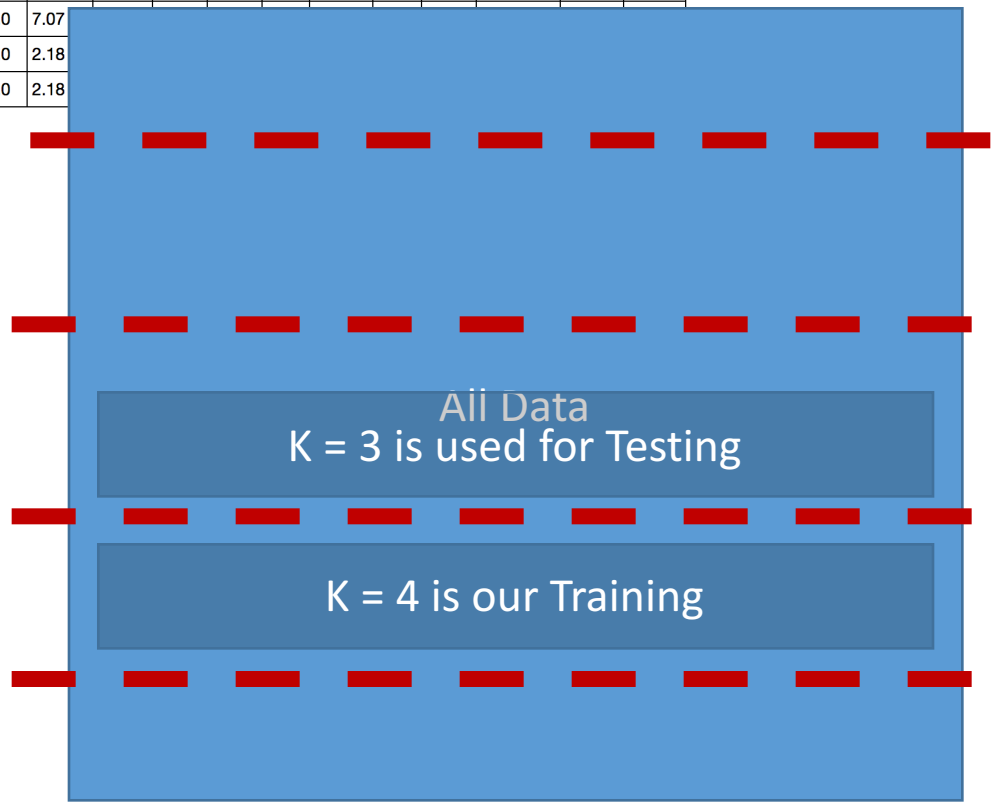
The data is divided into *K* folds (subsections, in this case 5 subs)

Once different subs have been selected, a score will be generated:

Score 1,2 = 3948….

K-Folds Subsectioning (iterating on..)

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07										
3	0.03237	0.0	2.18										
4	0.06905	0.0	2.18										



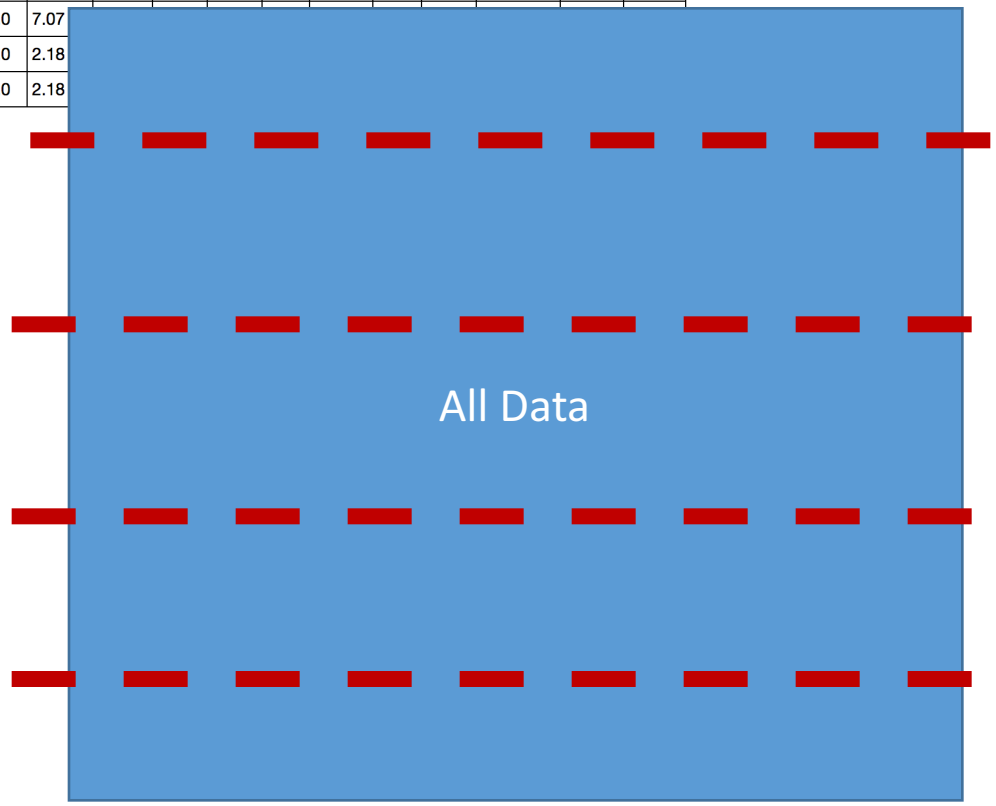
The data is divided into *K* folds (subsections, in this case 5 subs)

Once different subs have been selected, a score will be generated:

Score 4,3 = 1245….

K-Folds Subsectioning (iterating on..)

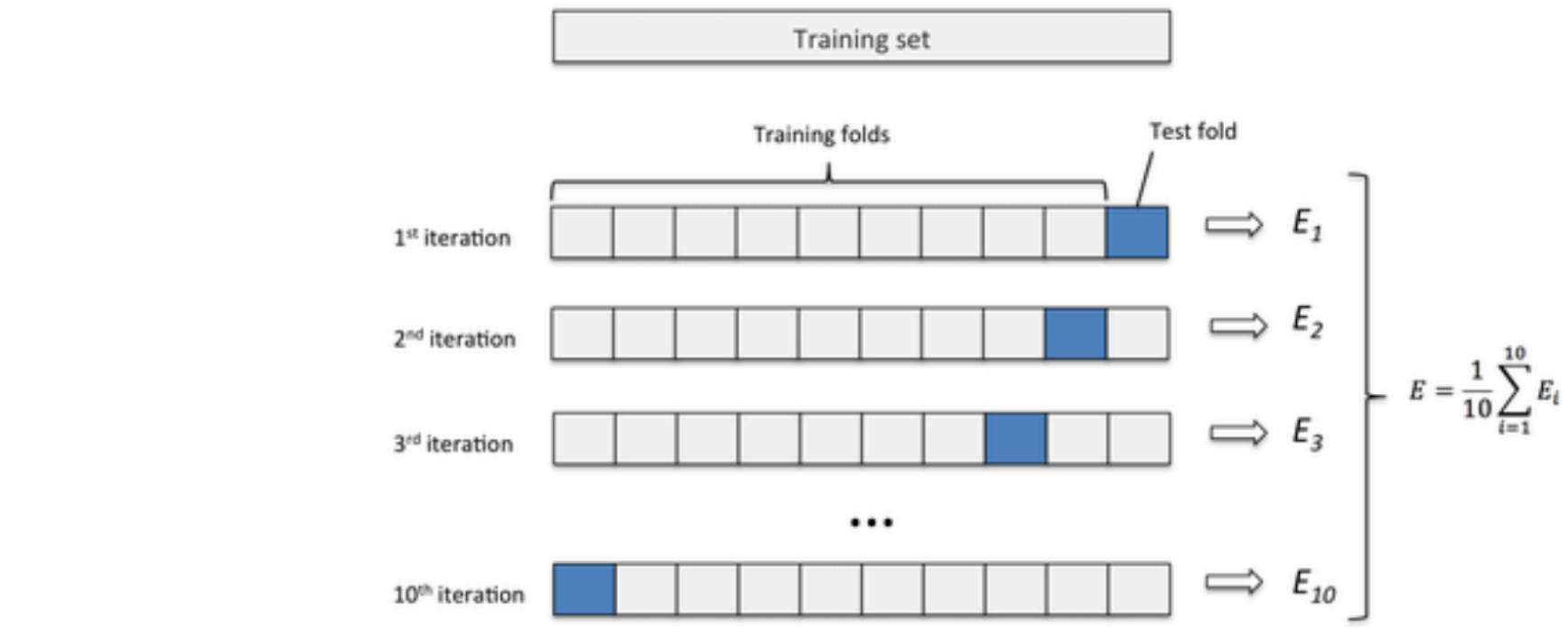
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07										
3	0.03237	0.0	2.18										
4	0.06905	0.0	2.18										



After all the testing has occurred. All the scores are compared and a mean score will be used. The consistency of the mean score will tell you a rough quality of the model

Score 1,2:
Score 1,3:
Score 3,4:
Score 4,2:
….

K Fold Validation in Action



A quick note about "Hold Out" validation

- Briefly, sometimes you'll hear about "hold out" validation. Essentially, you ommit a section of data from any validation setup, in order to truly test unknown data against your model.
- The main reason you might try this:
- Test your "winning" model against truly clean data
- "Truly clean" is data a model has never seen

Cross Validation: Train Test Split

```
]: X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.4)
   print X_train.shape, y_train.shape
   print X_test.shape, y_test.shape

   (265, 10) (265,)
   (177, 10) (177,)
```

- from sklearn.cross_validation import **train_test_split**
- **Df** = input dataframe
- **y** = headings that will be incorporated
- **Test_size =** percent of the test size,
  - **Implicity: 1-test_size = train_size**
- **Output :** training sets and result sets for X and Y

Once its split:

- 1. The training datasets are used
- 2. Then the testing datasets are used

```
:  lm = linear_model.LinearRegression()

   model = lm.fit(X_train, y_train)
   predictions = lm.predict(X_test)

   ## The line / model
   plt.scatter(y_test, predictions)
   plt.xlabel("True Values")
   plt.ylabel("Predictions")

   print "Score:", model.score(X_test, y_test)
```

How can this be done for *k* subsets?

- Cross Validation score or predict
- Cross_val_score
  - Takes in a model in and uses subsets to validate the scores.
  - Or can take a generic (linear or logistic) estimator and run all the items
- Cross val_predict

```
:
from sklearn.cross_validation import cross_val_score, cross_val_predict
from sklearn import metrics

# Perform 6-fold cross validation
scores = cross_val_score(model, df, y, cv=6)
print "Cross-validated scores:", scores

# Make cross validated predictions
predictions = cross_val_predict(model, df, y, cv=6)
plt.scatter(y, predictions)

accuracy = metrics.r2_score(y, predictions)
print "Cross-Predicted Accuracy:", accuracy
```

Summary

- **Conclusion:**
  - A model generated with only one data set can be scored
    - But multiple scores is much better than a single score.
  - This can be done manually, cutting up different datasets and testing
  - But K-folds testing can easily assist multiple-scoring, running k number of subsets and testing of a model against the dataset

Day 12:s Feature scaling

How to split your test data vs. your source data

Summary

- To round out our workflow in what we sometimes refer to as the **preprocessing pipeline**, we will be working with normalizing features within our dataset(s), and converting our data to design matrices with **Patsy**, before using models with a linear component (ie: linear regression).

Why Feature Scaling

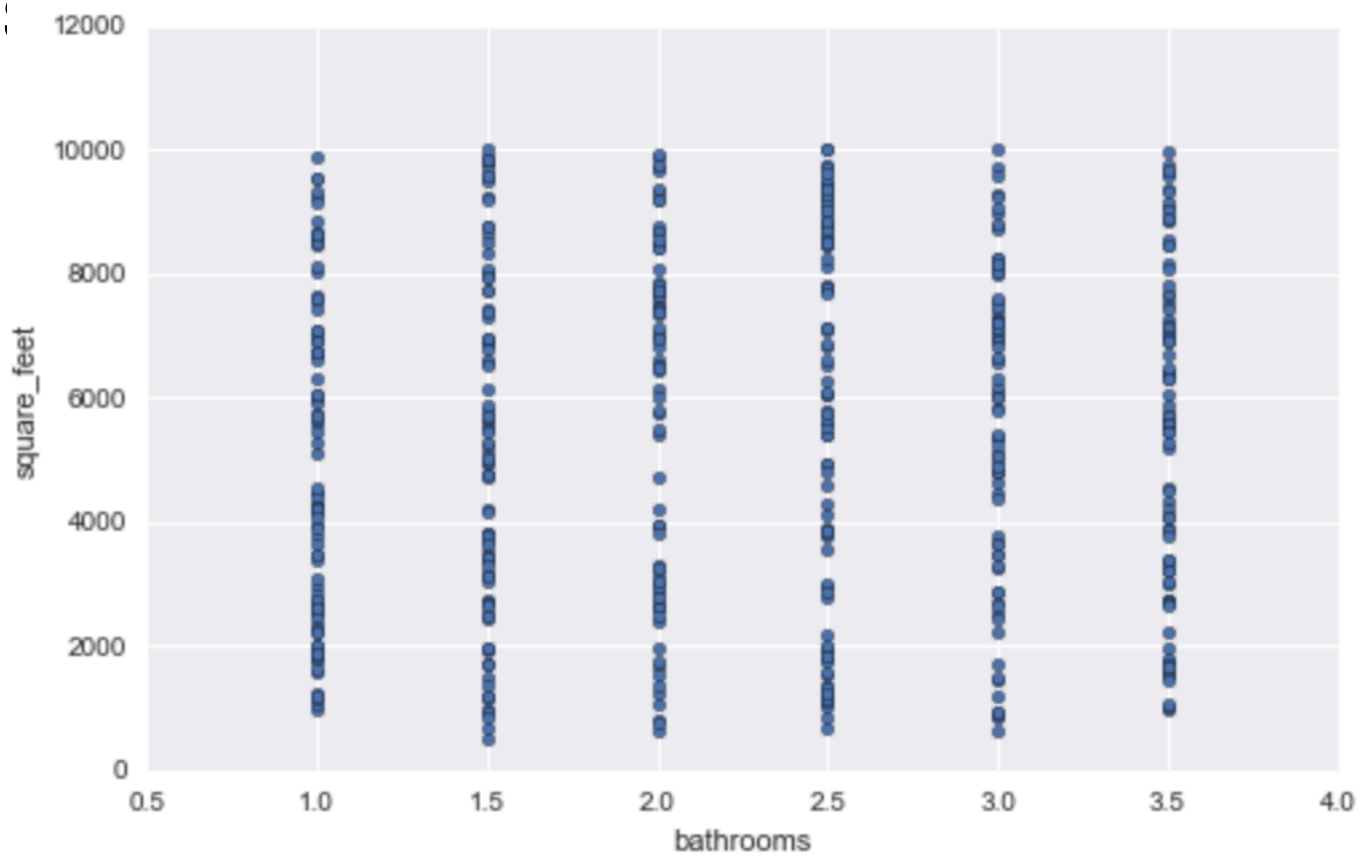
- The majority of machine learning methods will perform better when our variables are on the same scale.
- Some methods will optimize for larger errors in spaces that have a bigger range in the case where some features are weighted by errors.
- Other cases include methods that rely on euclidean distances (ie: KNN, K-Means), or linearly sperable points, which means that scaling can have a desirable impact.
- The most obvious cases were this makes sense is when you have features that are drastically different *and* we want them within a bound interval.

Why Feature Scaling

- **Normalizing or standarding prevents features from dominating each other in machine learning. The larger numbers are going to be larger in weight.¶**
- Consider the example we've talked about a times when we are predicting the value of real-estate properties, and we have a variable "square feet" within 1-10,000, and "number of bathrooms" between 1-3.

Sample of two features Sqft and Bdrms:

- Square foot vs. Bathrooms
- Bathrooms go from
  - 1-3.5
- Square Foot goes from:
  - 1000-10,000



Min-max scaling (all from 0 to 1)

- **Min-max scaling** (sometimes called *normalization*), which typically means rescaling features from 0 to 1. There are a lot of different methods for transforming your data to scale. The methods in sklean.preprocessing are extremely fast compared to doing this calculation by hand and are very well tested.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Standardization (all values from 0 to mean)

- Alternatively, we may want to use **standardization** to scale our variables to 0 for the mean, with a **standard deviantion of 1**, in effect creating a normal distribution. This actually makes it easier for machine learning methods to learn variable weights (ie: stochastic gradient descent, Newton-CG, or OLS regression!)

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean

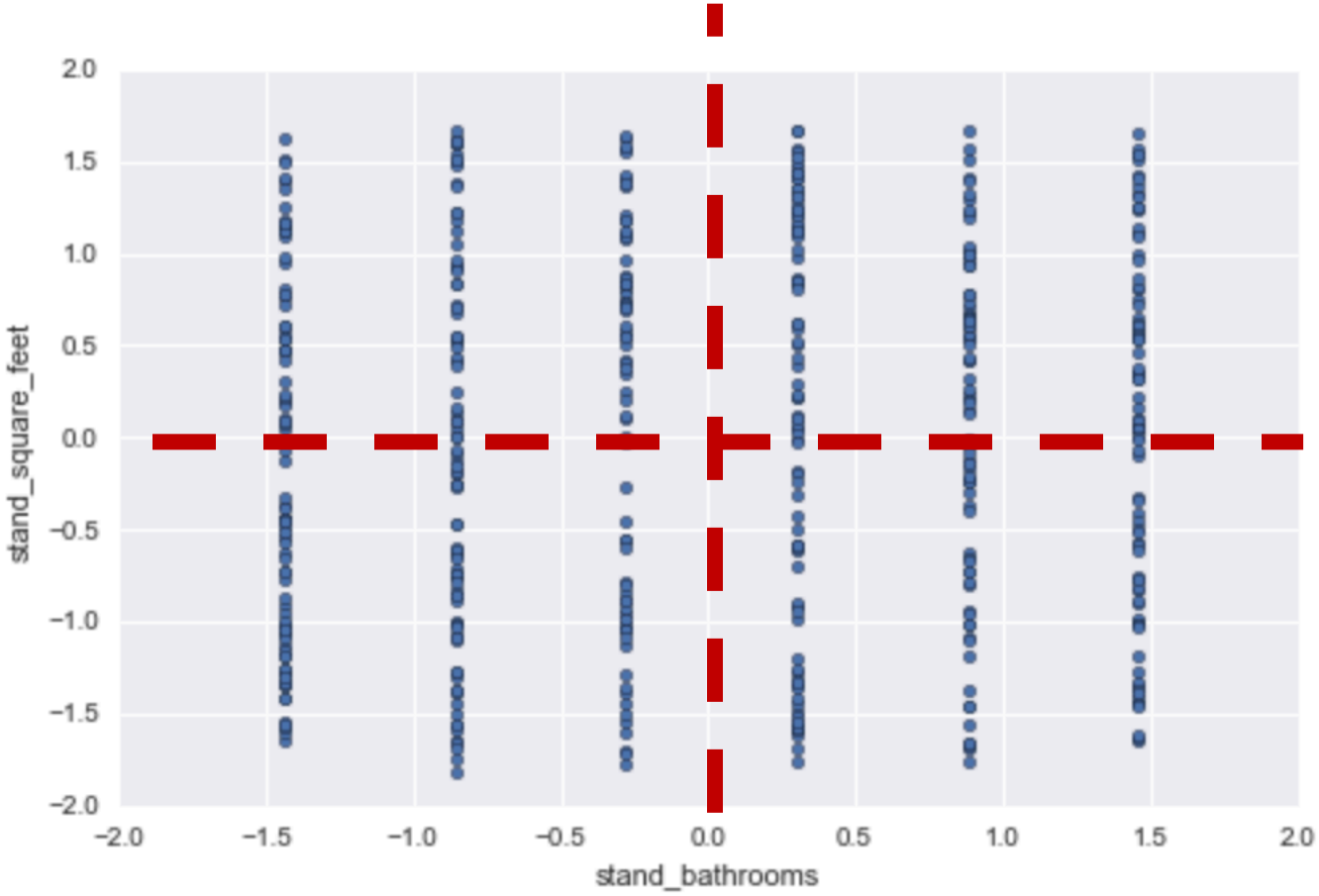
σ = Standard Deviation

Comparison

- Where **min-max normalization** is sensitive to outliers, **standardization** is not as sensitive to outliers but still retains some of those characteristics because the range isn't limited to a set of fixed values (0-1).

Scaled plot (same example)

*Note
Mean is 0



* Note the Mean is zero

Some Guidelines

- **Some Guidelines¶**
- Standardize when there is a huge variance in values.
- Standardize when a single feature can have both a positive and negative effect.
- **Consider** normalizing anything that will be used with Euclidean distance.

General Questions

- **General Questions**
- Is the algorithm heavily influenced by different scales of different features?
- Do your features share the comparable scales already?
- Does your algorithm already normalize your data? *(Linear Discriminant Analysis and Naive Bayes)*
- Does it care? *(ie: decision trees / CART models)*

Feature Scaling

- **Feature Scaling**
- Fitting your model will be faster on larger datasets.
- Prevents getting stuck in local optima.
- When scales of variables are vastly different.
- When we want small valued features to contribute more equally.

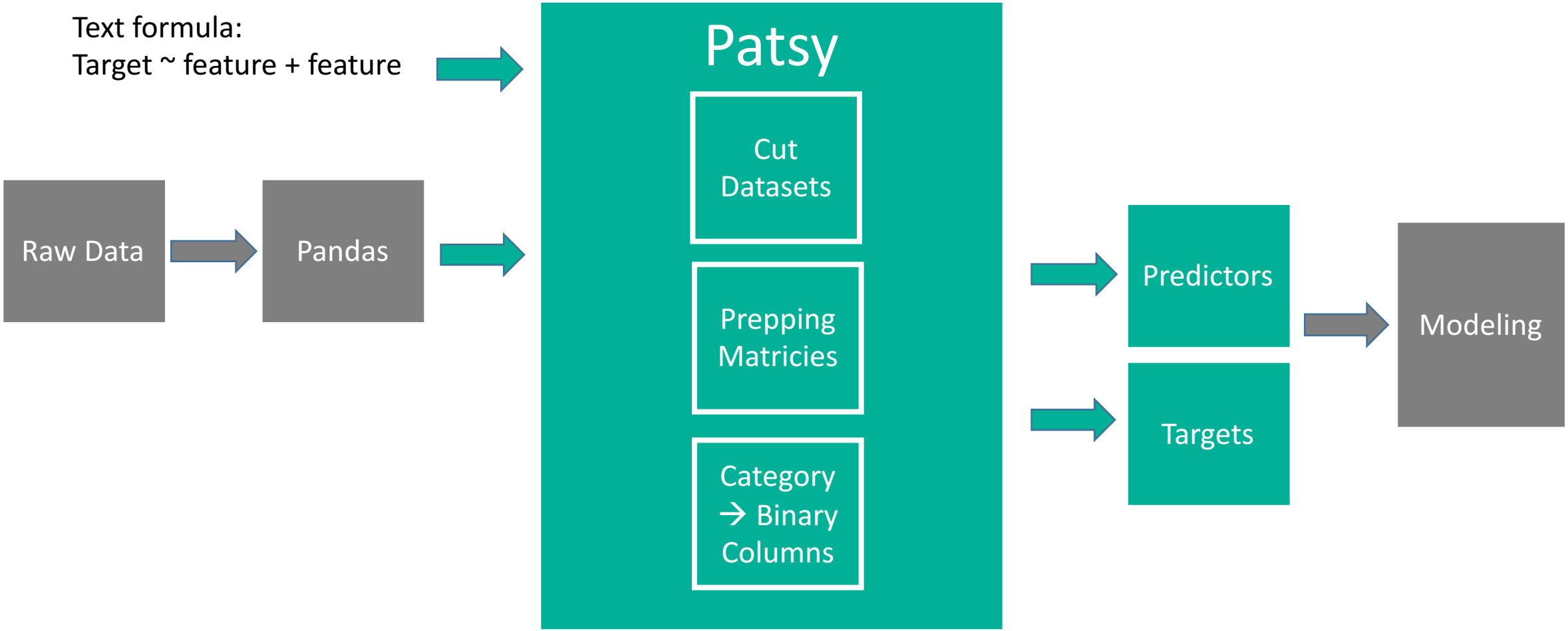
How to use the StandardScalar packages

from sklearn.preprocessing import StandardScaler

- Scalar = StandardScaler()
  - This could be either standard or minmax
- Columns = [‘ship_value’,’ship_speed’]
- df[columns] = scalar.fit_transform(df[columns]])

Day 13: Patsy!

Patsy Package Overview



Patsy! Package

- **Patsy!¶**
- *patsy is a Python package for describing statistical models (especially linear models, or models that have a linear component) and building design matrices. It is closely inspired by and compatible with the formula mini-language used in R and S.*
  - Yes we learned a lot this week. Yes Patsy may seem like yet another unnecessary step but we it's actually very useful. We will explore this utility that gives us the power of design matrices in a live coding example.
  - Check out the docs: https://patsy.readthedocs.io/en/latest/overview.html
  - Patsy is not just useful for linear models, but anytime you may want to encode or modify your dataset in preprocessing step before modeling.

Some sample code

- formula = **"baths ~ ship_type + ship_value + ship_speed”**
  - It’s a rough text interpretor of the data field headings
- y, X = patsy.dmatrices(formula, data=df, return_type = "dataframe")
  - Returns Y, and x (just as similar models decomposition Y answers and x inputs)
  - Will return a dataframe if specified, otherwise a complex data type is returned
  - Y returned is a split
- **Design Matrix** – specific to patsy

Log Space get a range of Log values

- Np.logspace

More on patsy formulas…

- formula = **"baths ~ ship_type + ship_value + ship_speed”**
  - It’s a rough text interpretor of the data field headings

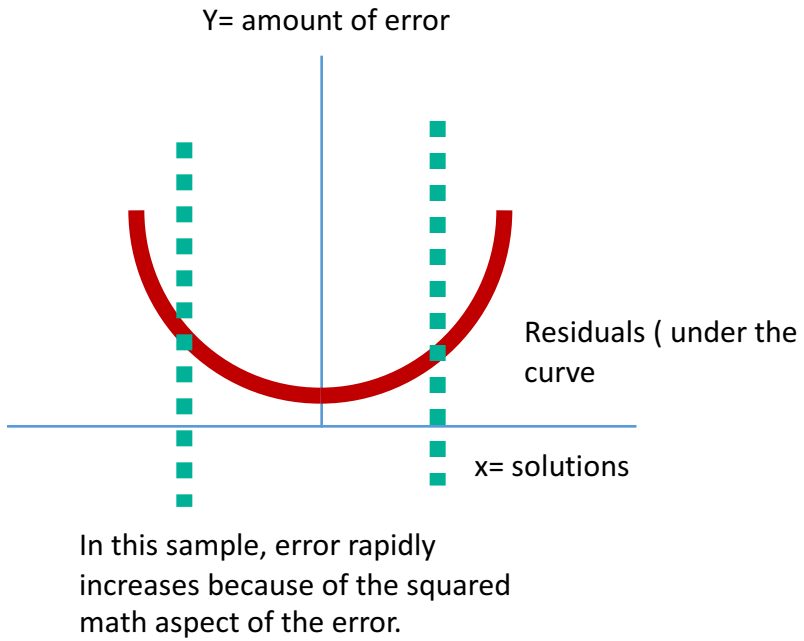
- Formula = **”target ~ (A + B)**2”**
- **(A + B)**2” = A**2 + AB + B**2**
- Formula – the above code looks at all interactions between multiple factors, can be used for any number of factors (A + B + C…..)

Day 13: Regularization

How do you reduce your error of your model?

Regularization – controlling the impact of features

- If you use regularization, you have to normalize all your predictors
- Lasso and Ridge Techniques
- Why?
- Considering the error of a predicted model
  - What if certain features are driving the steep error?
  - What if these features increase the error, and also do not contribute to accuracy?
  - They should be minimized / eliminated
  - This will be done with Coefficients



Regularization (cont'd)

- How do you minimize the effect of different features? (see right)
  - We change the b1
- So we add additional terms to counteract the different features

If x1 is bad -- How do you minimize x1?

$$y=b_0 + b_0x_0 + b_1x_1 + b_2x_2$$

$$\min(RSS) = \sum_{i=0}^{n} (quality_i - (b_0+b_1x))^2 + \lambda_1 \sum_{\beta_1}^{\beta_2} |\beta| + \lambda_2 \sum_{\beta_1}^{\beta_2} \beta^2$$

Lasso Regularization

Ridge Regularization

Equation Recap

$$\text{Ridge penalty} = \lambda_2 \sum_{i=1}^{n} \beta_i^2$$

$$\text{Lasso penalty} = \lambda_2 \sum_{i=1}^{n} |\beta_i|$$

Equation Recap

- Residuals

$$\text{minimize } RSS = \sum_{i=1}^{n}(y_i - \hat{y}_i) = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_i x_i))$$

- Ridge

$$\text{minimize } RSS = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_i x_i)) + \lambda_2 \sum_{i=1}^{n}\beta_i^2$$

- Lasso

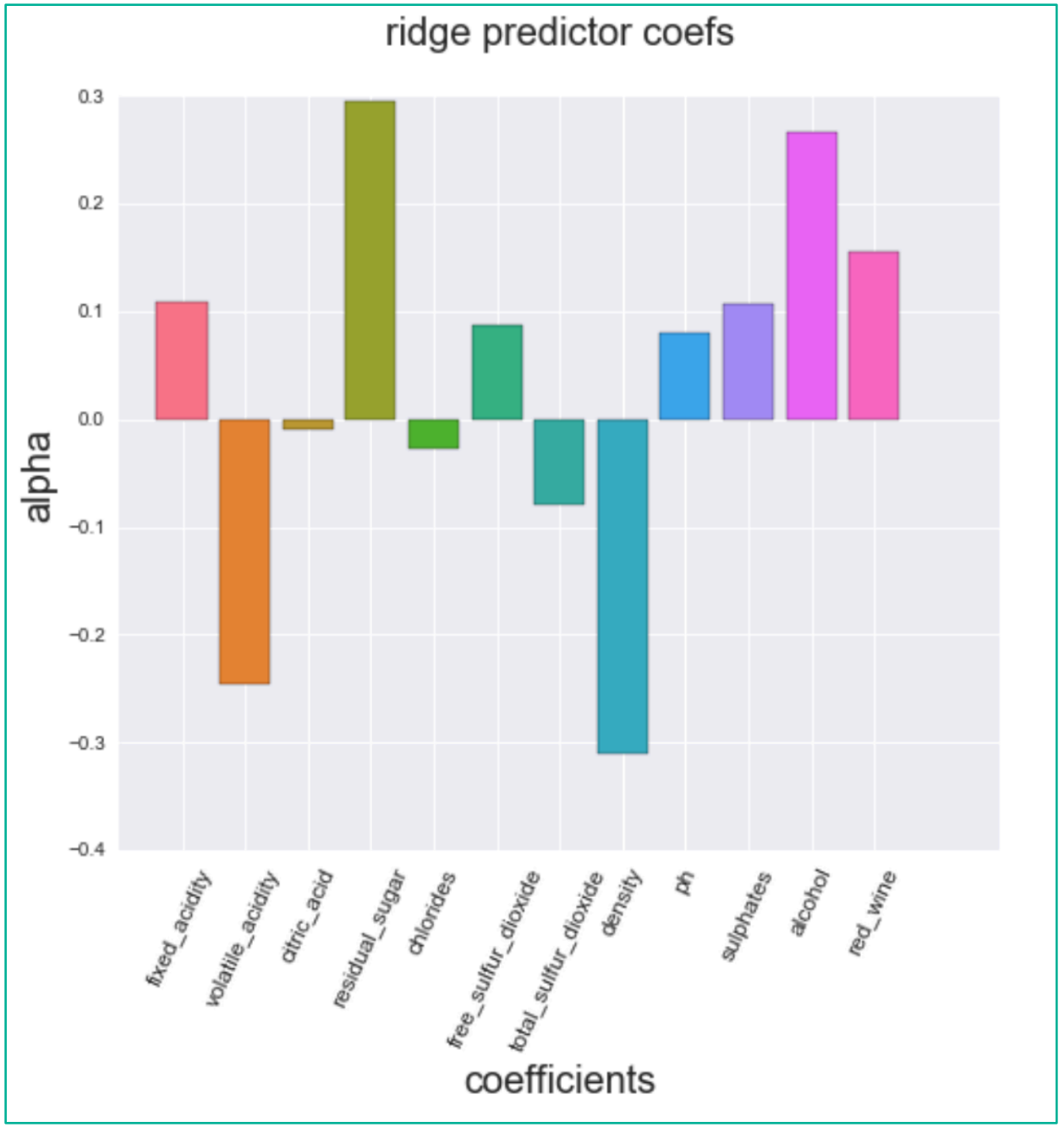
$$\text{minimize } RSS = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_i x_i)) + \lambda_1 \sum_{i=1}^{n}|\beta_i|$$

- Elastic

$$\text{minimize } RSS = \sum_{i=1}^{n}(y_i - (\beta_0 + \beta_i x_i)) + \lambda_1 \sum_{i=1}^{n}|\beta_i| + \lambda_2 \sum_{i=1}^{n}\beta_i^2$$

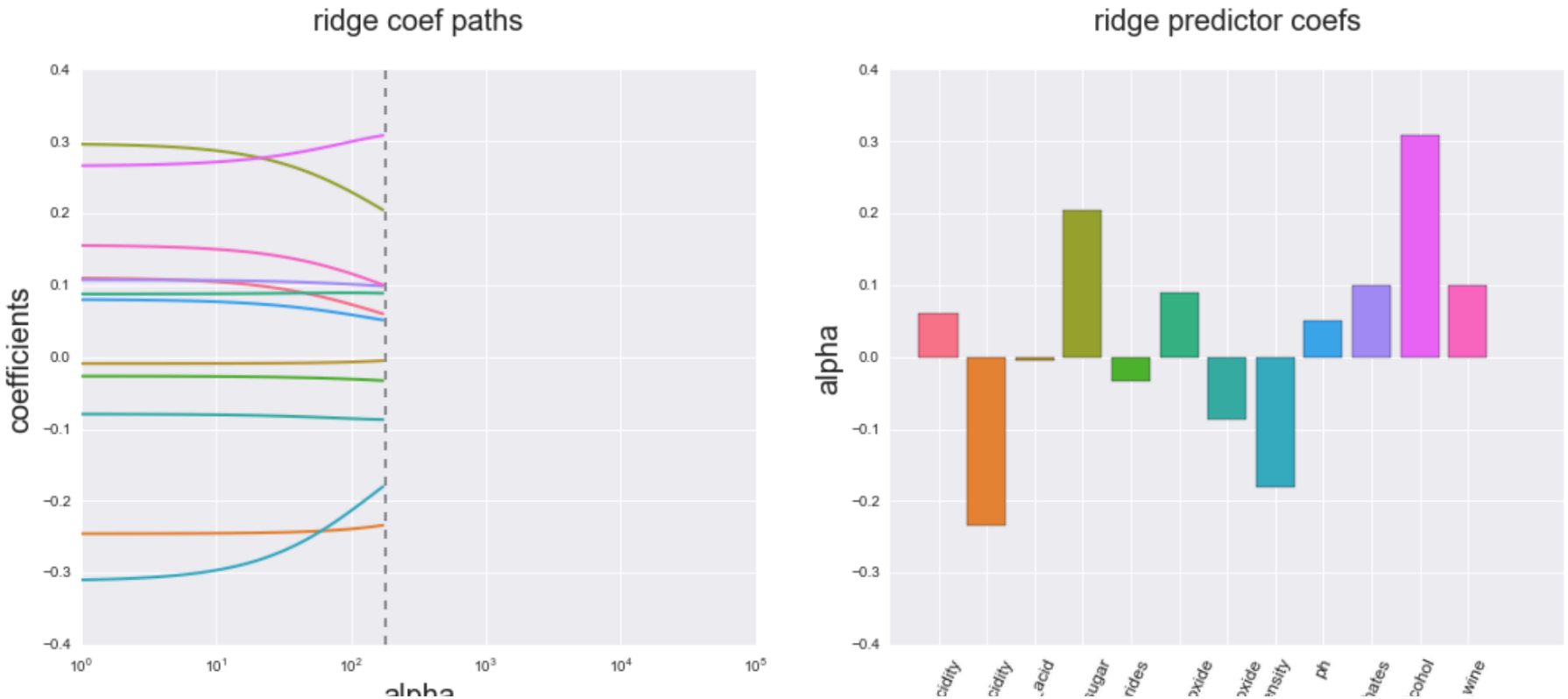
Sample

- **Before regularization (right)**
- **Ridge** – minimize large R^2 affect.
  - The difference between olive green and teal
- **Lasso** – drops unnecessary small features
- **Elastic** – drops small features and minimizes the others

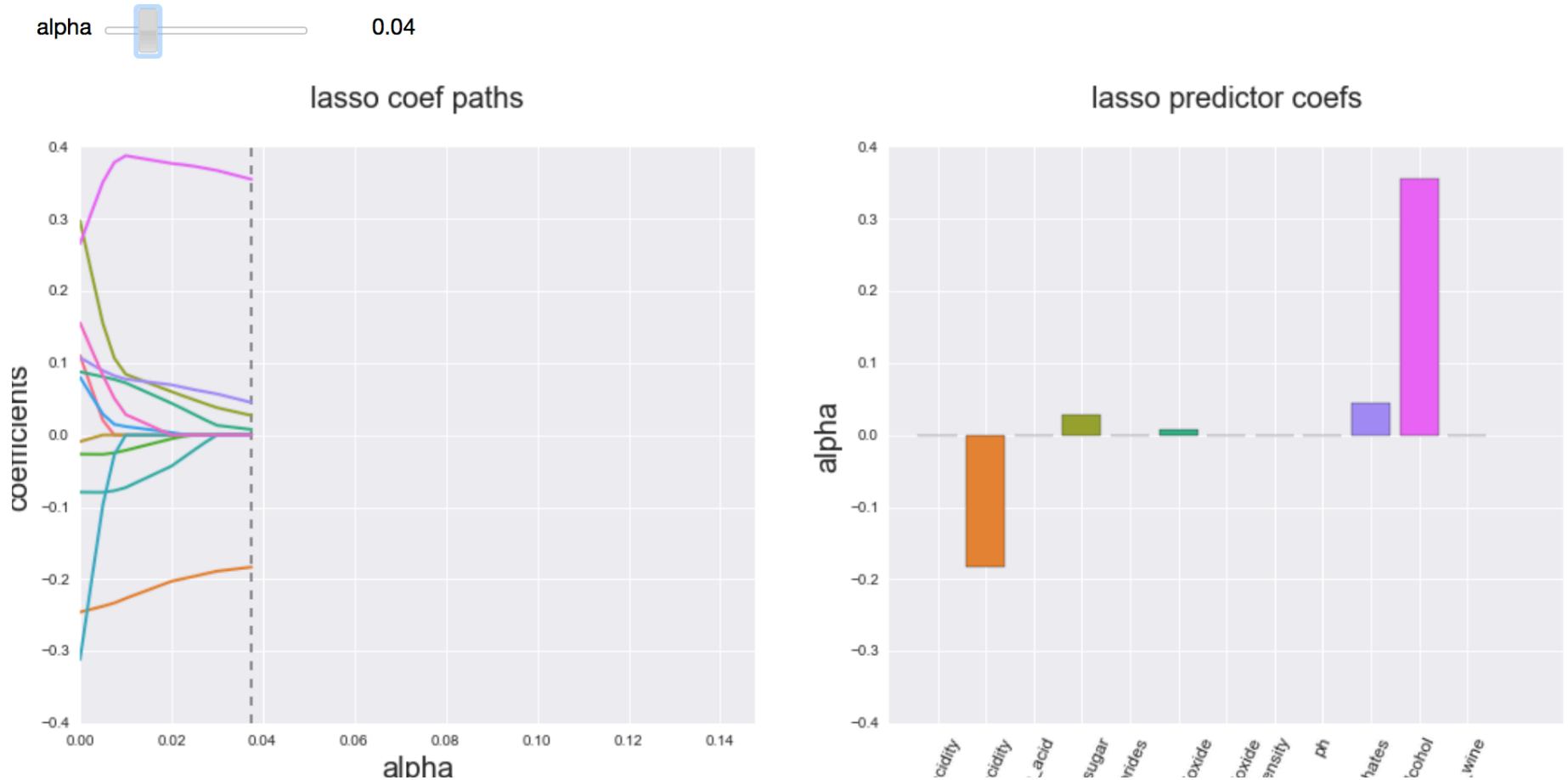


With Ridge – minimizes the R^2 effect, but leave all variables

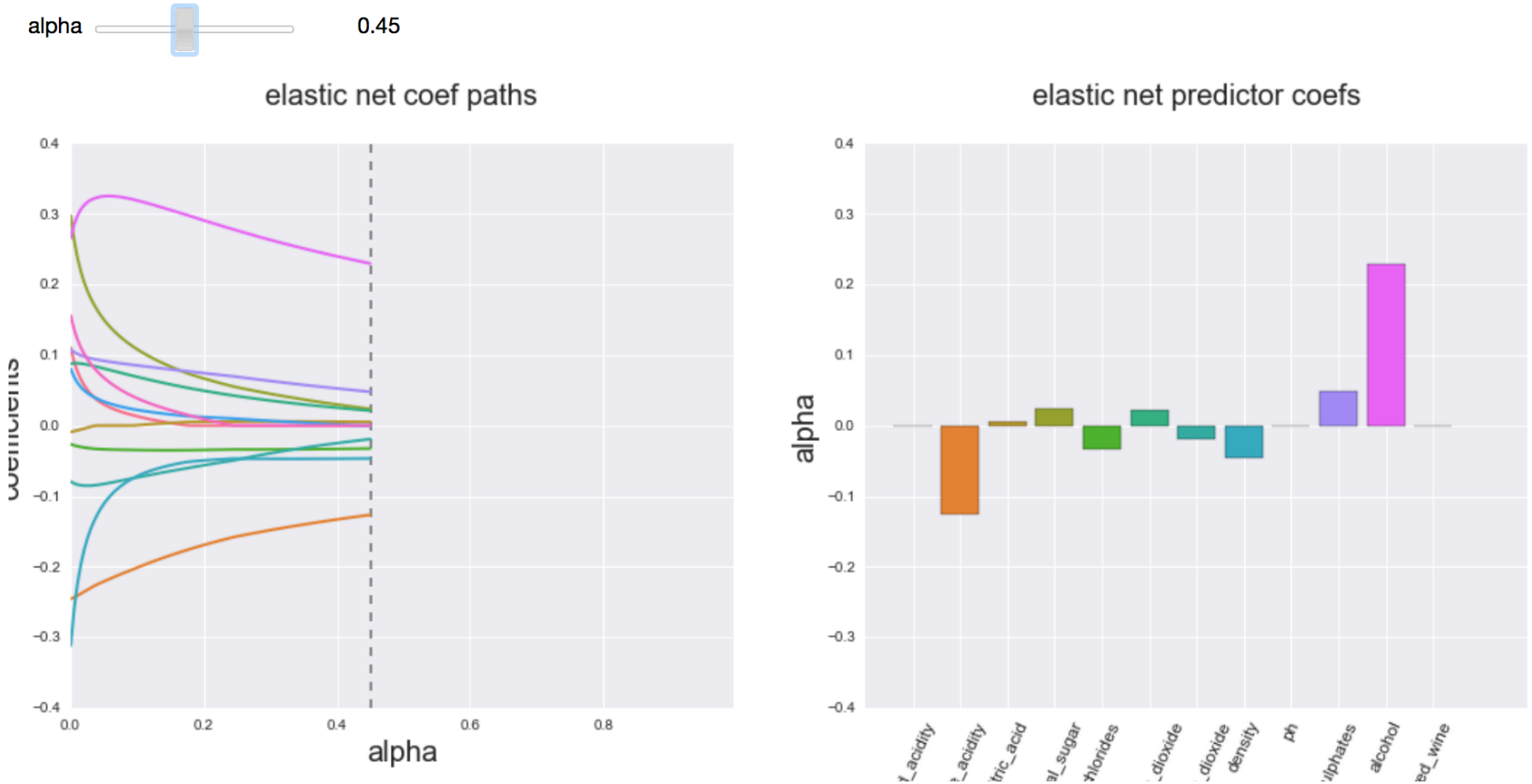
log_of_alpha 2.25



With Lasso – drops unnecessary small features



Elastic – take out bad variables and minimize



On Ridge CV and Lasso CV

- Will automatically test different conditions to output a valid:
  - Can input the different ranges of alpha to test
  - Can set the k-fold cross validation range

Day 14: Bias and Variance

How do you reduce your error of your model?

How good (or bad) is your predictive model

- You could build a terrible model. It could be –R^2. Which would mean whatever you made was worse than guessing the mean value without looking at everything else

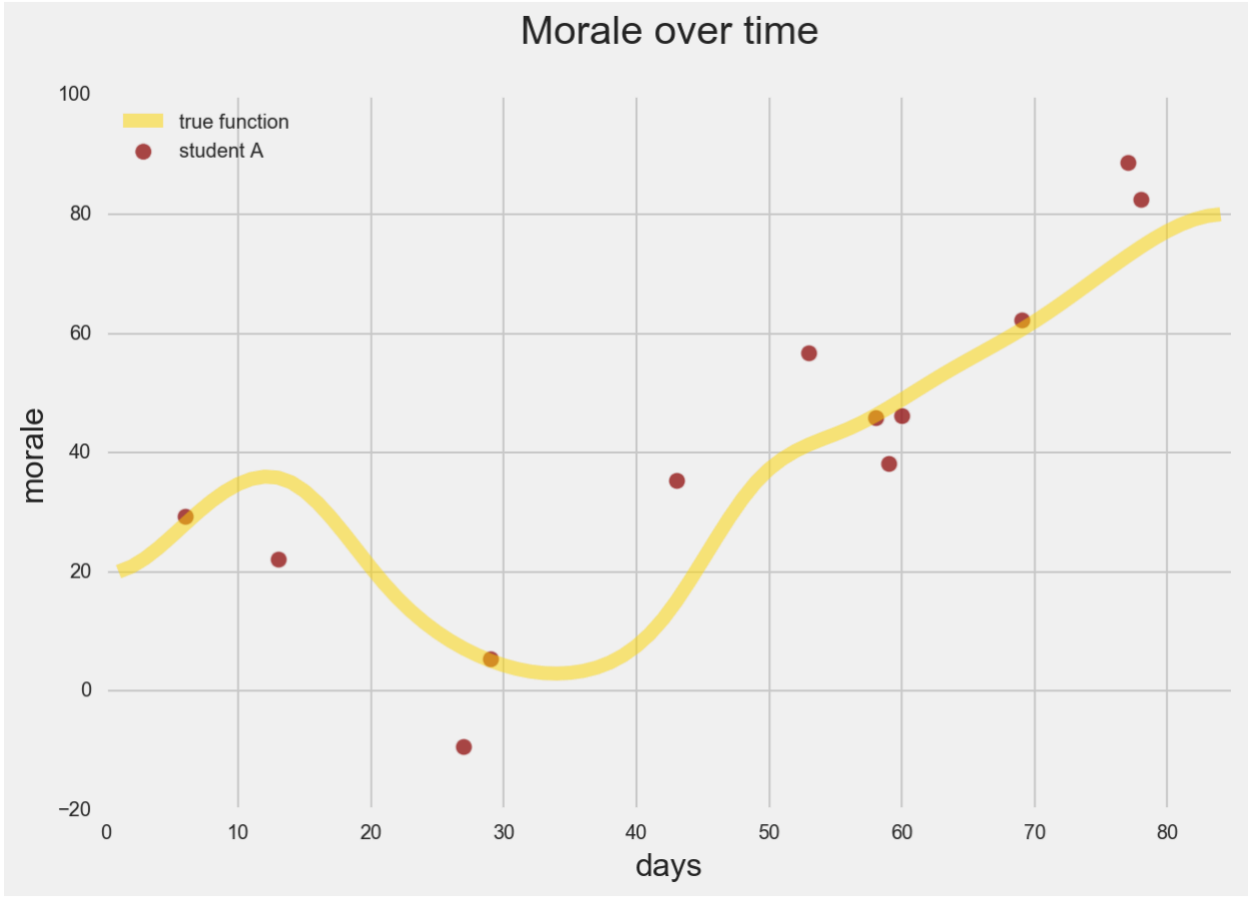
Types of Errors

- Example: rating the class
- **Model Error:**
- 1. Irreduceable error
  - If I hit a 4, but it registers a 3, nothing can be done about this type of error
- 2. Variance
- 3. Bias^2
- Will seek to minimize both

Sample – A perfect “true” curve, and single dataset

True function
Yellow,

Red dots are the training data

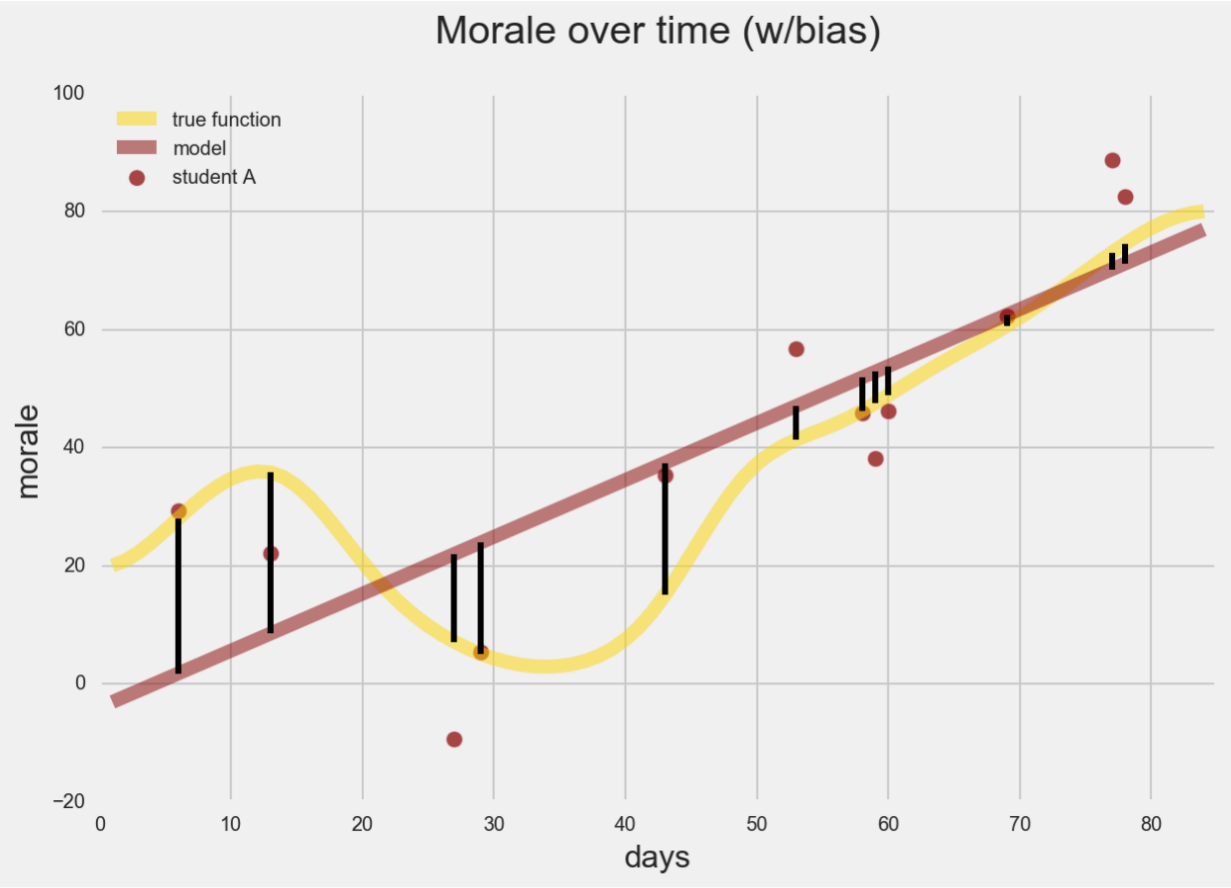


Sample: What if we overlayed a linear model? How off is it?

Bias (many different models) average pred. - y

- Bias^2 = (Expected value – actual)^2
  - How far off am I in this model from the true function
  - Similar to residuals but
  - Each student will have a different model (taylored)
  - The mean value of those values will be used against the actual
  - NOT a generic group of all students
  - In general, you cannot calculate the bias because we don't have the true function

What's your bias from the truth?

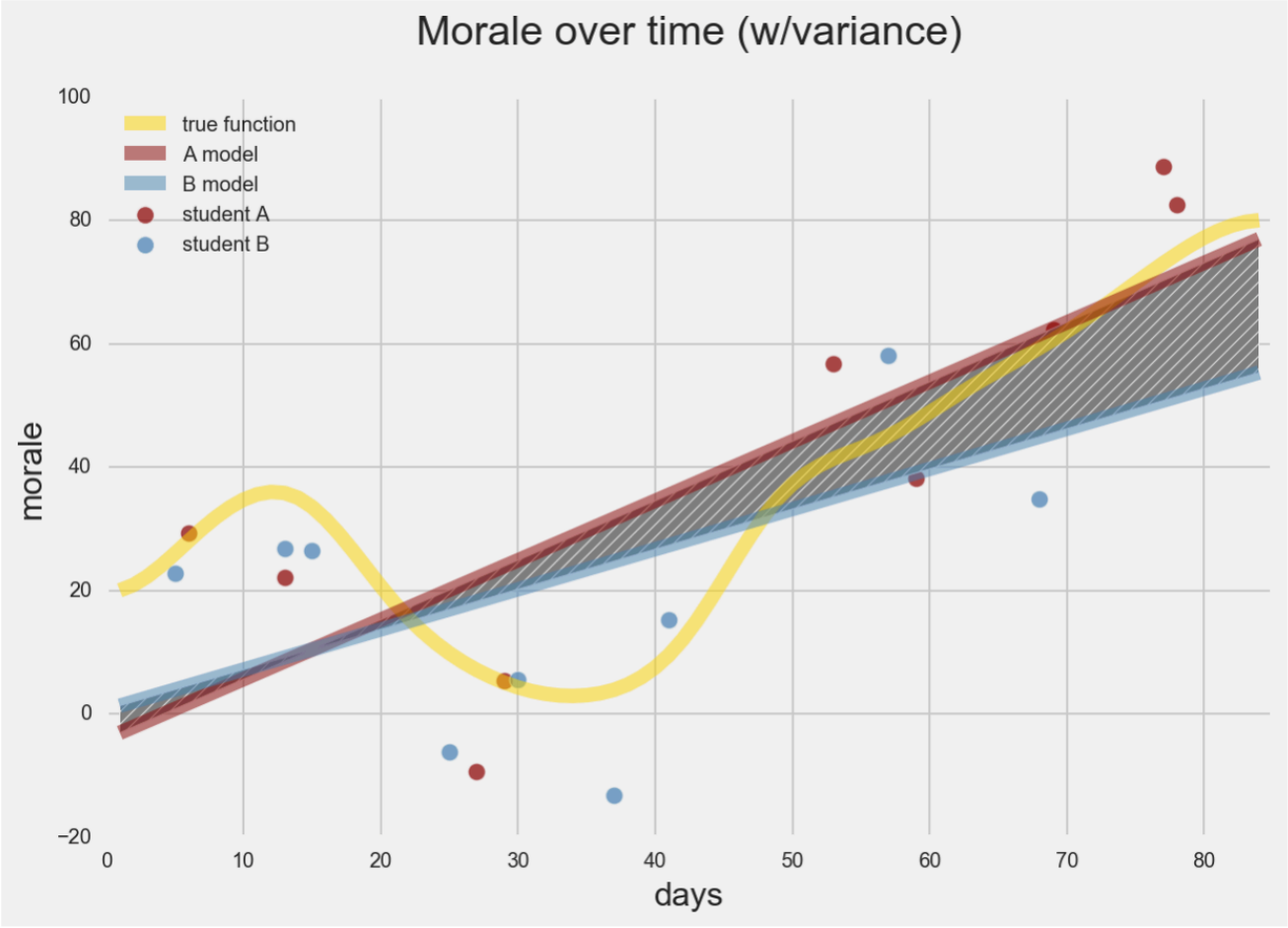


Variances – how consistent are my predictions

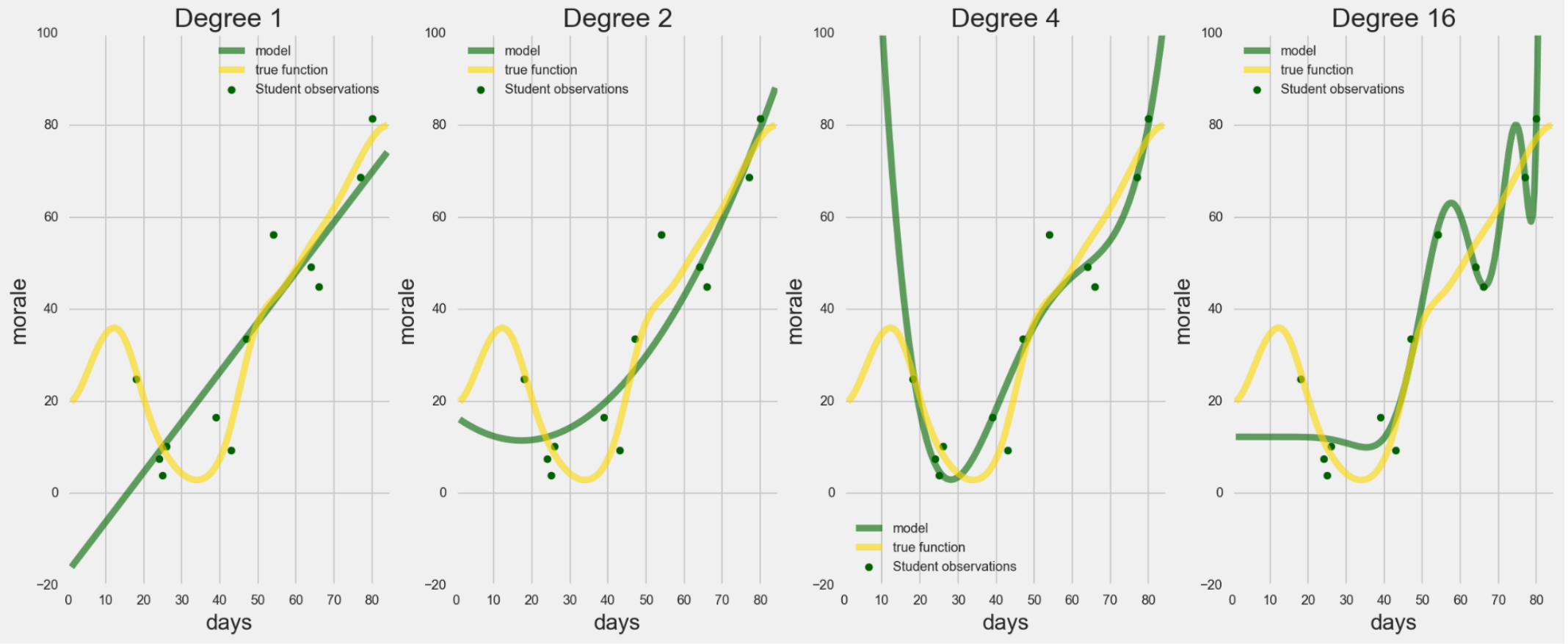
- Variance = (mean predictions– each set of predictions)^2
  - Between your different datasets, how much did your predictions vary from the mean predicion?
  - High-variance = overfitting

Whats your variance of your predictions?

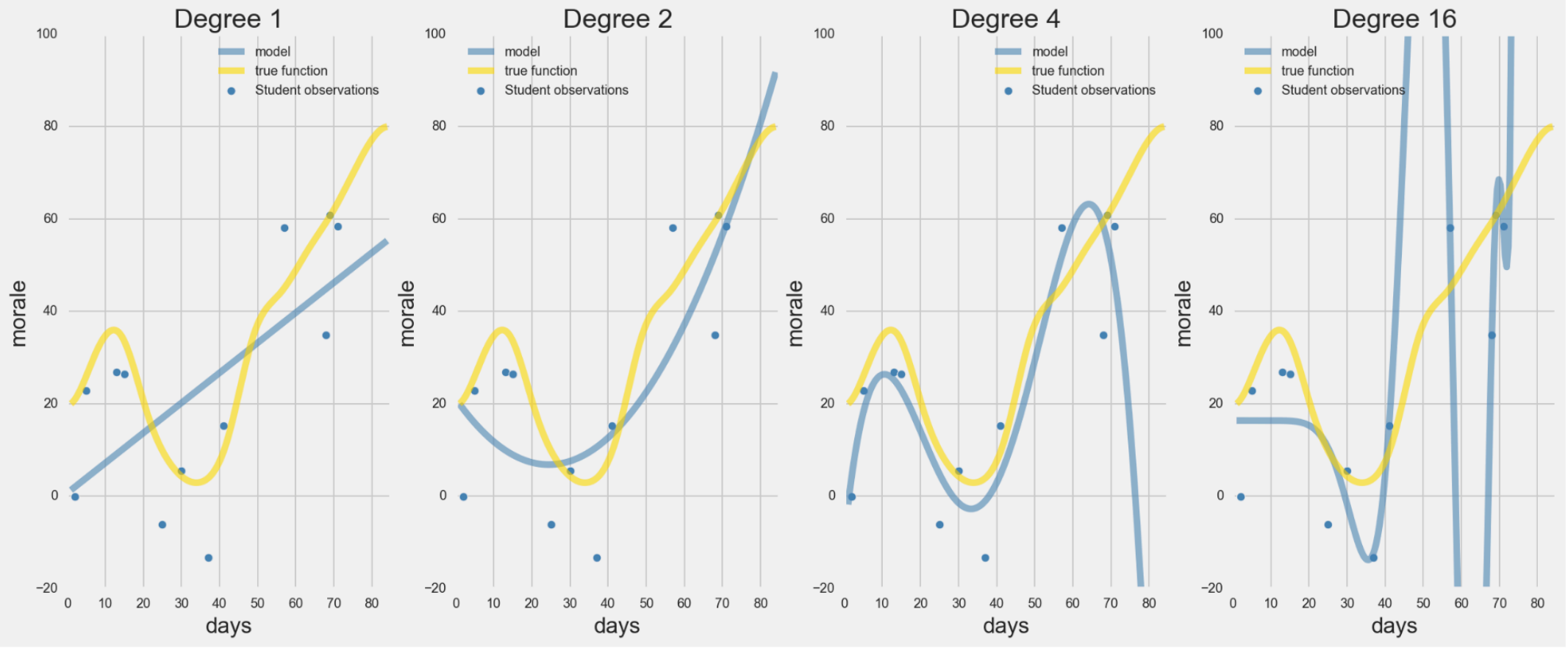
Whats your variance of your predictions?



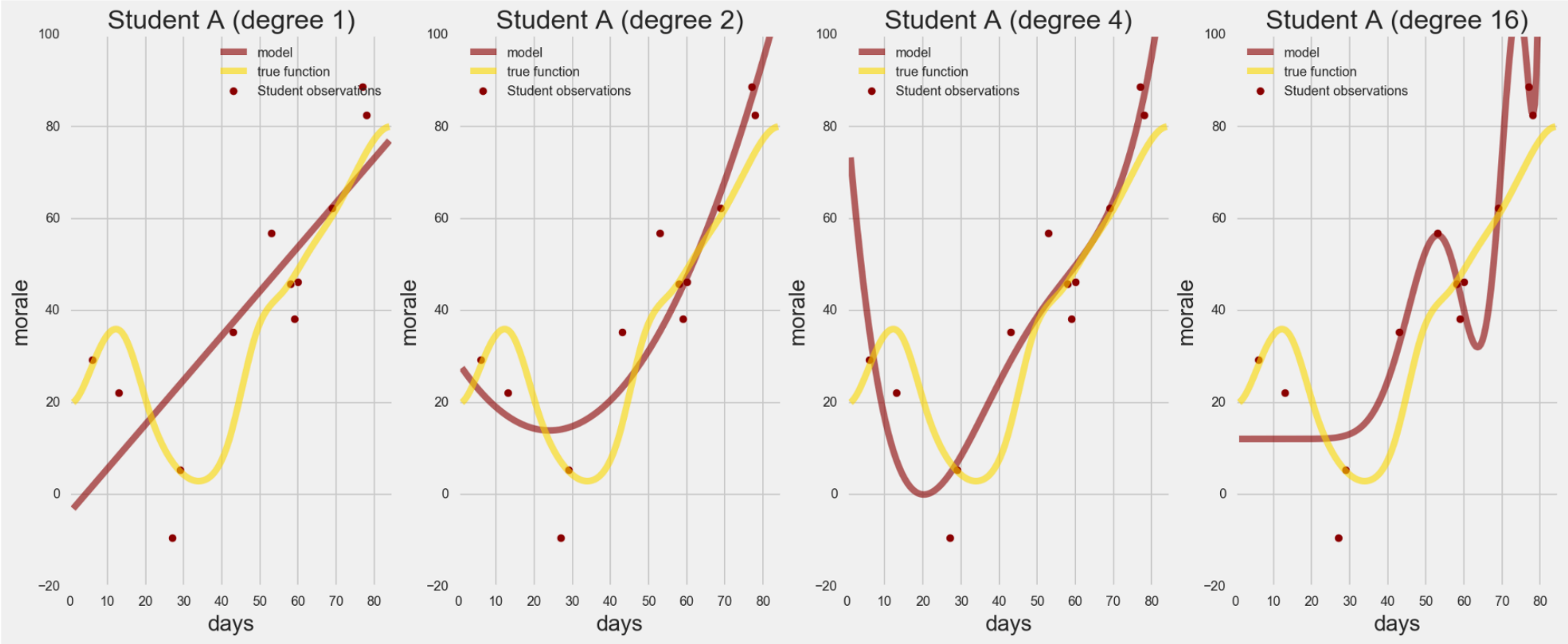
Increasing orders of approximation x --> x^5
May have good fits…



… but will create strange results when used on test data (outside of training set)



… again


Student A (degree 1)
Student A (degree 2)
Student A (degree 4)
Student A (degree 16)
model
true function
Student observations
morale
days

Day 14: Grid Search

How do you reduce your error of your model?

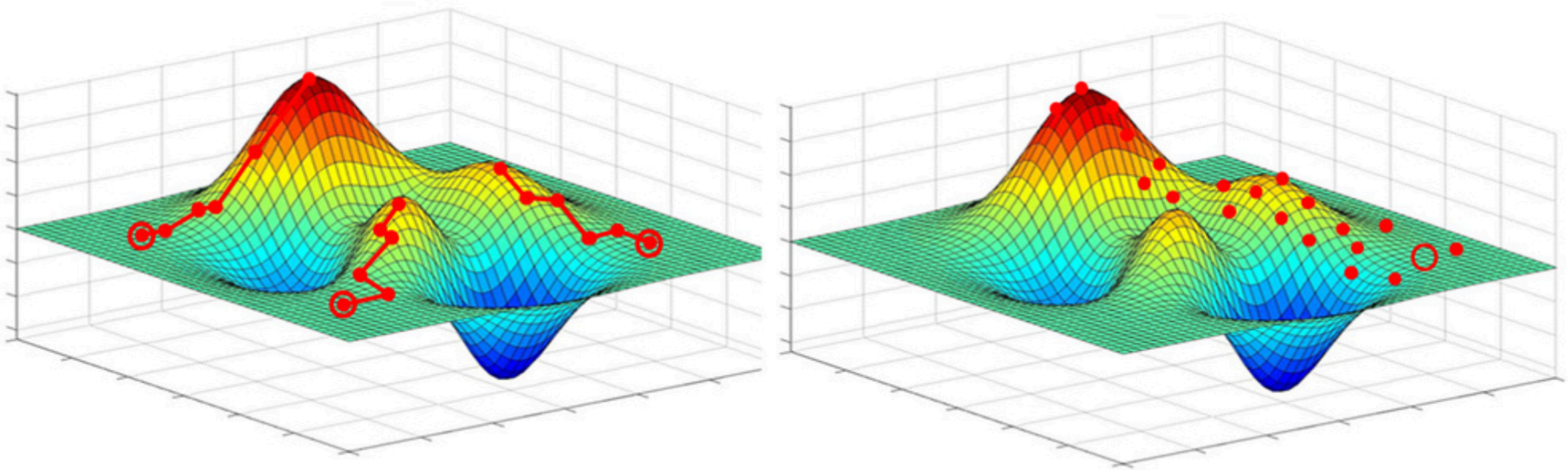
Recap: EDA – Exploratory Data Analysis

- When doing exploratory analysis and starting to think about model selection, we have a few good starting points.
- **Look at your data through subsetting and summary statistics**
  - Looking at coeficient matrices
  - Selecting features (variables) to use in our models
  - Considering parameters that might work, in a broad sense
  - Validation strategy
- A correlation matrix is used to investigate the **dependence between multiple variables at the same time**. The result is a table containing the correlation coefficients between each variable and the others. This is ideal for feature selection when deciding which features to use in a predictive model.

Grid Search

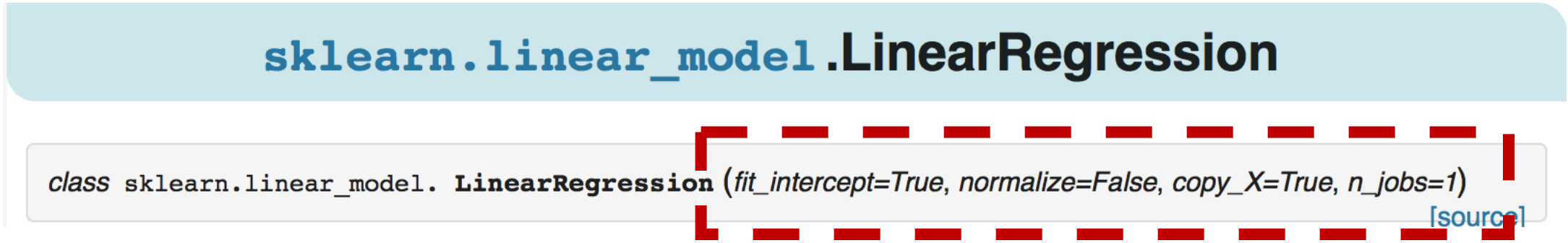
GridSearch

Week 3 | Lesson 4.3



As we learn more models

- They will get to be more complex
  - And they will have many options or **parameters**
  - And many ways of tuning their performance
- Example for Linear Regression:



Intro to Gridsearch (10 mins)

- What is "gridsearch"?
- **Gridsearch** is the process of searching for the optimal set of tuning parameters for a model.
- **Gridsearch** is a scikit-learn method. We use **Gridsearch** for searching across values of parameters, in combination with models, also using cross-validation to evaluate the effect to find the **best model**. It's called gridsearch because the idea is that there is a "grid" of parameters that are iteratively searched.

As an example:

- For linear regression,
- There are two options:
  - Fit_intercept = True
  - Normalize = False

	Fit Intercept True	Fit Intercept False
Normalize True	True , True Case 1	True, False Case 2
Normalize False	False, True Case 3	False, False Case 4

sklearn.linear_model.LinearRegression

class sklearn.linear_model. **LinearRegression** (*fit_intercept=True, normalize=False, copy_X=True, n_jobs=1*)

[source]

A More Sophisticated Example¶

- We haven't yet learned the intuition behind K-Nearest Neighbors, it's going to be introduced next week. So try to curb your curiousity for a moment about how this model works, and let's just talk about workflow and the programming mechanics for this example.
- Since **Gridsearch** is a method with a broad range of utility beyond regression problems, it's great for testing a combination of parameters on models that offer a broader range of **hyperparamters**.
- Lingo guide: **hyperparameter** is used interchangably with **parameter**, which is used with a given model. We are referring to the parameters we pass to our models to control their behavior.

What is Grid Search Doing?

What is GridSearch doing?

```
from sklearn import neighbors

# Search - 1
neighbors.KNeighborsClassifier(
    n_neighbors   =  1,
    weights       =  "uniform",
    algorithm     =  "ball_tree",
    leaf_size     =  30,
    etc...
)
# Search - 2
neighbors.KNeighborsClassifier(
    n_neighbors   =  2,
    weights       =  "uniform",
    algorithm     =  "ball_tree",
    leaf_size     =  30,
    etc...
)
# Search - 3
neighbors.KNeighborsClassifier(
    n_neighbors   =  3,
    weights       =  "uniform",
    algorithm     =  "ball_tree",
    leaf_size     =  30,
    etc...
)

...
```

Implementing GridsearchCV

- By default the cv parameter is 3. You can set this as high as you want! Keep in mind how it works and why it's used. If this is a mystery, please review our material on K-Folds validation.

In [25]:
```
# Load gridsearch, libraries, test data
from sklearn import grid_search, datasets
from sklearn.linear_model import LinearRegression
import pandas as pd, patsy

columns =  "age sex bmi map tc ldl hdl tch ltg glu".split()

data    =  datasets.load_diabetes()
df      =  pd.DataFrame(data.data, columns=columns)
df['target'] = data.target

# Setup patsy design matrix
y, X    =  patsy.dmatrices("target ~ age + sex + bmi + map + tc + ldl + glu", data=df, return_type="dataframe")
```

Setup GridSearchCV Parameters

In [26]:
```
# Setup our GridSearch Parmaters
search_parameters = {
    'fit_intercept':  [True, False],
    'normalize':      [False, True]
}

# Intialize a blank model object
lm = LinearRegression()

# Initialize gridsearch
estimator = grid_search.GridSearchCV(lm, search_parameters, cv=5)

# Fit some data!
results = estimator.fit(X, y)
```

Estimator result

- **Property** **Use**
- **results.param_grid** Displays parameters used
- **results.best_score_** Best score achieved
- **results.best_estimator_** Reference to model with best score. Is usable / callable. **Returns a model**
- **results.best_params_** The parameters that have been found to perform with the best score.
- **results.grid_scores_** Display score attributes with cooresponding parameters

Day 14: RECAP

How do you reduce your error of your model?

Linear Modeling Recap