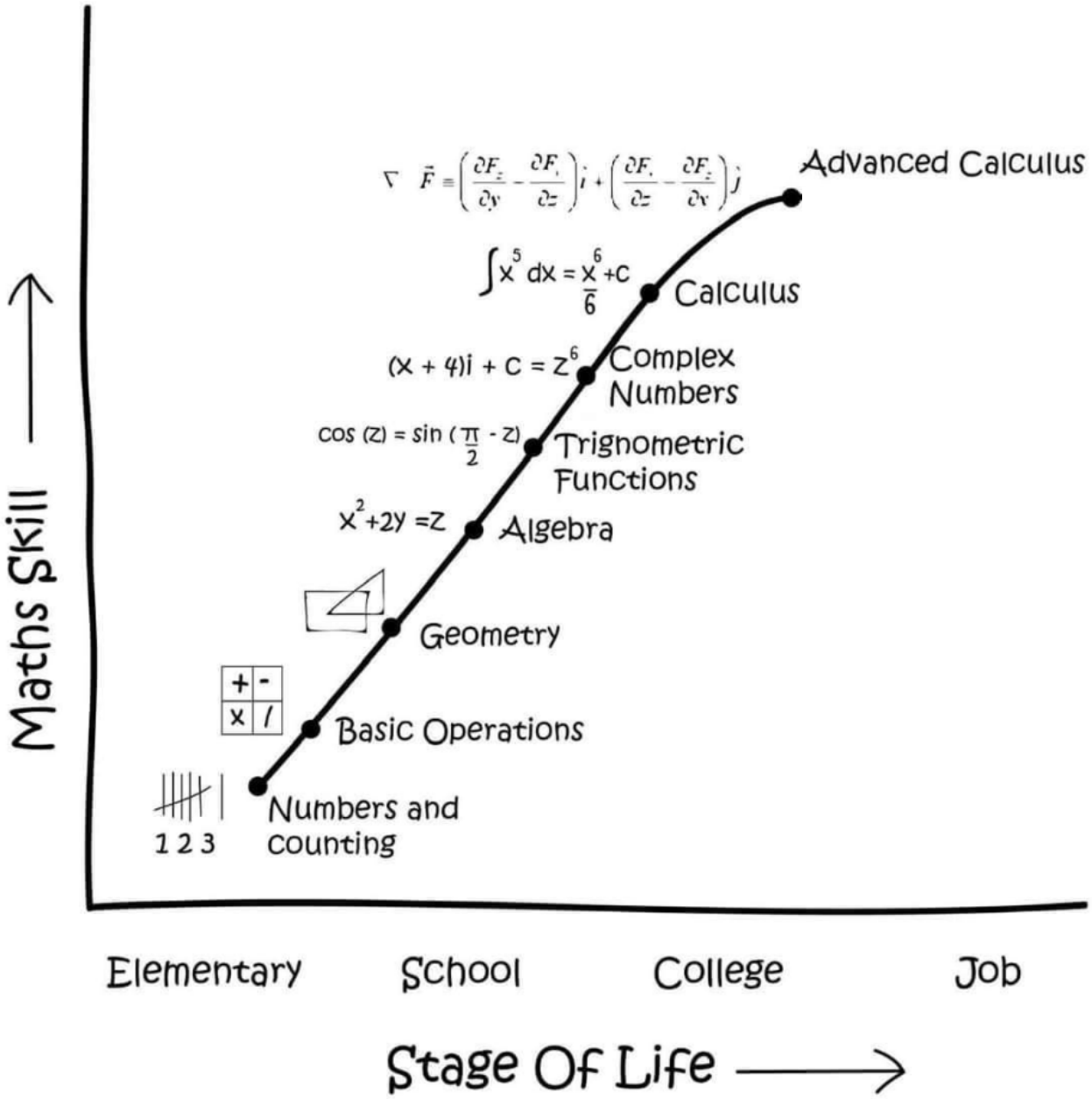
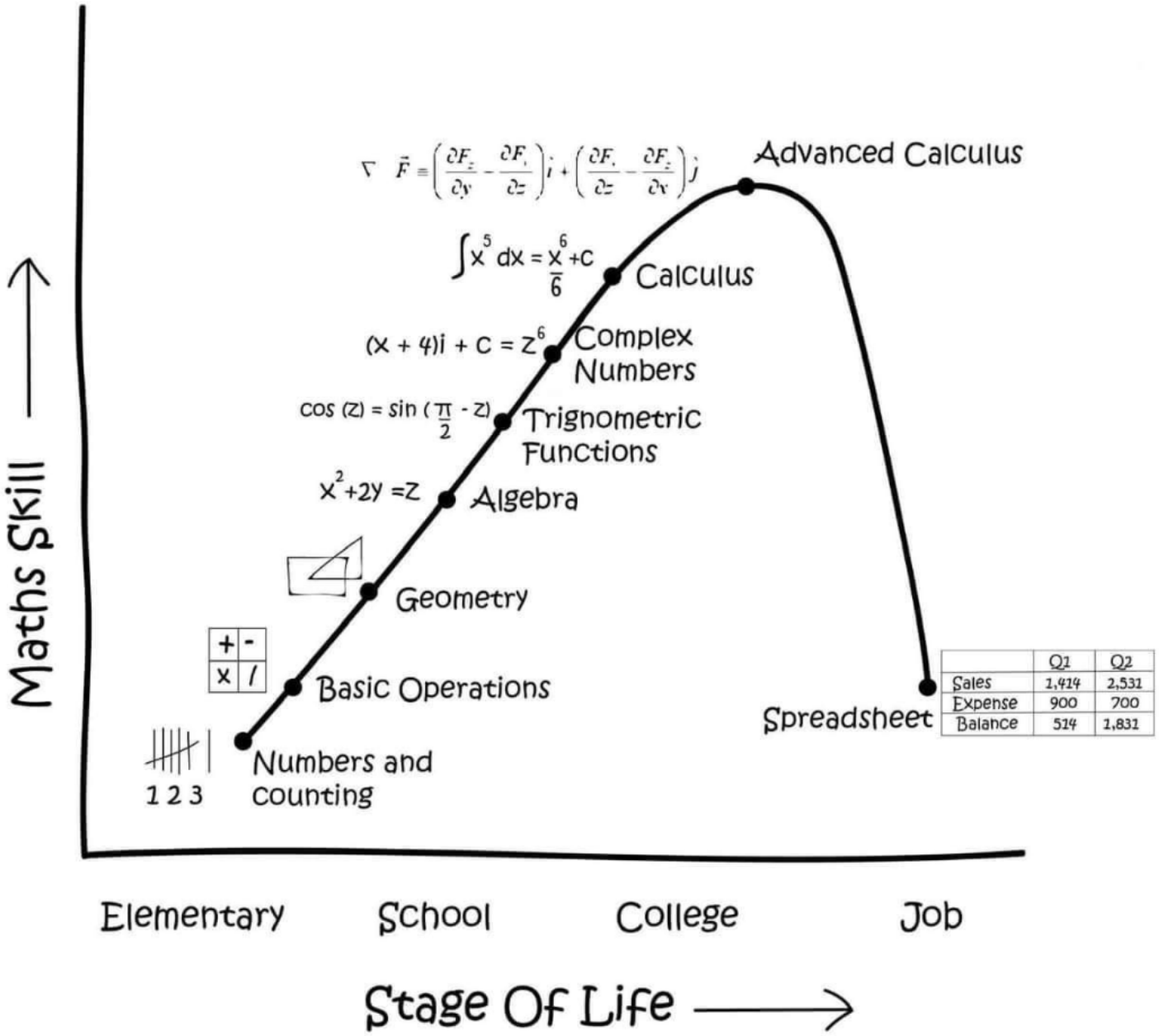
CERN School of Computing

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."

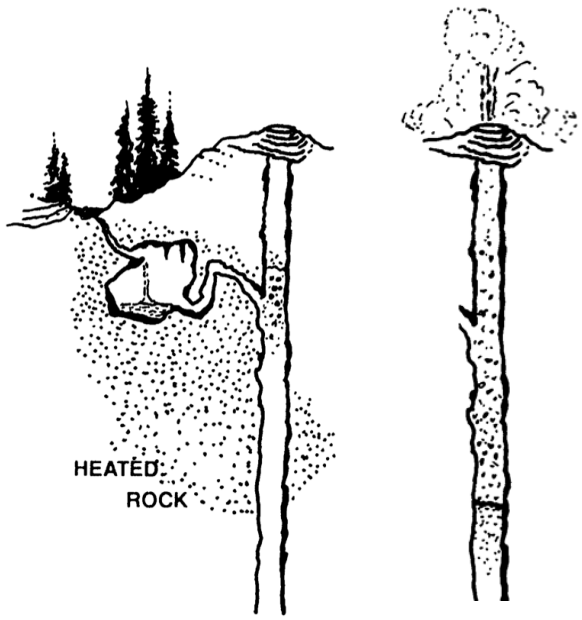

Advanced Calculus
$\nabla \vec{F} = \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z}\right)\hat{i} + \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x}\right)\hat{j}$
$\int x^5 dx = \frac{x^6}{6} + C$
Calculus
$(x + 4)i + C = z^6$
Complex Numbers
$\cos(z) = \sin(\frac{\pi}{2} - z)$
Trignometric Functions
$x^2 + 2y = z$
Algebra
Geometry
+ -
x /
Basic Operations
1 2 3
Numbers and counting
Maths Skill
Elementary
School
College
Job
Stage Of Life



	Q1	Q2
Sales	1,414	2,531
Expense	900	700
Balance	514	1,831

Predicting a Geyser's Eruptions

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
```

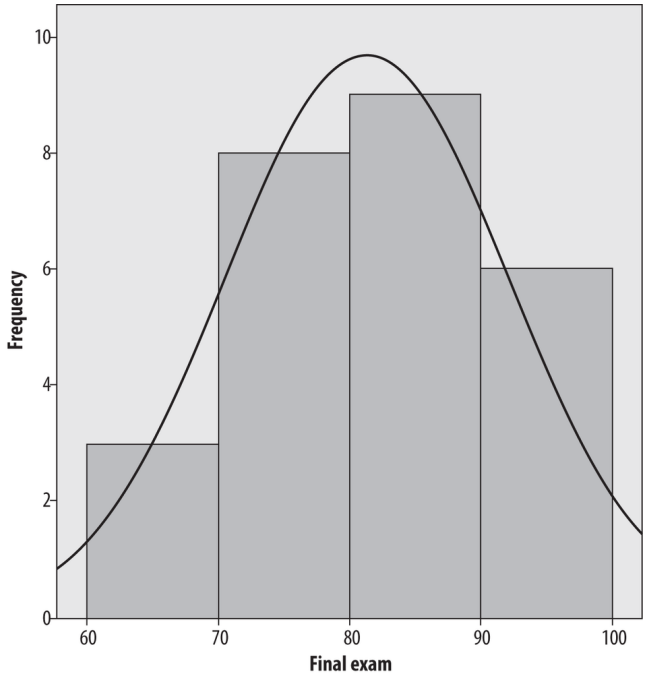
Out[8]:
```
96
```

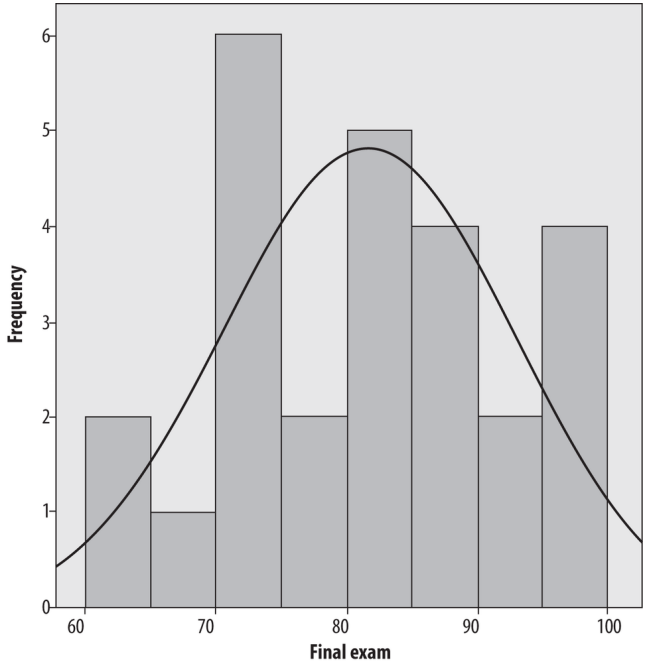
In [9]:
```
# While we're here, let's look at the other data we have
data['Duration'].mean(), data['Duration'].std()        # two statements on a line using commas
```

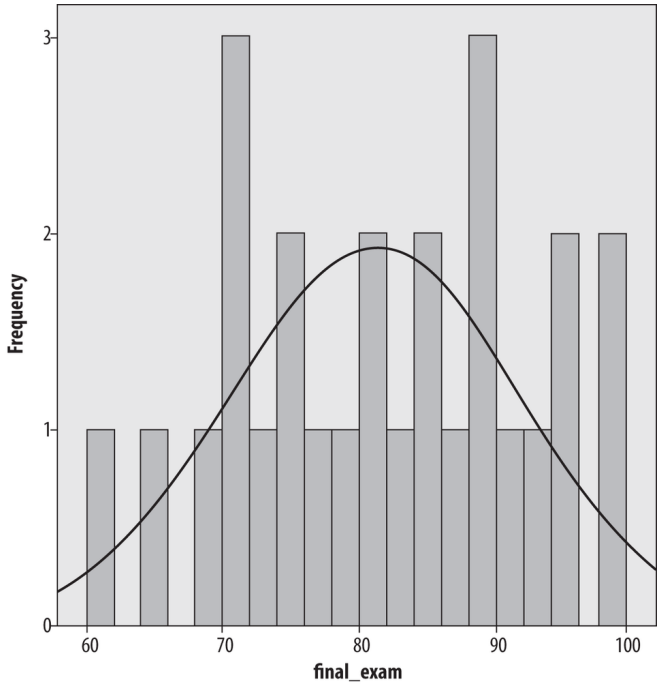
Out[9]:
```
(3.4877830882352936, 1.139271210225768)
```

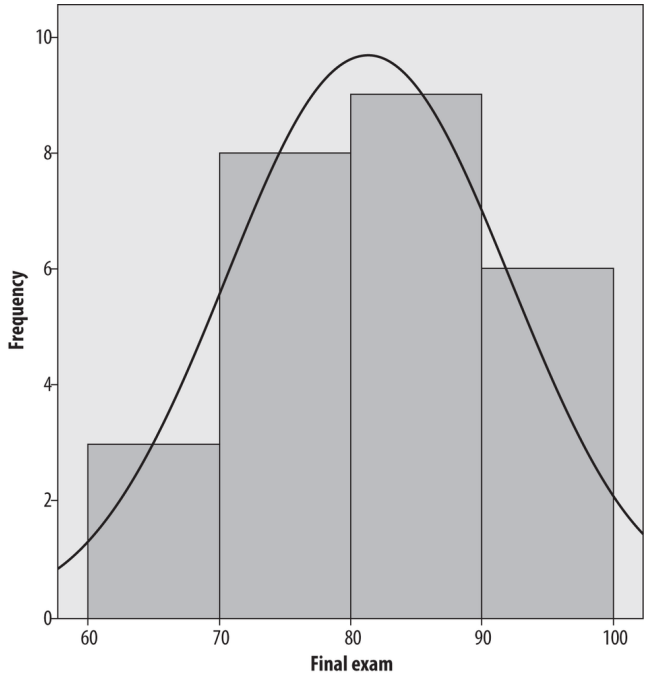
CERN School of Computing

Before we plot: On binning

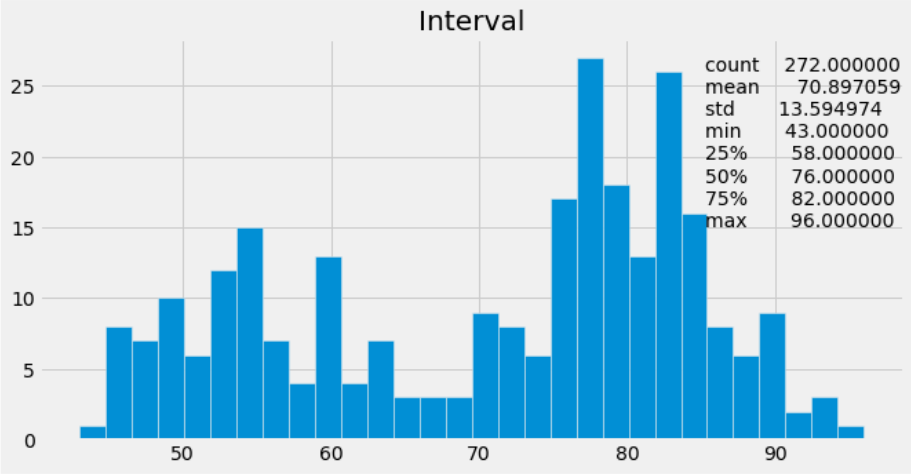




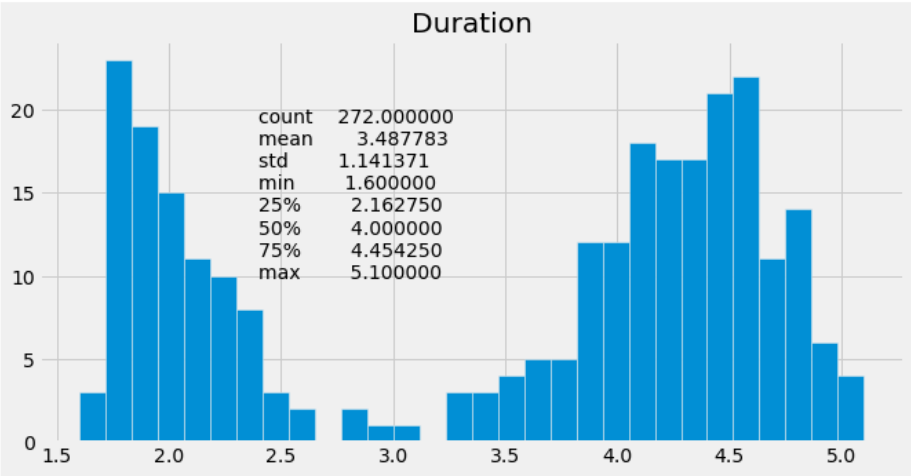

Frequency
3
2
1
0
60
70
80
90
100
final_exam


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
```

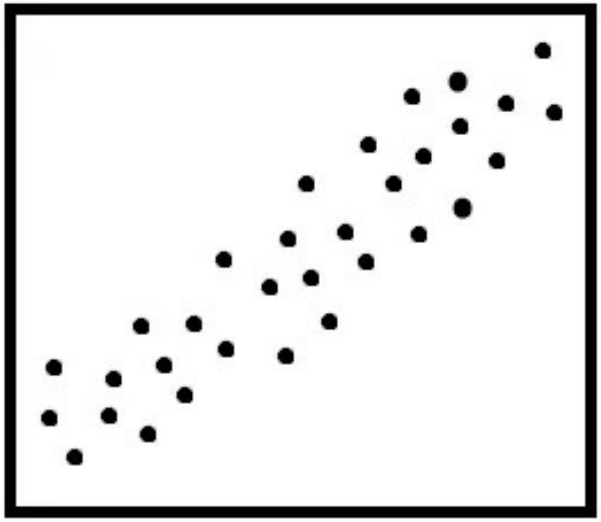


In [11]:
```
# Not particularly Gaussian!
# Maybe there's two peaks there. But that still doesn't give us a better way to predict the eruption.
# Look at other information we have:
plt.hist(data['Duration'], bins=30)
plt.figtext(0.3,0.4, data.to_df()['Duration'].describe().to_string())
plt.title("Duration");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

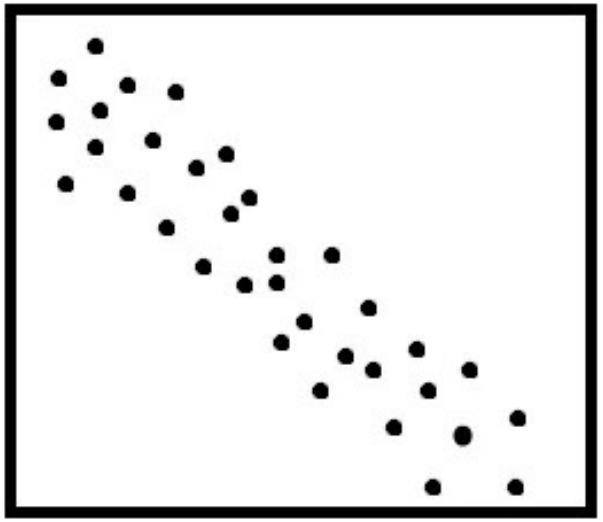


In [12]:
```
# Maybe there's a correlation?
np.corrcoef(data['Duration'], data['Interval'])
```

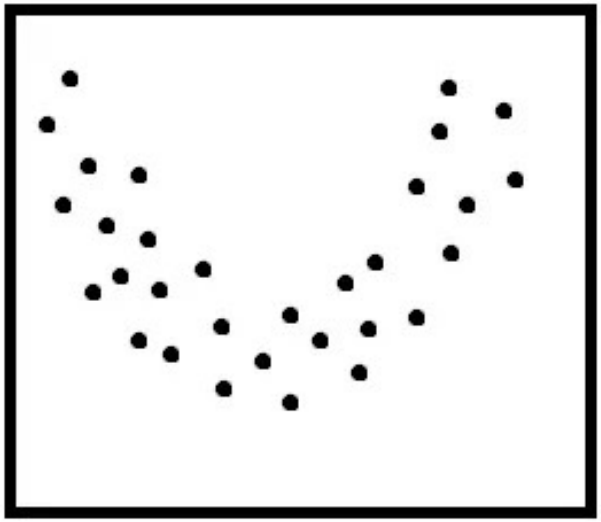
Out[12]:
```
array([[ 1.        ,  0.90081117],
       [ 0.90081117,  1.        ]])
```



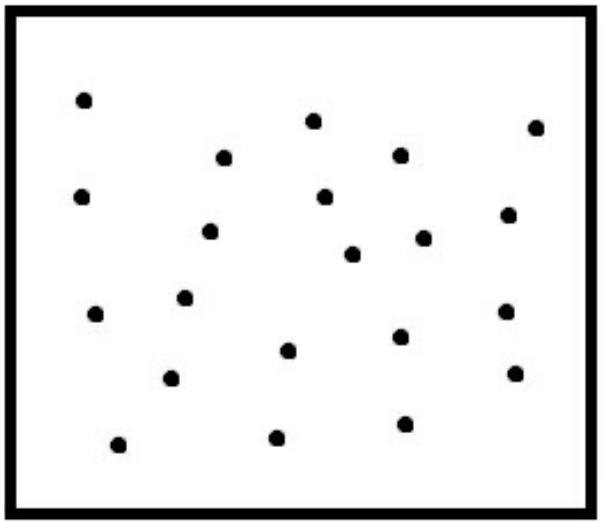
positive linear association



negative linear association



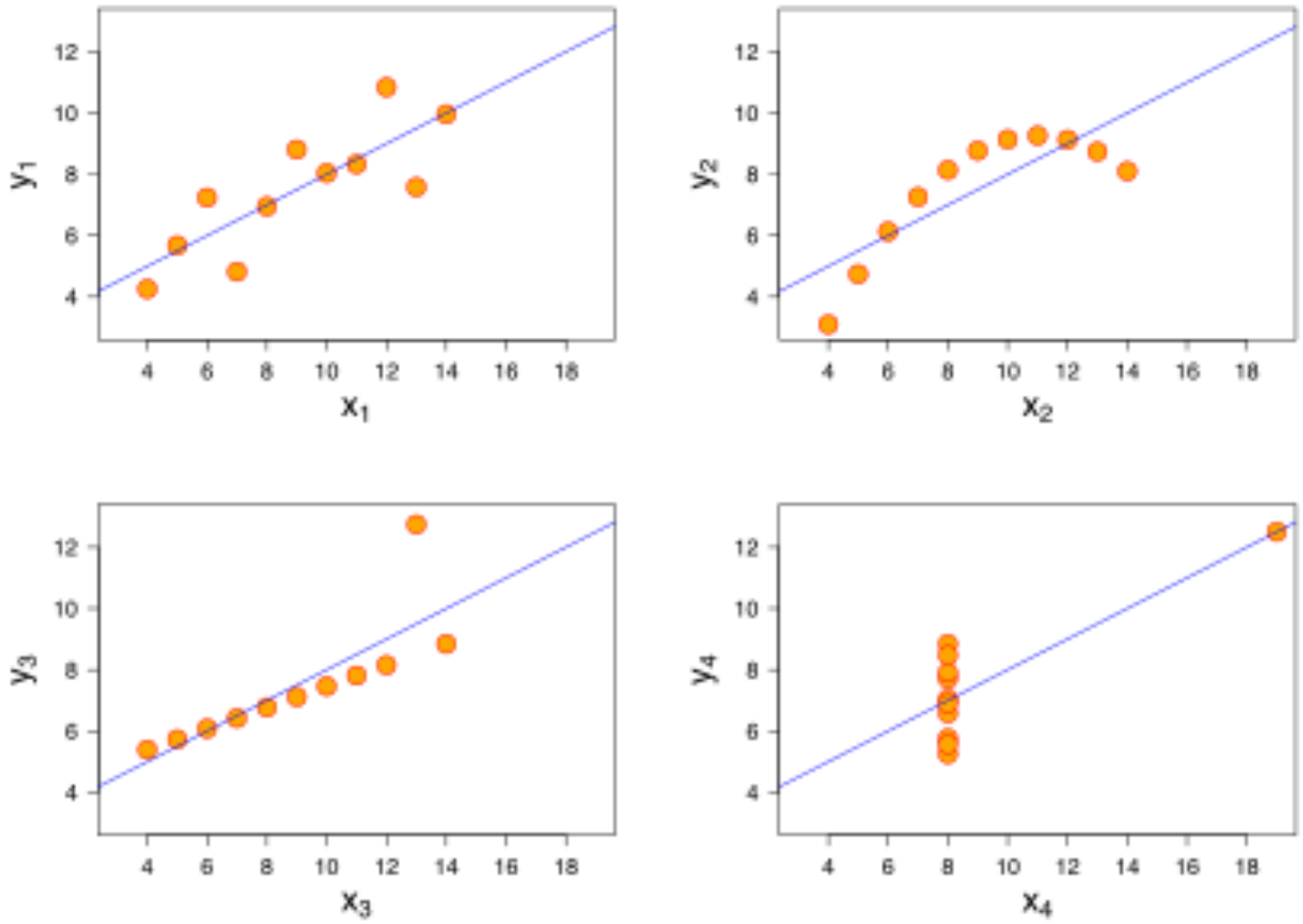
nonlinear association



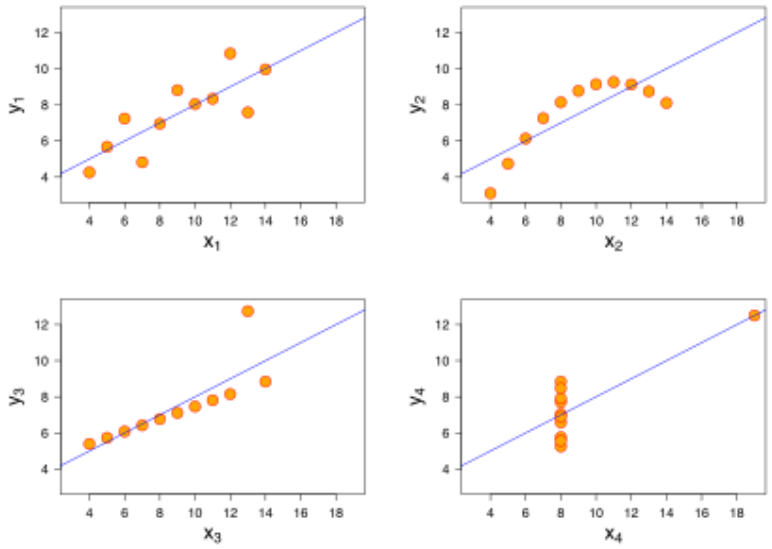
no association

CERN School of Computing

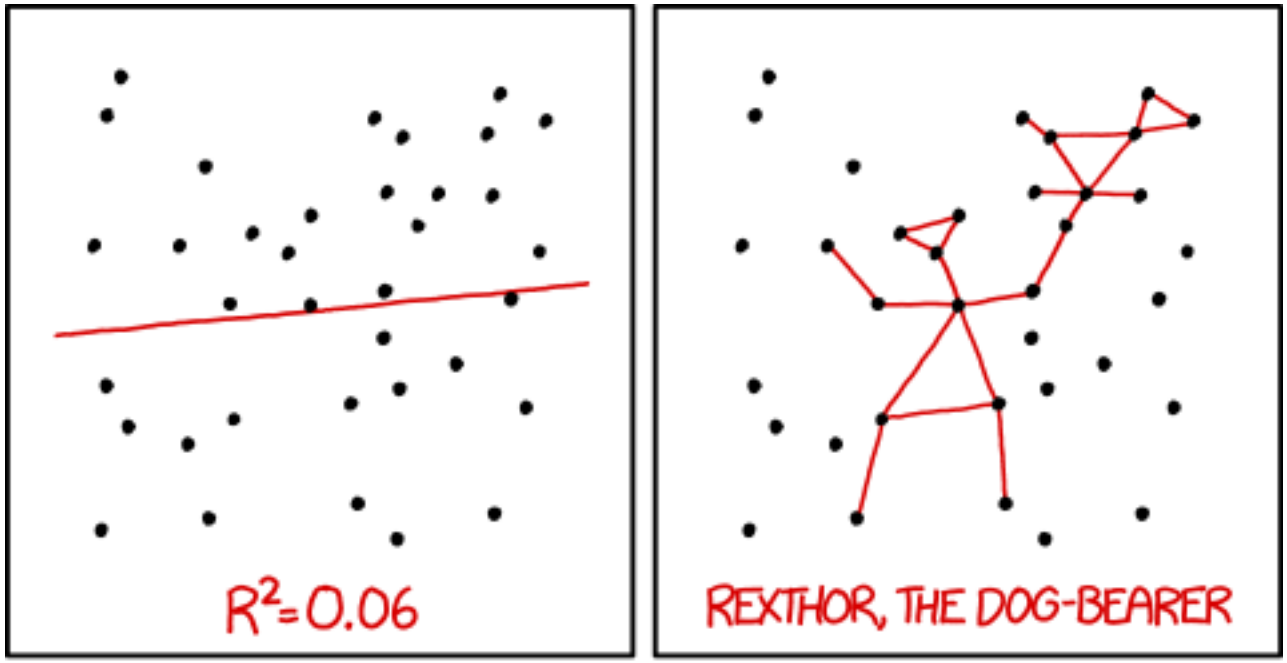
Anscombe's Quartet



Anscombe's Quartet

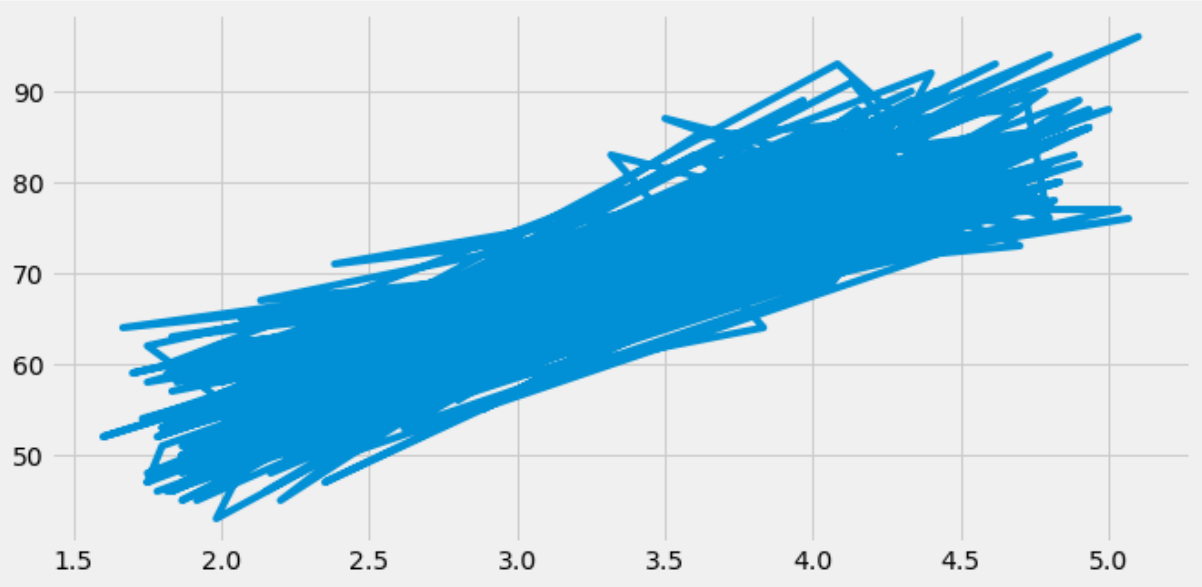


Property	Value	Accuracy
Mean of x	9	exact
Sample variance of x : s_x^2	11	exact
Mean of y	7.50	to 2 decimal places
Sample variance of y : s_y^2	4.125	±0.003
Correlation between x and y	0.816	to 3 decimal places
Linear regression line	$y = 3.00 + 0.500x$	to 2 and 3 decimal places, respectively
Coefficient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

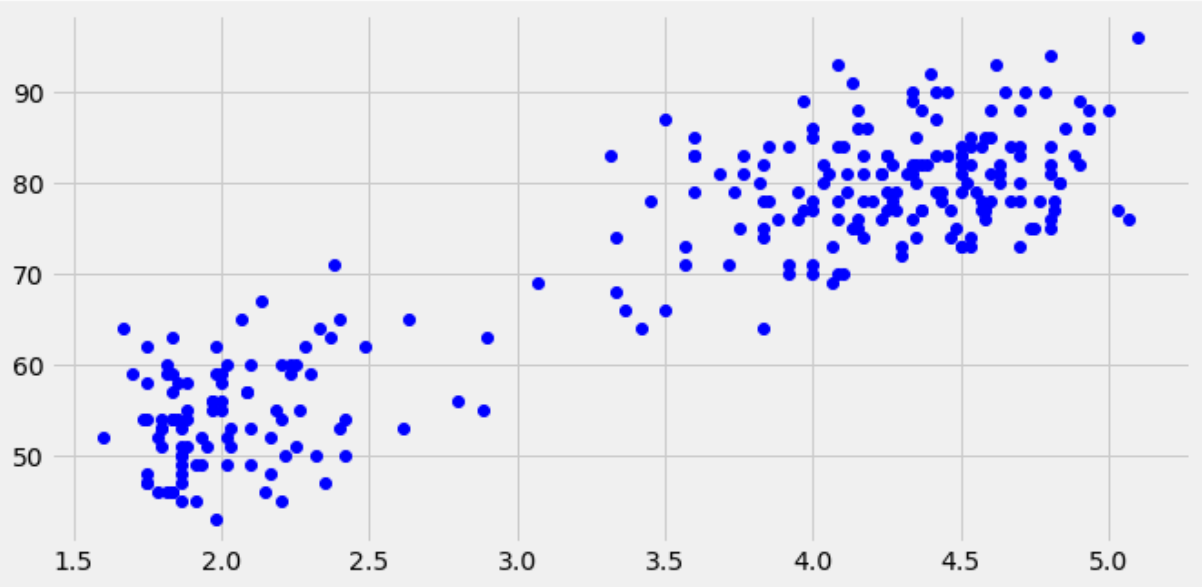


I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER TO GUESS THE DIRECTION OF THE CORRELATION FROM THE SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

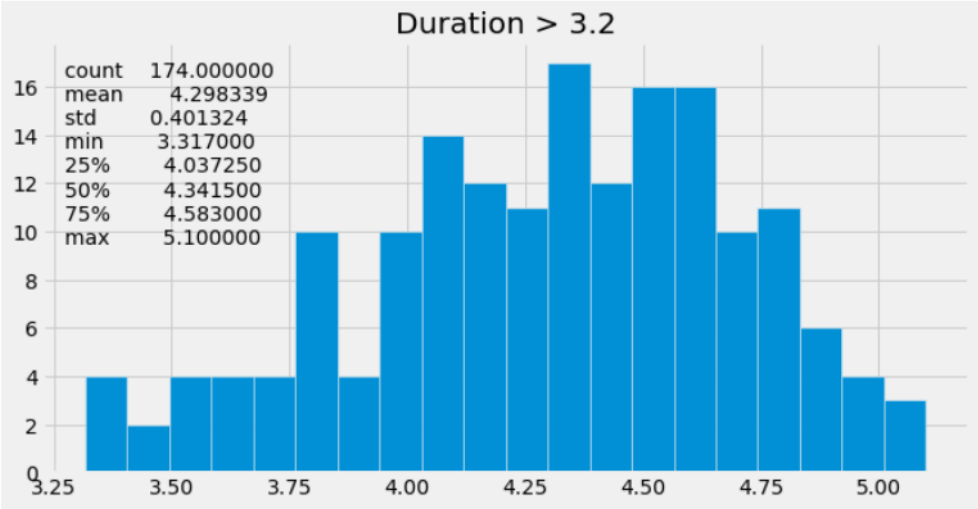
In [13]:
```
# that's pretty strong, let's look at it
plt.plot(data['Duration'], data['Interval']);
```



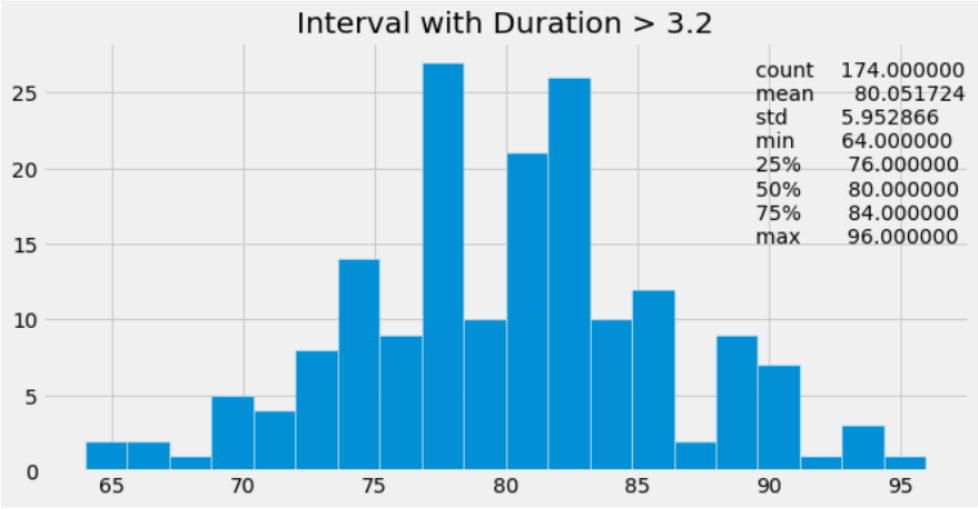
In [14]:
```
# Maybe plotting as points would be better...
plt.plot(data['Duration'], data['Interval'],"ob");   # o: dots  b: blue
```



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");
```

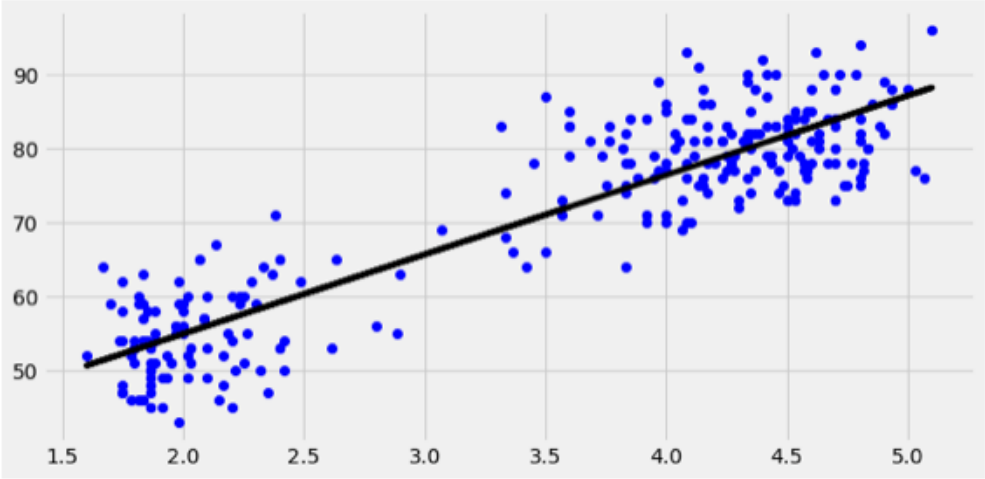


In [16]:
```
# But of course duration is more compact because we selected a narrower range,  How about interval?
plt.hist(long_duration_data['Interval'], bins=20)
plt.figtext(0.75,0.5, long_duration_data.to_df()['Interval'].describe().to_string())
plt.title("Interval with Duration > 3.2");
```

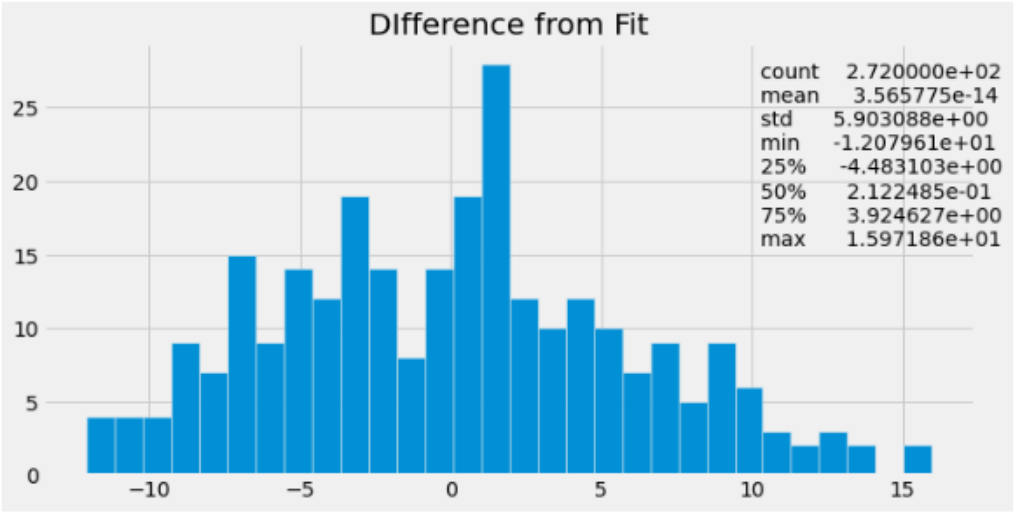


In [17]:
```
# We're down to 50% in 8 minutes and an RMS of 6 minutes on a mean of 80; 10%!
#
```

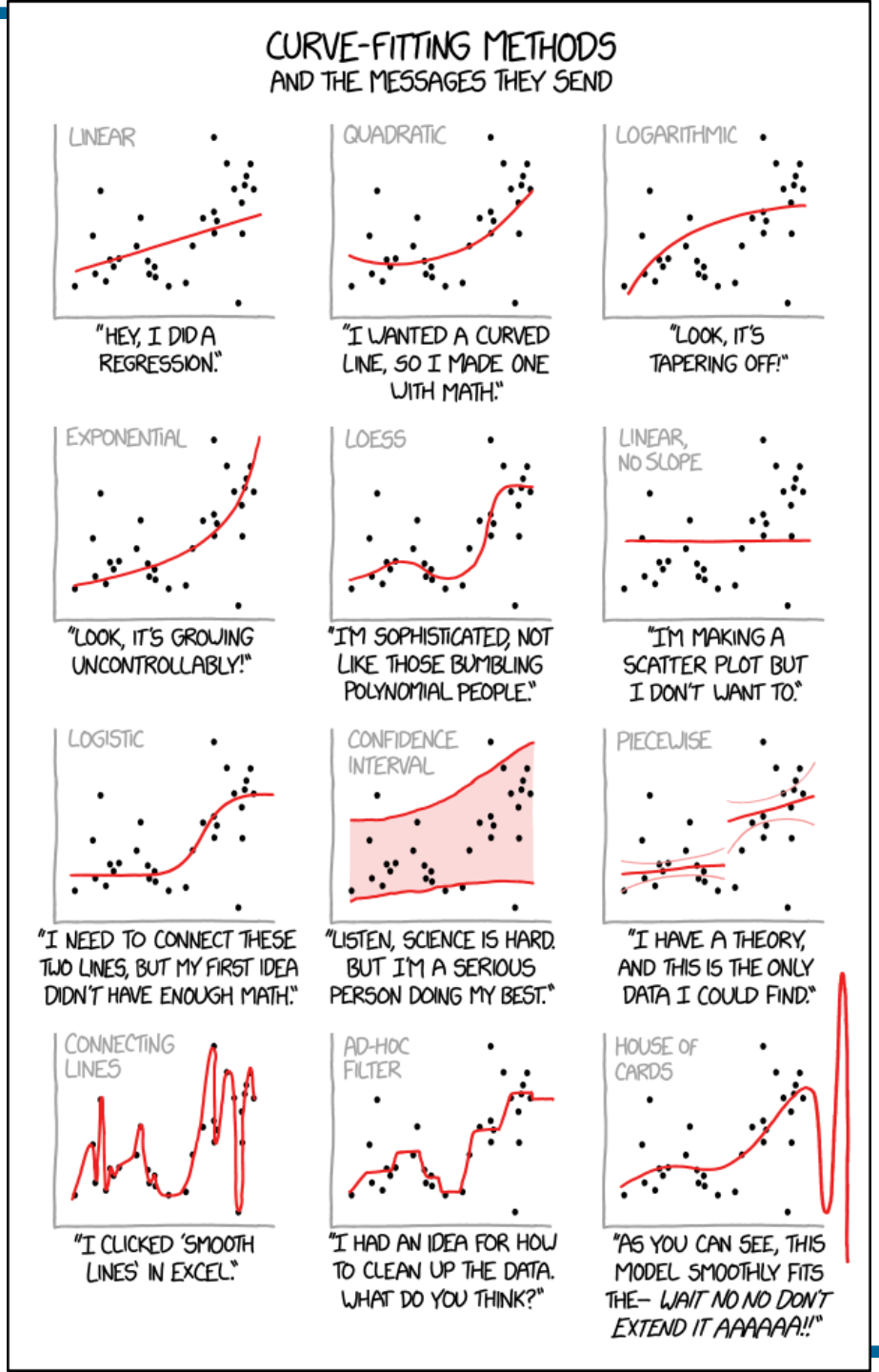
In [18]:
```
# Try fitting a line instead using two populations
d = np.polyfit(data['Duration'], data['Interval'],1)
f = np.poly1d(d)
data['trendline'] = f(data['Duration'])

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



In [19]:
```
# 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]:
```
# 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

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.

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:

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

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]

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

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.

In [2]:
```
# First, list two possible results
sides = make_array('Butter Side Up', 'Butter Side Down')
```

In [3]:
```
# Make that into a table
possible_outcomes = Table().with_column('Outcome', sides)
```

In [4]:
```
possible_outcomes
```

Out[4]:

Outcome
Butter Side Up
Butter Side Down

In [5]:
```
# Ask for 48 cases where the output is sampled (chosen) from those two possibilities
simulated_experiment = possible_outcomes.sample(48)
```

In [6]:
```
simulated_experiment
```

Out[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.

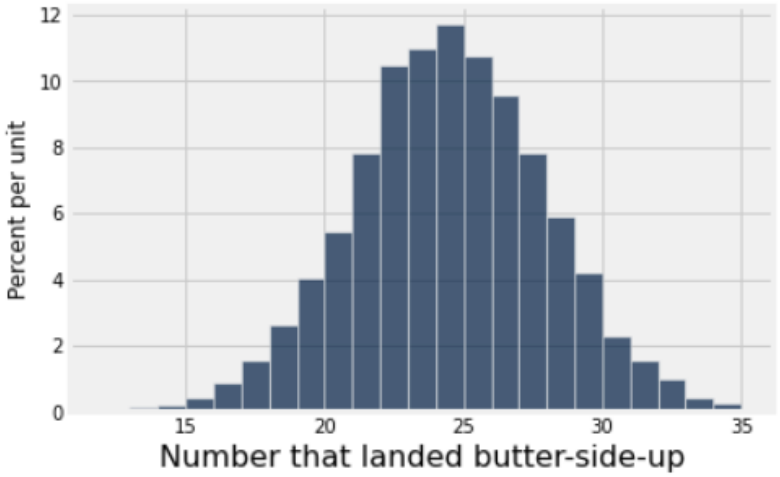
In [10]:
```
counts = make_array()
for i in np.arange(10000): # 10000 repetitions
    one_simulated_episode = possible_outcomes.sample(48)
    number_up = count_up(one_simulated_episode)
    counts = np.append(counts, number_up)
results = Table().with_column('Number that landed butter-side-up', counts)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

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()
```

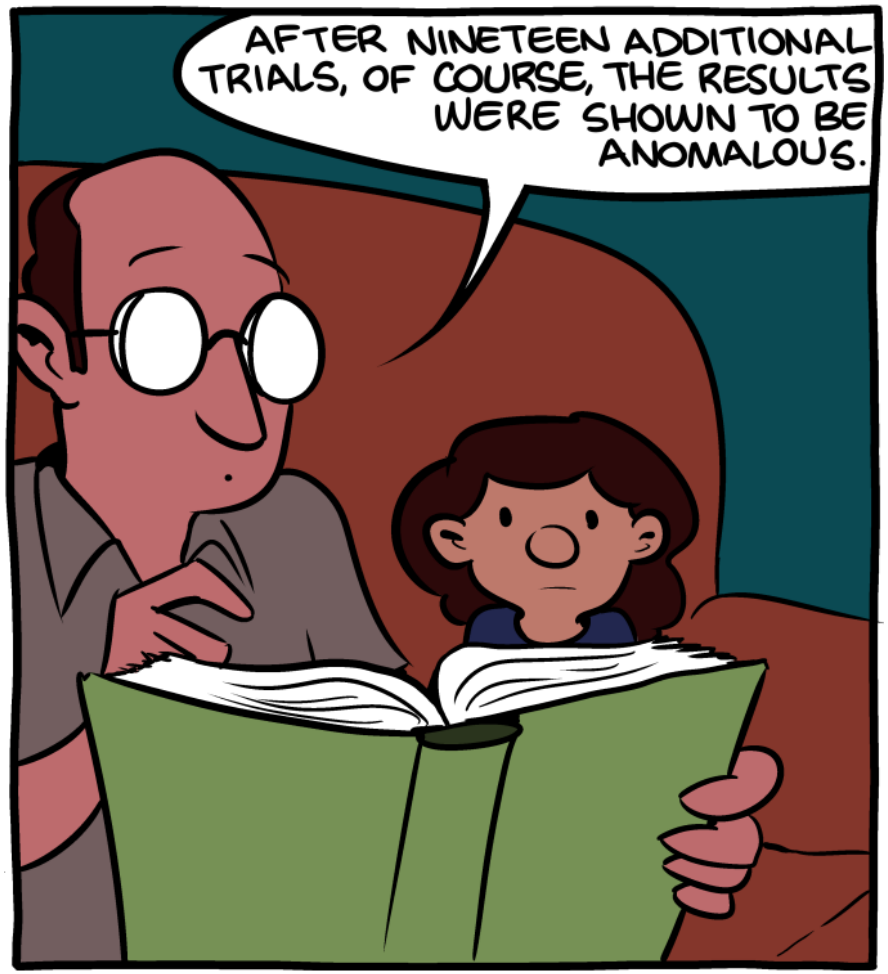
Out[14]:
```
(23.982099999999999, 3.4885784483081359)
```

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?
```

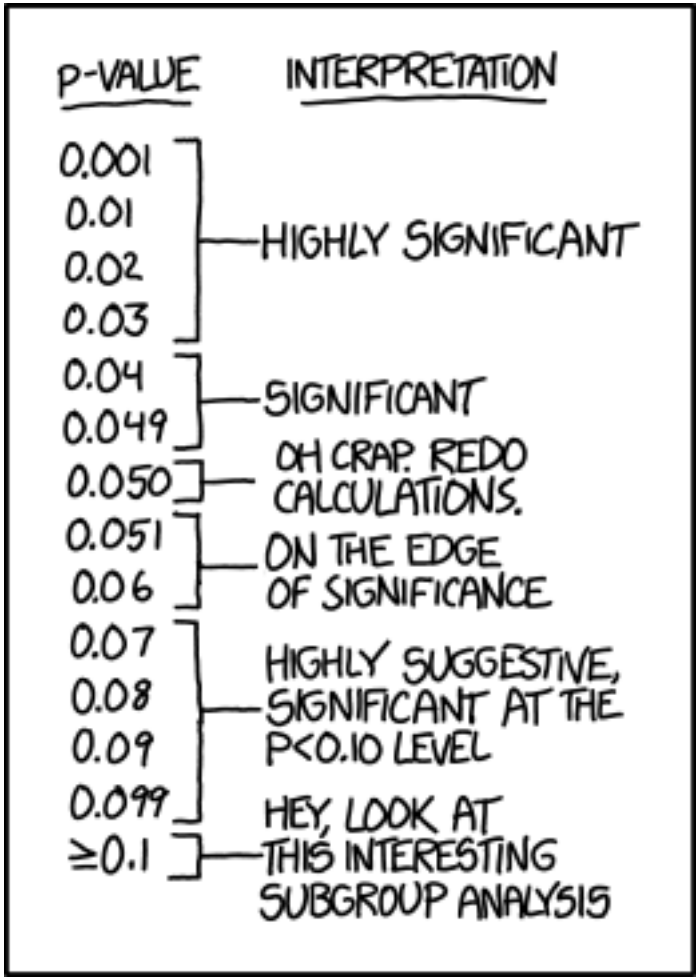
Sometimes you need to run the experiment for longer & get more data…



"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

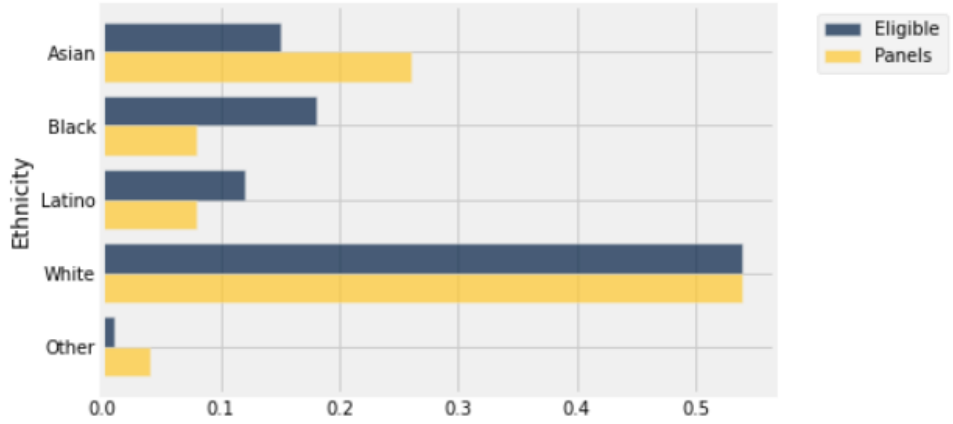
Alameda County Juries

In [2]:
```
# Data from an ACLU 2018 report
# Racial and Ethnic Disparities in Alameda County Jury Pools
# https://www.aclunc.org/sites/default/files/racial_and_ethnic_disparities_in_alameda_county_jury_pools.pdf
#
# "Eligible" is those adults who are eligible to serve on a jury
# "Panels" is the group that's selected by the lawyers to serve on a jury
# 1453 people were included in the panels

alameda = Table().with_columns(       # build by columns (see below for by-row)
    'Ethnicity', make_array('Asian', 'Black', 'Latino', 'White', 'Other'),
    'Eligible', make_array(0.15, 0.18, 0.12, 0.54, 0.01),
    'Panels', make_array(0.26, 0.08, 0.08, 0.54, 0.04)
)

alameda.set_format([1, 2], PercentFormatter(0))  # the data columns hold a 0:1 number, but show in percent
```

Out[2]:

Ethnicity	Eligible	Panels
Asian	15%	26%
Black	18%	8%
Latino	12%	8%
White	54%	54%
Other	1%	4%

In [3]:
```
# plot categorial (i.e. not numeric) data as bar chart
alameda.barh(0)
```



Total Variation Distance

In [4]:
```
# Use the difference between two values as a metric of how much they vary
diff = alameda.with_column('Difference',
                           alameda.column('Eligible') - alameda.column('Panels'))
diff
```

Out[4]:

Ethnicity	Eligible	Panels	Difference
Asian	15%	26%	-0.11
Black	18%	8%	0.1
Latino	12%	8%	0.04
White	54%	54%	0
Other	1%	4%	-0.03

In [5]:
```
# take absolute value to keep all differences raising the metric
abs_diff = diff.with_column('Abs. Difference',
                            np.abs(diff.column('Difference')))
abs_diff
```

Out[5]:

Ethnicity	Eligible	Panels	Difference	Abs. Difference
Asian	15%	26%	-0.11	0.11
Black	18%	8%	0.1	0.1
Latino	12%	8%	0.04	0.04
White	54%	54%	0	0
Other	1%	4%	-0.03	0.03

In [6]:
```
sum(abs_diff.column('Abs. Difference')) / 2  # if one bar goes up, another goes down => divide by 2
```

Out[6]:
```
0.14000000000000001
```

Simulating the statistic

In [10]:
```
# define a function to create a random panel
def get_one_simulated_panel():
    """ Create a panel of 1453 people """
    return alameda.select('Ethnicity').sample(1453, weights=alameda.column('Eligible'))
```

In [11]:
```
# do a single simulation by adding a "Random" column
def simulate_once():
    """ Create one simulated table """
    simulated_panel = get_one_simulated_panel()
    counts = simulated_panel.group('Ethnicity')
    sim_proportions = counts.select('Ethnicity').with_column('Random',
                                    counts.column('count') / 1453)
    sim_proportions.set_format(1, PercentFormatter(0))
    return alameda.join('Ethnicity', sim_proportions)
```

In [12]:
```
simulate_once()
```

Out[12]:

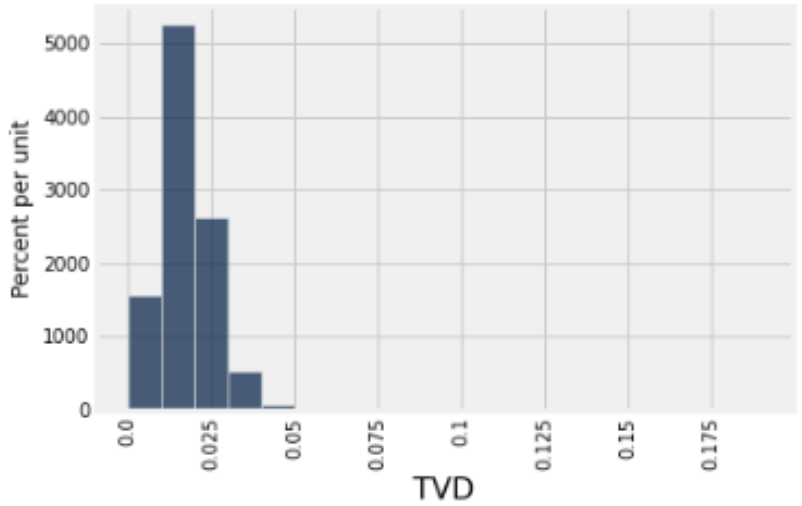
Ethnicity	Eligible	Panels	Random
Asian	15%	26%	17%
Black	18%	8%	18%
Latino	12%	8%	12%
Other	1%	4%	1%
White	54%	54%	52%

In [13]:
```
# Compute the empirical distribution of TVDs by simulation
tvds = make_array()

for i in np.arange(5000): # 5000 repetitions of the simulation
    sim_results = simulate_once()
    tvds = np.append(tvds, table_tvd(sim_results, 'Eligible', 'Random'))

results = Table().with_column('TVD', tvds)
results
```

Out[13]:

TVD
0.00920853
0.0152512
0.0113696

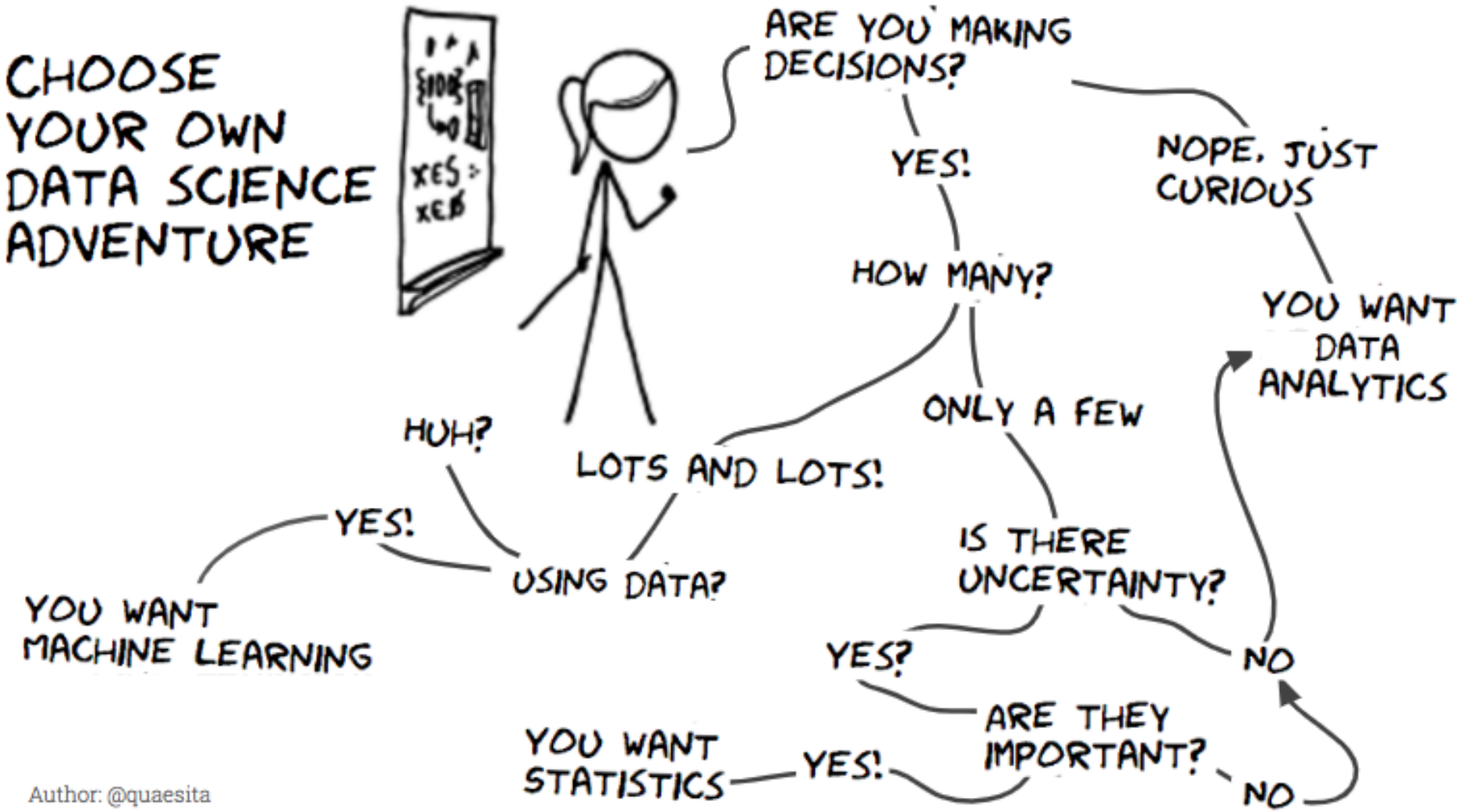
In [33]:
```
results.hist(bins=np.arange(0, 0.2, 0.01))
```



P-value

In [34]:
```
results.where(0, are.above_or_equal_to(0.14)).num_rows / results.num_rows
```

Out[34]: 0.0


CHOOSE YOUR OWN DATA SCIENCE ADVENTURE
ARE YOU MAKING DECISIONS?
YES!
NOPE. JUST CURIOUS
HOW MANY?
YOU WANT DATA ANALYTICS
ONLY A FEW
HUH?
LOTS AND LOTS!
YES!
USING DATA?
IS THERE UNCERTAINTY?
YOU WANT MACHINE LEARNING
YES?
NO
ARE THEY IMPORTANT?
YOU WANT STATISTICS
YES!
NO
Author: @quaesita

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/1254984/contributions/5272131/

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!