

O'REILLY®

# Security

BUILD BETTER DEFENSES

[oreillysecuritycon.com](http://oreillysecuritycon.com)

#oreillysecurity

## Foundations of Security Data Science *Introductions*

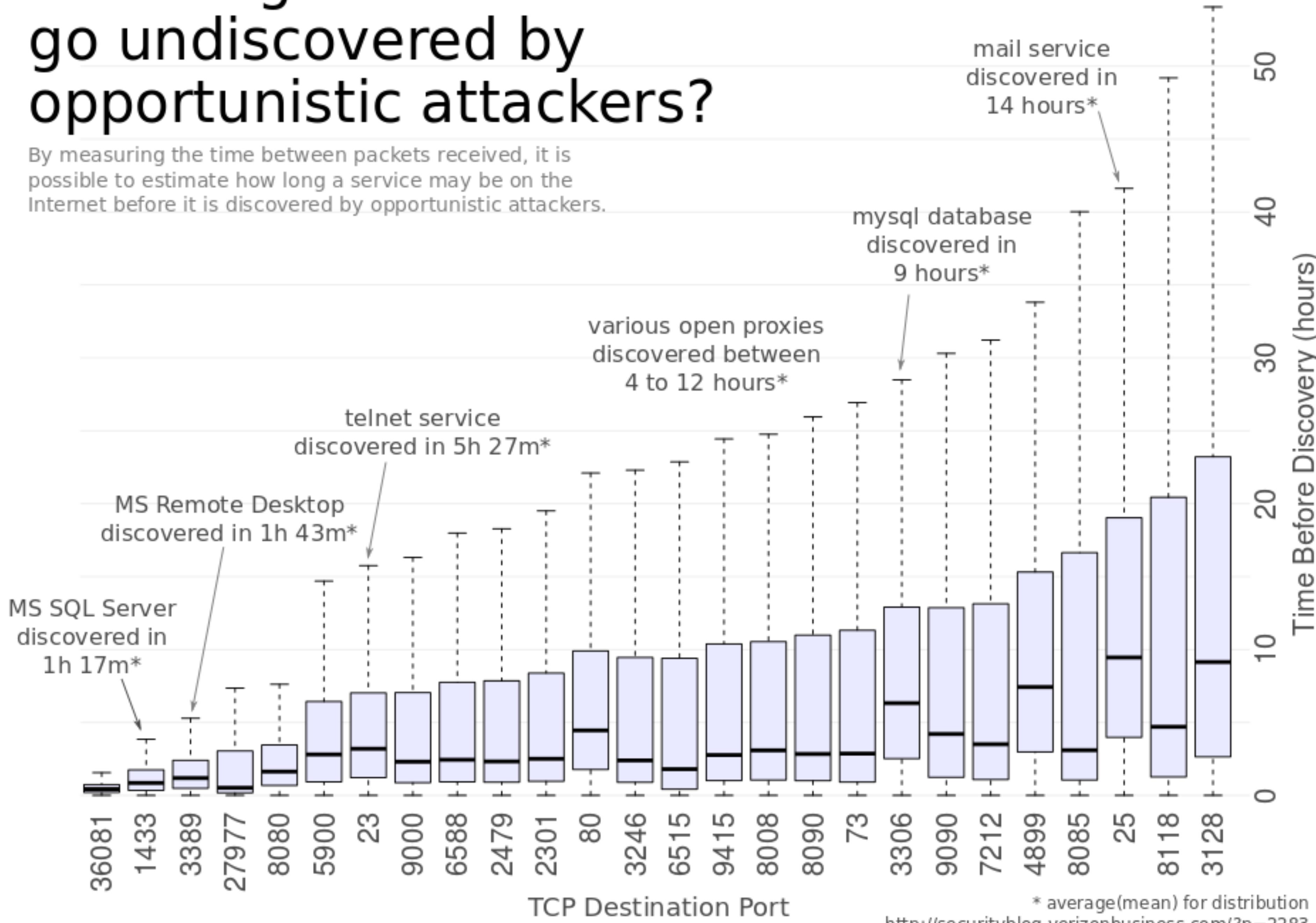
Jay Jacobs  
Charles Givre



2012 Slopegraph

# How long will a service go undiscovered by opportunistic attackers?

By measuring the time between packets received, it is possible to estimate how long a service may be on the Internet before it is discovered by opportunistic attackers.



INDUSTRY	POS INTRUSION	WEB APP ATTACK	INSIDER MISUSE	THEFT/ LOSS	MISC. ERROR	CRIME-WARE	PAYMENT CARD SKIMMER	DENIAL OF SERVICE	CYBER ESPIONAGE	EVERYTHING ELSE
Accommodation [72]	75%	1%	8%	1%	1%	1%	<1%	10%		4%
Administrative [56]		8%	27%	12%	43%	1%		1%	1%	7%
Construction [23]	7%		13%	13%	7%	33%			13%	13%
Education [61]	<1%	19%	8%	15%	20%	6%	<1%	6%	2%	22%
Entertainment [71]	7%	22%	10%	7%	12%	2%	2%	32%		5%
Finance [52]	<1%	27%	7%	3%	5%	4%	22%	26%	<1%	6%
Healthcare [62]	9%	3%	15%	46%	12%	3%	<1%	2%	<1%	10%
Information [51]	<1%	41%	1%	1%	1%	31%	<1%	9%	1%	16%
Management [55]		11%	6%	6%	6%		11%	44%	11%	6%
Manufacturing [31,32,33]		14%	8%	4%	2%	9%		24%	30%	9%
Mining [21]			25%	10%	5%	5%	5%	5%	40%	5%
Professional [54]	<1%	9%	6%	4%	3%	3%		37%	29%	8%
Public [92]		<1%	24%	19%	34%	21%		<1%	<1%	2%
Real Estate [53]		10%	37%	13%	20%	7%			3%	10%
Retail [44,45]	31%	10%	4%	2%	2%	2%	6%	33%	<1%	10%
Trade [42]	6%	30%	6%	6%	9%	9%	3%	3%		27%
Transportation [48,49]		15%	16%	7%	6%	15%	5%	3%	24%	8%
Utilities [22]		38%	3%	1%	2%	31%		14%	7%	3%
Other [81]	1%	29%	13%	13%	10%	3%		9%	6%	17%

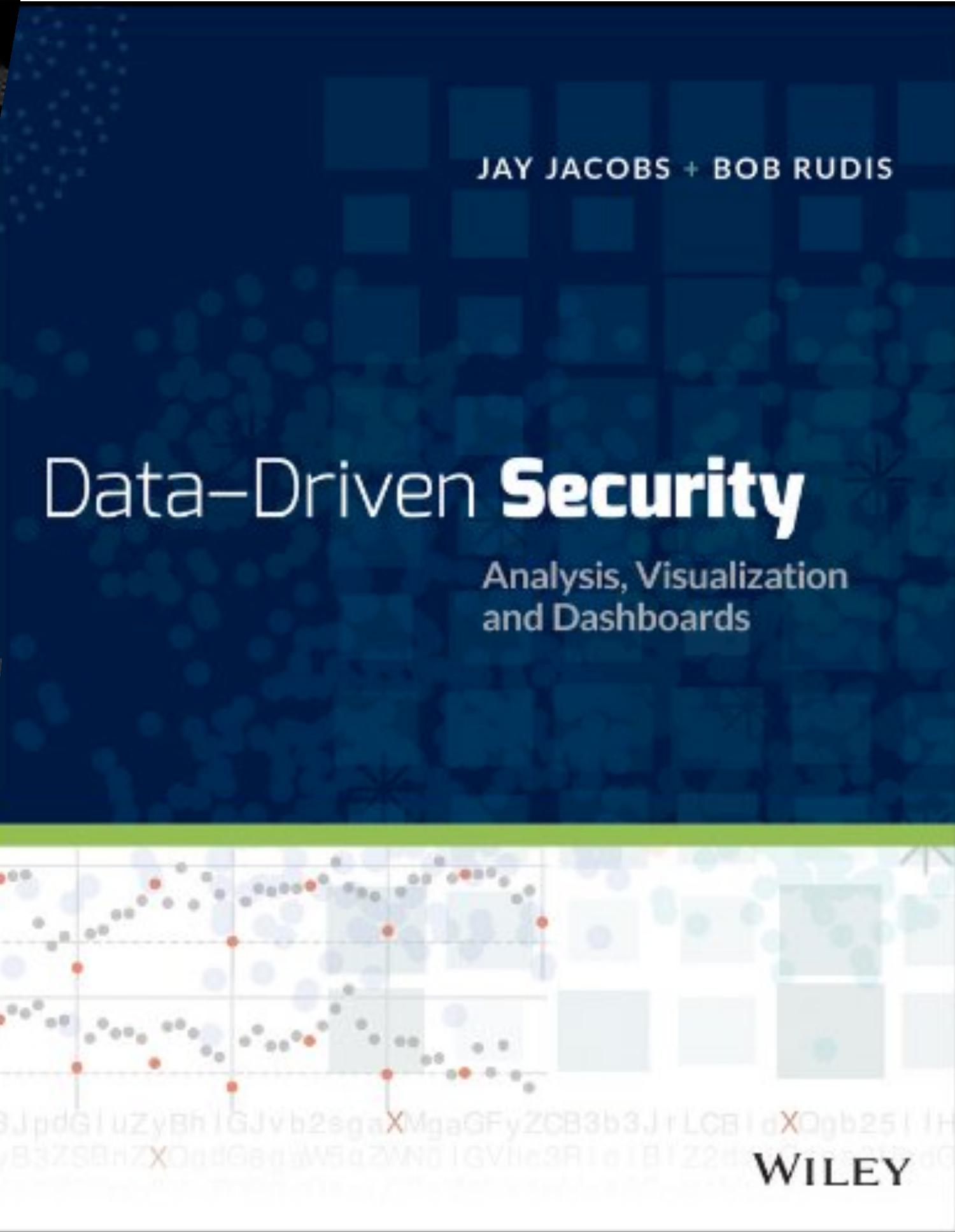
2014  
DBIR

**< Insert Movie >**

(shared slides cannot have movies larger than 50k)

# Jay Jacobs

- Sr. Data Scientist at BitSight
- 4 years on Verizon DBIR
- Author, Data-Driven Security
- Degree in Applied Statistics
- President, Society of Information Risk Analysis (SIRA)

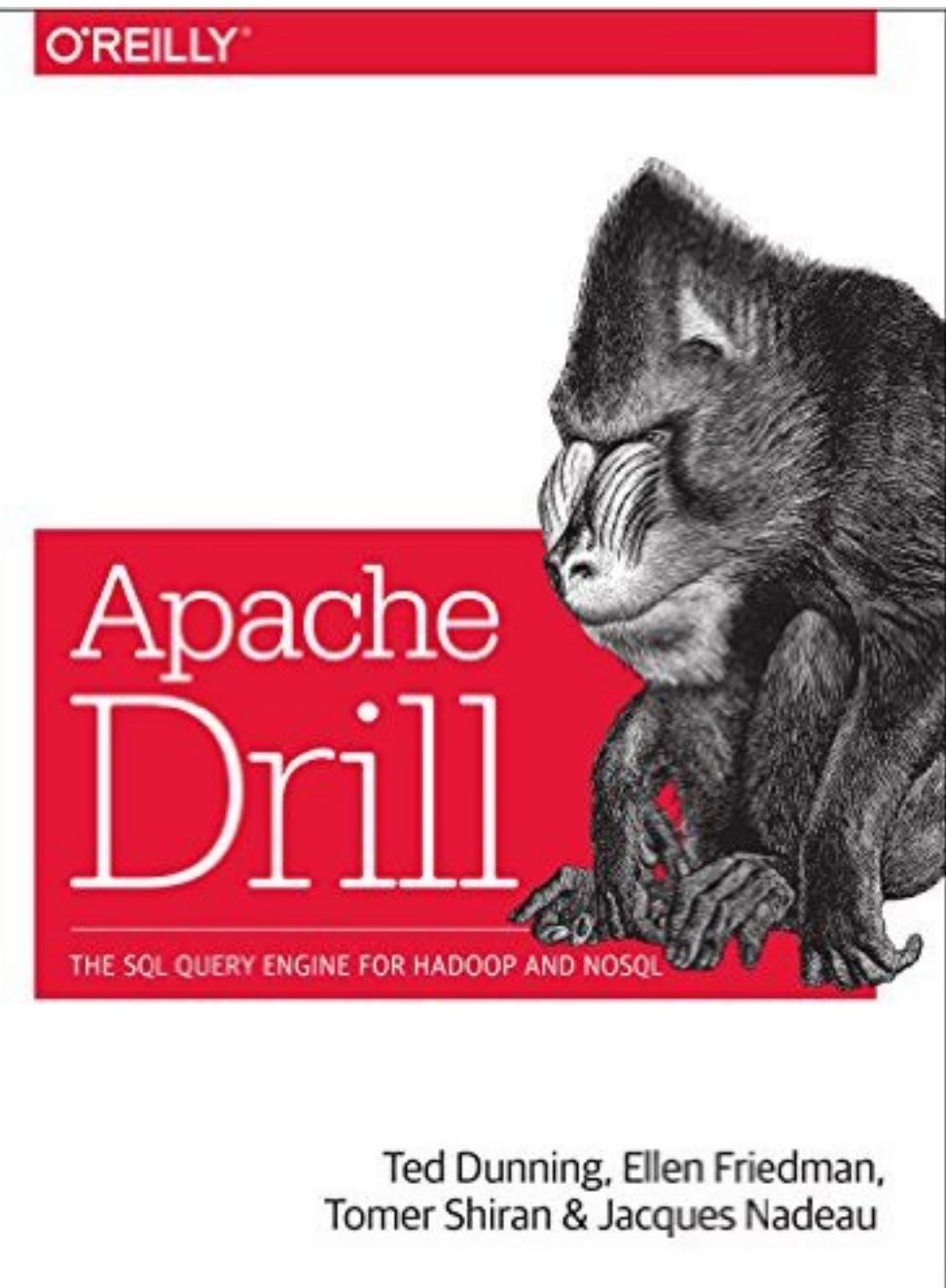




- Data Scientist/Solutions Architect at Orbital Insight
- 7 years Data Scientist @ Booz Allen Hamilton
- 5 years @ CIA
- Committer to Apache Drill, Working on Drill book
- Degrees in Computer Science



Orbital Insight



Ted Dunning, Ellen Friedman,  
Tomer Shiran & Jacques Nadeau

# Roles/Skills within Data Science

- **Domain Expertise**

Setting and maintaining purpose in analysis

- **Data Engineering**

Being able to prepare, store and maintain data

- **Programming**

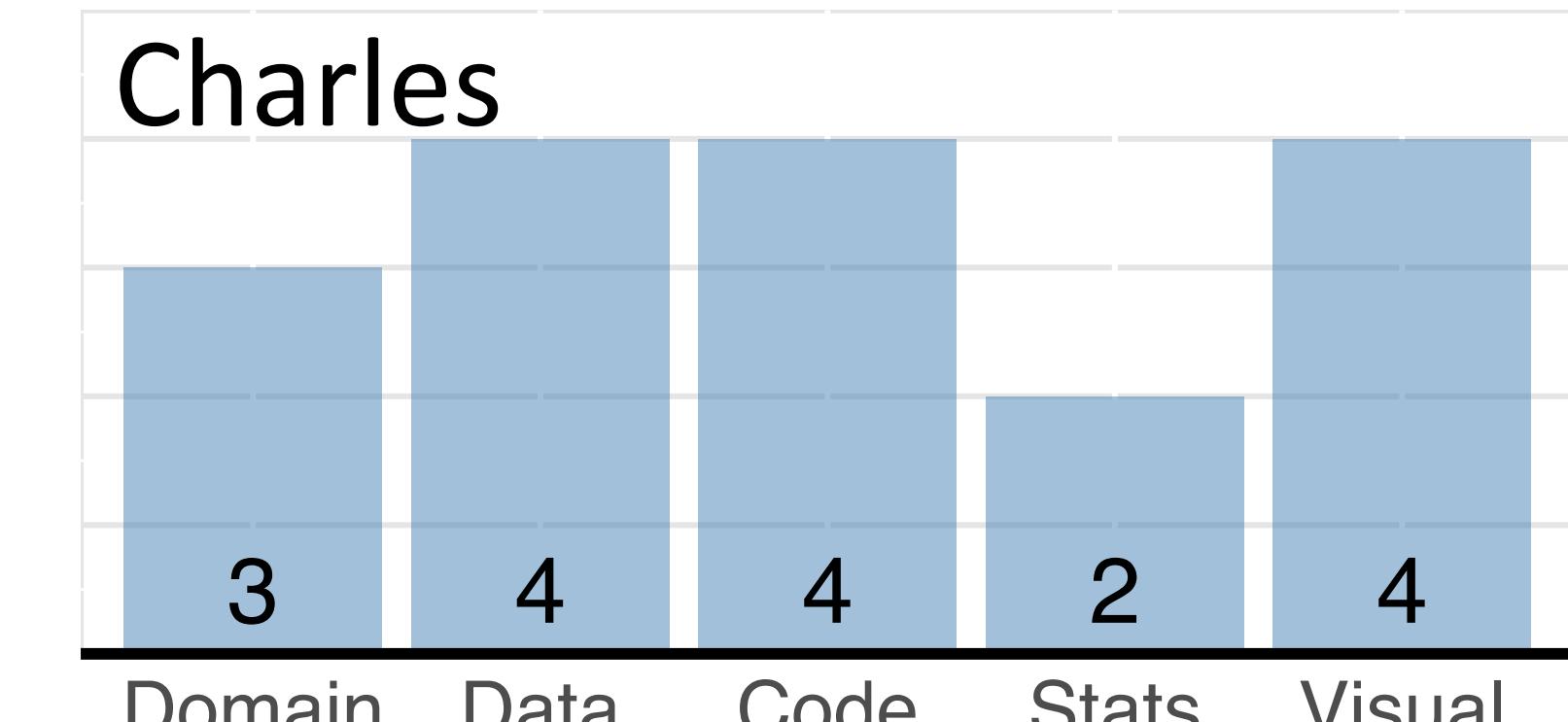
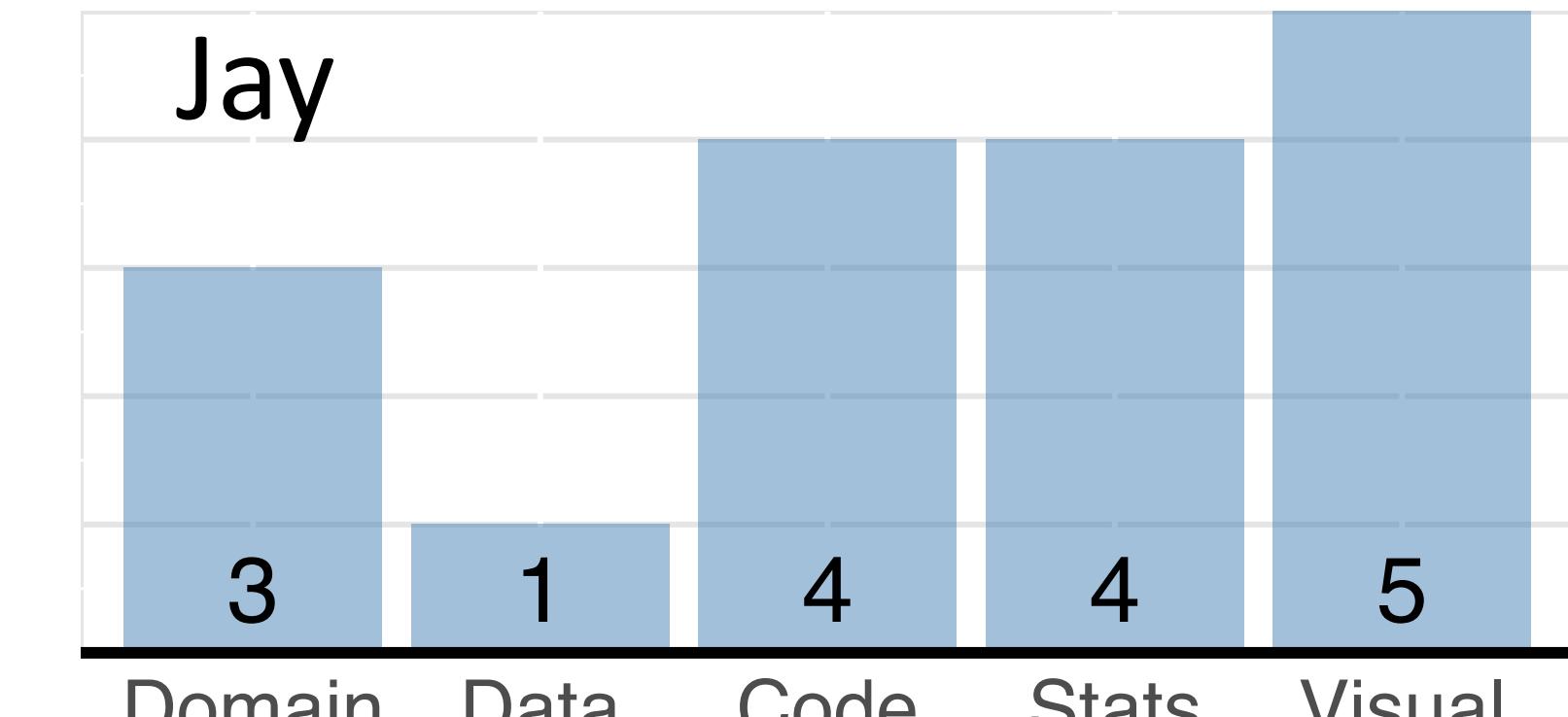
The glue that connects everything

- **Statistics/Machine Learning**

To learn from the data (or not be fooled by data)

- **Visualization**

Communicating the results effectively



# And you?

- Name
- Where do you work?
- What do you do?
- What projects do you have?
- What you are hoping to learn from this training?

# This Course

## Day One

- **Intro (this)**
  - Lab: working environment
- **Exploratory Data Analysis**
  - Lab: exploratory analysis of “dailybots”
- Lunch
- **Data Visualization**
  - Lab: full exploratory analysis of “dailybots”

## Day Two

- **ML and Modeling Overview**
- **Unsupervised Learning**
  - Lab: URL clustering
- Lunch
- **Supervised Learning**
  - Lab: DGA Classification
- Review

## GRIFFON



<https://github.com/gtkcyber/griffon-vm>

# About the VM



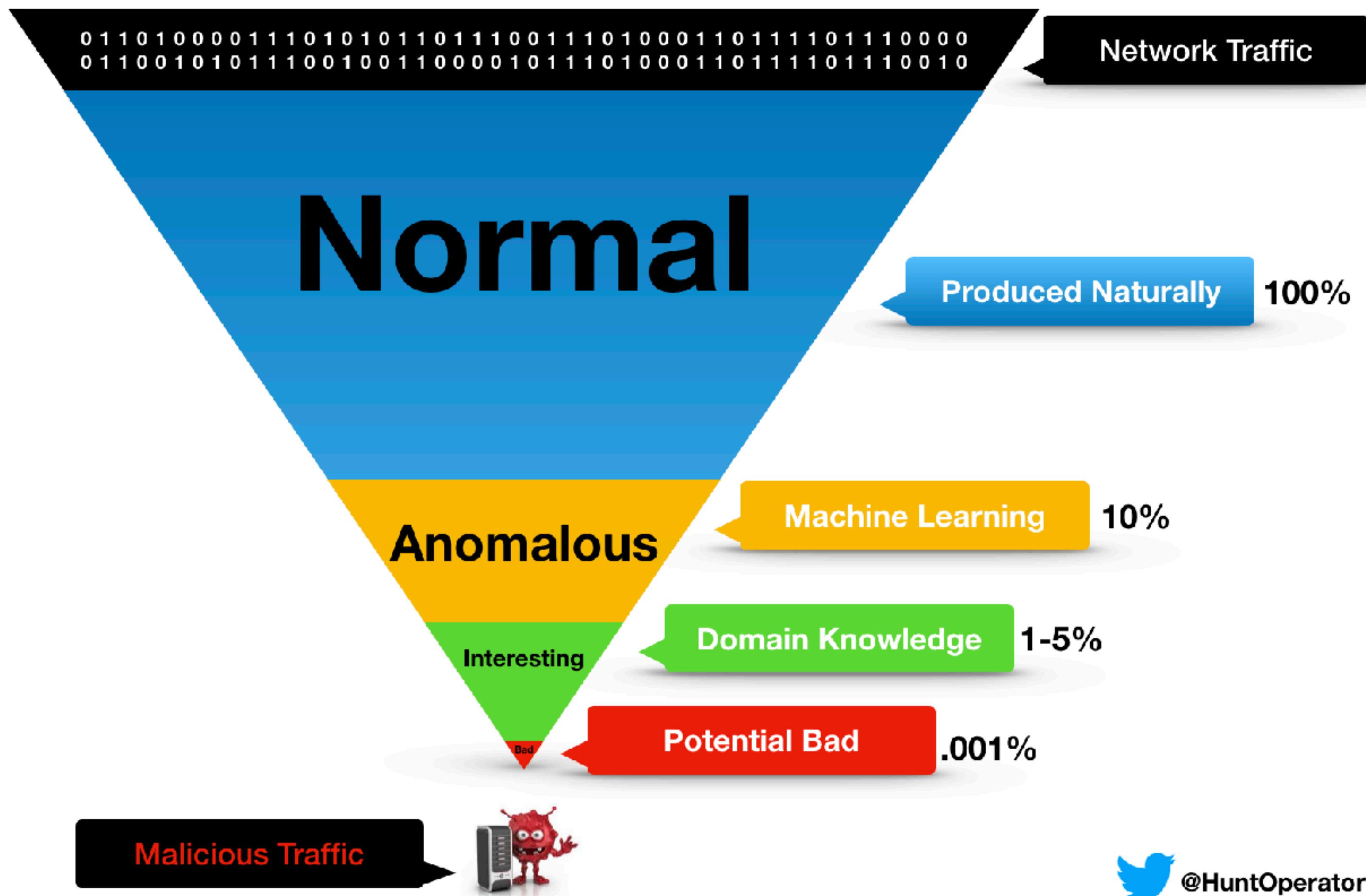
# What are we Talking About?

- **Statistics**
- **Machine Learning**
- **Data Science**
- **Big Data**

# Where is the I.T. / Security Data Science?

- Spam Detection  
From “SpamAssassin” to “Naive Bayes”
- Malware Identification
- SIEM / Intrusion Detection Systems?
- Phishing detection
- Network Proxy Filters (URL, domain, C&C detection)
- Risk Analysis (predictive analytics)
- Audit and 3rd party assessments
- and more...

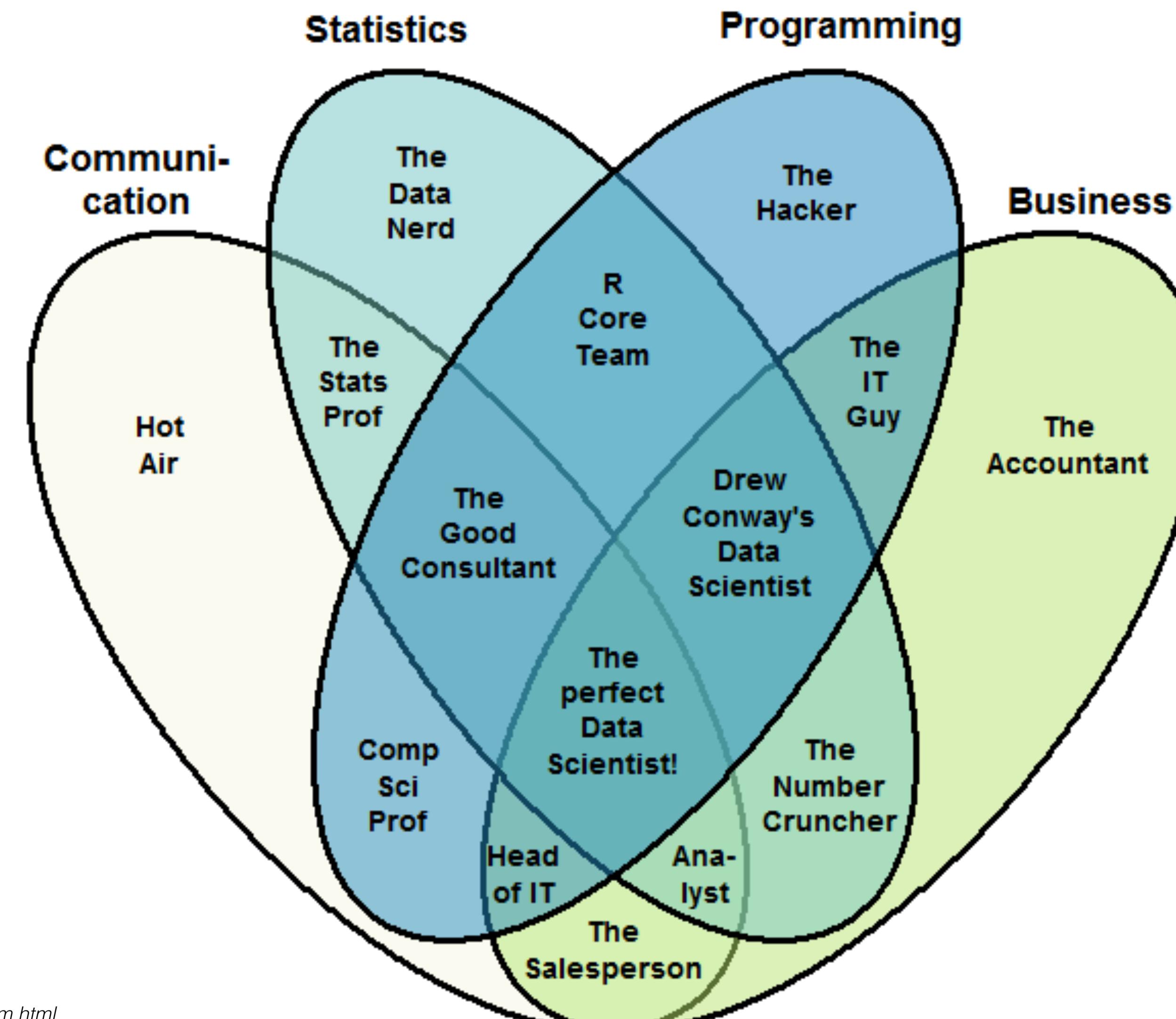
# Data Science in Practice



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

# Data Scientist

## The Data Scientist Venn Diagram



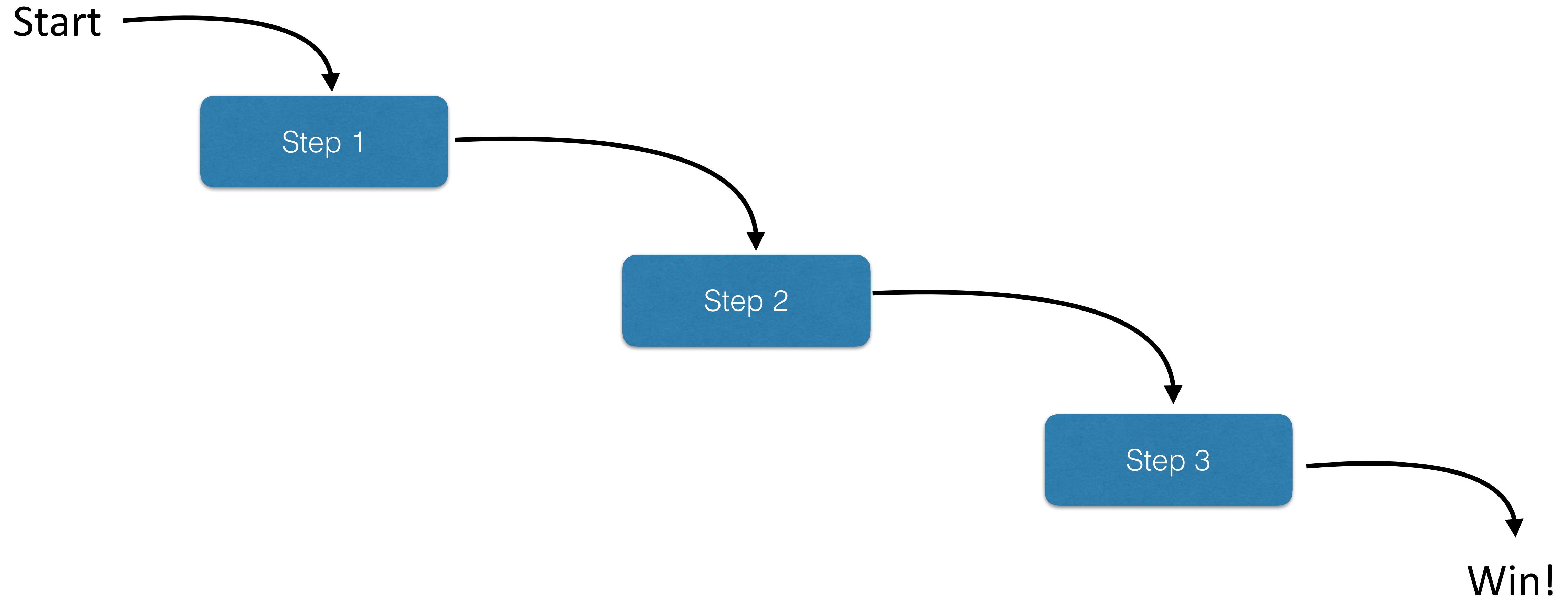
# Data Science Workflow



80-90% of your time is spent data “wrangling”  
(obtaining data, cleaning, fixing, preparing, etc)



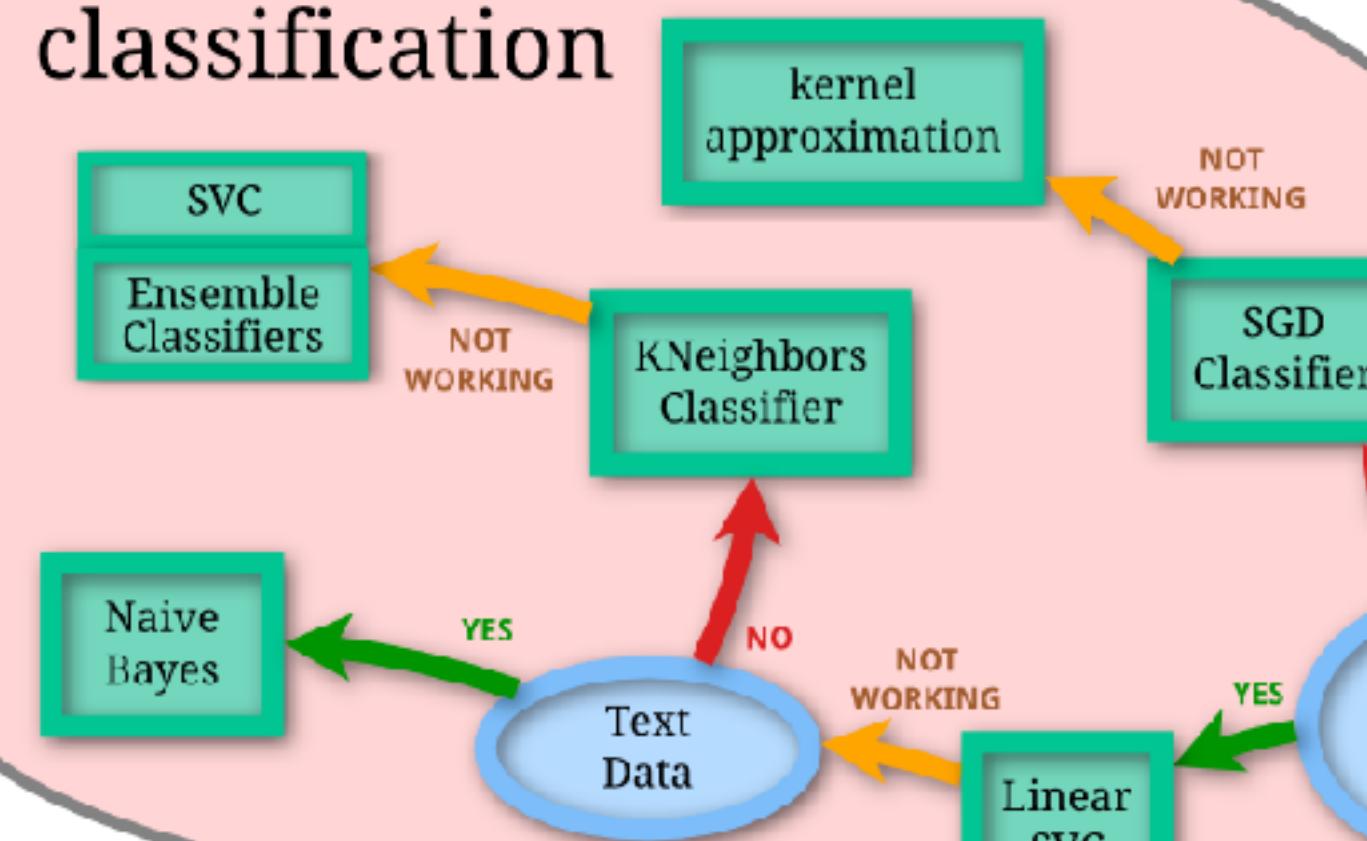
# What Data Science is Not



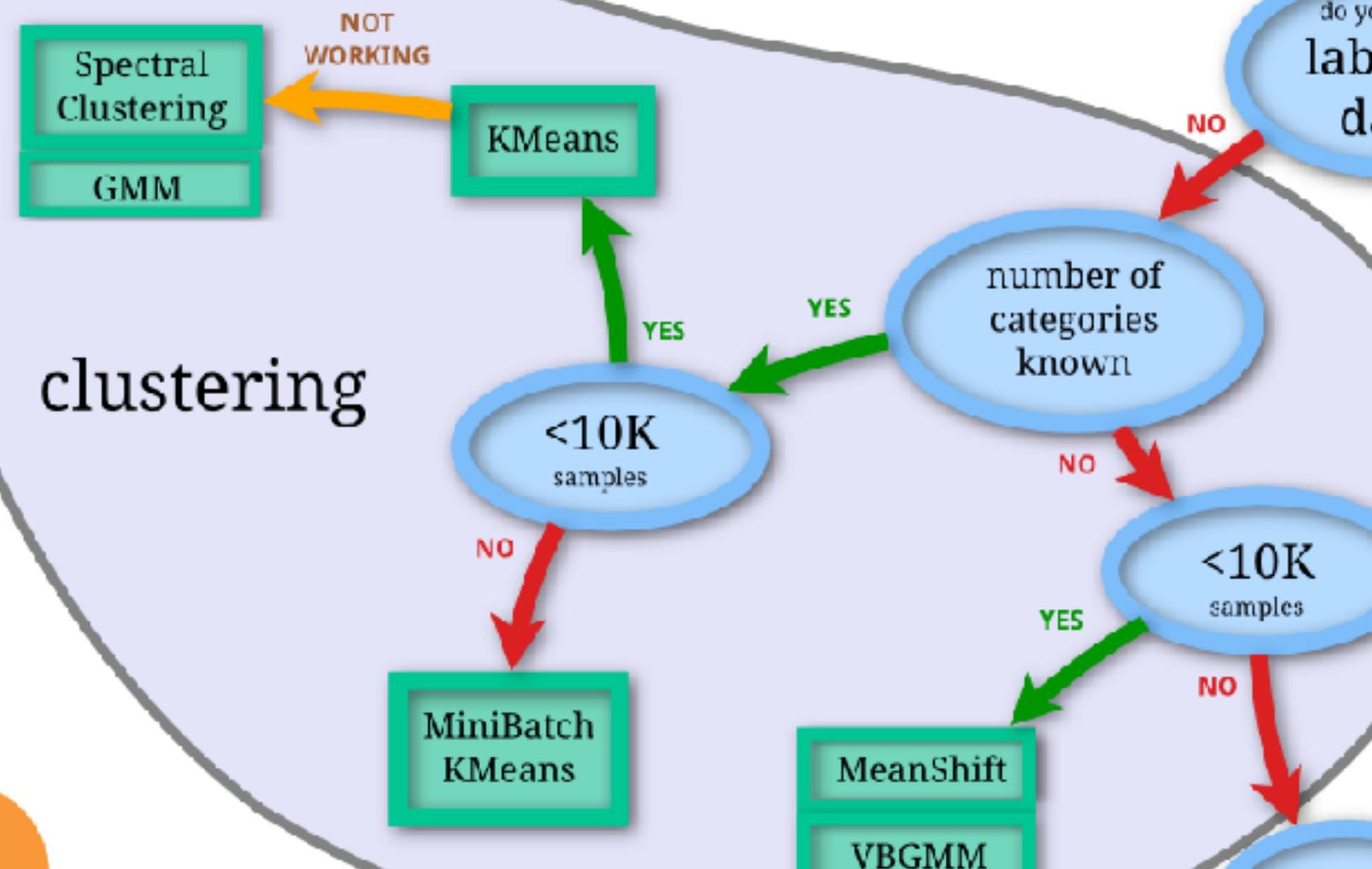
# What Data Science is Not (sorry sklearn)

scikit-learn  
algorithm cheat-sheet

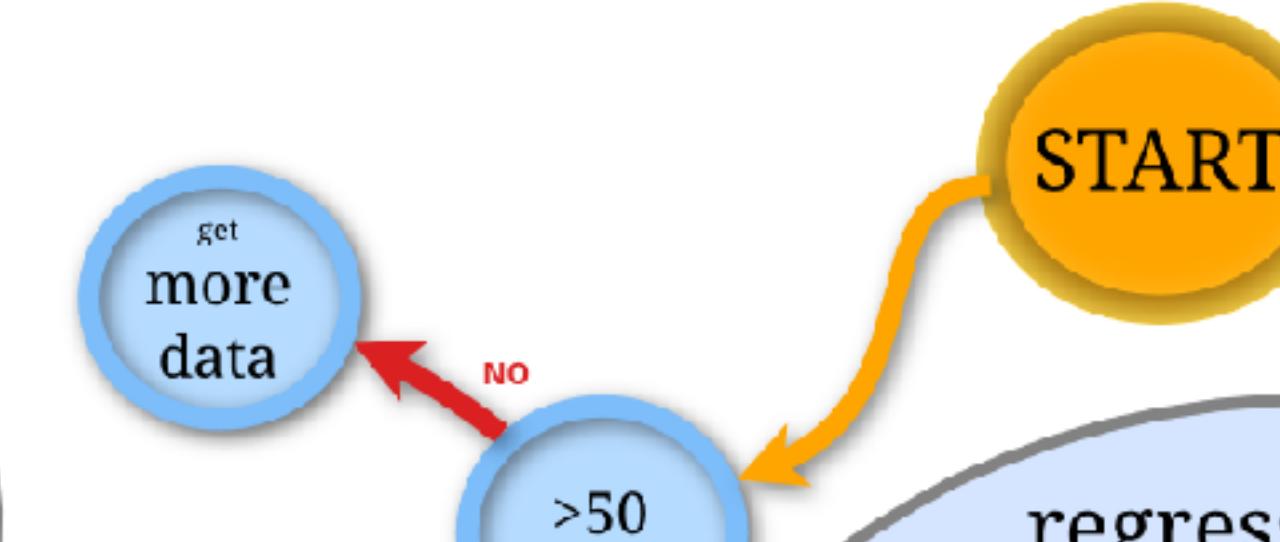
classification



clustering



scikit  
learn



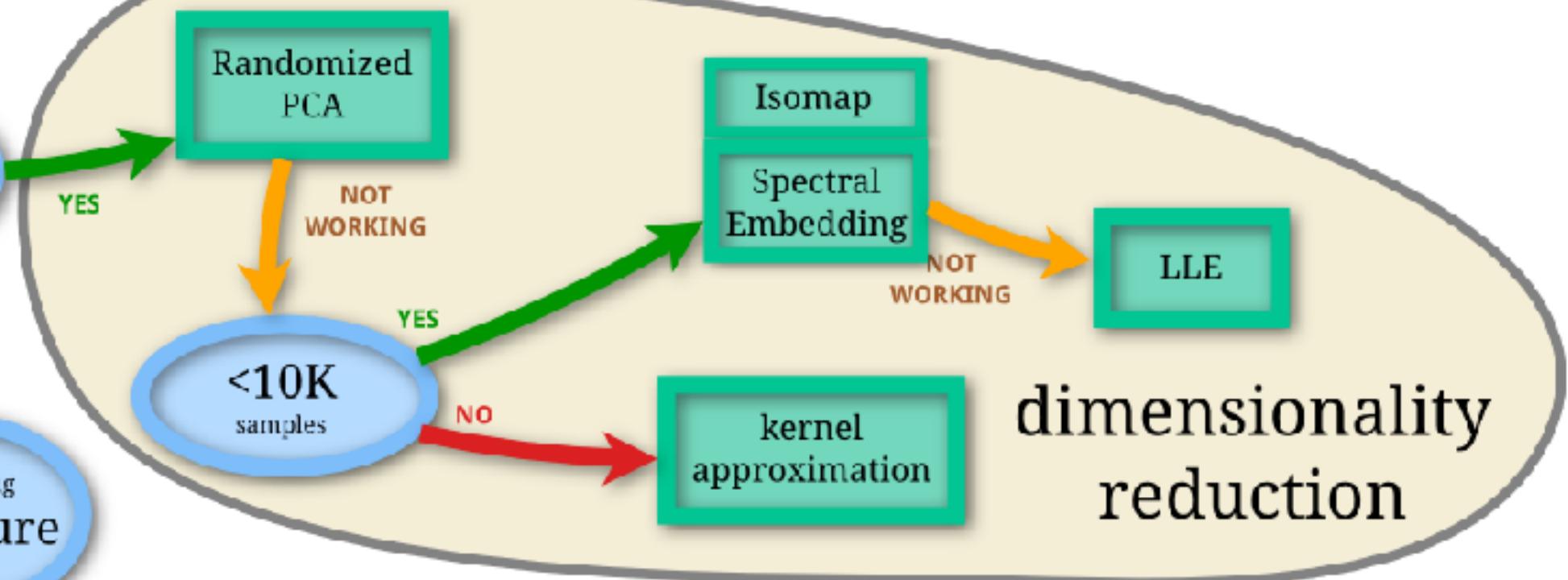
regression

predicting a category

predicting a quantity

just looking

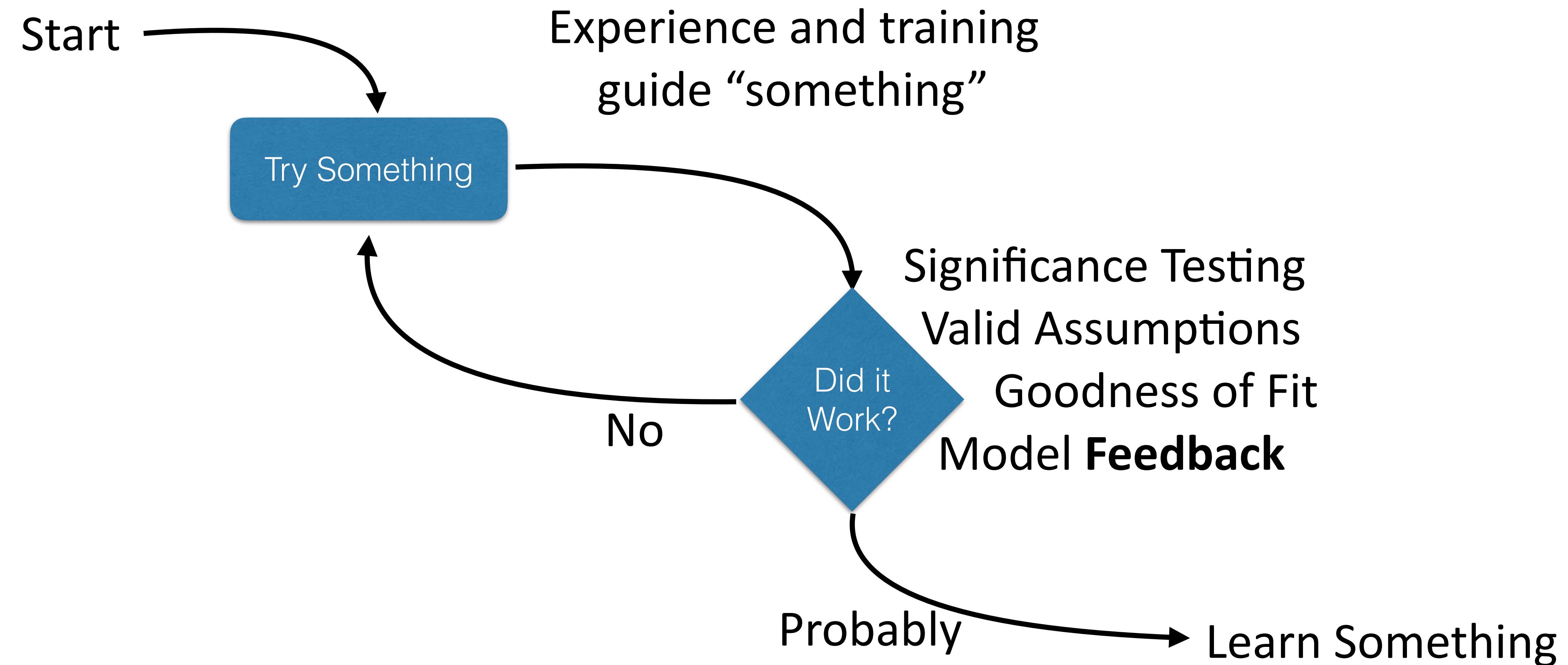
predicting structure



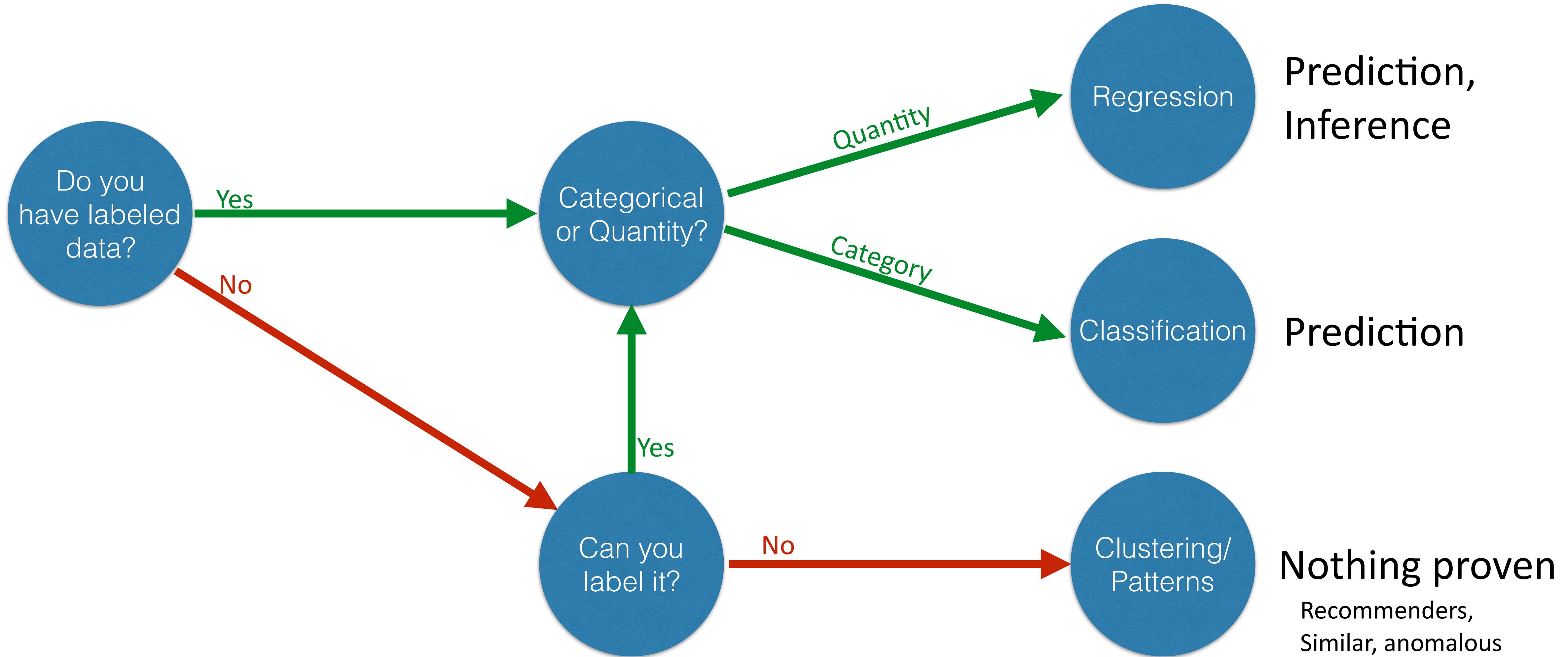
dimensionality  
reduction

Back

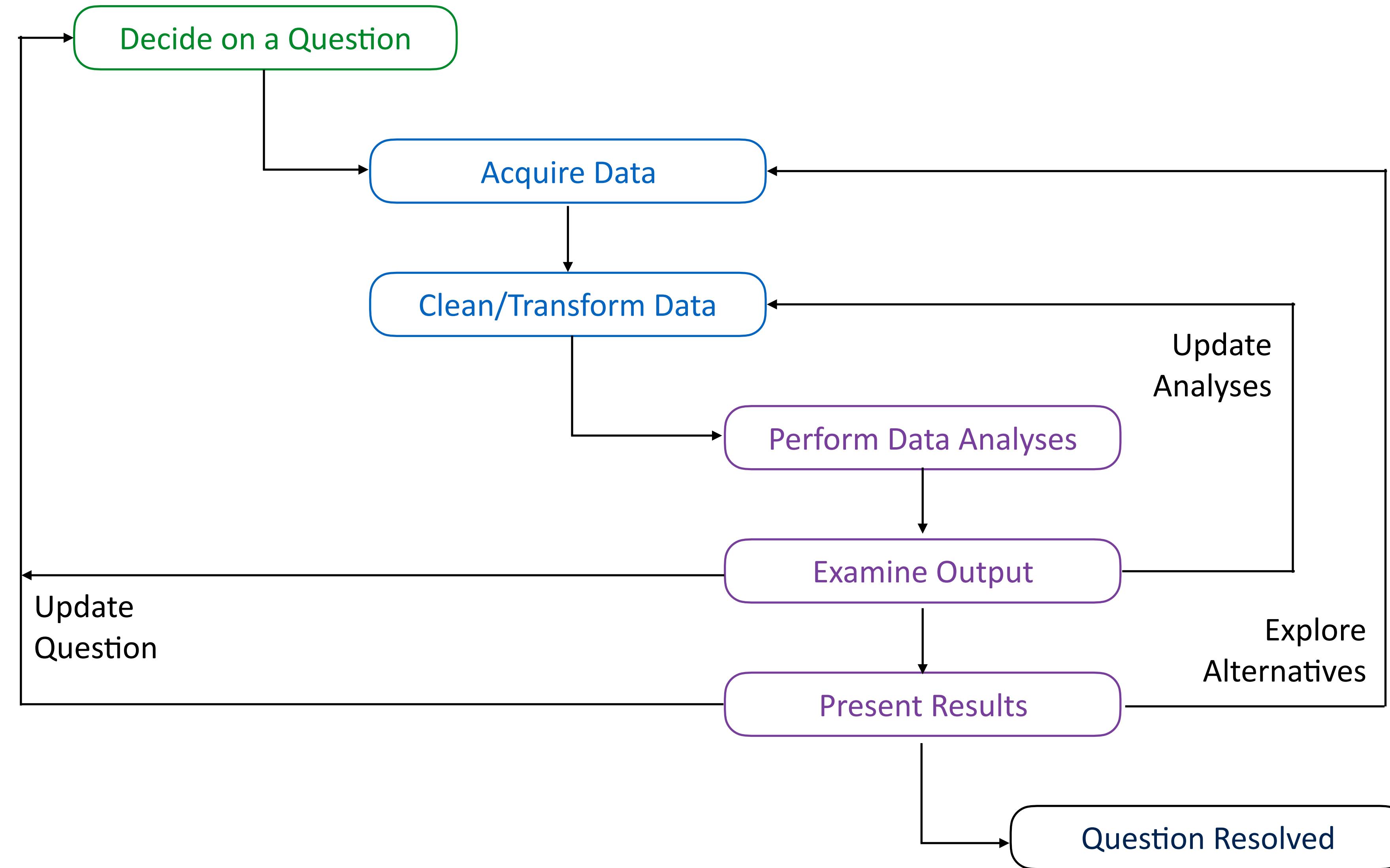
# What Data Science is



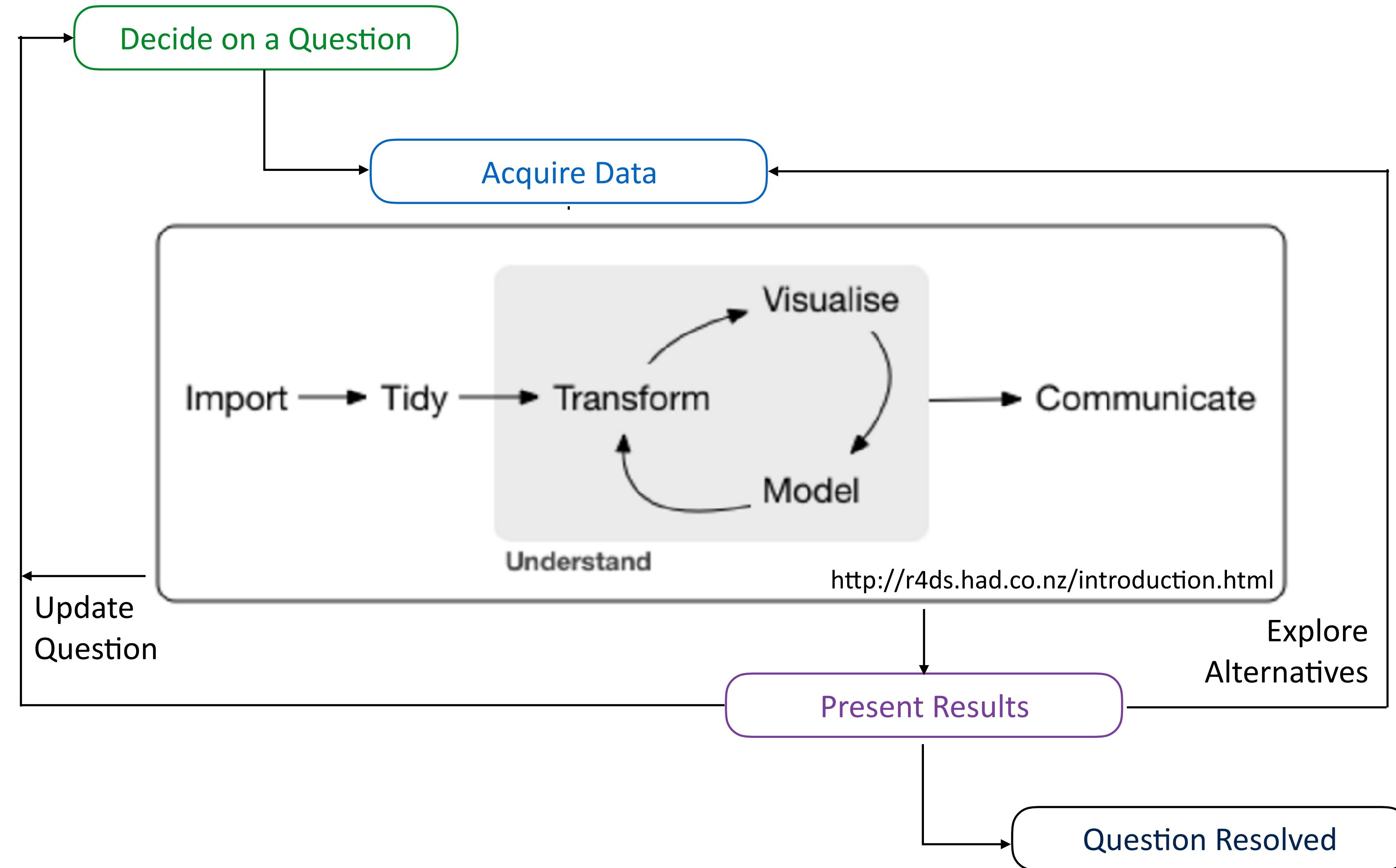
# What Data Science kind of is...



# Research Workflow

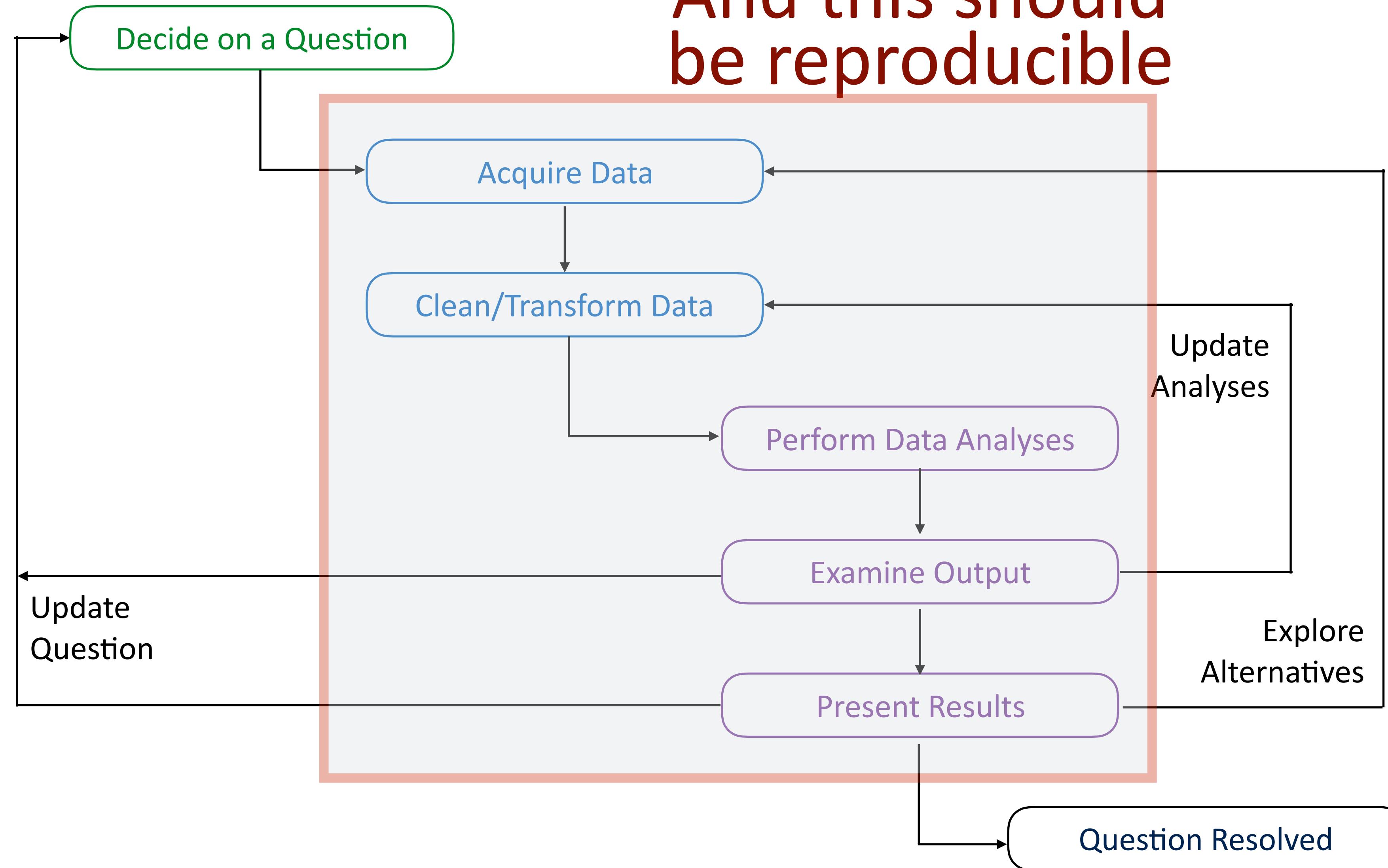


# Research Workflow



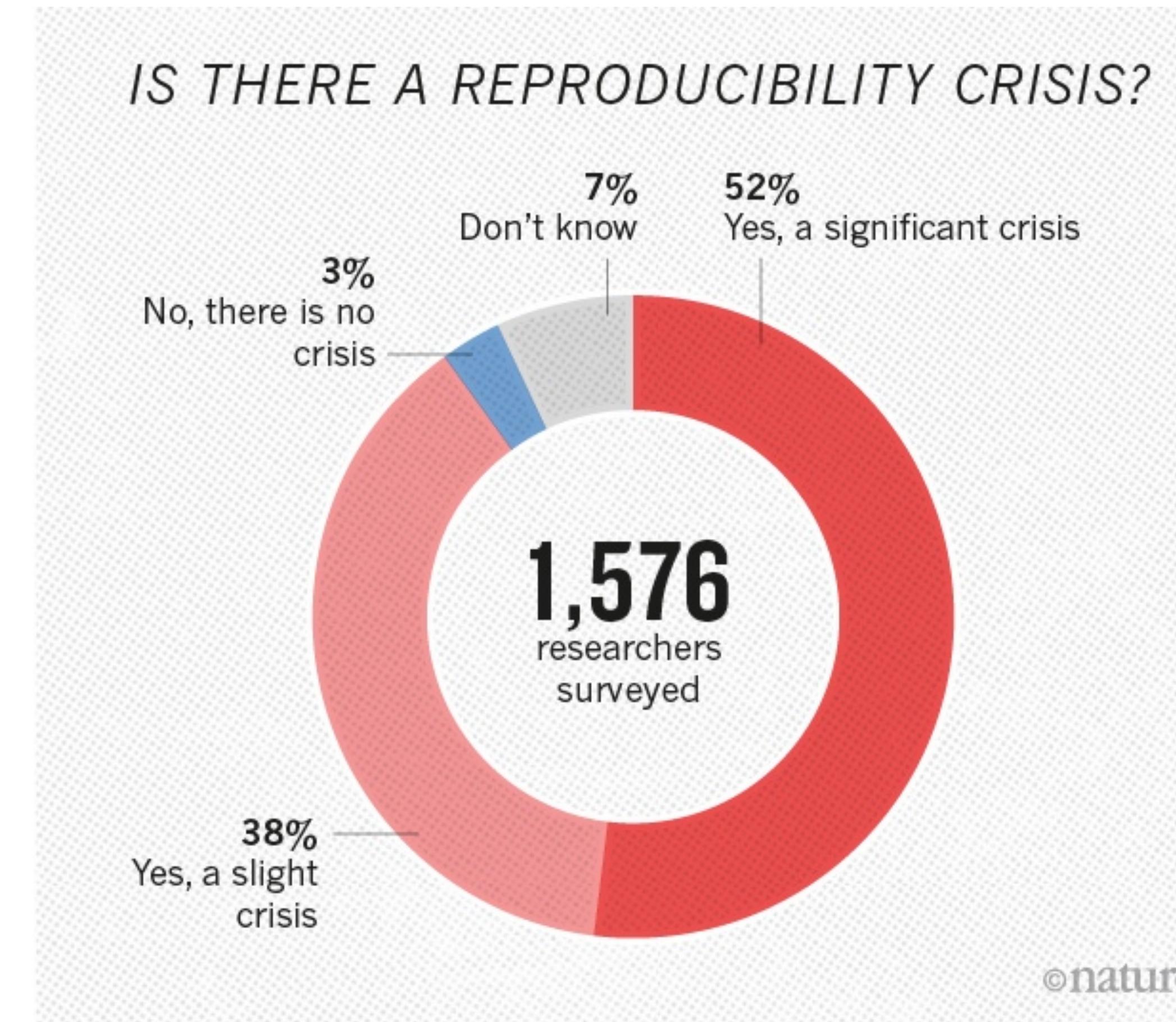
# Research Workflow

And this should  
be reproducible



# Reproducible Research

- Way more than reducing technical debt
- **the Science:** Data collection, methodology, techniques, results
- **the Process:** Raw data prep, transformations, analysis, interpretation.
- **the Technical:** platform and environment, code, dependancies and library versions
- **Nothing** happens outside of code



O'REILLY®

# Security

BUILD BETTER DEFENSES

## Getting into coding

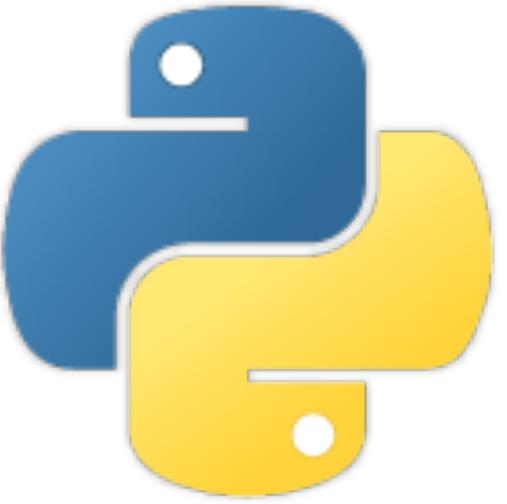
[oreillysecuritycon.com](http://oreillysecuritycon.com)

#oreillysecurity

# Our Tools

## Python

- Jupyter
- Pandas
- Numpy
- Matplotlib/Seaborn/YellowBrick
- scikit-learn



## R

- RStudio
- RMarkdown
- “tidyverse”
- ggplot2
- Shiny



- 
- By Developers
  - Starts counting at zero
  - Awesome as a general language
  - Getting better for data/stats/ML
  - Getting better for visualization

- By Statisticians
- Starts counting at one
- Not terrible as a general language
- Awesome for data/stats/ML
- Awesome for visualization

# Our Tools

## Python

- Jupyter
- Pandas
- Numpy
- Matplotlib
- Seaborn

## R

- RStudio
- RMarkdown
- dplyr
- ggplot2
- “tidyverse”

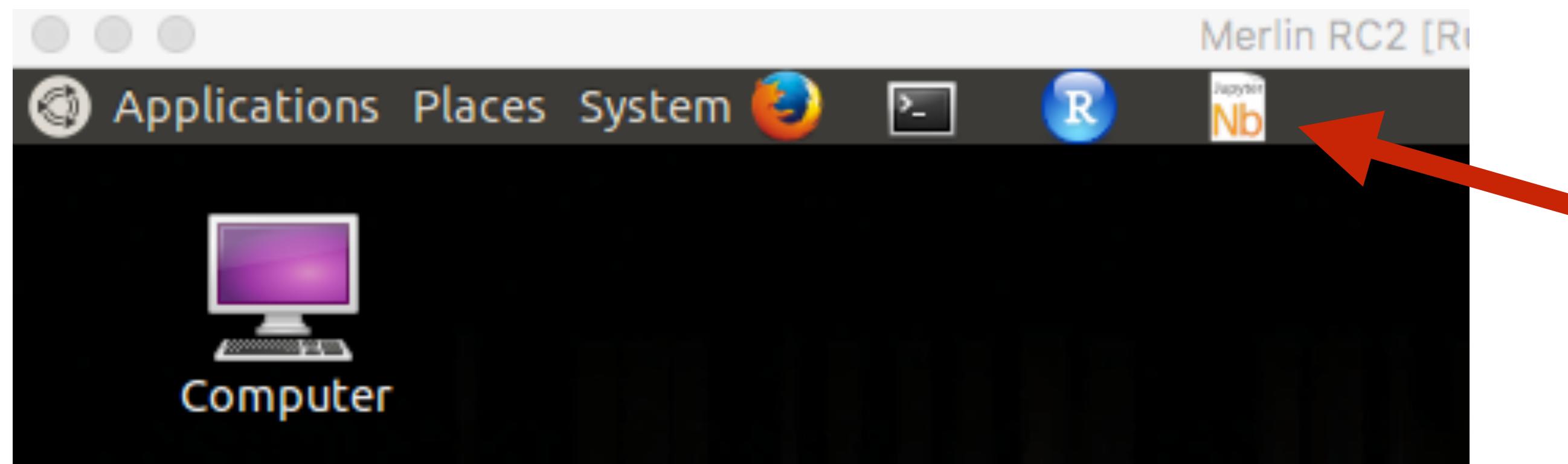
**Start up your VM's (or environment) now!**

**GRIFFON**

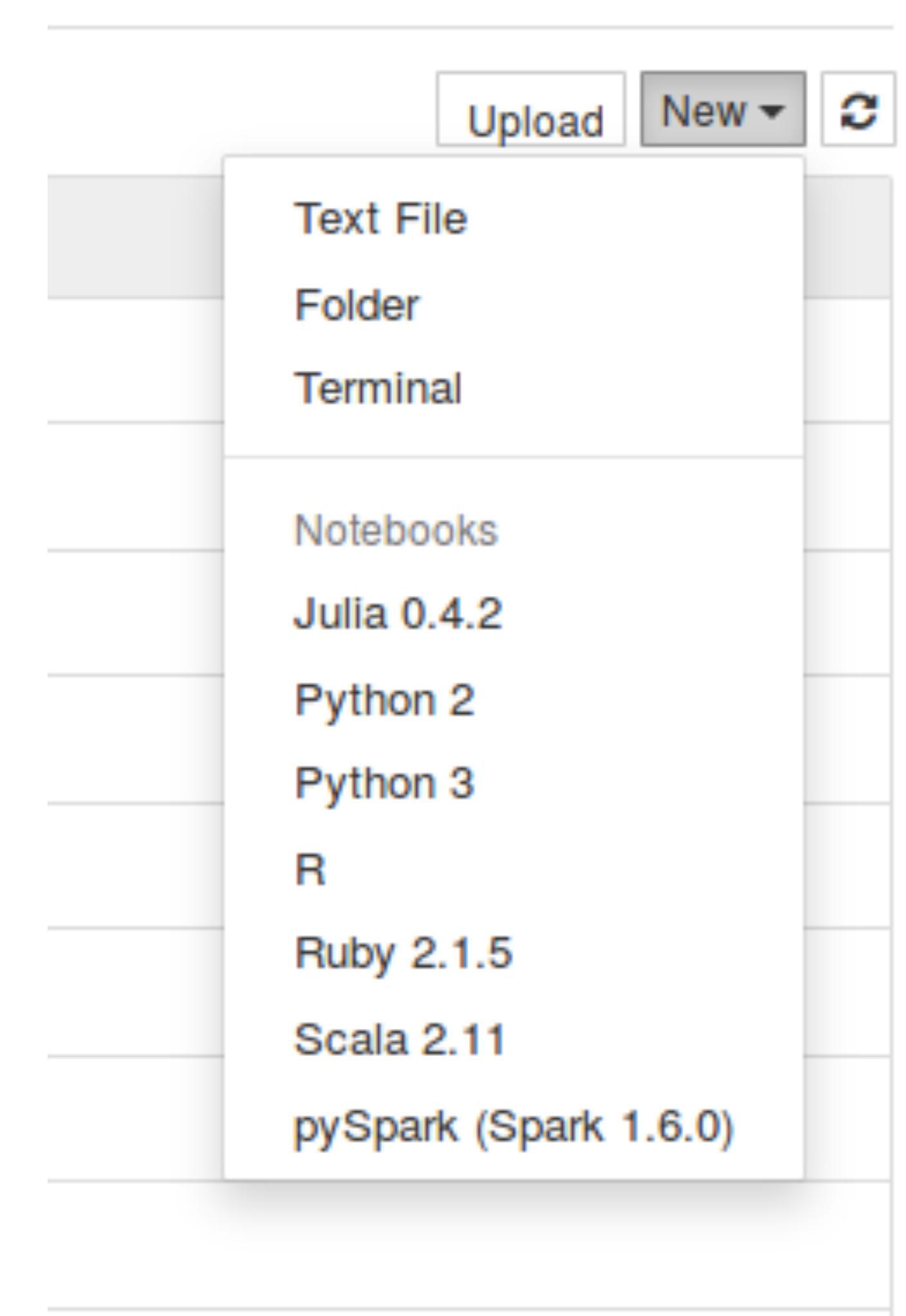
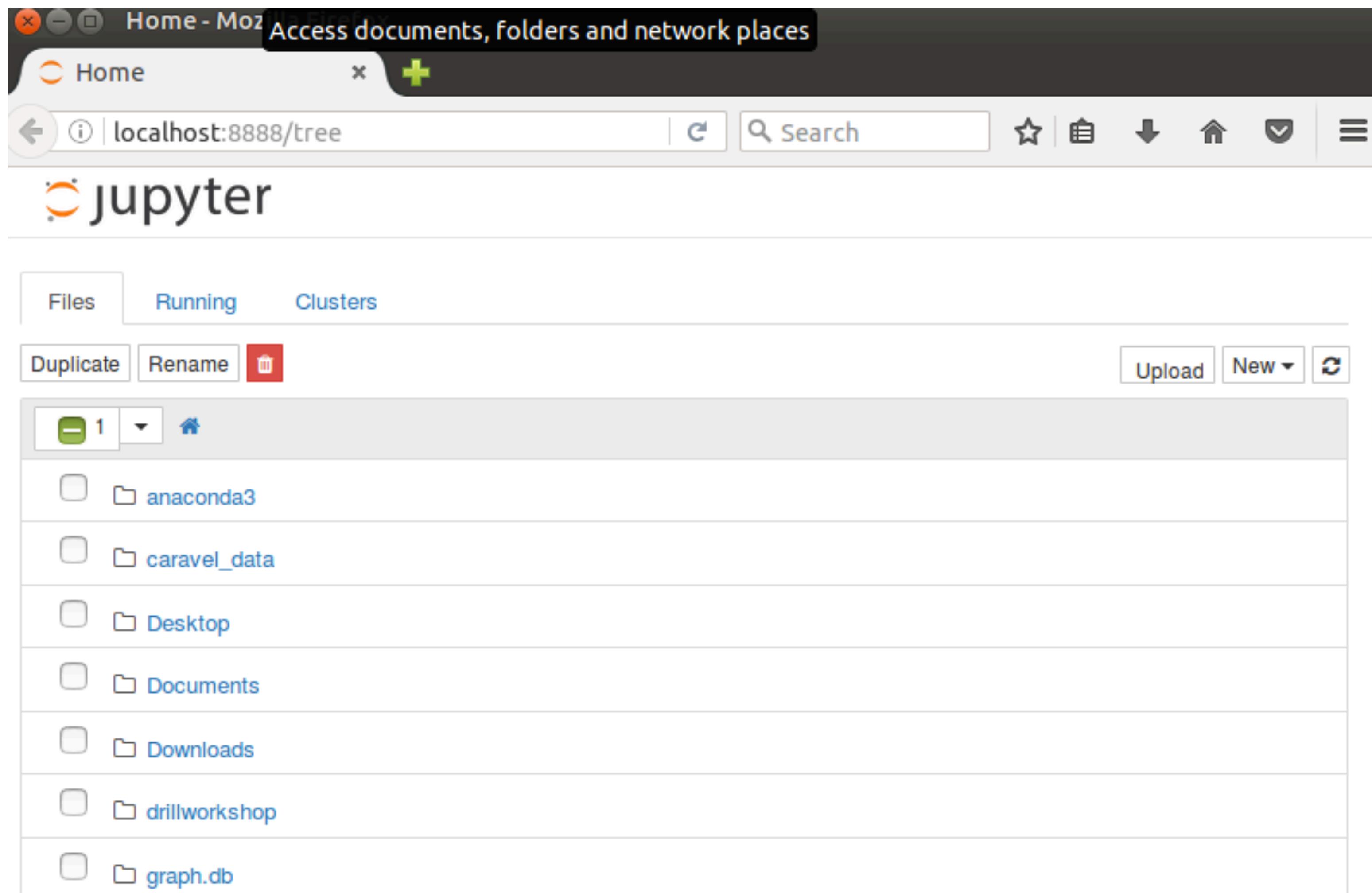


<https://github.com/gtkcyber/griffon-vm>

## Step 1: Launch Jupyter



Step 2: Navigate to your notebook, or create a new one.



## Step 3: Start Coding!

```
In [1]: x = "Hi"  
       print( x )
```

Hi

```
In [ ]:
```

# RStudio

The screenshot shows the RStudio interface with several panels:

- Source files**: The leftmost panel displays an R Markdown document titled "Untitled1". It includes code chunks for generating an HTML document, displaying the R Markdown syntax, and creating plots.
- Console**: The bottom-left panel shows the R console output, including the R license information and various help messages.
- Environment**: The rightmost panel shows the Global Environment pane, which is currently empty.
- Central Area**: A large central area contains the text:

Environment is empty  
**Environ, Build,  
History, VCS**

**Introduction to dplyr**  
**Plots, Helps, etc**

When working with data you must:

  - Figure out what you want to do.
  - Describe those tasks in the form of a computer program.
  - Execute the program.

The dplyr package makes these steps fast and easy:

  - By constraining your options, it simplifies how you can think about common data manipulation tasks.
  - It provides simple “verbs”, functions that correspond to the most common data manipulation tasks, to help you translate those thoughts into code.
  - It uses efficient data storage backends, so you spend less time waiting for the computer.

This document introduces you to dplyr’s basic set of tools, and shows you how to apply them to data frames. Other vignettes provide more details on specific topics:

# Python: Rodeo

The screenshot displays the Rodeo Python IDE interface. The top menu bar includes File, Edit, View, Window, Session, and Help. The main window features a code editor on the left containing a script with Python code, and a file browser on the right listing various files and folders. A central panel contains tabs for Environment, History, and a search bar. A sidebar on the right provides links to ScienceOps, Tutorials, Docs, Feedback, Github, and Settings.

Rodeo

Rodeo File Edit View Window Session Help

New File Run Line Run Script

# Press COMMAND + ENTER to run a single line in the console  
print 'Welcome to Rodeo!'  
# Press COMMAND + ENTER with text selected to run multiple lines  
# For example, select the following lines  
x = 7  
x\*\*2  
# and remember to press COMMAND + ENTER  
# You can also run code directly in the console below.  
#####  
# Here is an example of using Rodeo:  
# We'll use the popular package called Pandas  
# Install it with pip  
import pip  
pip.main(['install', 'pandas'])  
# Import it as 'pd'  
import pandas as pd  
# Create a dataframe  
df=pd.DataFrame({'Animal':["don","dolphin","chicken","ant","sn", "pig"], "Name": ["Dolphin", "Don", "Chicken", "Ant", "Sn", "Pig"]})

Name Type Value

Environment History

Console

IPython -- An enhanced Interactive Python.  
? -> Introduction and overview of IPython's features.  
%quickref -> Quick reference.  
help -> Python's own help system.  
object? -> Details about 'object', use 'object??' for extended info.  
>>>

Files Plots Packages

.. anaconda3 caravel\_data Desktop Documents Downloads drillworkshop graph.db hbase metastore\_db

?

Docs

Feedback

Github

Settings

# Getting Documentation and help

- Google
- <https://www.python.org/doc/>
- `help("<function>")`

Python

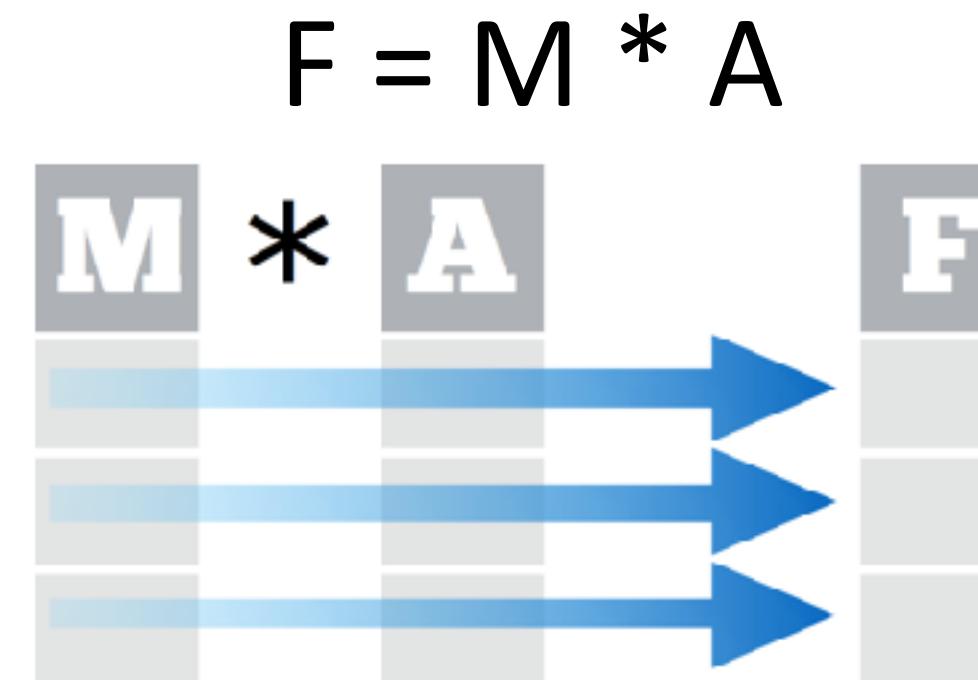
- 
- Google
  - `help("<function>")`
  - `?<function>`
  - `??<search for word>`

R

# Concepts

## Data concepts differ from development concepts

- From certainty to probability
  - Data Quality - imperfect world (missing, wrong, noisy)
- Vectorized Operations



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

&



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

# Vectorized Operations

```
# this is wrong
def add10(x):
    for i in range(0, len( x )):
        x[i] = x[i] + 10
    return x

# this is right
def add10(x):
    return x + 10
```

```
>>> y = pd.Series([ 1,2,3,4,5 ])
>>> y + 10
0    11
1    12
2    13
3    14
4    15
dtype: int64
```

Python

```
# this is wrong
add10 <- function(x) {
  for(i in length(x) {
    x[i] <- x[i] + 10
  }
  x
}
```

```
> x <- c(1,2,3,4,5)
> x + 10
[1] 11 12 13 14 15
```

```
# this is right
add10 <- function(x) x + 10
```

R

# Concepts: Core Data objects

Dimensions	Description	Python (pandas)	R
1	Traditional array or python list	Series	Vector
2	Two dimensional table (excel spreadsheet)	Data Frame	Data Frame (tibble)
3	Three dimensional mutable data structure	Panel	Maybe a list? [you are making things too complicated]

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

Python

R

```
library(tidyverse)
```

ggplot2, for data visualisation.  
dplyr, for data manipulation.  
tidyr, for data tidying.  
readr, for data import.  
purrr, for functional programming.  
tibble, a modern re-imagining of data frames.

# Creating 1-Dimensional

```
myData = pd.Series( <data>, index=<index> )
series1 = pd.Series( [ 'a' , 'b' , 'c' , 'd' , 'e' ] )
series2 = pd.Series( { 'firstName' : 'Charles' ,
                      'lastName' : 'Givre' } )
```

Python

```
myData <- c( <data> )
series1 <- c( 'a' , 'b' , 'c' , 'd' , 'e' )
series2 <- c( 'firstName' = 'Jay' ,
            'lastName' = 'Jacobs' )
```

R

Still a vector, just with names

# Creating 1-Dimensional

```
> series1 = pd.Series( [ 'a', 'b', 'c', 'd', 'e' ] )  
> print(series1)  
0    a  
1    b  
2    c  
3    d  
4    e  
dtype: object
```

Python

---

```
> series1 <- c( 'a', 'b', 'c', 'd', 'e' )  
> print(series1)  
[1] "a" "b" "c" "d" "e"
```

R

# 1-Dimensional Series/Vectors

```
> series1 = pd.Series( [ 'a', 'b', 'c', 'd', 'e' ] )
> print(series1[3])
d
> print(series1[1:3])
1    b
2    c
dtype: object
```

Python

---

```
> series1 <- c( 'a', 'b', 'c', 'd', 'e' )
> print(series1[3])
[1] "c"
> print(series1[1:3])
[1] "a" "b" "c"
```

R

# Creating 1-Dimensional

```
> record = pd.Series( {'firstName' : 'Charles',  
                      'lastName': 'Givre' } )  
> print(record)  
firstName      Charles  
lastName       Givre  
dtype: object  
> print(record['firstName'])  
Charles
```

Python

```
record <- c( 'firstName' = 'Jay',  
            'lastName' = 'Jacobs' )  
> print(record)  
firstName lastName  
"Jay"    "Jacobs"  
> print(record["firstName"])  
firstName  
"Jay"
```

R

# Creating 1-Dimensional

```
> randomNumbers = pd.Series( np.random.random_integers(1, 100, 50) )  
> randomNumbers.head()  
0    48  
1    34  
2    84  
3    85  
4    58  
dtype: int64
```

Can also  
use “tail”

```
> randomNumbers <- as.integer(runif(50, min=0, max=100))  
> head(randomNumbers)  
[1] 93 14 24 49 48 44
```

Python

R

# Applying a function to a 1-D Set

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

x = pd.Series( [1,2,3,4,5] )                    x.apply( lambda n: n + 2 )

x.apply( addTwo )
0    3
1    4
2    5
3    6
4    7
dtype: int64

>>> x = pd.Series( [ 'one', 'two', 'three' ] )
>>> x.apply(len)
0    3
1    3
2    5
dtype: int64
```

Python

```
addTwo <- function(n) n + 2
x <- c(1,2,3,4,5)
sapply(x, addTwo)
[1] 3 4 5 6 7

> x <- c('one', 'two', 'three')
> sapply(x, nchar) # "nchar" counts characters
one   two three
3      3      5
```

R

# Creating 1-Dimensional

```
> randomNumbers = pd.Series( np.random.random_integers(1, 100, 50) )
> randomNumbers.mean()
55.82
> randomNumbers.median()
52.5
> randomNumbers.sum()
2791
```

Python

```
> randomNumbers <- as.integer(runif(50, min=0, max=100))
> mean(randomNumbers)
[1] 44.04
> median(randomNumbers)
[1] 38.5
> sum(randomNumbers)
[1] 2202
```

R

# Creating 1-Dimensional: filtering

```
# randomNumbers[ <boolean> ]  
  
> randomNumbers[ randomNumbers < 10 ]  
1     9  
7     7  
11    6  
28    8  
38    6  
42    3  
  
dtype: int64
```

Python

```
# randomNumbers[ <boolean> ]  
  
> randomNumbers[ randomNumbers %% 2 == 0 ] # %% is remainder, get evens  
[1] 28 84 8 72 98 38 96 28 14 82 46 8 92 18 62 2
```

R

# Creating 1-Dimensional: string manipulation

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

Python

Function	Explanation	
grepl(<pattern>, <data>)	Returns true/false if text matches a pattern	
sum(grepl(<pattern>, <data>))	Returns number of occurrences of a pattern in a string	
strings::str_extract(<string>, <pattern>)	Returns matching groups from a string (stringr package)	“stringr” is the package for fast string manipulation
match(<string>, <pattern>)	Returns index of first occurrences of a substring	
nchar(<data>)	Returns the length of text	
gsub(<pat>, <replace>, )	Replaces matches with a replacement string	

R

# Creating 1-Dimensional: string manipulation

```
mySeries = pd.Series( [1,2,3,4] )  
#Sort by value  
mySeries.sort_values()  
#Sort by index  
mySeries.sort_index()  
#Reverse order  
mySeries.sort( ascending=False )  
#Combining series  
mySeries.append( pd.Series( [5] ) )
```

Python

```
myVector <- c(1,2,3,4)  
# sort by value  
sort(myVector)  
# Sort by index  
order(myVector)  
#Reverse order  
sort(myVector, decreasing = TRUE )  
#Combining series  
myVector <- c(myVector, 5)
```

R

# 1-D Statistical Operations

Python	R	Description
<b>Series.abs()</b>	<b>abs(vector)</b>	Absolute Values
<b>Series.count()</b>	<b>length(vector)</b>	Returns number of values
<b>Series.max()</b>	<b>max(vector)</b>	Returns maximum value
<b>Series.mean()</b>	<b>mean(vector)</b>	Returns the mean
<b>Series.median()</b>	<b>median(vector)</b>	Returns the median
<b>Series.min()</b>	<b>min(vector)</b>	Returns the minimum value
<b>Series.mode()</b>	-	Returns the modal value in a Series
<b>Series.quantile([q])</b>	<b>quantile(vector, &lt;q&gt;)</b>	Returns the quantiles (percentiles)
<b>Series.sum</b>	<b>sum(vector)</b>	Returns the sum
<b>Series.std</b>	<b>sd(vector)</b>	Returns the standard deviation

# 1-D Exploratory Operations

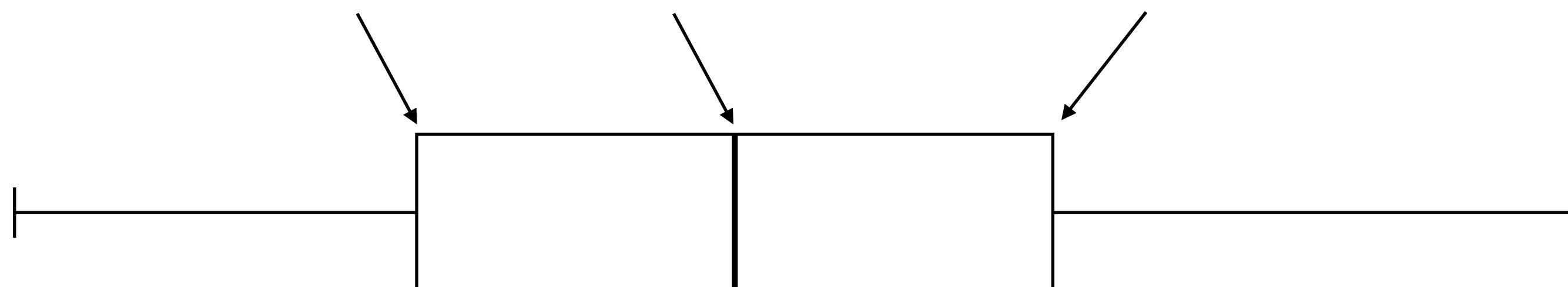
`Series.describe()`

# Creating 1-Dimensional: data inspection

```
>>> Series.describe()  
count      50.000000  
mean      52.000000  
std       29.154759  
min       3.000000  
25%     27.500000  
50%     52.000000  
75%     76.500000  
max     101.000000  
dtype: float64
```

Tukey's "Five  
Number Summary"

```
> summary(vector)  
   Min. 1st Qu. Median      Mean 3rd Qu. Max.  
0.06053 23.62000 49.24000 49.50000 75.79000 99.99000
```



Python

R

# Creating 1-Dimensional: data inspection

```
>>> sizes = pd.Series( ["small", "small", "medium", "large", "large", "large"] )  
>>> sizes.describe()  
count          6  
unique         3  
top           large  
freq          3  
dtype: object
```

Python

```
> sizes <- c("small", "small", "medium", "large", "large", "large")  
> summary(sizes)  
  Length   Class    Mode  
       6  character  character  
> sizes <- factor(sizes) ← Factors tell R “Treat this as a categorical variable”  
> summary(sizes)  
 large medium small  
      3        1        2
```

R

# Creating 1-Dimensional: data inspection

```
>>> sizes.unique()
array(['small', 'medium', 'large'], dtype=object)
>>> sizes.value_counts()
large      3
small      2
medium     1
dtype: int64
```

Python

```
> unique(sizes)
[1] small  medium large
Levels: large medium small ←———— Still a factor
> table(sizes)
sizes
  large medium small
    3       1      2
```

R

# { LAB } time

Please Complete “One Dimensional Worksheet”



# Answers: One Dimensional Worksheet - Exercise 1

```
#First initialize the series by calling the pd.Series() function
randomNumbers = pd.Series( np.random.randint(1, 100, 100) )
```

```
#Display the first 5 random numbers
print( randomNumbers.head() )
```

```
#Next filter out the odd numbers by using the mod operator and reset the index
evenRandomNumbers = randomNumbers[ randomNumbers % 2 == 0 ].reset_index( drop=True )
```

```
#Display the first 5
evenRandomNumbers.head()
```

```
set.seed(1)
x <- floor(runif(100, min=1, max=100)) # or round or ceiling
# x <- sample(100, 100, replace=TRUE) # if replace=FALSE, it will just re-arrange 1 to
100

odnum <- x[x %% 2 == 1]
print(odnum[1:20]) # or head(odnum, 20)
```

Python

R

# Answers: One Dimensional Worksheet - Exercise 2

```
#Create the phone numbers series  
phoneNumbers = pd.Series( numbers )
```

*#Next filter the phone numbers by using the str.match function*

```
validPhoneNumbers = phoneNumbers[ phoneNumbers.str.match( r'\\(\\d{3}\\)\\d{3}-  
\\d{4}' ) ].reset_index( drop=True )
```

```
phone <- grep('\\\\(\\\\d{3}\\\\)\\\\d{3}-\\\\d{4}', numbers, value = TRUE)
```

Python

R

# Answers: One Dimensional Worksheet - Exercise 3

```
tempsinCelsius = tempsInFarenheit.apply( toCelsius )
print( tempsinCelsius)
```

Python

```
toCelsius <- function(x) ((5/9)*x-32)

tempSF <- c(92, 33, -5, 17, 122, 87)
tempSC <- sapply(tempSF, toCelsius)
print(tempSC)
```

R

# Answers: One Dimensional Worksheet - Exercise 4

```
numSeries = pd.Series( numList)  
numSeries.value_counts()
```

Python

```
numList <- c(1,1,1,1,1,2,4,5,7,5,4,5,6,4,3,5,5,5,6,9,0,7,6,7,5,4,4,7)  
table(numList)
```

R

# Answers: One Dimensional Worksheet - Exercise 5

```
IPData = pd.Series( hosts )
privateIPs = IPData[IPData.apply( lambda x : ip_address(x).is_private ) ]
print( privateIPs )
```

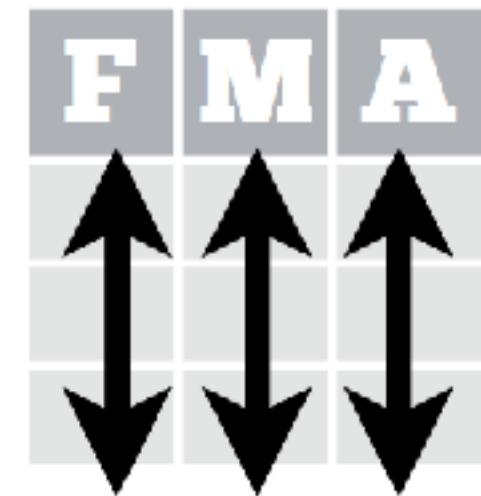
Python

R

# DataFrames

```
>>> pd.DataFrame( { 'count' : [ 1,2,3 ],  
                    'letter' : [ 'a','b','c' ],  
                    'decimal' : [ 1.23, 2.34, 3.45 ] } )
```

	count	decimal	letter
0	1	1.23	a
1	2	2.34	b
2	3	3.45	c



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

Python

```
> data.frame('count' = c(1,2,3),  
            'letter' = c('a','b','c'),  
            'decimal' = c(1.23, 2.34, 3.45))
```

	count	letter	decimal
1	1	a	1.23
2	2	b	2.34
3	3	c	3.45



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

R

# Loading Various Sources

Function	Python	R
Read from CSV	<code>pd.read_csv( &lt;file&gt;   &lt;URL&gt; )</code>	<code>readr::read_csv( &lt;file   URL&gt; )</code>
Read Excel Doc	<code>pd.read_excel( &lt;file&gt;, sheetname=&lt;sheetname&gt; )</code>	<code>readxl::read_excel( &lt;file&gt;, &lt;sheet&gt; )</code>
Read JSON	<code>pd.read_json( &lt;file   URL&gt; )</code>	<code>rjson::fromJSON( &lt;file   URL&gt; )</code>
Read SQL	<code>pd.read_sql( &lt;query&gt;, &lt;connection&gt; )</code>	<code>RODBC::sqlQuery( &lt;conn&gt;, &lt;query&gt; )</code>
Read HTML	<code>pd.read_html( &lt;source&gt; )</code>	<code>readLines( &lt;file   URL&gt; )</code> <code>rvest::html( &lt;src&gt; )</code>

# 2-Dimensional: creating

```
>>> raw_data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
...             'last_name': ['Miller', 'Jacobson', '.', 'Milner', 'Cooze'],
...             'age': [42, 52, 36, 24, 73] }
>>> people = pd.DataFrame(raw_data, columns = [ 'first_name', 'last_name', 'age' ])
>>> people
   first_name last_name  age
0      Jason    Miller    42
1     Molly  Jacobson    52
2      Tina        .    36
3      Jake    Milner    24
4      Amy     Cooze    73
```

Python

```
> people <- data.frame('first_name' = c('Jason', 'Molly', 'Tina', 'Jake', 'Amy'),
+                         'last_name' = c('Miller', 'Jacobson', '.', 'Milner', 'Cooze'),
+                         'age' = c(42, 52, 36, 24, 73))
> people
   first_name last_name  age
1      Jason    Miller    42
2     Molly  Jacobson    52
3      Tina        .    36
4      Jake    Milner    24
5      Amy     Cooze    73
```

R

## 2-Dimensional: creating

```
> iris = pd.read_csv('iris.csv')

> iris.columns.values.tolist()    # or just list(iris)
['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width', 'Species']

> iris.head(2)
   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
0          5.1         3.5          1.4         0.2   setosa
1          4.9         3.0          1.4         0.2   setosa
```

Python

```
> iris <- read.csv("iris.csv")      # use read_csv for tibble

> colnames(iris)
[1] "Sepal.Length" "Sepal.Width"   "Petal.Length" "Petal.Width"   "Species"

> head(iris, 2)
   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
1          5.1         3.5          1.4         0.2   setosa
2          4.9         3.0          1.4         0.2   setosa
```

R

## 2-Dimensional: functions by variable

```
>>> iris[ 'Sepal.Length' ].mean()
5.843333333333337
>>> iris[ 'Sepal.Length' ].min()
4.2999999999999998
>>> iris[ 'Sepal.Length' ].max()
7.900000000000004
```



???

Python

```
> mean(iris$Sepal.Length)
[1] 5.843333
> min(iris$Sepal.Length)
[1] 4.3
> max(iris$Sepal.Length)
[1] 7.9
```

R

# DataFrames

```
twitterData = pd.read_csv("data/twitter1.csv")
users = twitterData['Username']
usersAndFriends = twitterData[ [ 'Username', 'Friends' ] ]
tweetCounts = twitterData[ 'Username' ].value_counts()
>>> tweetCounts.head()
HoolohaTube          155
Rasu24              150
HOOLOHASPORT        126
mahboobali3          119
EminemsRealWife      116
Name: Username, dtype: int64
```

# Data frames: Subsetting

```
# Read in CSV
twitterData = pd.read_csv("data/twitter1.csv")
# just get "Username" in a Series
users = twitterData['Username']
# create new data frame with Username and Friends
usersAndFriends = twitterData[ [ 'Username', 'Friends' ] ]
```

Python

```
# Read in CSV
twitter = read_csv("data/twitter1.csv")
# just get "Username" in a vector
users <- twitter$Username
# create new data frame with Username and Friends
usersAndFriends <- twitter %>% select(Username, Friends)
# or the "classic" method
usersAndFriends <- twitter[ , c('Username', 'Friends') ]
```

R

# Data frames: Subsetting

```
# Count Tweets by Username
tweetCounts = twitterData['Username'].value_counts()
# Look at first few lines
tweetCounts.head()

HoolohaTube      155
Rasu24          150
HOOLOHASPORT    126
mahboobali3     119
EminemsRealWife  116
Name: Username, dtype: int64
```

```
# Count Tweets by Username
tweetCounts <- twitter %>% count(Username) %>% arrange(desc(n))
# Look at first few lines
head(tweetCounts)

  Username     n
1 HoolohaTube 155
2 Rasu24       150
3 HOOLOHASPORT 126
4 mahboobali3  119
5 EminemsRealWife 116
6 byezekiel    89
```

Python

R

# R: dplyr (tidyverse)

- “Code the way you think”

```
iris %>%  
  group_by(Species) %>%  
  summarise(avg = mean(Sepal.Width)) %>%  
  arrange(avg)
```

- `%>%` basically puts the output on the left as the first argument on the right.

`x %>% f(y)` is the same as `f(x, y)`

`y %>% f(x, ., z)` is the same as `f(x, y, z)`

- Handful of core concepts:

- **filter**: subset rows
- **select**: subset columns
- **group\_by**: set variable(s) as group
- **summarize**: simplify by grouping
- **mutate**: modify/add variable
- **arrange**: sort by variable(s)

# { LAB } time

Please Complete “Two Dimensional Worksheet”



# Answers: Two Dimensional Worksheet - Exercise 1

```
tweetCounts = twitterData['Username'].value_counts()  
tweetCounts.head(10)
```

Python

```
twitter %>% count(Username) %>% top_n(10) %>% arrange(desc(n))
```

R

# Answers: Two Dimensional Worksheet - Exercise 2

```
twitterSummary = twitterData[ [ 'Username' , 'Friends' , 'Followers' ] ]
twitterSummary[ 'ffratio' ] = twitterSummary[ 'Friends' ] / twitterSummary[ 'Followers' ]

twitterSummary.head( )
```

Python

```
twitterSummary <- twitter %>%
  select(Username, Friends, Followers) %>%
  mutate(ffratio = Friends / Followers)
# it will automatically show it pretty
twitterSummary
```

R

# Answers: Two Dimensional Worksheet - Exercise 3

```
studentData = pd.read_csv(' ../../data/studentData.csv')

studentData[ 'average' ] = studentData[ [ 'Test1', 'Test2', 'Test3', 'Test4',
'Test5' ] ].mean(axis=1)

studentData.sort_values( 'average', axis=0, ascending=False )
```

Python

```
students <- read_csv( '../../data/studentData.csv')

students %>%
  gather(cat, val, -`Student ID`) %>% # now in "tall" format
  group_by(`Student ID`) %>%
  summarize(avgtest = mean(val)) %>%
  arrange(-avgtest)
```

R

# Answers: Two Dimensional Worksheet - Exercise 4

```
newData = twitterData[ twitterData['URL'].fillna("") .str.contains('facebook') ]
newData['FacebookID'] = newData['URL'].str.extract( 'profile.php\?id=(\d+)' ,
expand=False)
newData.dropna( inplace=True )
```

Python

```
library(stringr) # do you have this installed?
getfacebook <- function(x) {
  ifelse(grepl('\?\id=\d+', x),
    str_extract(str_extract(x, "\?\id=\d+"), "\d+"),
    NA)
}
foo <- twitter %>%
  ## filter for all facebook to reduce computations later
  filter(grepl('facebook.com', URL)) %>%
  ## run all the URLs through the above function
  mutate(FacebookID = getfacebook(URL))
foo %>% select(URL, FacebookID) %>% filter(!is.na(FacebookID))
```

R