Week 2 Exercises

Exercise 1 Bernoulli grid search

Suppose you design a Bernoulli experiment that generates successes or failures with an unknown probability π. You want to estimate π, so you run the experiment three times and get the outcomes `y <- c(0, 1, 0)`, where `0` is a failure and `1` is a success.

Use ML to estimate `pi`. *But don't find the maximum analytically* or with a hill-climbing algorithm. Instead, use a *grid search* to find the maximum. Use `seq()` to create a range of ten to twenty candidate values for π, compute the log-likelihood for each candidate value, and locate the candidate value that produces the largest log-likelihood. Report your results in figure and a table.

Solution

First, create a function to compute the log-likelihood.

```
# create function to compute log-likelihood
compute_log_lik <- function(pi, y) {
  k <- sum(y)
  N <- length(y)
  log_lik <- k*log(pi) + (N - k)*log(1 - pi)
  return(log_lik)
}
```

Now compute the log-likelihood for 11 evenly spaced values of π from 0 to 1. It similarly shows that the log-likelihood is maximized (among the candidate values) when $\pi = 0.4$.

```
# load packages
library(tidyverse)

# create data set
y <- c(0, 1, 0)
```

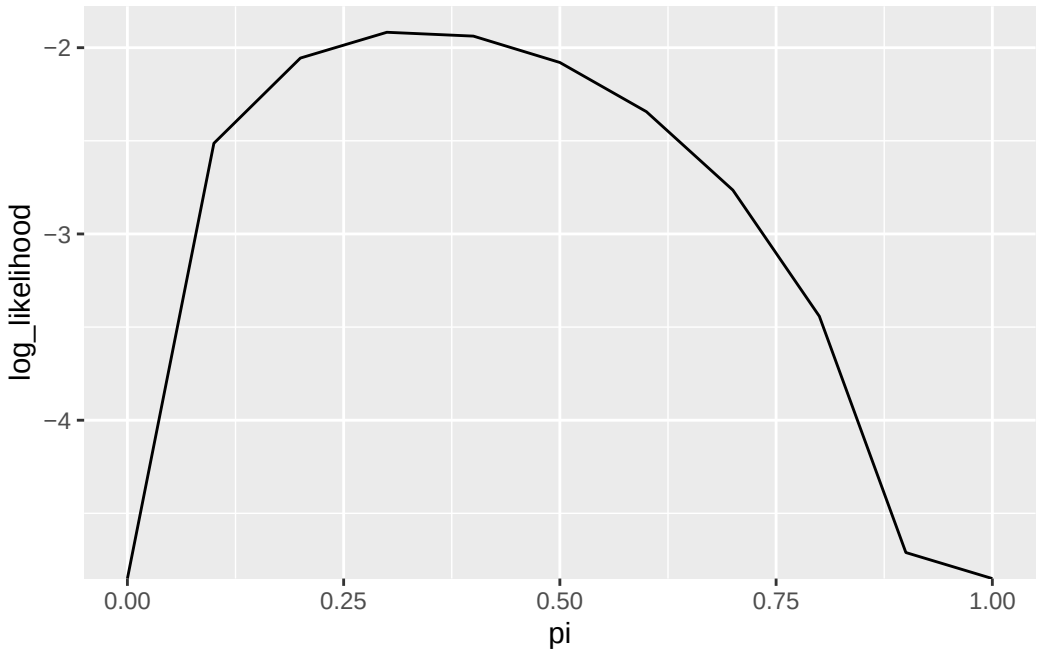
Candidate Value	Log-Likelihood
0	-Inf
0.1	-2.51
0.2	-2.06
0.3	-1.92
0.4	-1.94
0.5	-2.08
0.6	-2.34
0.7	-2.76
0.8	-3.44
0.9	-4.71
1	-Inf

```
# create candidate values
pi <- seq(0, 1, length.out = 11)

# compute log-likelihood for each candidate value
log_likelihood <- compute_log_lik(pi, y)

# combine candidates and their log-likelihoods into df
df <- data.frame(pi, log_likelihood)

# make a table
df |>
 rename("Candidate Value" = pi,
        "Log-Likelihood" = log_likelihood) |>
 tinytable::tt(digits = 3)
```

The largest value of the log-likelihood occurs at $\pi = 0.4$. I knew it would be 0.3 or 0.4 because the ML estimate is the sample mean and the sample mean is 0.333 in this case.

The figure below plots the log-likelihood for each candidate value. Again, we see that 0.4 produces the largest log-likelihood (among the candidate values).

```
# plot log-likelihoods
ggplot(df, aes(x = pi, y = log_likelihood)) +
  geom_line()
```



Exercise 2 Equivalence of numerical and closed-form solutions

We found that the sample average is the ML estimate of the parameter λ of the Poisson distribution. For the data set `y <- c(12, 7, 9, 12, 10)` show that optimizing the Poisson log-likelihood function with `optim()` produces the same answer and the closed-form solution.

Solution

```
# data set
y <- c(12,  7,  9, 12, 10)
```

First, the closed-form solution.

```
# closed-form maximum
mean(y)
```

```
[1] 10
```

Next, the numerical solution.

```r
# create log-likelihood function to maximize
ll_pois <- function(par, y) {
  sum(dpois(y, lambda = par, log = TRUE))
}

# maximize the log-likelihood function
est <- optim(par = 1,
      fn = ll_pois,
      y = y,
      control = list(fnscale = -1),
      method = "Brent",
      lower = 0, upper = 100)

# print lambda that maximizes log-likelihood
est$par
```

```
[1] 10
```

Exercise 3

DeGroot and Schervish, q. 9, p. 425. Suppose a distribution $f(x;\theta) = \theta x^{\theta-1}$ for $0 < x < 1$ and $\theta > 0$. Find the ML estimator of θ.

Solution

We have the pdf $f(x;\theta) = \theta x^{\theta-1}, \quad 0 < x < 1,\ \theta > 0.$

Begin by writing the likelihood function:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{N} f(x_i;\theta) \\ &= \prod_{i=1}^{N} \theta x_i^{\theta-1} \\ &= \theta^N \cdot \prod_{i=1}^{N} x_i^{\theta-1} \end{aligned}$$

Take the log of the likelihood:

$$\begin{aligned}\log L(\theta) &= \log(\theta^N) + \log\left(\prod x_i^{\theta-1}\right) \\ &= N\log\theta + (\theta-1)\sum_{i=1}^{N}\log x_i\end{aligned}$$

Take the derivative with respect to θ:

$$\frac{d}{d\theta}\log L(\theta) = \frac{N}{\theta} + \sum_{i=1}^{N}\log x_i$$

Set the derivative equal to 0 and solve for $\hat{\theta}$:

$$\begin{aligned}\frac{N}{\hat{\theta}} + \sum \log x_i &= 0 \\ \frac{N}{\hat{\theta}} &= -\sum \log x_i \\ \hat{\theta} &= \frac{N}{-\sum_{i=1}^{N}\log x_i}\end{aligned}$$

So the maximum likelihood estimator is $\hat{\theta} = \frac{N}{-\sum \log x_i}$.

Exercise 4

Suppose you have a binary outcome `y <- c(0, 1, 0, 1, 1, 1, 0)`. Suppose you want to estimate the *mean* of this distribution. You'd normally model the binary outcome with a Bernoulli distribution and estimate the parameter Bernoulli parameter π with ML. But instead of using the Bernoulli distribution, you model these data with a normal distribution. What would your estimate of the mean be? What if you used the Poisson? Exponential? Explain.

Solution

Distribution	Parameter(s)	ML Estimates of Parameters	Estimate of Mean (Using Invariance Property)
Bernoulli	π	$\hat{\pi} = \text{avg}(y)$	$\hat{\pi} = \text{avg}(y)$ (No change needed, because $E(Y) = \pi$)
Normal	μ, σ^2	$\hat{\mu} = \text{avg}(y)$, $\hat{\sigma}^2 = \text{var}(y)$	$\hat{\mu} = \text{avg}(y)$
Poisson	λ	$\hat{\lambda} = \text{avg}(y)$	$\hat{\lambda} = \text{avg}(y)$
Exponential	λ	$\hat{\lambda} = 1/\text{avg}(y)$	$1/\hat{\lambda} = \text{avg}(y)$

No matter what distribution we use, we obtain the *identical* estimate. Not the same in expectation (over a large number of imaginary repetitions). Not the same asymptotically (as the sample size grows large). The exact same number every time. This means that any property that the Bernoulli estimator has, the others have as well. If the Bernoulli estimator is consistent, then the other three estimators are consistent as well. It's also worth noting that all three are unbiased estimators as well.

This illustrates an important point. The quality of an estimator doesn't always depend on the correctness of all parts of the model. It often depends on what quantity of interest you are targeting.

Exercise 5

Let $\hat{\theta}$ be the ML estimate of θ. Suppose we are interested in estimating $\psi = \theta^2$. What is the ML estimate of ψ?

Solution

By the **invariance property**, the MLE of a function of a parameter is the function of the MLE:

$$\hat{\psi} = \hat{\theta}^2$$

This allows us to avoid deriving a new likelihood for ψ directly. Instead, we just plug the ML estimate of θ into the transformation.

Exercise 6 Uniform distribution

Suppose a discrete uniform distribution from 0 to K. The pmf is $f(x; K) = \frac{1}{K+1}, \quad x \in \{0, 1, ..., K\}$. Suppose I have a sample of size 3 from the distribution: 276, 159, and 912.

a. Find the ML estimate of K. *Hint: The log-likelihood is discontinuous, so the usual optimization routine might mislead you. But the maximum is immediately apparent once you write out the likelihood.*
b. Find the method of moments estimate of K. *Hint: The mean of this uniform distribution is $\frac{K}{2}$. Set the sample mean equal to the model mean and solve.*
c. Discuss any problems you notice with each estimator.

Solution

Part (a)

The pmf for the uniform distribution from 0 to K is $\frac{1}{K+1}$ for $x \in \{0, 1, \ldots, K\}$ and 0 otherwise. Thus the likelihood function is

$$L(K) = \prod_{i=1}^{N} \frac{1}{K+1} = \left(\frac{1}{K+1}\right)^N$$

if all values of x fall in $\{0, 1, \ldots, K\}$, and 0 if *any* values of x fall outside. To maximize this expression, we need K as small as possible, subject to $K \geq \max(x)$. Hence, $\widehat{K}_{\text{MLE}} = \max(x)$.

Part (b)

The mean of this uniform distribution is $\frac{K}{2}$. To find the method of moments estimator $\widehat{K}_{MM}$, we set the sample mean equal to the distribution mean $\text{avg}(x) = \frac{\widehat{K}_{MM}}{2}$. Solving gives $\widehat{K}_{MM} = 2 \cdot \text{avg}(x)$.

Part (c)

These estimators are both interesting, but neither is very satisfying on its face by simple inspection.

- The ML estimator is the smallest logically possible value of K. Intuitively, it may seem better to guess "a little" higher than the largest data point.

- The MM estimator is just twice the average. This means that our estimate can fall below the largest observation. For example, the MM estimate for the given data is 898, but we observed a value of 912, so K cannot be smaller than 912.

Exercise 7 Exponential model

The exponential distribution has pdf $f(t; \lambda) = \lambda e^{-\lambda t}$ for $t \geq 0$ and $\lambda > 0$. We sometimes use this distribution to model time spells t, such as the time until an event or the time between events. For example, some political scientists are interested in how long a government lasts after a government formation in a parliamentary system. We might use an exponential distribution to model this time or "duration."

a. Show that the cdf of the exponential distribution is $F(t; \lambda) = \Pr(T \leq t; \lambda) = 1 - e^{-\lambda t}$. Use the cdf to find the survival function $S(t; \lambda) = \Pr(T > t; \lambda) = 1 - F(t; \lambda)$. How can we interpret the survival function (i.e., for input t, what does the survival function return)?

b. Find $\Pr(T > t + s \mid T > s)$ using the survival function and the definition of conditional probability. Compare $P(T > t)$ to $P(T > t + s \mid T > s)$. What do you notice? Interpret the result. Using a specific political outcome as a concrete example (e.g., time between major protests, government durations, time between constitutional amendments), what does this property say about the time until an event occurs, given that it has been some given time since an event occurred?

c. Suppose we collect N random samples $t = \{t_1, t_2, ..., t_N\}$ and model each draw as an exponential random variable random. Find the ML estimator of λ.

d. Find an ML estimator for the mean, which is $\frac{1}{\lambda}$.

Optional: Show that the mean of the exponential distribution is $\frac{1}{\lambda}$. This requires integration by parts.

Solution

Part (a) To find the cumulative distribution function (cdf) $F(t; \lambda) = \Pr(T \leq t)$ for the exponential distribution, we need to integrate the pdf from 0 to t. This gives $F(t; \lambda) = \int_0^t \lambda e^{-\lambda u} du$. The indefinite integral of $e^{-\lambda u}$ w.r.t. u is $\int e^{-\lambda u} du = \frac{-1}{\lambda} e^{-\lambda u} + C$. The definite integral from $u = 0$ to $u = t$ is

$$\int_0^t \lambda e^{-\lambda u} du = \lambda \left[\frac{-1}{\lambda} e^{-\lambda u} \right]_0^t = -e^{-\lambda u} \Big|_0^t = -e^{-\lambda t} + e^0 = 1 - e^{-\lambda t}.$$

Thus, the cdf is $F(t; \lambda) = 1 - e^{-\lambda t}$.

We can subtract the CDF from one to obtain the **survival function**.

$$S(t; \lambda) = \Pr(T > t) = 1 - F(t; \lambda) = 1 - (1 - e^{-\lambda t}) = e^{-\lambda t}$$

The survival function $S(t; \lambda)$ outputs the probability that the event *has not yet occurred* by time t. (The CDF outputs the probability than an event *has* occurred). For example, if $t = 3$ years and $S(3; \lambda) = 0.20$, then there's a 20% chance the event will occur **after** 3 years.

Part (b)

Using the definition of conditional probability, $\Pr(T > t + s \mid T > s) = \frac{\Pr(T > t+s \text{ and } T > s)}{\Pr(T > s)}$. But when $T > t + s$, then T is also greater than s, so $\Pr(T > t + s \mid T > s) = \frac{\Pr(T > t+s)}{\Pr(T > s)}$.

Substituting in the survival function,

$$\Pr(T > t + s \mid T > s) = \frac{e^{-\lambda(t+s)}}{e^{-\lambda s}} = \frac{e^{-\lambda t} e^{-\lambda s}}{e^{-\lambda s}} = e^{-\lambda t}$$

This is interesting… we showed that $\Pr(T > t + s \mid T > s) = \Pr(T > t)$. This means that distribution of T and the distribution of $T \mid T > s$. Are exactly the same. This is

the **memoryless property** of the exponential distribution. It means that the probability of waiting at least t more units of time *does not depend* on how long you've already waited. The distribution has no "memory" of past events.

Political example: Suppose we use an exponential distribution to model government durations. If a government has already lasted two years, the chance that the government lasts at least three *more* years is the same is the chance it lasted three years in the first place. Both chances are $e^{-3\lambda}$. The fact that two years have already passed provides no information about how much longer the government will last. That's why we say the exponential distribution is "memoryless." (The exponential distribution is the *only* distribution with this property.)

To illustrate this further, consider human lifespans. If I asked you to guess how much longer a person will live, you'd definitely want to know their age. The life expectancy is completely different for someone who is 2, 20, and 80. This is because human lifespans are not memoryless.

Part (c)

Suppose we observe data $t = \{t_1, t_2, \dots, t_N\}$, where each t_i is drawn independently from an exponential distribution with unknown rate λ.

We begin by writing the likelihood function:

$$\begin{aligned} L(\lambda) &= \prod_{i=1}^{N} f(t_i; \lambda) \\ &= \prod_{i=1}^{N} \lambda e^{-\lambda t_i} \\ &= \lambda^N \cdot e^{-\lambda \sum_{i=1}^{N} t_i} \end{aligned}$$

Take the log of the likelihood:

$$\begin{aligned} \log L(\lambda) &= \log(\lambda^N) + \log\left(e^{-\lambda \sum t_i}\right) \\ &= N \log \lambda - \lambda \sum_{i=1}^{N} t_i \end{aligned}$$

Take the derivative with respect to λ:

$$\frac{d}{d\lambda} \log L(\lambda) = \frac{N}{\lambda} - \sum_{i=1}^{N} t_i$$

Set the derivative equal to 0 and solve for $\hat{\lambda}$:

$$\frac{N}{\hat{\lambda}} - \sum t_i = 0$$
$$\frac{N}{\hat{\lambda}} = \sum t_i$$
$$\hat{\lambda} = \frac{N}{\sum_{i=1}^{N} t_i}$$

Recognizing that $\sum t_i = N \cdot \text{avg}(t)$, we can simplify:

$$\hat{\lambda} = \frac{1}{\text{avg}(t)}$$

So the maximum likelihood estimator of λ is the **reciprocal of the sample average**.

Part (d)

By the invariance property ML estimator for the mean $\mu = \frac{1}{\lambda}$ of the exponential distribution is simply $\hat{\mu} = \frac{1}{\hat{\lambda}} = \frac{1}{\frac{1}{\text{avg}(x)}} = \text{avg}(x)$.

(Optional)

To find the mean of the exponential distribution, compute:

$E(T) = \int_0^\infty t\lambda e^{-\lambda t} dt$

This requires **integration by parts**. Let:

- $u = t$, so $du = dt$
- $dv = \lambda e^{-\lambda t} dt$, so $v = -e^{-\lambda t}$

Then:

$E(T) = -te^{-\lambda t}\big|_0^\infty + \int_0^\infty e^{-\lambda t} dt = 0 + [\frac{1}{\lambda}] = \frac{1}{\lambda}$

So the mean of the exponential distribution is $\frac{1}{\lambda}$.

Exercise 8 Simulating memorylessness in R

Use R to simulate 10,000 draws from an exponential distribution with rate $\lambda = 1$. Each of the 10,000 values represents a duration (i.e., time until an event occurs). For

```
set.seed(123)
durations <- rexp(10000, rate = 1)
```

For durations larger than five, compute the *remaining* duration.

```
remaining_durations <- durations[durations > 5] - 5
```

Use a histogram or plots of the ECDF to compare the distribution of `remaining_durations` to the original `durations`. Explain the similarity, why it's expected, and why it's important.

Solution

Begin by simulating 10,000 draws from an exponential distribution with rate $\lambda = 1$. Each value represents a waiting time (or duration) until an event occurs.

```
set.seed(123)
durations <- rexp(10000, rate = 1)
```

Now compute the **remaining durations** for those observations where the event did not occur by time 5. If the time is in years, this is asking: "given that we waited 5 years without seeing the event, how much longer did we need to wait?"

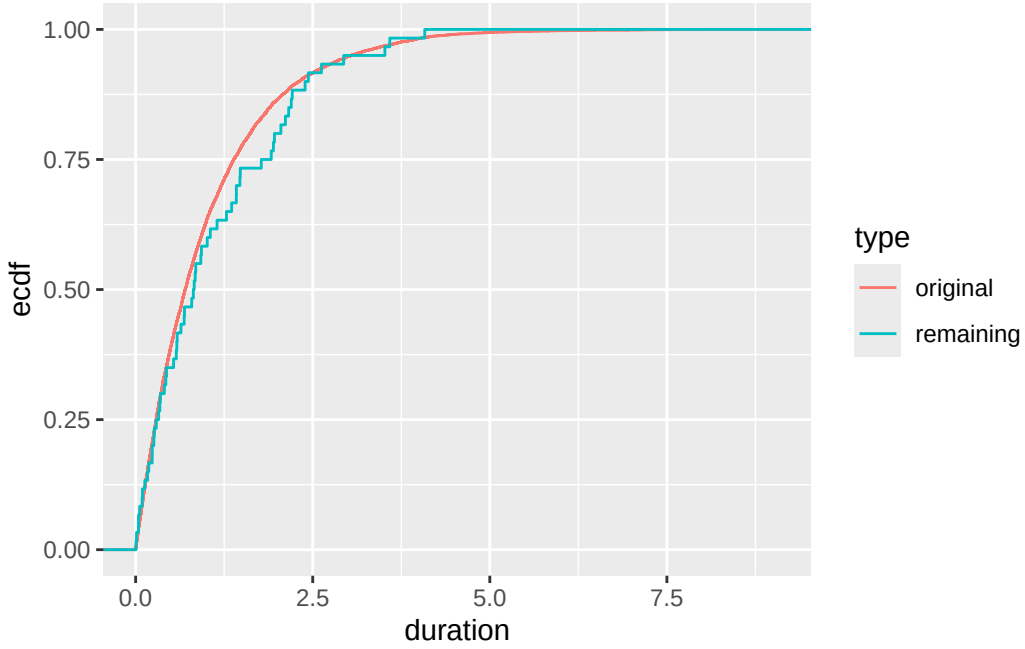
```
remaining_durations <- durations[durations > 5] - 5
```

Now compare the distribution of the `remaining_durations` to the original `durations`.

```
# put simulations in a tidy data frame
df1 <- data.frame(duration = durations,
                  type = "original")
df2 <- data.frame(duration = remaining_durations,
                  type = "remaining")
df <- bind_rows(df1, df2) |>
  glimpse()
```

```
Rows: 10,060
Columns: 2
$ duration <dbl> 0.84345726, 0.57661027, 1.32905487, 0.03157736, 0.05621098, 0~
$ type     <chr> "original", "original", "original", "original", "original", "~
```

```r
# plot the ecdfs
ggplot(df, aes(x = duration, color = type)) +
  stat_ecdf()
```



This distribution of `remaining_durations` is indistinguishable from the original `durations`!

We know from the theory that the differences are simply due to noise. This occurs because of the **memorylessness** of the exponential distribution: the probability of waiting *at least* t more units of time **does not depend** on how long you've already waited.

If we're modeling the **time between major protests** with an exponential distribution, then the probability that a protest occurs in the next day is the same whether it's been 2 days or 20 days since the last protest. For many applications, this memorylessness won't make sense and we'll need a more flexible distribution.

Exercise 9 The `faithful` data set

The `faithful` data set in base R contains 272 observations of `eruptions` (eruption time in minutes) and `waiting` (waiting time to next eruption in minutes) for the Old Faithful geyser

in Yellowstone National Park. See `?faithful` for more details.

```
glimpse(faithful)
```

```
Rows: 272
Columns: 2
$ eruptions <dbl> 3.600, 1.800, 3.333, 2.283, 4.533, 2.883, 4.700, 3.600, 1.95~
$ waiting   <dbl> 79, 54, 74, 62, 85, 55, 88, 85, 51, 85, 54, 84, 78, 47, 83, ~
```

Model `eruptions` as an exponential distribution. Estimate the rate and mean. Use the predictive distribution to evaluate the fit of the exponential model to these data.

Optional. Repeat for `waiting`.

Solution

For `eruptions`.

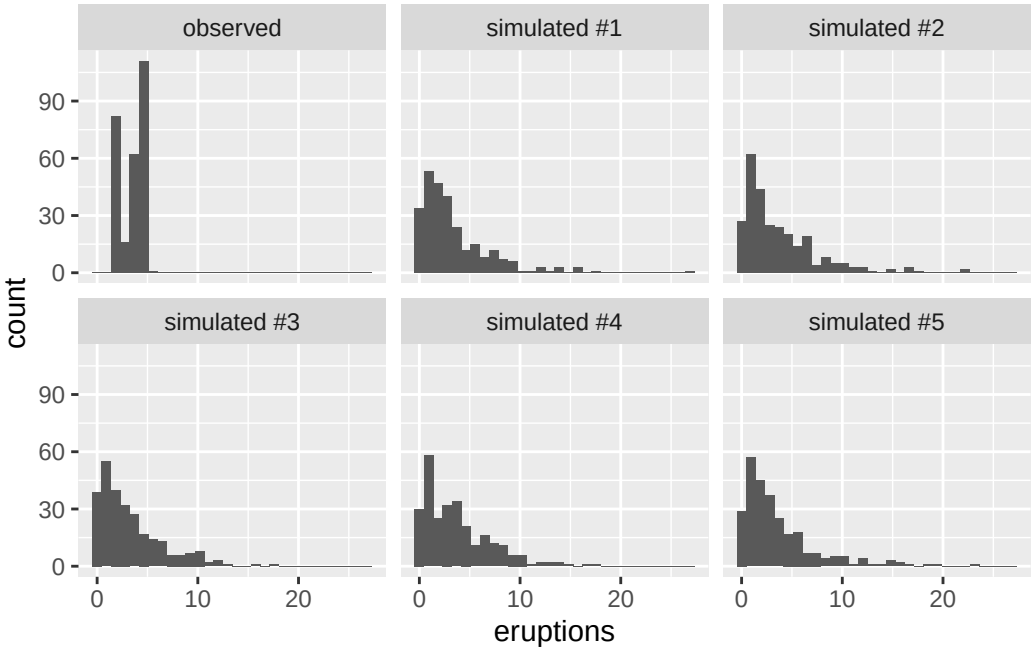
```
# compute ml estimates of lambda (exponential rate)
lambda_hat <- 1/mean(faithful$eruptions)

# create observed data set
observed_data <- tibble(eruptions = faithful$eruptions, type = "observed") %>%
  glimpse()
```

```
Rows: 272
Columns: 2
$ eruptions <dbl> 3.600, 1.800, 3.333, 2.283, 4.533, 2.883, 4.700, 3.600, 1.95~
$ type      <chr> "observed", "observed", "observed", "observed", "observed", ~
```

```
# simulate five fake data sets
sim_list <- list()
n <- length(faithful$eruptions)
for (i in 1:5) {
  y_pred <- rexp(n, rate = lambda_hat)
  sim_list[[i]] <- tibble(eruptions = y_pred,
                          type = paste0("simulated #", i))
}

# combine the fake and observed data sets
gg_data <- bind_rows(sim_list) %>%
  bind_rows(observed_data) %>%
  glimpse()
```

```
Rows: 1,632
Columns: 2
$ eruptions <dbl> 0.66964861, 1.70461944, 2.59265294, 3.76124617, 3.54745264, ~
$ type      <chr> "simulated #1", "simulated #1", "simulated #1", "simulated #~
```

```
# plot the observed and fake data sets
ggplot(gg_data, aes(x = eruptions)) +
  geom_histogram() +
  facet_wrap(vars(type))
```



The exponential model does not fit these data well. Why the simulated data have roughly the same average, then simulated data have a much larger SD. The R code below computes the average and SD for the observed and simulated data sets. Much like the Poisson distribution, the mean and the variance of the distribution are linked. In the case of the exponential, the mean equals the SD (you can see that in the summaries below). However the mean and the SD are quite different in the observed data, so the exponential model *can't* fit these data well.

```
summarize(gg_data,
          avg = mean(eruptions),
          sd = sd(eruptions),
          .by = type)
```

```
# A tibble: 6 x 3
```

```
  type           avg    sd
  <chr>        <dbl> <dbl>
1 simulated #1  3.46  3.58
2 simulated #2  3.69  3.79
3 simulated #3  3.32  3.16
4 simulated #4  3.69  3.36
5 simulated #5  3.58  3.76
6 observed      3.49  1.14
```

For `waiting`.

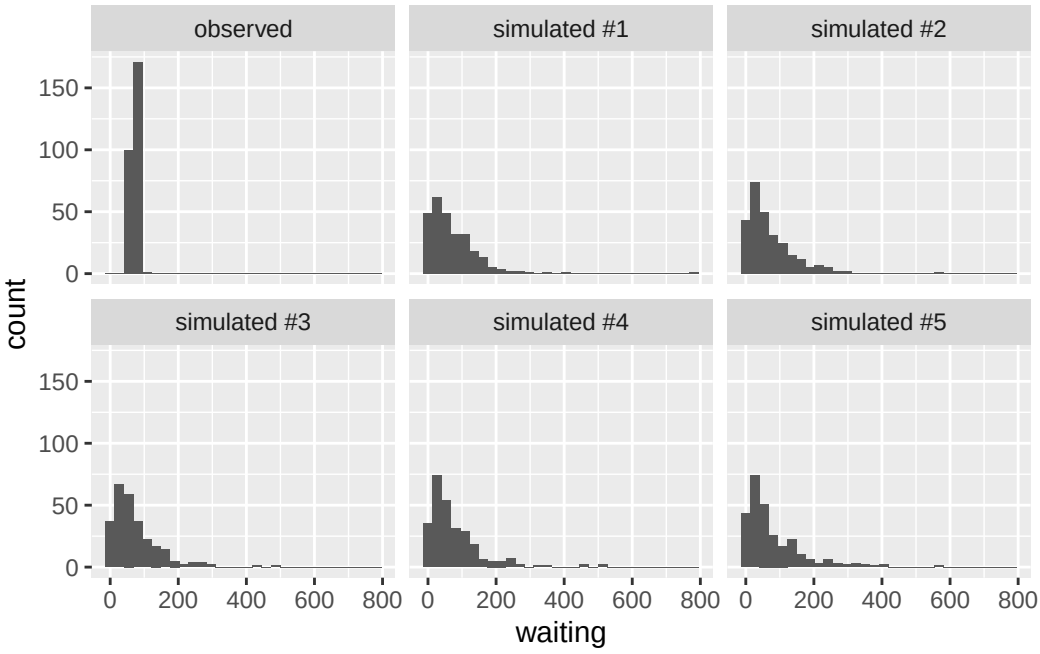
```
# compute ml estimates of lambda (exponential rate)
lambda_hat <- 1/mean(faithful$waiting)

# create observed data set
observed_data <- tibble(waiting = faithful$waiting, type = "observed") %>%
  glimpse()
```

```
Rows: 272
Columns: 2
$ waiting <dbl> 79, 54, 74, 62, 85, 55, 88, 85, 51, 85, 54, 84, 78, 47, 83, 52~
$ type    <chr> "observed", "observed", "observed", "observed", "observed", "o~
```

```
# simulate five fake data sets
sim_list <- list()
n <- length(faithful$waiting)
for (i in 1:5) {
  y_pred <- rexp(n, rate = lambda_hat)
  sim_list[[i]] <- tibble(waiting = y_pred,
                          type = paste0("simulated #", i))
}

# combine the fake and observed data sets
gg_data <- bind_rows(sim_list) %>%
  bind_rows(observed_data) %>%
  glimpse()
```

```
Rows: 1,632
Columns: 2
$ waiting <dbl> 9.880443, 121.205849, 22.685621, 19.431747, 15.665561, 66.8877~
$ type    <chr> "simulated #1", "simulated #1", "simulated #1", "simulated #1"~
```

```
# plot the observed and fake data sets
ggplot(gg_data, aes(x = waiting)) +
  geom_histogram() +
  facet_wrap(vars(type))
```



The results for `waiting` are nearly identical to the results for `eruptions`. The exponential model cannot fit these data well because the observed values are much less dispersed than an exponential model assumes for a mean of about 70.

```
summarize(gg_data,
          avg = mean(waiting),
          sd = sd(waiting),
          .by = type)
```

```
# A tibble: 6 x 3
  type           avg    sd
  <chr>        <dbl> <dbl>
1 simulated #1  72.5  76.7
2 simulated #2  71.2  70.2
3 simulated #3  73.1  69.2
4 simulated #4  76.8  80.5
5 simulated #5  78.2  85.2
6 observed      70.9  13.6
```

Exercise 10 Herron's hockey data set

Herron's hockey data set contains data on the time between hits[1] in the regulation periods of the 82 regular season games for the Chicago Blackhawks. The data are available via GitHub Gist here. You can load the data directly from the web, but you need to click "Raw" and copy the URL for the raw CSV data. Model `seconds_btw_hits` as an exponential distribution. Estimate the rate and mean. Use the predictive distribution to evaluate the fit of the exponential model to these data. Discuss whether you find the fit surprising and explain why.

```
# load data directly from web
hockey <- read_csv("https://gist.githubusercontent.com/carlislerainey/0bc3018cd2377022fd045e

# quick look
glimpse(hockey)
```

```
Rows: 1,175
Columns: 5
$ game_id                 <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, ~
$ period_id               <dbl> 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1, 1, 2, ~
$ time_of_hit             <chr> "25S", "2M 8S", "2M 58S", "3M 8S", "4M 13S", ~
$ seconds_played_in_season <dbl> 25, 128, 178, 188, 253, 560, 643, 1407, 1825,~
$ seconds_btw_hits        <dbl> NA, 103, 50, 10, 65, 307, 83, 764, 418, 535, ~
```

Solution

```
# compute ml estimates of lambda (exponential rate)
y <- na.omit(hockey$seconds_btw_hits)
lambda_hat <- 1/mean(y)

# create observed data set
observed_data <- tibble(seconds_btw_hits = y, type = "observed") %>%
  glimpse()
```

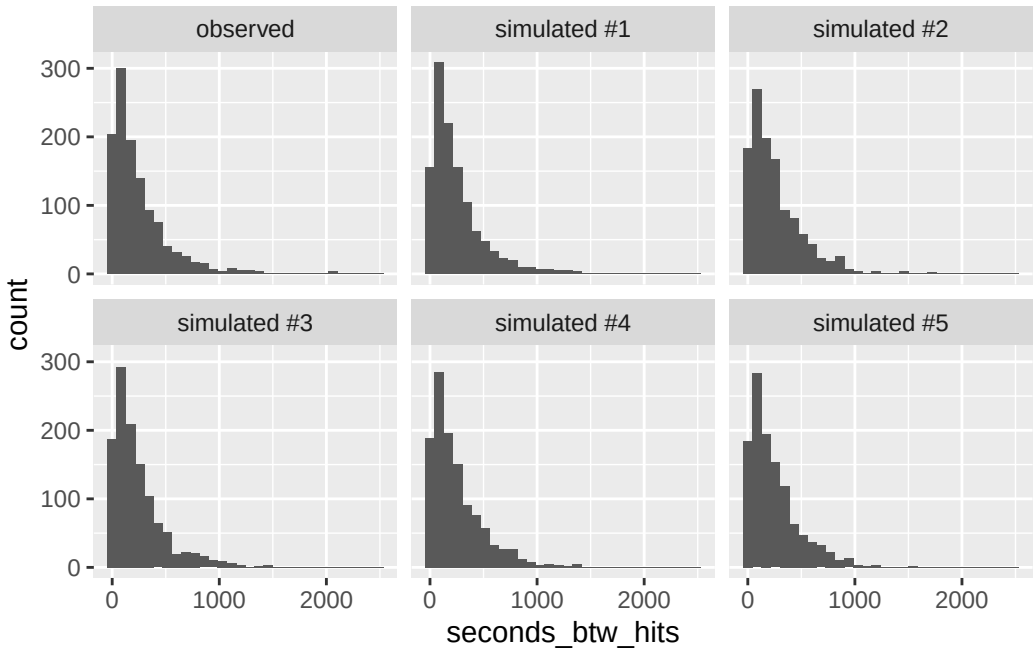
```
Rows: 1,174
Columns: 2
$ seconds_btw_hits <dbl> 103, 50, 10, 65, 307, 83, 764, 418, 535, 2022, 88, 17~
$ type             <chr> "observed", "observed", "observed", "observed", "obse~
```

[1] From Wikipedia, a hit is defined as: *"Intentionally initiated contact with the player possessing the puck that causes that player to lose possession of the puck. Loss of possession may or may not involve a turnover. If the contact results in a penalty, no hit is awarded."*

```
# simulate five fake data sets
sim_list <- list()
n <- length(y)
for (i in 1:5) {
  y_pred <- rexp(n, rate = lambda_hat)
  sim_list[[i]] <- tibble(seconds_btw_hits = y_pred,
                          type = paste0("simulated #", i))
}

# combine the fake and observed data sets
gg_data <- bind_rows(sim_list) %>%
  bind_rows(observed_data) %>%
  glimpse()
```

```
Rows: 7,044
Columns: 2
$ seconds_btw_hits <dbl> 66.0054302, 104.4624419, 114.6189980, 287.8180699, 4.~
$ type             <chr> "simulated #1", "simulated #1", "simulated #1", "simu~
```

```
# plot the observed and fake data sets
ggplot(gg_data, aes(x = seconds_btw_hits)) +
  geom_histogram() +
  facet_wrap(vars(type))
```



These distributions look *very* similar. For a more refined comparison, let's plot the ECDFs and put them all in the same panel. Even with this very precise comparison between the two, it's almost impossible to distinguish between the observed and fake data.

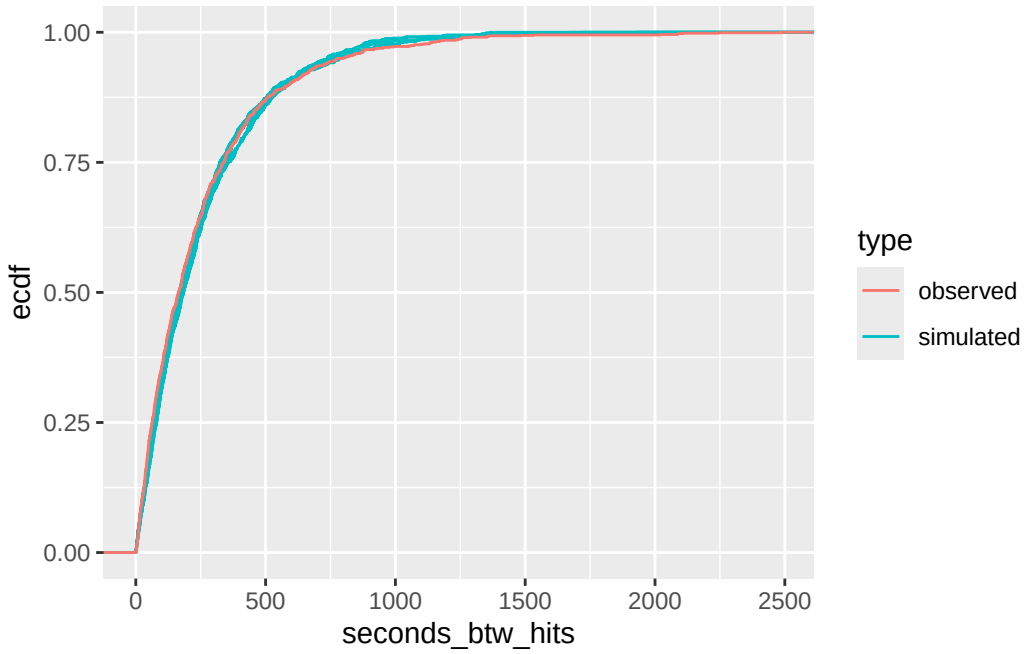
In sports, we often then about momentum and mental states. We imagine big swings in favor of one team and against another. We might imagine that hits are related. If there has been a hit or two or three recently, we'll see more in the near future. Similarly, we might imagine that if it's been a while since we've seen a hit, we're unlikely to see one soon. But these data show that's not the case. The time since that last hit tells us nothing about the likelihood of a hit in the near future! Maybe momentum is overrated?

```
# separate the labels of the simulated data into two parts
gg_data2 <- gg_data |>
  separate(type, into = c("type", "version")) |>
  glimpse()
```

```
Warning: Expected 2 pieces. Missing pieces filled with `NA` in 1174 rows [5871, 5872,
5873, 5874, 5875, 5876, 5877, 5878, 5879, 5880, 5881, 5882, 5883, 5884, 5885,
5886, 5887, 5888, 5889, 5890, ...].
```

```
Rows: 7,044
Columns: 3
$ seconds_btw_hits <dbl> 66.0054302, 104.4624419, 114.6189980, 287.8180699, 4.~
$ type             <chr> "simulated", "simulated", "simulated", "simulated", "~
$ version          <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1"~
```

```
# plot histograms of real and fake data
ggplot(gg_data2, aes(x = seconds_btw_hits, color = type, group = version)) +
  stat_ecdf()
```



Exercise 11 The location-scale t distribution

Some Theory

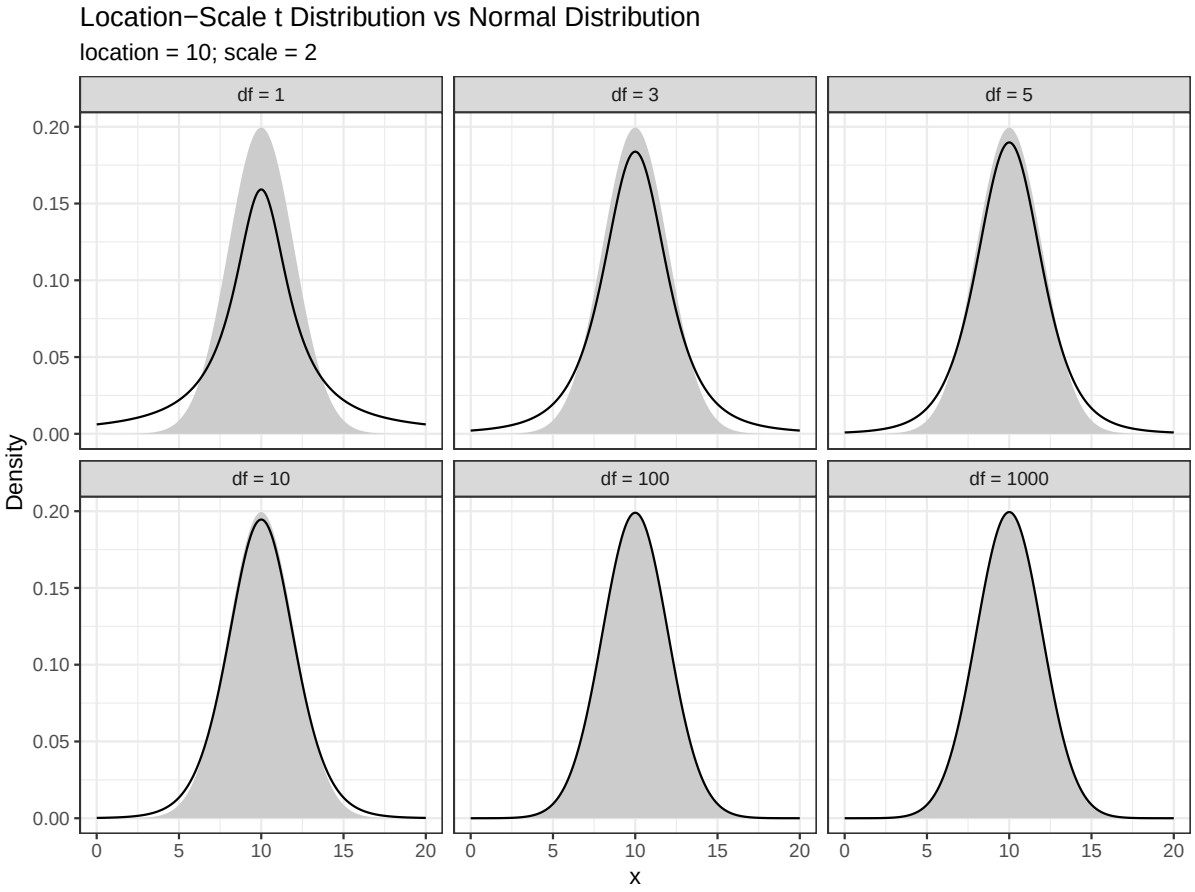
The location-scale t distribution is a flexible model for data that may exhibit heavier-than-normal tails. This is can be helpful when your data have more extreme observations than expected with a normal model.

The pdf of the location-scale t distribution is

$$f(y \mid \mu, \sigma, \nu) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\nu\pi}\,\sigma}\left[1 + \frac{1}{\nu}\left(\frac{y-\mu}{\sigma}\right)^2\right]^{-\frac{\nu+1}{2}},$$

where $\Gamma(\cdot)$ is the gamma function, μ is the location parameter (which shifts the distribution much like the mean of the normal distribution), σ is the scale parameter (which changes the spread of the distribution much like the SD parameter of the normal distribution), and ν controls the heaviness of the tails—a smaller value of ν produces a heavier tails and the distribution converges to the normal distribution as $\nu \to \infty$. For $\nu > 10$, the t and normal distributions are very similar. We refer to the special case of $\nu = 1$ as the Cauchy distribution.

The figure below compares the location-scale t and normal distributions. As x moves away from the center of the distribution, notice that that the t density remains well above zero for *much* longer when the df is less than five or so.



These heavy tails make the location-scale t distribution useful for robust modeling, where we want to capture central tendencies while being less sensitive to unusual values.

In R, we have the `metRology::dt.scaled()` function that computes the density or likelihood. Some important things to note about this function: the argument `mean` refers to the location parameter μ, though μ isn't always a mean. The argument `sd` refers to the scale parameter σ, though σ isn't the SD. The argument `df` refers to the parameter ν.

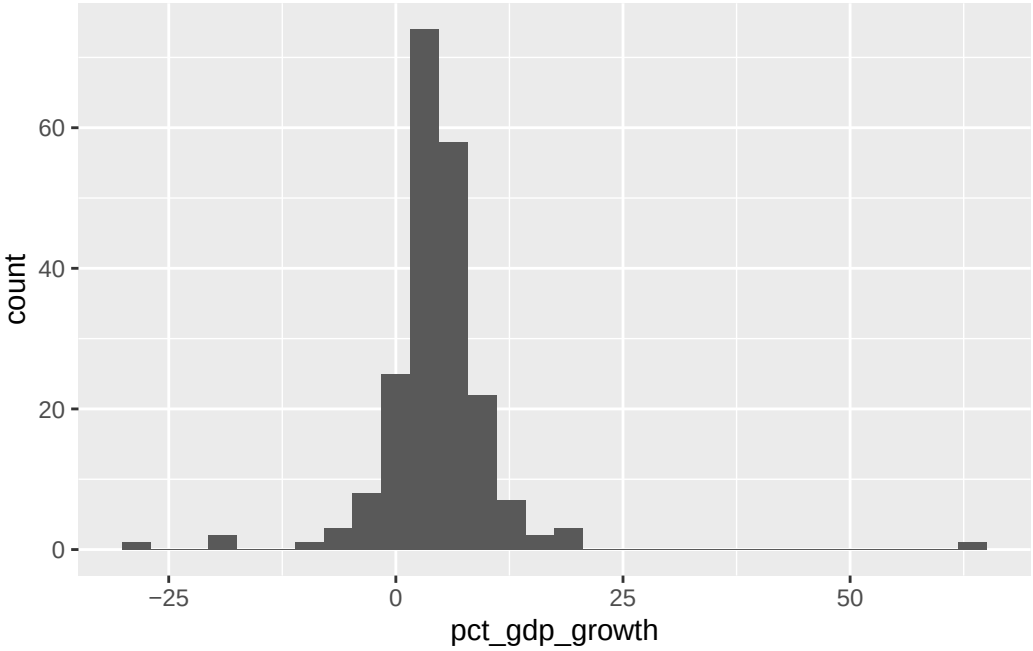
Some Questions

a. As a baseline, model the percentage GDP growth in the data below as using a normal model. Use the predictive distribution to assess the fit. What does the normal model miss?
b. As an alternative, try a location-scale t model. Use `optim()` to estimate the parameters μ, σ, and ν. Compare the predictive distribution for the t model to the normal model. (The `metRology::dt.scaled(..., log = TRUE)` function will compute the log-likelihood for the individual observations, and `metRology::rt.scaled()` will draw samples.)
c. What is the ML estimate of the degrees of freedom parameter? Discuss it's importance. What patterns can the t distribution capture that the normal distribution cannot?

```r
# load package
library(WDI)

# get annual % gdp growth (annual %) for 2022
# - note: "NY.GDP.MKTP.KD.ZG" is percentage gdp growth
#         see https://data.worldbank.org/indicator/NY.GDP.MKTP.KD.ZG
df <- WDI(indicator = "NY.GDP.MKTP.KD.ZG",
                      start = 2022,
                      end = 2022,
                      extra = TRUE) %>%
  # data includes aggregates (e.g., European Union); filter these out
  filter(region != "Aggregates") %>%
  # change variable name
  mutate(pct_gdp_growth = NY.GDP.MKTP.KD.ZG) %>%
  # select needed variables
  select(country, year, region, pct_gdp_growth) %>%
  # remove missing values
  na.omit() %>%
  glimpse()
```

```
Rows: 207
Columns: 4
$ country        <chr> "Afghanistan", "Albania", "Algeria", "American Samoa", ~
$ year           <int> 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2022, 2~
$ region         <chr> "South Asia", "Europe & Central Asia", "Middle East & N~
$ pct_gdp_growth <dbl> -6.240172, 4.826696, 3.600000, 1.735016, 9.564612, 3.04~
```

```r
# plot histogram
ggplot(df, aes(x = pct_gdp_growth)) +
  geom_histogram()
```



Solution

Part (a): Normal model

```
# create a function to fit the normal model
fit_normal_model <- function(y) {
  mu_hat <- mean(y)
  sigma2_hat <- sum((y - mu_hat)^2)/length(y)
  res <- c("mu_hat" = mu_hat,
           "sigma2_hat" = sigma2_hat,
           "sigma_hat" = sqrt(sigma2_hat))
  return(res)
}

# fit normal model and print estimates
normal_ests <- fit_normal_model(df$pct_gdp_growth)
normal_ests
```

```
    mu_hat sigma2_hat  sigma_hat
  4.478777  45.931083   6.777247
```

```
# compute ml estimates of lambda (exponential rate)
y <- df$pct_gdp_growth
n <- length(y)
```

```
# create observed data set
observed_data <- tibble(pct_gdp_growth = y, type = "observed") %>%
  glimpse()
```

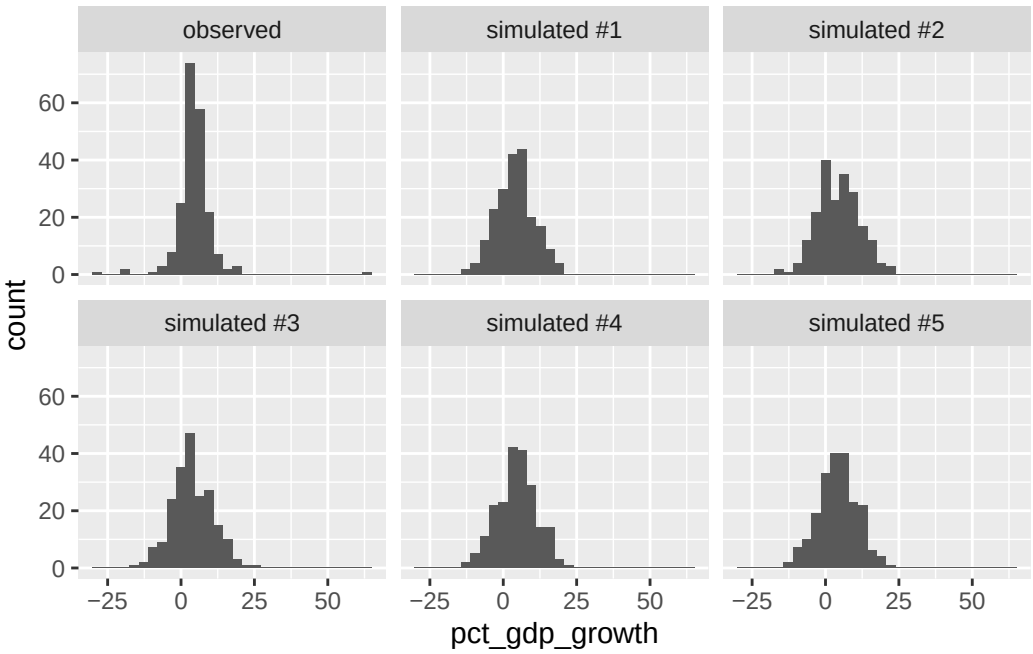
```
Rows: 207
Columns: 2
$ pct_gdp_growth <dbl> -6.240172, 4.826696, 3.600000, 1.735016, 9.564612, 3.04~
$ type           <chr> "observed", "observed", "observed", "observed", "observ~
```

```
# simulate five fake data sets
sim_list <- list()
for (i in 1:5) {
  y_pred <- rnorm(n, mean = normal_ests[1], sd = normal_ests[3])
  sim_list[[i]] <- tibble(pct_gdp_growth = y_pred,
                          type = paste0("simulated #", i))
}

# combine the fake and observed data sets
gg_data <- bind_rows(sim_list) %>%
  bind_rows(observed_data) %>%
  glimpse()
```

```
Rows: 1,242
Columns: 2
$ pct_gdp_growth <dbl> 4.6972129, 5.0900301, 15.6011669, 5.6945016, 3.0741227,~
$ type           <chr> "simulated #1", "simulated #1", "simulated #1", "simula~
```

```
# plot the observed and fake data sets
ggplot(gg_data, aes(x = pct_gdp_growth)) +
  geom_histogram() +
  facet_wrap(vars(type))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



In order to accommodate the extreme values, the normal distribution needs to have a larger SD. However, this larger SD means that the fitted distribution has too little clustering close to the average. The observed variable has more data points falling closer to the average and a few data points falling far from the average.

```
# load packages
library(metRology)  # for location-scale t density

# create log-likelihood for the location-scale t distribution
ll_t <- function(par, y) {
  # par contains par[1] = mu, par[2] = sigma, and par[3] = nu
  mu <- par[1]     # location
  sigma <- par[2]  # scale
  nu <- par[3]     # degrees of freedom (heaviness of tails)

  # evaluate if parameters are within bounds
  in_bounds <- nu > 0 & sigma > 0

  # evaluate log-likelihood
  ll <- ifelse(in_bounds,
               # if in bounds
               sum(dt.scaled(y, log = TRUE,
                      mean = mu, sd = sigma, df = nu)),
               # if not in bounds
```

```
                -Inf)
  return(ll)
}

# create a function to fit the t model
fit_t_model <- function(y) {

  # starting values
  mu_start <- median(y)
  sigma_start <- sd(y)
  nu_start <- 10
  init_par <- c(mu_start, sigma_start, nu_start)

  # optimize
  opt <- optim(par = init_par,
               fn = ll_t,
               y = y,
               method = "Nelder-Mead",
               control = list(fnscale = -1))

  # organize output
  res <- c("mu_hat" = opt$par[1],
           "sigma_hat" = opt$par[2],
           "nu_hat" = opt$par[3])
  return(res)
}

# fit model
t_ests <- fit_t_model(df$pct_gdp_growth); t_ests
```

```
   mu_hat sigma_hat    nu_hat
 4.321876  2.478035  1.826382
```

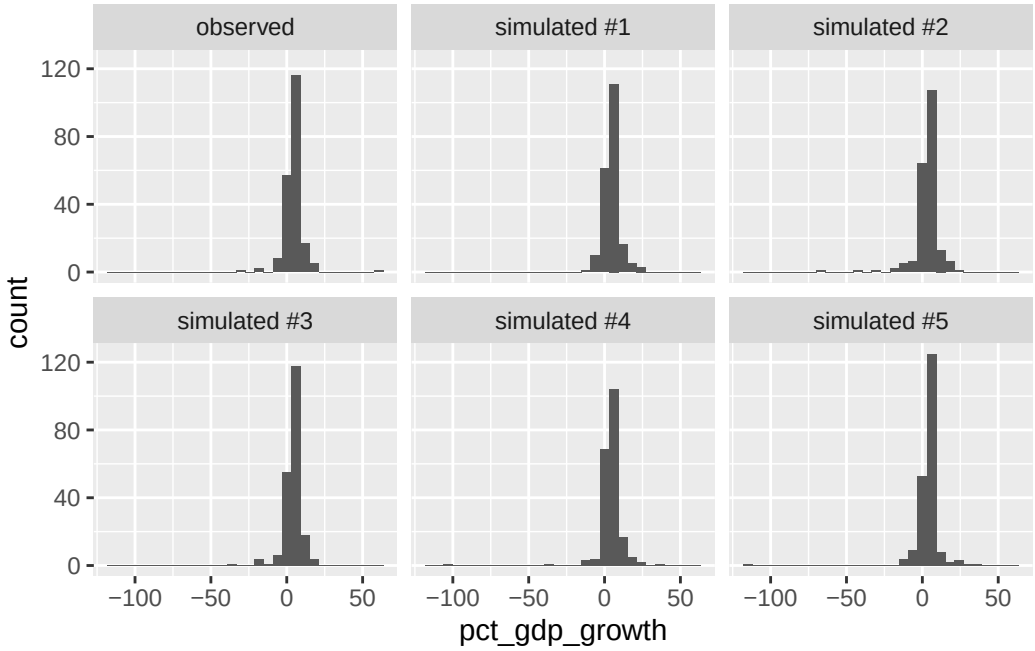
```
# simulate five fake data sets
sim_list <- list()
for (i in 1:5) {
  y_pred <- rt.scaled(n, mean = t_ests[1],
                         sd =   t_ests[2],
                         df =   t_ests[3])
  sim_list[[i]] <- tibble(pct_gdp_growth = y_pred,
                          type = paste0("simulated #", i))
```

```
}

# combine the fake and observed data sets
gg_data <- bind_rows(sim_list) %>%
  bind_rows(observed_data) %>%
  glimpse()
```

```
Rows: 1,242
Columns: 2
$ pct_gdp_growth <dbl> 6.8064589, 3.6182219, 1.6719102, 4.0425009, 7.4338720, ~
$ type           <chr> "simulated #1", "simulated #1", "simulated #1", "simula~
```

```
# plot the observed and fake data sets
ggplot(gg_data, aes(x = pct_gdp_growth)) +
  geom_histogram() +
  facet_wrap(vars(type))
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



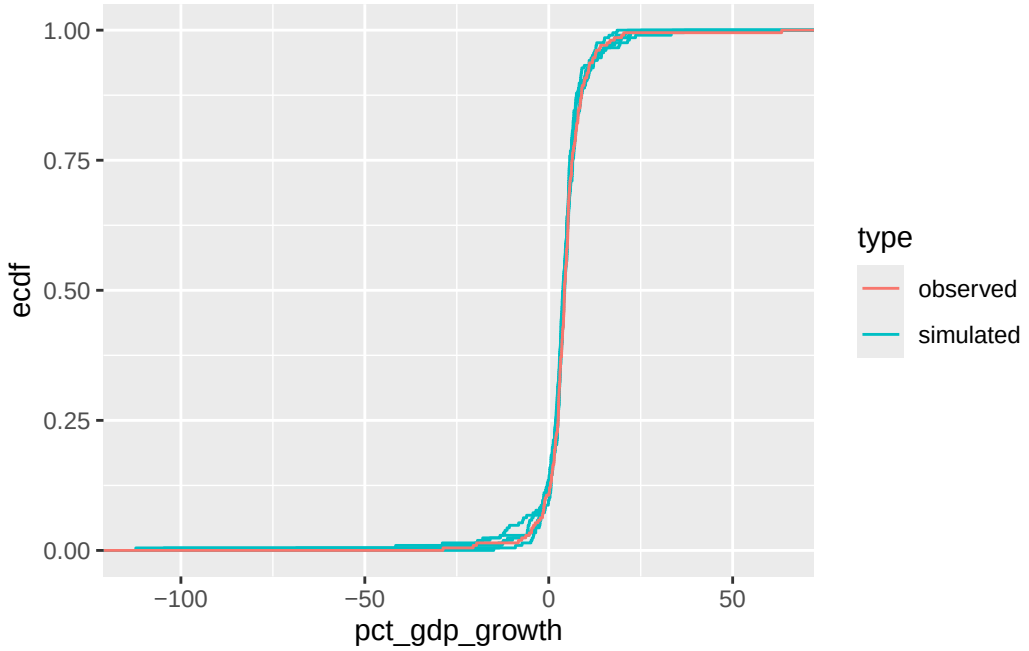
These fitted t distributions match the observed data quite a bit better. We put the ECDFs on the same panel to see that they match really well.

```r
# separate the labels of the simulated data into two parts
gg_data2 <- gg_data |>
  separate(type, into = c("type", "version")) |>
  glimpse()
```

```
Warning: Expected 2 pieces. Missing pieces filled with `NA` in 207 rows [1036, 1037,
1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050,
1051, 1052, 1053, 1054, 1055, ...].
```

```
Rows: 1,242
Columns: 3
$ pct_gdp_growth <dbl> 6.8064589, 3.6182219, 1.6719102, 4.0425009, 7.4338720, ~
$ type           <chr> "simulated", "simulated", "simulated", "simulated", "si~
$ version        <chr> "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", ~
```

```r
# plot histograms of real and fake data
ggplot(gg_data2, aes(x = pct_gdp_growth, color = type, group = version)) +
  stat_ecdf()
```



Exercise 12 Corrupted data

For the WDI data above, corrupt the data by replacing one observation with a severe data entry error (something like `pct_gdp_growth` of 10,000) and re-fit the normal and t models. How did the estimates of the *location* of each change? Why? Is this a desirable property?

Solution

```
# save variable as y (to simplify the code)
y <- df$pct_gdp_growth

# corrupt the data
y_corrupt <- y
y_corrupt[1] <- 10000

# fit normal models
fit_normal_model(y)
```

```
    mu_hat sigma2_hat  sigma_hat
  4.478777  45.931083   6.777247
```

```
fit_normal_model(y_corrupt)
```

```
     mu_hat  sigma2_hat   sigma_hat
    52.8181 480367.8380    693.0857
```

The data entry error shifts estimated mean in the normal model from about 4.4 to about 52. Given that we're talking about the percent GDP growth, this is a huge overestimate.

```
# fit t models
fit_t_model(y)
```

```
   mu_hat sigma_hat    nu_hat
 4.321876  2.478035  1.826382
```

```
fit_t_model(y_corrupt)
```

```
   mu_hat sigma_hat    nu_hat
 4.313909  2.242405  1.408114
```

The estimated mean in the t model is about 4.2 in both in both the correct data and the data set with a data entry errors.

The t model can have heavy tails, so it can easily dismiss the data entry error as a "weird draw" that isn't informative about the location.

This might be a desirable property for statistical or substantive reasons. If we have data entry errors, then it's good that our estimates don't swing wildly in response. But it's also possible that some cases in the data set are correctly measured, but wildly different that the other, "typical" cases that our theory has in mind. So substantively, we might want a model that allows these "very unusual cases" to appear in the data set without pulling the estimates far from the typical values.

Exercise 13 *Optional*: R function to simulate the predictive distribution

Note: This is a challenging function to write, so only attempt this problem if you are comfortable with writing other, simpler functions.

We've use my code to simulate the predictive distribution a few times and combine them together with the observed data set really long and repetitive. In his excellent *R for Data Science 2(3)*, Hadley Wickham writes: "A good rule of thumb is to consider writing a function whenever you've copied and pasted a block of code more than twice (i.e. you now have three copies of the same code)."

The inputs might be (1), the name of the outcome variable, (2) an observed data set, (3) a *function* to simulate the fake data sets, and (4) the number of fake data sets to generate.

```
sim_fake <- function(variable, observed, sim_fn, n = 5) {
  ...
}
```

Exercise 14 Reflection

The exercises above ask you to compare fitted distributions to observed data. In some cases, we saw close correspondence between the models and the data. In other cases, the two diverged substantially. You may have noticed that even the poorly fitting distributions tend to have the same mean as the observed data. Is it important that our models mimic the other features of the data? To what extent is the mean the most important feature (or the only important feature) of the distribution? When and why might we care about the other features substantively (e.g., SD, heavy tails, memorylessness)? Why might we care about these other features statistically?