Probabilistic Programming in Private
Part 1

Overview

Introduction

To be able to analyse data robustly and to protect participants' privacy, we are going to employ probabilistic programming using the Private language. Understanding probabilistic models is not easy at first, but pays off in the long run. So make sure to ask questions on the discussion board if there is anything you are having trouble with or struggling to conceptualise. I guarantee there will be others who will have similar questions and may initially be equally confused about some of the concepts presented here.

Learning Objectives

- Build simple Bayesian models in the Private language
- Generate fake (or dummy) data from Bernoulli and Binomial distributions
- Estimate the parameters of Bernoulli, Binomial and Normal variables
- Estimate the parameters of a linear regression model.

There are also many online materials that can help to get you started with Bayesian statistics. Some good ones are:

- **Introduction to Bayesian Data Analysis: Part 1: What is Bayes?:**
  https://www.youtube.com/watch?v=3OJEae7Qb_o
  Duration: 29:29
  Take away information: *What* Bayesian data analysis is

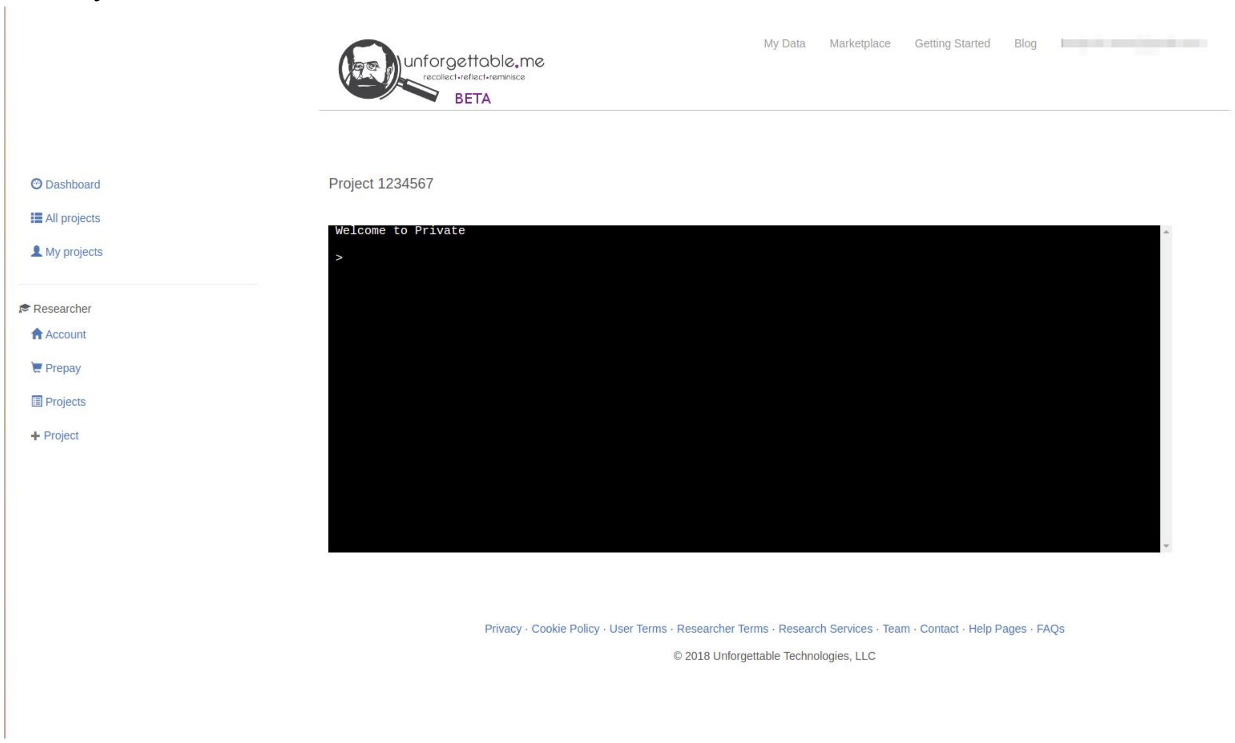
- **Introduction to Bayesian Data Analysis: Part 2: Why use Bayes?**
  https://www.youtube.com/watch?v=mAUwjSo5TJE
  Duration: 22:59
  Take away information: *Why* Bayesian data analysis is useful.

- **Tiny Data, Approximate Bayesian Computation and the Socks of Karl Broman**
  https://www.youtube.com/watch?v=nKCT-Cdk0xY
  Duration: 19:40
  Take away information: Introduction to Approximate Bayesian Computation

- **Introduction to Bayesian statistics, part 1: The basic concepts**
  https://www.youtube.com/watch?v=0F0QoMCSKJ4
  Duration: 9:11
  Take away information: Understanding the prior distribution, the likelihood distribution and posterior distribution.

Accessing Private to complete the exercises – Instructions

The Private programming language can be accessed through www.unforgettable.me. If you have an account, login and navigate to the marketplace and to the demo project. There you will see a screen a screen that looks like this:



unforgettable.me
recollect•reflect•reminisce
BETA
My Data
Marketplace
Getting Started
Blog
[redacted]
Dashboard
All projects
My projects
Researcher
Account
Prepay
Projects
Project
Project 1234567
Privacy · Cookie Policy · User Terms · Researcher Terms · Research Services · Team · Contact · Help Pages · FAQs
© 2018 Unforgettable Technologies, LLC

```
Welcome to Private

>
```

Figure 1: Unforgettable.me website with console window

This terminal provides access to the Private programming language that we will be using during the course. If you chose not to sign up for an account, then you can access private using the following link https://private.mall-lab.com/. This is a site which contains a terminal window which has the Private programming language installed. Fake data has been loaded into this console which can be used for practicing analysis.

When going through this tutorial, it will be helpful to have two internet browser windows open side by side. One window will be for LMS, with this tutorial open in this window. A second window can have unforgettable.me open to the Private shell. (Although not recommended, the other workflow option is to have two tabs open and switch between the two tabs).

As you work through this tutorial, type the code in yourself (you can copy and paste in the datasets, eg flips = [1, 0, 1, 0, 0, 0, 0, 1, 0, 0], but be sure to manually type in all of the commands). Just reading code passively is not an effective learning strategy, as you may fail to notice aspects of the language and will not be presented with the error messages when you make a mistake. At the bottom of the tutorial, you will find a list of useful Private commands and operators, as well as an explanation of the messages you may see when programming in Private. Also, make sure you work through the solutions yourself before you check the provided solutions.

Estimating the bias of a coin

We are going to begin our exploration of the Private language by considering how one would go about estimating the degree of bias in a coin. If a coin is fair, it has a 0.5 probability of landing on heads and a 0.5 probability of landing on tails. However, it is also possible that we have been handed a 'trick coin' which is weighted to favour either heads or tails. In this exercise, we are going to be estimating the probability of our (possibly) weighted coin landing on heads.

We'll start by entering our data. Let's assume that we flipped our coin 10 times and saw the following pattern:

Head, Tail, Head, Tail, Tail, Tail, Tail, Head, Tail, Tail

We are going to code the data using a one for the heads and a zero for the tails. To enter the data into Private we type:

```
flips = [1, 0, 1, 0, 0, 0, 0, 1, 0, 0]
```

So, we have the data (3 heads and 7 tails) that we are going to model. Now we want to define the Bayesian model that we will use to estimate the rate parameter.

To define a Bayesian model, we need to specify the likelihood of the data given the parameters and the priors of the parameters. We are going to assume that the model that generated the data is a Bernoulli distribution. The Bernoulli distribution describes a random variable that has two possible outcomes given a parameter that determines the probability of each. So, we can go ahead and define this as follows:

```
flips ~ Bernoulli(r)
```

where r is the unknown rate variable. Note we have used the "~" character here rather than the "=". That is because we are providing the probabilistic definition of the variable now - rather than setting its value. We are saying we believe flips is a Bernoulli random variable.

Before we can estimate r, we also need to specify the prior. We know that r is a probability, so it has to lie between 0 and 1, but we don't know anything else about it, so we will define it as a uniform distribution as follows:

```
r ~ Uniform(0,1)
```

To see all the variables, type:

```
sv
```

sv stands for "show variables."

```
flips = [1, 0, 1, 0, 0, 0, 0, 1, 0, 0]   [1, 0, 1, 0, 0, 0, ...]
flips ~ Bernoulli(r)                      [1, 0, 1, 0, 0, 0, ...]
r ~ Uniform(0, 1)                         Computing
```

You can now see everything you have defined so far. On the left hand side is the code that you typed in and on the right hand side are the values of the variables. When interacting with the shells of languages like python or R, the values of the variables are retained, but the code that generated those values is discarded (which is why a separate code editor is necessary). In Private, the code is retained, so that it can be automatically run again in the future if necessary. In the first line, the set of values on the left hand side is there because they were included in the code. They are repeated on the right hand side because that is the value of the variable flips.

Note r is now computing because all definitions have been provided, but the computer is still generating posterior samples of the random variable r. This means that the computer is generating many estimates of the true rate, given what it knows from the data, the likelihood, and the prior. There is some randomness in the generation of these estimates, but more likely estimates will be generated more often.

While we wait for the computer to finish, we can define some other variables that will be useful. Let's compute the mean and the standard deviation of r as follows:

```
meanr = mean(r)
stdr = std(r)
```

Now type:

```
sv
```

If the sampler has finished you'll see something like the following:

```
flips = [1, 0, 1, 0, 0, 0, 0, 1, 0, 0]   [1, 0, 1, 0, 0, 0, ...]
meanr = mean(r)                           0.32523311603796623
stdr = std(r)                             0.13683603635844127
flips ~ Bernoulli(r)                      [1, 0, 1, 0, 0, 0, ...]
r ~ Uniform(0, 1)                         [0.519417 ... 0.350927]
```

Inside the brackets next to r are all of the samples that have been generated. It is truncated due to space requirements, so only the first and last values are displayed. Because the sampler uses a random number generator to calculate the samples, there may be some differences in your samples if you were to start again and run the same analysis, but they should be very similar. If you want to see all of the samples type the name of the variable:

```
r
```

And you should see:

```
[0.519 0.392 0.299 0.363 0.34  0.34  0.34  0.215 0.44  0.314 0.192 0.474
 0.393 0.595 0.544 0.449 0.404 0.364 0.077 0.347 0.454 0.357 0.646 0.292
 0.45  0.321 0.282 0.557 0.47  0.732 0.443 0.491 0.178 0.158 0.158 0.23
 0.23  0.207 0.276 0.465 0.461 0.29  0.247 0.282 0.282 0.18  0.274 0.528
 0.138 0.199 0.278 0.492 0.706 0.221 0.228 0.41  0.515 0.328 0.331 0.356
 0.218 0.526 0.265 0.265 0.287 0.251 0.161 0.194 0.422 0.385 0.372 0.431
 0.474 0.329 0.312 0.272 0.259 0.392 0.336 0.225 0.488 0.262 0.231 0.329
 0.429 0.536 0.537 0.408 0.414 0.334 0.35  0.35  0.369 0.134 0.171 0.11
 0.702 0.174 0.263 0.251 0.204 0.428 0.453 0.315 0.395 0.07  0.134 0.265
 0.235 0.285 0.288 0.278 0.241 0.308 0.257 0.515 0.255 0.406 0.45  0.693
 0.358 0.363 0.411 0.353 0.385 0.459 0.359 0.37  0.396 0.314 0.171 0.311
 0.316 0.344 0.368 0.398 0.375 0.157 0.166 0.159 0.189 0.128 0.221 0.192
 0.192 0.164 0.202 0.266 0.269 0.516 0.516 0.256 0.29  0.069 0.105 0.146
 0.322 0.322 0.453 0.511 0.476 0.239 0.211 0.236 0.254 0.286 0.298 0.383
 0.383 0.334 0.282 0.248 0.251 0.249 0.641 0.319 0.145 0.204 0.223 0.214
 0.37  0.361 0.39  0.296 0.387 0.314 0.43  0.393 0.393 0.646 0.646 0.468
 0.468 0.602 0.42  0.461 0.295 0.323 0.112 0.114 0.151 0.468 0.18  0.169
 0.557 0.168 0.254 0.398 0.278 0.143 0.173 0.119 0.405 0.405 0.459 0.224
 0.244 0.469 0.474 0.614 0.323 0.469 0.358 0.452 0.236 0.254 0.175 0.223
 0.22  0.262 0.287 0.16  0.54  0.144 0.505 0.496 0.37  0.446 0.388 0.286
 0.14  0.124 0.16  0.133 0.08  0.181 0.474 0.408 0.408 0.431 0.506 0.478
 0.467 0.437 0.249 0.229 0.448 0.36  0.426 0.482 0.482 0.482 0.217 0.15
 0.251 0.278 0.11  0.139 0.156 0.243 0.285 0.285 0.205 0.498 0.34  0.638
 0.559 0.289 0.289 0.218 0.109 0.183 0.196 0.194 0.175 0.544 0.72  0.09
 0.09  0.068 0.21  0.163 0.568 0.411 0.485 0.485 0.192 0.316 0.188 0.308
 0.244 0.316 0.316 0.316 0.139 0.212 0.365 0.306 0.217 0.125 0.471 0.344
 0.493 0.191 0.261 0.254 0.316 0.216 0.249 0.284 0.39  0.295 0.301 0.276
 0.357 0.426 0.461 0.501 0.25  0.256 0.132 0.181 0.535 0.272 0.316 0.31
 0.31  0.253 0.209 0.188 0.096 0.162 0.136 0.152 0.127 0.248 0.298 0.35
 0.324 0.287 0.287 0.386 0.5   0.202 0.215 0.261 0.261 0.113 0.047 0.065
 0.081 0.08  0.526 0.355 0.502 0.452 0.416 0.543 0.297 0.188 0.527 0.532
 0.409 0.503 0.324 0.206 0.246 0.178 0.541 0.561 0.382 0.43  0.416 0.416
 0.416 0.416 0.5   0.382 0.35  0.455 0.235 0.419 0.338 0.418 0.502 0.501
 0.41  0.323 0.351 0.351]
```

The samples of r are now available and meanr and stdr have been calculated. Note meanr is 0.325, which makes sense as there were 3 heads and 7 tails in our data, so we would expect the rate to be around 0.3. (Once again, meanr and stdr may be slightly different when you run the code, due the randomness in generating samples, but your result should be very similar)

To make it easier to try out different data sets, we are going to generate fake data with a known probability of the coin landing on heads, so that we can see how well we are able to recover the correct value. To generate 100 flips with a coin that has a probability of 0.3 type:

```
realr = 0.3
flips = Bernoulli(realr, 100)
```

We say the variable “flips” contains 100 samples of a Bernoulli variable with a rate parameter of 0.3.

To view the 100 samples that were created, type in the variable name - in this case flips:

```
flips
```

And you should see your 100 samples:

```
[1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 1 1 0
0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 1 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 1 1 1 0 1 0 0]
```

Now if you type sv you’ll see:

```
flips = Bernoulli(realr, 100)        [1.000000 ... 0.000000]
meanr = mean(r)                      0.3424256211042369
stdr = std(r)                        0.04404791606443669
realr = 0.3                          0.3
flips ~ Bernoulli(r)                 [1.000000 ... 0.000000]
r ~ Uniform(0, 1)                    [0.305225 ... 0.356029]
```

Note that meanr is close to the realr value that we used to generate the data.

One of the nice features of Private is that if we change the value of a variable then the other variables change accordingly. For instance, if we now type:

```
realr = 0.9
```

And then sv, we will get:

```
flips = Bernoulli(realr, 100)        [0.000000 ... 1.000000]
meanr = mean(r)                      0.9193716800844921
stdr = std(r)                        0.02647437986790655
realr = 0.9                          0.9
flips ~ Bernoulli(r)                 [0.000000 ... 1.000000]
r ~ Uniform(0, 1)                    [0.965137 ... 0.899771]
```

So again meanr is a reasonable approximation to the rate we used to generate the data.

You can now generate a histogram of the samples of r like this:

```
rplot = distplot(r)
```

If you show the variables using sv you will see:

```
flips = Bernoulli(realr, 100)  [0.000000 ... 1.000000]
meanr = mean(r)               0.9193716800844921
stdr = std(r)                 0.02647437986790655
realr = 0.9                   0.9
rplot = distplot(r)           <_io.Byt...c9edb8f0>
flips ~ Bernoulli(r)          [0.000000 ... 1.000000]
r ~ Uniform(0, 1)             [0.965137 ... 0.899771]
```

Notice that rplot appears as a variable (we don’t have to worry about the text inside the < >, this is just telling us where the plot has been stored on the computer) and you will also see the histogram of the values:

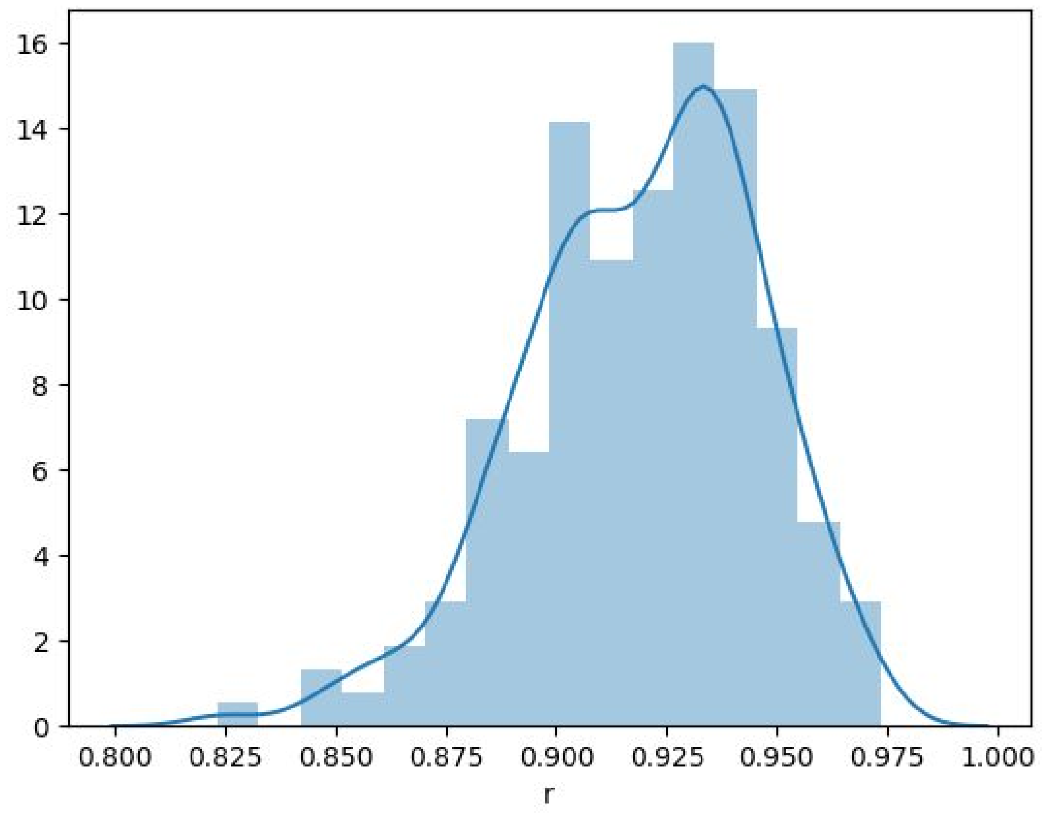


Figure 2: rplot plot generated by private

Most of the values of r are around 0.9. The blue line is called the kernel density estimator and is a smooth representation of the histogram.

If you change the data again

realr = 0.5

The values will change:

l = Bernoulli(realr, 100)	[1.000000 ... 1.000000]
stdr = std(r)	0.04965781015319354
realr = 0.5	0.5
meanr = mean(r)	0.5177240847858943
rplot = distplot(r)	<_io.Byt...eed434d0>
l ~ Bernoulli(r)	[1.000000 ... 1.000000]
r ~ Uniform(0, 1)	[0.584806 ... 0.525501]

and the plot will update automatically:

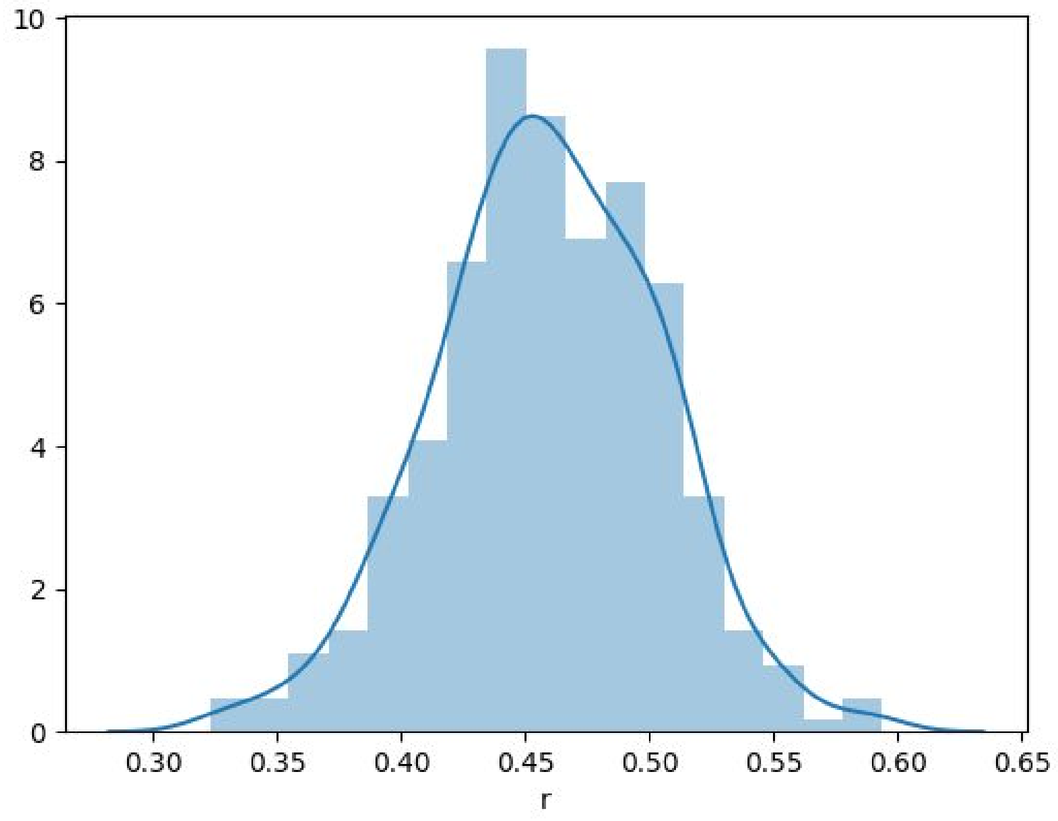


Figure 3: rplot plot updated for new variables and generated by private

Exercise 1: Standard deviations of the rate variable

Look at the standard deviations of the rate variable when the the real rate was 0.3, 0.5 and 0.9. Notice that it is greatest for 0.5 and smallest for 0.9. Why is this the case??

Solution for Exercise 1

When the true rate is .5, there are many likely estimates on both sides of 0.5 which the model needs to account for. When the rate is .9 however, the distribution will peak at .9, but because there is an upper bound of 1, there is a much smaller pool of possible values greater than .9 that the estimate can take, and while the distribution has a tail to the left, the smaller amount of variance to the right (greater than .9) will decrease the total variance. A rate of .3 is similar to .9, except it is capped by the lower bound of 0.

We have now generated code for a model that estimates the bias of the coin. We have also shown that the estimate of coin bias will change appropriately if we alter the data that we give to the model we have created.

Estimating the bias of multiple coins

Now that we have mastered the estimation of the rate of a single coin, lets spread our wings and consider the case of estimating the rate of multiple identical coins when we flip them all. To do this we are going to use the binomial distribution. The binomial distribution describes the probability of the number of heads given a rate parameter and the number of coins.

To begin remove all of the variables we set up for the first example:

```
clear
```

You should see:

```
All variables removed.
```

And sv should return nothing.

Again we are going to cheat (I mean facilitate our understanding) by using fake data for which we know the correct rate parameter. We can generate binomial values as follows:

```
NumberOfHeads = Binomial(10, 0.3, 100)
```

Which means flip 10 coins, weighted r=0.3 towards tails, 100 times. Now type:

```
sv
```

And you should see:

```
NumberOfHeads = Binomial(10, 0.3, 100)  [4.000000 ... 3.000000]
```

To see all of the values type:

```
NumberOfHeads
```

And you should see:

```
[4 4 3 2 4 3 2 2 3 2 6 3 6 1 2 3 3 4 3 3 4 2 3 4 2 4 4 4 1 4 3 4 2 2 2 2 3
 3 4 4 1 1 3 1 4 2 2 6 1 4 6 4 3 4 4 2 3 1 4 4 4 4 2 4 4 3 2 1 5 2 7 0 4 2
 4 4 1 2 1 2 3 2 4 3 3 2 3 2 3 4 0 5 3 4 5 1 4 2 4 3]
```

We can generate the histogram of the samples using distplot:

```
hplot = distplot(NumberOfHeads)
```

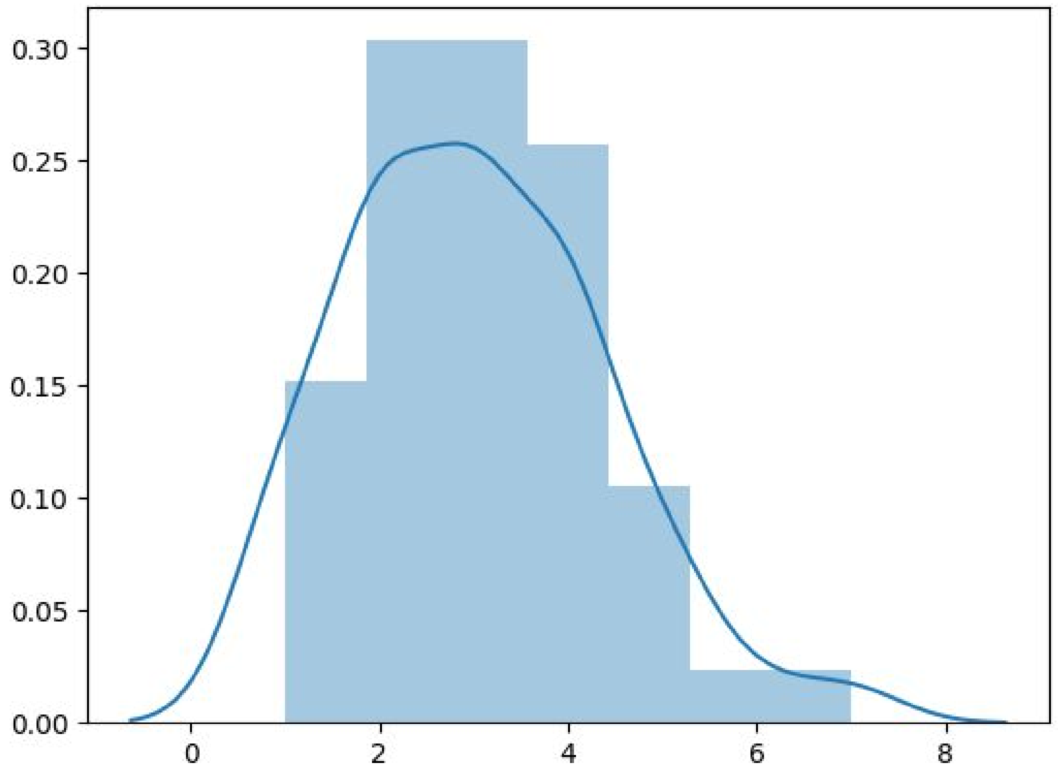


Figure 4: hplot plot for number of heads generated by private

Most of the values are around 3 with a spread from 1 to 7.

As in the case of one coin, we now need to define the likelihood and the prior. We know that the data was drawn from a binomial distribution, so we can define the likelihood as:

```
NumberOfHeads ~ Binomial(10, r)
```

where 10 is the number of coins we are flipping and r is the probability that each of them will land on heads (r being the parameter we are trying to estimate).

Again, we will assume a priori that the rate is drawn from a uniform distribution:

```
r ~ Uniform(0,1)
```

Type sv and you should see:

```
NumberOfHeads = Binomial(10, 0.3, 100)      [3.000000 ... 6.000000]
hplot = distplot(NumberOfHeads)             <_io.Byt...4e749dd0>
NumberOfHeads ~ Binomial(10, r)             [3.000000 ... 6.000000]
r ~ Uniform(0, 1)                           [0.313081 ... 0.332171]
```

To see the mean we can type:

```
mean(r)
```

And you should see:

```
0.32570338330324733
```

Or a number close to that. Again, our model has generated a pretty good estimate of the rate parameter 0.3. (Reminder, we know the actual rate parameter is 0.3 because we generated samples based on a rate of 0.3. When performing this kind of analysis in a real life setting, we typically wouldn't know the true rate that we are trying to estimate.)

Exercise 2: Changing the number of data points

Repeat the Bernoulli experiment but with 1000 data points instead of 100? What happens to the mean r? What happens to the standard deviation of r? Why?

Solution to Exercise 2

You will notice that the means stay roughly the same, but the standard deviation decreases when more samples are added, as the posterior distribution is becoming more precise.

Visually observing the graphs for both distributions, when you have more information (i.e., high n) the posterior becomes more peaked. This means that you are more certain about what values are plausible, and what values are not.

Exercise 3: Changing the Number Of Coins
Change the number of coins in the binomial example to 100. What happens to the mean rate? What happens to the standard deviation of the rate? Why?

Solution for Exercise 3:

The standard deviation is much smaller when 100 coins are flipped. This is because each data point is providing more information about the rate.

Troubleshooting: To update your code, you will need to change the number of coins in both the samples you are generating and the likelihood distribution. If you only update the number of coins in the samples you generate, the parameters for the likelihood will not match the data and you will get an error message.

In the last two sections, we have investigated the use of the Bernoulli and Binomial distributions. These distributions describe processes that have two outcomes, typically True and False, and are useful for modelling quantities like button responses (e.g. Did the participant press “Happy” during this period?) or the occurrence of events (e.g. Did it rain during this period?). In the next section, we will model continuous quantities using the Normal distribution.

Estimating the mean and standard deviation of male heights

The average male over the age of 18 is about 178 centimeters tall. There is also, of course, quite a bit of variation. The standard deviation of the distribution of male heights is about 10 centimeters. In this section, we will estimate these values from data. Below is a data set of male heights. Clear your work space and copy and paste this dataset into Private:

```
heights = [176, 181, 164, 176, 176, 181, 180, 191, 152, 182, 169, 188, 182, 182, 190,
180, 189, 169, 190, 194, 173, 179, 155, 183, 191, 186, 178, 174, 179, 182, 188, 169,
186, 169, 182, 173, 171, 172, 174, 169, 179, 170, 174, 184, 201, 180, 183, 184, 196,
180, 168, 194, 174, 161, 189, 174, 186, 168, 176, 193, 177, 176, 187, 170, 175, 181,
168, 179, 159, 168, 186, 182, 162, 177, 176, 187, 172, 178, 178, 197, 175, 170, 170,
181, 186, 207, 187, 175, 163, 174, 169, 173, 170, 169, 164, 178, 201, 178, 198, 164]
```

You can check the length of the data using the len function:

```
len(heights)
```

should give:

```
100
```

You can also plot the data:

```
heightsplot = distplot(heights)
```

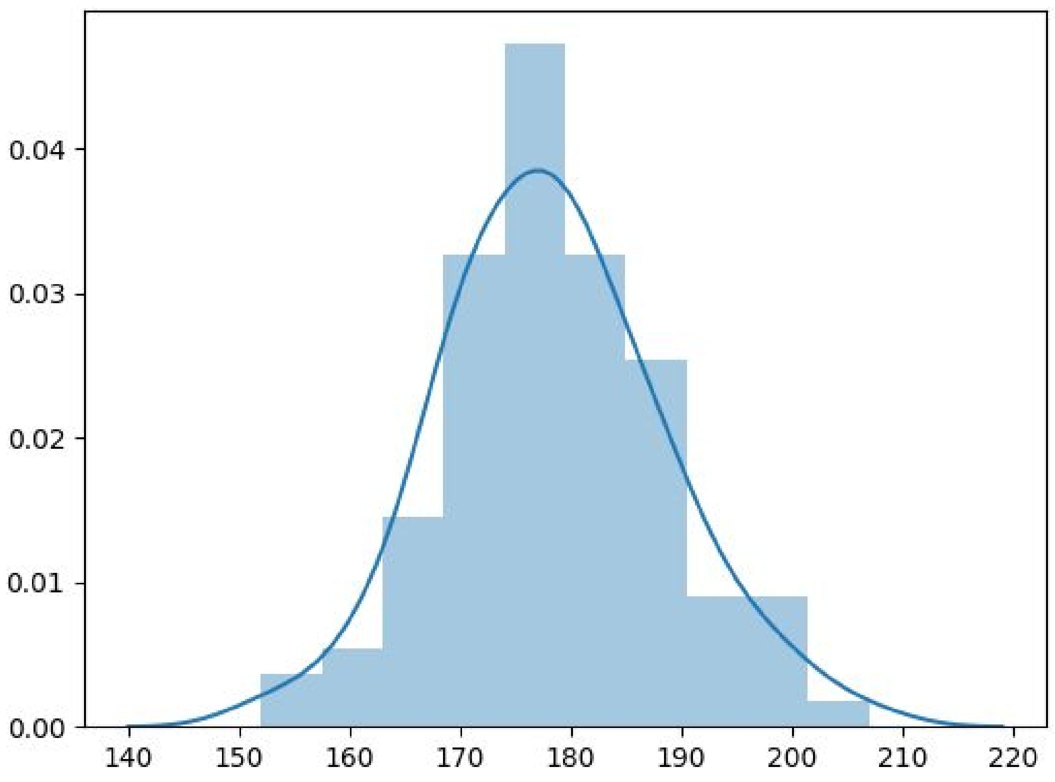


Figure 5: plot for heights generated by private

Now we need to specify the likelihood and priors of the model we will use. In this case, we are going to use the Normal distribution:

```
heights ~ Normal(mu, sigma)
```

mu is the parameter that will represent the mean and sigma is the parameter that will represent the standard deviation. Now we need to specify the priors. We expect heights to be around 150 centimeters, so we specify a Normal distribution with a mean of 150 and a standard deviation that is large (100) to indicate that we don't want our preconceptions to influence the result too much. We expect the the standard deviation of the heights will be positive. Therefore we will use a prior distribution that can only be positive. There are many to choose from, but a good one is the HalfNormal. The HalfNormal is created by taking the absolute value of values from a Normal distribution with mean 0.

To see what the HalfNormal distribution looks like we can generate some samples and then plot them:

```
hn = HalfNormal(100, 1000)
hnplot = distplot(hn, kde=False)
```

Note how we used “kde=False” to stop distplot from adding the smoothing line.

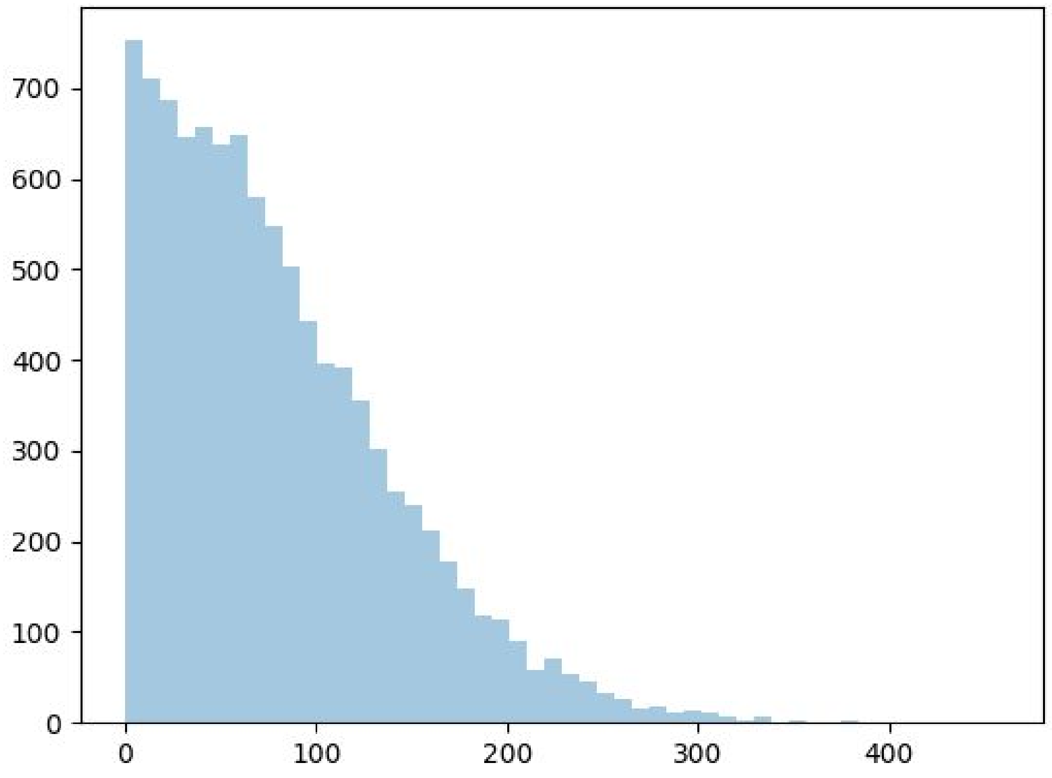


Figure 6: Plot of the half normal distribution, generated by private

Specifying the priors:

```
mu ~ Normal(150, 100)
sigma ~ HalfNormal(100)
```

Exercise 4: Now let’s see how we did. Calculate the mean of mu. How close did we get to the real value? Now calculate the mean of sigma. How close did we get to the real value of the standard deviation?

Solution to Exercise 4

Type:

mean(mu)

and you should get:

178.18825441515514

which is pretty close to 178 centimetres and:

mean(sigma)

which should give:

10.253385821710577

which is pretty close to 10 centimetres. So we have recovered the values quite well.

Exercise 5: Generate a plot of the samples of mu and another of the samples of sigma.

Solution to Exercise 5

Use the distplot() function to generate the plots. Create a variable with an appropriate name and assign the plot to this variable.

Type:
muPlot = distplot(mu)
sigmaPlot = distplot(sigma)

You may be thinking to yourself that we had to do a lot of work just to estimate the bias of a coin or the mean of some heights. In the case of the bias, you could just divide the number of heads by the total number of flips. In the case of the heights, you could just add the heights and divide by 100. There are two main reasons that we go to this trouble. Firstly, by thinking about these simple operations as Bayesian models it is easier to see how we can extend them to more complicated models that would not be so easy to estimate. Secondly, simple statistics like the rate or the mean can be used to reverse engineer people's private data. In the next section, we will demonstrate the first point by showing how to extend our model of heights to a regression model in which we want to determine the impact of a predictor on height. We will leave the second point, until subsequent tutorials.

How does gender impact height?

Woman tend to be shorter than men. On average a woman over the age of 18 is about 165 centimeters tall. The standard deviation of the distribution of female heights is slightly smaller than for men - around 9 centimeters. In this section, we are going to define a simple regression model that predicts height based on gender.

Start by entering the data:

```
heights = [176, 181, 164, 176, 176, 181, 180, 191, 152, 182, 169, 188, 182, 182, 190,
180, 189, 169, 190, 194, 173, 179, 155, 183, 191, 186, 178, 174, 179, 182, 188, 169,
186, 169, 182, 173, 171, 172, 174, 169, 179, 170, 174, 184, 201, 180, 183, 184, 196,
180, 168, 194, 174, 161, 189, 174, 186, 168, 176, 193, 177, 176, 187, 170, 175, 181,
168, 179, 159, 168, 186, 182, 162, 177, 176, 187, 172, 178, 178, 197, 175, 170, 170,
181, 186, 207, 187, 175, 163, 174, 169, 173, 170, 169, 164, 178, 201, 178, 198, 164,
161, 152, 163, 168, 166, 155, 166, 185, 161, 173, 161, 164, 184, 158, 138, 145, 160,
164, 173, 176, 182, 174, 178, 163, 153, 156, 165, 166, 158, 173, 147, 150, 171, 155,
167, 167, 161, 183, 156, 154, 162, 165, 169, 165, 171, 151, 176, 169, 179, 167, 165,
169, 162, 177, 169, 163, 154, 170, 199, 153, 169, 154, 156, 169, 162, 172, 153, 178,
168, 156, 169, 146, 162, 163, 167, 175, 174, 174, 161, 160, 172, 164, 184, 167, 166,
173, 159, 172, 153, 159, 152, 163, 143, 157, 154, 155, 172, 164, 154, 168]
```

The first 100 values are male heights and the second 100 values are female heights. We need to generate a predictor variable to indicate which data are for females are which data are for males. We can do that as follows:

```
gender = [1] * 100 + [0] * 100
```

We are using 1 to indicate a male and a 0 to indicate a female. [1] is a list containing a 1 and when we multiply that by 100 we are indicating we want to create a list that has 100 copies of [1]. We do the same for [0] and then add these lists together to concatenate them into one long list. If you type gender now you should see:

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Now we are ready to specify our regression model. We start by specifying heights as a Normal variable as we did before:

```
heights ~ Normal(mu, sigma)
```

we'll define sigma in the same way also:

```
sigma ~ HalfNormal(100)
```

To determine mu, we will assume that we have an intercept called muFemale (which will effectively be an estimate of the female height) and then a parameter “diff” that indicates how many centimeters to increase the mean by to capture male heights. Our equation for mu then is:

```
mu ~ muFemale + gender * diff
```

Now we need to add the priors. As far as we know apriori, muFemale is similar to mu for the male heights so we will use the Normal distribution with mean 150 and standard deviation 100. The difference in height between men and women will be a smaller number, but still positive, so we will use a HalfNormal with a standard deviation of 30:

```
muFemale ~ Normal(150, 100)
diff ~ HalfNormal(30)
```

When you type sv you should now see something like:

```
heights = [176, 181, 164, 176, 176, 181, 180, 191,      [176, 181, 164, 176, 176, 181, ...]
gender = [1] * 100 + [0] * 100                          [1, 1, 1, 1, 1, 1, ...]
heights ~ Normal(mu, sigma)                     [176, 181, 164, 176, 176, 181, ...]
mu ~ muFemale + gender * diff                   'Not retained.'
sigma ~ HalfNormal(100)                                 [9.689555 ... 10.918070]
muFemale ~ Normal(150, 100)                     [163.865988 ... 164.503482]
diff ~ HalfNormal(30)                           [14.233914 ... 13.784624]
```

Private has generated samples of muFemale and diff, but has not kept the samples of mu. This is because these internally calculated values can be very large and so to avoid storing all that data, the sampler makes a strategic decision to throw them away.

Exercise 6: Calculate the mean of muFemale. Does it correspond to the real value?

Solution to Exercise 6

```
meanMuFemale = mean(muFemale)
```

The mean should be very close to the true value, somewhere around 165 (plus or minus 1 or 2 centimeters).

Exercise 7: Calculate samples of the mean of the male heights? (Hint: You will need to use muFemale and diff). Do they correspond to reality?

Solution to Exercise 7

muMale = muFemale + diff
meanMuMale = mean(muMale)

Alternative Solution to Exercise 7

meanMuMale = mean(muFemale + diff)

They correspond quite well.

Exercise 8: If we wanted to allow for the possibility that females are taller than males, how would we need to change the code?

Solution to Exercise 8

diff would have to be specified as a distribution that can take on negative values, such as the normal distribution.

Summary

That concludes our introduction to the Private language. You should now understand how to construct models of both binary and continuous data and how to calculate statistics of and plot the samples that are draw of the parameters.

Some useful private commands and operators

Command	Explanation	Example
=	**"Assignment Operator"** Assigns a variable	X = 3
del	**"Delete variable"** Removes a variable	del x
sv	**"Show Variable"** - Prints all assigned variables to the console	sv
~	**"Is distributed as"** Defines the distribution by	X ~ Normal(Rate,n) Y ~ Bernoulli(0.3,100)

	which the data is distributed as	
mean()	**“Mean Function” Calculates the mean of any variable you put inside the parentheses**	meanr = mean(r)
std()	**“Standard Deviation Function” Calculates the standard deviation of any variable you put inside the parentheses**	stdr = std(r)

Private Messages You May See

Message	**Explanation**
Computing	The computer has not finished generating the samples and therefore can not display the samples or run any calculations on the samples. Give it some time to finish running, and try sv again. You can continue to code while you wait for the sampler to run.
Stale	Stale state is information in a variable that does not reflect reality either because it depends on a variable that is not defined or has yet to be computed.
Private	This variable is kept private by the Private programming language. It will not be released (shown) to the researcher because this could be used to reveals sensitive information (more on this in week 6)