GTK CYBER

GET ON THE FAST TRACK

Machine Learning for Security Professionals - Day 1

Module 1: Introduction & Data Preparation with Pandas

GTK CYBER
GET ON THE FAST TRACK

Expectations

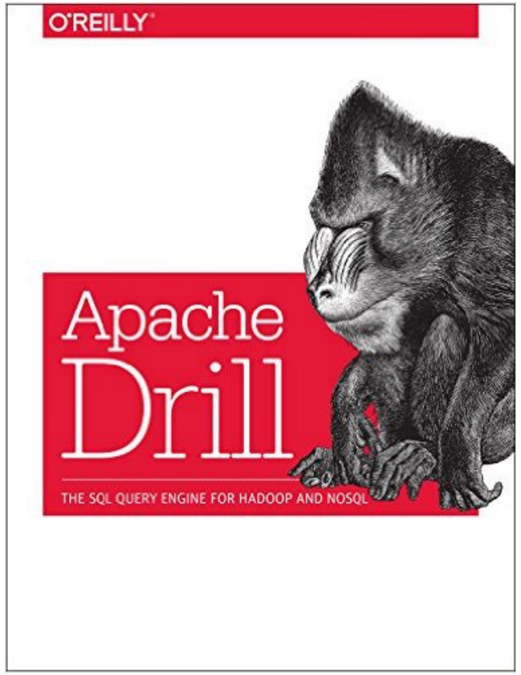
- Please participate and **ask questions.**
- Please follow along and **TRY OUT** the examples yourself during the class
- All the answers are in the slide decks or GitHub repository, but please try to complete the exercises **without looking at the answers**.
- Have fun!

Introduction

Who am I?

Charles Givre

- Committer to Apache Drill Project
- Senior Lead Data Scientist @ Booz Allen
- 5 Years @ CIA
- Working on Apache Drill Book
- Masters Degree in Middle Eastern Studies
- Undergraduate in Comp.Sci & Music



gtkcyber.com

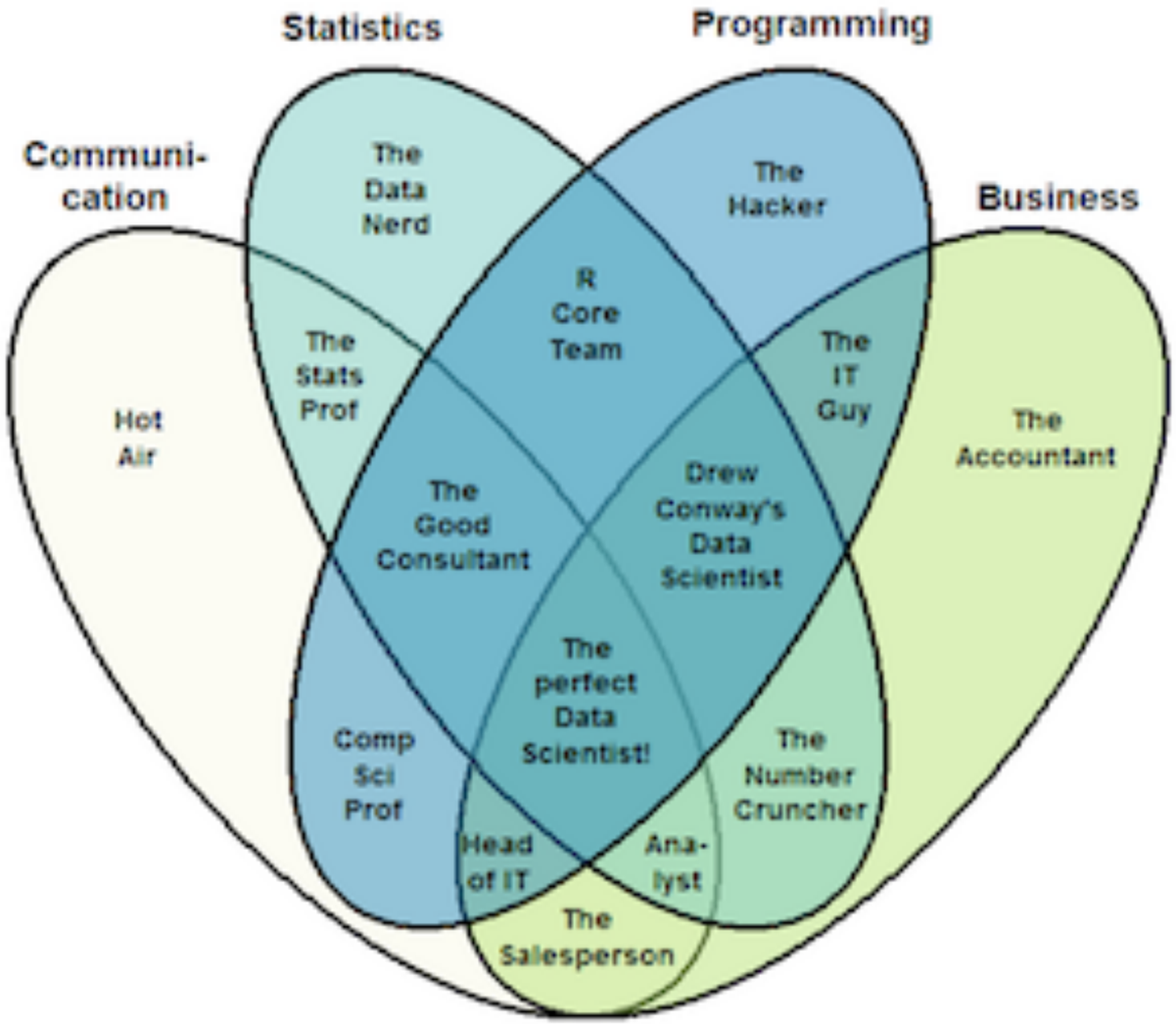
What is Data Science?

Data Science is the automated extraction of information from raw data.

Data Science is the art of **turning data into actions**. This is accomplished through the creation of data products, which provide actionable information without exposing decision makers to the underlying data or analytics

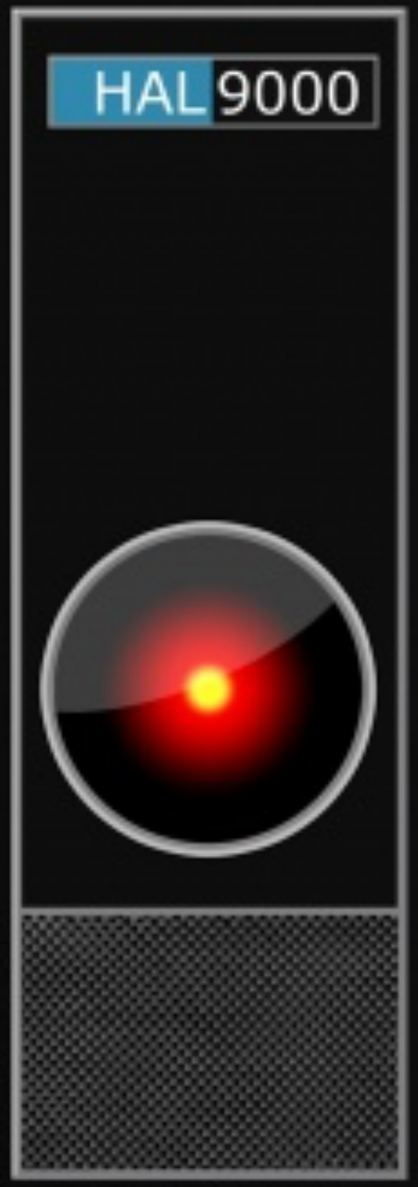
Booz Allen Hamilton, Field Guide to Data Science, Pg. 17

The Data Scientist Venn Diagram

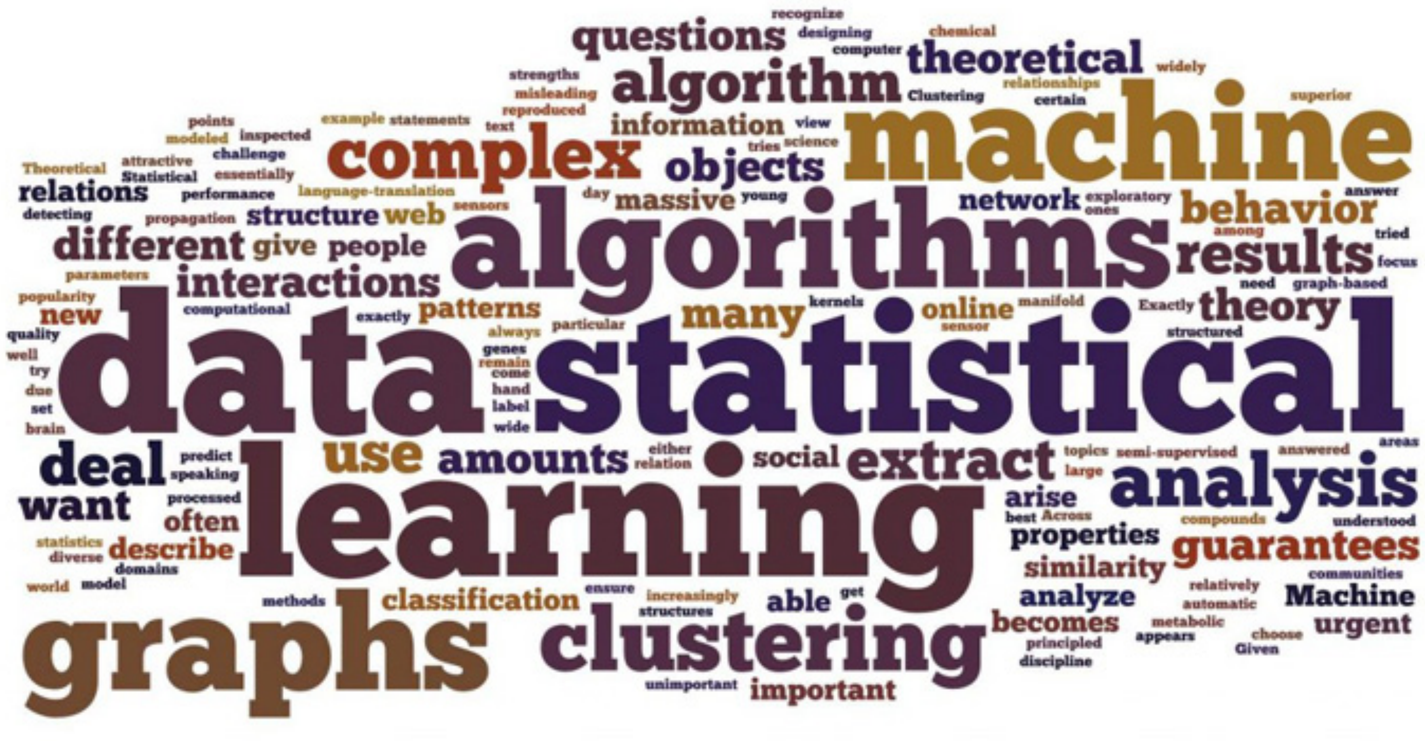


https://www.kdnuggets.com/2017/03/data-science-data-scientist-do.html

Machine Learning



What is
Machine Learning (ML)
Artificial Intelligence (AI)



“I have no idea. The latest buzzword? ”

–The guy in the back of the class

“Machine learning is the science of getting computers to **act without being explicitly programmed.** ”

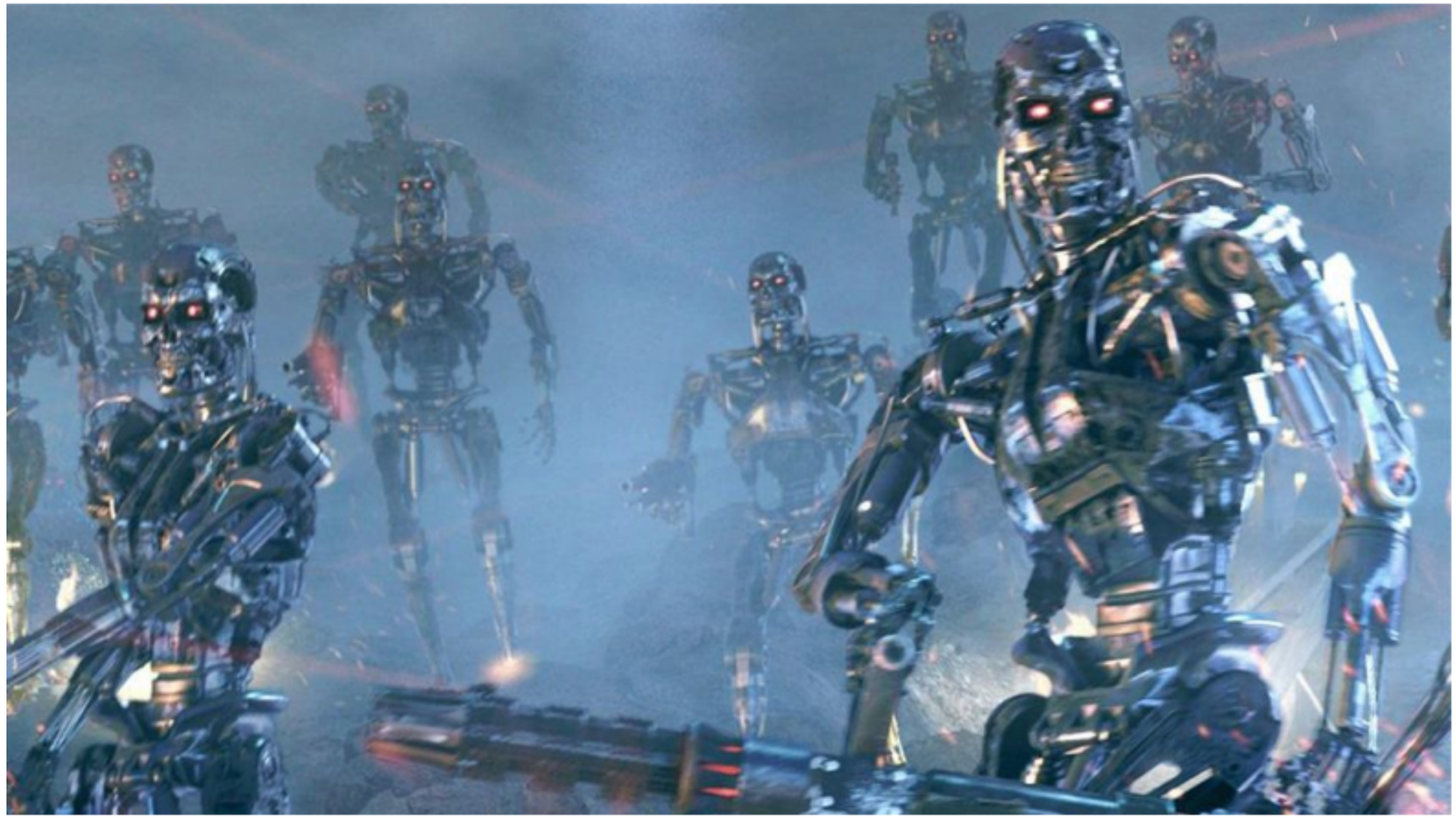
– *https://www.coursera.org/course/ml*

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

–Tom Mitchell, Carnegie Mellon University

"Machine learning explores the construction and study of algorithms that can learn from and **make predictions on data.** Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, **rather than following strictly static program instructions.**"

–*https://en.wikipedia.org/wiki/Machine_learning*

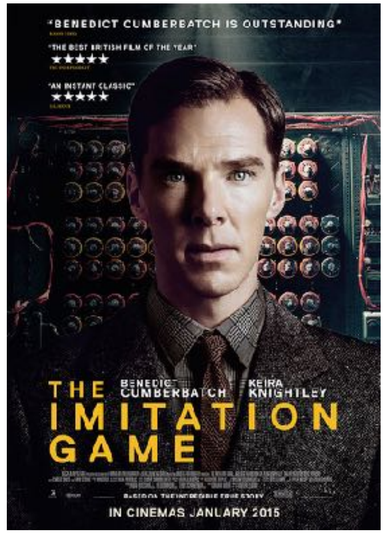


What it is not

History of AI and ML

- Mythical figure Pygmalion: Breath of life in a statue (ancient Greece)

- Turing Test 1950s by Alan Turing: Test of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from, that of a human.



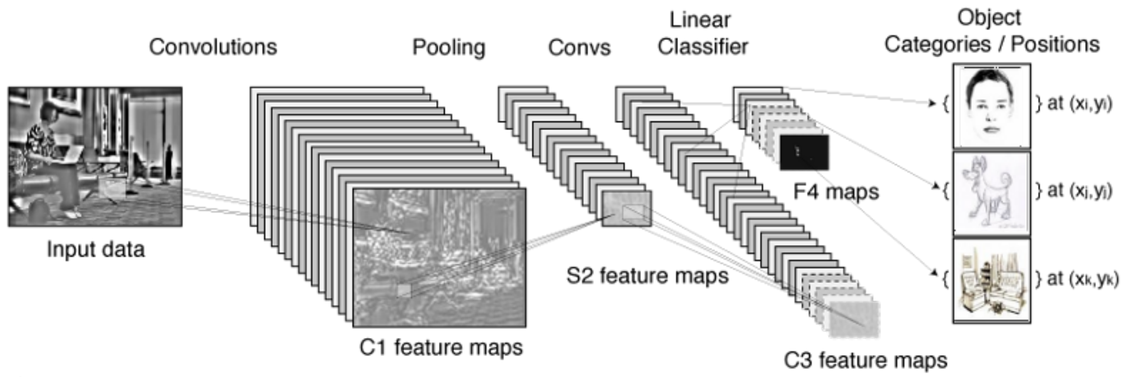
- Early AI: solve problems that are difficult for humans, but straightforward for computers (e.g. formal mathematical rules) IBM Deep Blue chess-playing system defeated world champion Garry Kasparov 1997.

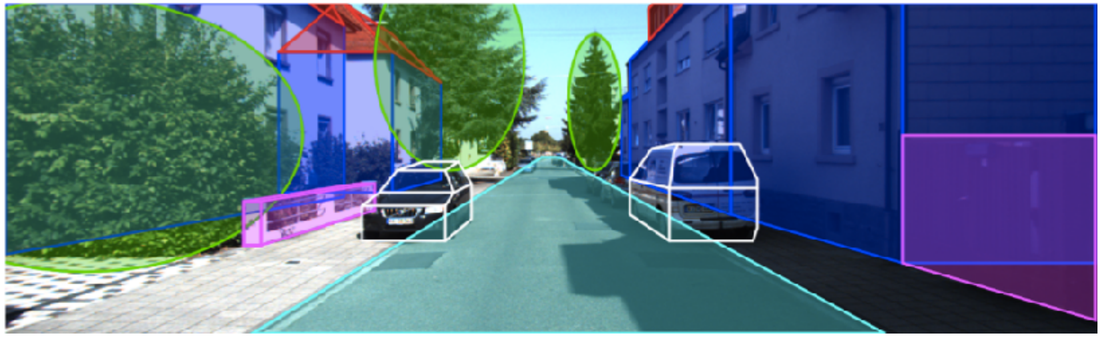
- Real Challenge: Intuitive, easy problems for humans. They are hard to describe formally (e.g recognizing faces, words).

http://www.deeplearningbook.org/

Path to Artificial Intelligence?

- Hard-coding knowledge failed (e.g. Cyc 1989).
- **Gathering knowledge from experience** eliminates the need to formally specify all knowledge the computer needs.
- AI needs to acquire own knowledge by **extracting pattern from raw data.**
- Success depends heavily on **representation of data, aka features.**
- How to choose features?
- Is Deep Learning the answer?



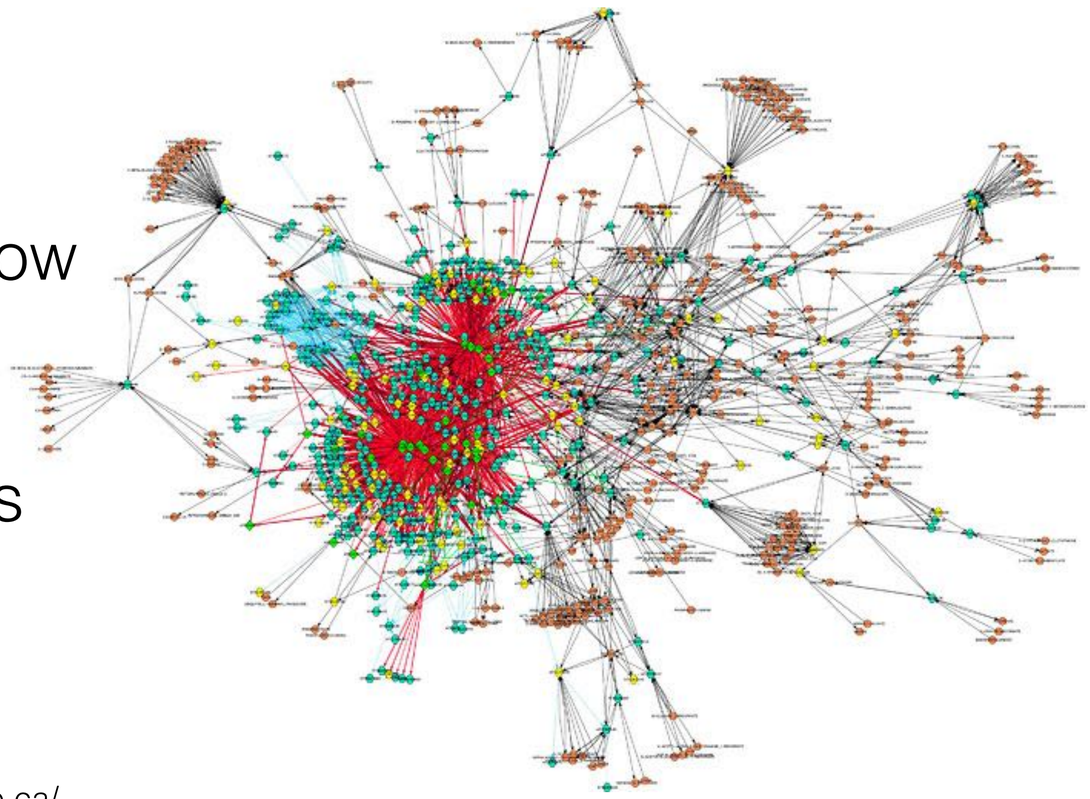


http://www.deeplearningbook.org/

Applications to Cyber

Network-Based Intrusion Detection

- Derive Features from Network Traffic Captures “pcap” at packet level or NetFlow level (tools: tshark, tcpdump, bro…)
- Example Features based on header information: 2s-windowing of connections > duration, protocol, src and dat bytes, service.
- Get data sets: http://www.netresec.com/?page=PcapFiles, https://maccdc.org/, http://www.westpoint.edu/crc/SitePages/DataSets.aspx http://www.unb.ca/cic/research/datasets/,



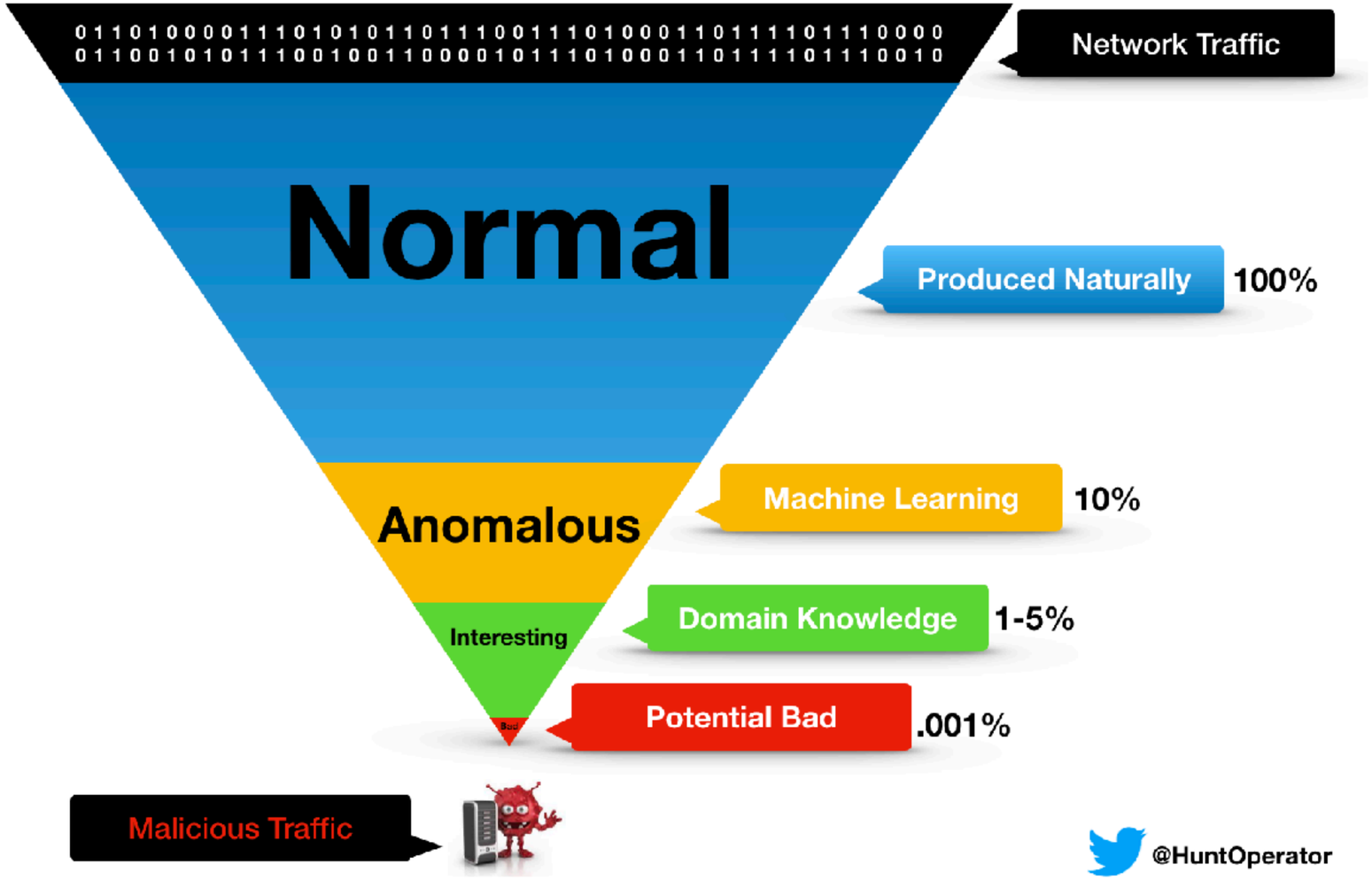
Malware Detection/Classification

- Derive Features from Binary Content and metadata manifest (function calls, string obtained from IDA Disassembler)
- Example Features: opcode count (n-grams), segment count, asm pixel intensity, n-gramming of bytes, function name.
- Featureless Deep Learning with word2vec embedding
- Get open source malware samples: Vx Heaven, Virus Share, Maltrieve, Open Malware

More Applications

- Domain Generation Algorithm (DGA) Detection
- **Malicious URL Detection**
- Network Traffic: Beaconing Detection
- Detection of new classes of malware
- General Network Traffic Anomaly Detection
- Log Analysis - Anomaly Detection
- Phishing Detection
- Identifying SQL Injection
- Identifying XSS cross-site scripting
- DOS/DDOS Detection
- Authentication

Data Science Hunting Funnel

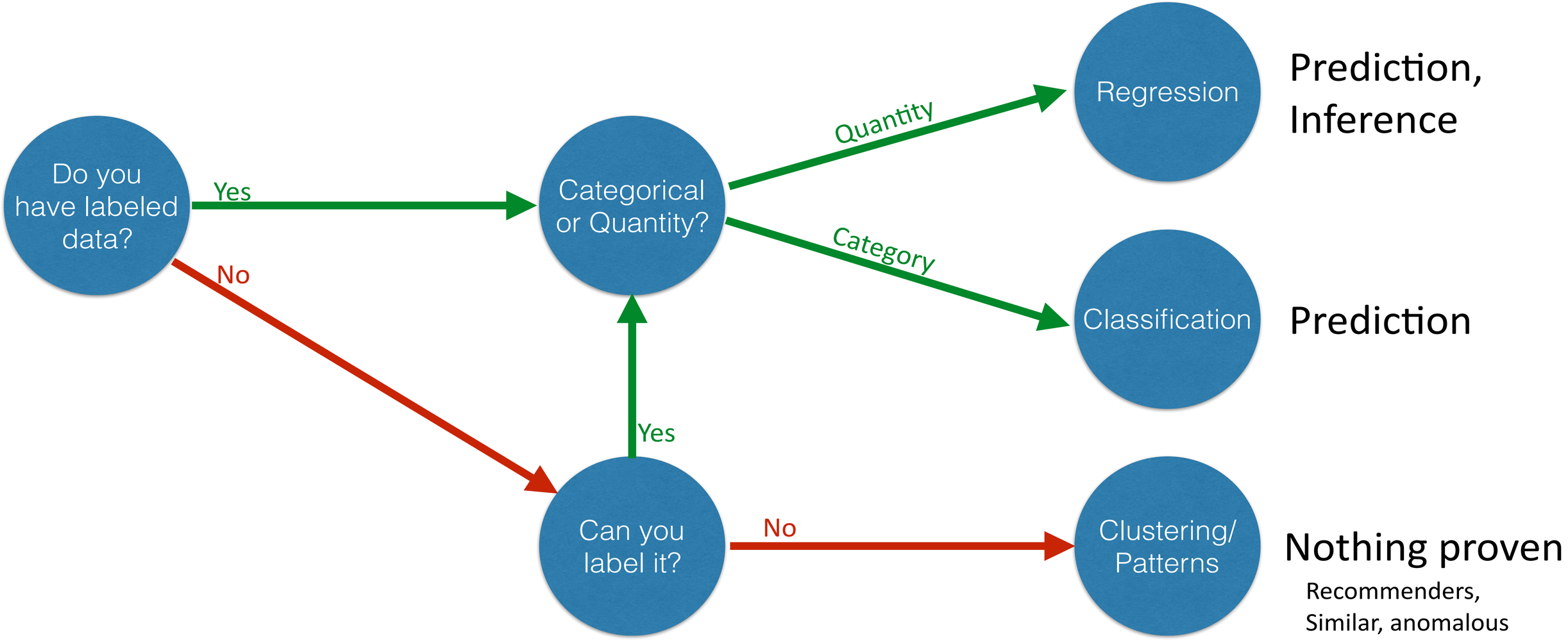


http://www.austintaylor.io/network/traffic/threat/data/science/hunting/funnel/machine/learning/domain/expertise/2017/07/11/data-science-hunting-funnel/

Machine Learning Problems

- **Supervised Learning:** Supervised Learning is a class of Machine Learning in which a model is "trained" using a set of pre-existing labeled data.

- **Unsupervised Learning:** A class of Machine Learning algorithms in which a model is built without the use of labeled data.


Do you have labeled data?
Yes
No
Categorical or Quantity?
Quantity
Category
Regression
Classification
Can you label it?
Yes
No
Clustering/ Patterns
Prediction, Inference
Prediction
Nothing proven
Recommenders, Similar, anomalous

Classification Example

Spam Filtering: Spam filtering is a common use of classification algorithms. Spam filters will collect examples of legit and spam emails and use **Naive Bayes Classifiers**, or **Deep Learning** classifiers to keep spam out of your inbox.

- https://hackernoon.com/how-to-build-a-simple-spam-detecting-machine-learning-classifier-4471fe6b816e
- https://www.booleanworld.com/building-spam-filter-using-machine-learning/

Regression Example

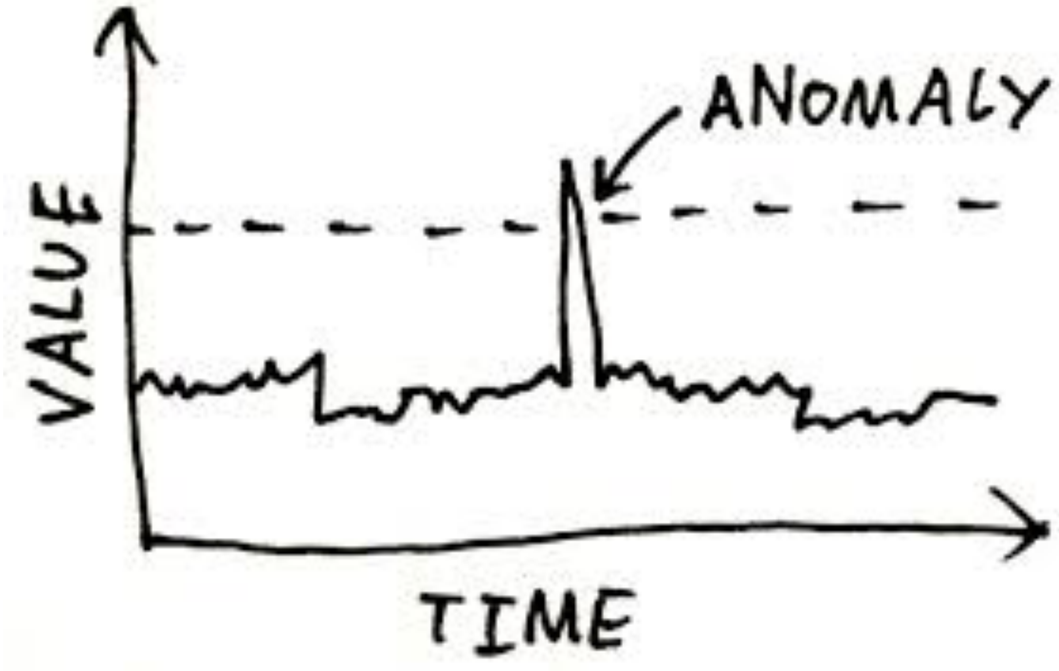
Server Capacity Prediction: Regression analysis can be used to predict a server's capacity (or CPU usage) based on the server's historical performance.

https://www.researchgate.net/publication/256645877_LiRCUP_Linear_Regression_based_CPU_Usage_Prediction_Algorithm_for_Live_Migration_of_Virtual_Machines_in_Data_Centers

https://jgreenemi.com/predicting-capacity-with-linear-regression-ml/

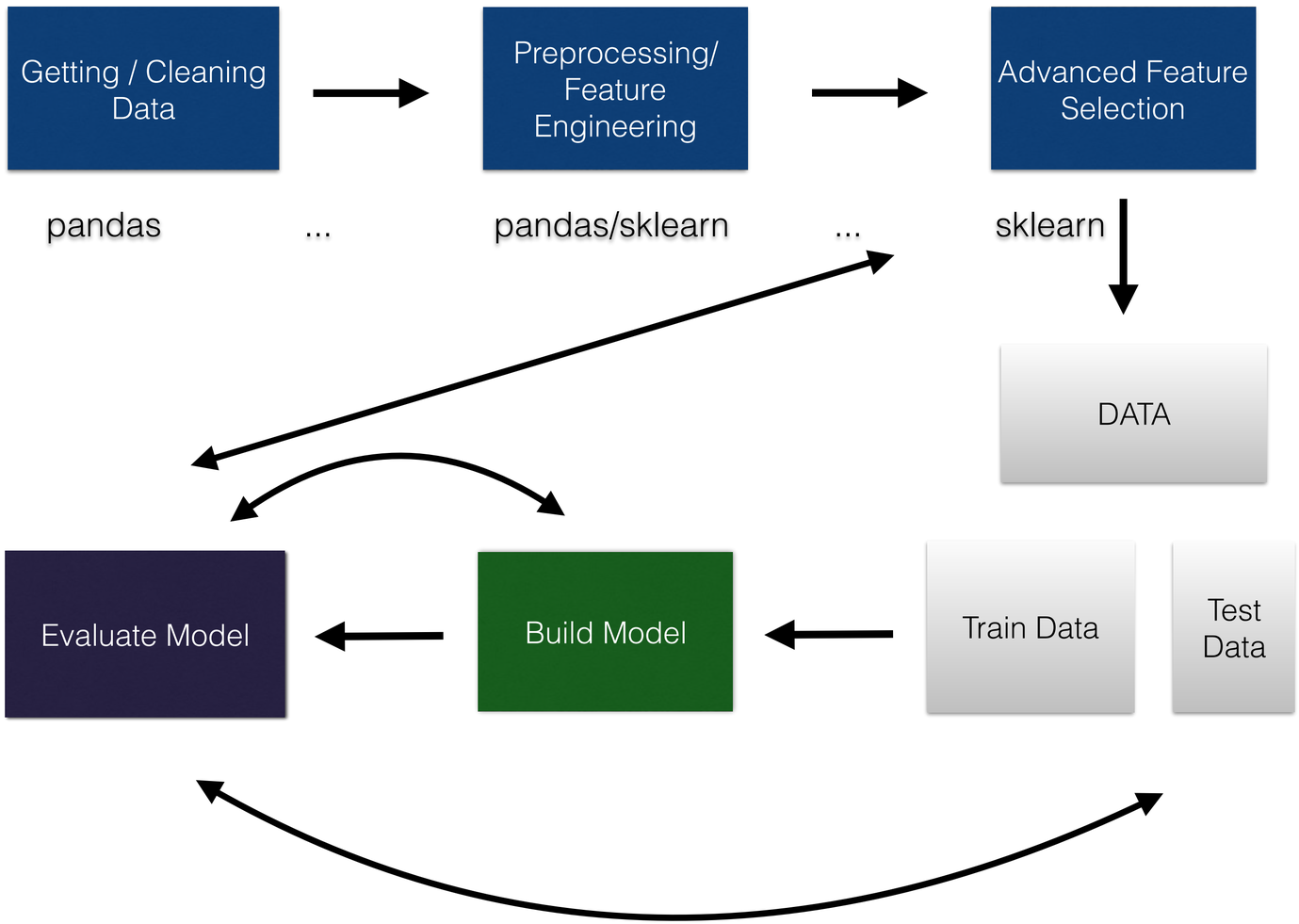
Clustering Example

Anomaly Detection: Clustering techniques can be used to detect anomalous traffic or loads or anything really.



Machine Learning Process

Machine Learning Process



First, define your analytic question.

What are you trying to do?

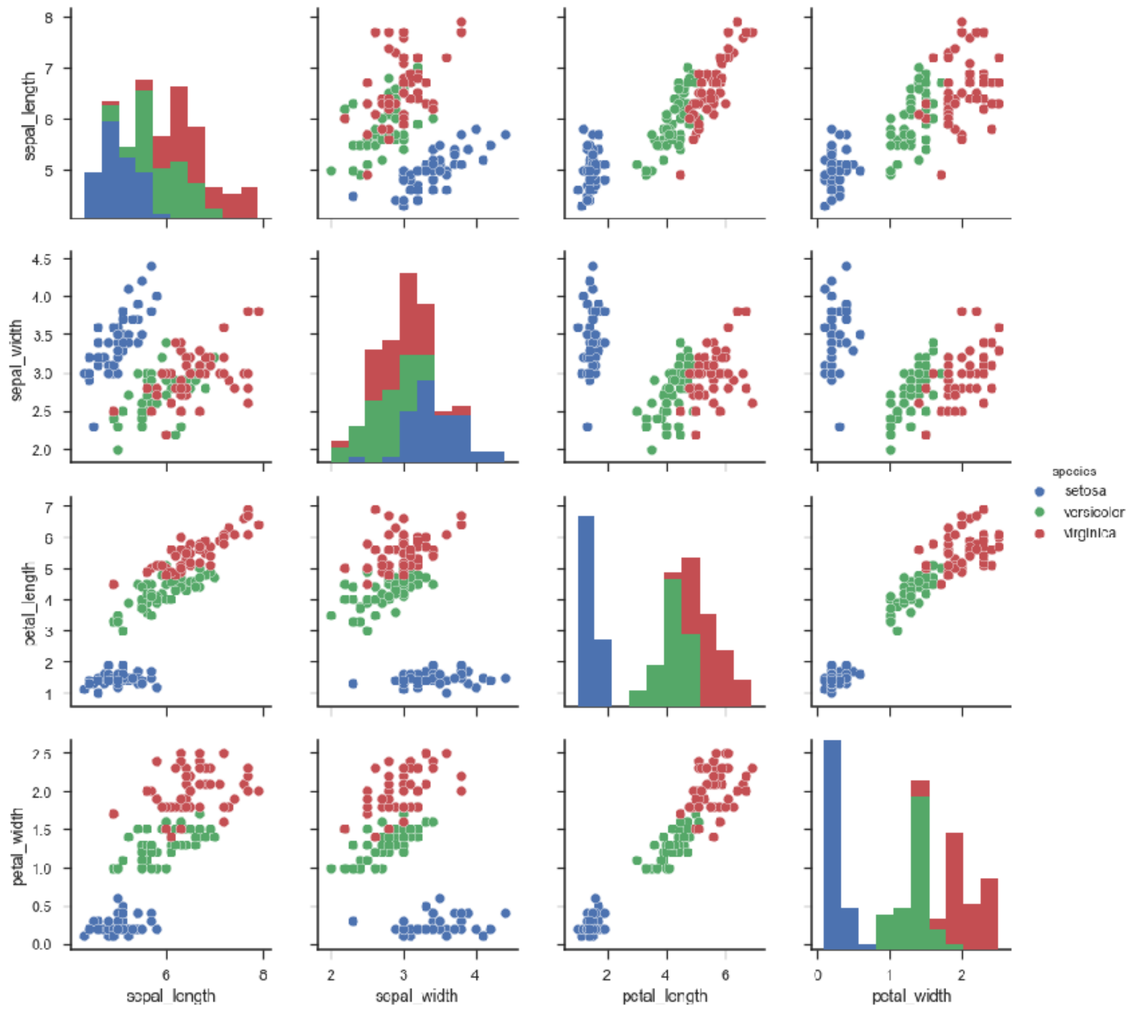
Choose data sources

- What is available?
- Is it enough?
- Is the data reliable/clean/consistent?
- What other data could you use?

Other Considerations

- Policies
- Legal contraints
- Biases in Data
- Latency
- Data size

Gather and Explore Your Data



Is the data good enough?

What are the rules governing its use?

Do I have enough?

Do problems or biases exist in the data that could cause problems?

Feature Engineering

- Define what you are trying to measure. These will become the observations or rows of your final dataset

- Define how you will mathematically represent your data. This will be come the features or columns of your final dataset.

Build and Tune your Model

- Believe it or not, this is the easy part.
- Most of this is **done using libraries** like scikit-learn, mllib, tensorflow, caret or keras, and **many steps can be automated**.

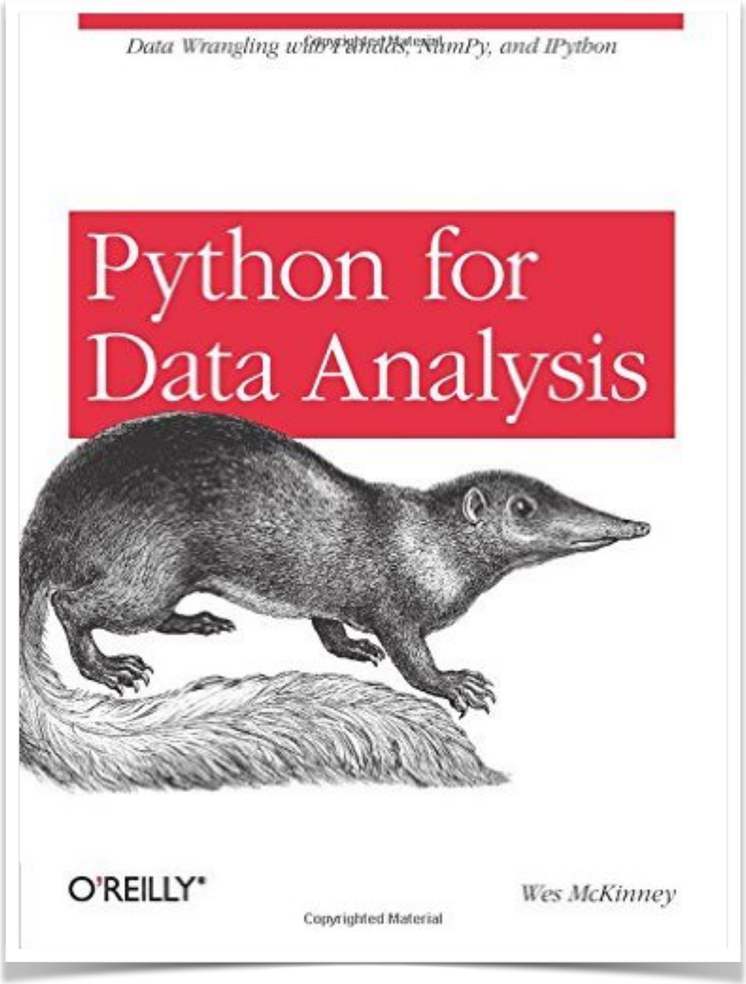
Evaluate Performance

- Use various scoring methods, or write your own to determine model performance.
- Go back to step 1 and repeat! (Do not pass go, do not collect $200)

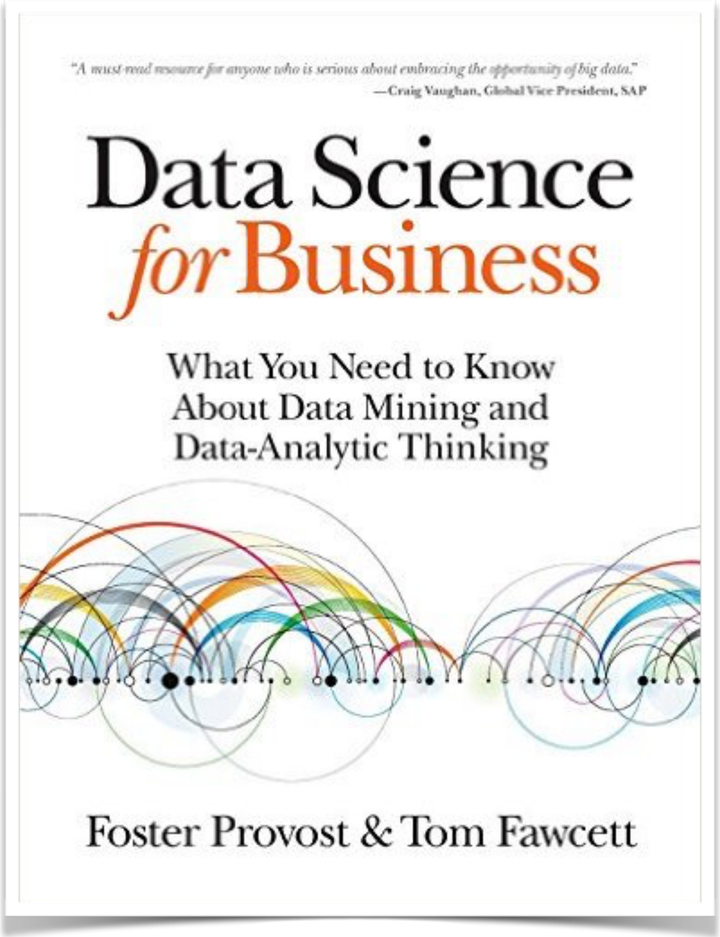
Machine Learning Ecosystem

- **Data Gathering:** Pandas, Drill, BeautifulSoup, PyDBAPI
- **Feature Extraction:** Pandas, NumPy
- **Machine Learning**
  - **Regular ML:** Scikit-learn (sklearn), h2o, mllib (PySpark)
  - **Deep Learning:** Tensorflow, Keras, Theano, Caffe
- **Visualization:** Matplotlib, Seaborn, Yellowbrick, LIME, ggplot, plot.ly,

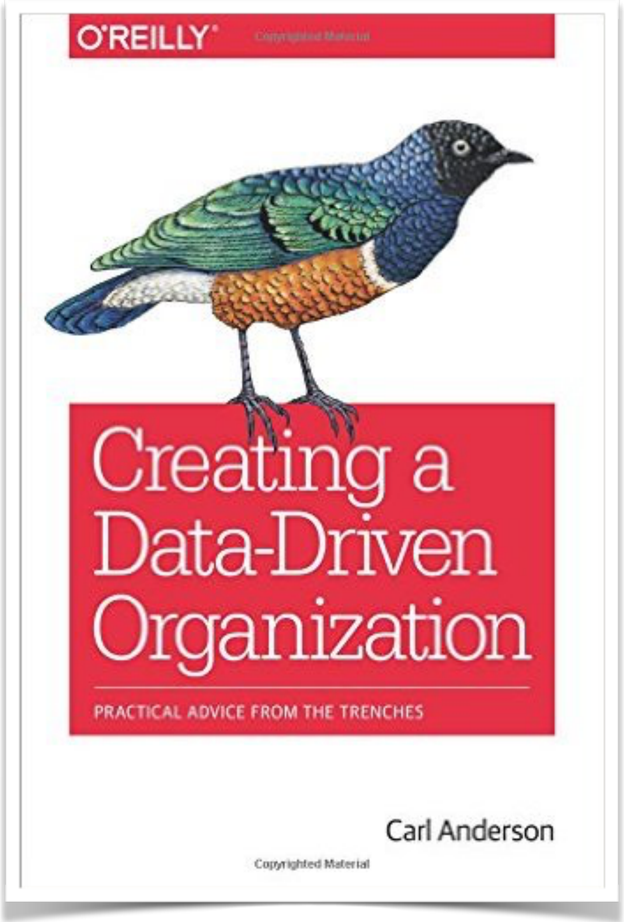
Recommended Reading



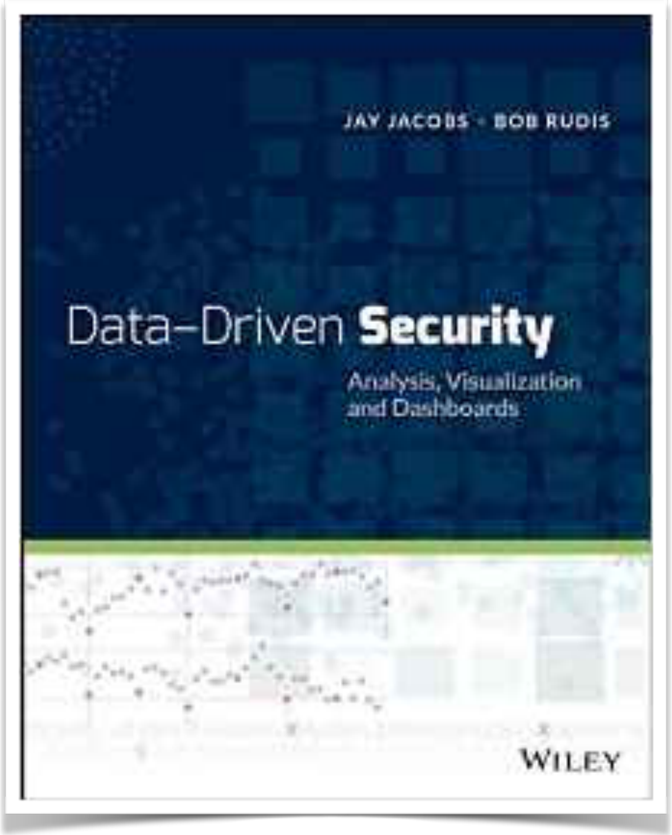
Recommended Reading



Recommended Reading



Recommended Reading

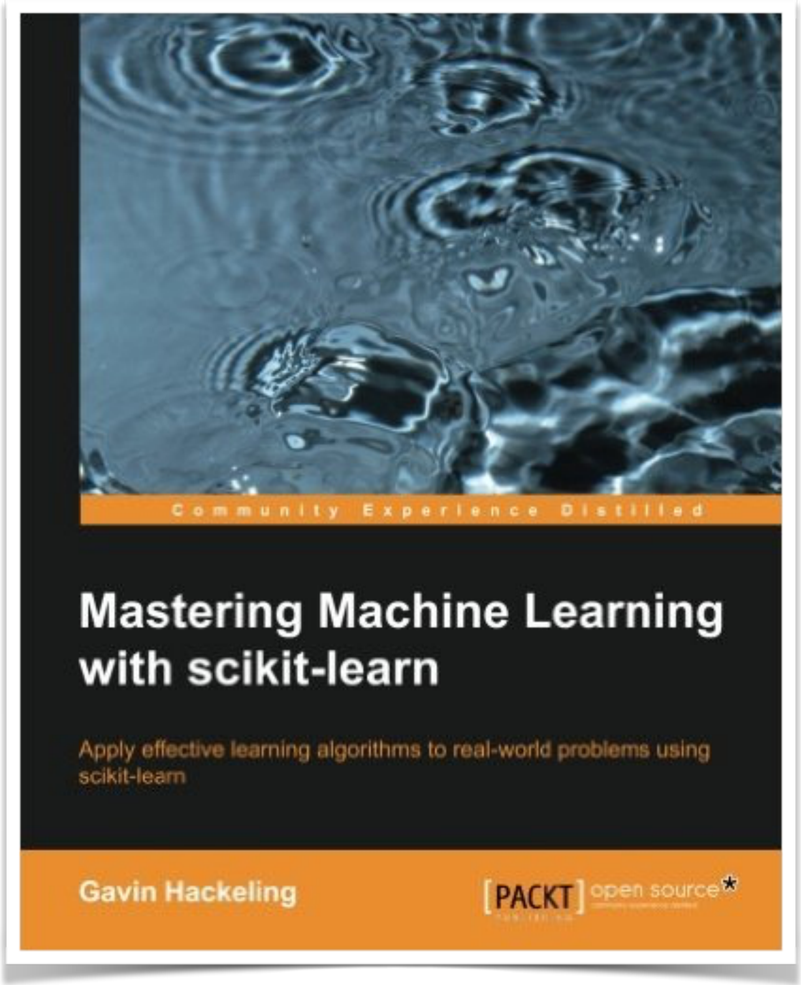


http://datadrivensecurity.info

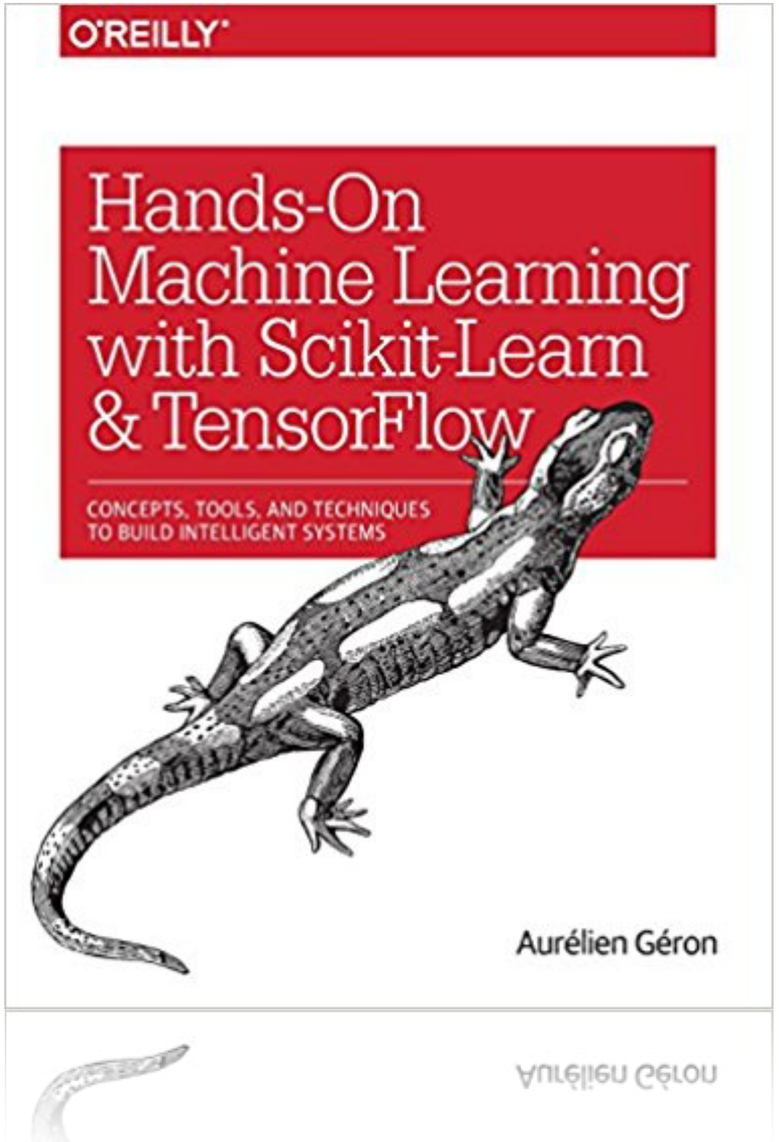
Recommended Reading

http://www.boozallen.com/media/file/The-Field-Guide-to-Data-Science.pdf

Recommended Reading



Recommended Reading



10 Min Break

Exploratory Data Analysis & Feature Engineering

Part 1: One Dimensional Data

GTK CYBER
GET ON THE FAST TRACK

What is it?

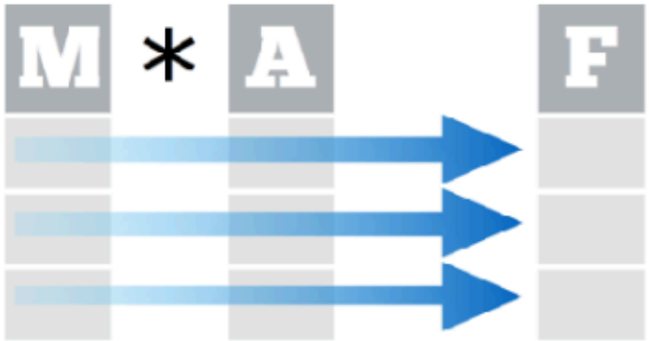
Vectorized Data Structures let you **perform operations** on your data **all at once**

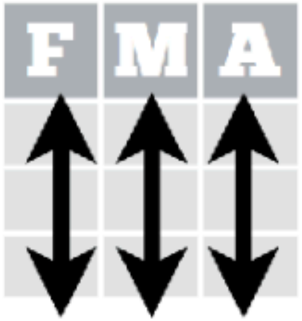
```
for x in range(0, len( data )):
    data[x] = data[x] + 1
```

```
for x in range(0, len( data )):
    data[x] = data[x] + 1
```

```
odds = evens + 1
```

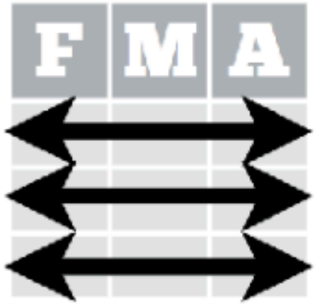
F = M * A





Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

No loops

Dimensions	Name	Description
1	Series	Indexed 1 dimensional data structure
1	Timeseries	Series using timestamps as an index
2	DataFrame	A two dimensional table
3	Panel	A three dimensional mutable data structure (Too complicated)

The Series Object

A **Series** is like a list or a dictionary **but BETTER!!**

The **Series** is the **primary building block** for all the other data structures we will discuss

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
myData  = pd.Series( <data>, index=<index> )
```

Creating a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])

series2 = pd.Series( {'firstName':'Charles',
'lastName':'Givre'} )
```

Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])
```

```
print( series1 )
```

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])

print( series1[3] )
```

```
d
```

Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])
```

```
print( series1[1:3] )
```

```
1    b
2    c
dtype: object
```

Accessing a Series

```
series1 = pd.Series( ['a','b','c','d', 'e'])
```

```
print( series1[1:3] )
```

```
1    b
2    c
dtype: object
```

```
series2 = series1[1:3]
series2.reset_index( drop=True )
```

Accessing a Series

```
record = pd.Series(
    {‘firstname':'Charles',
     'lastname':'Givre',
    'middle':'classified' } )
```

```
record['firstname']                    Charles
```

Accessing a Series

```
record = pd.Series(
   {‘firstname':'Charles',
    'lastname':'Givre',
   'middle':'classified' } )
```

```
record[['firstname', ‘lastname']]
```

```
firstname    Charles
lastname       Givre
dtype: object
```

Accessing a Series

```
record = pd.Series(
   {‘firstname':'Charles',
    'lastname':'Givre',
   'middle':'classified' } )
```

```
record[0]
```

```
Charles
```

Accessing a Series

```
randomNumbers = pd.Series(
 np.random.random_integers(1, 100, 50) )

randomNumbers.head()
0     48
1     34
2     84
3     85
4     58
dtype: int64
```

Accessing a Series

```
randomNumbers = pd.Series(
 np.random.random_integers(1, 100, 50) )

randomNumbers.tail( 7 )
43    66
44    66
45    43
46    55
47    99
48    82
49    19
dtype: int64
```

Filtering Data in a Series

```
randomNumbers [ <boolean condition> ]
randomNumbers [ randomNumbers < 10 ]
```

```
12    5
21    2
24    1
27    1
29    1
dtype: int64
```

```
record.str.contains('Cha')
firstname     True
lastname     False
middle       False
dtype: bool

record[ record.str.contains('Cha') ]
firstname    Charles
dtype: object
```

String Functions

Function	Explanation
Series.str.contains(<pattern>)	Returns true/false if text matches a pattern
Series.str.count(<pattern>)	Returns number of occurrences of a pattern in a string
Series.str.extract(<pattern>)	Returns matching groups from a string
Series.str.find(<string>)	Returns index of first occurrences of a substring (Note: not regex)
Series.str.findall(<pattern>)	Returns all occurrences of a regex
Series.str.len()	Returns the length of text
Series.str.replace(<pat>, <replace>)	Replaces matches with a replacement string

Manipulating Data

Pandas Data Structures allow you
to **perform operations**
on your **entire data set at once.**

```
for x in range(0, len( data )):
    data[x] = data[x] + 1
```

```
for x in range(0, len( data )):
    data[x] = data[x] + 1
```

```
odds = evens + 1
```

```
combinedSeries = series1.add( series2 )
```

```
def addTwo( n ):
    return n + 2

odds.apply( addTwo )
```

```
odds.apply( lambda x: x + 1 )
```

```
Series.dropna()
Series.fillna(value="<something>")
```

Questions?

In Class Exercise:

Please complete Day 1 - Working with One Dimensional Data

The Data Frame

```
data = pd.DataFrame( <data>, <index>,
                     <column_names> )
```

CSV

```
data = pd.read_csv( <file> )
```

Excel

```
data = pd.read_excel( <file>, sheetname=<sheetname> )
```

```
[
  {
   “changed":"2015-04-0300:00:00",
   "created":"2014-04-10 00:00:00”,
   “dnssec":"False",
   "expires":"2016-04-10 00:00:00”,
   “isMalicious":0,
   “url":"nuteczki.com"
  },
…
]
```

JSON

```
data = pd.read_json( <file> )
or
data = pd.read_json( <url> )
```

From a Database

```
data = pd.read_sql( <query>, <connection> )
```

HTML

```
data = pd.read_html( <source> )
```

Log Files…

```
070823 21:00:32        1 Connect     root@localhost on test1
070823 21:00:48        1 Query       show tables
070823 21:00:56        1 Query       select * from category
070917 16:29:01       21 Query       select * from location
070917 16:29:12       21 Query       select * from location where id = 1 LIMIT 1
```

```
070823 21:00:32        1 Connect     root@localhost on test1
070823 21:00:48        1 Query       show tables
070823 21:00:56        1 Query       select * from category
070917 16:29:01       21 Query       select * from location
070917 16:29:12       21 Query       select * from location where id = 1 LIMIT 1
```

```
logdf = pd.read_table('../data/mysql.log', names=['raw'])
```

	raw
0	070823 21:00:32 1 Connect root@local...
1	070823 21:00:48 1 Query show tables
2	070823 21:00:56 1 Query select * f...
3	070917 16:29:01 21 Query select * f...
4	070917 16:29:12 21 Query select * f...

```
logdf = pd.read_table('../data/mysql.log', names=['raw'])
logdf['raw'].str.extract('(\d{6}\s\d{2}:\d{2}:\d{2})\s+(\d+)\s(\S+)
\s(.+)', expand=False)
```

	0	1	2	3
0	070823 21:00:32	1	Connect	root@localhost on test1
1	070823 21:00:48	1	Query	show tables
2	070823 21:00:56	1	Query	select * from category
3	070917 16:29:01	21	Query	select * from location
4	070917 16:29:12	21	Query	select * from location where id = 1 LIMIT 1

```
logdf = pd.read_table('../data/mysql.log', names=['raw'])
logdf['raw'].str.extract('(?P<date>\d{6}\s\d{2}:\d{2}:\d{2})\s+(?
P<PID>\d+)\s(?P<Action>\S+)\s(?P<Query>.+)', expand=False)
```

	Date	PID	Action	Query
0	070823 21:00:32	1	Connect	root@localhost on test1
1	070823 21:00:48	1	Query	show tables
2	070823 21:00:56	1	Query	select * from category
3	070917 16:29:01	21	Query	select * from location
4	070917 16:29:12	21	Query	select * from location where id = 1 LIMIT 1

Web Server Logs

Web Server Logs

```
195.154.46.135 - - [25/Oct/2015:04:11:25 +0100] "GET /linux/
doing-pxe-without-dhcp-control HTTP/1.1" 200 24323 "http://
howto.basjes.nl/" "Mozilla/5.0 (Windows NT 5.1; rv:35.0)
Gecko/20100101 Firefox/35.0"
```

Web Server Logs

```
195.154.46.135 - - [25/Oct/2015:04:11:25 +0100] "GET /linux/
doing-pxe-without-dhcp-control HTTP/1.1" 200 24323 "http://
howto.basjes.nl/" "Mozilla/5.0 (Windows NT 5.1; rv:35.0)
Gecko/20100101 Firefox/35.0"
```

```
import apache_log_parser
line_parser = apache_log_parser.make_parser("%h %l %u %t
\"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"")
```

Web Server Logs

```
import apache_log_parser
line_parser = apache_log_parser.make_parser("%h %l %u %t
\"%r\" %>s %b \"%{Referer}i\" \”%{User-agent}i\”")

server_log = open(“../data/hackers-access.httpd”, "r")
parsed_server_data = []
for line in server_log:
    data = {}
    data = line_parser(line)
    parsed_server_data.append( data )

server_df = pd.DataFrame( parsed_server_data  )
```

Web Server Logs

er	request_first_line	request_header_referer	request_header_user_agent	request_http_ver	request_method	request_url
	GET /linux/doing-pxe-without-dhcp-control HTTP...	http://howto.basjes.nl/	Mozilla/5.0 (Windows NT 5.1; rv:35.0) Gecko/20...	1.1	GET	/linux/doing-pxe-v
	GET /join_form HTTP/1.0	http://howto.basjes.nl/	Mozilla/5.0 (Windows NT 5.1; rv:35.0) Gecko/20...	1.0	GET	/join_form
	POST /join_form HTTP/1.1	http://howto.basjes.nl/join_form	Mozilla/5.0 (Windows NT 5.1; rv:35.0) Gecko/20...	1.1	POST	/join_form
	GET /join_form HTTP/1.0	http://howto.basjes.nl/	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0) G...	1.0	GET	/join_form
	POST /join_form HTTP/1.1	http://howto.basjes.nl/join_form	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0) G...	1.1	POST	/join_form
	GET /acl_users/credentials_cookie_auth/require...	http://howto.basjes.nl/join_form	Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0) G...	1.1	GET	/acl_users/creden

Nested Data?

Nested Data?


```
{"time": 1084443427.311224,
 "timestamp": "2004-05-13T10:17:07.311224",
"IP": {
   "version": 4,
   "ttl": 128,
   "proto": 6,
   "options": [],
   "len": 48,
   "dst": "65.208.228.223",
   "frag": 0,
   "flags": 2, "src": “145.254.160.237",
   "chksum": 37355
   },
   "Ethernet": {"src": "00:00:01:00:00:00", "type": 2048, "dst": “fe:ff:20:00:01:00"},
   …
```

Nested Data

```
pd.read_json( ‘nested_data.json’ )
```

Nested Data

```
pd.read_json( 'nested_data.json' )
```

	DNS	Ethernet	IP	TCP	UDP	time	timestamp
0	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 48, 'version'...	{'window': 8760, 'chksum': 49932, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:07.311224
1	NaN	{'type': 2048, 'dst': '00:00:01:00:00:00', 'sr...	{'dst': '145.254.160.237', 'len': 48, 'version...	{'window': 5840, 'chksum': 23516, 'sport': 80,...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
2	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 40, 'version'...	{'window': 9660, 'chksum': 31076, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
3	NaN	{'type': 2048, 'dst': 'fe:ff:20:00:01:00', 'sr...	{'dst': '65.208.228.223', 'len': 519, 'version...	{'window': 9660, 'chksum': 43352, 'sport': 337...	NaN	1.084443e+09	2004-05-13 10:17:08.222534
4	NaN	{'type': 2048, 'dst': '00:00:01:00:00:00', 'sr...	{'dst': '145.254.160.237', 'len': 40, 'version...	{'window': 6432, 'chksum': 33825, 'sport': 80,...	NaN	1.084443e+09	2004-05-13 10:17:08.783340

Nested Data

```
from pandas.io.json import json_normalize
import json
import pandas as pd

with open(‘nested.json') as data_file:
    pcap_data = json.load(data_file)

df = pd.DataFrame( json_normalize(pcap_data) )
```

Nested Data

```
df = pd.DataFrame( json_normalize(pcap_data) )
```

...	TCP.seq	TCP.sport	TCP.urgptr	TCP.window	L
...	951057939.0	3372.0	0.0	8760.0	
...	290218379.0	80.0	0.0	5840.0	
...	951057940.0	3372.0	0.0	9660.0	
...	951057940.0	3372.0	0.0	9660.0	
...	290218380.0	80.0	0.0	6432.0	

Manipulating a DataFrame

```
df = data['column1']
```

Returns a series

```
df['ip'].value_counts().head()
```

```
df = data[['column1','column2','column3']]
```

Returns a DataFrame

Creating a new column

```
df['new_column'] = df['old_column'] * 2
```

Filtering a DataFrame

```
df[<boolean condition>]
```

```
df[['col1','col2']][col3 > 5]
```

Columns

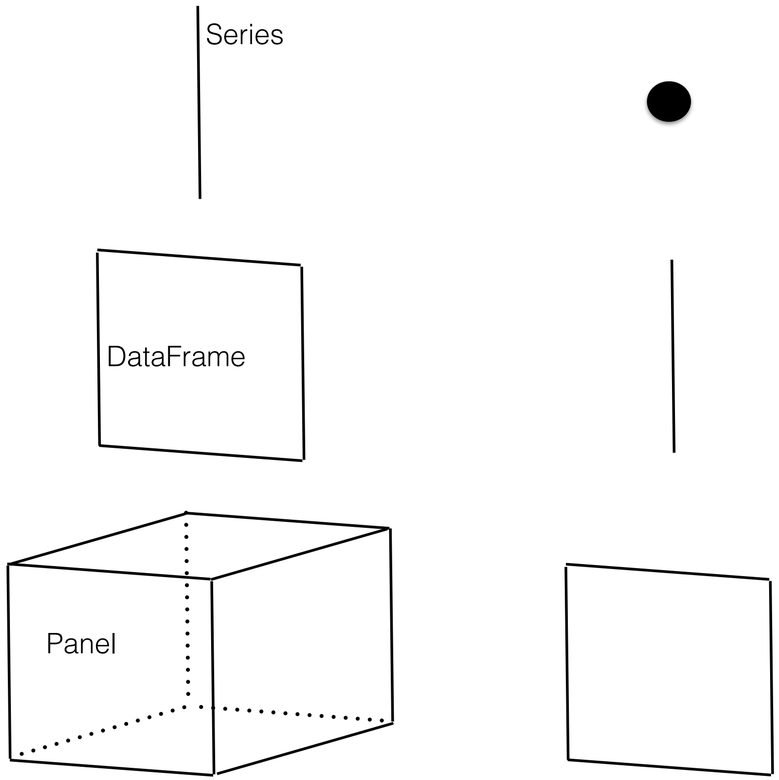
Filter

```
data.loc[<index>]
data.loc[<list of indexes>]
data.sample(<n>)
```

data.apply(<function>)

IF
Call Apply On...

THEN
Function
Receives



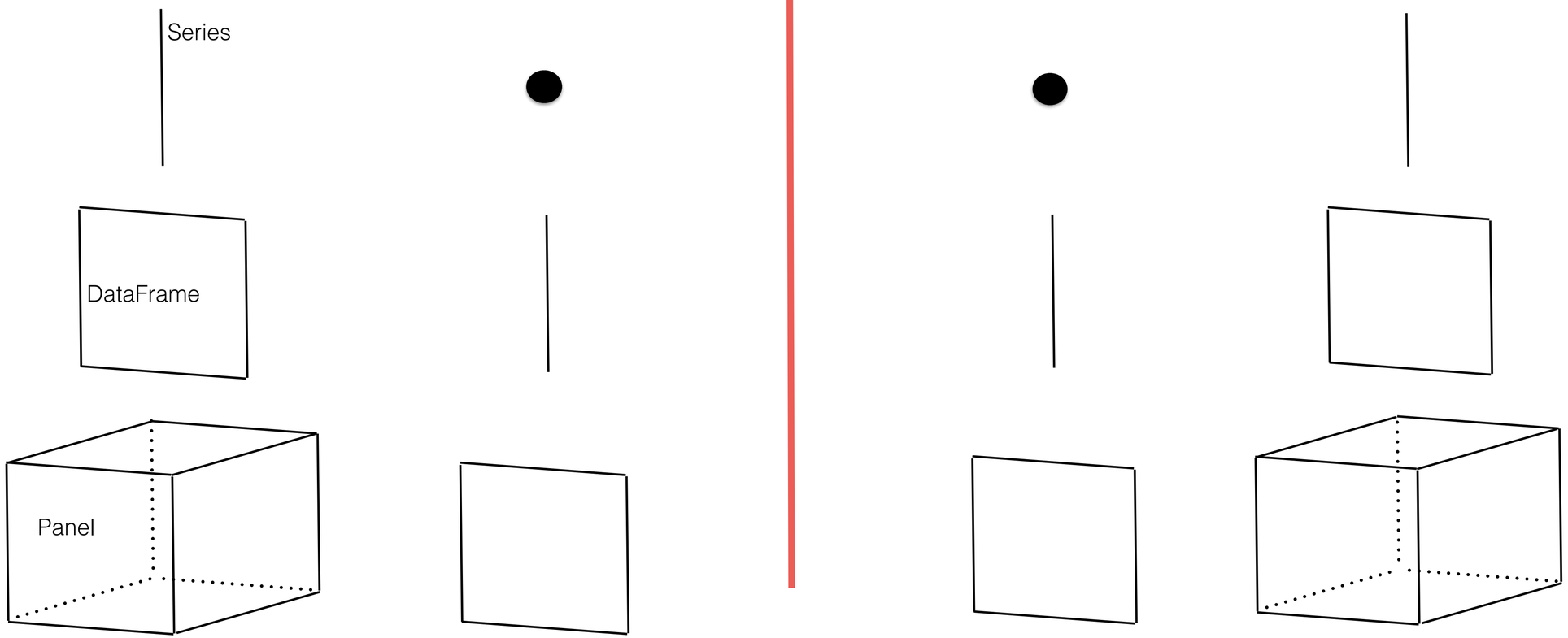
data.apply(<function>)

IF
Call Apply On...

THEN
Function
Receives

IF
Function
Returns

THEN
Apply
Returns...



1	2	3
4	5	6
7	8	9

```
data.T
```

1	4	7
2	5	8
3	6	9

```
#Gets you the sum of columns
data.sum( axis=0 )

#Gets you the sum of the rows
data.sum( axis=1 )
```

```
DataFrame.drop(labels,
axis=0,
level=None,
inplace=False,
errors='raise')¶
```

Merging Data Sets

Union

Union

Series 1

Index	Value
0	6
1	4
2	2
3	3

Union

Series 1

Index	Value
0	6
1	4
2	2
3	3

Series 2

Index	Value
0	7
1	5
2	3
3	4

Union

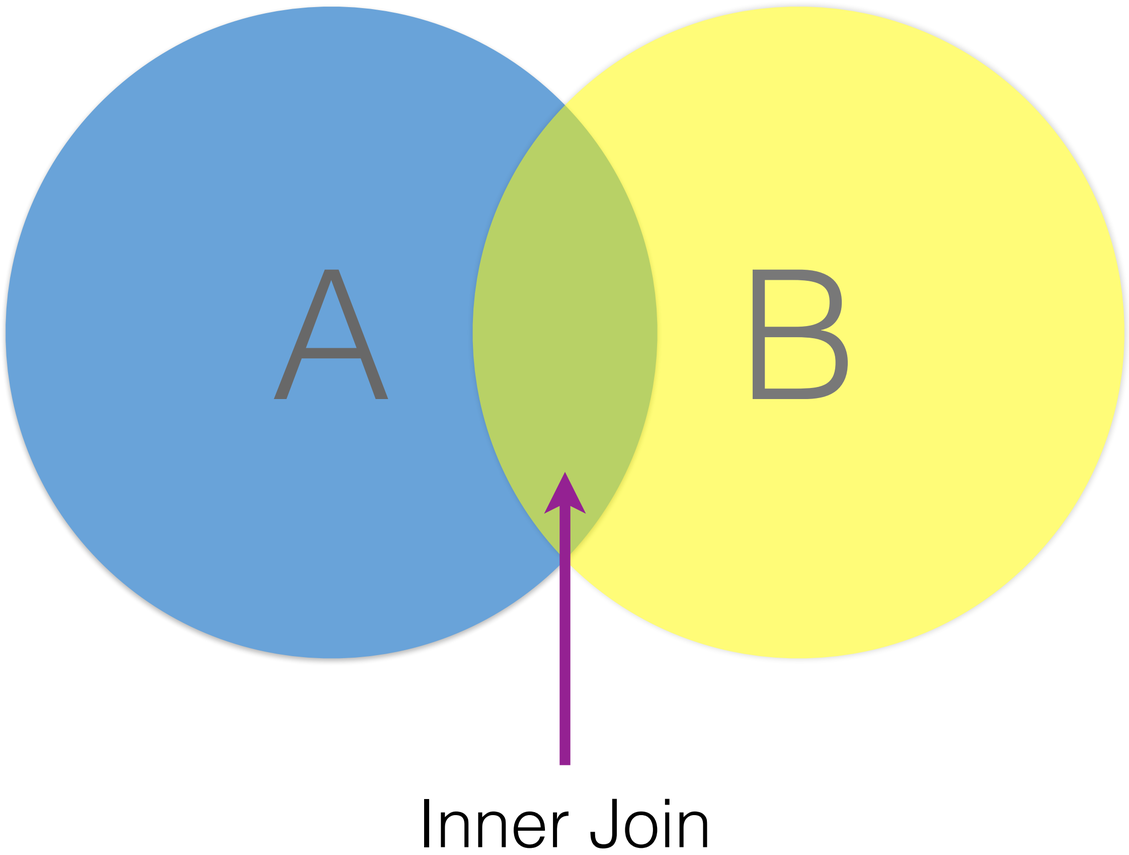
combinedSeries

Index	Value
0	6
1	4
2	2
3	3
4	7
5	5
6	3
7	4

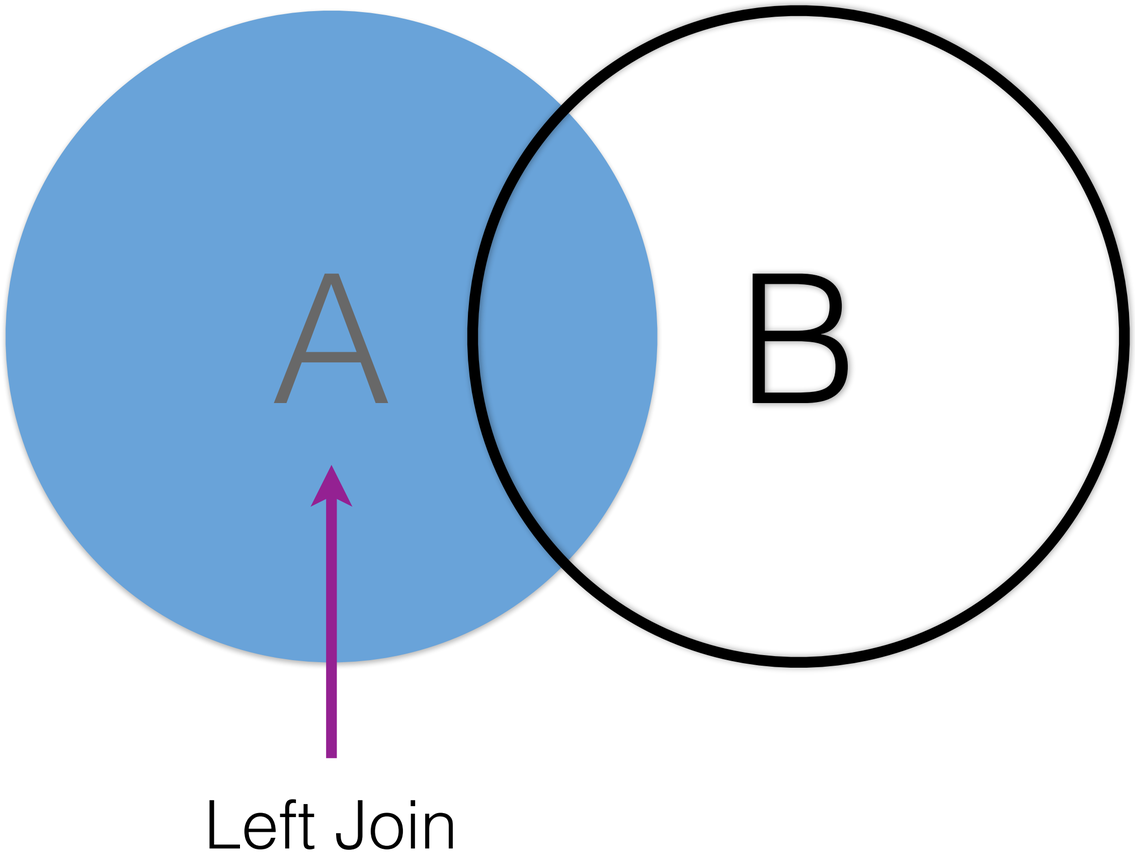
```
combinedSeries = pd.concat(
[series1, series2],
ignore_index=True )
```

Joins

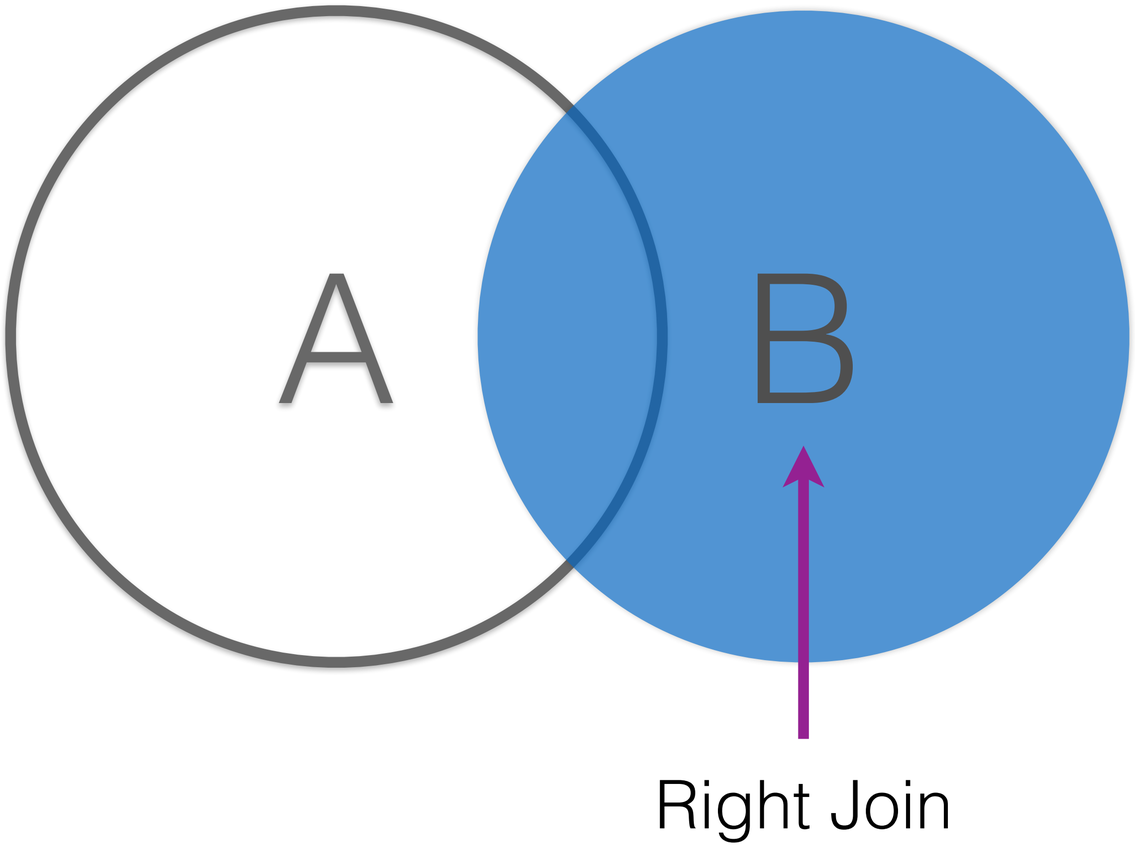
Inner Join (Intersection)



Left Join



Right Join



```
pd.merge( leftData, rightData )
```

```
pd.merge( leftData, rightData, how="<join type>" )
```

```
pd.merge( leftData, rightData, how="<join type>" )
```

Option	SQL Equivalent	Description
inner	INNER JOIN	Intersection
left	LEFT OUTER JOIN	Returns items in Set A, but not in Set B
right	RIGHT OUTER JOIN	Returns items in Set B, but not in Set A
outer	FULL OUTER JOIN	Returns the union of both sets

```
pd.merge( leftData, rightData,
how="<join type>",
on=<field list> )
```

Grouping and Aggregating Data

```
df_grouped = df.groupby(
['Protocol', 'Source','Destination'])
print(df_grouped.size())
```

```
Protocol  Source              Destination
 ARP       MoxaTech_27:8c:37  Westermo_1a:61:83      610
           Pegatron_3a:0d:e8  Ruggedco_64:85:c2      871
           Ruggedco_64:85:c2  Pegatron_3a:0d:e8      871
           SiemensN_27:64:11  Siemens-_89:59:82      428
           Westermo_1a:61:83  MoxaTech_27:8c:37      610
 DNS       192.168.88.1       192.168.88.61         9938
           192.168.88.61      192.168.88.1          9937
           192.168.89.2       8.8.8.8               2194
 ICMP      192.168.89.1       192.168.89.2          2194
 LOOP      CiscoInc_95:1d:8b  CiscoInc_95:1d:8b     1992
 S7COMM    10.10.10.10        10.10.10.20          19913
           10.10.10.20        10.10.10.10          19914
 TCP       10.10.10.10        10.10.10.20          23409
           10.10.10.20        10.10.10.10          59739
Name: Protocol, dtype: int64
```

```
Stats_Packets = df_grouped['Length'].agg({'No
Packets': len, 'Volume': sum,\
        'SD': lambda x: np.std(x, ddof=1) if
len(x)>2 else 0})
```

			Volume	SD	No Packets
Protocol	Source	Destination			
ARP	MoxaTech_27:8c:37	Westermo_1a:61:83	45140	0.000000	610
	Pegatron_3a:0d:e8	Ruggedco_64:85:c2	52260	0.000000	871
	Ruggedco_64:85:c2	Pegatron_3a:0d:e8	52260	0.000000	871
	SiemensN_27:64:11	Siemens-_89:59:82	25680	0.000000	428
	Westermo_1a:61:83	MoxaTech_27:8c:37	36600	0.000000	610
DNS	192.168.88.1	192.168.88.61	725474	0.000000	9938
	192.168.88.61	192.168.88.1	725401	0.000000	9937
	192.168.89.2	8.8.8.8	152391	1.442727	2194
ICMP	192.168.89.1	192.168.89.2	213823	1.442727	2194
LOOP	CiscoInc_95:1d:8b	CiscoInc_95:1d:8b	119520	0.000000	1992
S7COMM	10.10.10.10	10.10.10.20	2070952	0.000000	19913
	10.10.10.20	10.10.10.10	3046842	0.000000	19914
TCP	10.10.10.10	10.10.10.20	2280712	15.683145	23409
	10.10.10.20	10.10.10.10	5436249	43.840987	59739

Another Approach: Drill

Another Approach: Drill

- Drill is an open source query engine which allows you to query many kinds of self-describing data using ANSI SQL.
- Drill is fast, scalable and extremely versatile.

Another Approach: Drill

Out of the box, Drill can query:

- Delimited Data (csv, tsv, psv)
- Apache log files
- Avro
- Parquet
- JSON
- PCAP
- And more...

There are extensions available on GitHub that allow you to query log files and other data.

Another Approach: Drill

Out of the box, Drill can connect to:

- MySQL (or any JDBC compliant database)
- Hadoop
- S3/Azure/Google Cloud
- HBase
- Hive
- MongoDB
- And others...

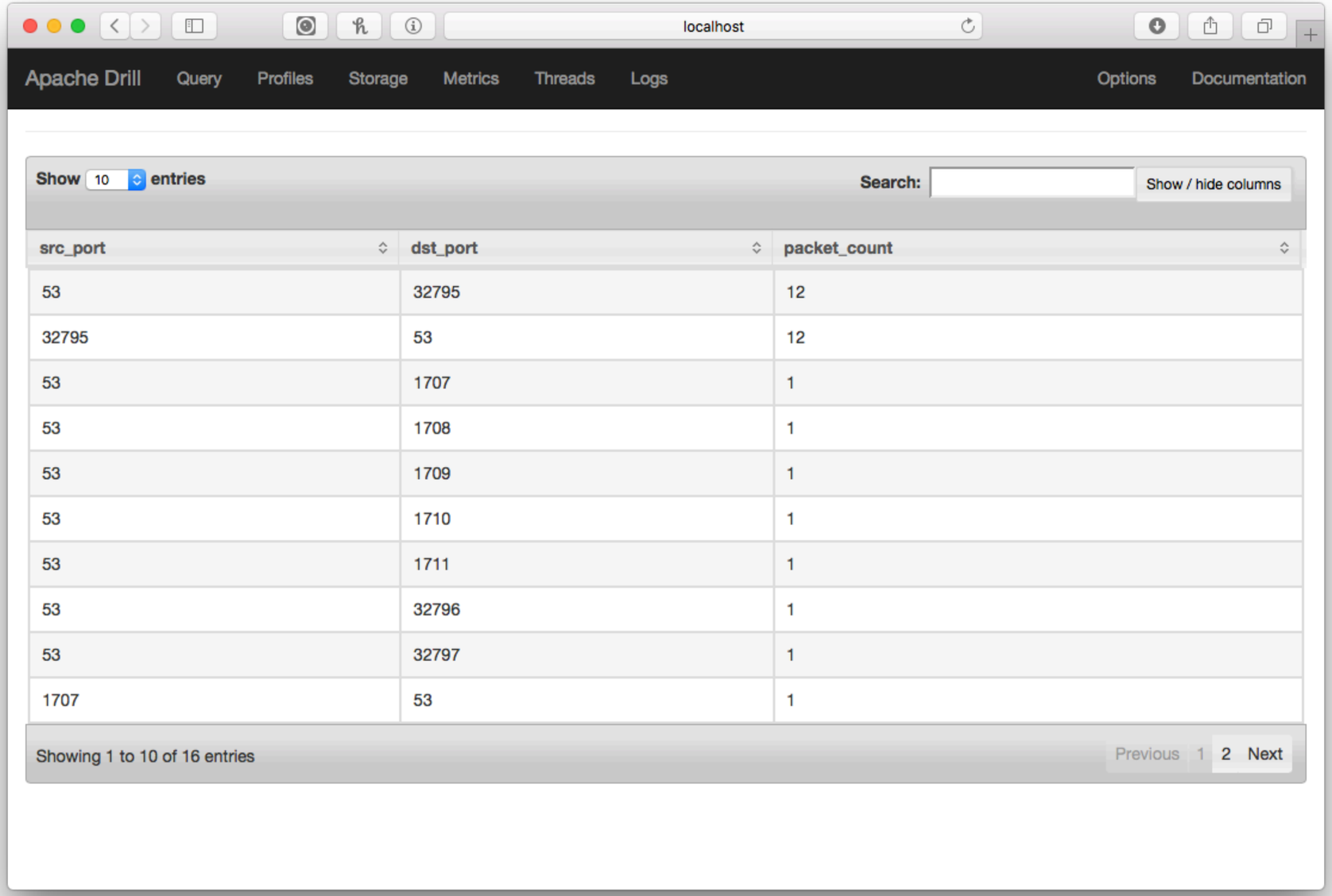
Another Approach: Drill

```
from pydrill.client import PyDrill

drill = PyDrill(host='localhost', port=8047)

if not drill.is_active():
    raise ImproperlyConfigured('Please run Drill first')

query = drill.query('''<Your query here>''')

df = query.to_dataframe()
```

Another Approach: Drill

```
SELECT src_port, dst_port, COUNT(*) AS packet_count
FROM dfs.test.`dns.pcap`
GROUP BY src_port, dst_port
ORDER BY packet_count DESC
```

Another Approach: Drill



src_port	dst_port	packet_count
53	32795	12
32795	53	12
53	1707	1
53	1708	1
53	1709	1
53	1710	1
53	1711	1
53	32796	1
53	32797	1
1707	53	1

In Class Exercise:

Please complete Day 1 - Working with Two Dimensional Data

Questions?

Explore Your Data

- **Mean**: The mean is the average value of a set of numbers

- **Median**: The median is the middle value of an ordered set of numbers

- **Mode**: The mode is the value from a set which is repeated the most frequently.

- **Range**: Difference between minimum and maximum

- **Standard Deviation**: Measures dispersion in a data set. Close to zero indicates little dispersion.

Intro Stats: Variance

The **variance** of a set of values, σ^2, is the average of the square of the difference of the values in the dataset from the dataset's mean.

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \mu)^2}{n}$$

```
iris_data_final.var(axis=0)
```

Intro Stats:
Std. Deviation

The standard deviation, σ, is the square root of the variance:

We use the **standard deviation** much more regularly than the **variance** because it is on the same scale as the original data:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

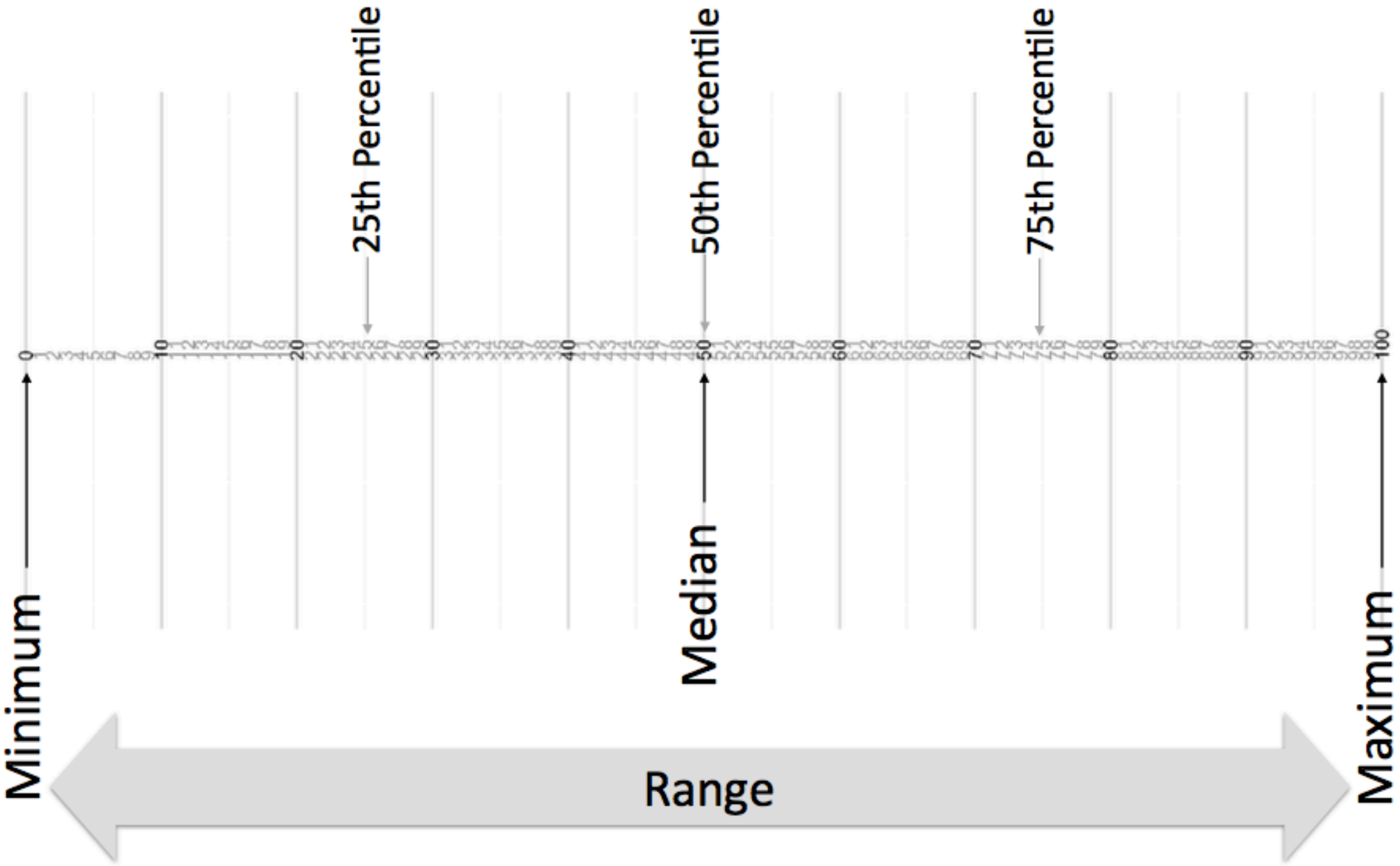
```
iris_data_final.std(axis=0)
```

Intro Stats: Covariance

Covariance, like the variance, is a measure of spread, however it also measures how closely two datasets track each other.

- **Covariance** is a squared quantity, so it is not on the same scale as the mean
- **Covariance** of different pairs of variables can have completely different scales

$$Cov(x, y) = \frac{\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y)}{n}$$


Minimum
0
10
20
25th Percentile
30
40
Median
50
60
70
75th Percentile
80
90
100
Maximum
Range

`Series.abs()`	Absolute Value of the Series
`Series.count()`	Returns number of non-empty values in the series
`Series.max()`	Returns maximum value in the Series
`Series.mean()`	Returns the mean of a Series
`Series.median()`	Returns the median of a Series
`Series.min()`	Returns the minimum value in a Series
`Series.mode()`	Take a guess..
`Series.quantile([q])`	Returns the quantiles of a Series
`Series.sum`	Returns the sum of a series
`Series.std`	Returns the standard deviation of a Series

Tukey 5 Number Summary

- **Minimum:** The smallest value in the dataset
- **Lower Quartile:** Smallest 25% of the dataset
- **The Median:** The middle value of the dataset
- **Upper Quartile:** The largest 25% of the dataset
- **Maximum:** The largest value in the dataset

```
Series.describe()
```

Series.describe()

```
>>> random_numbers.describe()
count    50.000000
mean     50.620000
std      30.102471
min       1.000000
25%      25.500000
50%      54.000000
75%      73.000000
max      99.000000
dtype: float64
```

```
names = pd.Series(
 ['Jim','Bob',
 'Bob','Steve','Jim,'Jane','Steph','Emma','Rachel
 '])

names.describe()
```

```
names = pd.Series(
 ['Jim','Bob',
 'Bob','Steve','Jim,'Jane','Steph','Emma','Rachel
 '])

names.describe()
```

```
count        9
unique       7
top        Jim
freq         2
dtype: object
```

Finding Unique Values

```
unique_nums = []
for key, value in random_numbers.iteritems():
    if value not in unique_nums:
        unique_nums.append(value)
```

NO!!!

```
series.drop_duplicates()
series.unique()
```

Counting Unique Values

```
series.value_counts()
```

```
series.value_counts(bins=5)
```

```
series.value_counts(bins=5,
normalize=True)
```

```
series.value_counts(bins=5,
normalize=True,
dropna=False)
```

Intro Stats: Correlation

Correlation is sort of like **standard deviation** generalized to pairs of datasets.

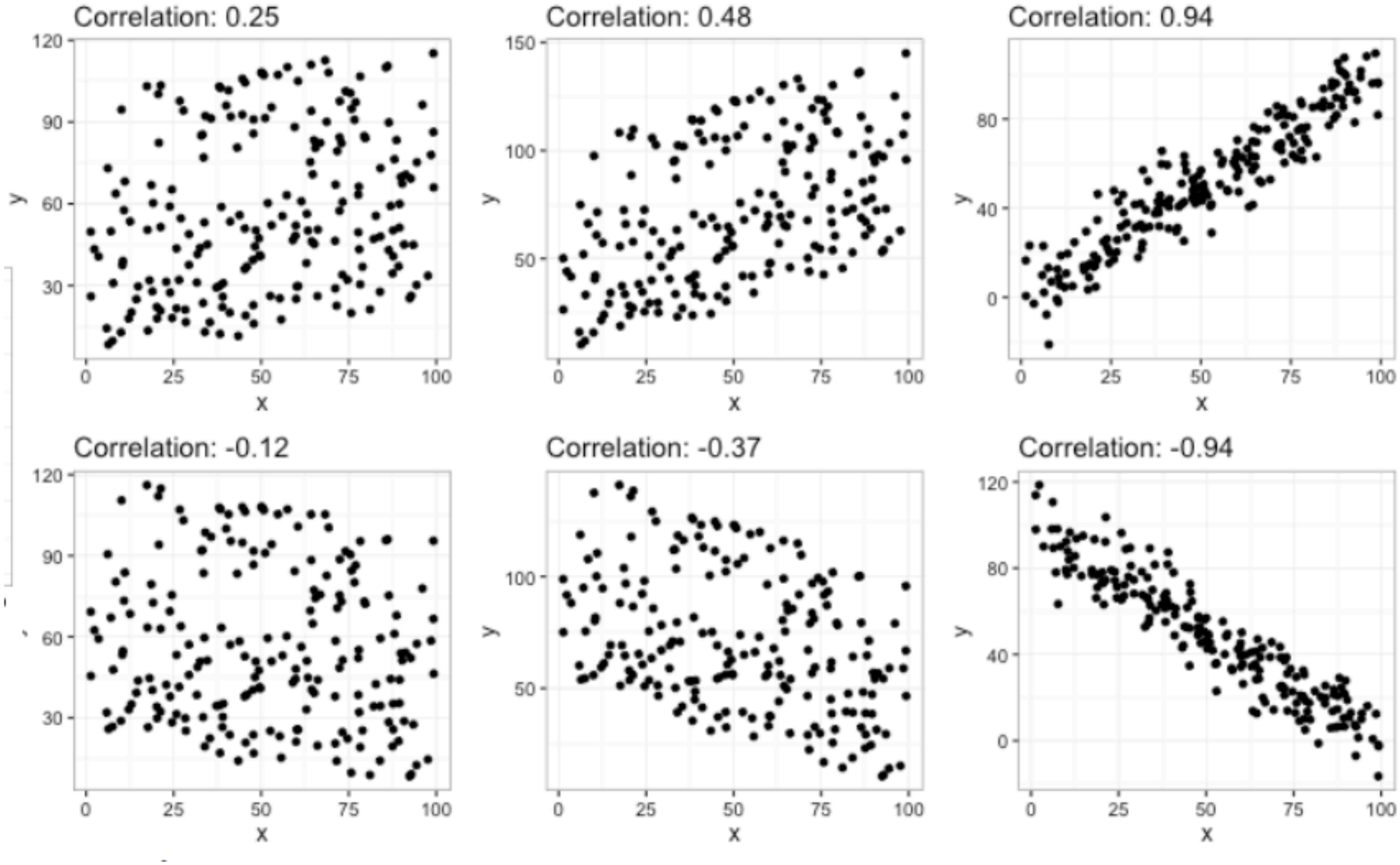
Really, the **correlation** is a **covariance** scaled by each dataset's **standard deviation**, so that it can only take on values from -1 to +1. This allows us to compare two pairs of variables and quickly tell if one pair of datasets is more related than an other .

- A **positive correlation** indicates the sets of values change together.
- A **negative correlation** indicates that the variables change in opposite directions.

Intro Stats: Correlation

$$r(x, y) = \frac{cov(x, y)}{\sigma_x \sigma_y}$$

Correlations

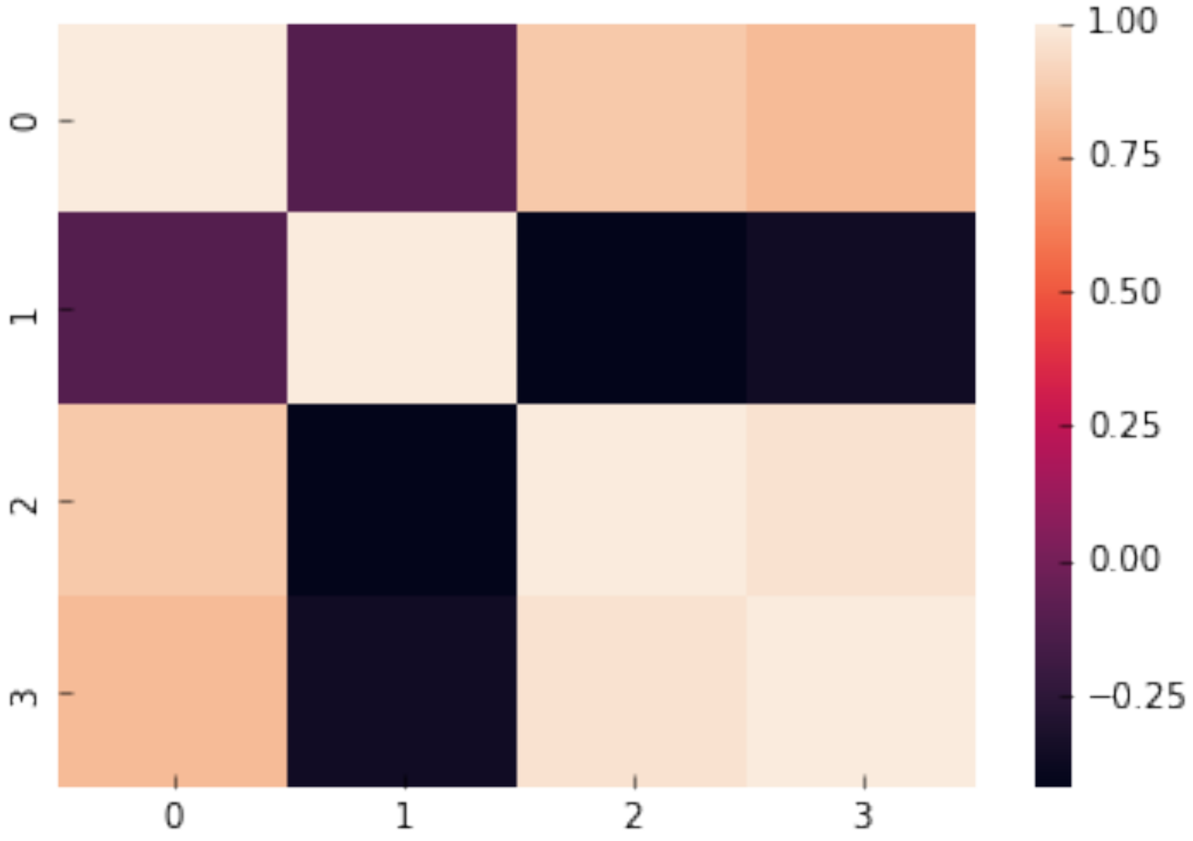


Correlations

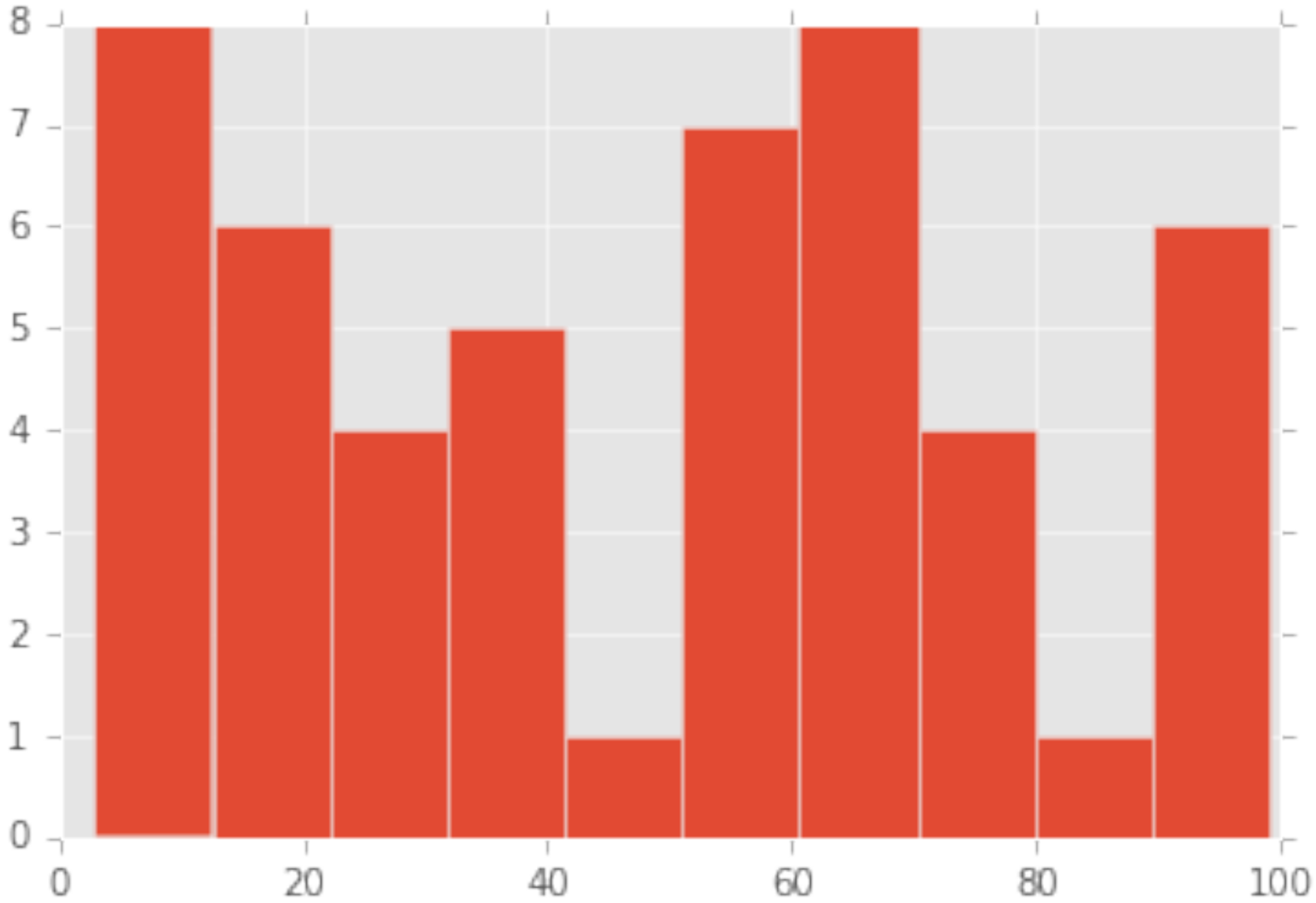
```
series1.corr( series2 )
```

Intro Stats: Correlation

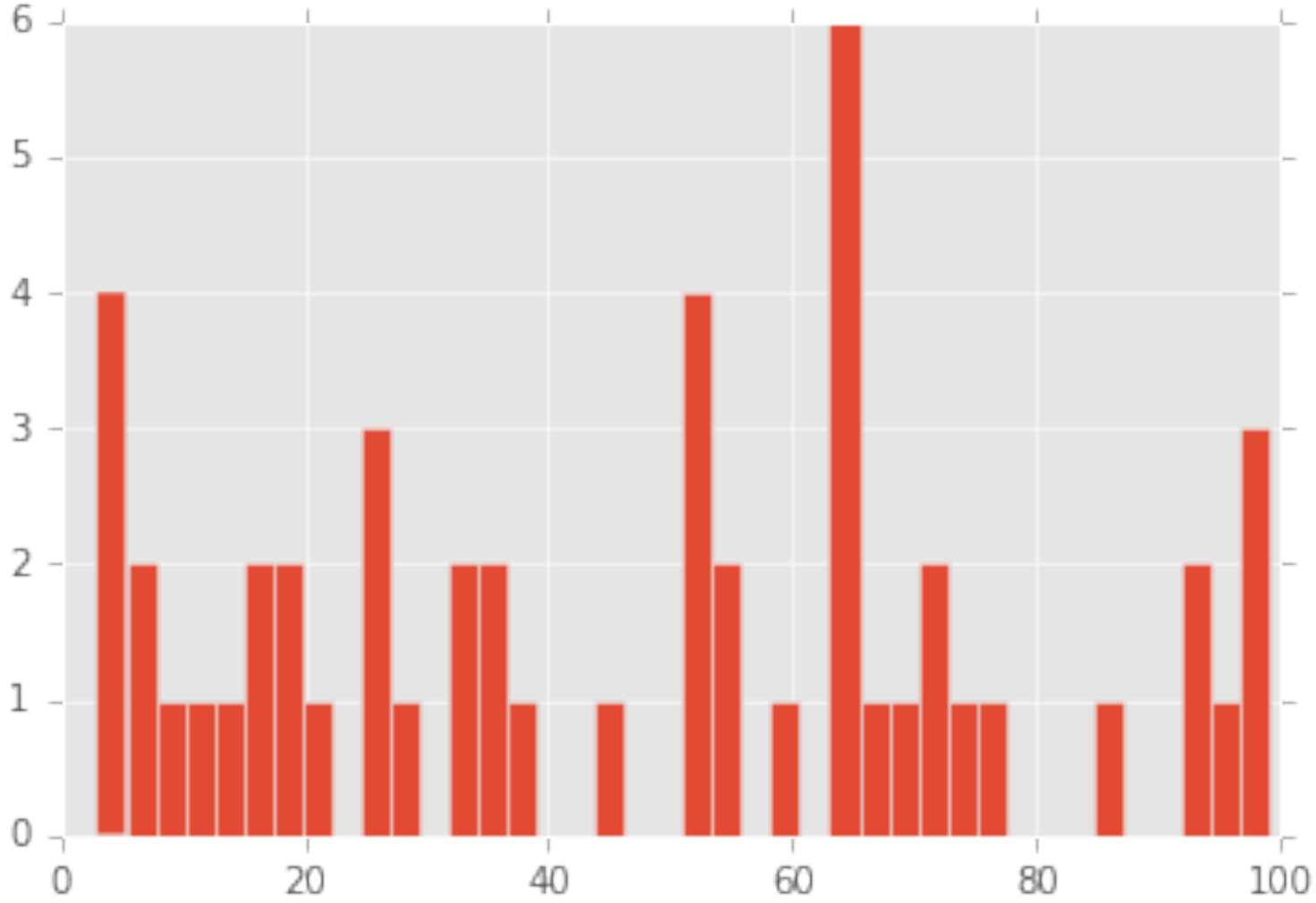
```
sns.heatmap(<data>)
```



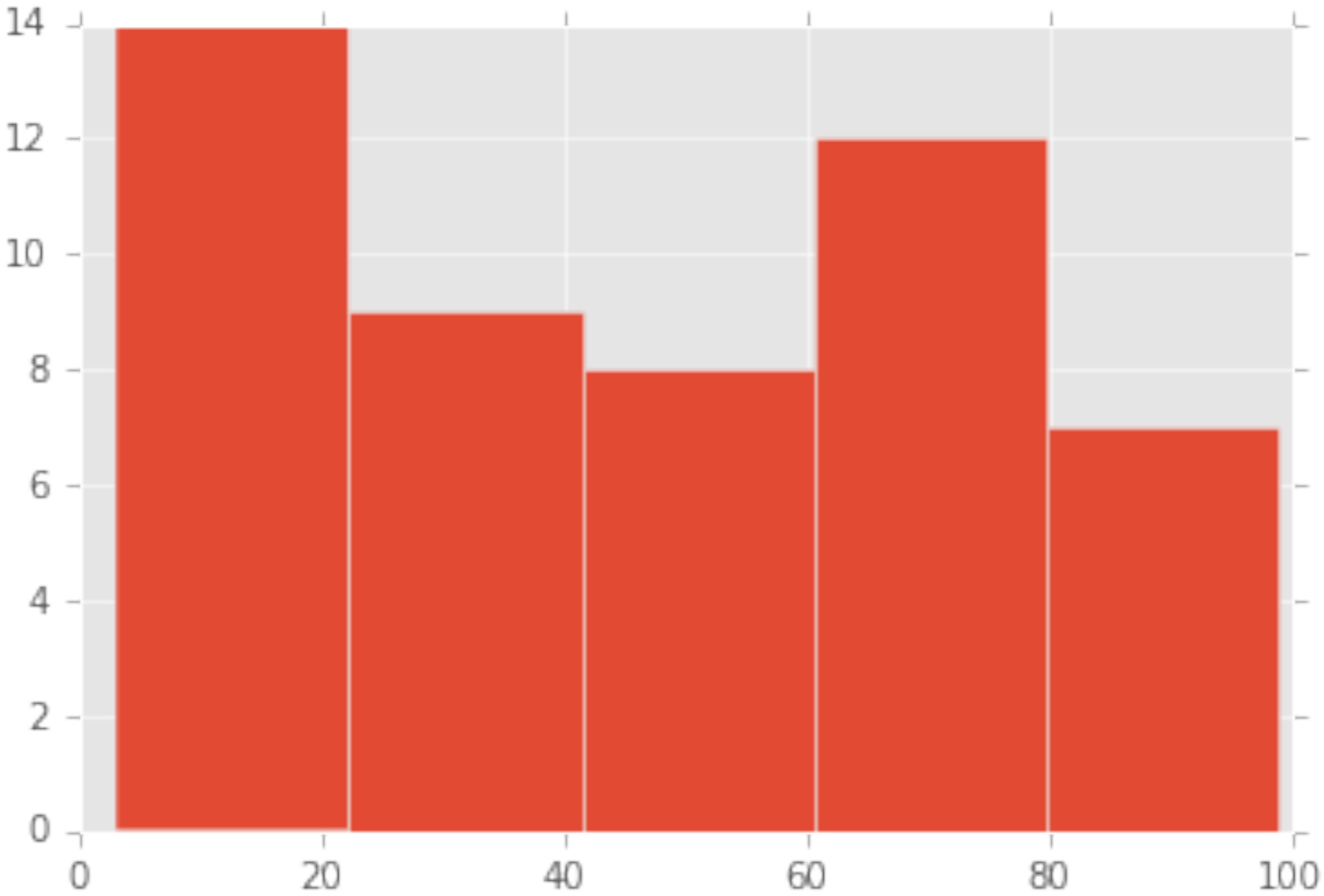
series.hist()



```
series.hist(bins=40)
```

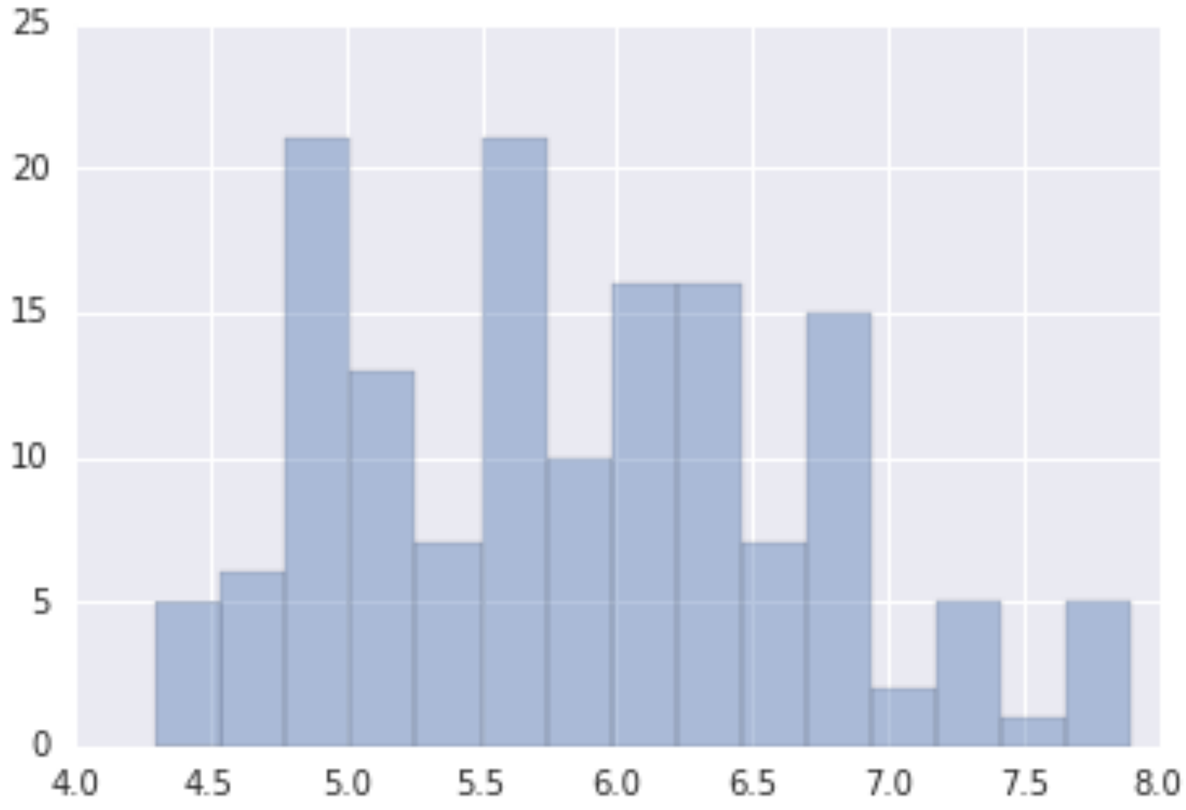


series.hist(bins=5)



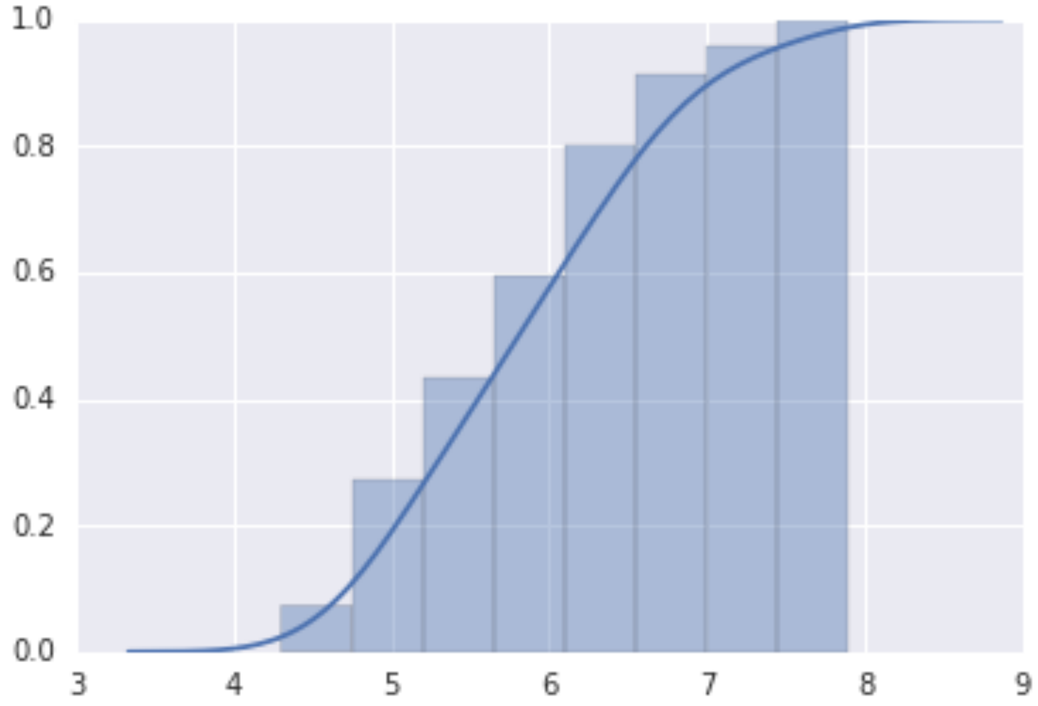
Intro Stats: Histograms

```
sns.distplot(<data>,kde=False,bins=15)
```



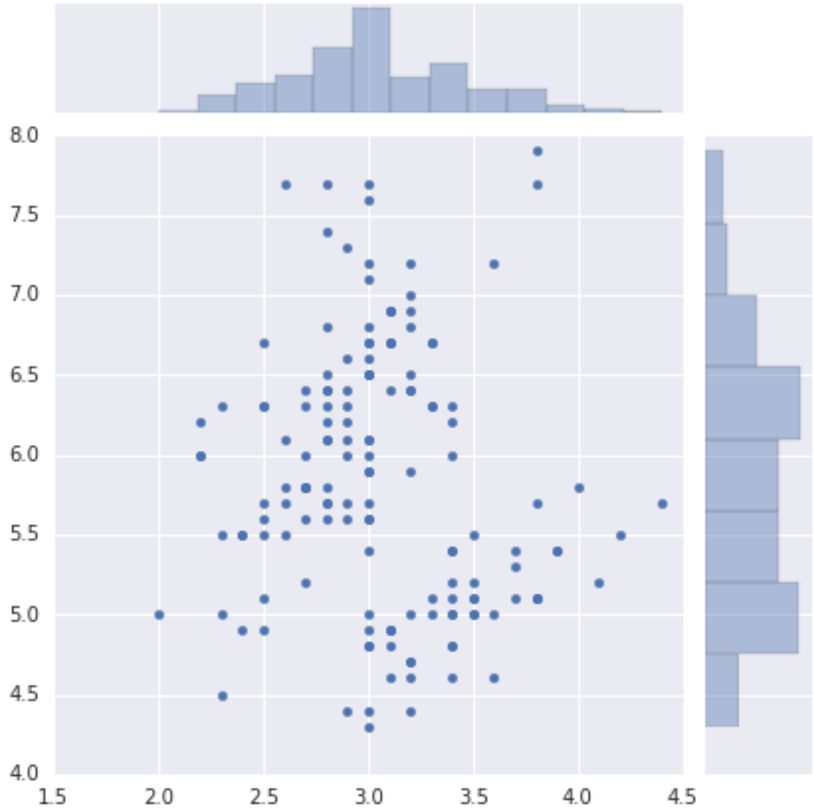
Intro Stats: Histograms

```
sns.distplot(<data>,
hist_kws={"cumulative":True},kde_kws={"cumulative":True})
```



Intro Stats: Scatterplot

```
sns.jointplot(<x>, <y>,stat_func=None)
```



Intro Stats:
Distributions: Normal Distribution

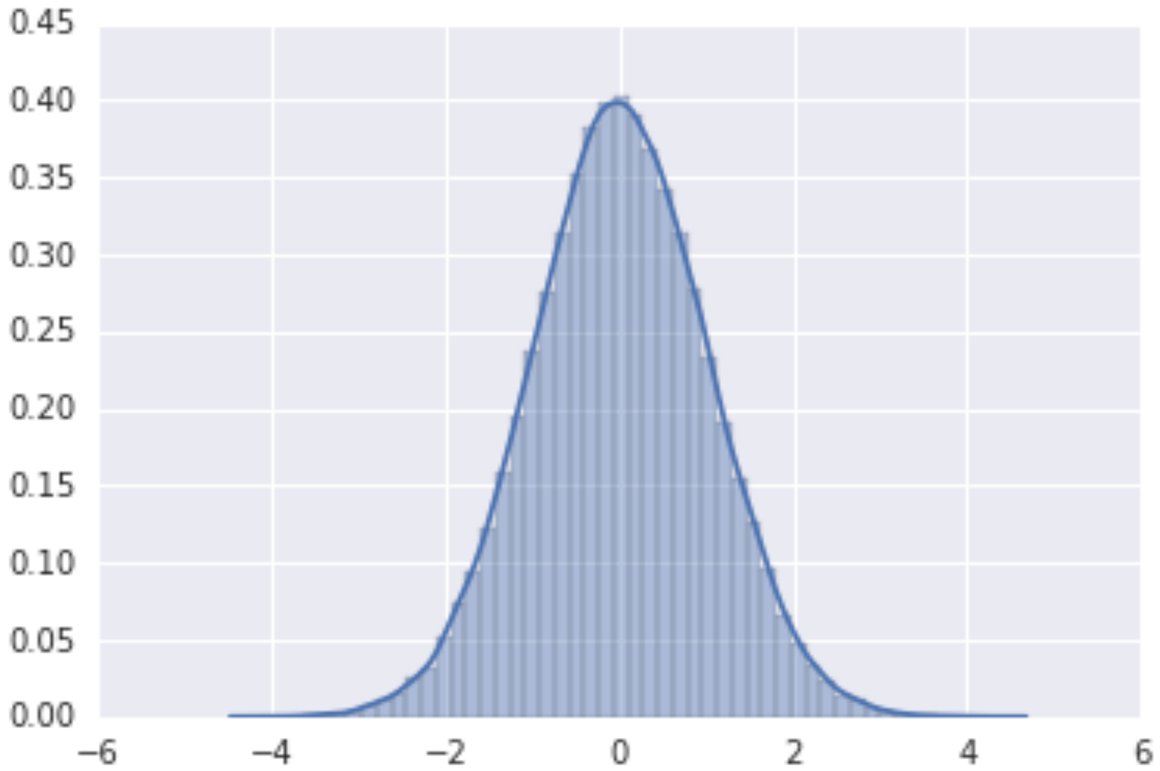
The normal distribution is the most important distribution in all of statistics. It is a continuous probability distribution, unlike the binomial, which is discrete.

It's super important because as datasets become larger and larger and larger, they tend to look more and more like the normal distribution.

The normal distribution is fully specified by 2 parameters, the mean (μ) and the standard deviation (σ).

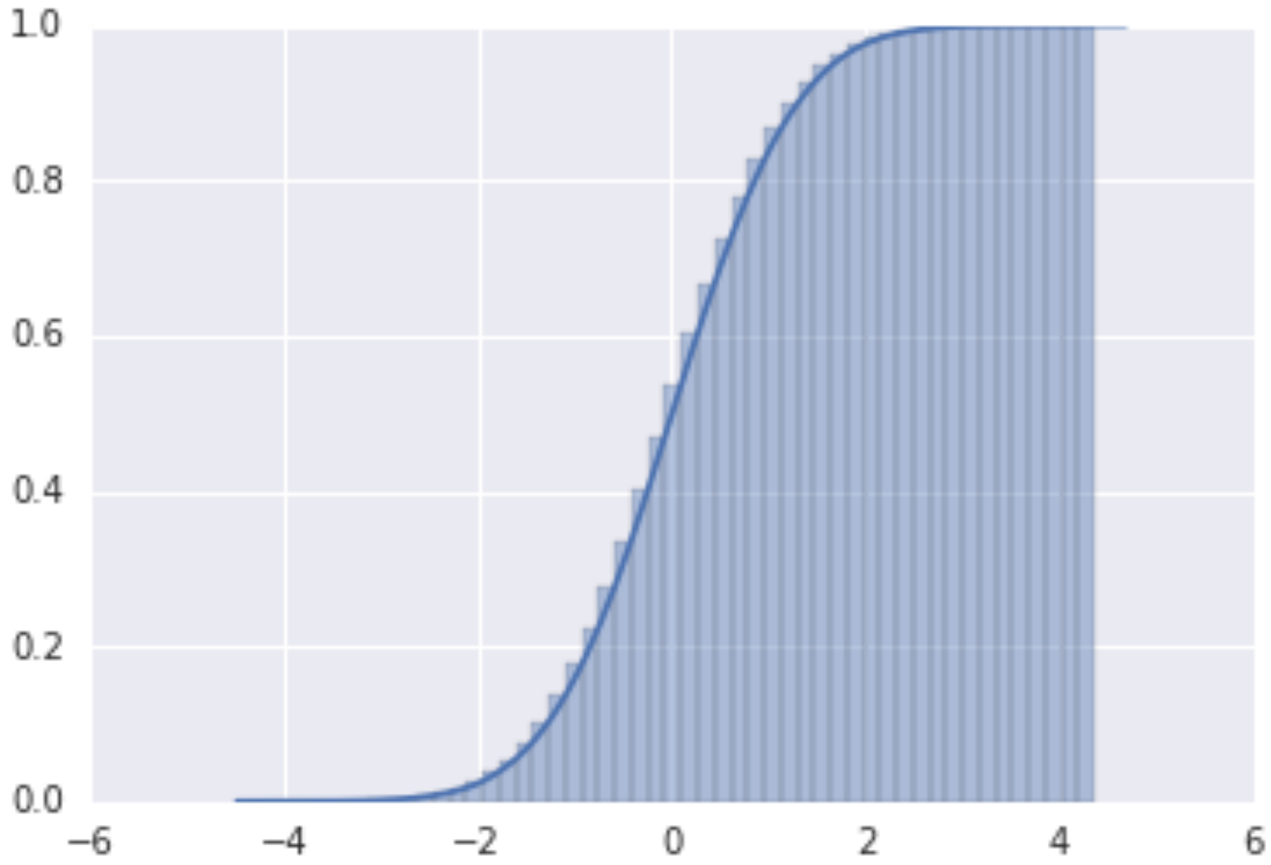
Intro Stats:
Distributions: Normal Distribution

```
mu = 0
sigma = 1
n_samples = 100000
normal_data = np.random.normal(mu, sigma, n_samples)
sns.distplot(normal_data)
```



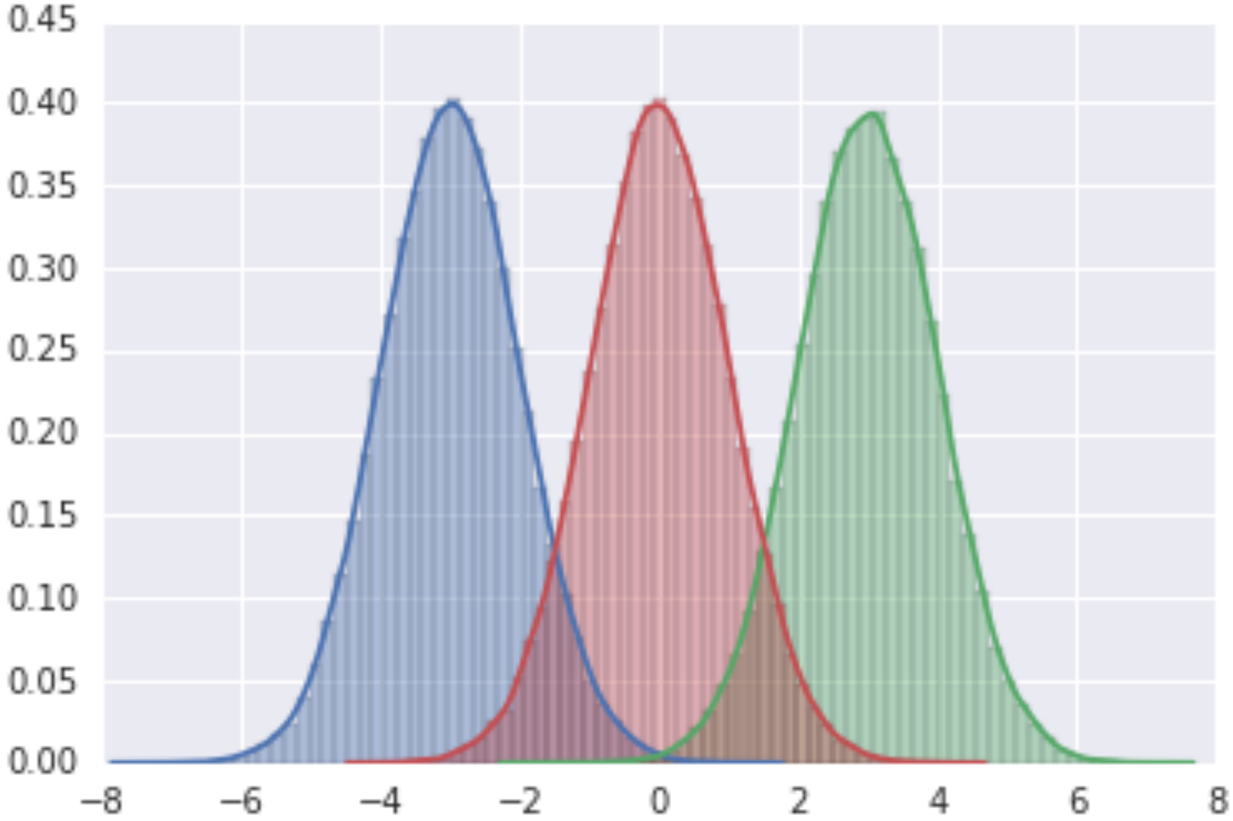
Intro Stats:
Distributions: Normal Distribution

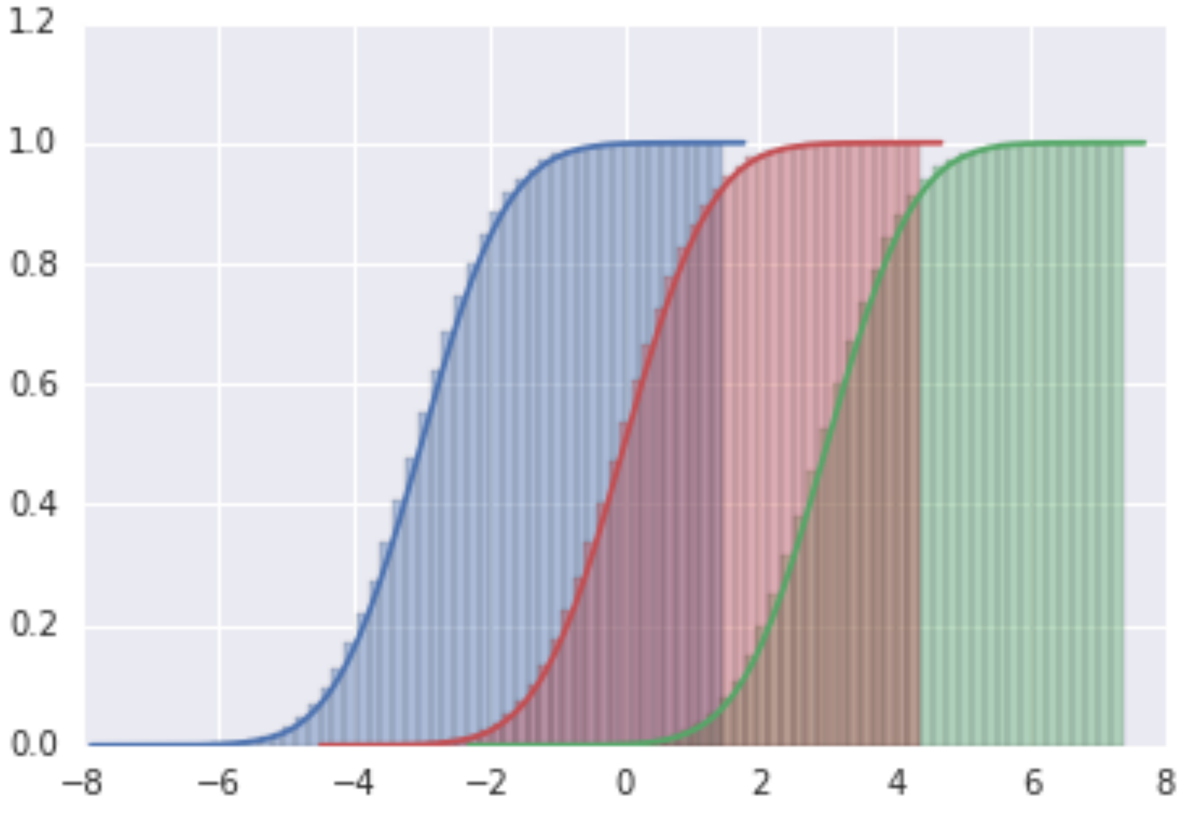
```
sns.distplot(normal_data,
hist_kws={"cumulative":True},kde_kws={"cumulative":True})
```



Intro Stats:
Distributions: Normal Distribution

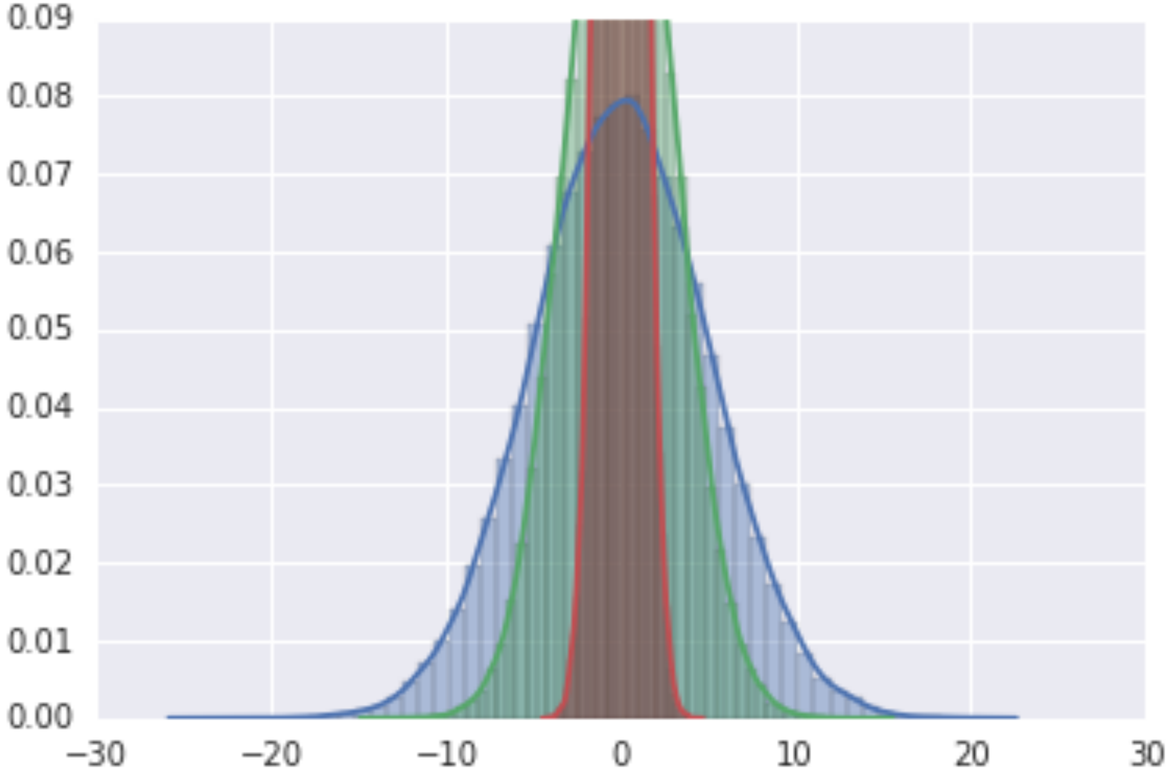
Altering the mean changes the location

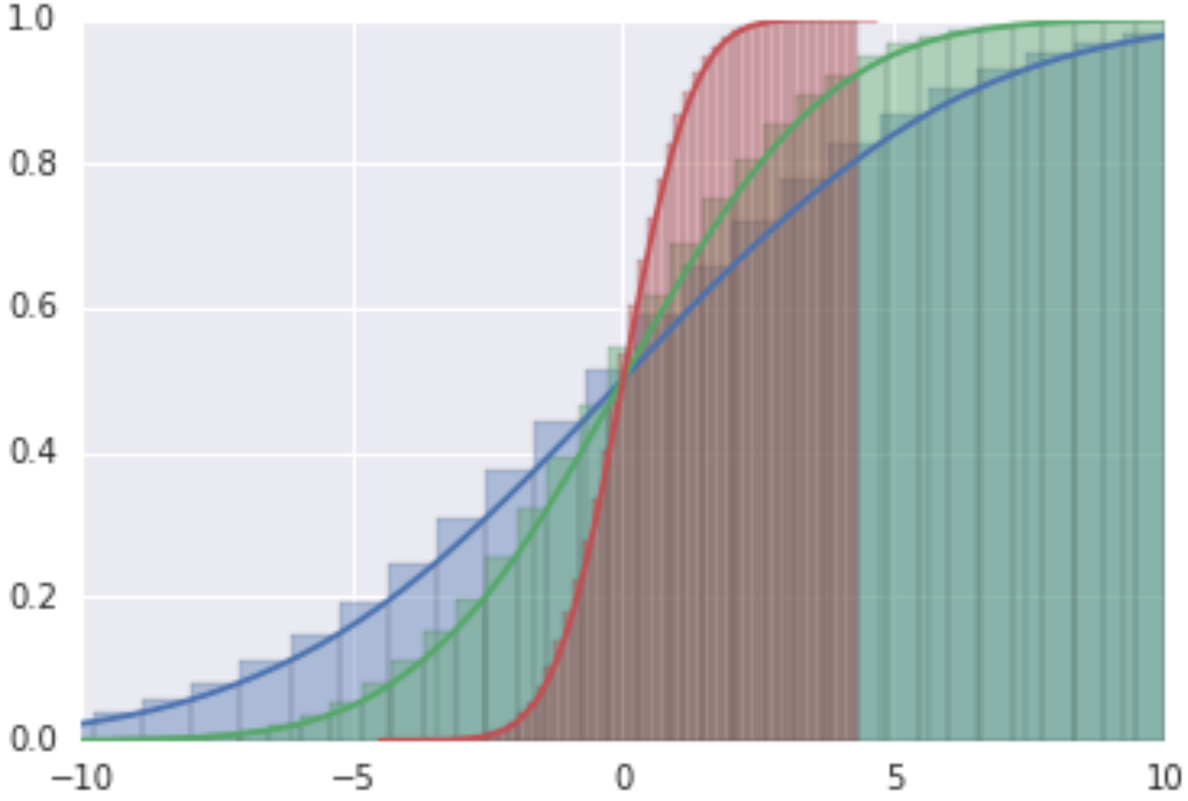




Intro Stats:
Distributions: Normal Distribution

Altering the std. deviation changes the width of the curve.





Intro Stats:
Distributions: Exponential Distribution

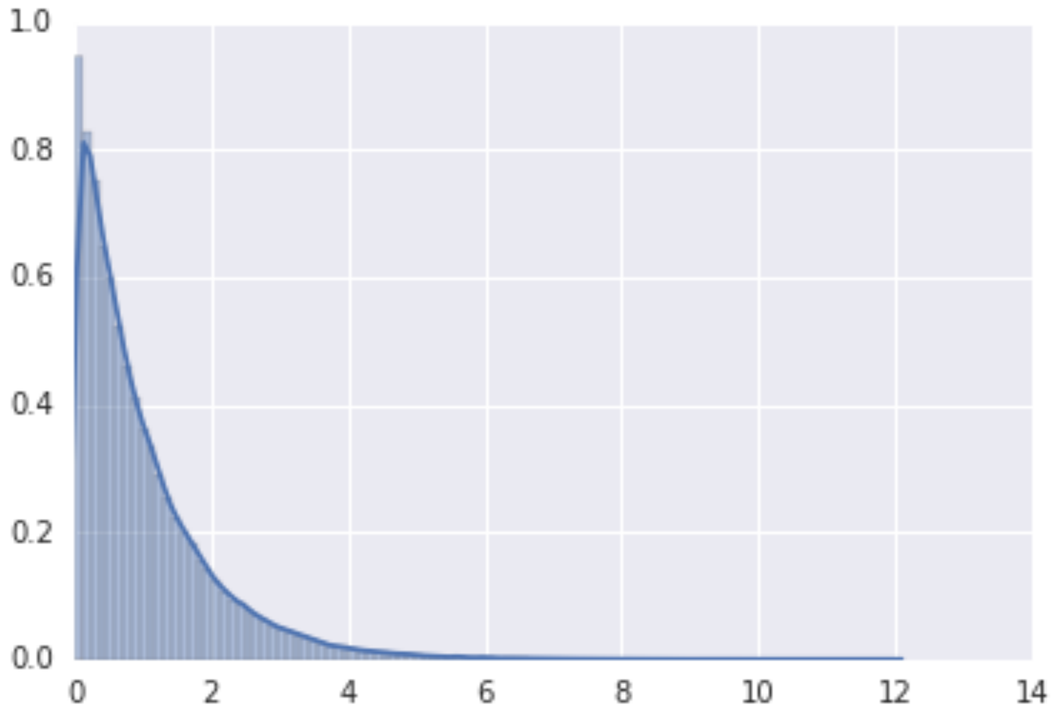
The **exponential distribution** is another very common distribution. In stats-speak, it is the probability distribution that describes the time between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate.

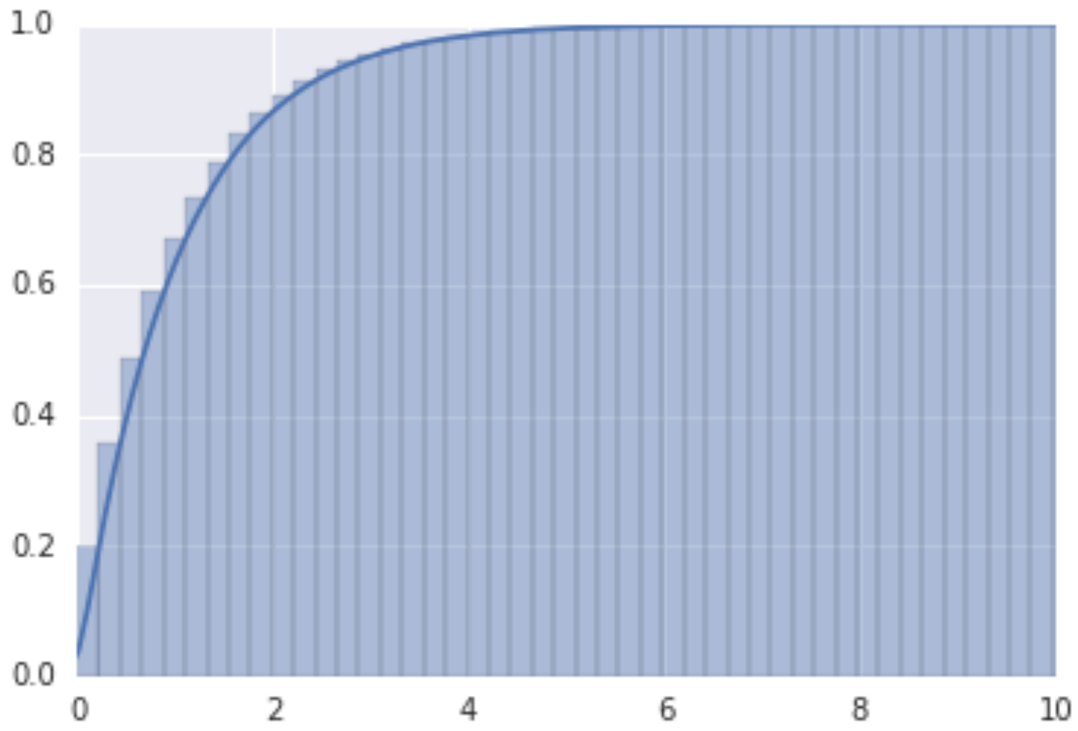
An example exponential distribution would describe the time between airplanes landing and taking off at LaGuardia airport during hours that the airport is operating.

The exponential distribution is interesing because it has the property known as being **memoryless**. This means that knowing the history of the distribution has no effect on being able to predict what will happen next.

Intro Stats:
Distributions: Exponential Distribution

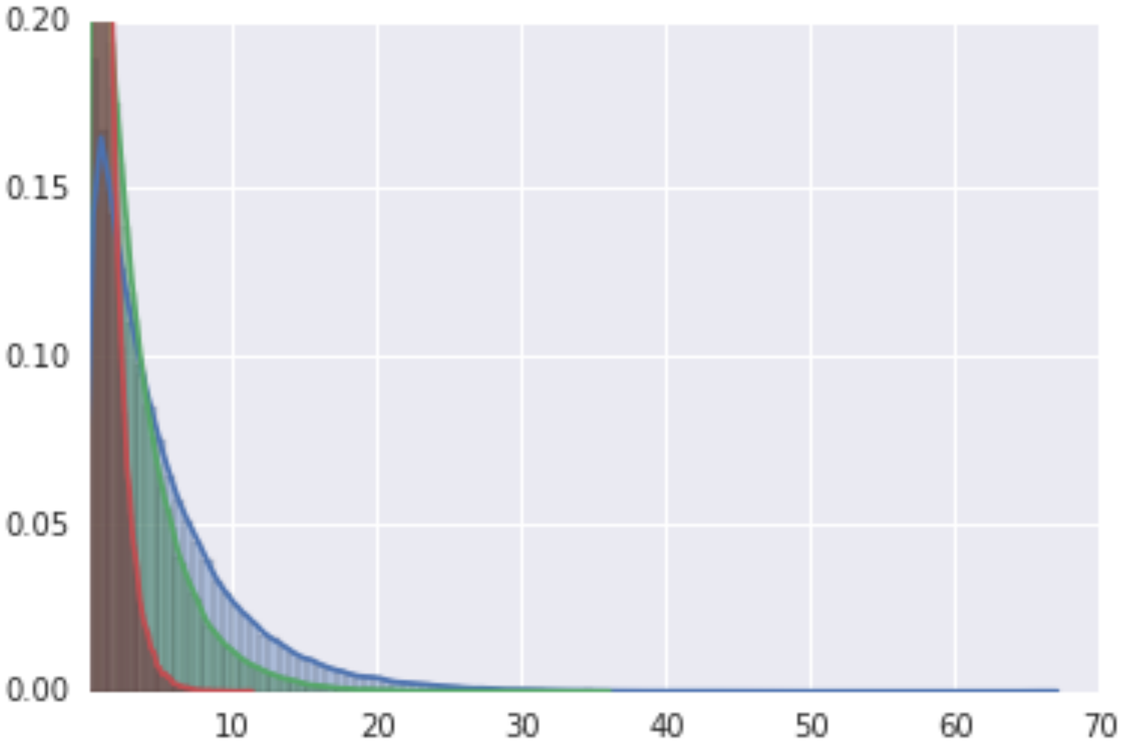
```
e = np.random.exponential(size=n_samples)
sns.distplot(e,bins=100)
plt.xlim(0,)
```

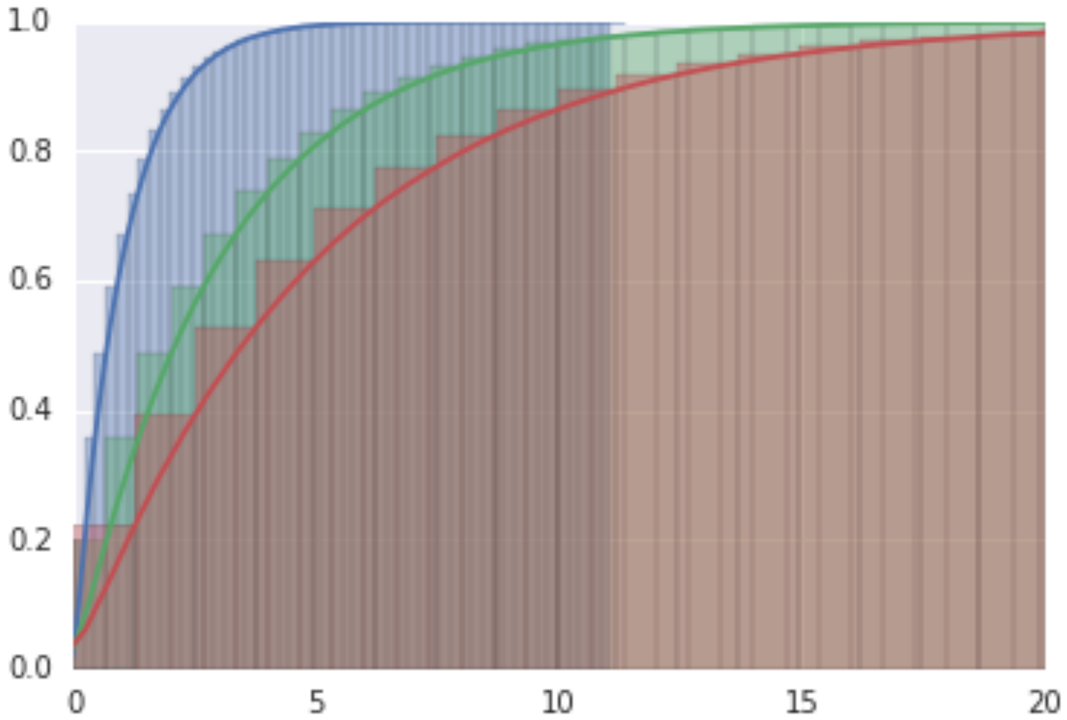




Intro Stats:
Distributions: Normal Distribution

Altering the beta (exponent) changes the location





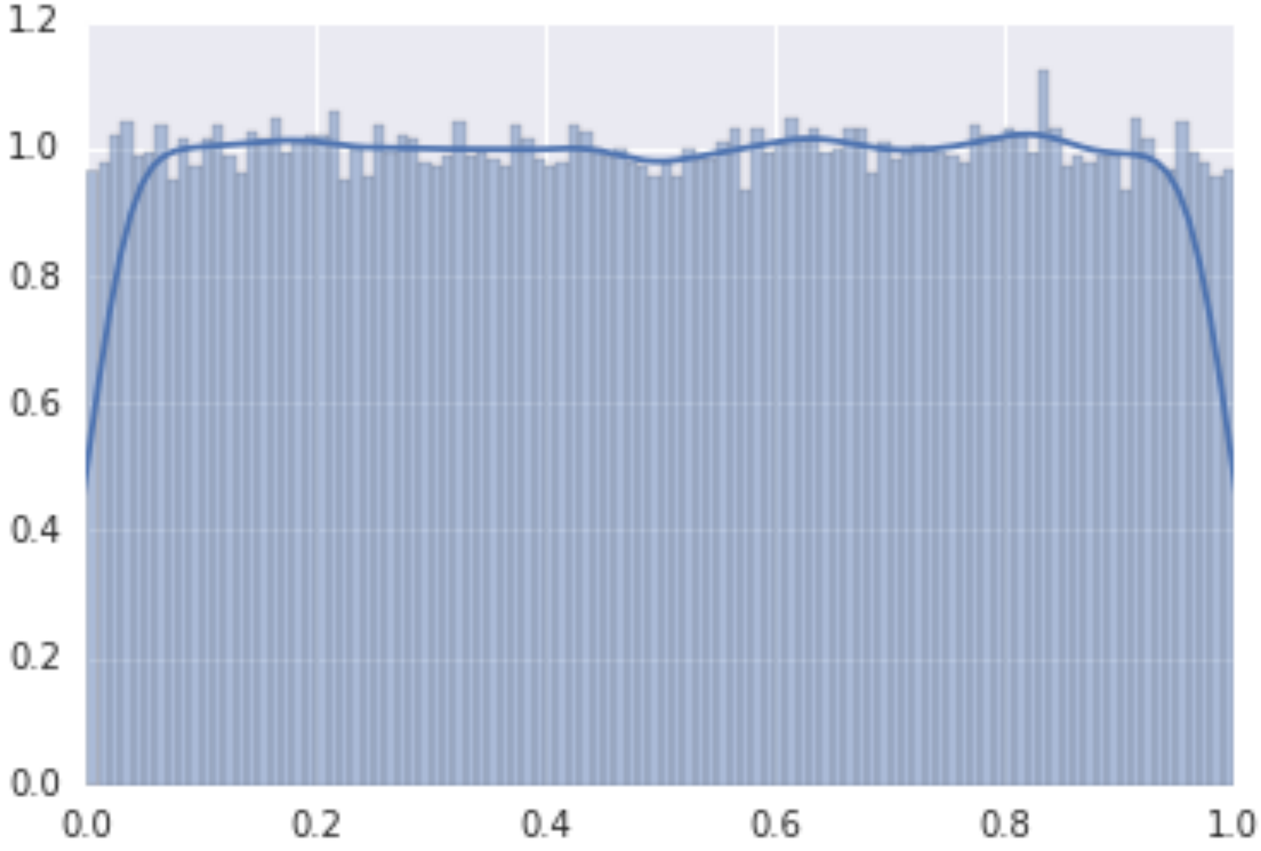
Intro Stats:
Distributions: Uniform Distribution

The final distribution we will talk about is the **uniform distribution**. It simply describes cases where all values have the exact same frequency. It is a useful distribution because it is used for unbiased sampling.

```
uni = np.random.uniform(size=n_samples)
sns.distplot(uni, hist=True,bins=100)
plt.xlim(0,1)
```

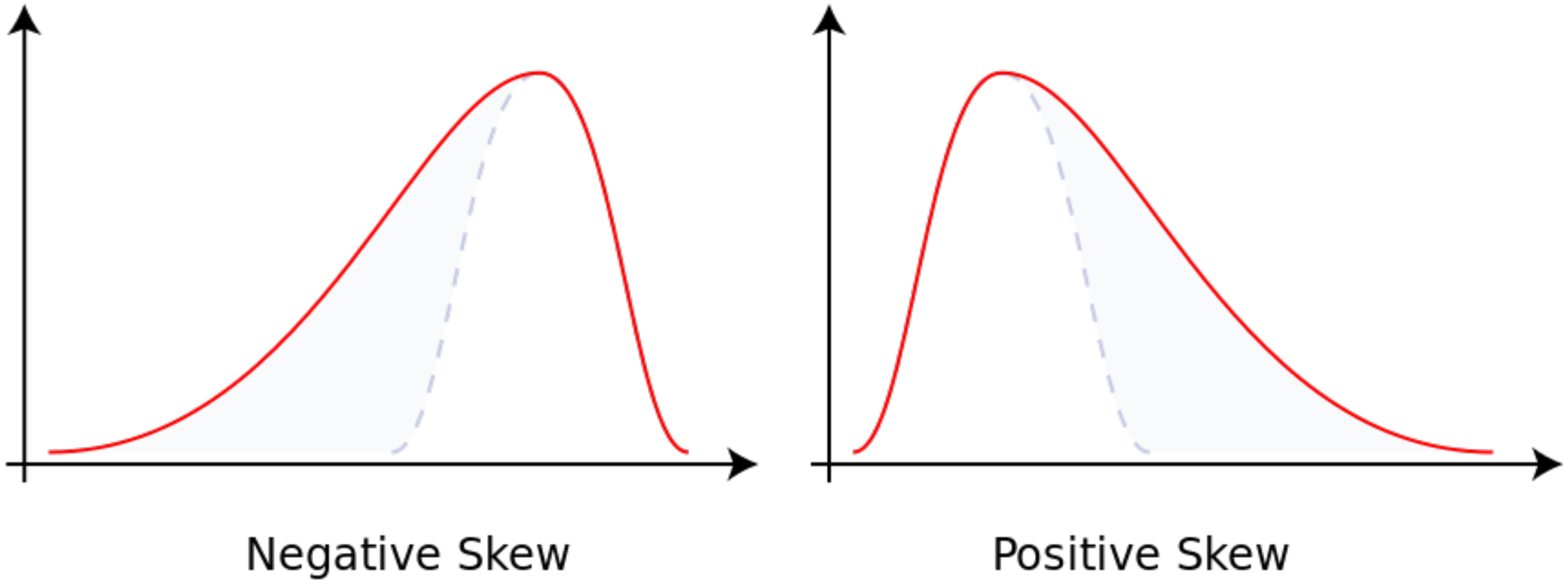
Intro Stats:
Distributions: Uniform Distribution

```
uni = np.random.uniform(size=n_samples)
sns.distplot(uni, hist=True,bins=100)
plt.xlim(0,1)
```



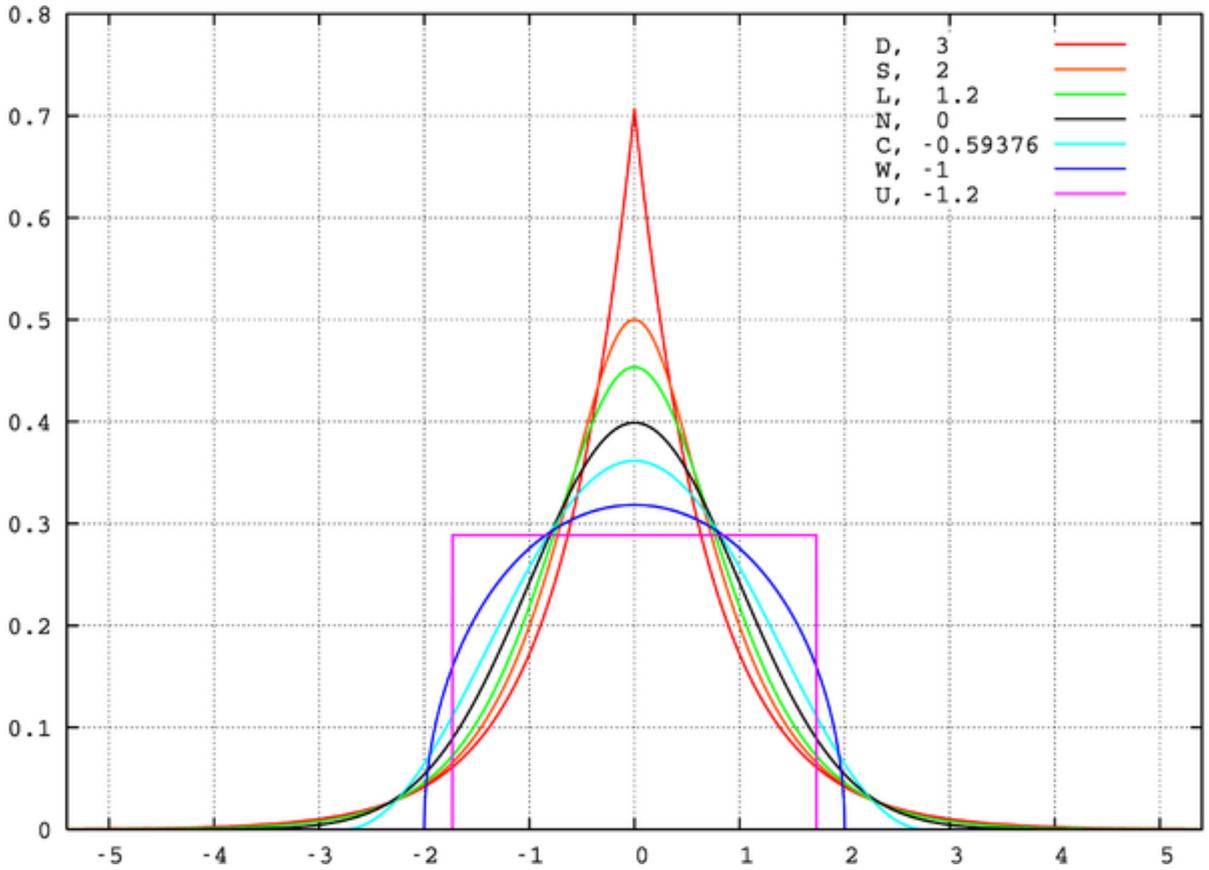
Intro Stats: Skewness

```
uni = np.random.uniform(size=n_samples)
sns.distplot(uni, hist=True,bins=100)
plt.xlim(0,1)
```



Intro Stats: Kurtosis

Kurtosis is a measure of the "peakedness" of a given probability distribution and can only be measured for continuous distributions.



In Class Exercise:

Please complete Day 1: Exploring your Data

Agenda for Tomorrow

- Feature Selection & Engineering
- Math free overview of classification models
- Evaluating Model Performance
- Improving model performance

Questions?