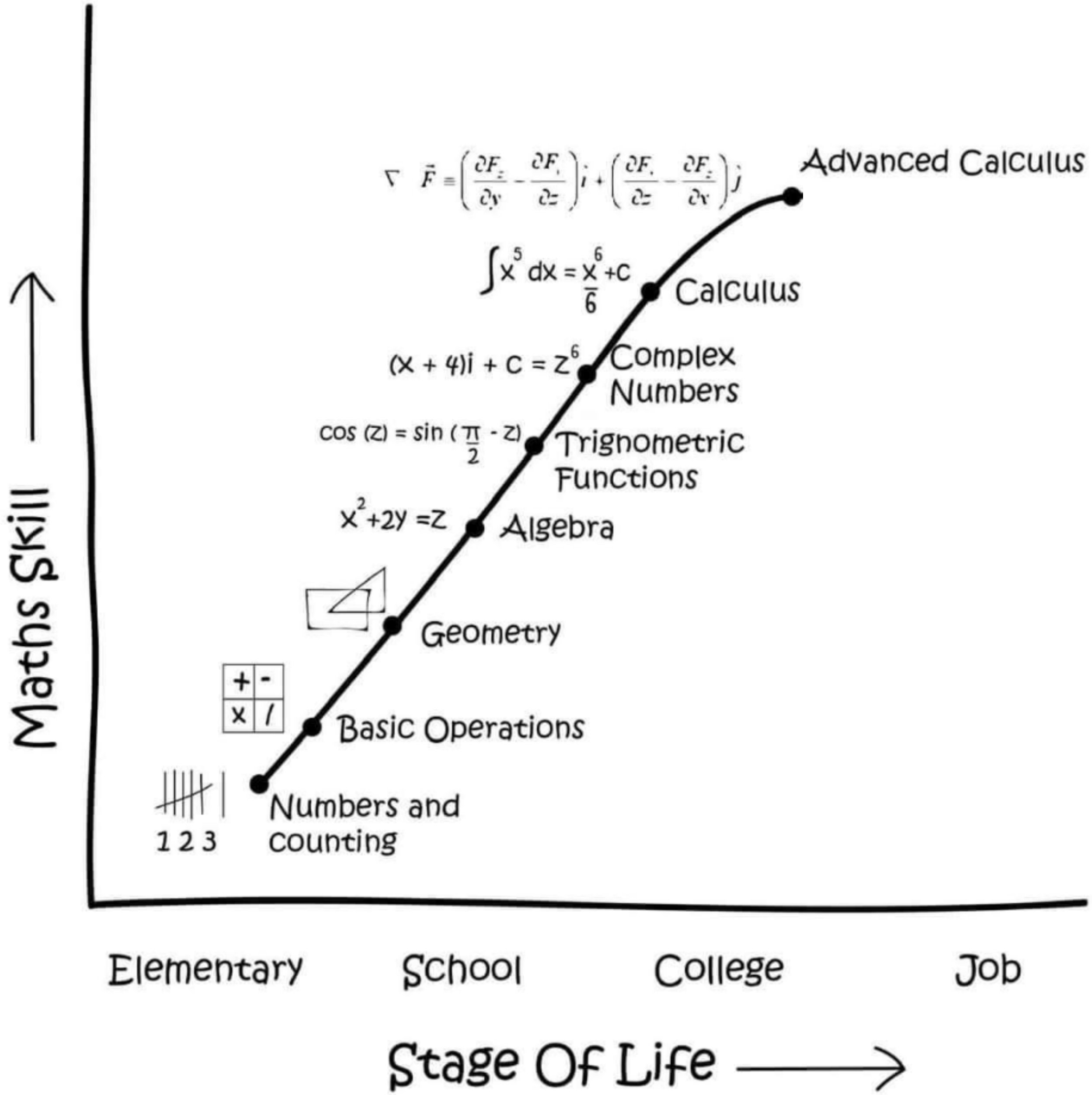
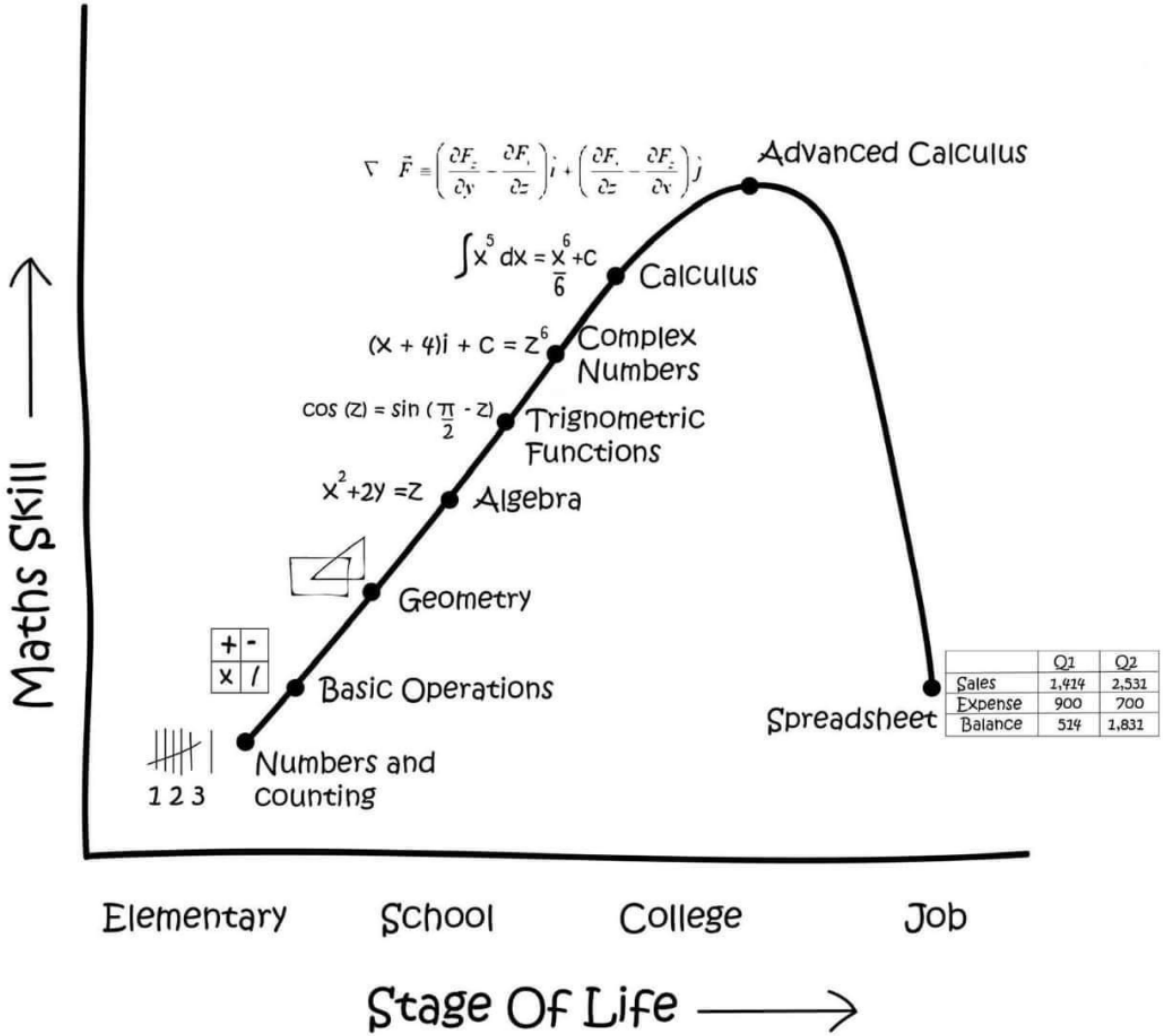
Data Science Tools for Interactive Exploration

“According to all the big data we’ve gathered, our discussions about big data are up 72% this year alone.”



Advanced Calculus
Calculus
Complex Numbers
Trignometric Functions
Algebra
Geometry
Basic Operations
Numbers and counting
1 2 3
Maths Skill
Elementary
School
College
Job
Stage Of Life

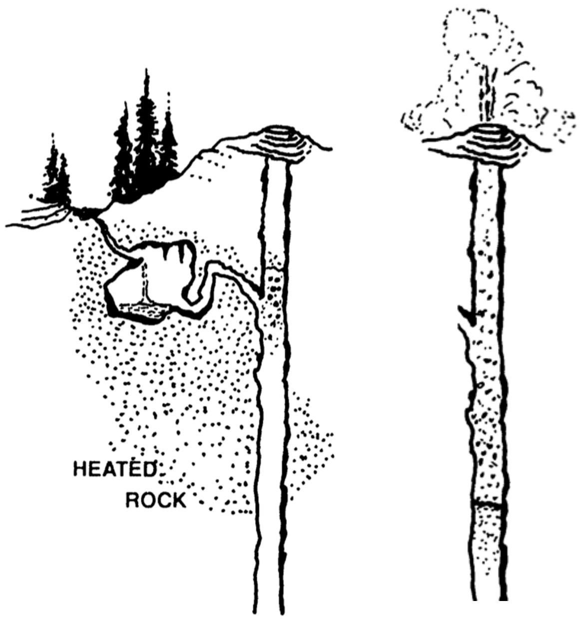


Advanced Calculus
Calculus
Complex Numbers
Trignometric Functions
Algebra
Geometry
Basic Operations
Numbers and counting
1 2 3
+ - x /
x^2+2y =Z
cos (Z) = sin (π/2 - Z)
(X + 4)i + C = Z^6
∫x^5 dx = x^6/6 +C
Spreadsheet
Q1 Q2
Sales 1,414 2,531
Expense 900 700
Balance 514 1,831
Maths Skill
Elementary
School
College
Job
Stage Of Life

Predicting a Geyser's Eruptions

CERN School of Computing

Physics of a Geyser



Long column of water heated from the bottom

Pressure at bottom high, raises boiling point

Eventually, bottom does start to boil

Bubbles rise, start to push out water

Pressure reduces, so boiling point reduces

Entire column flashes into steam and jets upwards

Top of column ends up empty

Water enters, starts to warm up, process repeats

csc-exercises > ... > OldFaithful
(autosaved)

FILE EDIT VIEW INSERT CELL KERNEL WIDGETS HELP

Trusted | Python 3

Markdown

Memory: 2.9 GB / 8 GB

Old Faithful

In [1]:
```
# Data file in this notebook is from https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat
# The original paper is available as https://tommasorigon.github.io/StatI/approfondimenti/Azzalini1990.pdf
```

In [2]:
```
# Standard definitions and options
from datascience import Table     # high-level abstraction
import pandas as pd               # mid-level data frames and series
import numpy as np                # low-level arrays and vectors

import matplotlib                 # plotting
matplotlib.use('Agg')                         # make nice screen plots
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')              # a particular plot format
plt.rcParams['figure.figsize'] = (10.0, 5.0) # wide plots to use space well
```

In [3]:
```
# Read in the data from a CSV file - headers taken from file
data = Table.read_table("oldfaithful.csv")
```

In [4]:
```
# Take a look at the data
data
```

Out[4]:

N	Duration	Interval
1	3.6	79
2	1.8	54
3	3.333	74
4	2.283	62
5	4.533	85
6	2.883	55

In [4]:
```
# Take a look at the data
data
```

Out[4]:

N	Duration	Interval
1	3.6	79
2	1.8	54
3	3.333	74
4	2.283	62
5	4.533	85
6	2.883	55
7	4.7	88
8	3.6	85
9	1.95	51
10	4.35	85

... (262 rows omitted)

In [5]:
```
# Old Faithful is famous for its repeatability - lets check some statistics
data[2].mean()             # data[2] is the Interval column
```

Out[5]: 70.897058823529406

In [6]:
```
data['Interval'].std()    # but we can also refer to it by name
```

Out[6]: 13.569960017586371

In [7]:
```
data['Interval'].min()
```

Out[7]: 43

In [8]:
```
data['Interval'].max()     # all the usual summary statistics are available
```

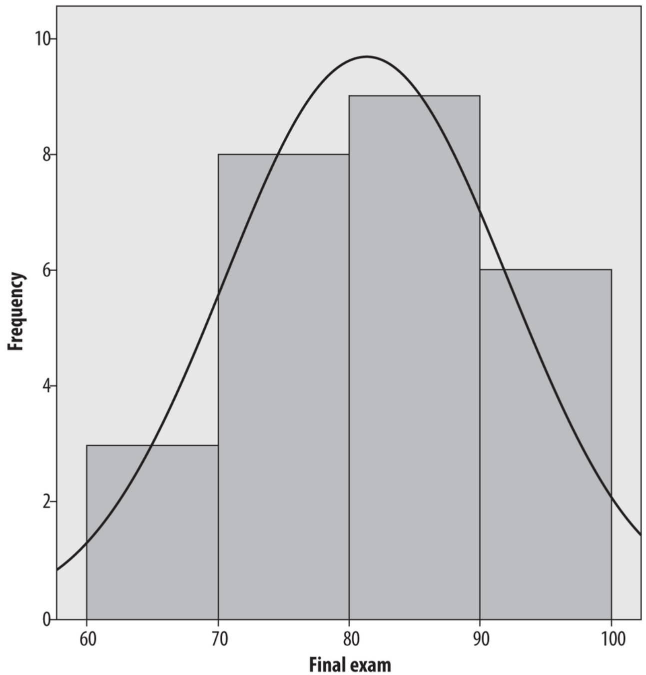
Out[8]: 96

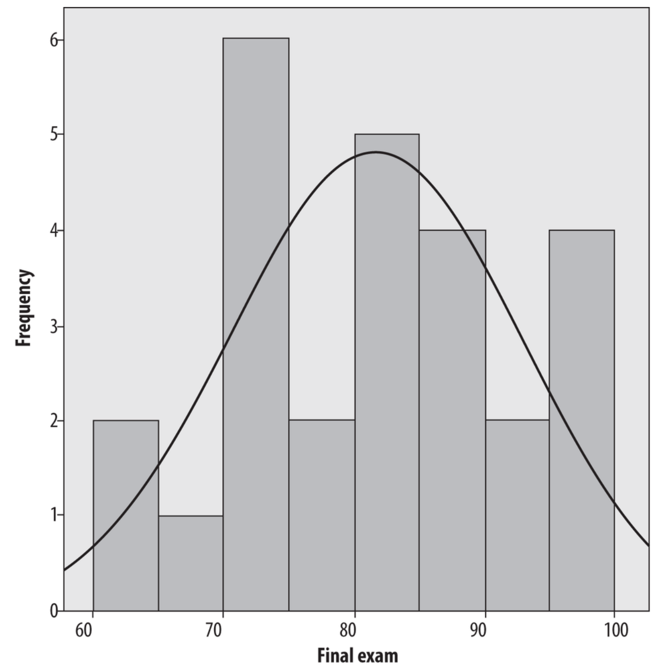
In [9]:
```
# While we're here, let's look at the other data we have
data['Duration'].mean(), data['Duration'].std()         # two statements on a line using commas
```

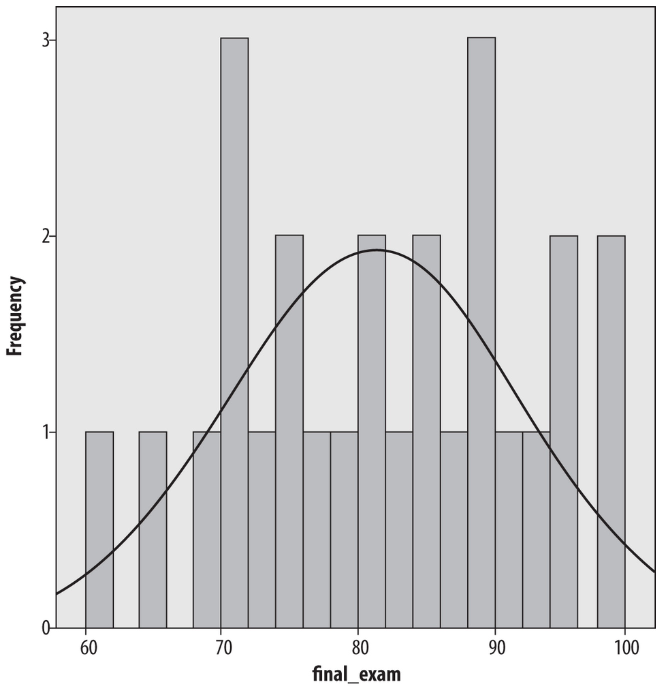
Out[9]: (3.4877830882352936, 1.139271210225768)

CERN School of Computing

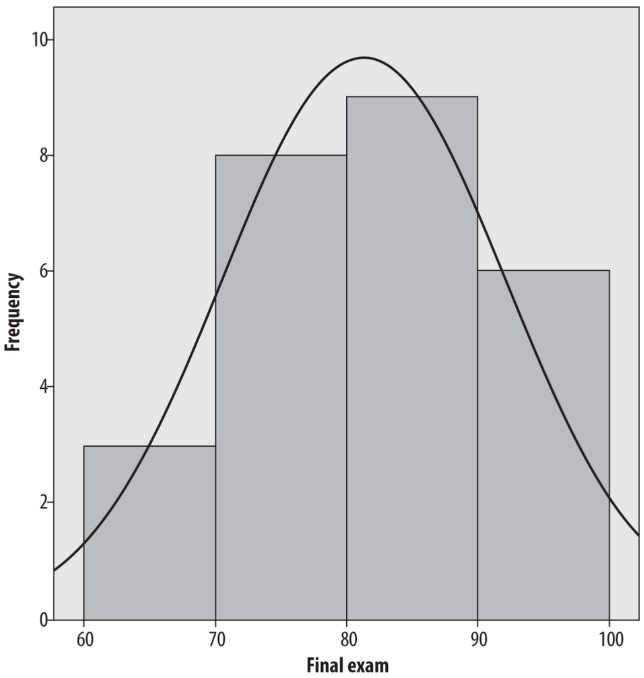
Before we plot: On binning





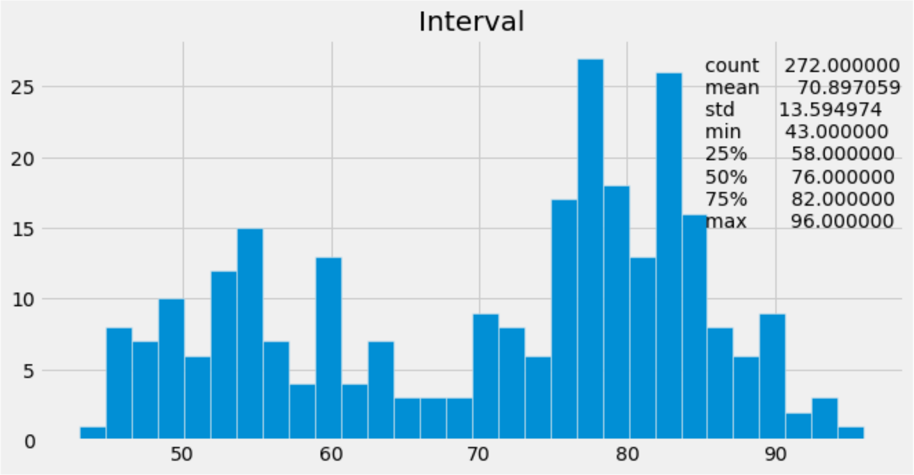


Frequency
3
2
1
0
60
70
80
90
100
final_exam

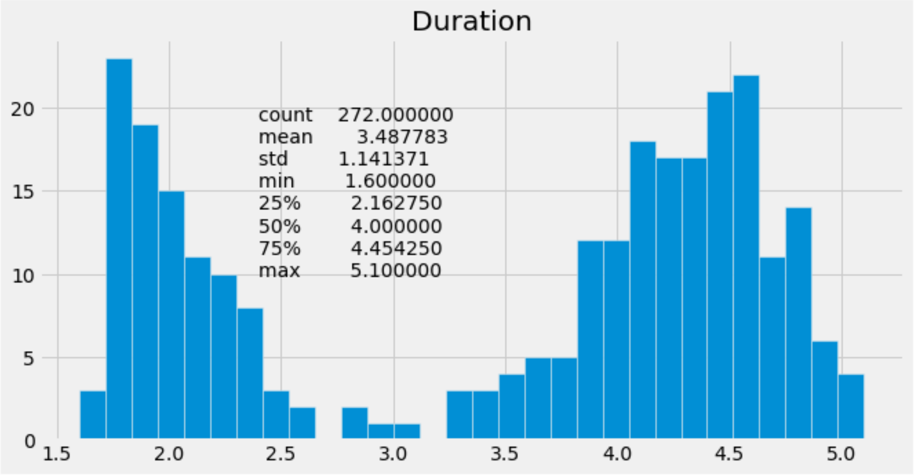


Frequency
10
8
6
4
2
0
60
70
80
90
100
Final exam

In [10]:
```
# Let's see what the distribution looks like
plt.hist(data['Interval'], bins=30)
plt.figtext(0.75,0.5, data.to_df()['Interval'].describe().to_string())   # add descripitive text block from pandas
plt.title("Interval");                                                     # semicolon suppresses printing value
```

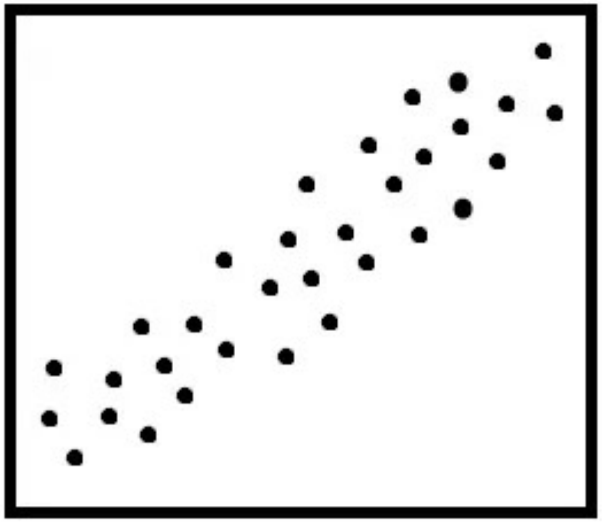


In [11]:
```
# Not particularly Gaussian!
# Maybe there's two peaks there. But that still doesn't give us a better way to predict the eruption.
# Look at other information we have:
plt.hist(data['Duration'], bins=30)
plt.figtext(0.3,0.4, data.to_df()['Duration'].describe().to_string())
plt.title("Duration");
```

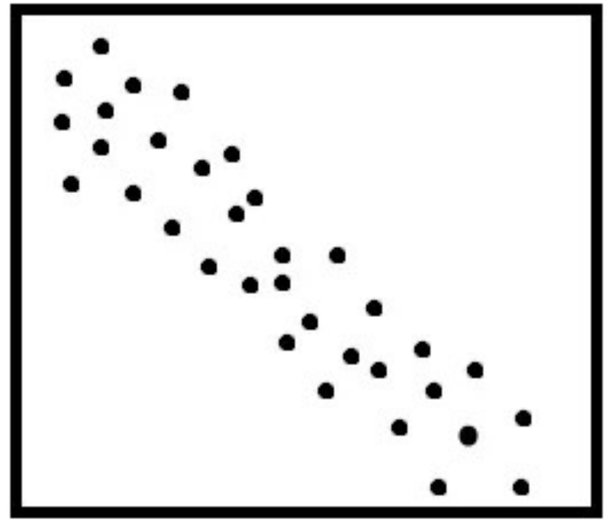


In [12]:
```
# Maybe there's a correlation?
np.corrcoef(data['Duration'], data['Interval'])
```

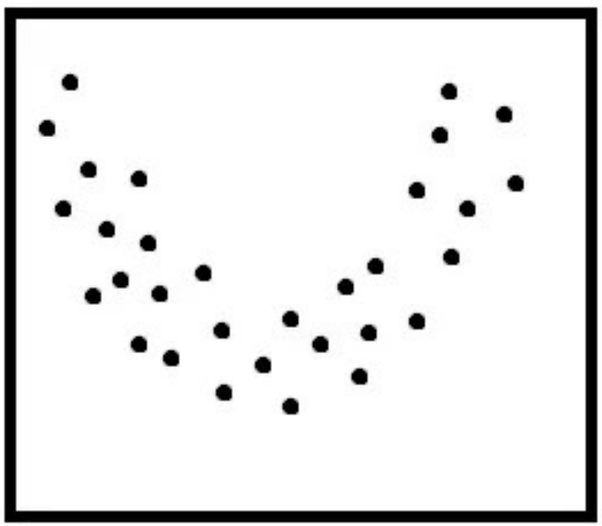
Out[12]:
```
array([[ 1.        ,  0.90081117],
       [ 0.90081117,  1.        ]])
```



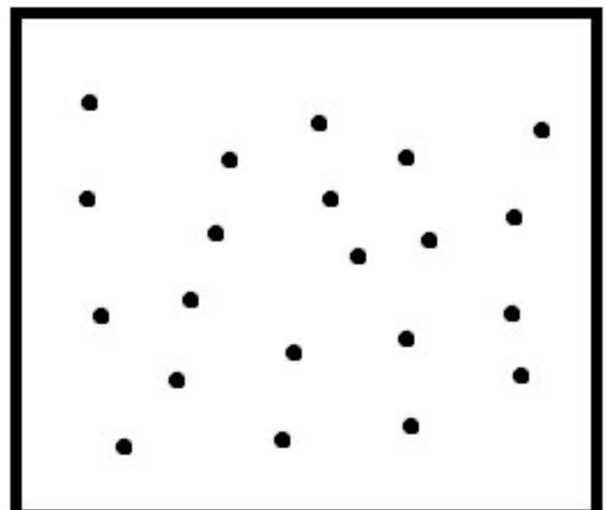
positive linear association



negative linear association



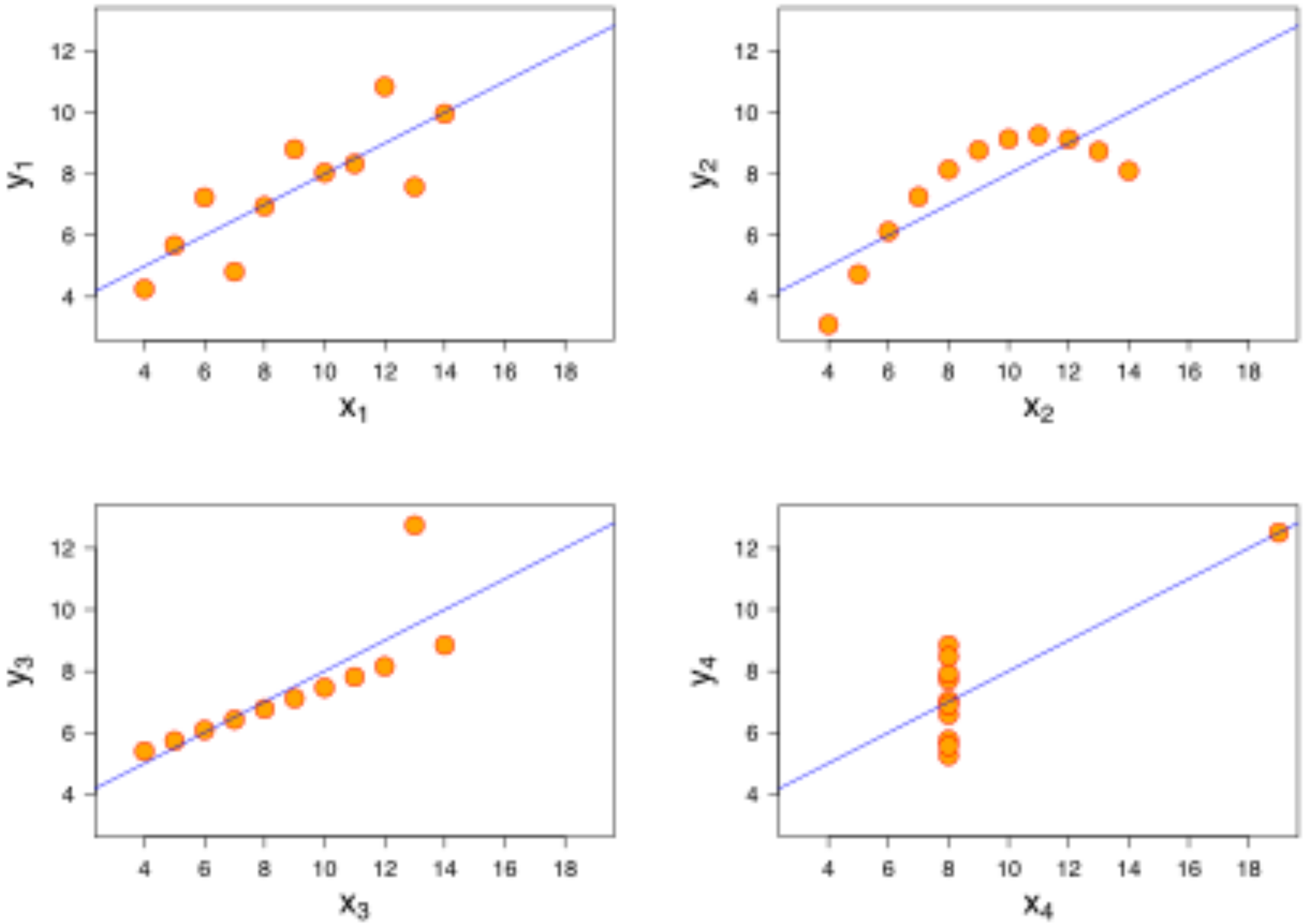
nonlinear association



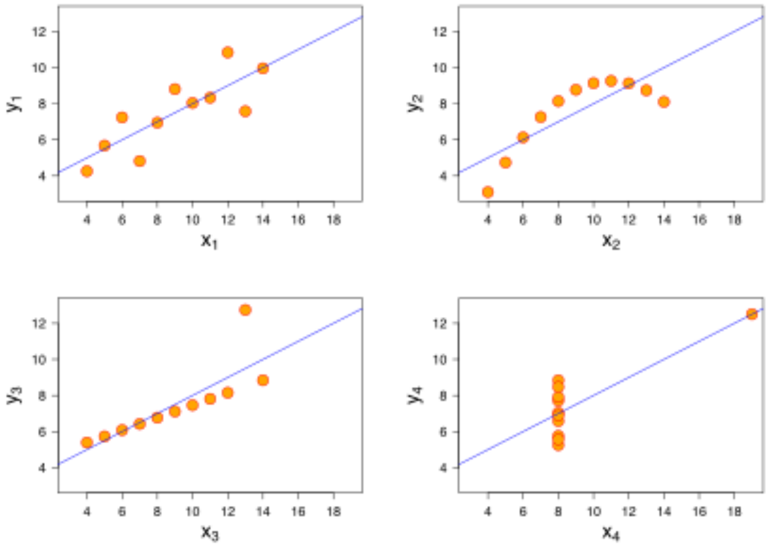
no association

CERN School of Computing

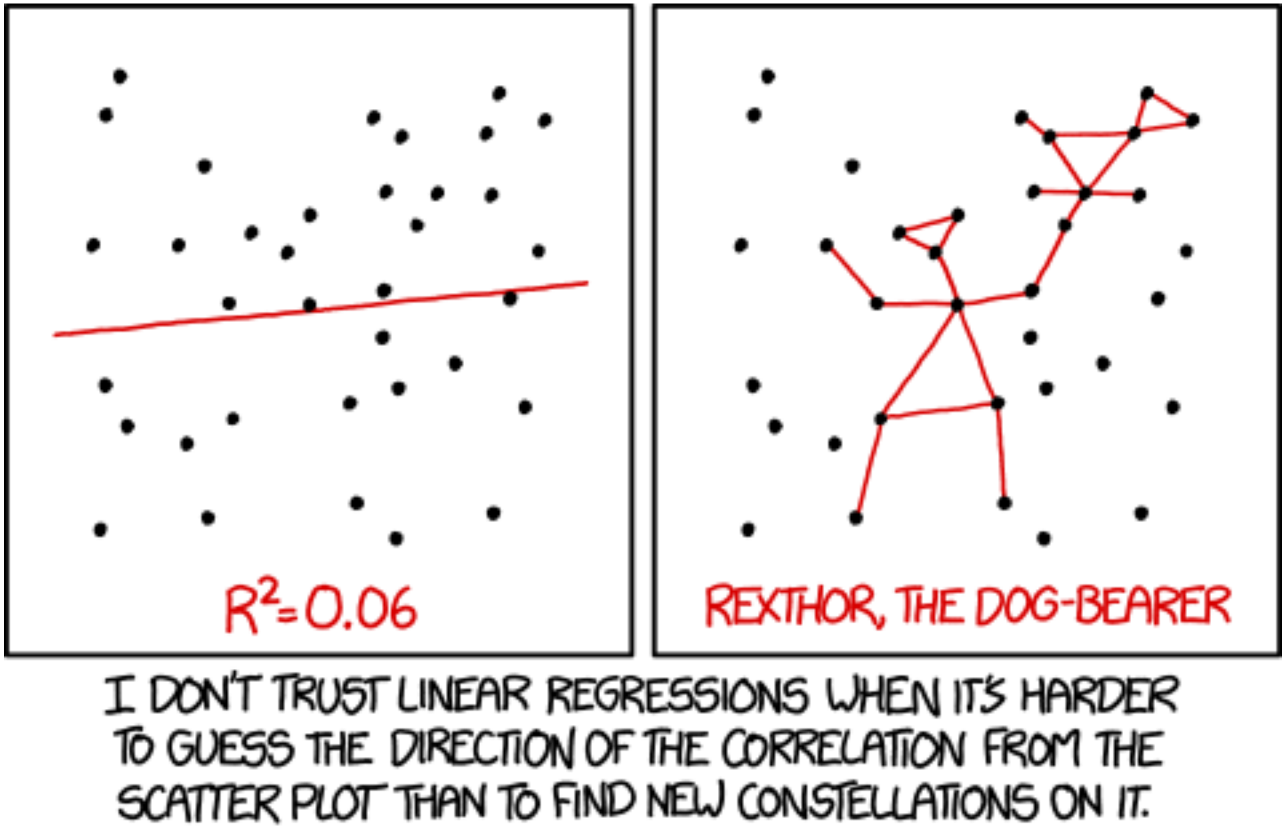
Anscombe's Quartet



Anscombe's Quartet

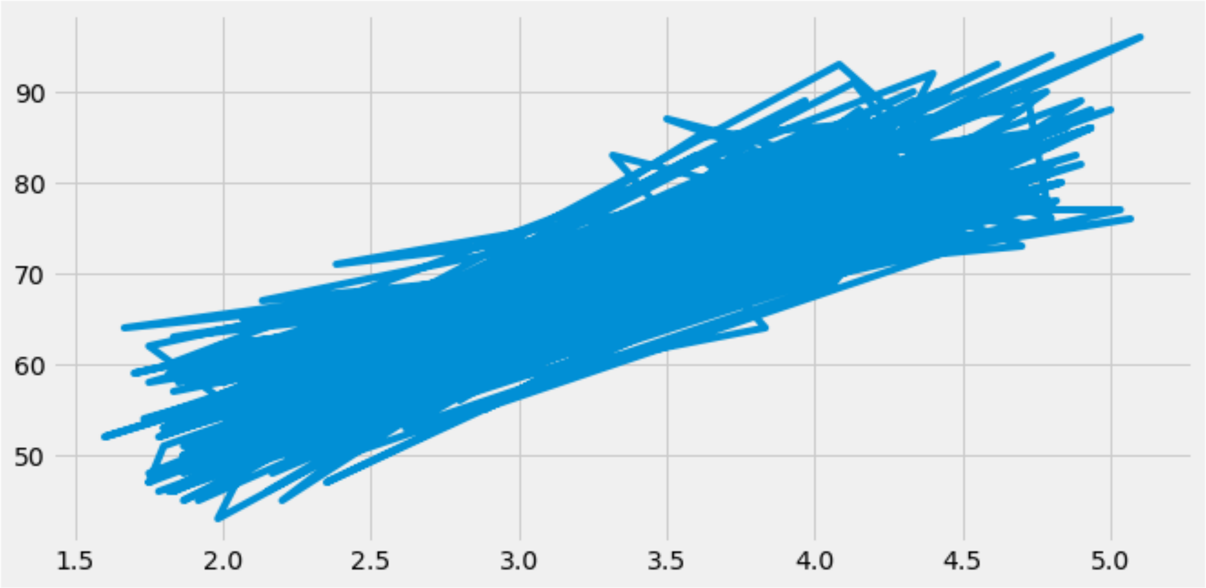


Property	Value	Accuracy
Mean of x	9	exact
Sample variance of x : s_x^2	11	exact
Mean of y	7.50	to 2 decimal places
Sample variance of y : s_y^2	4.125	±0.003
Correlation between x and y	0.816	to 3 decimal places
Linear regression line	$y = 3.00 + 0.500x$	to 2 and 3 decimal places, respectively
Coefficient of determination of the linear regression : R^2	0.67	to 2 decimal places

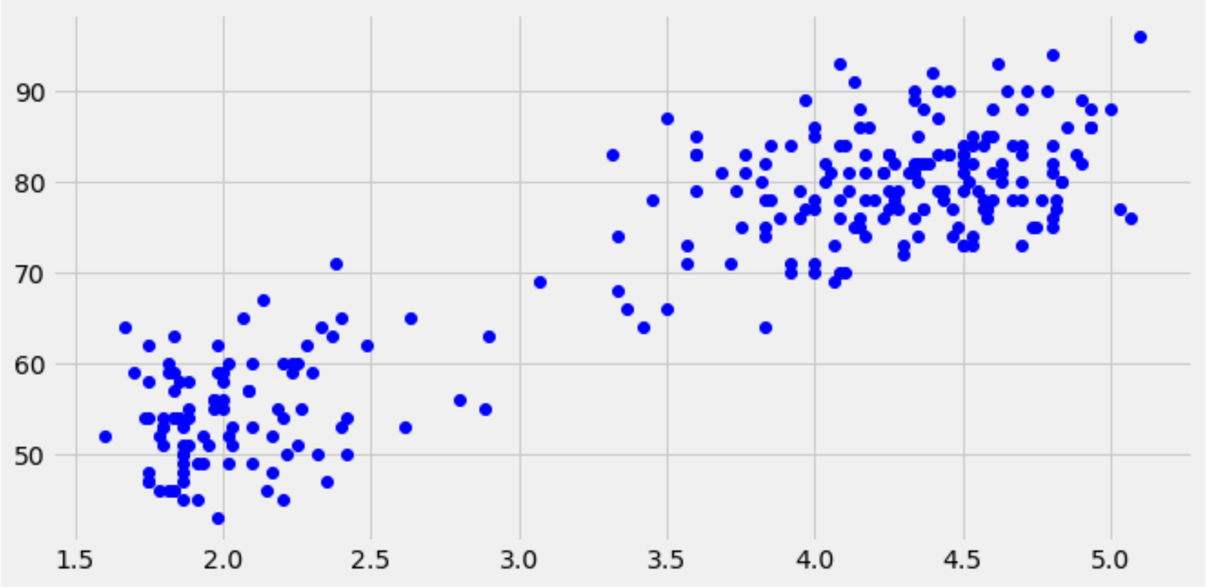


R²=0.06
REXTHOR, THE DOG-BEARER
I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER TO GUESS THE DIRECTION OF THE CORRELATION FROM THE SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

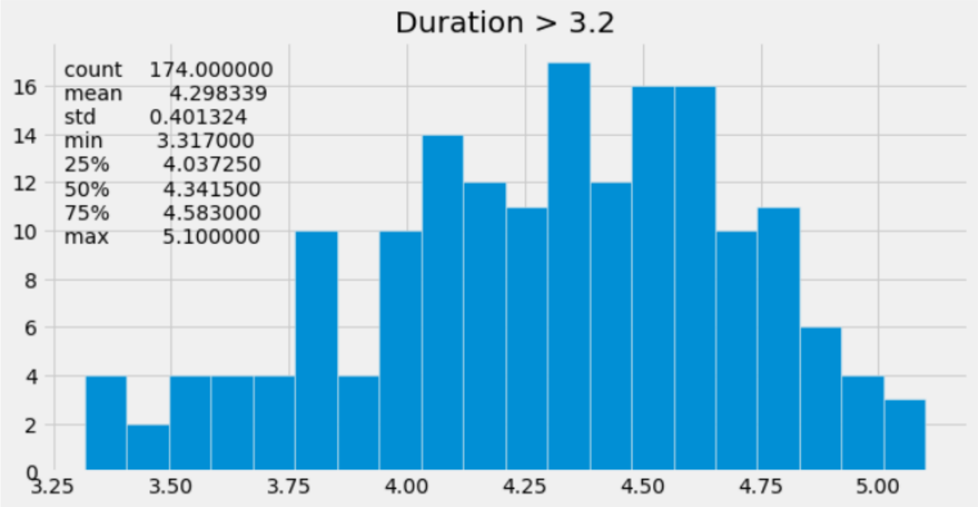
In [13]:
```
# that's pretty strong, let's look at it
plt.plot(data['Duration'], data['Interval']);
```



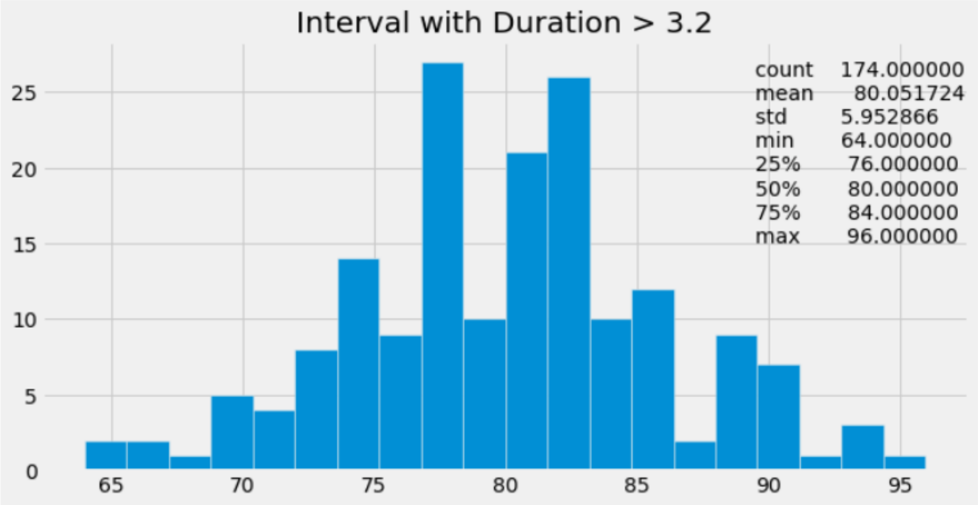
In [14]:
```
# Maybe plotting as points would be better...
plt.plot(data['Duration'], data['Interval'],"ob");   # o: dots  b: blue
```



In [15]:
```
# There seems to be two populations there!

# If we select just one:
long_duration_data = data.where(data['Duration'] > 3.2)
plt.hist(long_duration_data['Duration'], bins=20)
plt.figtext(0.1,0.5, long_duration_data.to_df()['Duration'].describe().to_string())
plt.title("Duration > 3.2");
```

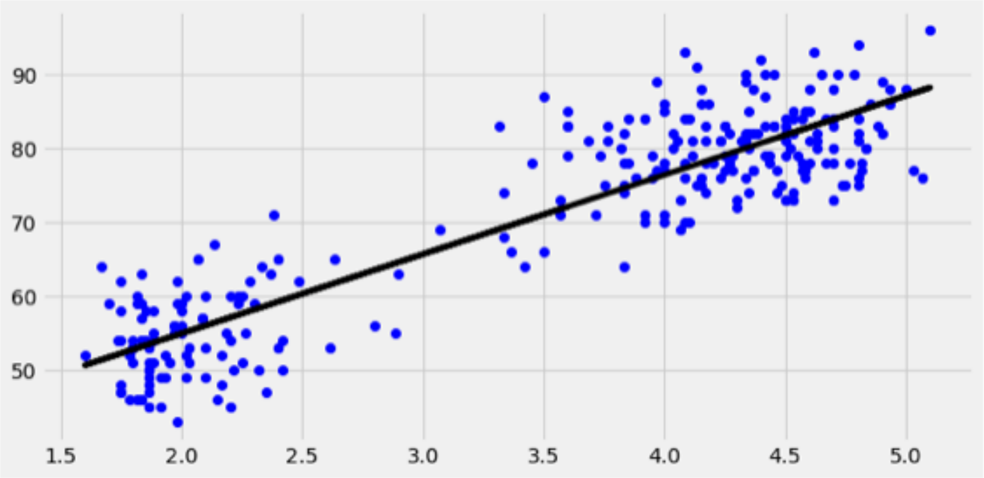


In [16]:
```
# But of course duration is more compact because we selected a narrower range,  How about interval?
plt.hist(long_duration_data['Interval'], bins=20)
plt.figtext(0.75,0.5, long_duration_data.to_df()['Interval'].describe().to_string())
plt.title("Interval with Duration > 3.2");
```

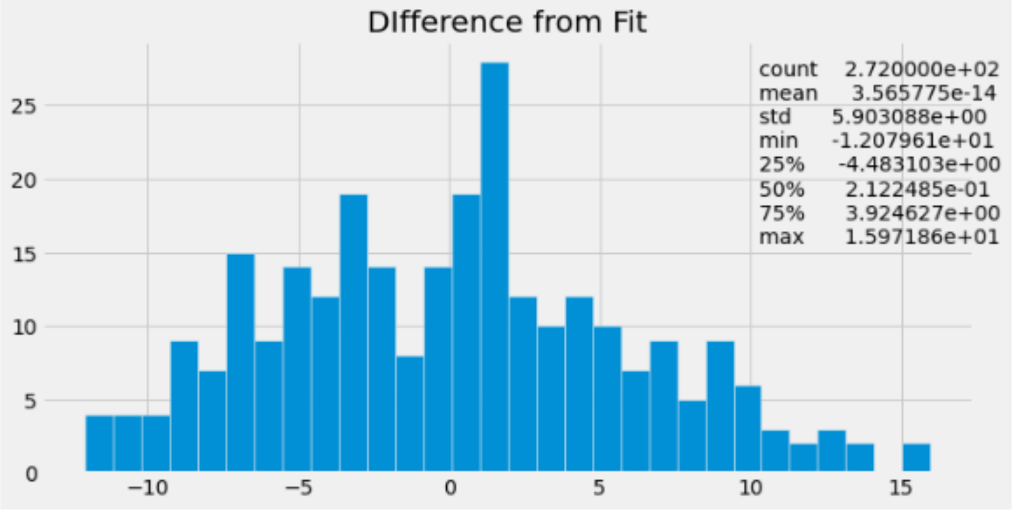


In [17]:
```
# We're down to 50% in 8 minutes and an RMS of 6 minutes on a mean of 80; 10%!
#
```

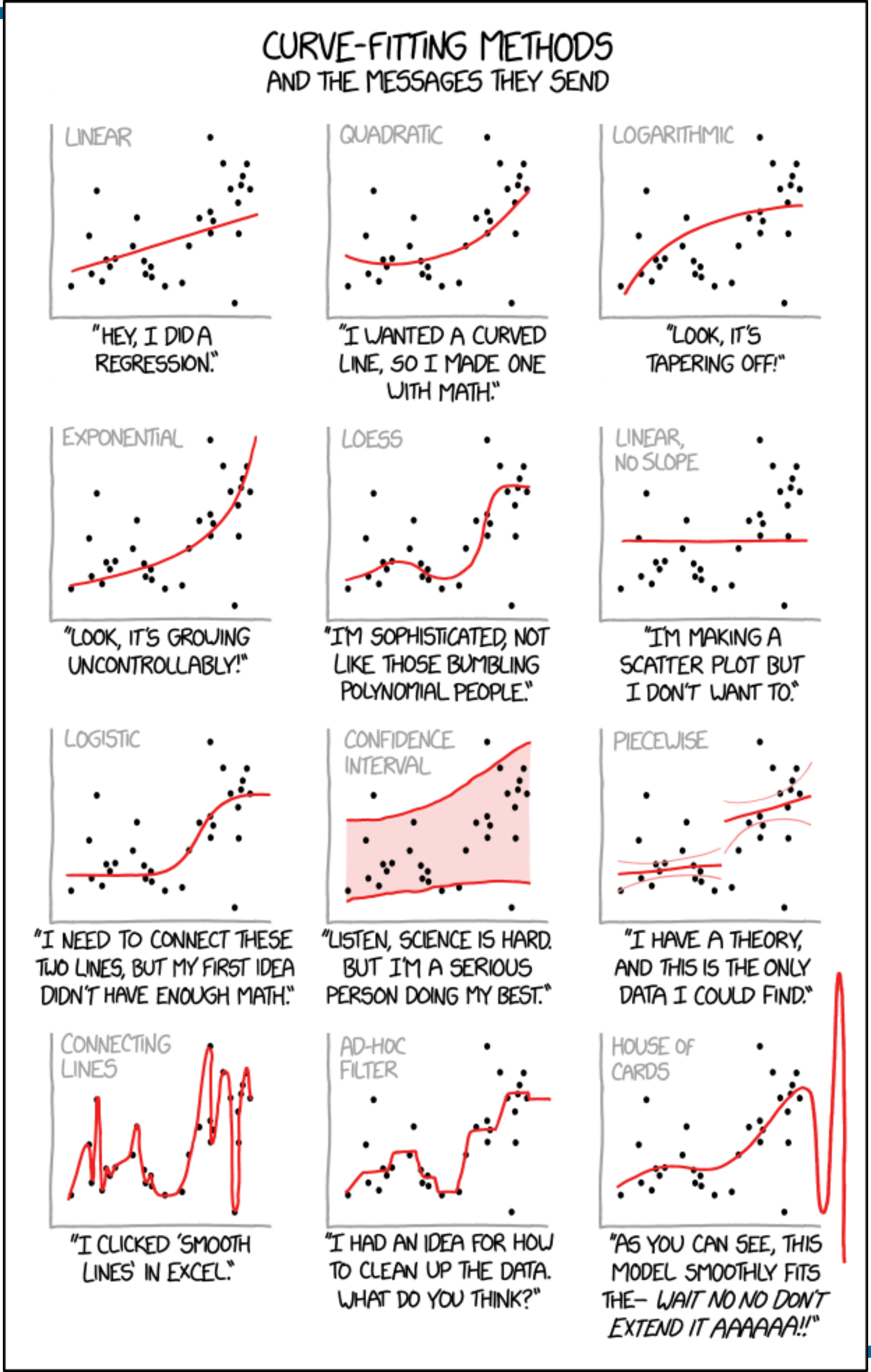
In [18]:
```python
# Try fitting a line instead using two populations
d = np.polyfit(data['Duration'], data['Interval'],1)
f = np.poly1d(d)
data['trendline'] = f(data['Duration'])

plt.plot(data['Duration'], data['Interval'],"ob");
plt.plot(data['Duration'], data['trendline'],"k");
```



In [19]:
```python
# See how wide the difference from the linear fit is
plt.hist(data['Interval']-data['trendline'], 30)
plt.figtext(0.75,0.5, (data.to_df()['Interval']-data.to_df()['trendline']).describe().to_string())
plt.title("DIfference from Fit");
```



In [20]:
```python
# Performance is about the same.  Is there a reason to prefer one method over another here?
```


CURVE-FITTING METHODS
AND THE MESSAGES THEY SEND
LINEAR
"HEY, I DID A REGRESSION."
QUADRATIC
"I WANTED A CURVED LINE, SO I MADE ONE WITH MATH."
LOGARITHMIC
"LOOK, IT'S TAPERING OFF!"
EXPONENTIAL
"LOOK, IT'S GROWING UNCONTROLLABLY!"
LOESS
"I'M SOPHISTICATED, NOT LIKE THOSE BUMBLING POLYNOMIAL PEOPLE."
LINEAR, NO SLOPE
"I'M MAKING A SCATTER PLOT BUT I DON'T WANT TO."
LOGISTIC
"I NEED TO CONNECT THESE TWO LINES, BUT MY FIRST IDEA DIDN'T HAVE ENOUGH MATH."
CONFIDENCE INTERVAL
"LISTEN, SCIENCE IS HARD. BUT I'M A SERIOUS PERSON DOING MY BEST."
PIECEWISE
"I HAVE A THEORY, AND THIS IS THE ONLY DATA I COULD FIND."
CONNECTING LINES
"I CLICKED 'SMOOTH LINES' IN EXCEL."
AD-HOC FILTER
"I HAD AN IDEA FOR HOW TO CLEAN UP THE DATA. WHAT DO YOU THINK?"
HOUSE OF CARDS
"AS YOU CAN SEE, THIS MODEL SMOOTHLY FITS THE- WAIT NO NO DON'T EXTEND IT AAAAAA!!"

Understanding what we're seeing - Toast

CERN School of Computing

Why does dropped toast always land buttered-side down?

Experimental question!

First establish: Does dropped toast always land butter side down?

Or even more often than 50/50?

How do you assess the experimental result?

See how likely the result is without an effect, i.e with 50/50

This is a "null hypothesis", which gives a probability for result: the p value


I CAN'T BELIEVE SCHOOLS ARE STILL TEACHING KIDS ABOUT THE NULL HYPOTHESIS.
I REMEMBER READING A BIG STUDY THAT CONCLUSIVELY DISPROVED IT YEARS AGO.

CERN School of Computing

Approach it analytically

$X \sim B(n, p)$. The probability of getting exactly k successes in n independent Bernoulli trials is given by the probability mass function:

$$f(k,n,p) = \Pr(k;n,p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 0, 1, 2, \ldots, n$, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

The cumulative distribution function can be expressed as:

$$F(k;n,p) = \Pr(X \le k) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i},$$

where $\lfloor k \rfloor$ is the "floor" under k, i.e. the greatest integer less than or equal to k.

It can also be represented in terms of the regularized incomplete beta function, as follows:[3]

$$\begin{aligned} F(k;n,p) &= \Pr(X \le k) \\ &= I_{1-p}(n-k, k+1) \\ &= (n-k)\binom{n}{k} \int_0^{1-p} t^{n-k-1} (1-t)^k \, dt. \end{aligned}$$

which is equivalent to the cumulative distribution function of the F-distribution:[4]

$$F(k;n,p) = F_{F\text{-distribution}}\left(x = \frac{1-p}{p}\frac{k+1}{n-k}; d_1 = 2(n-k), d_2 = 2(k+1)\right).$$

The Toast Myth

The Mythbusters TV show did an experiment with 48 pieces of toast, where 29 landed butter side up and 19 butter side down. Let's see if we can figure out how likely this outcome would be, if toast was equally likely to land on either side. In particular, we'll play a "what-if" game: what if toast was equally likely to land on both sides? Let's simulate what would happen, under that assumption.

n [2]:

```
# First, list two possible results
sides = make_array('Butter Side Up', 'Butter Side Down')
```

n [3]:

```
# Make that into a table
possible_outcomes = Table().with_column('Outcome', sides)
```

n [4]:

```
possible_outcomes
```

ut[4]:

Outcome
Butter Side Up
Butter Side Down

n [5]:

```
# Ask for 48 cases where the output is sampled (chosen) from those two possibilities
simulated_experiment = possible_outcomes.sample(48)
```

n [6]:

```
simulated_experiment
```

ut[6]:

Outcome
Butter Side Down
Butter Side Down
Butter Side Up
Butter Side Down
Butter Side Up
Butter Side Up
Butter Side Up
Butter Side Up
Butter Side Up

In [7]:
```
# Group them, which also counts them.
simulated_experiment.group('Outcome')
```

Out[7]:

Outcome	count
Butter Side Down	25
Butter Side Up	23

In [8]:
```
# To make this a bit more automatic, define a function that provides the butter-side-up count
def count_up(sample):
    counts = sample.group('Outcome').where('Outcome', 'Butter Side Up')
    number_up = counts.column('count').item(0)
    return number_up
```

In [9]:
```
# Always test things!
count_up(simulated_experiment)
```

Out[9]: 23

Simulation

Above we saw how to simulate an episode of the TV show (i.e., one experiment), under the "what-if" assumption that toast is equally likely to land on both sides. Now we're going to repeat the simulation 10000 times, and keep track of the statistic (the number of times the toast landed butter-side-up) we get from each simulated TV episode.

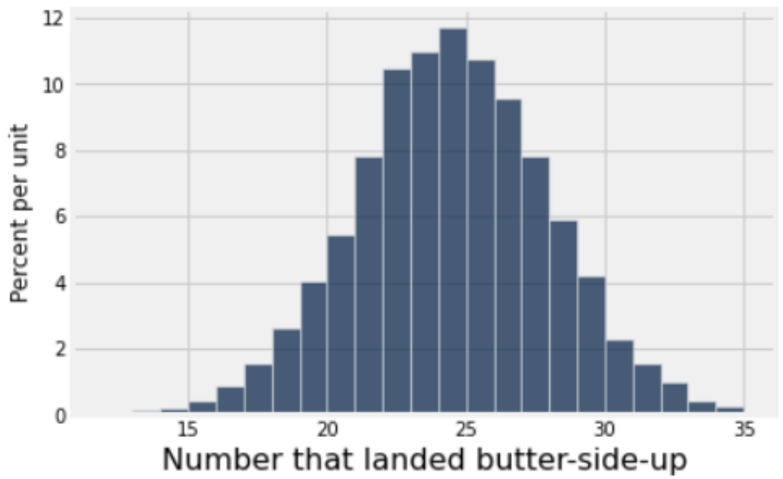
In [10]:
```
counts = make_array()
for i in np.arange(10000): # 10000 repetitions
    one_simulated_episode = possible_outcomes.sample(48)
    number_up = count_up(one_simulated_episode)
    counts = np.append(counts, number_up)
results = Table().with_column('Number that landed butter-side-up', counts)
```

In [11]:
```
results
```

Out[11]:

Number that landed butter-side-up
21
29
24
26
24
25

In [12]:
```
results.hist(bins=np.arange(12,36,1))  # an alternate form of plotting
# note that this method of plotting gives plots/unit and allows close control over binning
```



In [13]:
```
# With this data, what's the chance of the value they saw or higher?
# This is known as the p-value
results.where(results['Number that landed butter-side-up'] >= 29).num_rows / 10000
```

Out[13]:
```
0.0966
```

In [14]:
```
# Quick, without looking at the number from here,
# what do you expect the mean and std dev of that distribution to be?
results[0].mean(), results[0].std()
```

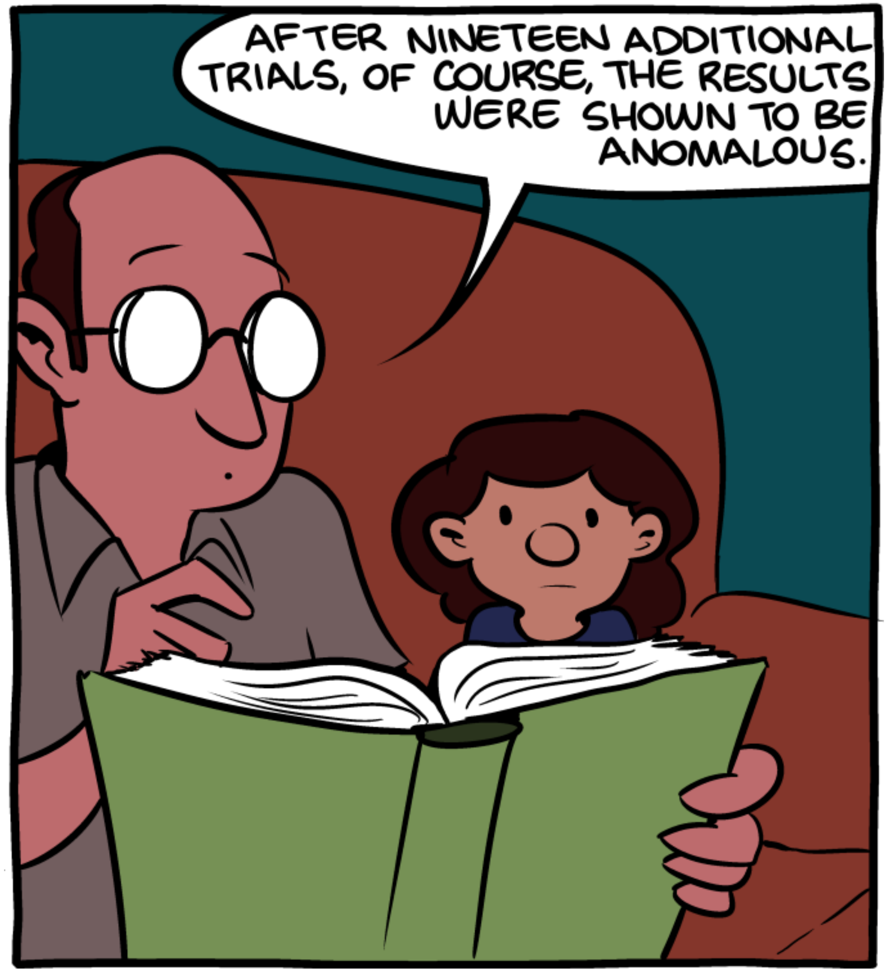
Out[14]:
```
(23.982099999999999, 3.4885784483081359)
```

In [15]:
```
# Many expect it to be sqt(24), because of Gaussian or Poisson distributions.
# But this is actually binomial distribution, where the std dev is smaller because you pick one of two
math.sqrt(24), math.sqrt(24)/math.sqrt(2)
```

Out[15]:
```
(4.898979485566356, 3.464101615137754)
```

In [16]:
```
# try simulating the British school study:
# 9821 waist-high drops with 6101 butter down landings
# With just a B written on the toast: 9748 drops with 5663 B-down
# from 2.5m: 2038 with 953 B-side down (sign reversed!)

# is there something going on?
```

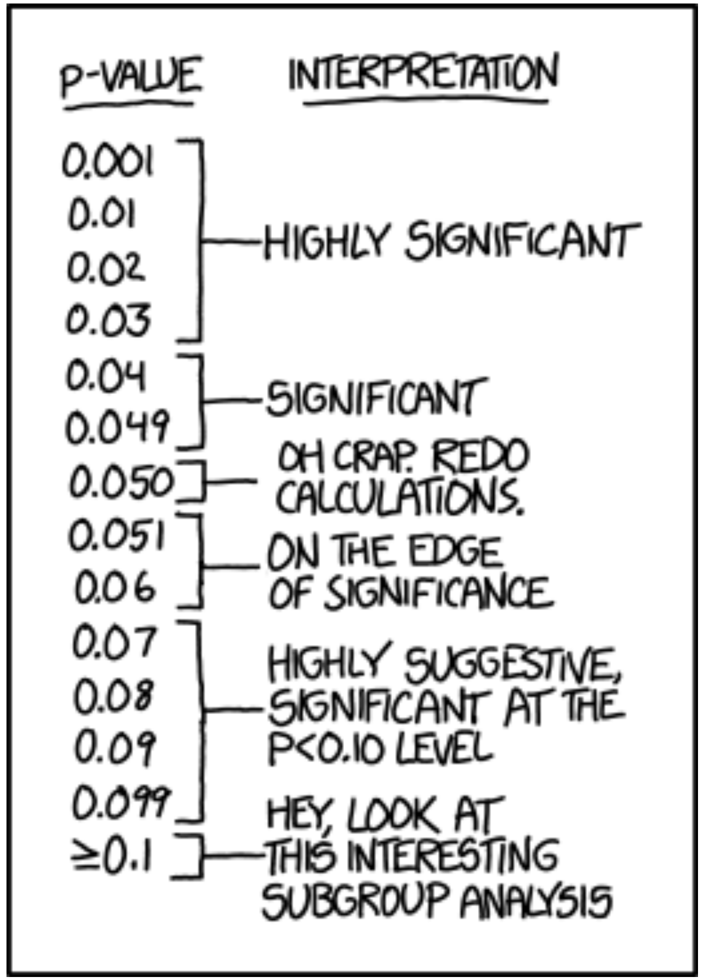
Sometimes you need to run the experiment for longer & get more data…

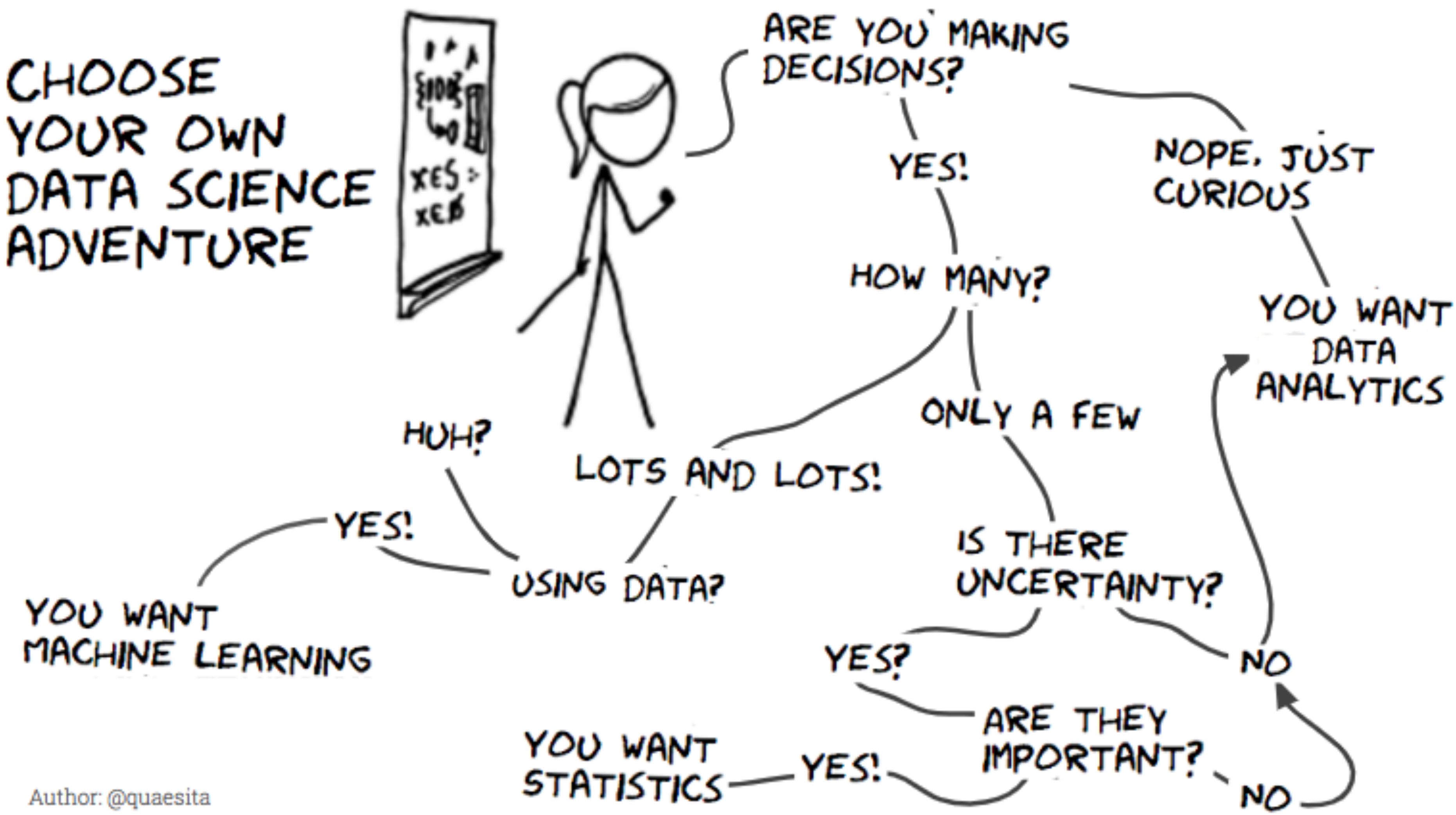


"The Tortoise And The Hare" is actually a fable about small sample sizes.

Toast with higher statistics:

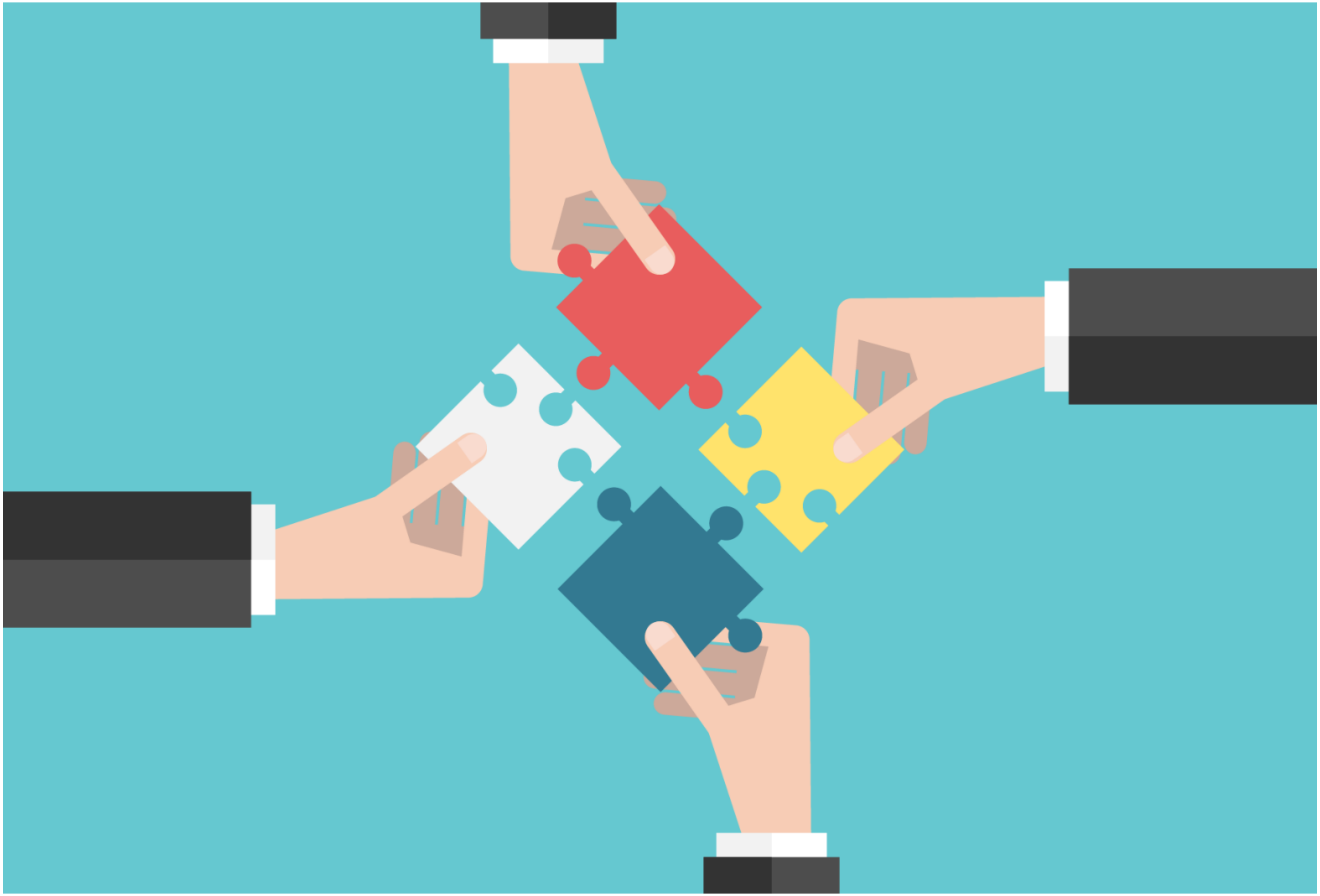
https://web.archive.org/web/20101120232606/http://www.counton.org/thesum/issue-07/issue-07-page-05.htm


P-VALUE
INTERPRETATION
0.001
0.01
0.02
0.03
HIGHLY SIGNIFICANT
0.04
0.049
SIGNIFICANT
0.050
OH CRAP. REDO CALCULATIONS.
0.051
0.06
ON THE EDGE OF SIGNIFICANCE
0.07
0.08
0.09
0.099
HIGHLY SUGGESTIVE, SIGNIFICANT AT THE P<0.10 LEVEL
≥0.1
HEY, LOOK AT THIS INTERESTING SUBGROUP ANALYSIS



Author: @quaesita

Data Comes From Many Sources



Merging data - Drinks

In [2]:
```
# create a table of drinks available at several places with there prices
drinks = Table(['Drink', 'Cafe', 'Price']).with_rows([  # a table of menus for cafes
    ['Milk Tea', 'Tea One', 4],
    ['Espresso', 'Nefeli',  2],
    ['Latte',    'Nefeli',  3],
    ['Espresso', "Abe's",   2]
])
drinks
```

Out[2]:

Drink	Cafe	Price
Milk Tea	Tea One	4
Espresso	Nefeli	2
Latte	Nefeli	3
Espresso	Abe's	2

In [3]:
```
# create a table of available discounts
discounts = Table().with_columns(                      # A table of discounts by cafe
    'Coupon % off', make_array(25, 50, 5),
    'Location', make_array('Tea One', 'Nefeli', 'Tea One')
)
discounts
```

Out[3]:

Coupon % off	Location
25	Tea One
50	Nefeli
5	Tea One

In [4]:
```
# combine the tables by matching cafe names
t = drinks.join('Cafe', discounts, 'Location')
t             # note you don't have a discount for Abe's
```

Out[4]:

Cafe	Drink	Price	Coupon % off
Nefeli	Espresso	2	50
Nefeli	Latte	3	50
Tea One	Milk Tea	4	25
Tea One	Milk Tea	4	5

In [5]:
```
# Compute a column of discounted price
t.with_column('Discounted', t.column(2) * (1 - t.column(3)/ 100))
```

Out[5]:

Cafe	Drink	Price	Coupon % off	Discounted
Nefeli	Espresso	2	50	1
Nefeli	Latte	3	50	1.5
Tea One	Milk Tea	4	25	3
Tea One	Milk Tea	4	5	3.8

In [6]:
```python
# What do all possible two-drink orders cost?
# Join with itself, matching on Cafe (you only order in one place)
two = drinks.join('Cafe', drinks)
two
```

Out[6]:

Cafe	Drink	Price	Drink_2	Price_2
Abe's	Espresso	2	Espresso	2
Nefeli	Espresso	2	Espresso	2
Nefeli	Espresso	2	Latte	3
Nefeli	Latte	3	Espresso	2
Nefeli	Latte	3	Latte	3
Tea One	Milk Tea	4	Milk Tea	4

In [7]:
```python
# Add a total price
two.with_column('Total', two.column('Price') + two.column('Price_2'))
```

Out[7]:

Cafe	Drink	Price	Drink_2	Price_2	Total
Abe's	Espresso	2	Espresso	2	4
Nefeli	Espresso	2	Espresso	2	4
Nefeli	Espresso	2	Latte	3	5
Nefeli	Latte	3	Espresso	2	5
Nefeli	Latte	3	Latte	3	6
Tea One	Milk Tea	4	Milk Tea	4	8

Another way to understand data - GIS - Bikes

In [1]:
```python
# usual imports
from datascience import *
import numpy as np
import pandas as pd

%matplotlib inline
import matplotlib.pyplot as plots
#plots.style.use('fivethirtyeight')

# Configure for presentation
#np.set_printoptions(threshold=50, linewidth=50)
import matplotlib as mpl
#mpl.rc('font', size=16)
```

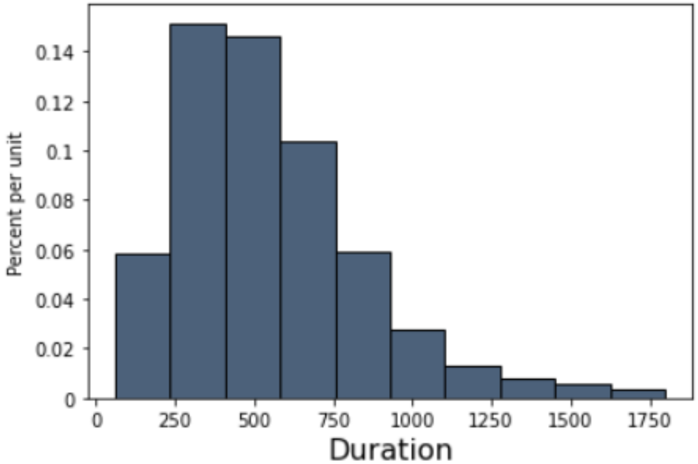
Bikes

In [2]:
```python
 # Read a dataset from a bike-rental firm containing 354k rentals
trips = Table.read_table('trip.csv')
# see what columns are available in this data set:
trips
```

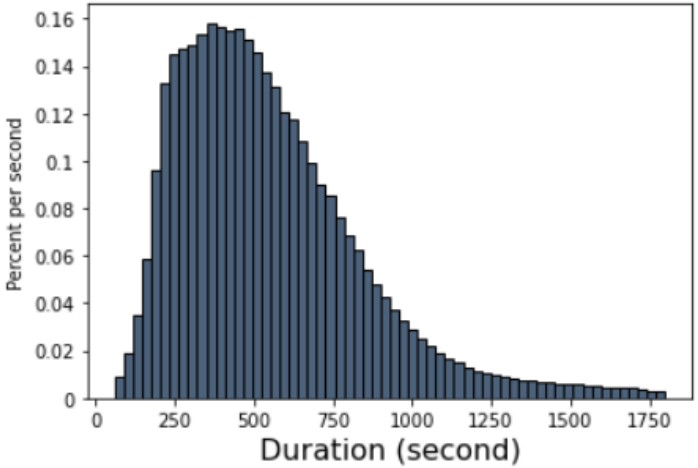
Out[2]:

Trip ID	Duration	Start Date	Start Station	Start Terminal	End Date	End Station	End Terminal	Bike #	Subscriber Type	Zip Code
913460	765	8/31/2015 23:26	Harry Bridges Plaza (Ferry Building)	50	8/31/2015 23:39	San Francisco Caltrain (Townsend at 4th)	70	288	Subscriber	2139
913459	1036	8/31/2015 23:11	San Antonio Shopping Center	31	8/31/2015 23:28	Mountain View City Hall	27	35	Subscriber	95032
913455	307	8/31/2015 23:13	Post at Kearny	47	8/31/2015 23:18	2nd at South Park	64	468	Subscriber	94107
913454	409	8/31/2015 23:10	San Jose City Hall	10	8/31/2015 23:17	San Salvador at 1st	8	68	Subscriber	95113
913453	789	8/31/2015 23:09	Embarcadero at Folsom	51	8/31/2015 23:22	Embarcadero at Sansome	60	487	Customer	9069
913452	293	8/31/2015 23:07	Yerba Buena Center of the Arts (3rd @ Howard)	68	8/31/2015 23:12	San Francisco Caltrain (Townsend at 4th)	70	538	Subscriber	94118
913451	896	8/31/2015 23:07	Embarcadero at Folsom	51	8/31/2015 23:22	Embarcadero at Sansome	60	363	Customer	92562
913450	255	8/31/2015	Embarcadero at Sansome	60	8/31/2015	Steuart at Market	74	470	Subscriber	94111

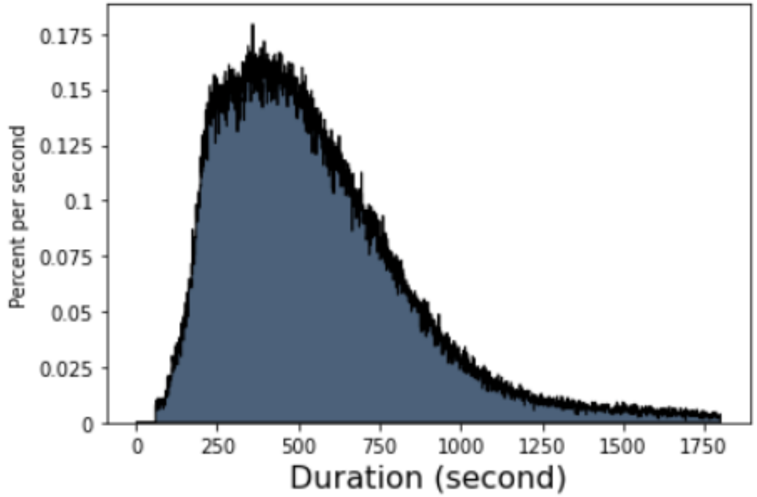
In [3]:
```
# identify a subsample of "commuters"
commute = trips.where('Duration', are.below(1800))   # Why is this here?  Are there significant ones above that?
commute.hist('Duration')
```



In [4]:
```
commute.hist('Duration', bins=60, unit='second')  # clean the plot up a bit
```



In [5]:
```
commute.hist('Duration', bins=np.arange(1801), unit='second')  # there are 354K rows
```



In [6]:
```
# group by starting location to get counts, then sort to get largest values
starts = commute.group('Start Station').sort('count', descending=True)
starts
```

Out[6]:

Start Station	count
San Francisco Caltrain (Townsend at 4th)	25858
San Francisco Caltrain 2 (330 Townsend)	21523
Harry Bridges Plaza (Ferry Building)	15543
Temporary Transbay Terminal (Howard at Beale)	14298
2nd at Townsend	13674
Townsend at 7th	13579
Steuart at Market	13215
Embarcadero at Sansome	12842
Market at 10th	11523
Market at Sansome	11023

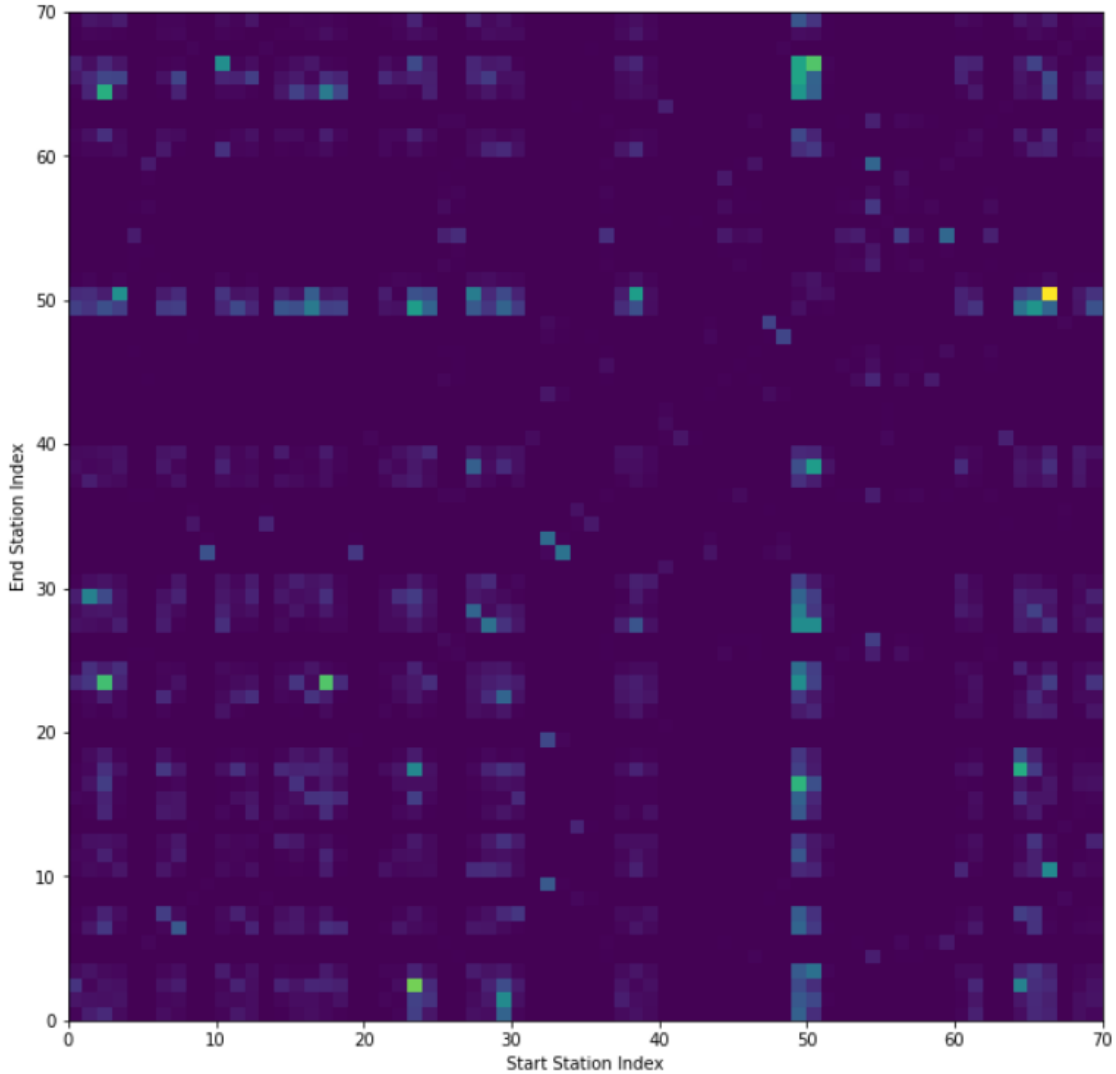
... (60 rows omitted)

In [7]:
```python
# Compute a table counting start -> end trips
pivot = commute.pivot('Start Station', 'End Station')
pivot
```

Out[7]:

End Station	2nd at Folsom	2nd at South Park	2nd at Townsend	5th at Howard	Adobe on Almaden	Arena Green / SAP Center	Beale at Market	Broadway St at Battery St	California Ave Caltrain Station	Castro Street and El Camino Real	Civic Center BART (7th at Market)	Clay at Battery	Commercial at Montgomery	Ur
2nd at Folsom	54	190	554	107	0	0	40	21	0	0	44	78	54	
2nd at South Park	295	164	71	180	0	0	208	85	0	0	112	87	160	
2nd at Townsend	437	151	185	92	0	0	608	350	0	0	80	329	168	
5th at Howard	113	177	148	83	0	0	59	130	0	0	203	76	129	
Adobe on Almaden	0	0	0	0	11	4	0	0	0	0	0	0	0	
Arena Green / SAP Center	0	0	0	0	7	64	0	0	0	0	0	0	0	
Beale at Market	127	79	183	59	0	0	59	661	0	0	201	75	101	
Broadway St at Battery St	67	89	279	119	0	0	1022	110	0	0	62	283	226	
California Ave Caltrain Station	0	0	0	0	0	0	0	0	38	1	0	0	0	
Castro Street and El Camino Real	0	0	0	0	0	0	0	0	0	30	0	0	0	

In [8]:
```
# It's easier to interpret this graphically - the heat plot
plots.rcParams['figure.figsize'] = (11., 11.)  # make a square plot
plots.figure()
ct = pd.crosstab(commute['Start Station'], commute['End Station']) # pandas computation of pivot table
plots.grid(False)
plots.pcolor(ct)    # plot that dataframe as color spectrum
plots.xlabel('Start Station Index')
plots.ylabel('End Station Index')
plots.plot();
```



```
In [10]: duration = trips.select('Start Station', 'End Station', 'Duration')  # narrow down the table to three columns
         duration
```

Out[10]:

Start Station	End Station	Duration
Harry Bridges Plaza (Ferry Building)	San Francisco Caltrain (Townsend at 4th)	765
San Antonio Shopping Center	Mountain View City Hall	1036
Post at Kearny	2nd at South Park	307
San Jose City Hall	San Salvador at 1st	409
Embarcadero at Folsom	Embarcadero at Sansome	789
Yerba Buena Center of the Arts (3rd @ Howard)	San Francisco Caltrain (Townsend at 4th)	293
Embarcadero at Folsom	Embarcadero at Sansome	896
Embarcadero at Sansome	Steuart at Market	255
Beale at Market	Temporary Transbay Terminal (Howard at Beale)	126
Post at Kearny	South Van Ness at Market	932

... (354142 rows omitted)

```
In [11]: # Group the trips from each to each, then select the shortest duration trip in each bin
         shortest = duration.group(['Start Station', 'End Station'], min)
         shortest
```

Out[11]:

Start Station	End Station	Duration min
2nd at Folsom	2nd at Folsom	61
2nd at Folsom	2nd at South Park	61
2nd at Folsom	2nd at Townsend	137
2nd at Folsom	5th at Howard	215
2nd at Folsom	Beale at Market	219
2nd at Folsom	Broadway St at Battery St	351

CERN School of Computing

Maps

In [13]:
```
# Get the locations of the stations
stations = Table.read_table('station.csv')    # Table of station locations
stations                                      # landmark is the town containg the station
```

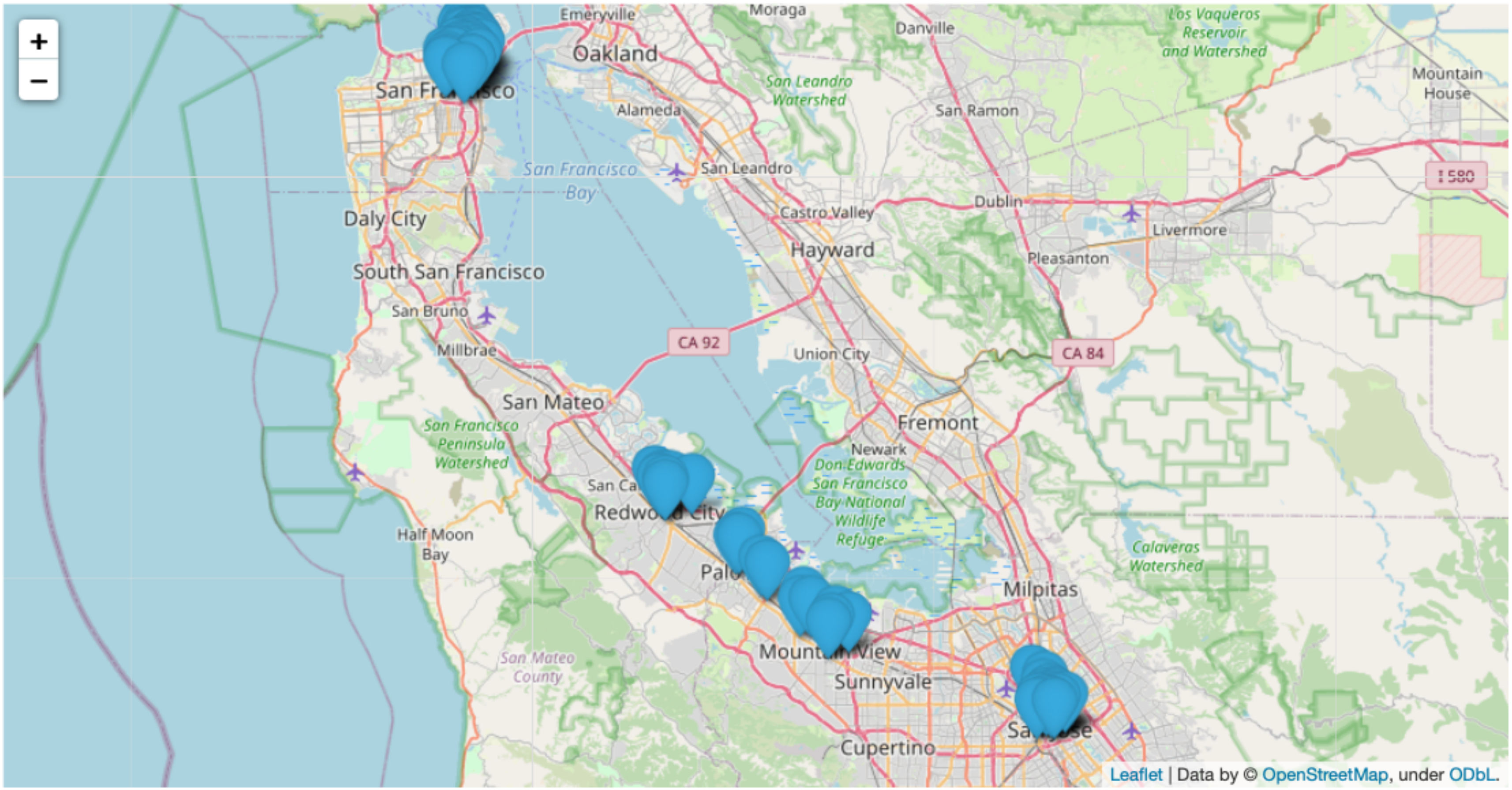
Out[13]:

station_id	name	lat	long	dockcount	landmark	installation
2	San Jose Diridon Caltrain Station	37.3297	-121.902	27	San Jose	8/6/2013
3	San Jose Civic Center	37.3307	-121.889	15	San Jose	8/5/2013
4	Santa Clara at Almaden	37.334	-121.895	11	San Jose	8/6/2013
5	Adobe on Almaden	37.3314	-121.893	19	San Jose	8/5/2013
6	San Pedro Square	37.3367	-121.894	15	San Jose	8/7/2013
7	Paseo de San Antonio	37.3338	-121.887	15	San Jose	8/7/2013
8	San Salvador at 1st	37.3302	-121.886	15	San Jose	8/5/2013
9	Japantown	37.3487	-121.895	15	San Jose	8/5/2013
10	San Jose City Hall	37.3374	-121.887	15	San Jose	8/6/2013
11	MLK Library	37.3359	-121.886	19	San Jose	8/6/2013

... (60 rows omitted)

In [14]:
```
# Map all the locations
Marker.map_table(stations.select('lat', 'long', 'name'))
```

Out[14]:

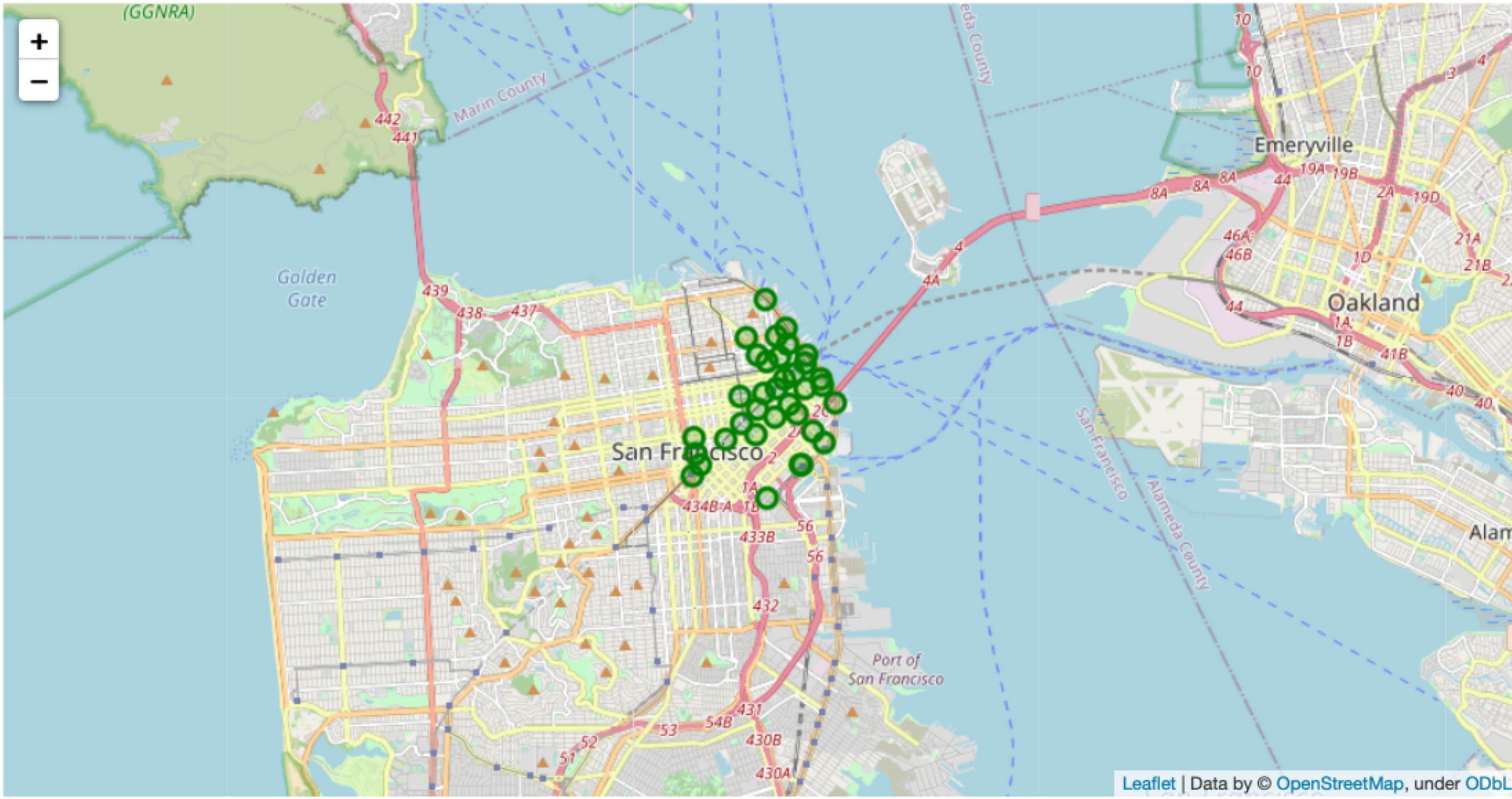


CERN School of Computing

In [15]:

```
# Show the San Francisco locations
sf = stations.where('landmark', 'San Francisco')
Circle.map_table(sf.select('lat', 'long', 'name'), color='green', area=100)
```

Out[15]:



CERN School of Computing

In [18]:
```
# Calculate the number of trips starting at each station by joining the two data sets
station_starts = stations.join('name', starts, 'Start Station')
station_starts
```

Out[18]:

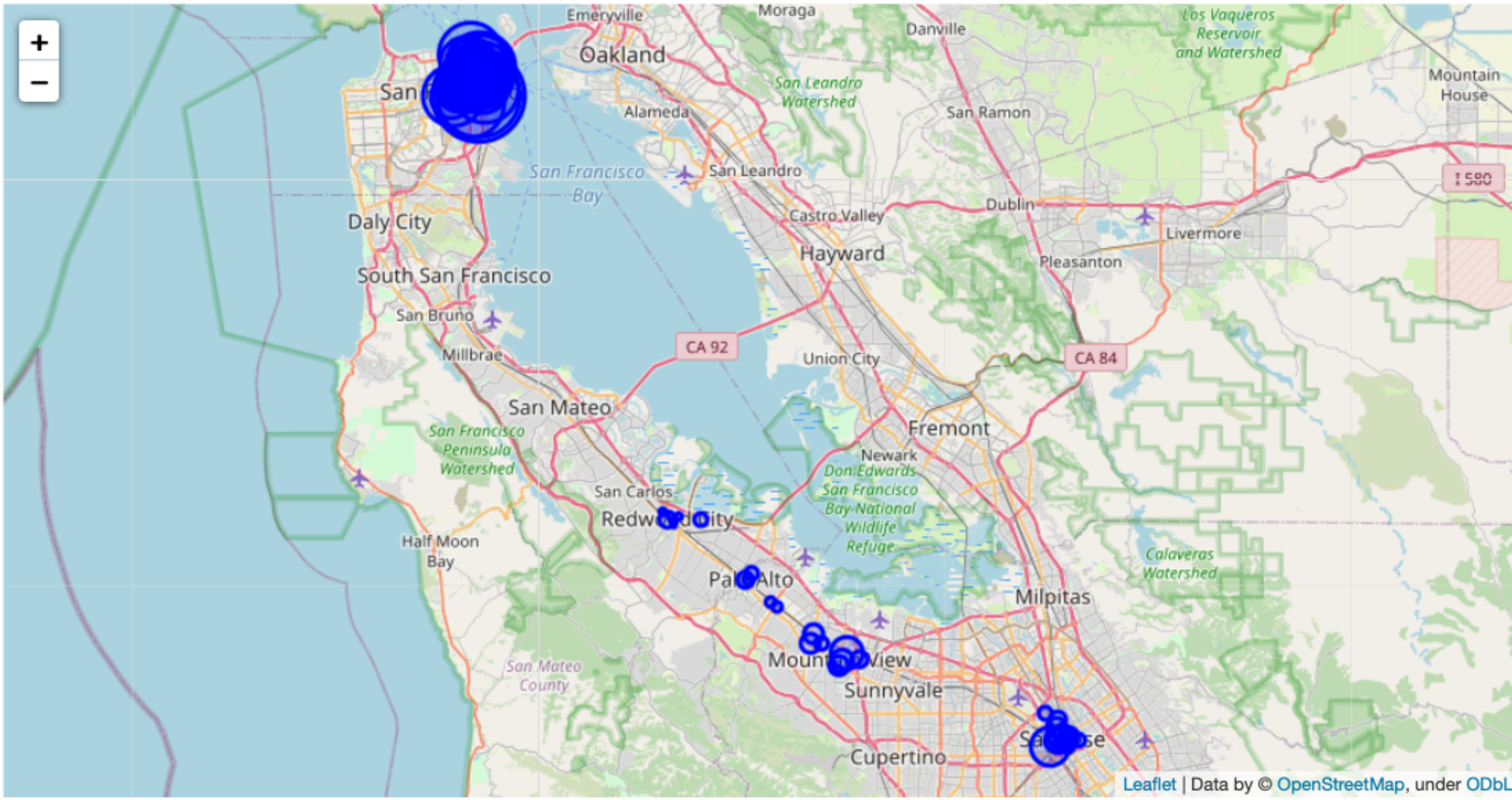
name	station_id	lat	long	dockcount	landmark	installation	count
2nd at Folsom	62	37.7853	-122.396	19	San Francisco	8/22/2013	7841
2nd at South Park	64	37.7823	-122.393	15	San Francisco	8/22/2013	9274
2nd at Townsend	61	37.7805	-122.39	27	San Francisco	8/22/2013	13674
5th at Howard	57	37.7818	-122.405	15	San Francisco	8/21/2013	7394
Adobe on Almaden	5	37.3314	-121.893	19	San Jose	8/5/2013	522
Arena Green / SAP Center	14	37.3327	-121.9	19	San Jose	8/5/2013	590
Beale at Market	56	37.7923	-122.397	19	San Francisco	8/20/2013	8135
Broadway St at Battery St	82	37.7985	-122.401	15	San Francisco	1/22/2014	7460
California Ave Caltrain Station	36	37.4291	-122.143	15	Palo Alto	8/14/2013	300
Castro Street and El Camino Real	32	37.386	-122.084	11	Mountain View	12/31/2013	1137

... (58 rows omitted)

CERN School of Computing

In [19]:
```
# Show how many trips start from each location?
Circle.map_table(station_starts.select('lat', 'long', 'name').with_columns(  # adding presentation options
    'color', 'blue',                                                      # show blue circles
    'area', station_starts.column('count') * 0.1                          # set circle size from number starts
))
```

Out[19]:



CERN School of Computing

Outside the Box: Text Analysis

In [1]:
```
# Examine the book "Little Women" to see what we can learn from its text

# usual imports
from datascience import *
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plots
plots.style.use('fivethirtyeight')
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

from urllib.request import urlopen
import re
def read_url(url):
    return re.sub('\\s+', ' ', urlopen(url).read().decode())
```

In [2]:
```
# Read the book and split into separate chapters
little_women_url = 'http://data8.org/materials-fa17/lec/little_women.txt'
little_women_text = read_url(little_women_url)
chapters = little_women_text.split('CHAPTER ')[1:]
```

In [3]:
```
# create a table with one chapter's text in each row
Table().with_column('Text', chapters)
```

Out[3]:

Text
ONE PLAYING PILGRIMS "Christmas won't be Christmas witho ...
TWO A MERRY CHRISTMAS Jo was the first to wake in the gr ...
THREE THE LAURENCE BOY "Jo! Jo! Where are you?" cried Me ...
FOUR BURDENS "Oh, dear, how hard it does seem to take up ...
FIVE BEING NEIGHBORLY "What in the world are you going t ...
SIX BETH FINDS THE PALACE BEAUTIFUL The big house did pr ...
SEVEN AMY'S VALLEY OF HUMILIATION "That boy is a perfect ...
EIGHT JO MEETS APOLLYON "Girls, where are you going?" as ...
NINE MEG GOES TO VANITY FAIR "I do think it was the most ...
TEN THE P.C. AND P.O. As spring came on, a new set of am ...

... (37 rows omitted)

In [4]:
```
# Simple check:  Count the number of times "Christmas" appears in each chapter
np.char.count(chapters, 'Christmas')
```

Out[4]:
```
array([8, 9, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 6, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0])
```

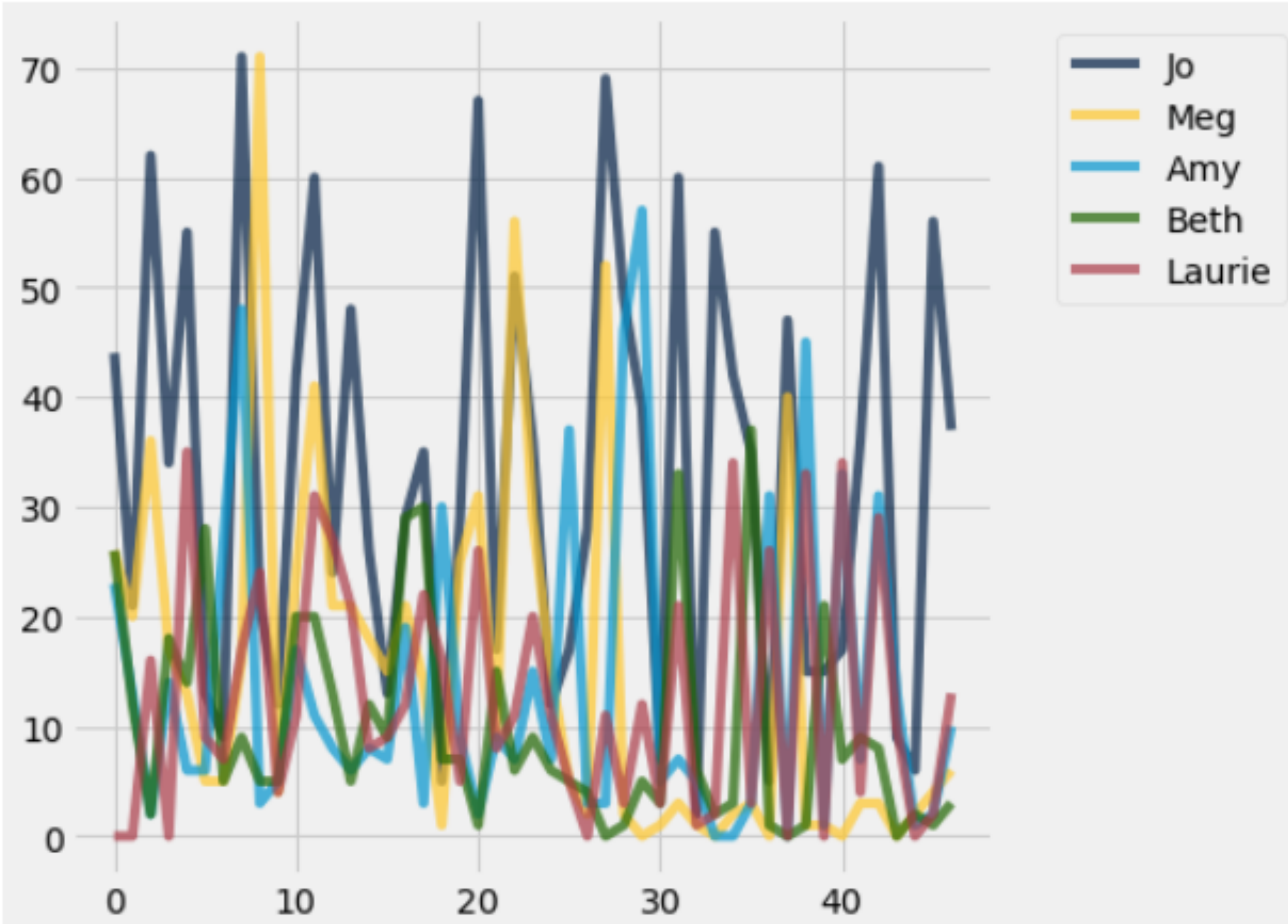
In [5]:
```
# Count the number of times the characters' names appear in each chapter
# and make a table with a column for each character
references = Table().with_columns([
    "Jo",    np.char.count(chapters, "Jo"),
    "Meg",   np.char.count(chapters, "Meg"),
    "Amy",   np.char.count(chapters, "Amy"),
    "Beth",  np.char.count(chapters, "Beth"),
    "Laurie", np.char.count(chapters, "Laurie")
])
references
```

Out[5]:

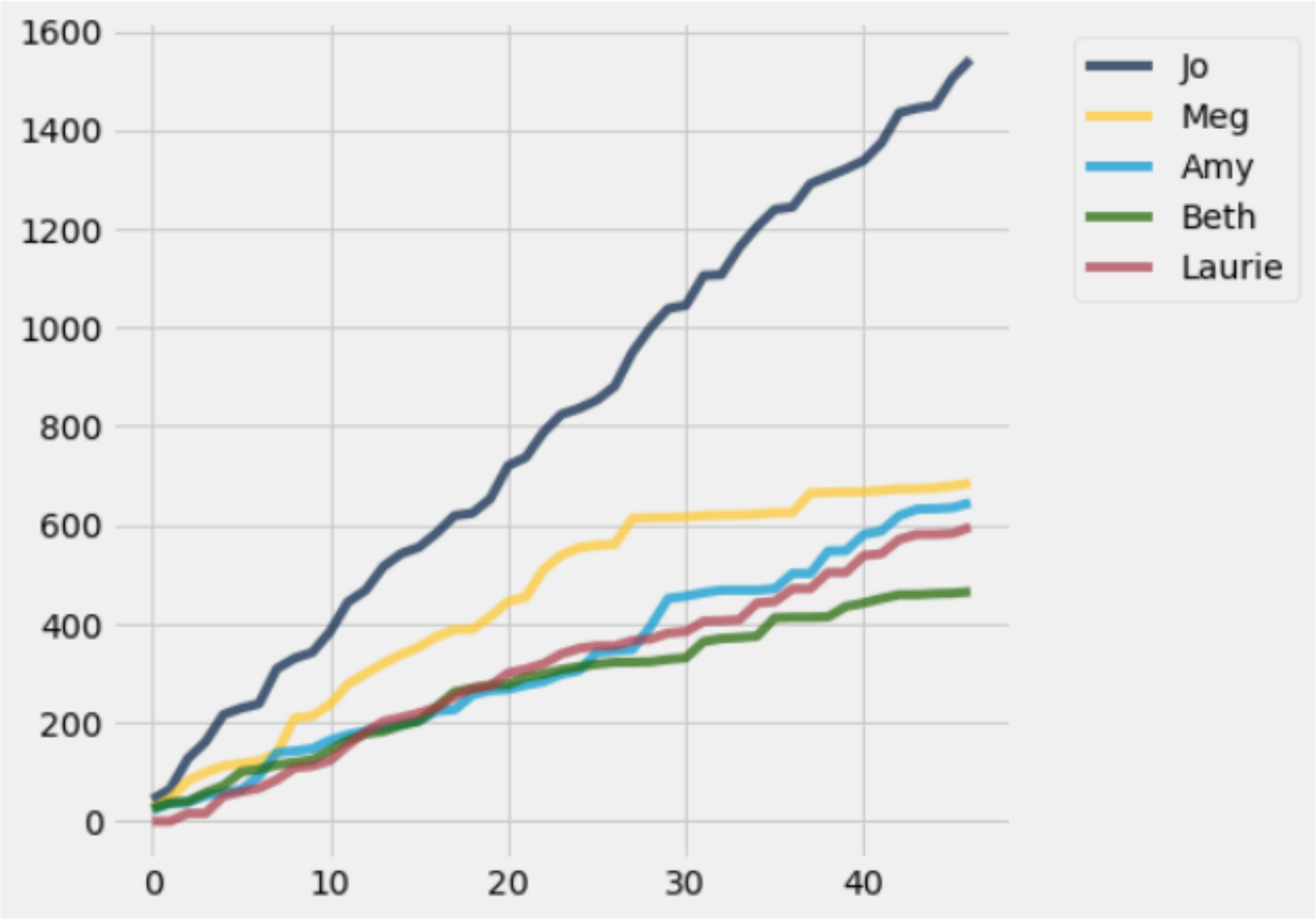
Jo	Meg	Amy	Beth	Laurie
44	26	23	26	0
21	20	13	12	0
62	36	2	2	16
34	17	14	18	0
55	13	6	14	35
13	5	6	28	9
9	5	27	5	7
71	16	48	9	17
21	71	3	5	24
12	4	5	5	4

... (37 rows omitted)

In [6]:
```
# plot appearances by chapter, one curve per character (column)
references.plot()
```



In [7]:
```
# the plot-by-chapter is hard to interpret.  Plot cumulative sums:
references.cumsum().plot()
```



In [8]:
```
# How would you see who's mentioned most often in each chapter?
```

Notebooks as persuasive objects

"When you two have finished arguing your opinions, I actually have data!"

Notebooks as persuasive objects

You've seen some of this already:

Plots and tables to show data

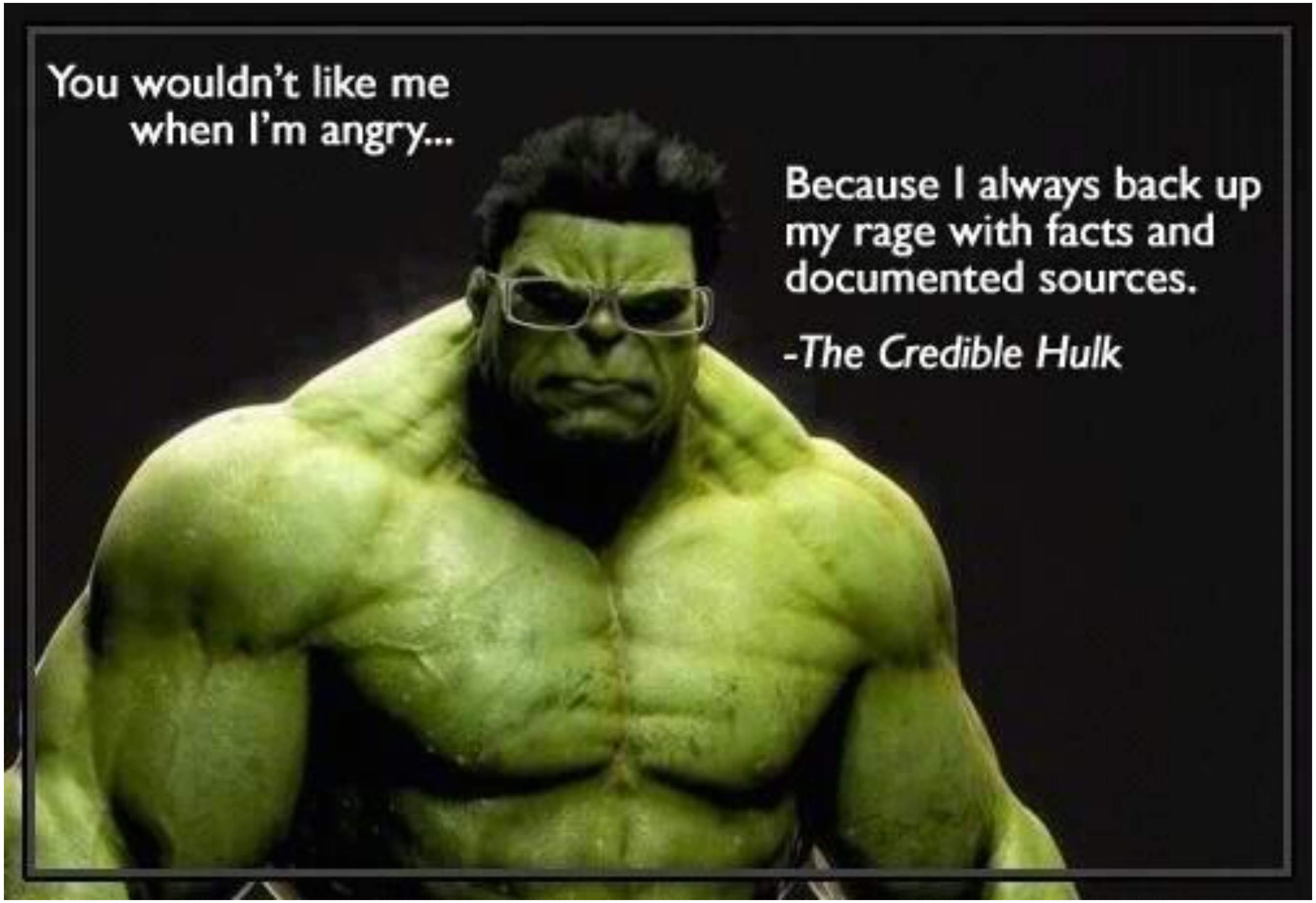
Links to document sources and background information

Ability to rapidly respond to "what if" questions

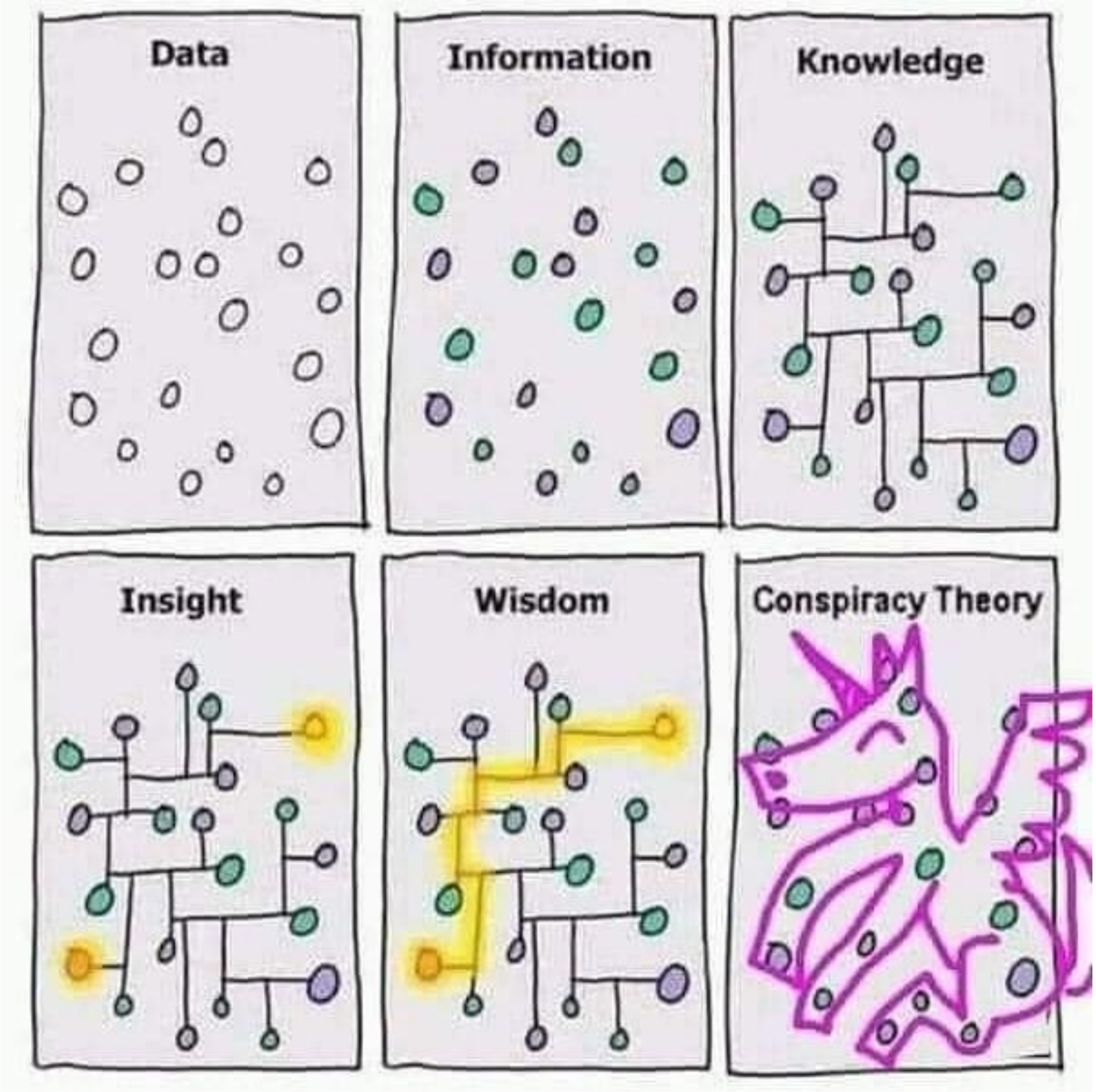
Markdown for pretty titles and text

Notebooks as persuasive objects

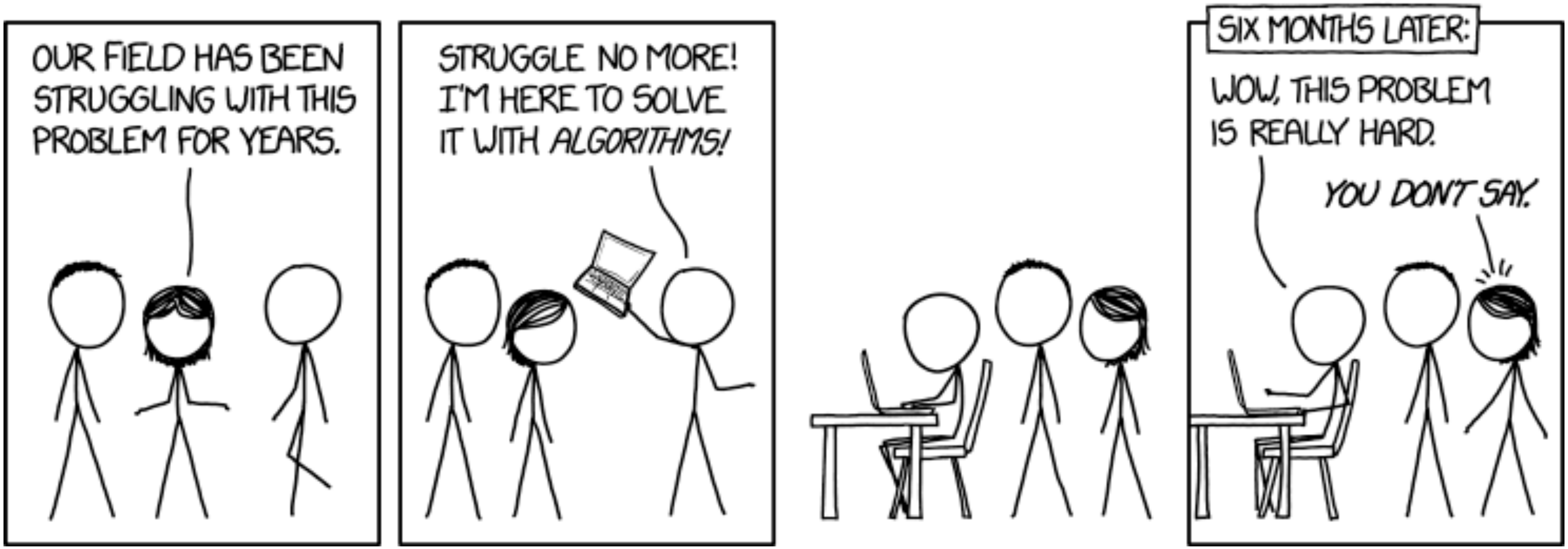
You've seen some of this already:

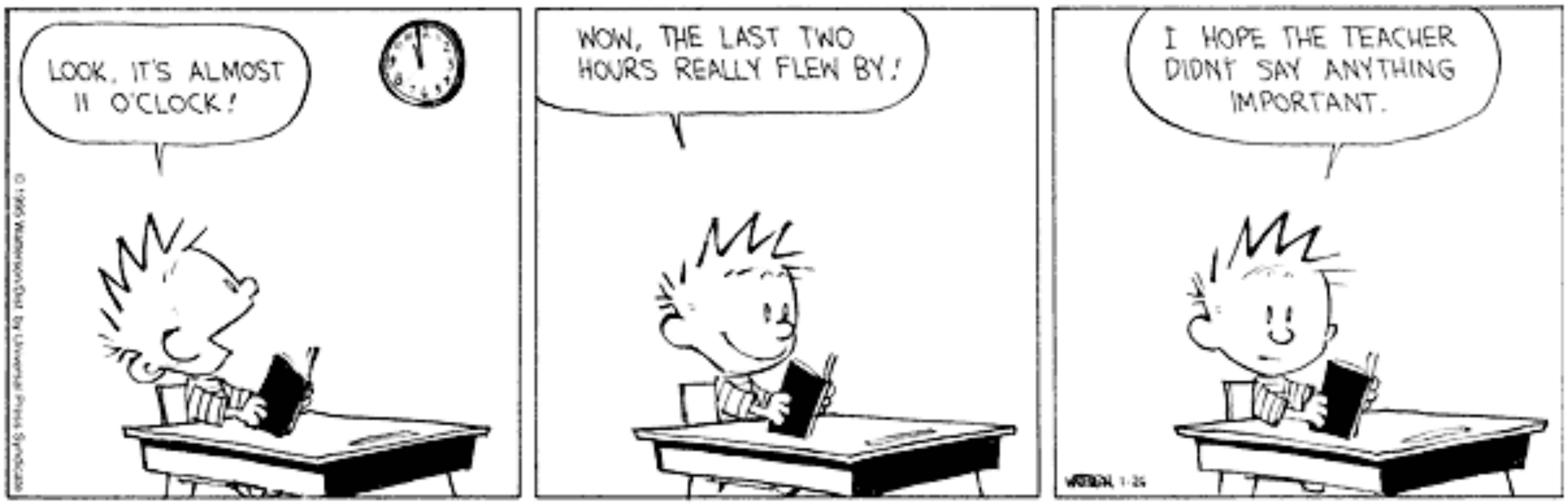


But your conclusions have to be proportionate



Data doesn't always make hard problems easier…





Questions? jacobsen@berkeley.edu

CERN School of Computing

Exercises

Intro - these notebooks & the SWAN service

Simple Applications

Project(s)!

Instructions to get started on Indico (Data Science E1)

https://indico.cern.ch/event/1376644/contributions/5945498/

If you get stuck, ask for help or do an internet search

Learn about each topic, spend more time on ones that interest you.

Don’t try to do every bit of every notebook; pick interesting ones.

Speed is not the issue: no reward for first done or most complete coverage

Not even keeping track

Think about what you’re doing: Learn to use these tools!


DO YOU HAVE AN IDEA FOR YOUR PROJECT YET?
NO, I'M WAITING FOR INSPIRATION.
YOU CAN'T JUST TURN ON CREATIVITY LIKE A FAUCET. YOU HAVE TO BE IN THE RIGHT MOOD.
WHAT MOOD IS THAT?
LAST-MINUTE PANIC.