Stan Is The Plan

Breck Baldwin

NYC PyData 2019

Goals

- Learn to code Stan models
- Focus on prediction—A.I. and machine learning
- Learn just enough statistics to learn more
- Learn the pitfalls and common mistakes of Stan

Outline Section 1

- Should have CmdStanPy installed by afternoon
- Run Final Example: puttBet.py
- Write Stan models of increasing complexity
- Explore execution environment
- Describe how inference is done

Resources

Tutorial repository—please download

- https://github.com/breckbaldwin/StanIsThePlanDist

Install CmdStanPy—also install CmdStan

- https://cmdstanpy.readthedocs.io/en/latest/index.html

Review if you are waiting

- Pre/Post test: https://forms.gle/J5kqACo5m6cy9WKd9
- Look ahead in the slides

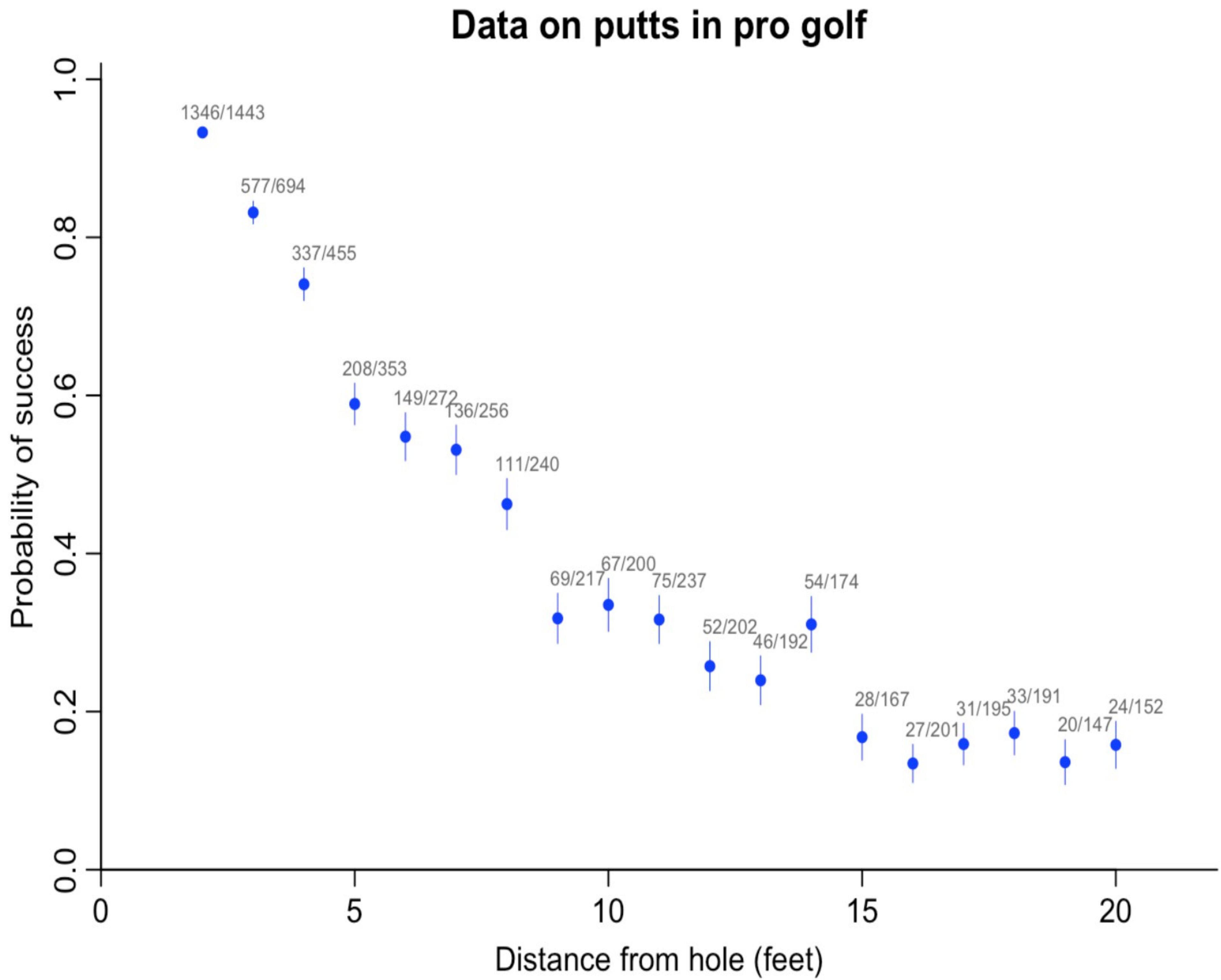
Our Application

PuttBet--Predict future golf putts

Derived from:

https://mc-stan.org/users/documentation/case-studies/golf.html

- Historical putting data
  - 2 feet, sink 1346 out of 1443 attempts
  - ...
  - 20 feet, sink 24 of 152 attempts
- Various distances of putt we want to bet on:
  - 2...30 feet
- Return 'chance_in_5' value



Data on putts in pro golf
Probability of success
Distance from hole (feet)
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
1346/1443
577/694
337/455
208/353
149/272
136/256
111/240
69/217
67/200
75/237
52/202
46/192
54/174
28/167
27/201
31/195
33/191
20/147
24/152

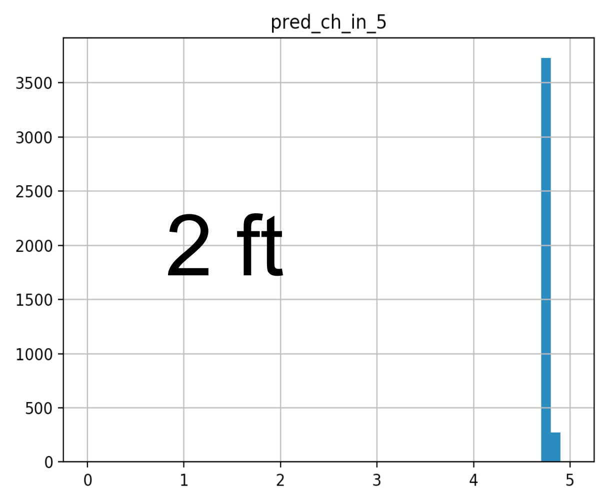
Flow

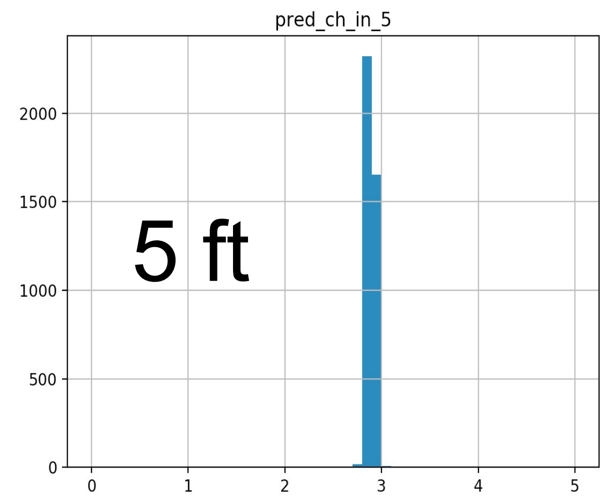
- Python reads from the command line
  - Stan file to run
  - Data to send, putt distance
- Runs Stan model
  - Compile if necessary
  - Fit data (not part of compile)
  - Predicts performance for stated putt distance
- Summarizes results
- Plots 'chance_in_5'

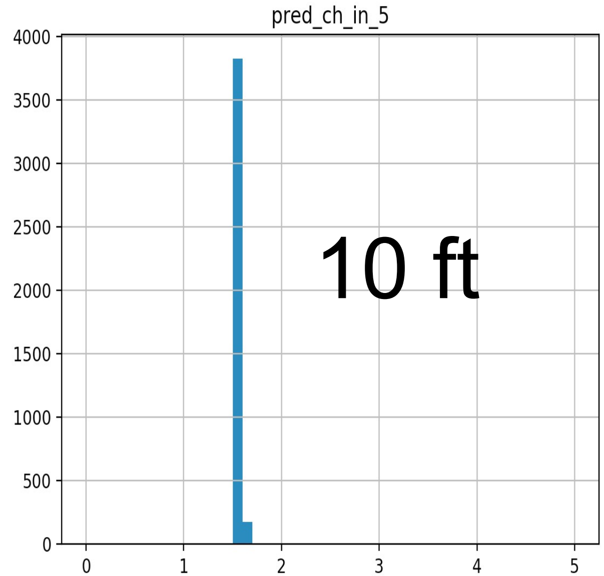
Run the goal program

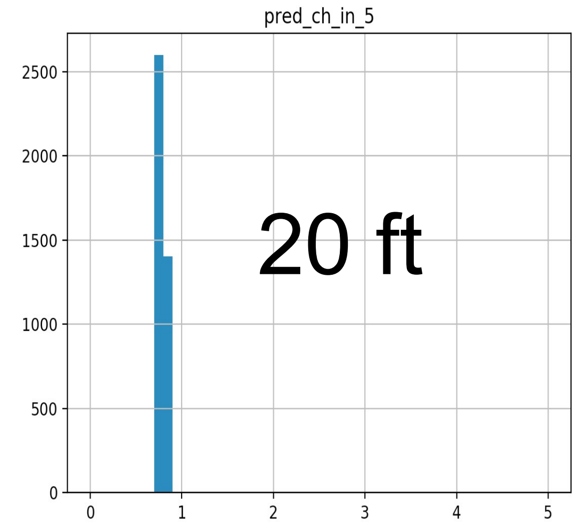
```
> cd <path>/StanIsThePlanDist
> python puttBet.py stan/mechanistic_golf.stan 2
```

Some Predictions







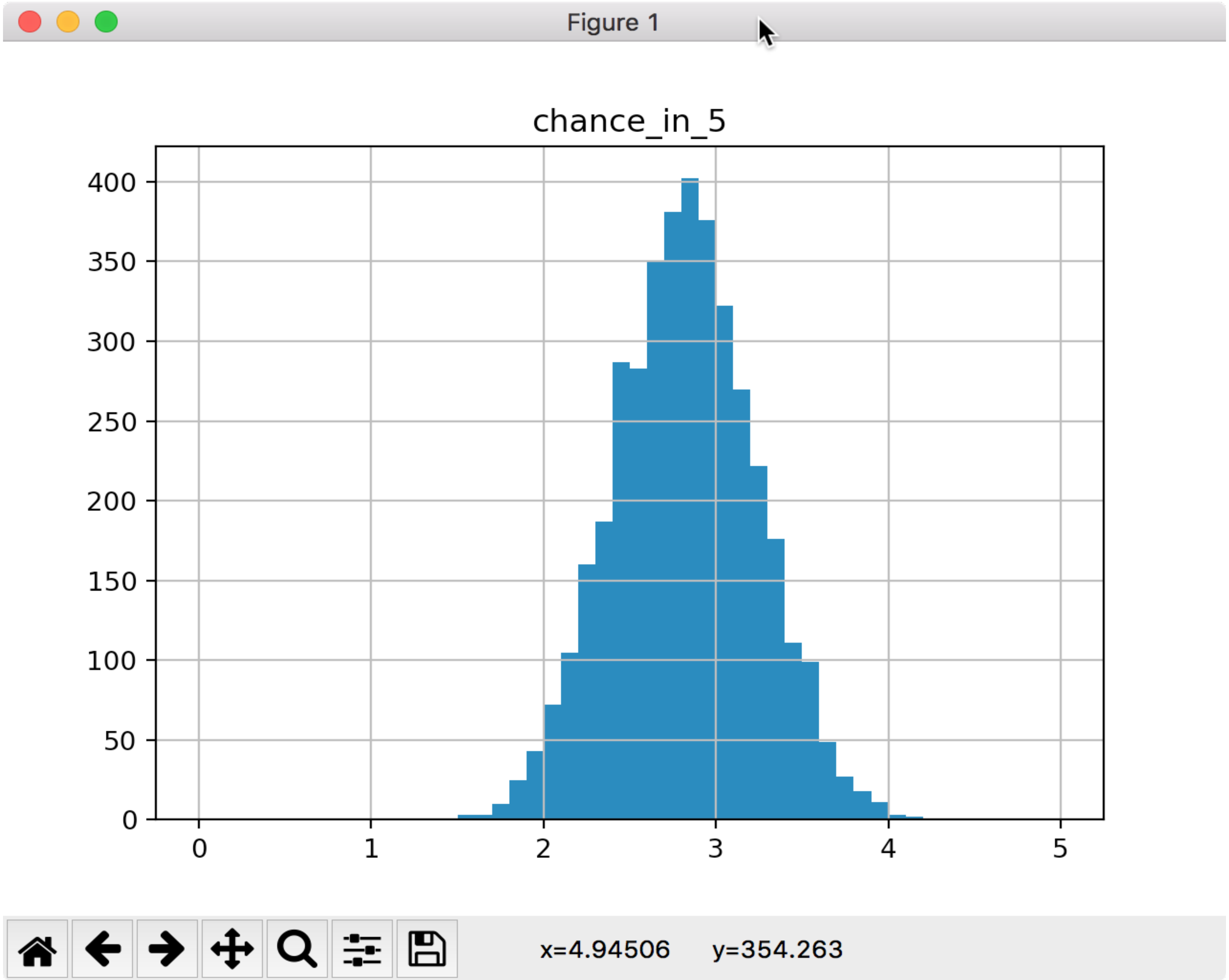


The Basic Idea

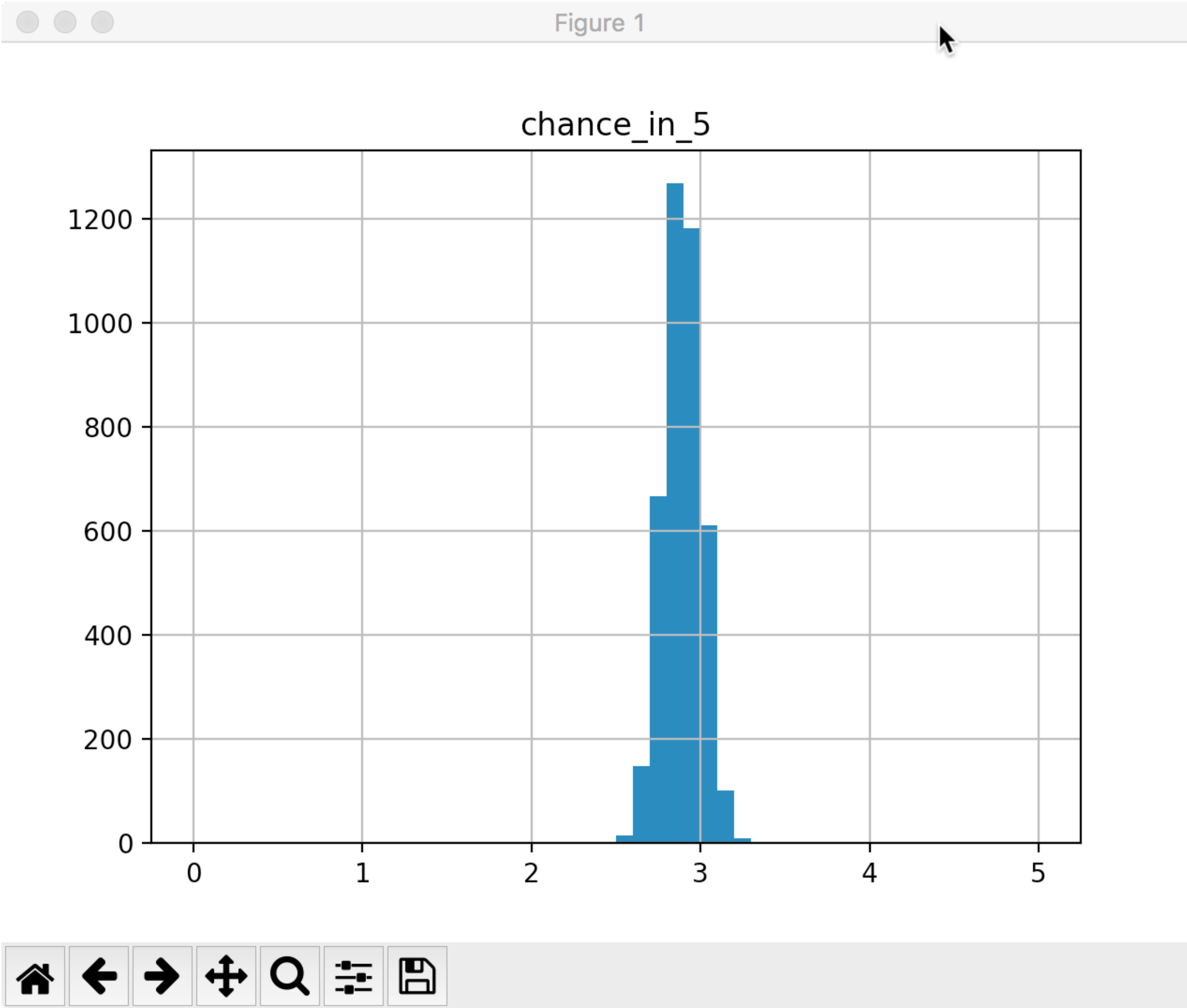
- Get a bunch of training data
- Build a program generalizes that training data so it will work with novel data
- Evaluate progress and refine
- But we are doing it in a Bayesian way

Bayesian Point 1

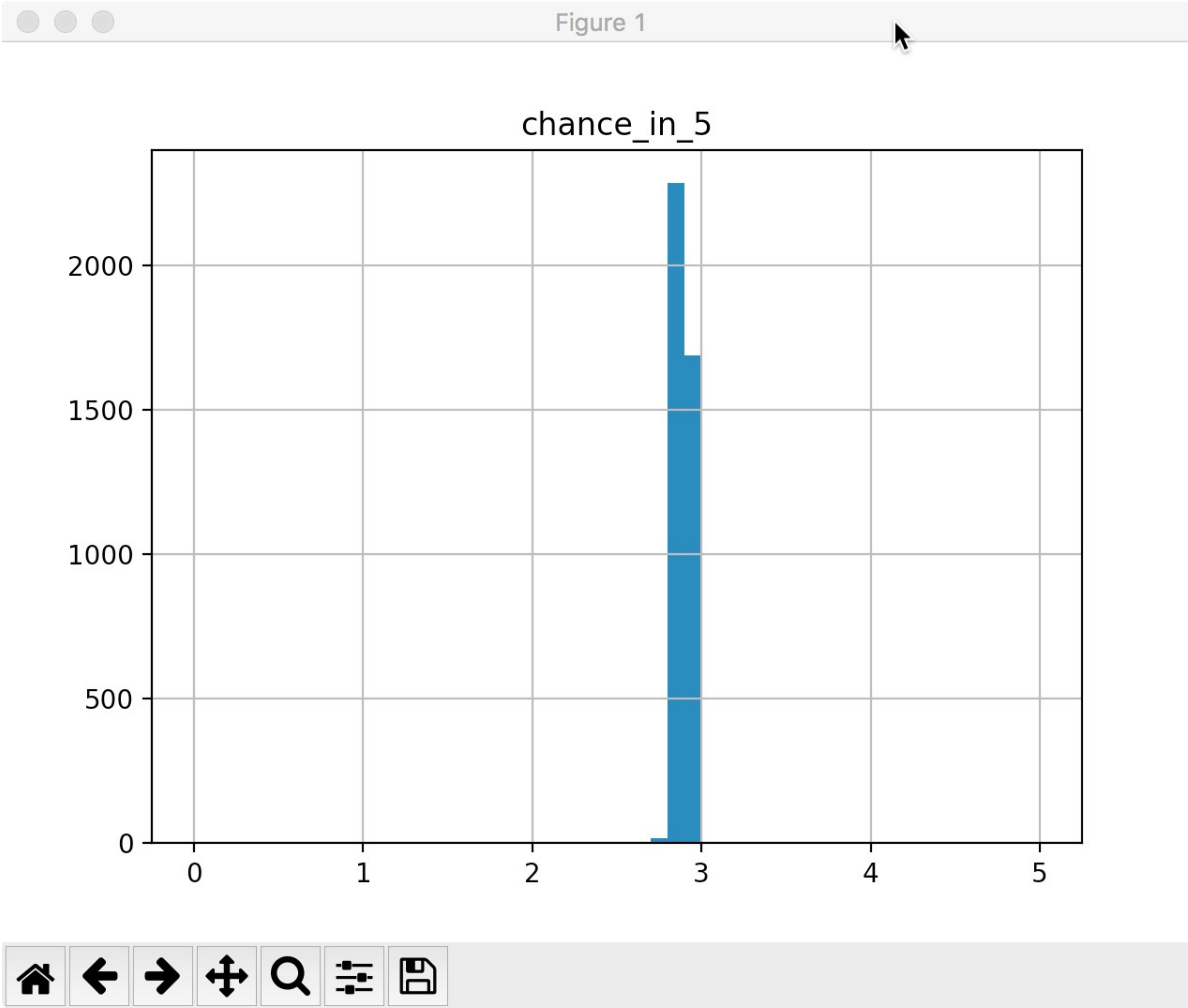
- Some AI systems will give you a probability for a prediction.
  - Our program reports a median value as the most likely value
- How sure is the system of that value?
- Watch what happens as we increase data
  - Median values remain roughly the same



$ python puttBetScale.py stan/mechanistic_golf_shrinkage.stan 5 .01 3



$ python puttBetScale.py stan/mechanistic_golf_shrinkage.stan 5 .1 10



$ python puttBetScale.py stan/mechanistic_golf_shrinkage.stan 5 1 19

Compare 5%-95% intervals

name	Mean	5%	50%	95%
chance_in_5 .01 3	2.823270	2.151960	2.824790	3.454040
chance_in_5 .1 10	2.895170	2.711690	2.894230	3.075980
chance_in_5 1 19	2.890480	2.833360	2.890500	2.948710

Less Data Means Less Certainty

- Is there a difference between 95% of posterior samples that range from:
  - 2.1 and 3.4?
  - 2.8 and 2.9?
- The house insists on even bets and $1 max
- 10,000 $1 bets
  - makes $ if the true chance_in_5 is 2.8
  - looses $ if the true chance_in_5 is 2.3

Working with the posterior

- The posterior is the combination of prior information with data via a likelihood.
- Stan programs do not produce posteriors
- Stan programs draw samples from posteriors.
- chance_in_5 will have around 4,000 samples
  - Each one a draw from the posterior
  - No probabilities yet

Values for chance_in_5

```
  [1] 4.983672319 0.504675345 0.956908493 2.875748848 0.997364239 0.417071450 4.501827071 0.936049387 0.863498351
 [10] 4.153279253 2.060605266 1.139810114 3.779605672 1.658594148 1.067616110 4.116676326 2.330651455 3.967579063
 [19] 1.842490519 0.455000172 1.157479467 2.752047595 0.915102101 4.982286286 4.391265367 4.992451716 2.447138984
 [28] 2.205051616 4.231899958 0.812689175 3.989032044 0.796250061 0.859436737 1.028724611 1.924055572 3.208221532
 [37] 2.210415693 2.136084733 1.248423340 3.315607704 2.254323347 4.784930724 4.885145022 1.982817767 4.984226181
 [46] 0.048460657 0.282379449 3.722942450 0.999613762 2.502718692 3.243868759 0.822543222 0.211249991 1.147694845
 [55] 4.896531799 4.199396308 0.665606386 3.141129394 2.607873759 4.690558455 4.696524164 4.031962569 2.755539678
 [64] 1.283222482 2.248947510 0.319166050 0.636175197 1.570415334 2.837057149 2.539923313 3.181580786 1.233043905
 [73] 3.061787637 4.764964321 0.918676119 0.480524564 1.491754639 3.321344687 0.492543512 2.089836199 4.983904438
 [82] 1.547706472 2.414447829 0.282179524 1.666381844 0.367431461 4.361380324 3.373313419 4.397065423 1.689771086
 [91] 1.141912694 2.459350776 4.159030935 3.116608167 3.076992472 0.823561946 2.969654653 1.165449756 1.658783029
[100] 0.351305357 1.694081739 1.259235890 1.387418442 2.528962612 4.852489185 1.374381734 2.941864848 1.157997846
[109] 0.168440287 4.373520019 3.886488608 1.093028201 4.987565650 4.812540735 4.595358682 0.655243261 2.834453881
[118] 2.728336762 1.977742924 2.179817460 0.320485894 3.711874986 0.189774115 2.125163616 2.366089869 2.752047595
[127] 2.270779435 2.060605266 0.497764905 1.841243712 0.039907438 3.040357791 2.607759999 1.721114910 2.466011311
...
```

Histograms

- End goal is to figure out probability for values of 'chance_in_5'.
- We bin 10 ways, 0-0.499, .5-.999, ...4.5-5
  - Any exact value is unlikely to be found in samples
  - Bins are human interpretable
- P(chance_in_5 < 2.8) = 50%
  - count(values < 2.8) / count(all values)
  - Look at values
    - `$ less puttbet-1.csv`

A tour of puttBet.py

```
import os
from cmdstanpy import cmdstan_path, CmdStanModel
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import fileinput
import sys

stan_program_path = sys.argv[1]
distance_of_putt = float(sys.argv[2])

stan_program = CmdStanModel(stan_file=stan_program_path)
stan_program.compile()

json_data = data= {"distance_of_putt":distance_of_putt}

fit = stan_program.sample(data=json_data,
                          csv_basename='./puttbet')
print(fit.summary())

fit.get_drawset(params=['pred_ch_in_5']).hist(bins=50,range=(0,5))

plt.show()
```

A tour of stan/mechanistic_golf.stan

```
data {
  real distance_of_putt;
}
transformed data {
  int J = 19;
  int x_distance[J] = {2,3,4,5,6,7,8,9,10,11,12,13,14...};
  int y_successes[J] = {1346,577,337,208,149,136,111...};
  int n_attempts[J] = {1443,694,455,353,272,256,240...};
  real r = (1.68/2)/12;
  real R = (4.25/2)/12;
  real threshold_angle[J];
  for (i in 1:J) {
    threshold_angle[i] = asin((R-r)/x_distance[i]);
  }
}
parameters {
  real<lower=0> sigma_error_in_radians;
}
model {
  for (i in 1:J) {
    real prob = 2*Phi(threshold_angle[i]/sigma_error_in_radians) - 1;
    y_successes[i] ~ binomial(n_attempts[i], prob);
  }
}
generated quantities {
  real sigma_error_in_degrees = (180/pi())*sigma_error_in_radians;
  real pred_ch_in_5;
  real threshold_angle_for_distance = asin((R-r)/distance_of_putt);
  pred_ch_in_5 =
     (2*Phi(threshold_angle_for_distance/sigma_error_in_radians) - 1) * 5;
}
```

Summarizing

- We know Stan inputs
  - Data
  - Model
- We know Stan outputs
  - Posterior draws
- We see roughly how they connect
  - puttBet.py
  - mechanistic_golf.stan
- Learned that size of data matters

Next

- Cover detailed mechanics of compiling/running Stan programs
- Start building PuttBet app from zero

Compiling with CmdStan

- `cd <path>/cmdstan-2.21.0`
- `mkdir tmp`
- edit/save tmp/test.stan

```
parameters {
  real <lower=0, upper = 5> chance_in_5;
}
model {}
```

- `make tmp/test`
- `ls tmp`

```
test    test.hpp  test.o    test.stan
```

Running a stan program

```
$ tmp/test sample

method = sample (Default)
  sample
    num_samples = 1000 (Default)
    num_warmup = 1000 (Default)
    …
Iteration: 1900 / 2000 [ 95%]  (Sampling)
Iteration: 2000 / 2000 [100%]  (Sampling)

 Elapsed Time: 0.010488 seconds (Warm-up)
               0.035355 seconds (Sampling)
               0.045843 seconds (Total)
```

Running a Stan program

```
$ less output.csv
….
lp__,accept_stat__,stepsize__,treedepth__,n_leapfrog__,divergent__,energy__,chance_in_5
# Adaptation terminated
# Step size = 0.947331
# Diagonal elements of inverse mass matrix:
# 2.56888
0.195175,0.999576,0.947331,1,3,0,0.0821981,2.91519
0.125106,0.980979,0.947331,1,1,0,-0.124422,3.26398
-0.367036,1,0.947331,1,1,0,0.803274,4.16915
0.153601,0.993894,0.947331,1,3,0,0.584455,1.85203
0.216061,1,0.947331,2,3,0,-0.163841,2.71003
0.220674,0.999161,0.947331,1,3,0,-0.211056,2.37584
0.0979697,0.964437,0.947331,1,3,0,-0.0598895,3.35753
0.0966662,0.999669,0.947331,1,1,0,-0.0630794,3.36171
0.0966662,0.624582,0.947331,1,3,0,1.44813,3.36171
-1.16498,0.724709,0.947331,1,3,0,1.34801,4.66572
-0.429226,1,0.947331,2,3,0,1.07796,4.23059
-0.942596,0.937841,0.947331,1,1,0,0.960438,4.57411
-0.703981,1,0.947331,1,1,0,1.01248,4.44343
-1.55183,0.92825,0.947331,1,1,0,1.55254,4.77831
...
```

Summarizing Output

```
$ bin/stansummary output.csv
Inference for Stan model: test_model
1 chains: each with iter=(1000); warmup=(0); thin=(1); 1000 iterations saved.

Warmup took (0.010) seconds, 0.010 seconds total
Sampling took (0.035) seconds, 0.035 seconds total

                 Mean     MCSE   StdDev     5%       50%   95%    N_Eff  N_Eff/s    R_hat
lp__            -0.30  3.6e-02  6.8e-01   -1.7  -7.5e-02  0.22  3.6e+02  1.0e+04  1.0e+00
accept_stat__    0.93  3.2e-03  1.1e-01   0.68   9.8e-01   1.0  1.2e+03  3.4e+04  1.0e+00
stepsize__       0.95      nan  2.9e-15   0.95   9.5e-01  0.95      nan      nan      nan
treedepth__       1.4  2.1e-02  5.6e-01    1.0   1.0e+00   2.0  6.7e+02  1.9e+04  1.0e+00
n_leapfrog__      2.7  5.4e-02  1.4e+00    1.0   3.0e+00   7.0  7.0e+02  2.0e+04  1.0e+00
divergent__      0.00      nan  0.0e+00   0.00   0.0e+00  0.00      nan      nan      nan
energy__         0.79  4.9e-02  9.5e-01  -0.16   5.3e-01   2.7  3.7e+02  1.1e+04  1.0e+00
chance_in_5       2.5  7.7e-02  1.4e+00   0.34   2.5e+00   4.7  3.3e+02  9.4e+03  1.0e+00

Samples were drawn using hmc with nuts.
For each parameter, N_Eff is a crude measure of effective sample size,
and R_hat is the potential scale reduction factor on split chains (at
convergence, R_hat=1).
```

Compile/Run from Python

- `$ cd <path>/StanIsThePlanDist`
- `$ python rv.py stan/test.stan`
- `$ python rv.py stan/test.stan chance_in_5`
- `$ python rv.py stan/test.stan chance_in_5 cat`

Outline: Model Authoring for Beginners

- Pick the simplest parameters that make sense
  - Betting/Putting app: Sink or miss.
- Eliminate the impossible (prior knowledge)
  - A persons height can't be negative or > 10 ft
- Find a way to incorporate data into priors (likelihood) that turns into a posterior
- Have a way to decide if the posterior is useful

An Important Philosophical Point

- Can't model world for detail and scope reasons
  - Detail: Computationally too complex & don't have theory
  - Scope: Computationally too big & don't have data
- We can approximate however by averaging things out
  - We approximate physics
  - We limit how much we look at
- This gets us uncertainty
  - Instead of 0 or 1, we say 0.01 or .5 or .99

Uncertainty

- .99 (99%) probability means that 99 times out of 100 we get X, 1 time we don't....over time....
- Often abused, a system will claim .99 but it is not.
- Difficult to separate poor calibration from uncertainty.
  - We don't get HTHT on coin flips all the time
  - Over large amounts of data we should get .5 H
  - We should also get HHHHH with enough data

1st Revision to our model

- Is PuttBet modeling the entire universe?
- Does PuttBet have access to the future?
- The PuttBet app will now return 'chance_in_5' to reflect our uncertainty.
  - 0 chance in 5 is 0% probability
  - 5 chance in 5 is 100%
  - 2.5 chance in 5 is 50%
- chance_in_5 keeps us from discussing probabilities of probabilities
- Instead, probability of a 'chance_in_5'

Playing with parameter scales

- Instead of 0 to 5 we use A-E
  - A=0-.999
  - B=1-1.999
  - C=2-2.999
  - D=3-3.999
  - E=4-5
- What is probability of E?

Code model up in Stan

Look at stan/no_putt.stan or just type into an editor

```
parameters {
        real <lower=0, upper=5> chance_in_5;
}

model {

}
```

Save as 'stan/no_putt.stan'

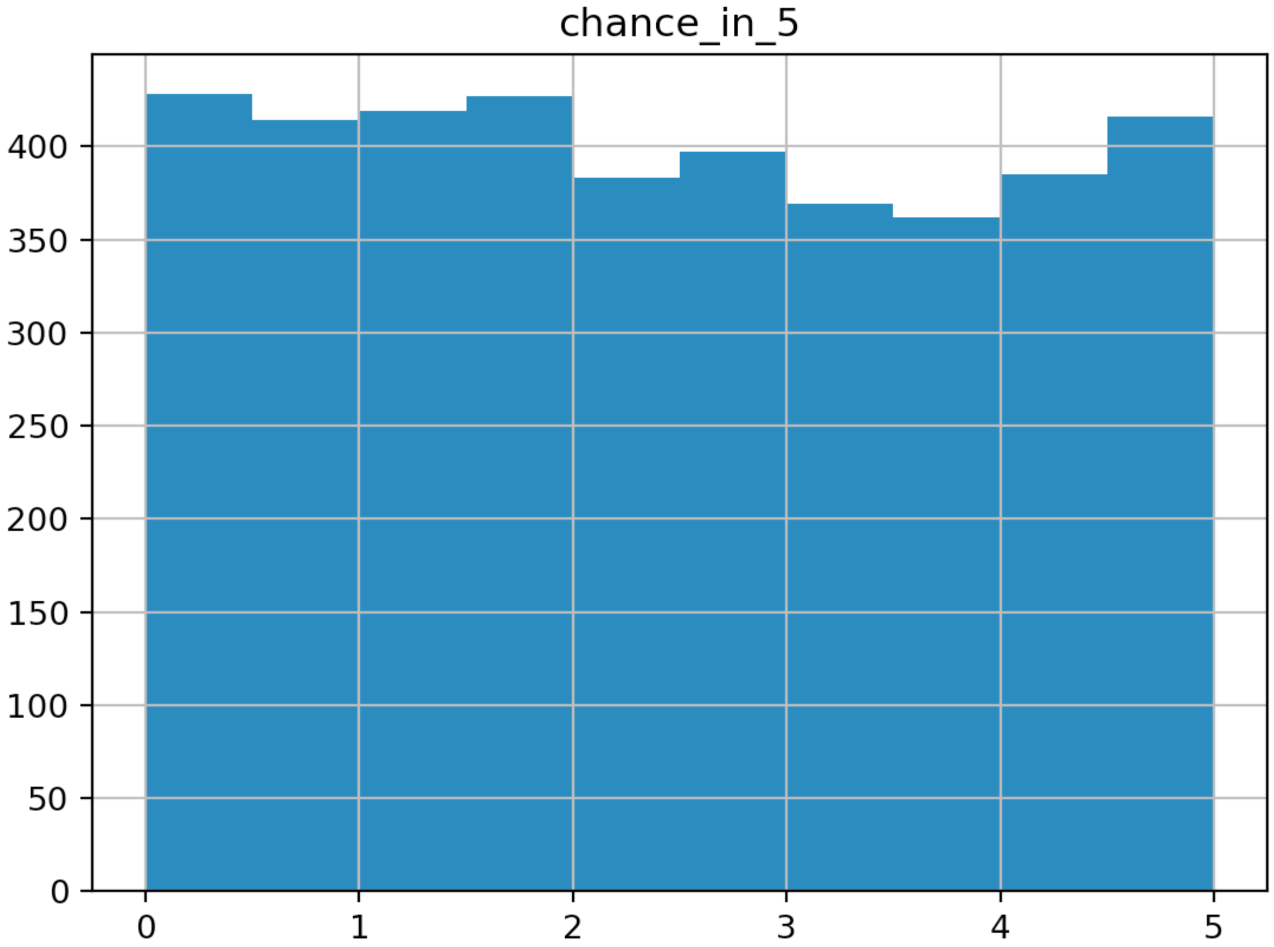
```
$ cd <path>/StanIsThePlanDist
$ python rv.py stan/no_putt.stan chance_in_5
```

Inspect output-1.csv

```
>less output-1.csv
lp__,accept_stat__,...,divergent__,energy__, chance_in_5
-0.788662,0.834613,1.12129,1,1,0,0.799103,   4.49443
-0.34793,0.967041,1.12129,2,3,0,1.39449,     0.850982
-1.74739,0.802025,1.12129,2,3,0,1.78157,     4.81924
-1.74739,0.963619,1.12129,1,1,0,2.66079,     4.81924
…
0.171325,1,1.12129,1,1,0,-0.14552,           3.0618
0.0610676,0.958562,1.12129,1,1,0,-0.048304,  3.46703
-0.56155,0.829213,1.12129,1,3,0,0.878503,    0.656531
```

Histogram of 'chance_in_5'

```
$ python rv.py stan/no_putt.stan chance_in_5
```



Welcome to the Uniform Distribution

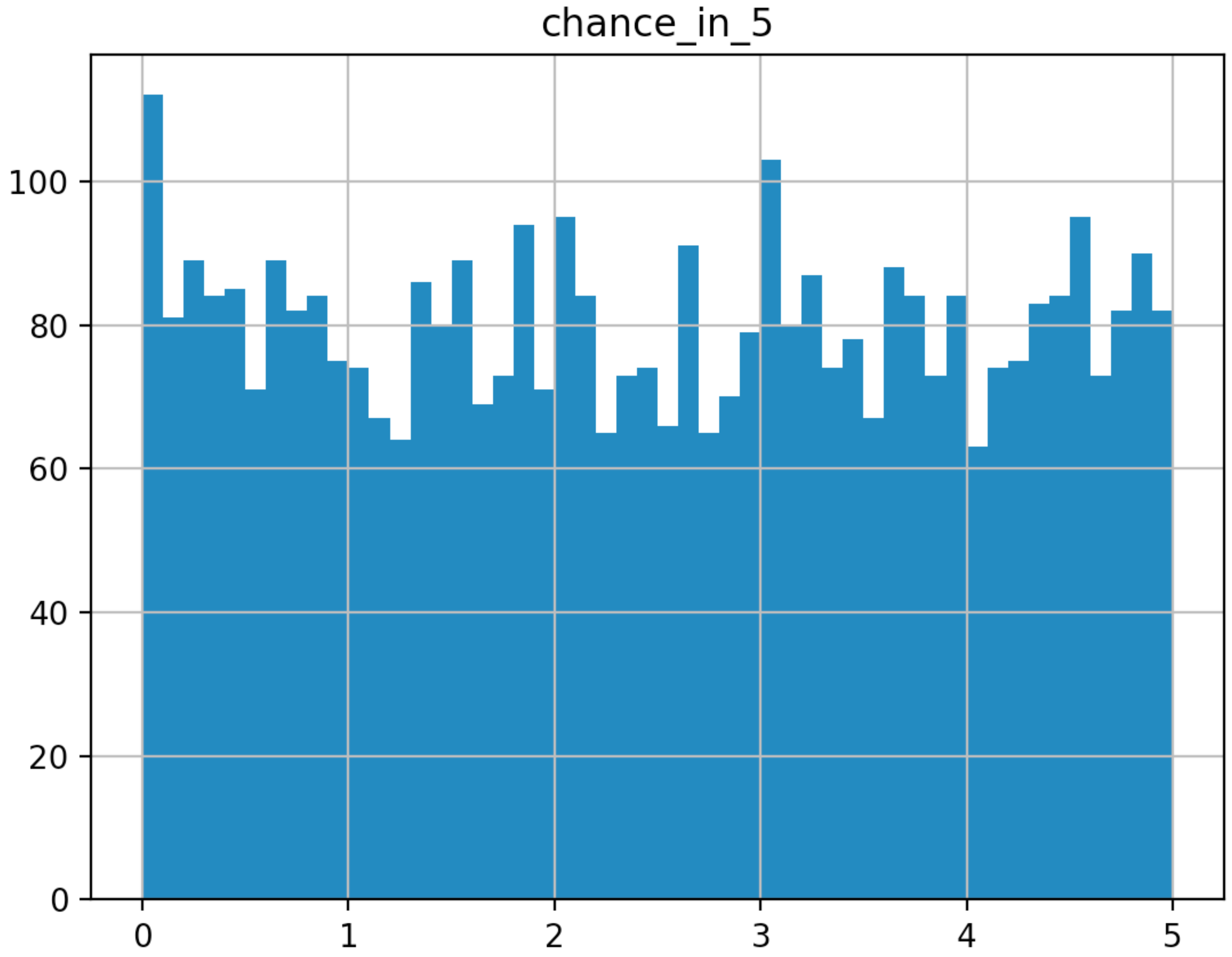
- Given the uniform distribution draws above:
  - What are some ranges of 'chance_in_5' that have 50% probability?
  - Is any value of the distribution more likely than any other?
  - Name some phenomenon and the relative parameter scale that is uniformly distributed.
  - Name some phenomenon that are not uniformly distributed.
  - The mean parameter value is 2.5, is that useful?

Messing with Distributions

```
$ python rv.py stan/uniform_uniform.stan chance_in_5
parameters {
      real <lower=0, upper = 5> chance_in_5;
}
model {
      chance_in_5 ~ uniform(0,5);
}
```

	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	N_Eff/s	R_hat
name									
lp__	-0.379682	0.022752	0.840617	-2.094090	-0.043359	0.221461	1365.13	5689.78	0.999403
chance_in_5	2.513630	0.036922	1.427840	0.227396	2.461970	4.743070	1495.51	6233.22	1.002910

Uniform x Uniform

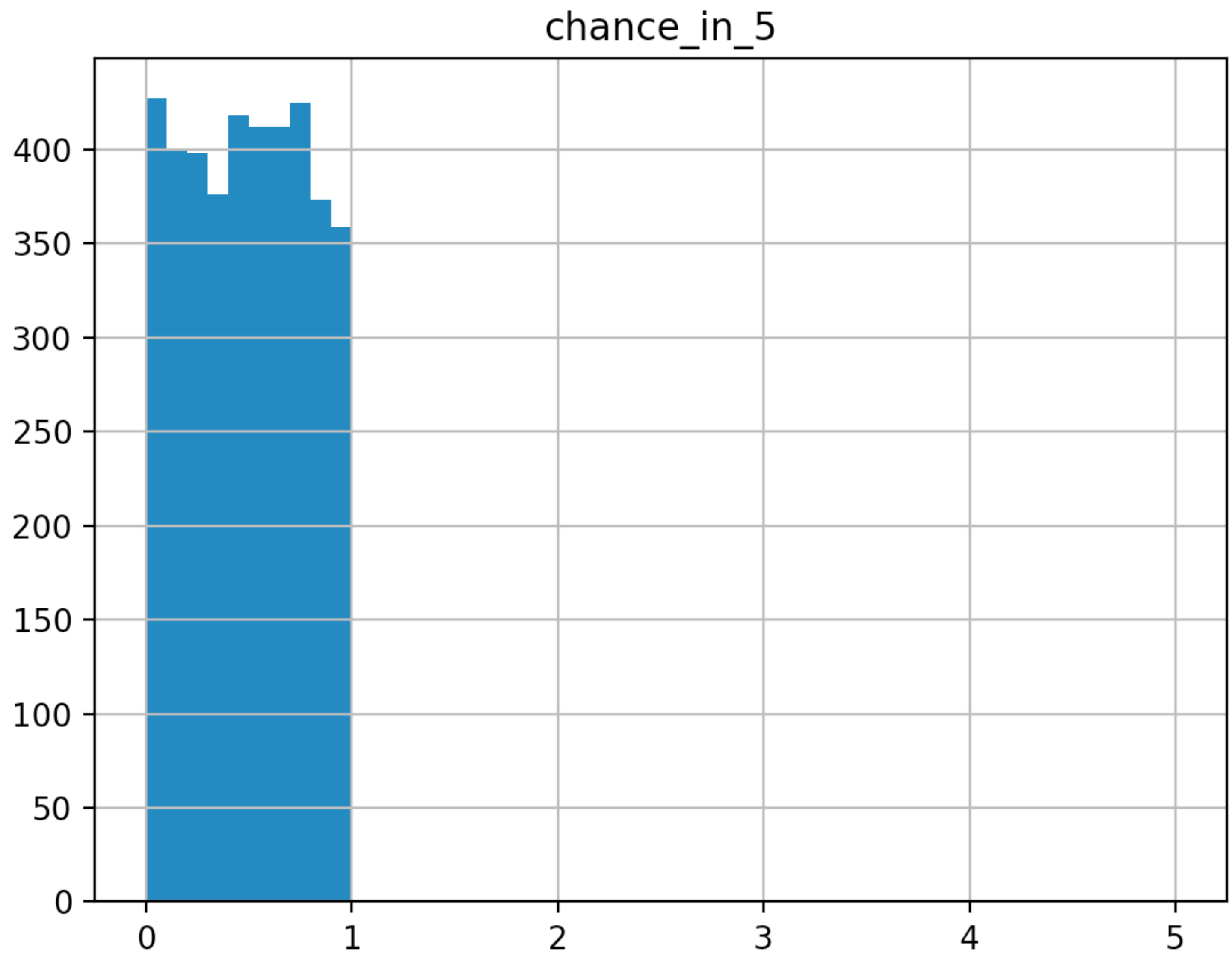


Uniform Haircut

```
parameters {
   real <lower=0, upper = 5> chance_in_5;
}


model {
  chance_in_5 ~ uniform(0,1);
}
> python rv.py stan/targ_unif_unif.stan
chance_in_5
```

uniform(0,5) x uniform(0,1)



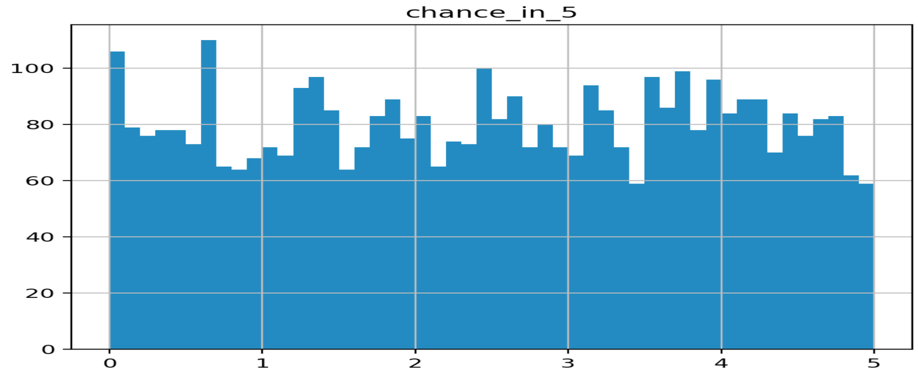
Fix by scaling

```
parameters {
      real <lower=0, upper = 5> chance_in_5;
}

transformed parameters {
  real chance_in_1 = chance_in_5/5;
}

model {
  chance_in_1 ~ uniform(0,1);
}

$ python rv.py stan/scaled_uniform_uniform.stan
chance_in_5
```



Adding some data to our model

Putt went in = 1, putt did not = 0

```
$ python rv.py stan/one_putt.stan chance_in_5

parameters {
   real<lower=0, upper=5> chance_in_5;
}

transformed parameters {
   real chance_in_1 = chance_in_5/5;
}

model {
  chance_in_1 ~ uniform(0,1);
  1 ~ bernoulli(chance_in_1);
}
```

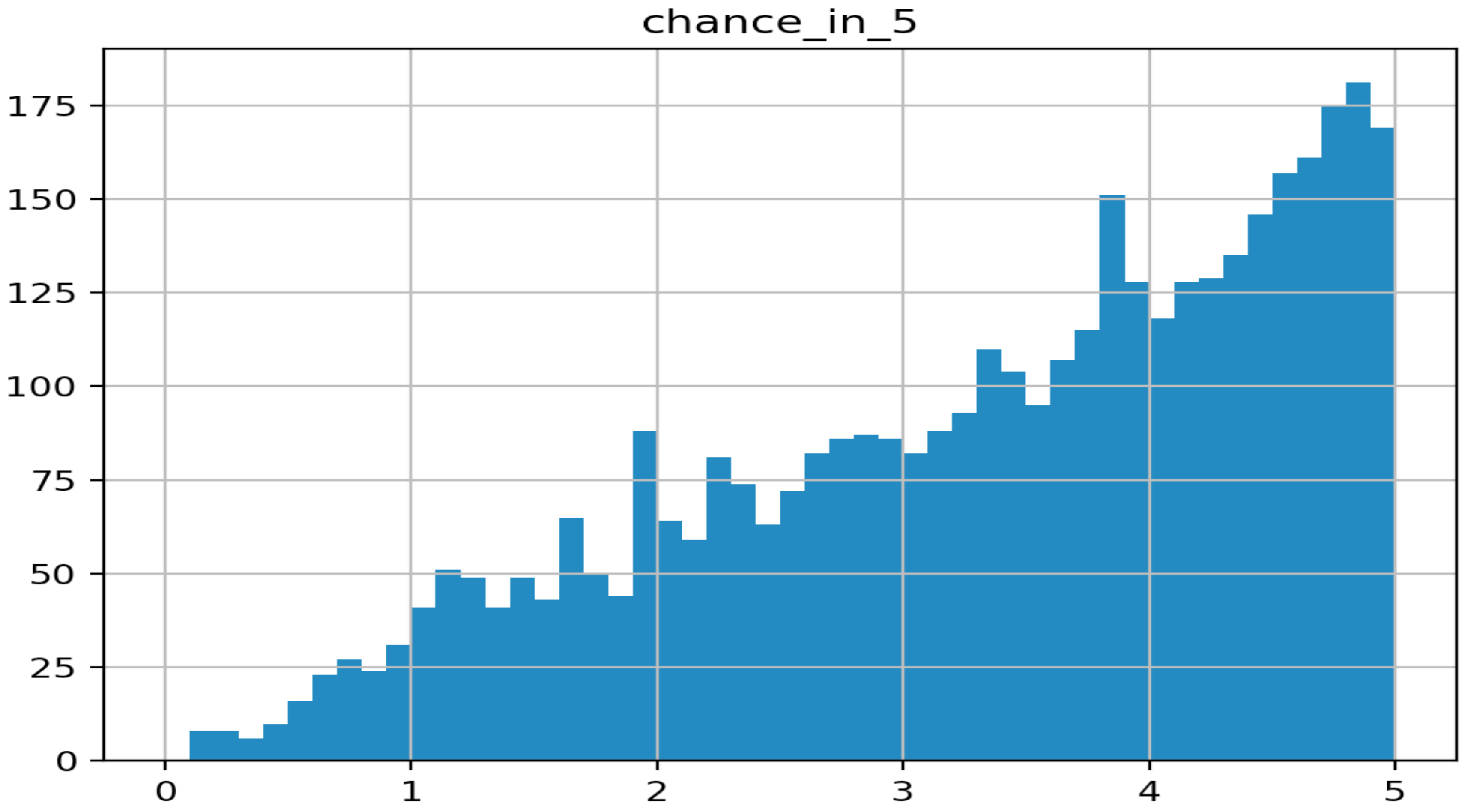
Running the model

```
> python rv.py stan/one_putt.stan chance_in_5

                 Mean      MCSE    StdDev       5%       50%       95%    N_Eff  N_Eff/s    R_hat
name
lp__        -0.890663  0.023695  0.784950 -2.54812 -0.592286 -0.303383  1097.41   6401.1  1.00204
chance_in_5  3.339720  0.031065  1.184030  1.13355  3.560120  4.876400  1452.69   8473.4  1.00132
chance_in_1  0.667945  0.006213  0.236806  0.22671  0.712025  0.975280  1452.69   8473.4  1.00132
```

Running the model

```
> python rv.py stan/one_putt.stan chance_in_5
```



What happened?

- Understand the Bernoulli distribution
- Expose the implicit loop around blocks
- Give the intuition around what the target() does

What is this Bernoulli thing?

- Putt goes in: 1
  - `1 ~ bernoulli(.99) = .99`
  - `1 ~ bernoulli(.1) = .1`
  - `1 ~ bernoulli(.5) = .5`

- Putt does not go in: 0
  - `0 ~ bernoulli(.99) = .1`
  - `0 ~ bernoulli(.1) = .99`
  - `0 ~ bernoulli(.5) = .5`

breck_noulli function

```
> python rv.py stan/breck_noulli.stan theta

functions {

  real breck_noulli_lpmf(int zero_or_one,
                         real param_to_return_prob_of) {
    if (zero_or_one == 1) {
      return log(param_to_return_prob_of);
    }
    return log(1.0-param_to_return_prob_of);
  }
}

parameters {
  real<lower=0,upper=1> theta;
}

model {
  1 ~ breck_noulli(theta);
}
```

Simplified (Wrong) Evaluation

1. From last iteration prev_target = .3
2. Propose a random value for chance_in_5 = 2:
3. Rescale to chance_in_1 = .4
4. Set target to 1
5. target = target*(1 ~ breck_noiulli(chance_in_1))
6. target = 1*(.4) = .4
7. If target > prev_target, accept proposal.
8. If target < prev_target
   1. accept proposal at 1/target rate, else keep last target().

How did Bernoulli change the posterior?

- exp(prev_target)= .5, values of params recorded.
- `exp(target()) = 1`
- `chance_in_1 = .4`
- exp(target()) * exp(uniform_lpdf(.4,1,0)) = 1
- exp(target()) * exp(bernoulli_lpmf(1|.4)) = .4
- .4/.5 < 1 so we accept 40% of the time
- 40% of .4 param survive, they reduce in histogram count.

```
generated quantities {
  real chance_in_5[250];
  real sigma_step_size = .5;
  real my_target[250];
  chance_in_5[1] = 2.5;
  my_target[1] = log(chance_in_5[1]);

   for (i in 2:250) {
    real proposal = chance_in_5[i-1] + normal_rng(0,sigma_step_size);
    real chance_in_1;
    real random_value = uniform_rng(0,1);
    my_target[i] = 0;
    proposal = fmax(0.0,proposal);
    proposal = fmin(5.0,proposal);
    chance_in_1 = proposal/5;
    my_target[i] += bernoulli_lpmf(1|chance_in_1);
    if (exp(my_target[i])/
        exp(my_target[i-1]) > random_value) {
      chance_in_5[i] = proposal;
    }
    else {
      chance_in_5[i] = chance_in_5[i-1];
    }
  }
  print("Chance_In_5: ",chance_in_5);
  print("Mean: ", mean(chance_in_5));
}
$ python run_met_hastings.py
```

Summary Section 1

- We know the basics of how to run a Stan program with CmdStanPy.
- Setup basic app, PuttData
- Have very simple program with one data point
- Next we will improve our model by
  - Adding data
  - Modifying the model

Homework

- Add more data to model
- Add another parameter to model
- Try different distribution in place of bernoulli
  - 1 ~ normal(a,.5)
  - 1 ~ normal(3,a)
  - a ~ normal(3,.5)
  - 1 ~ exponential(a)
- Change the number of bins in the histogram