O'REILLY®

Security

BUILD BETTER DEFENSES

oreillysecuritycon.com
#oreillysecurity

# Foundations of Security Data Science

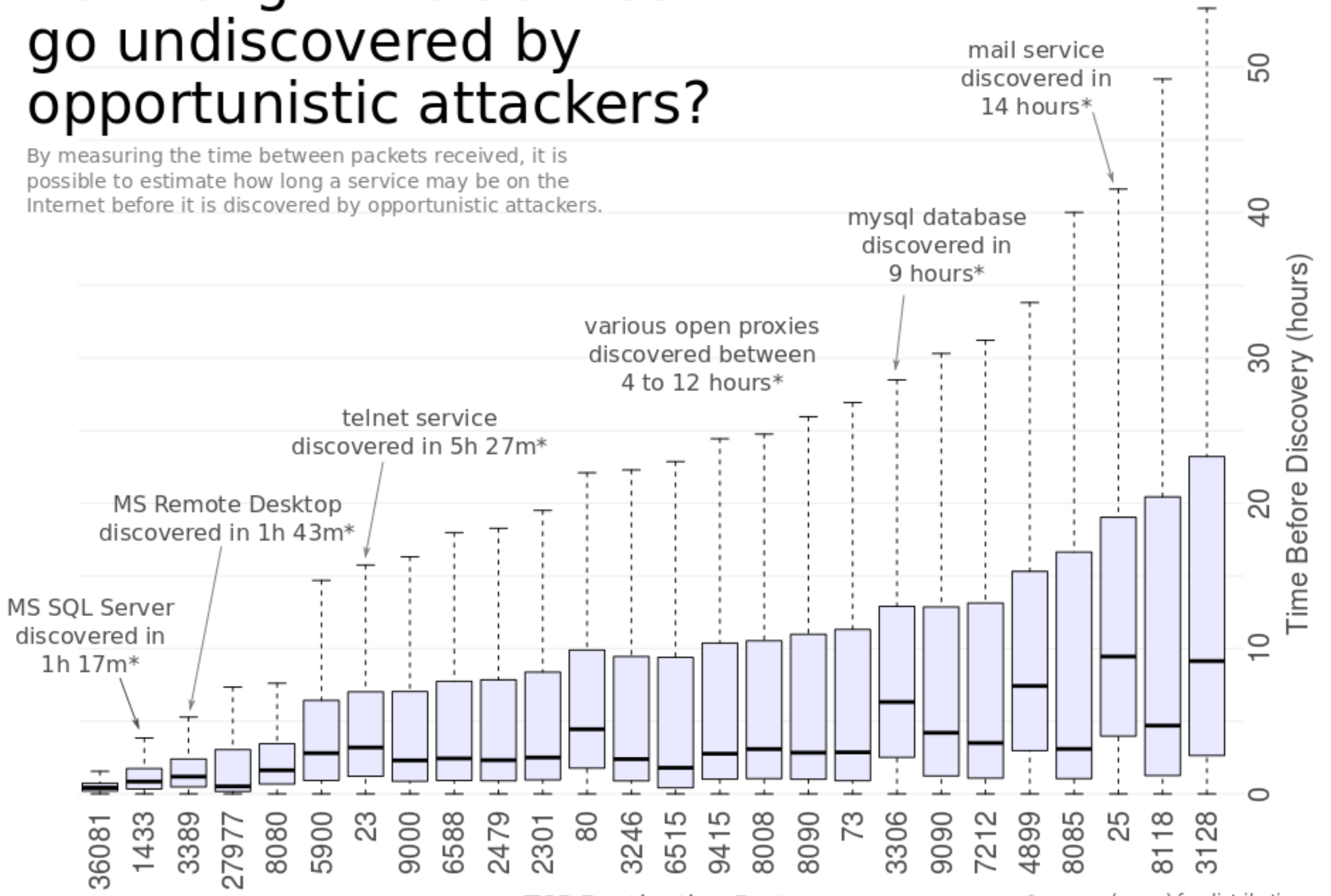
*Introductions*

Jay Jacobs
Charles Givre
Bob Rudis


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.



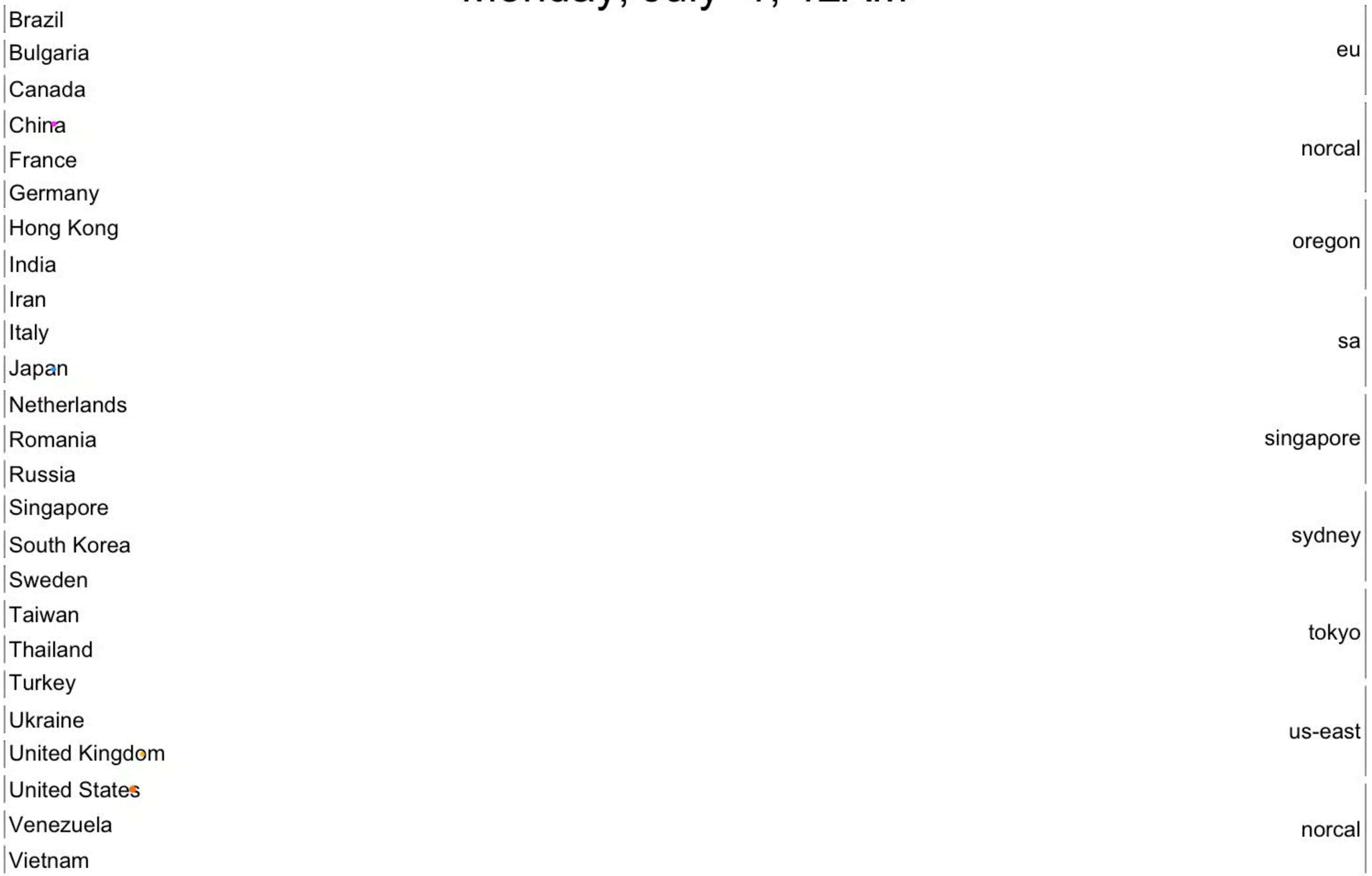

* average(mean) for distribution

http://securityblog.verizonbusiness.com/?p=2283

| 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

# Monday, July 1, 12AM

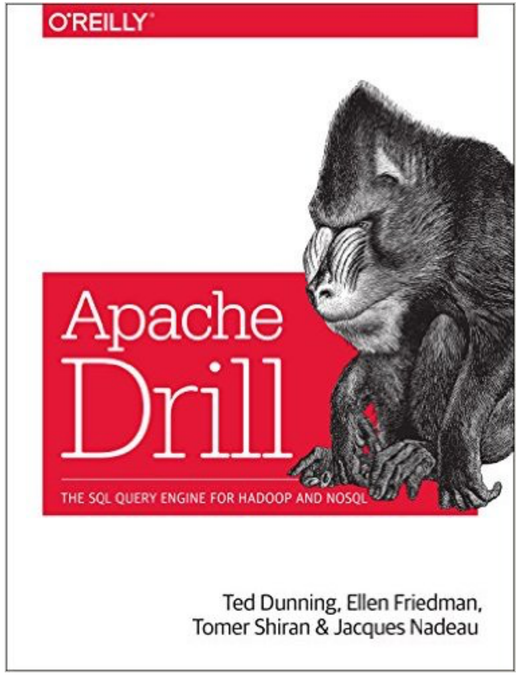


# Jay Jacobs

- 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)

# Charles Givre

- Sr. Lead Data Scientist @ Booz Allen Hamilton
- 5 years @ CIA
- Working on Apache Drill Book
- Degrees in Computer Science



# 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: malware clustering
- Lunch
- **Supervised Learning**
  - Lab: DGA Classification
- Review

# About the VM


darklabs.bah.com/merlin

# About the VM





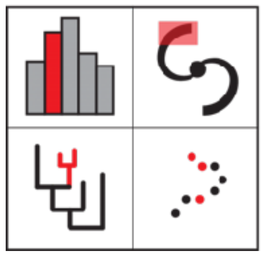
PostgreSQL


Scala

ERLANG



kibana


APACHE
DRILL



gedit

H2O.ai

julia

node
JS



mongoDB

jupyter

Java

neo4j

MySQL

hadoop

HIVE

APACHE
HBASE

# What are we Talking About?

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



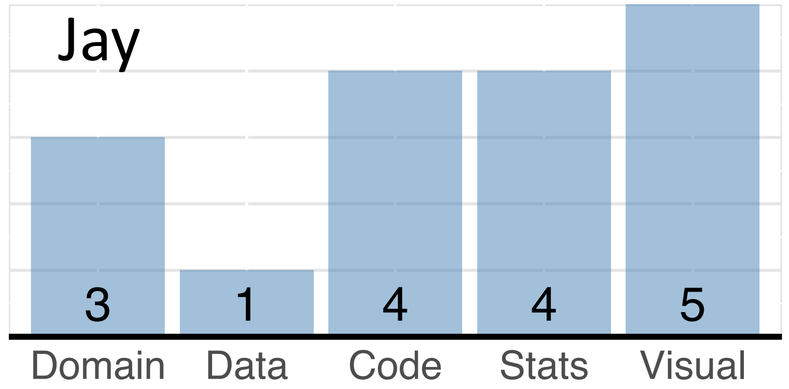
Definitions from Wikipedia

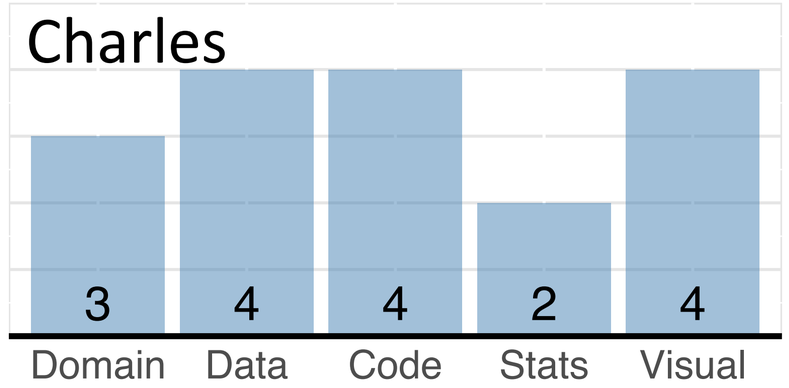
# Where is the I.T. / Security Data Science?

- 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…

# Roles/Skills within Data Science

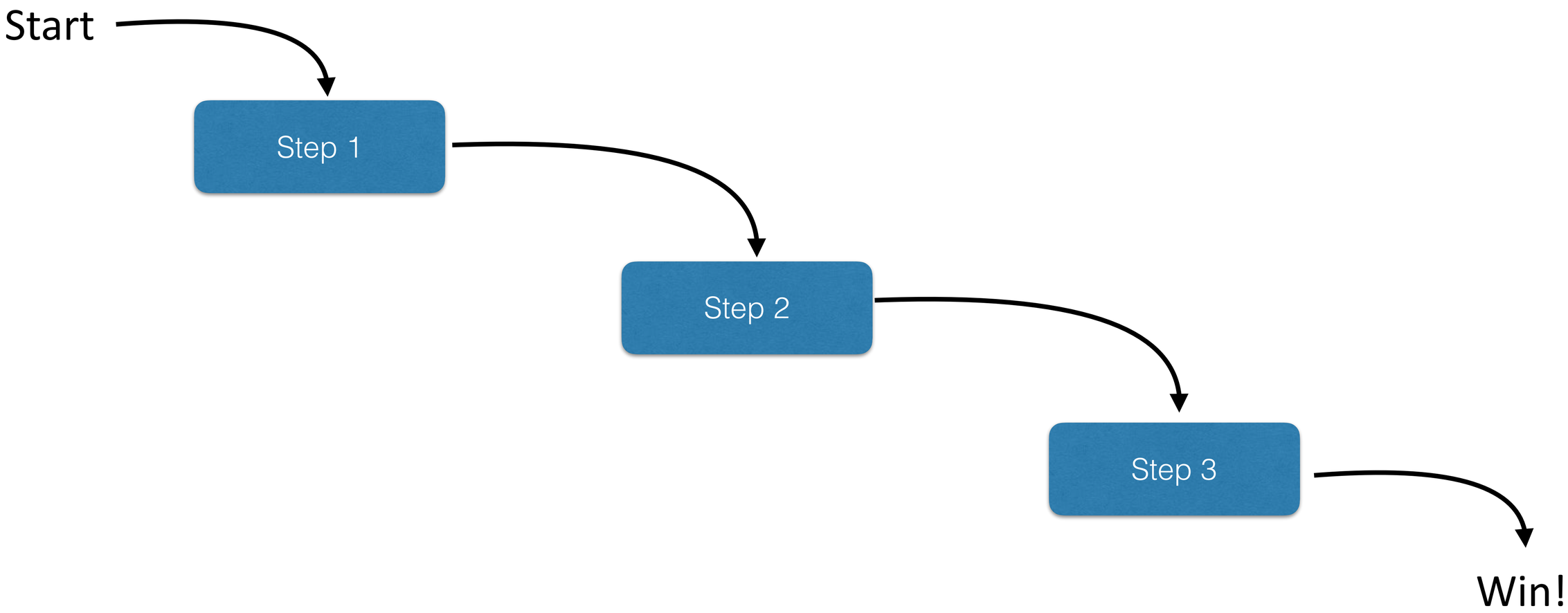
- **Domain Expertise**
  Setting and maintaining purpose in analysis
- **Data Engineering**
  Being able to prepare, store and maintain data
- **Programming**
  The glue the connected everything
- **Statistics/Machine Learning**
  To learn from the data (or not be fooled by data)
- **Visualization**
  Communicating the results effectively



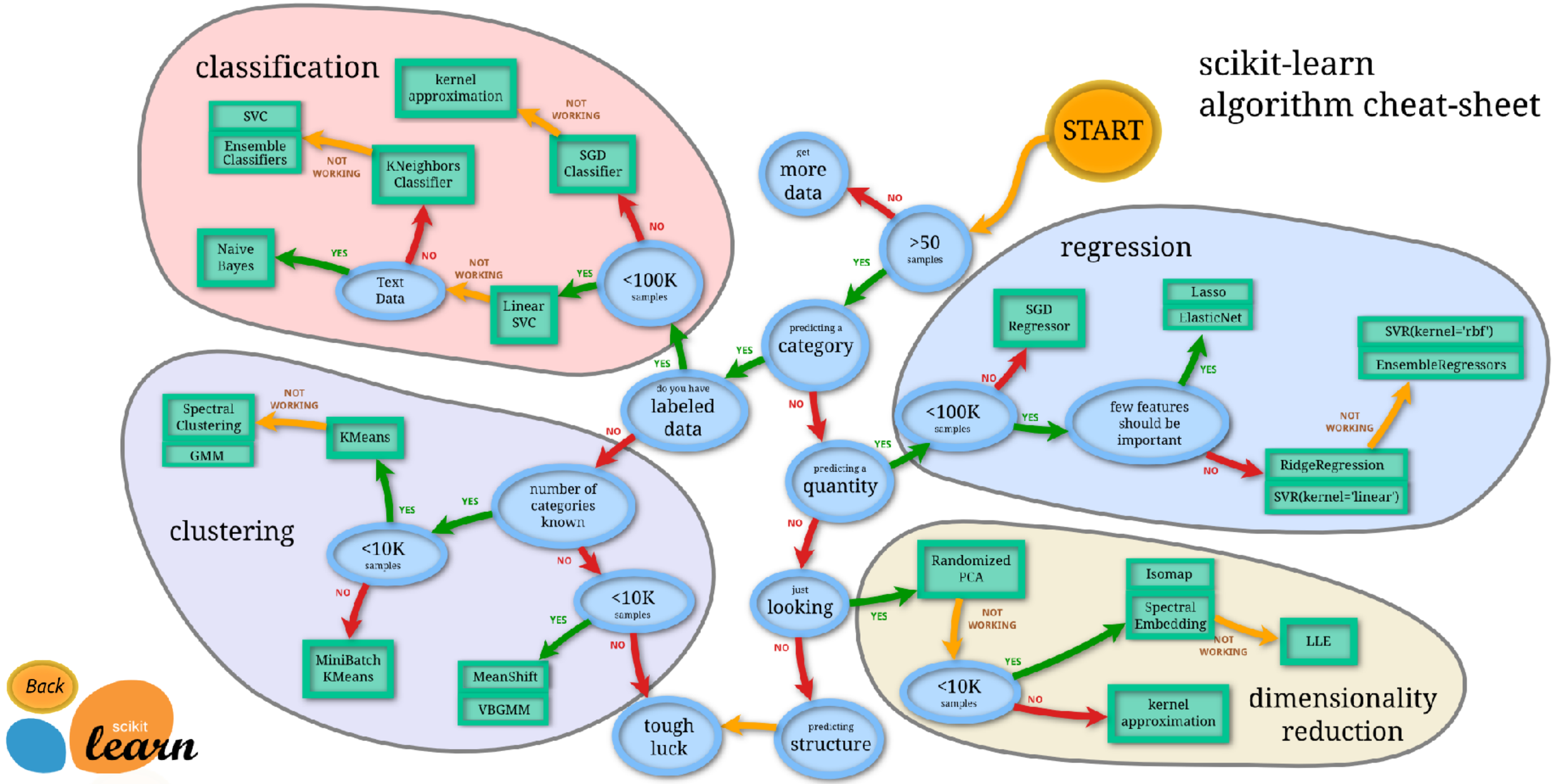



# Data Science Workflow

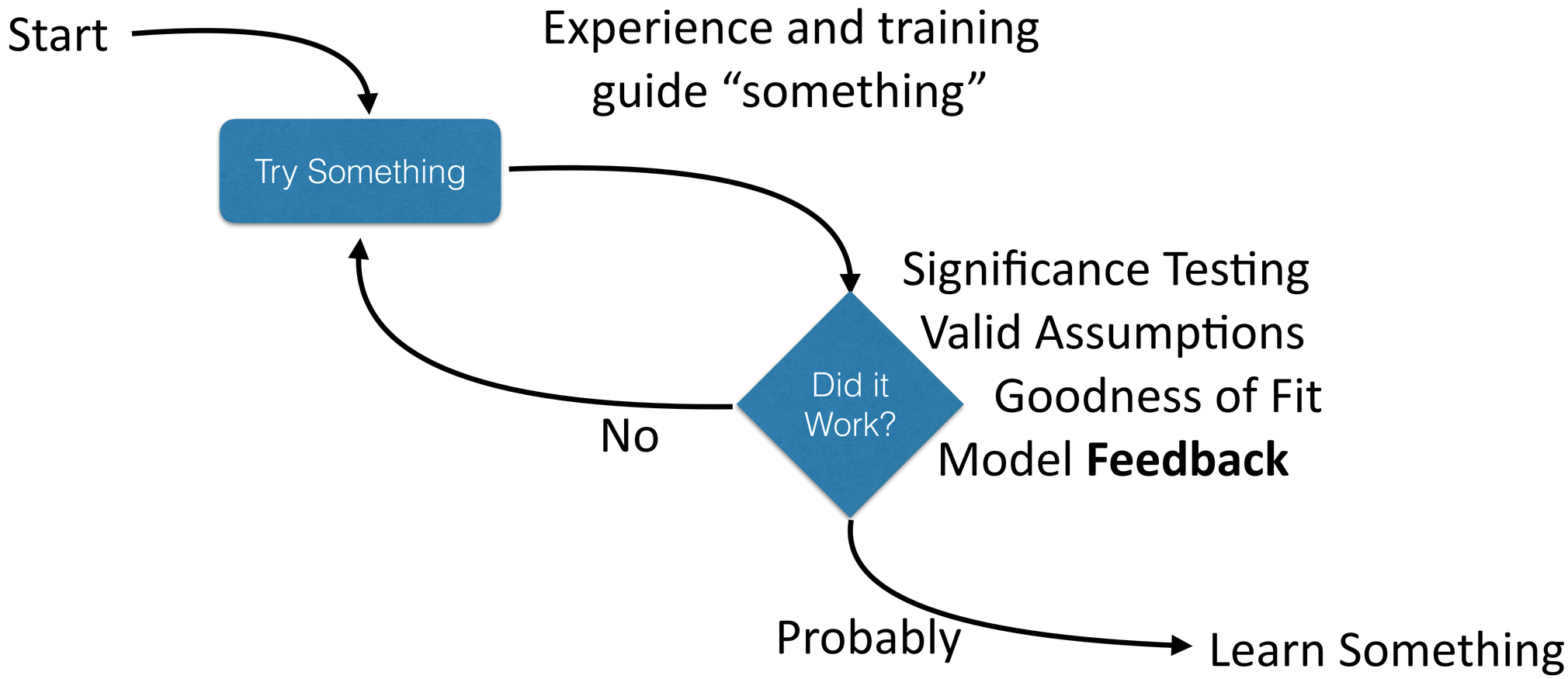
# What Data Science is Not



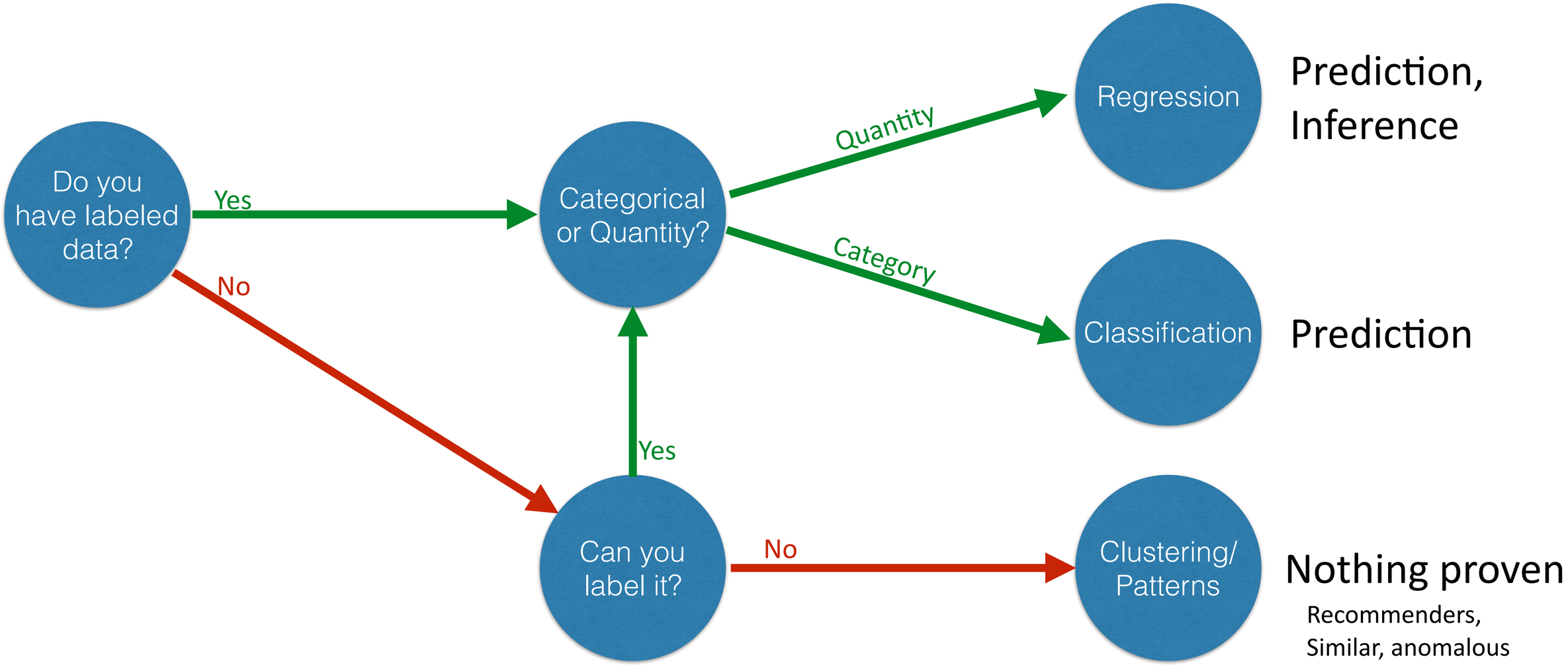
# What Data Science is Not (sorry sklearn)



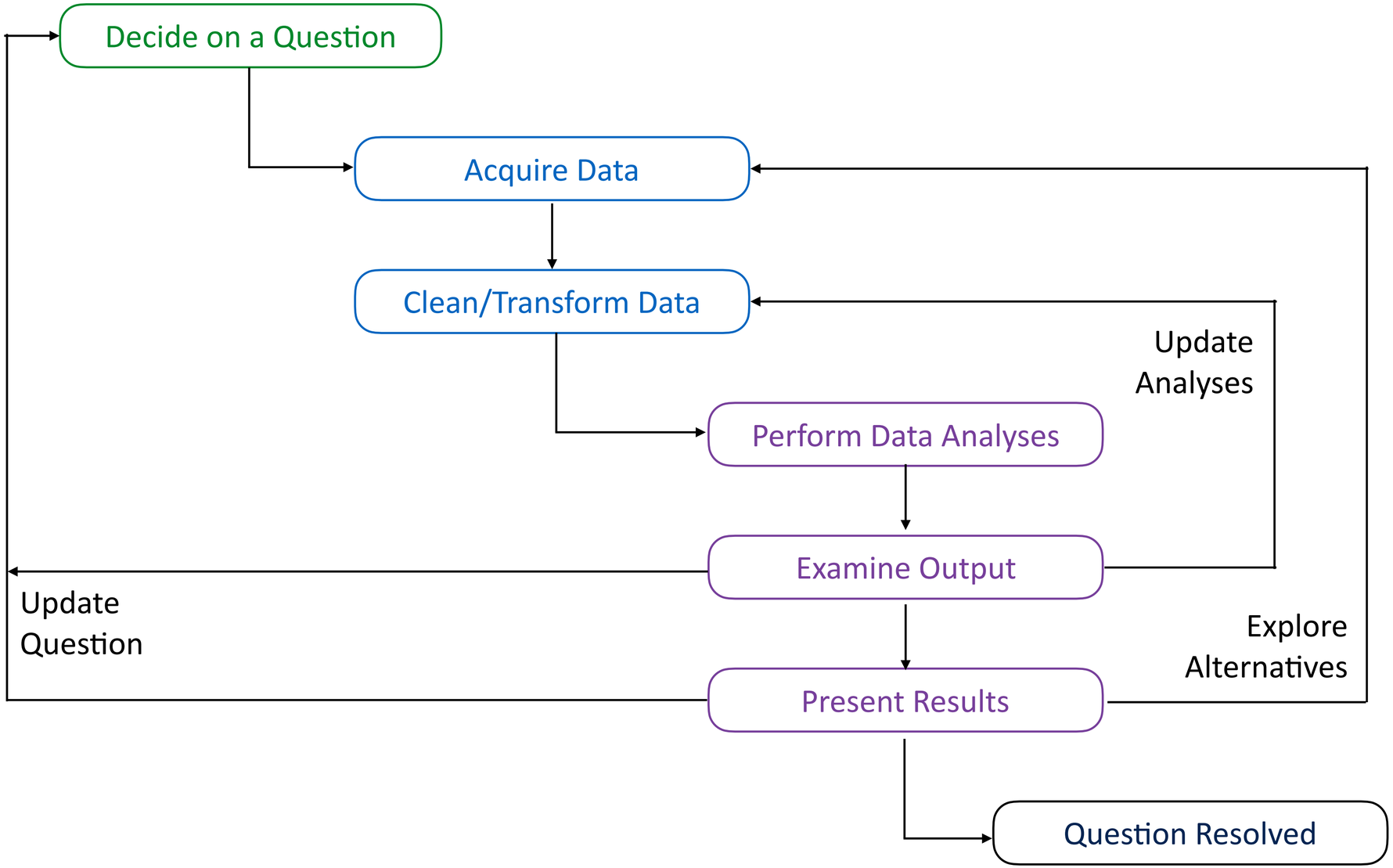
# What Data Science is



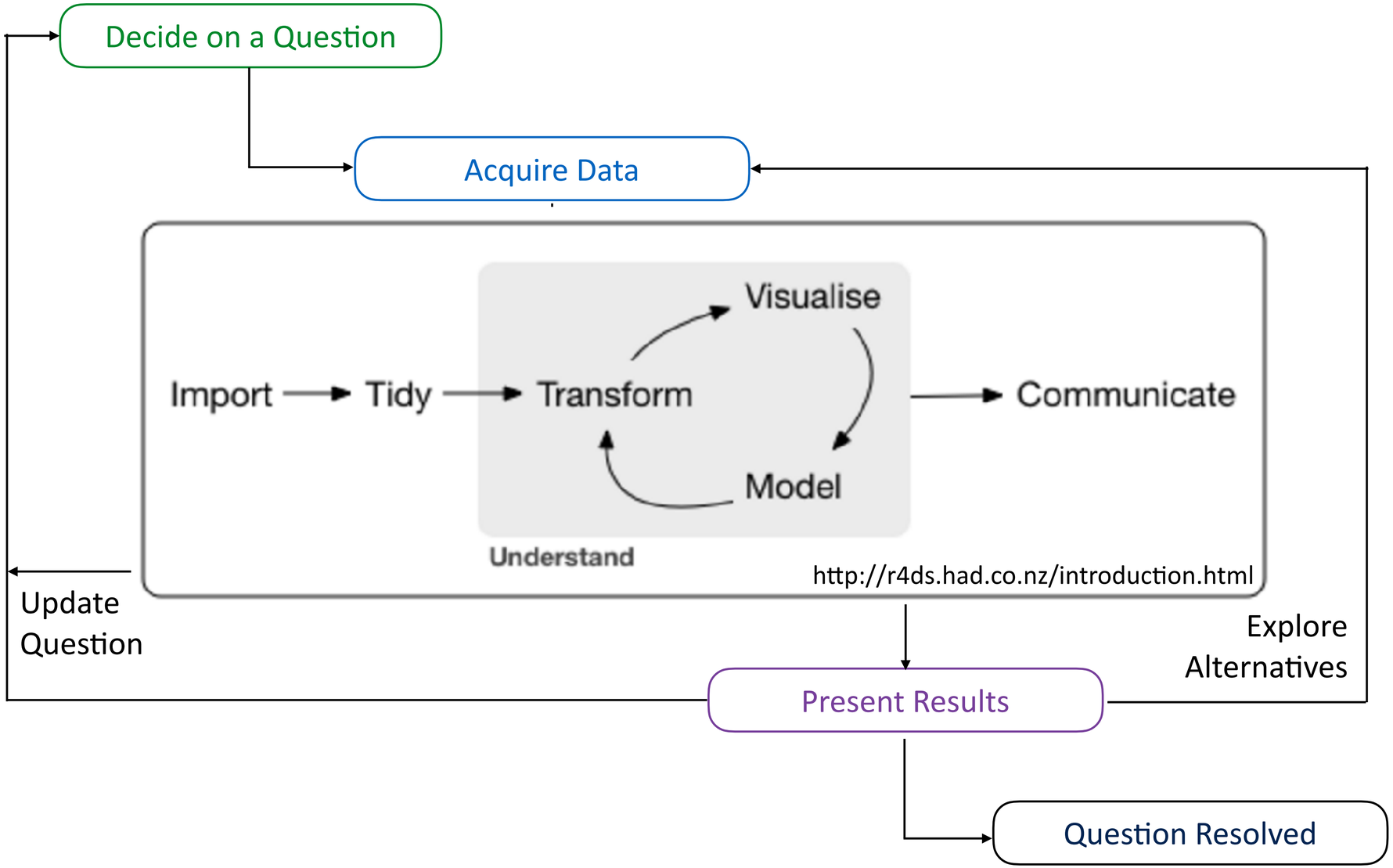
# What Data Science kind of is…



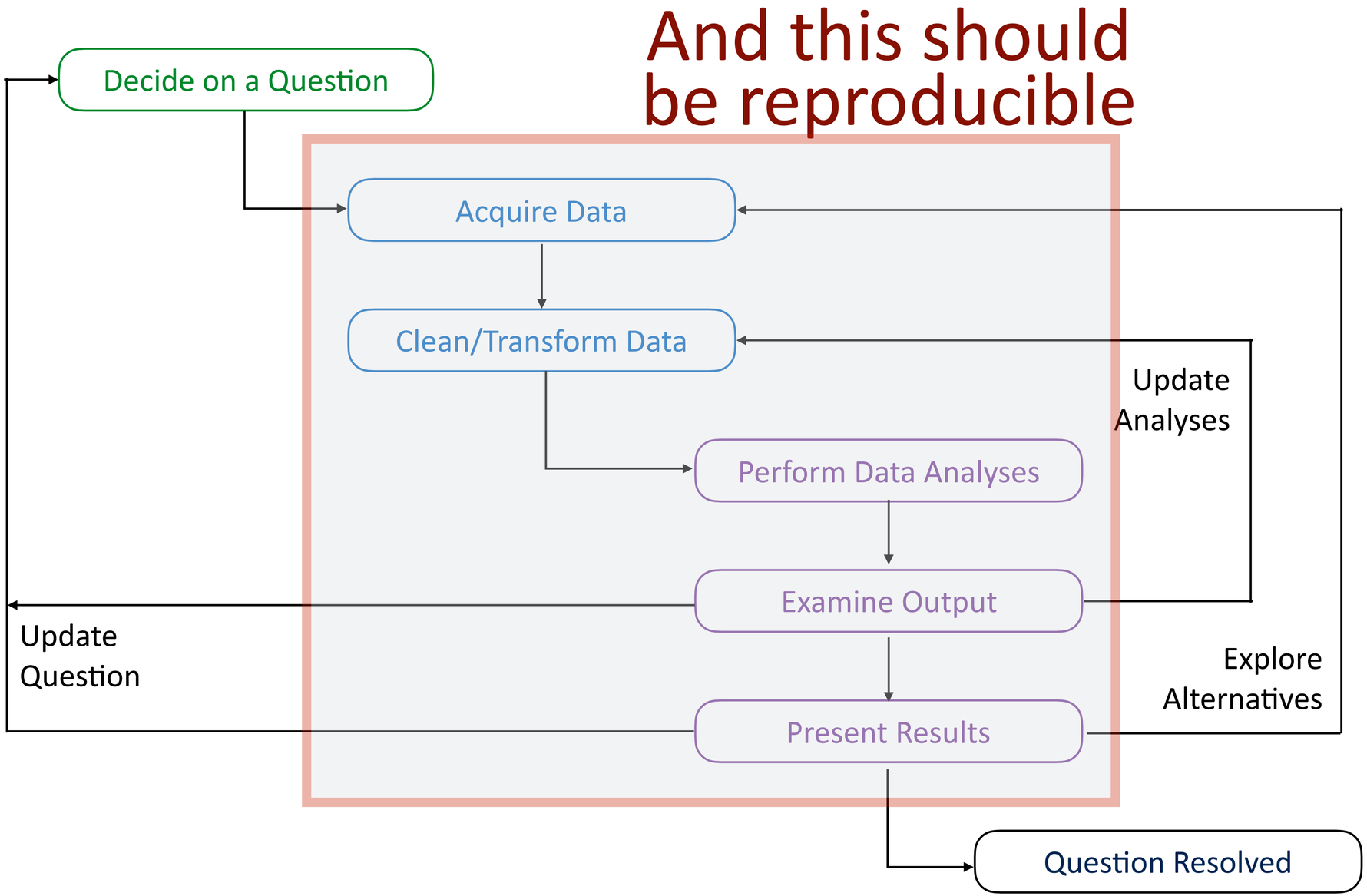
# Research Workflow



# Research Workflow

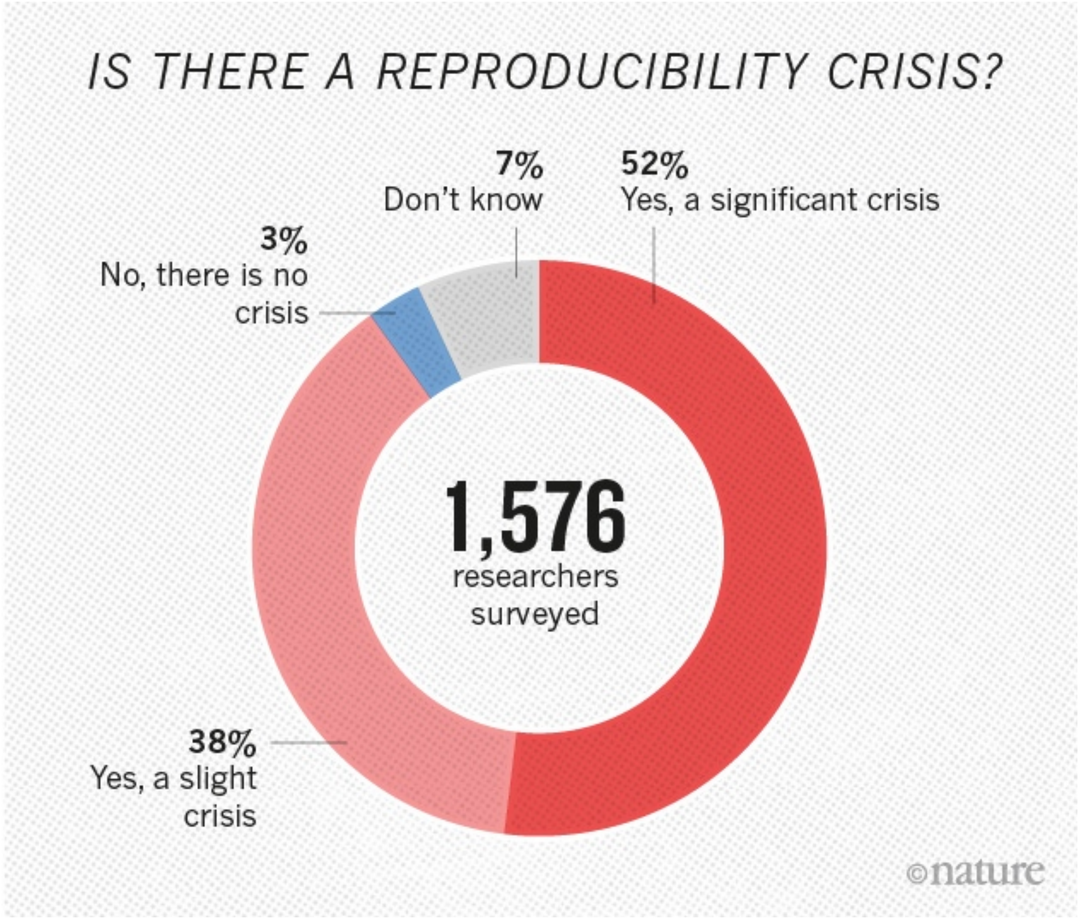


# Research Workflow



# 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

oreillysecuritycon.com

#oreillysecurity

# Getting into coding

# Our Tools

**Python**

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

---

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

**R**

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

---

- 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

R

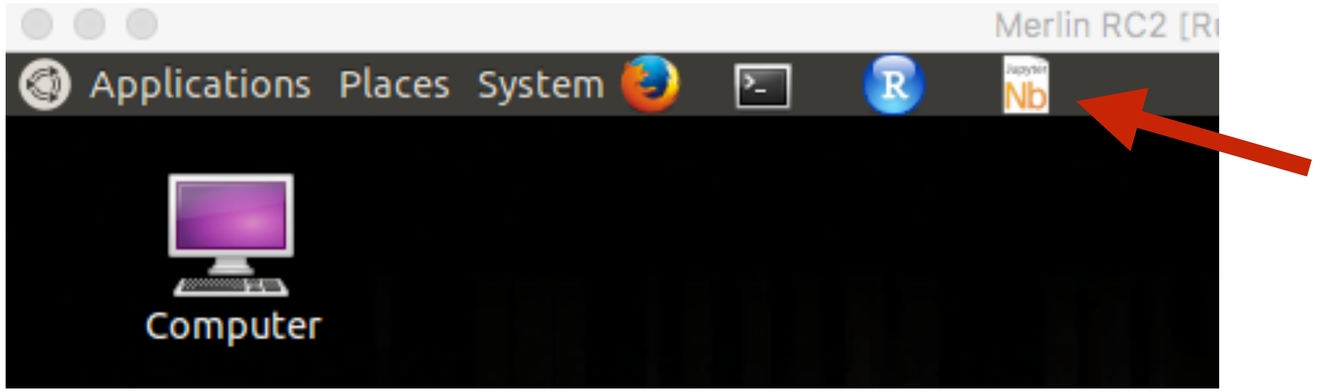
- RStudio
- RMarkdown
- dplyr
- ggplot2
- “tidyverse”

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


darklabs.bah.com/merlin

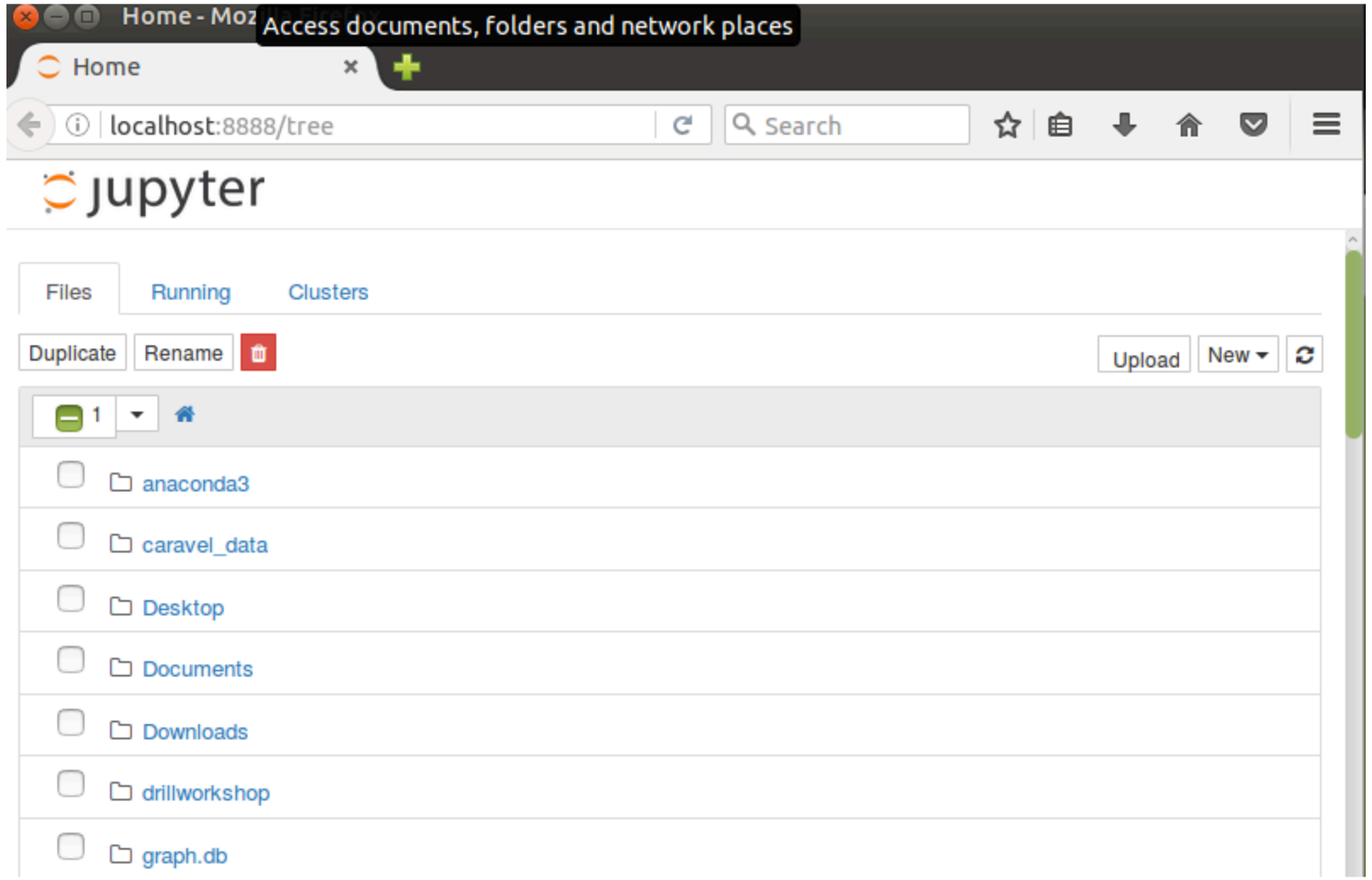
# Python

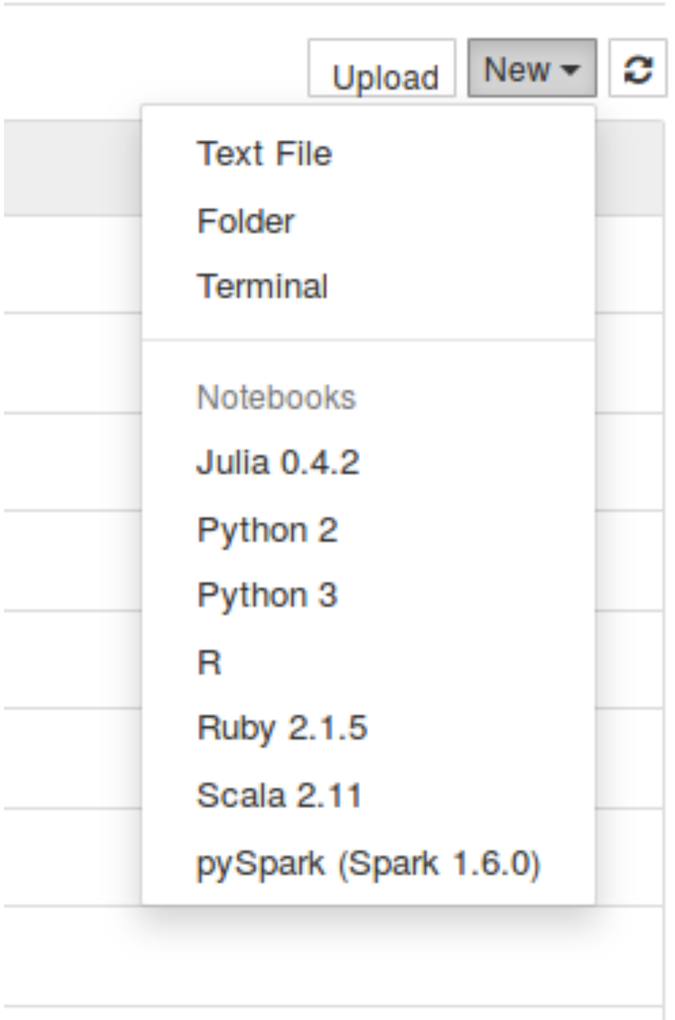
Step 1:  Launch Jupyter



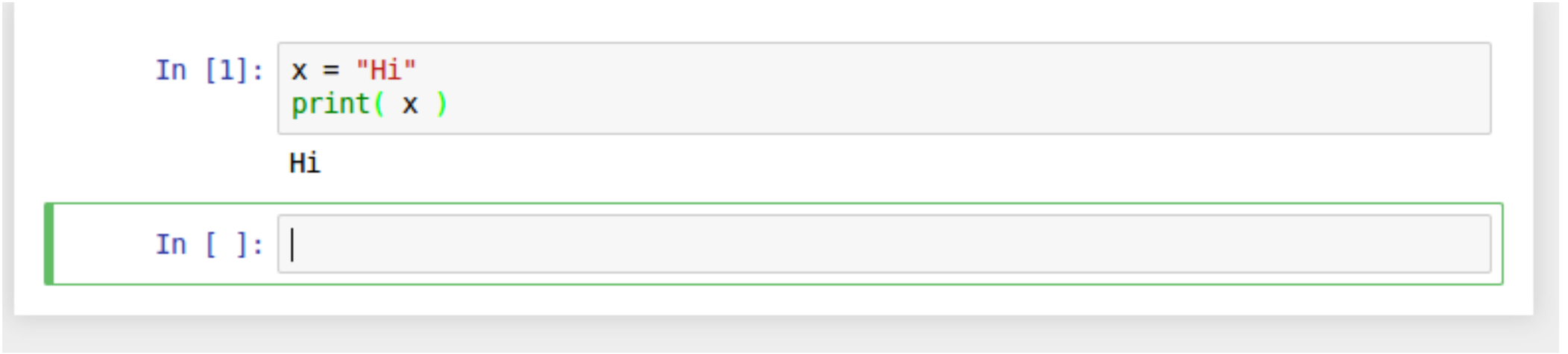
# Python

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

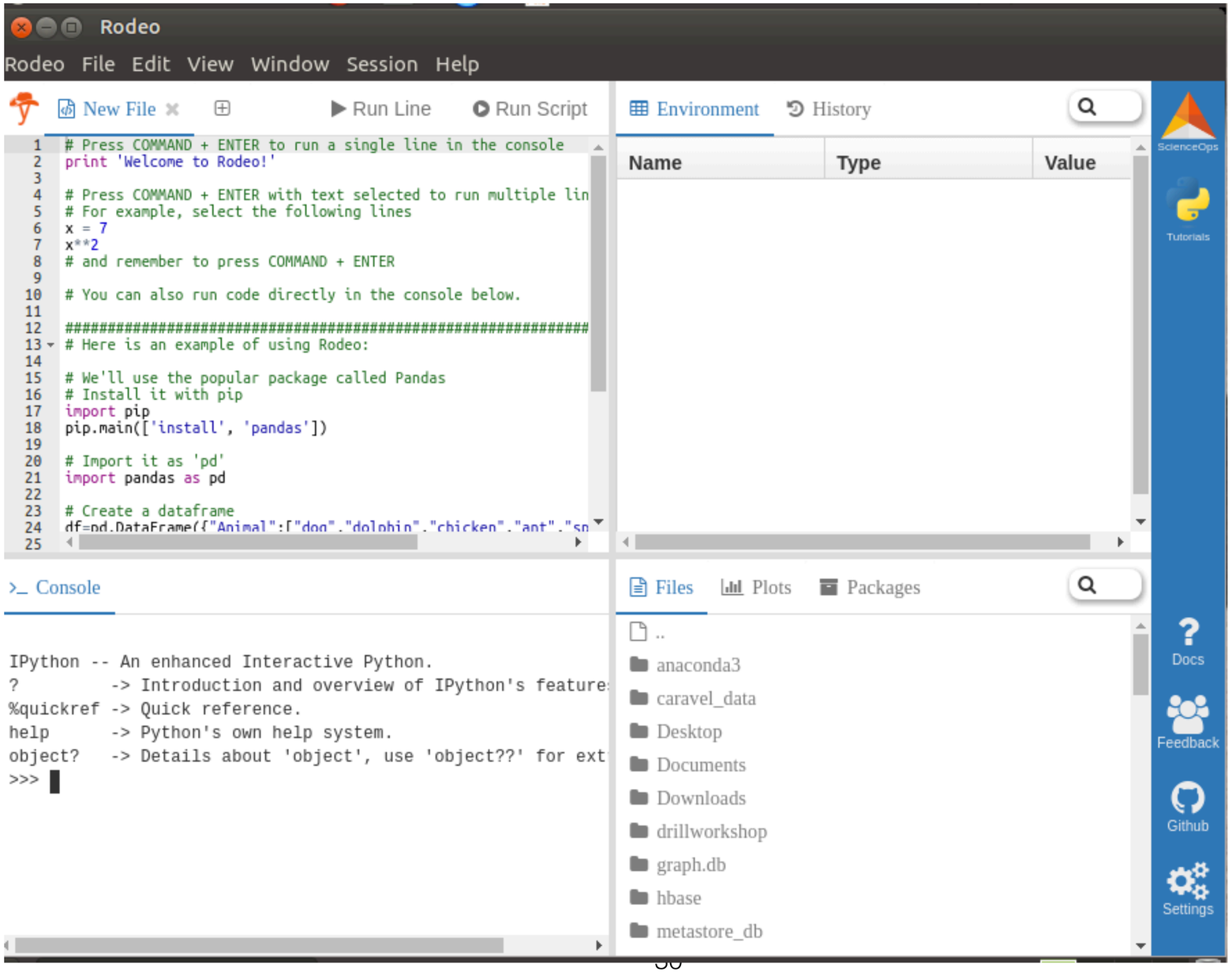


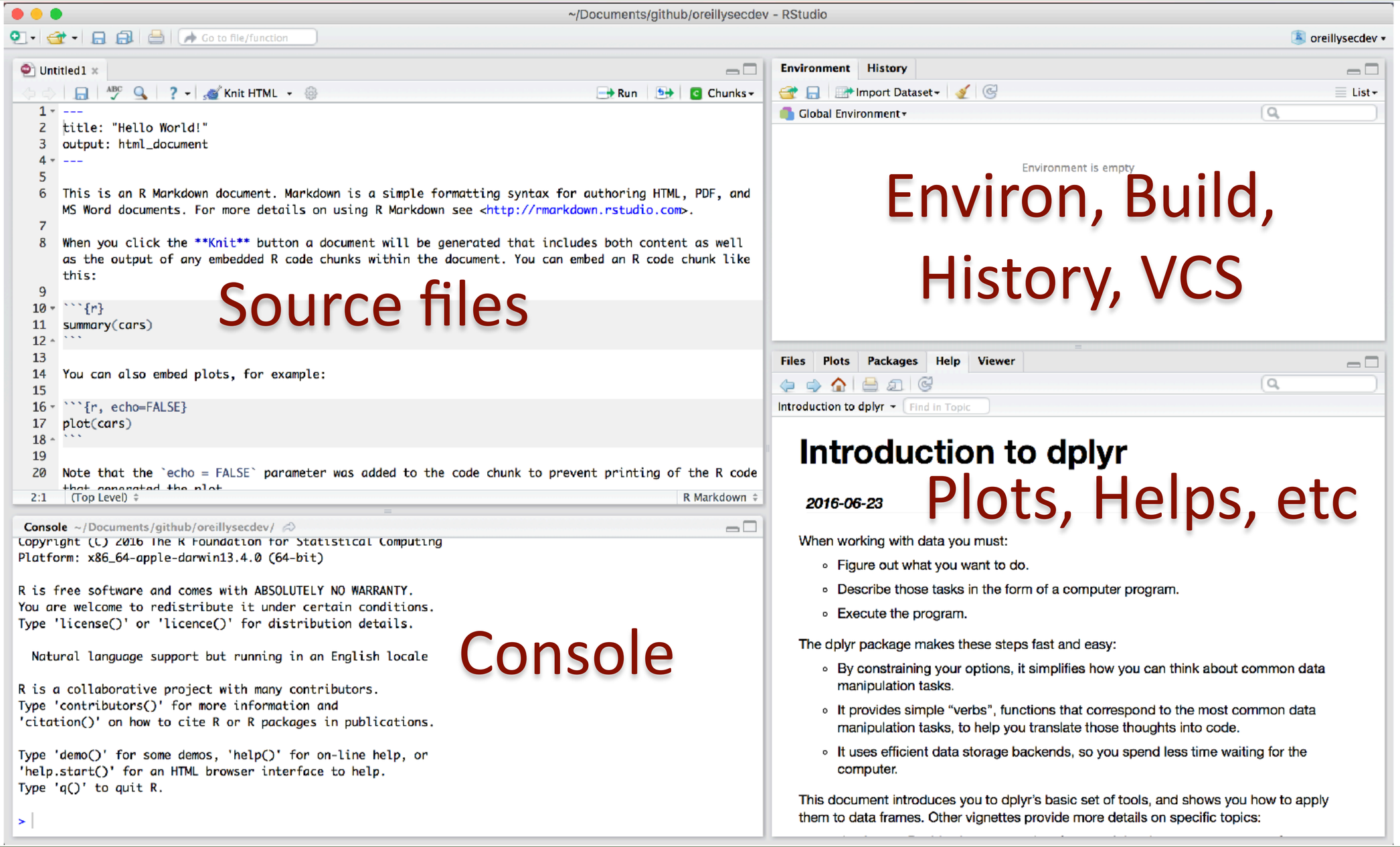
## Step 3:  Start Coding!



# Python: Rodeo



# RStudio


~/Documents/github/oreillysecdev - RStudio
Go to file/function
oreillysecdev
Untitled1
Knit HTML
Run
Chunks
1 ---
2 title: "Hello World!"
3 output: html_document
4 ---
5
6 This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.
7
8 When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:
9
10 ```{r}
11 summary(cars)
12 ```
13
14 You can also embed plots, for example:
15
16 ```{r, echo=FALSE}
17 plot(cars)
18 ```
19
20 Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code
2:1
(Top Level)
R Markdown
Source files
Environment
History
Import Dataset
List
Global Environment
Environment is empty
Environ, Build, History, VCS
Files
Plots
Packages
Help
Viewer
Introduction to dplyr
Find in Topic
Introduction to dplyr
2016-06-23
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:
Plots, Helps, etc
Console ~/Documents/github/oreillysecdev/
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Console

# Getting Documentation and help

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

**Python**

---

**R**

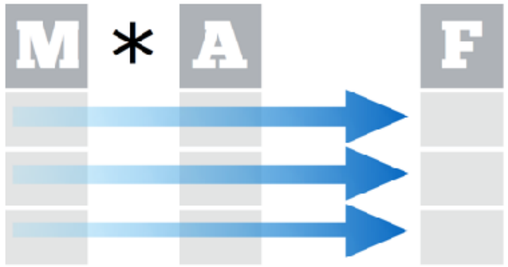
- Google
- help(“<function>”)
- ?<function>
- ??<search for word>

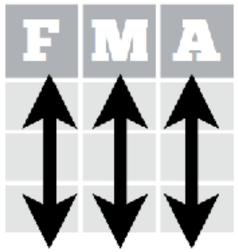
# Concepts

Data concepts differ from development concepts

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

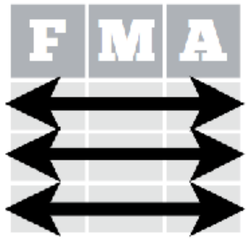
F = M * A




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
})

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

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

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] |

# The Startup env

```
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

Python

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

R

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

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

R

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

# 1-Dimensional Series/Vectors

Python

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

R

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

# 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
```

Python

Can also use “tail”

R

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

# Applying a function to a 1-D Set

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

x = pd.Series( [1,2,3,4,5] )
x.apply( addTwo )
0    3
1    4
2    5
3    6
4    7
dtype: int64
```

```
# OR
x.apply( lambda n: n + 2 )

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

Python

R

```
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
```

# Creating 1-Dimensional

Python

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

R

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

# Creating 1-Dimensional: filtering

Python

```
# randomNumbers[ <boolean> ]

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

R

```
# 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
```

# Creating 1-Dimensional: string manipulation

Python

| 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 |

R

| 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) |
| 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 |

"stringr" is the package for fast string manipulation

# Creating 1-Dimensional: string manipulation

Python

```
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] ) )
```

R

```
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)
```

# 1-D Statistical Operations

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

# 1-D Exploratory Operations

```
Series.describe()
```

# Creating 1-Dimensional: data inspection

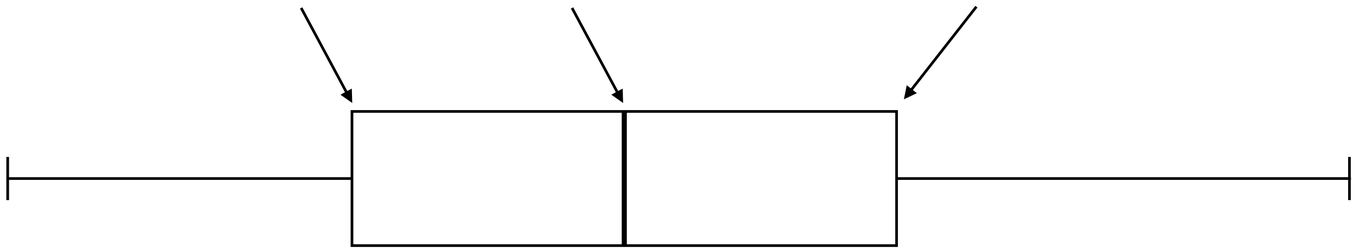
```
>>> Series.describe()
count      50.000000
mean       52.000000
std        29.154759
min         3.000000
25%        27.500000
50%        52.000000
75%        76.500000
max       101.000000
dtype: float64
```

Tukey's "Five Number Summary"

Python

R

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



# 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)
> summary(sizes)
 large medium  small
     3      1      2
```

Factors tell R “Treat this as a categorical variable”

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
> table(sizes)
sizes
 large medium  small
     3      1      2
```

Still a factor

R

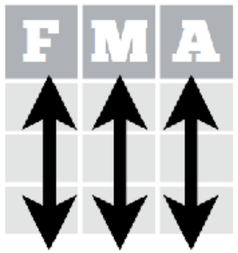
# {LAB} time

Please Complete “One Dimensional Worksheet”

# 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
```



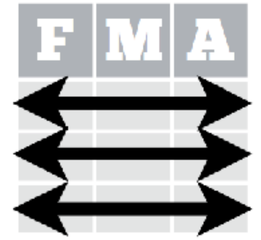

Each **variable** is saved in its own **column**

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
```

R




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

# Loading Various Sources

| Function | Python | R |
|---|---|---|
| Read from CSV | `pd.read_csv( <file> )` | `readr::read_csv( <file \| URL> )` |
| Read Excel Doc | `pd.read_excel( <file>, sheetname=<sheetname> )` | `readxl::read_excel( <file>, <sheet> )` |
| Read JSON | `pd.read_json( <file \| URL> )` | `rjson::fromJSON( <file \| URL> )` |
| Read SQL | `pd.read_sql( <query>, <connection> )` | `RODBC::sqlQuery( <conn>, <query> )` |
| Read HTML | `pd.read_html( <source> )` | `readLines( <file \| URL>)`<br>`rvest::html( <src> )` |

# 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

Python

```
> 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
```

R

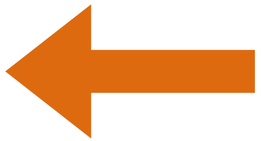
```
> iris <- read.csv("iris.csv")    # use read_csv for tibble

> 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
```

# 2-Dimensional: functions by variable

```
>>> iris['Sepal.Length'].mean()
5.843333333333337
>>> iris['Sepal.Length'].min()
4.299999999999998
>>> 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

Python

```
# 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']]
```

R

```
# 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')]
```

# Data frames: Subsetting

Python

```
# 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
```

R

```
# 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
```

# 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)

https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
Search for "dplyr cheat sheet"

# {LAB} time

Please Complete “Two Dimensional Worksheet”