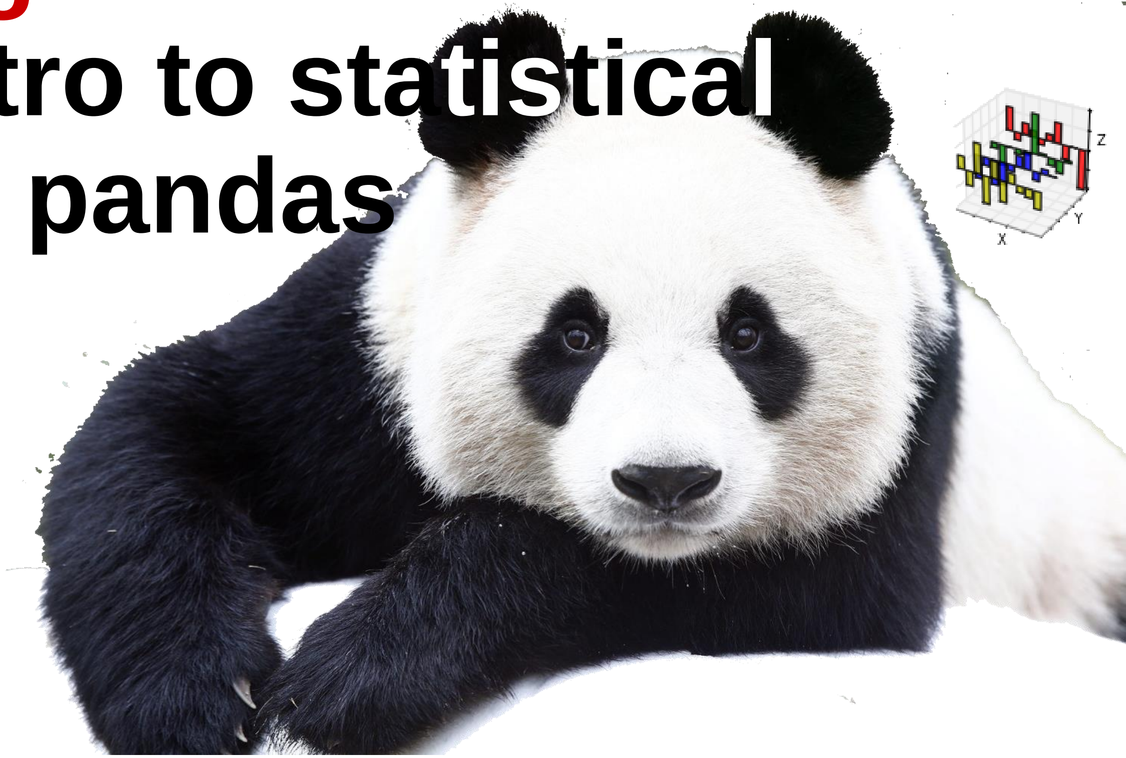
Northeastern University

INFO 6105
Data Sci Eng Mth & Tools
Lecture 2 Intro to statistical modelling & pandas

16 January 2019



pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Introduction to pandas

- Higher level data structures and functions than `NumPy`
- Primary object in Python: `list` (container object)
- Primary data object in NumPy: `array` (high performance row-structured matrix)
- Primary object in pandas: `DataFrame` (tabular column-oriented data structure), `Series` (1D labeled array)
  - High performance (built on top of NumPy)
  - Data manipulation capabilities of spreadsheets & relational dbs
- Author *Wes McKinney*, 2008, as econometric tool
- `DataFrame` named after R's `data.frame` object
- `pandas` == `pan`el `da`ta (econometrics for multi-d structured data sets), also play on `P`ython `da`ta `an`alysis
- `pandas` makes Python `R`-equivalent ☺
  - http://pandas.pydata.org/pandas-docs/stable/comparison_with_r.html

- Community lives @ https://pydata.org/
  - Products: https://pydata.org/downloads.html

Other libraries

- **`matplotlib` (graphing, http://matplotlib.org)**
  - **`%matplotlib` sets up integration with the library**
    - **To create multiple plot windows without interfering with the console session**
    - **In Jupyter notebook, use `%matplotlib inline`**
- **`seaborn` (library for data viz, https://seaborn.pydata.org/)**
  - Built on top of **`matplotlib`** and tightly integrated with the **`PyData`** stack, including support for **`numpy`** and **`pandas`** data structures and statistical routines from **`scipy`** and **`statsmodels`**
- **`statsmodels` (statistics, http://statsmodels.org)**
  - Python module that allows users to explore data, estimate statistical models, and perform statistical tests
    - **http://www.statsmodels.org/stable/index.html**

Installing packages in Anaconda environment

- `conda install packagename`
- `conda update packagename`
- `pip install packagename`
- `pip install –upgrade packagename`

pip

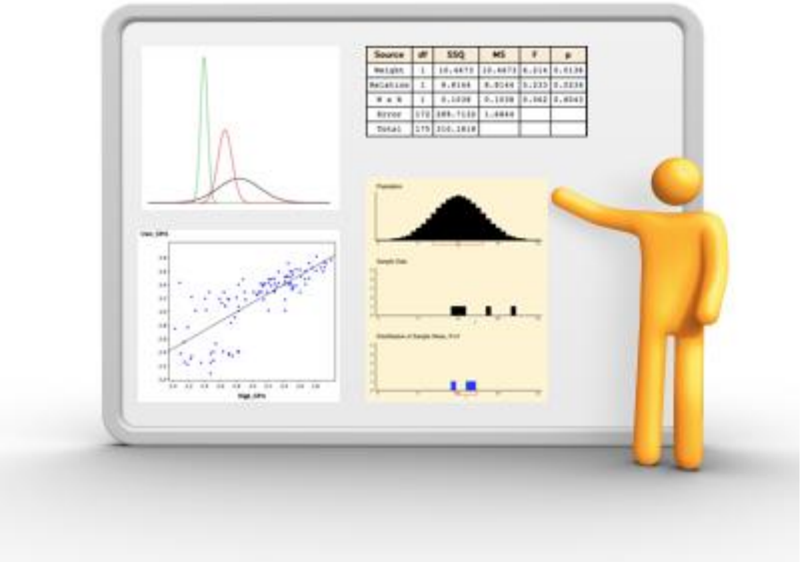
- Python packages only
- Compiles everything from source
- Now installs binary wheels too, if they are available

conda

- **Python agnostic**
  - **Main focus of existing packages are for Python, and indeed conda itself is written in Python, but you can also have conda packages for C libraries, or R packages, or anything**
- **Installs binaries**
  - **There is a tool called conda build that builds packages from source, but conda install itself installs things from already built conda packages**
- **Conda is the package manager of Anaconda, the Python distribution provided by Continuum Analytics**
  - **But it can be used outside of Anaconda too**
- **Conda is a packaging tool and installer that aims to do more than what pip does:**
  - **Handle library dependencies outside of the Python packages as well as the Python packages themselves**
  - **Conda also creates a virtual environment, like `virtualenv`**

Installing from Jupyter notebook or qt console

- **Install *binaries*: `!conda install pandas.datareader`**
- **Install *from source*: `!pip install pandas.datareader`**
- **Better: install with Anaconda command prompt (Windows) or bash shell (Mac)**
- **To find out about available packages, go to the anaconda Cloud (http://anaconda.org) and you can get the install command from there**



Part 1

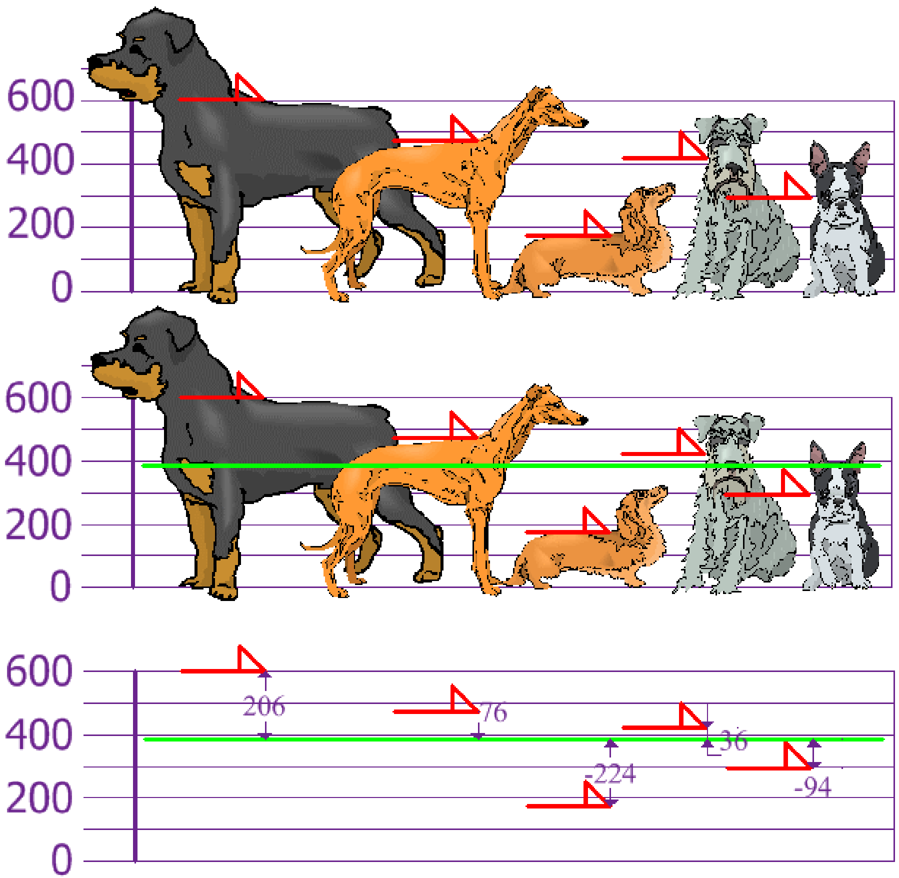
STATISTICS 101

Variance, Covariance

- **Consider two sets of measurements with zero means:**
  - **$A = \{a_1, a_2, . . . , a_n\}$, $B = \{b_1, b_2, . . . , b_n\}$**
- ***Variance*** **is sum of the squared distances of each term of one series, A *or* B, in the distribution from the mean (μ), divided by the number of terms in the distribution $\sigma^2_A = 1/n \Sigma_i a^2_i$, $\sigma^2_B = 1/n \Sigma_i b^2_i$ (if $\mu = 0$)**
- ***Covariance*** ***between*** **A and B is a straightforward generalization:**
  - **$\sigma^2_{AB} = 1/n \Sigma_i a_i b_i$ (if $\mu = 0$)**

Understanding Variance

- **Dog heights' *mean* and *variance*:**

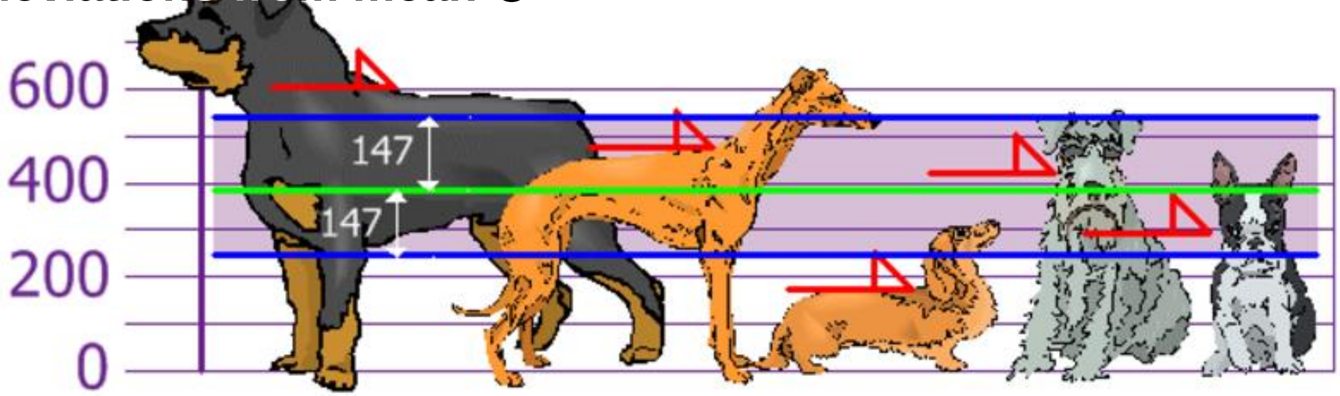


$$\text{Mean} = \frac{600 + 470 + 170 + 430 + 300}{5} = \frac{1970}{5} = 394$$

$$\begin{aligned}\text{Variance: } \sigma^2 &= \frac{206^2 + 76^2 + (-224)^2 + 36^2 + (-94)^2}{5} \\ &= \frac{42,436 + 5,776 + 50,176 + 1,296 + 8,836}{5} \\ &= \frac{108,520}{5} = 21,704\end{aligned}$$

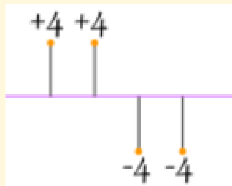
Standard Deviation

- *Standard Deviation* is just the square root of *Variance*:
  - $\sigma = \sqrt{21{,}704}$
    $= 147.32...$
    $= 147$ (to the nearest mm)
- Now we can show which heights are *within* one Standard Deviation (147mm) of the Mean:
  - Using the Standard Deviation we have a "standard" way of knowing what is normal, and what is extra large or extra small
  - Rottweilers are tall dogs. And Dachshunds are short because 2 std deviations from mean ☺



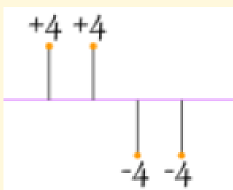
Why the <u>square</u> differences?

If we just add up the differences from the mean ... the negatives cancel the positives:



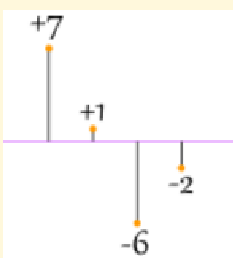
$$\frac{4 + 4 - 4 - 4}{4} = \boxed{0}$$

So that won't work. How about we use [absolute values]?



$$\frac{|4| + |4| + |-4| + |-4|}{4} = \frac{4 + 4 + 4 + 4}{4} = \boxed{4}$$

That looks good (and is the [Mean Deviation]), but what about this case:



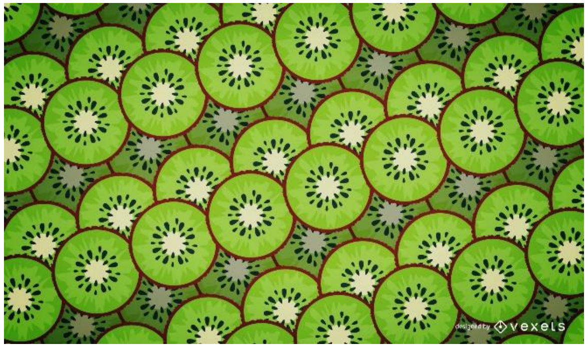
$$\frac{|7| + |1| + |-6| + |-2|}{4} = \frac{7 + 1 + 6 + 2}{4} = \boxed{4}$$

Oh No! It also gives a value of 4, Even though the differences are more spread out.

http://www.mathsisfun.com/data/standard-deviation.html

Understanding Covariance

- `Covariance` is a measure of ***how much two random variables vary together***
- Similar to `variance`, but whereas variance tells you how a *single* variable varies, covariance tells you how two variables vary together
- `Covariance` measures the degree of the *linear* relationship between two variables
  - A large value indicates high redundancy (similar data)
  - A small value indicates low redundancy (dissimilar data)
- Statistics is all about removing redundancy, because once removed, the central pattern appears!



Interpretation issues & correlation coefficient

- A *large covariance* can mean a *strong relationship between variables*, however, you can't compare variances over data sets with different scales (like pounds and inches)
  - A weak covariance in one data set may be a strong one in a different data set with different scales
- The main problem with interpretation is that the wide range of results that it takes on makes it hard to interpret
  - For example, your data set could return a value of 3 or 3,000
  - This wide range of values is caused by a simple fact: The larger the X and Y values, the larger the covariance
  - A value of 300 tells us that the variables are correlated, but that number doesn't tell us exactly how *strong* that relationship is
- The problem can be fixed by dividing the covariance by the standard deviation to get the `correlation coefficient`: $\mathrm{Corr}(X,Y) = \mathrm{Cov}(X,Y) / \sigma_X \sigma_Y$

Correlation Coefficient Advantages

- The Correlation Coefficient has several advantages over covariance for determining strengths of relationships:
  - Covariance can take on practically any number while a correlation is limited: -1 to +1
  - Because of it’s numerical limitations, correlation is more useful for determining how strong the relationship is between the two variables
  - Correlation does not have units. Covariance always has units
  - Correlation isn’t affected by changes in the center (i.e. mean) or scale of the variables

Covariance Matrix

- A *Covariance matrix* includes *variance* and *covariance* information of multiple datasets
- Let's move to m distributions: $x_1, x_2, .., x_m$ (m datasets, each of n dimensions)

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

e.g. age, salary, debt, education level for 100,000 college students to figure out political party affiliation. Here n = 100,00 and m = 4

  - Each *row* of X corresponds to all 4 measurements of a particular person (dataset)
  - Each column of X corresponds to a set of measurements for one particular feature
  - Covariance matrix: $C_x = 1/(n-1)\, XX^T$
  - The ijth element of C_X is the dot product between the vector of the ith person (ith dataset) with the vector of the jth person (jth dataset):
    $x_{i1}x_{j1} + x_{i2}x_{j2} + \ldots + x_{im}x_{jm}$
  - Thus, C_X captures the correlations between *all* possible (types) *pairs* of persons
    - Correlation values reflect the noise and redundancy in our measurements
    - Diagonal terms: Large (small) values correspond to interesting dynamics per feature (noise)
      $x^2_{i1} + x^2_{i2} + \ldots + x^2_{im}$
    - Off-diagonal terms: Large (small) values correspond to high (low) redundancy between features

Variables

- **The decision as to which variable in a data set is modeled as the dependent variable and which are modeled as the independent variables may be based on a presumption that the value of one of the variables is *caused by*, or *directly influenced* by the other variables**
- **Independent variables are also called *regressors*, *exogenous variables*, *explanatory variables*, *covariates*, *input variables*, and *predictor variables***
- **That decision *is the most important one in regression analysis***

Regression Analysis

- A statistical process for estimating relationships among variables
- It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables (or 'predictors')
- More specifically, regression analysis helps one understand how the typical value of the dependent variable (or 'criterion variable') changes when any one of the independent variables is varied, while the other independent variables are held fixed
- Regression analysis is widely used for prediction and forecasting, where its use has substantial overlap with the field of machine learning

Part 2

INTRODUCTION TO STATMODELS

Moral variables, *Guerry, France*

DataScience 101

```
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd
```

```
# load data
# https://cran.r-project.org/web/packages/HistData/HistData.pdf
```

Andre-Michel Guerry (1833) was the first to systematically collect and analyze social data on such things as *crime*, *literacy* and *suicide* with the view to determining social laws and the relations among these variables. The Guerry data frame comprises a collection of ’moral variables’ on the 86 departments of France around 1830. A few additional variables have been added from other sources

```
dat = sm.datasets.get_rdataset("Guerry", "HistData").data
dat.info()
dat.head()
dat.describe() #equivalent to R summary()
pd.DataFrame(dat, columns=['Literacy', 'Lottery’])
dat.Literacy.corr(dat.Lottery)
```

Data Science

- ```
  # Fit regression model (using natural log of one of the regressors)
  results = smf.ols('Lottery ~ Literacy + np.log(Pop1831)', data=dat).fit()
  ```
- ```
  # Inspect results
  print(results.summary())
  ```
- ```
  print('Predicted values: ', results.predict())
  ```
  ```

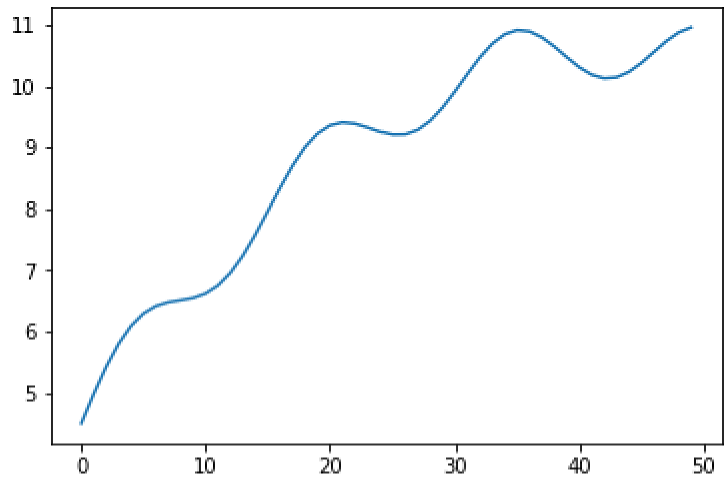
Artificial data

```
nsample = 50
sig = 0.5
x = np.linspace(0, 20, nsample)
X = np.column_stack((x, np.sin(x), (x-5)**2, np.ones(nsample)))
beta = [0.5, 0.5, -0.02, 5.]
y_true = np.dot(X, beta); plt.plot(y_true)
y = y_true + sig * np.random.normal(size=nsample); plt.plot(y)
res = sm.OLS(y, X).fit()
print(res.summary())
print('Parameters: ', res.params)
print('Standard errors: ', res.bse)
print('Predicted values: ', res.predict())
plt.plot(res.predict())
```

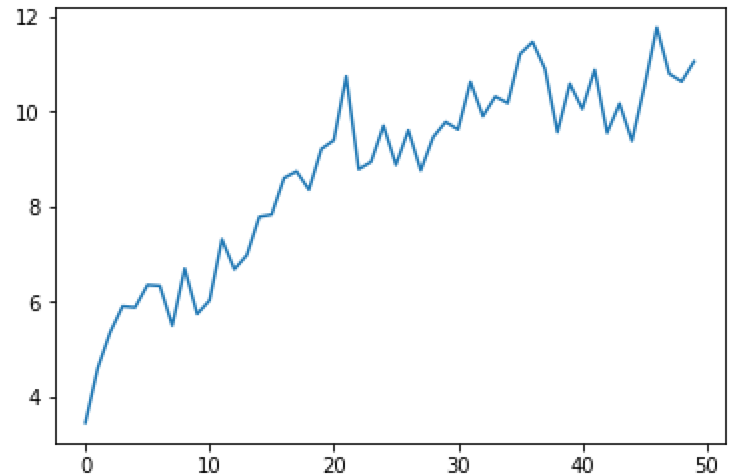
Plotting with subplots in matplotlib

- `import matplotlib.pylot as plt`
- `from statsmodels.sandbox.regression.predstd import wls_prediction_std`
- `prstd, iv_l, iv_u = wls_prediction_std(res)`
- `fig, ax = plt.subplots(figsize=(6,4))`
- `ax.plot(x, y, 'o', label="data")`
- `ax.plot(x, y_true, 'b-', label="True")`
- `ax.plot(x, res.fittedvalues, 'r--.', label="OLS")`
- `ax.plot(x, iv_u, 'r--')`
- `ax.plot(x, iv_l, 'r--')`
- `ax.legend(loc='best’);`
- `fig #to display the plot`

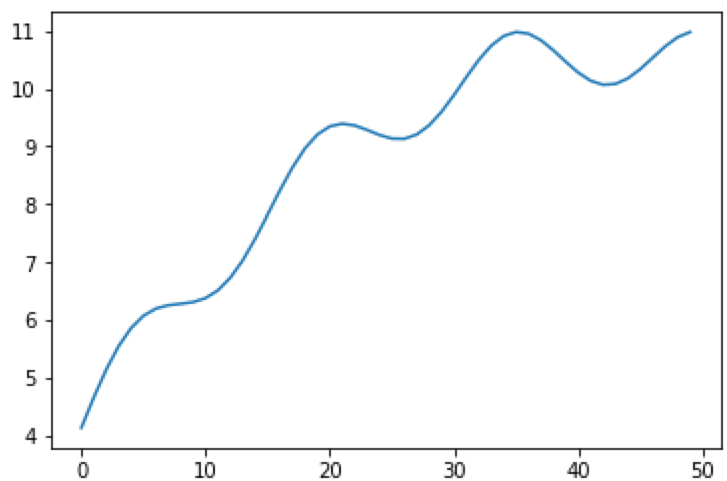
```
In [77]: plt.plot(y_true)
Out[77]: [<matplotlib.lines.Line2D at 0x2d01e38c518>]
```

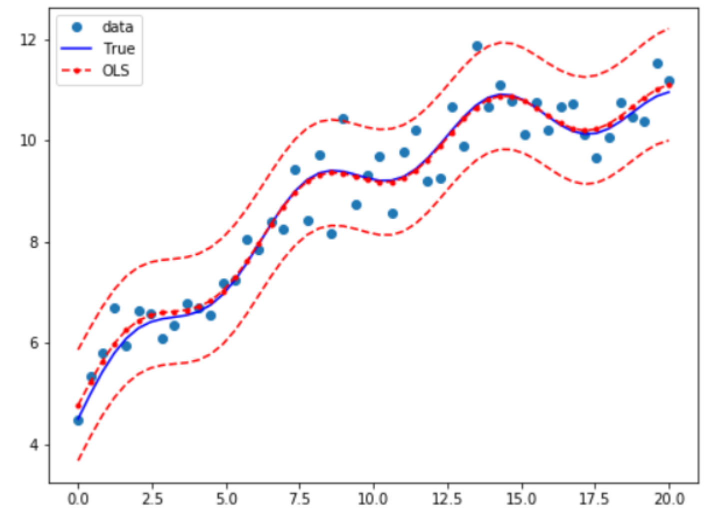



```
In [79]: plt.plot(y)
Out[79]: [<matplotlib.lines.Line2D at 0x2d01e3ed588>]
```




```
In [84]: plt.plot(res.predict())
Out[84]: [<matplotlib.lines.Line2D at 0x2d01e8c19b0>]
```





Part 3

ESSENTIAL PANDAS

`pandas` `Series` is a fixed-length ordered dict

- **1D array-like object containing a sequence of values (similar types to NumPy) and an associated data label called its `index`**
- `Import pandas as pd`
- `obj = pd.Series([10, 20, 30, 40])` `#did not specify index`
- `obj, obj.values, obj.index`
- `obj2 = pd.Series([10, 20, 30, 40]), index = ['a', 'b', 'c', 'd']`
- `obj2[obj2 > 20]`
- `obj * 2`
- `np.exp(obj2)`
- `sdata = {'Celtics': 3, 'Cavs': 1}`
- `obj3 = pd.Series(sdata)`
- `teams = ['Celtics', 'Cavs', 'Warriors']`
- `obj4 = pd.Series(sdata, index = teams)`
- `obj4.isnull()`

pandas DataFrame

- Rectangular table of data with ordered collection of columns, each of which can be of a different type
  - Think of it as a dictionary where each entry is set/list-valued
- `nba = {'east': {'Celtics', 'Knicks', 'Cavs', 'Heat'}, 'west': {'Warriors', 'Lakers', 'Rockets'} } #option1`
- `nba = {'east': ['Celtics', 'Knicks', 'Cavs', 'Heat'], 'west': ['Warriors', 'Lakers', 'Rockets'] } #option2`

```
{'east': ['Celtics', 'Knicks', 'Cavs', 'Heat'],
 'west': ['Warriors', 'Lakers', 'Rockets']}
```

- ```
  frame = pd.DataFrame(nba)  #error because dim mismatch
  frame = pd.DataFrame.from_dict(nba, orient='index')
  frame                                    #introduces None
  ```

	0	1	2	3
east	Celtics	Knicks	Cavs	Heat
west	Rockets	Warriors	Lakers	None

- ```
  frame[0] #column
  frame.loc['east'] #row
  ```

pandas DataFrame, explicit

- `my = pd.DataFrame(nba, columns = ['east', 'south'], index = ['Celtics', 'Lakers', 'Warriors', 'Bulls'])`
  `#introduces NaN for missing dimensions`
  `#no need for frame_dict!`
- `my`

	east	south
Celtics	{Celtics, Knicks, Cavs, Heat}	NaN
Lakers	{Celtics, Knicks, Cavs, Heat}	NaN
Warriors	{Celtics, Knicks, Cavs, Heat}	NaN
Bulls	{Celtics, Knicks, Cavs, Heat}	NaN

pandas DataFrame, outer & inner keys

- `nba2 = {'east':{'MA':'Celtics', 'NY':'Knicks', 'OH':'Cavs', 'FL':'Heat'}, 'west': {'CA-N':'Warriors', 'CA-S':'Lakers', 'TX':'Rockets'}} #outer keys as columns, inner keys as indexes`
- `nba2`

```
{'east': {'FL': 'Heat', 'MA': 'Celtics', 'NY': 'Knicks', 'OH': 'Cavs'},
 'west': {'CA': 'Warriors', 'CA-S': 'Lakers', 'TX': 'Rockets'}}
```

- `frame2 = pd.DataFrame(nba2)`

	east	west
CA	NaN	Warriors
CA-S	NaN	Lakers
FL	Heat	NaN
MA	Celtics	NaN
NY	Knicks	NaN
OH	Cavs	NaN
TX	NaN	Rockets

- `frame2.columns, frame2.index`
- `frame2.T`

	CA	CA-S	FL	MA	NY	OH	TX
east	NaN	NaN	Heat	Celtics	Knicks	Cavs	NaN
west	Warriors	Lakers	NaN	NaN	NaN	NaN	Rockets

reindex()

- ```
  obj = pd.Series([10, 20, 30, 40]), index = ['a', 'b', 'c', 'd'])
  obj
  ```
- obj2 = obj.reindex(`['b', 'a', 'd', 'z']`
- ```
  frame = pd.DataFrame(np.arrange(0,9).reshape((3,3)), index=['a', 'c', 'd'], columns=['MA', 'NY', 'CA'])
  frame
  ```
- ```
  frame2 = frame.reindex(['a', 'b', 'c', 'd'])
  frame2
  ```
- ```
  south=['FL', 'AL', 'TX']
  frame2.reindex{columns = state)
  frame2
  ```

Dropping, selection, filtering

- `import numpy as np`
- `data = pd.DataFrame(np.arange(9),reshape((3,3)), index=['MA', 'NY', 'TX'], columns=['Celts', 'Knicks', 'Spurs’])`
- `data`

	Celts	Knicks	Spurs
MA	0	1	2
NY	3	4	5
TX	6	7	8

- `data.drop('TX’)`

	Celts	Knicks	Spurs
MA	0	1	2
NY	3	4	5

- `data.drop(['Knicks', 'Spurs'], axis = 'columns’).T`

	MA	NY	TX
Celts	0	3	6

Function application

- `np.random,randn(4,3)`

```
array([[ 1.1738488 ,  0.6222295 , -0.99689746],
       [ 0.4713837 , -0.36254197,  0.54068353],
       [ 0.22319906, -1.57464014,  1.37068552],
       [-1.03648198, -1.34252123,  0.99177792]])
```

- `frame = pd.DataFrame(np.random.randn(4,3), columns = list('abc'), index = ['Celts', 'Cavs', 'Warriors', 'Rockets'])`

	a	b	c
Celts	0.633176	0.716141	-1.449136
Cavs	0.387074	0.604390	-1.833565
Warriors	0.028142	-2.645760	2.063051
Rockets	0.069785	-0.205026	0.510103

- `f = lambda x: x.max() - x.min()`
- `frame.apply(f)   #once per column`

```
a     0.605034
b     3.361900
c     3.896616
dtype: float64
```

- `frame.apply(f, axis = 'columns')   #once per row`

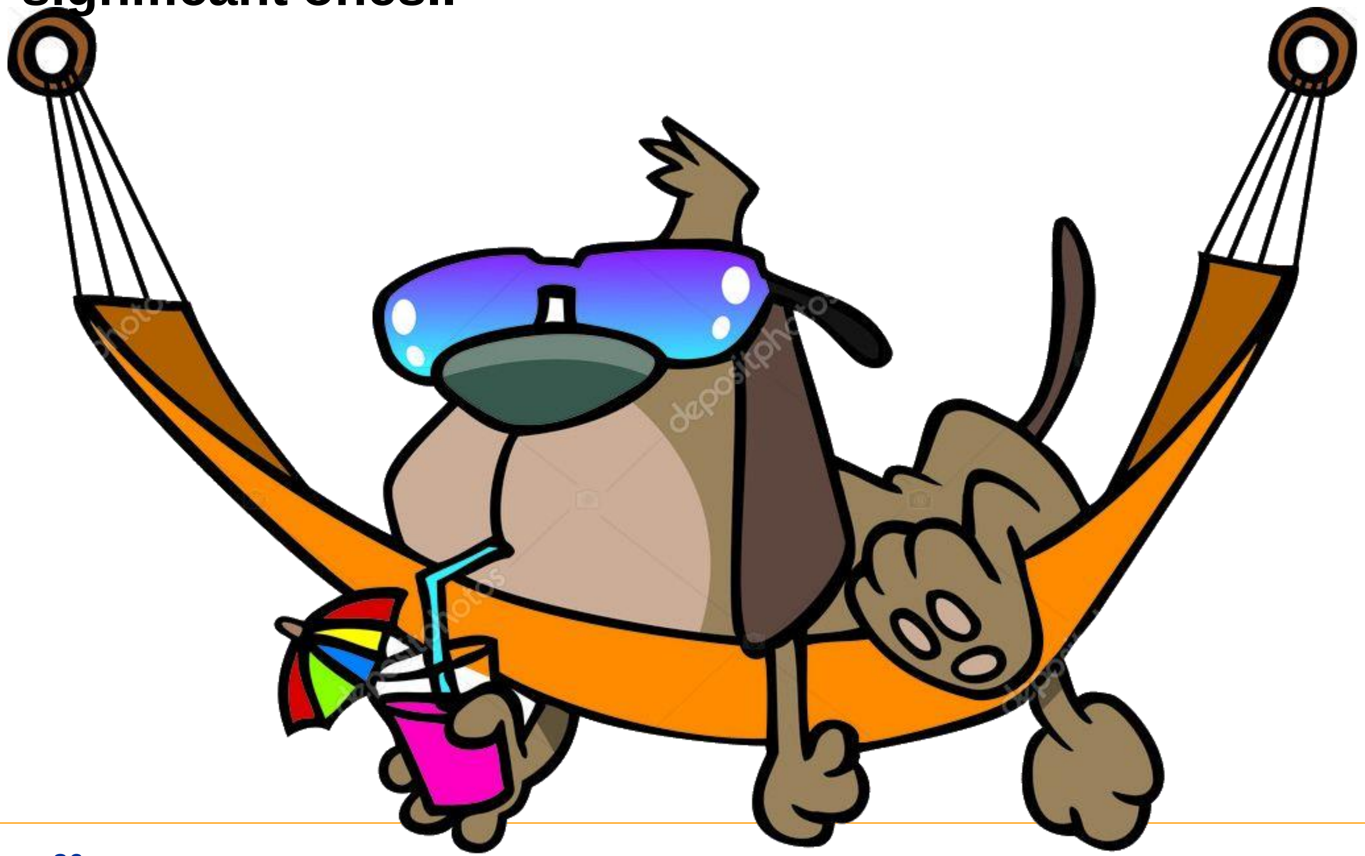
```
Celts       2.165277
Cavs        2.437955
Warriors    4.708811
Rockets     0.715129
dtype: float64
```

Descriptive statistics (oops.. YAHOO! )


```
frame.sum()  #column sums
frame.sum(axis='columns') #sums across columns
frame.describe()
conda install pandas.datareader #binary pckg not found
pip install pandas.datareader    #install from source
import pandas_datareader.data as web
all_data = {ticker: web.get_data_yahoo(ticker) for
ticker in ['AAPL', 'MSFT', 'GOOG', 'AMZN', 'FB'] }
price = pd.DataFrame({ticker: data['Adj Close']
for ticker, data in all_data.items()})
volume = pd.DataFrame({ticker: data['Volume'] for
ticker, data in all_data.items()})
returns = price.pct_change(); returns.tail()
returns['MSFT'].corr(returns['AAPL'])
returns['MSFT'].cov(returns['AAPL'])
returns.corr()
```

Key Takeaway

- `pandas` is like programmable excel spreadsheets, only better
- Also makes python and R equivalent in syntax
- Lots of other pandas APIs to learn, but these are the significant ones..



Homework

- Finish your `wordcloud-in-your-own-language` lab, leveraging `udpipe`
- Work on `pandas-exercises.ipynb` notebook, and do homework question (titanic dataset)


CLASS
DISMISSED
iStockphoto