CS 1675 Homework: 03

Assigned: February 3, 2020; Due: February 17, 2020

Prayush Luitel

Submission time: February 17, 2020 at 9:00PM

Collaborators Include the names of your collaborators here.

Overview

This homework assignment has two primary goals. First, you will program the Laplace Approximation for the Gaussian likelihood with an unknown mean and unknown noise. Second, you will be introduced to resampling. You will use 5-fold cross-validation to identify that a more complex model is overfitting to the training set, and is thus fooled by noise.

Completing this assignment requires filling in missing pieces of information from existing code chunks, programming complete code chunks from scratch, typing discussions about results, and working with LaTeX style math formulas. A template .Rmd file is available to use as a starting point for this homework assignment. The template is available on CourseWeb.

IMPORTANT: Please pay attention to the `eval` flag within the code chunk options. Code chunks with `eval=FALSE` will **not** be evaluated (executed) when you Knit the document. You **must** change the `eval` flag to be `eval=TRUE`. This was done so that you can Knit (and thus render) the document as you work on the assignment, without worrying about errors crashing the code in questions you have not started. Code chunks which require you to enter all of the required code do not set the `eval` flag. Thus, those specific code chunks use the default option of `eval=TRUE`.

Load packages

This assignment uses the `dplyr` and `ggplot2` packages, which are loaded in the code chunk below. The resampling questions will make use of the `modelr` package, which we will load later in this report. The assignment also uses the `tibble` package to create tibbles, and the `readr` package to load CSV files. All of the listed packages are part of the `tidyverse` and so if you downloaded and installed the `tidyverse` already, you will have all of these packages. This assignment will use the `MASS` package to generate random samples from a MVN distribution. The `MASS` package should be installed with base R, and is listed with the System Library set of packages.

```
library(dplyr)
library(ggplot2)
```

Problem 1

You are tasked by a manufacturing company to study the variability of a component that they produce. This particular component has an important figure of merit that is measured on every single manufactured piece.

The engineering teams feel that the measurement is quite noisy. Your goal will be to learn the unknown mean and unknown noise of this figure of merit based on the noisy observations.

The data set that you will work with is loaded in the code chunk below and assigned to the `df_01` object. As the `glimpse()` function shows, `df_01` consists of two variables, `obs_id` and `x`. There are 405 rows or observations. `obs_id` is the "observation index" and `x` is the measured value.

```
df_01 <- readr::read_csv("https://raw.githubusercontent.com/jyurko/CS_1675_Spring_2020/master/hw_data/h
                         col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   obs_id = col_double(),
##   x = col_double()
## )
```

```
df_01 %>% glimpse()
```

```
## Observations: 405
## Variables: 2
## $ obs_id <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,...
## $ x      <dbl> 7.3299610, 4.0119121, -0.5633241, -8.2168261, -1.3070010, 1.44...
```

1a)

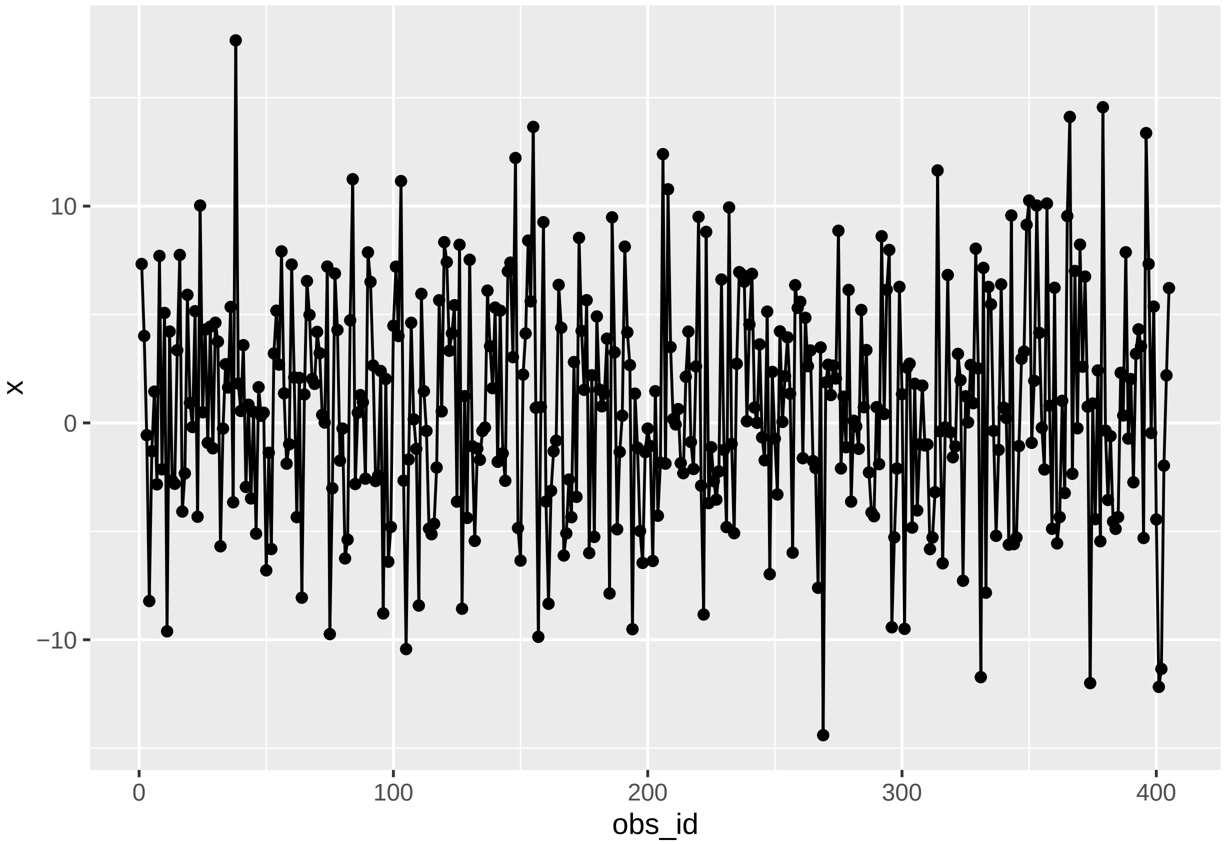
Let's start out by visualizing and summarizing the available measurements.

PROBLEM **Create a "run chart" of the measurments by piping the `df_01` data set into `ggplot()`. Map the `x` aesthetic to `obs_id` and the `y` aesthetic to `x`. Use `geom_line()` and `geom_point()` geometric objects to display the measured value with respect to the observation index.**

Visually, do any values appear to stand out from the "bulk" of the measurements?

SOLUTION Your solution here.

```
### run chart of measurements
df_01 %>%
  ggplot(mapping = aes(x = obs_id, y = x)) +
  geom_line() +
  geom_point(mapping = aes(x = obs_id))
```



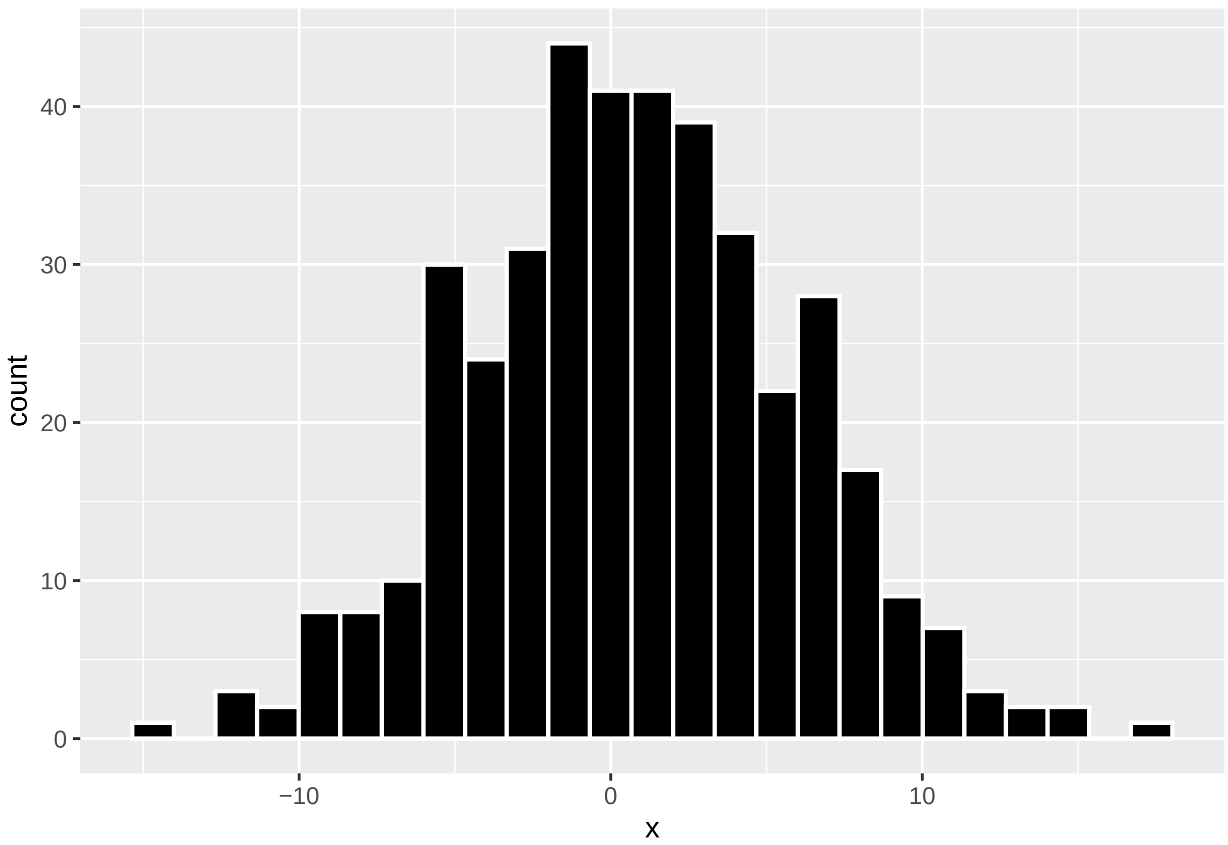
1b)

PROBLEM **Create a histogram with `geom_histogram()`, by mapping the `x` aesthetic to the variable `x`. Set the number of bins to be 25.**

Based on the histogram and the run chart, where are most of the measurement values concentrated? What shape does the distribution look like?

SOLUTION Your solution here.

```
### run chart of measurements
df_01 %>%
  ggplot(mapping = aes(x = x)) +
  geom_histogram(bins= 25, color = "white", fill = "black", size = .75)
```



Based on the run chart and the histogram, most of the values are concentrated around the middle or the value 0. This suggests that the shape of the distribution is close to a Gaussian.

1c)

PROBLEM Use the `summary()` function to summarize the `df_01` data set. Then calculate the 5th and 95th quantile estimates of the variable `x`. Lastly, calculate the standard deviation on the `x` variable. Are the summary statistics in line with your conclusions based on the run chart and histogram?

HINT: The quantiles can be estimated via the `quantile()` function. Type `?quantile` into the `R` Console for help with the arguments to the quantile function.

```
### apply the summary function
df_01 %>%
  summary()
```

SOLUTION

```
##      obs_id           x
##  Min.   :  1   Min.   :-14.4013
##  1st Qu.:102   1st Qu.: -2.6592
##  Median :203   Median :  0.6984
##  Mean   :203   Mean   :  0.7837
##  3rd Qu.:304   3rd Qu.:  4.2237
##  Max.   :405   Max.   : 17.6454
```

```
### calculate the quantile estimates
quantile(df_01 $x, c(0.05, 0.95))
```

```
##        5%       95%
## -7.785702  9.234694
```

```
### calculate the empirical standard deviation of x
sd(df_01 $x)
```

```
## [1] 5.158638
```

Based on the summary statistics, the value of x is centered around 0 as given by the mean and median. Furthermore, the 5% and 95% quantiles suggests that 90% of the values are within -7.79 and 9.23. This is in line with my conclusion based on the run chart and histogram as the distribution is representative of the Gaussian.

1d)

The summary statistics in Problem 1c) were estimated by using all of the 405 observations. The manufacturing company produces several other components which could also be measured. It will take a considerable effort to measure the same number of components as are available in the current data set. Therefore, it is important to understand the behavior of the sample average with respect to the number of observations.

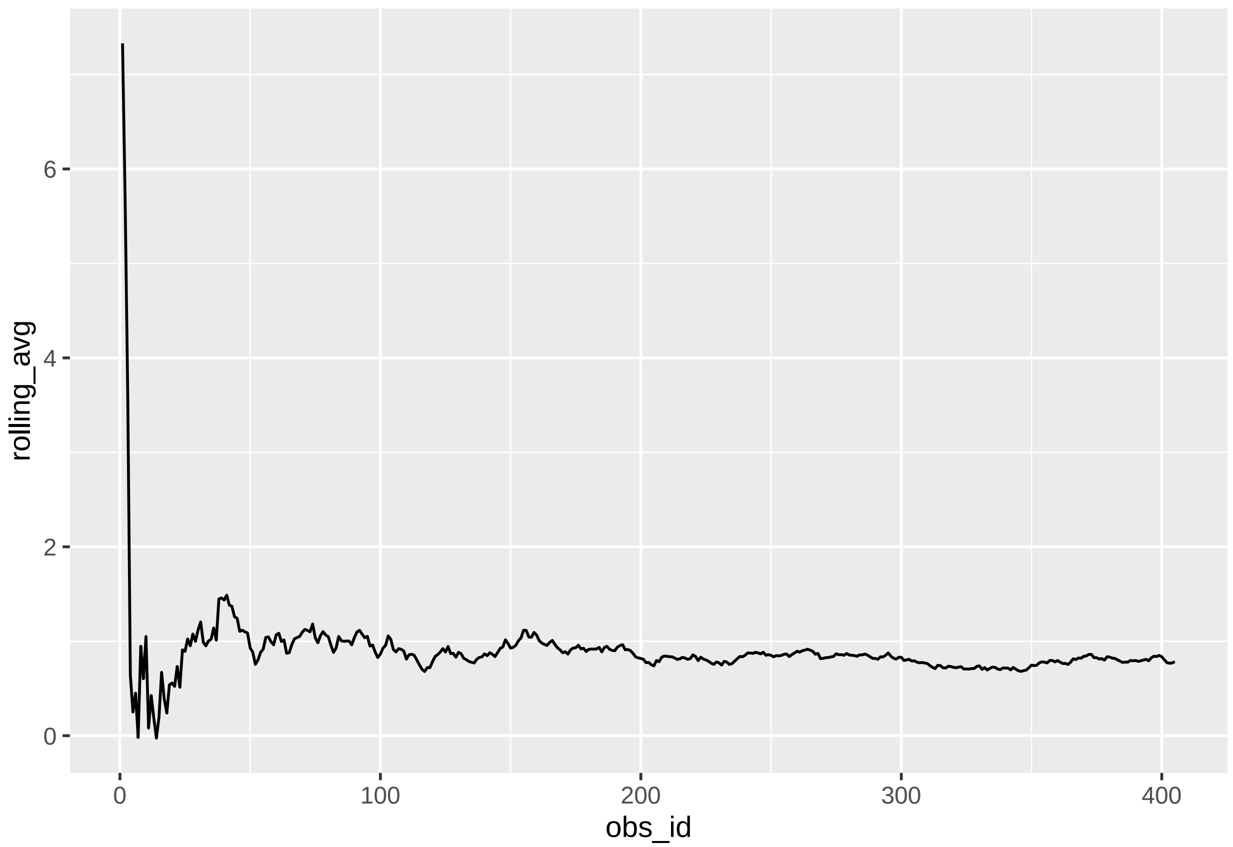
In `dplyr` we can calculate "rolling" sample averages with the `cummean()` function. This function "accumulates" all of the observations up to a current index and calculates the sample average. We can call the `cummean()` function directly within a `mutate()` call, which allows considerable flexibility, since we do not have use for-loops to create this operation. Let's visualize how the "rolling sample average" changes over time.

PROBLEM Pipe the `df_01` data set into a `mutate()` call. Create a new column (variable) in the data set, `rolling_avg`, which is equal to the rolling average of `x`. Pipe the result into `ggplot()` and map the `x` aesthetic to `obs_id` and the `y` aesthetic to `rolling_avg`. Display the rolling average with respect to the observation index with the `geom_line()` geometric object.

Which portion of the rolling average appears the most variable? Does the rolling average steady-out at all?

SOLUTION Your solution here.

```
df_01 %>%
  mutate(rolling_avg = cummean(x)) %>%
  ggplot(mapping = aes(x = obs_id, y= rolling_avg)) + geom_line()
```



The first fifty observations in the rolling average seem to be the most variable but after this and definately by 200 observations the rolling average is more constant, with small variations.

1e)

In this problem, you will focus on two specific sample sizes. You will calculate the sample average after 5 observations and 45 observations.

PROBLEM What does the sample average equal to after 5 observations? What does the sample average equal to after 45 observations?

SOLUTION Your solution here.

```
mydf <- subset(df_01, obs_id <6)
mean(mydf$x)
```

```
## [1] 0.2509444
```

```
mydf <- subset(df_01, obs_id <46)
mean(mydf$x)
```

```
## [1] 1.242825
```

Problem 2

With a simple Exploratory Data Analysis (EDA) completed, it's time to begin modeling. You are interested in learning the unknown mean of the figure of merit and the unknown noise in the measurement process. Because the manufacturing company is interested in understanding the behavior of the estimates "over time", it will be critical to assess uncertainty in the unknowns at low sample sizes. Therefore, you use a Bayesian approach to estimating the unknown mean, μ, and unknown noise, σ.

The figure of merit, x, is a continuous variable, and after discussions with the engineering teams, it seems that a Guassian likelihood is a reasonable assumption to make. You will assume that each observation is conditionally independent of the others given μ and σ. You will also assume that the two unknowns are a-priori independent. The joint posterior distribution on μ and σ conditioned on the observations $\mathbf{x}$ is therefore proportional to:

$$p\left(\mu, \sigma \mid \mathbf{x}\right) \propto \prod_{n=1}^{N} \left(\text{normal}\left(x_n \mid \mu, \sigma\right)\right) \times p\left(\mu\right) \times p\left(\sigma\right)$$

2a)

You will be applying the Laplace Approximation to learn the unknown μ and σ. In lecture, and in homework 02, it was discussed that it is a good idea to first transform σ to a new variable φ, before executing the Laplace Approximation.

PROBLEM Why is it a good idea to transform σ before performing the Laplace Approximation?

SOLUTION The Laplace approximation approximates the posterior distribution as a Gaussian/normal. However, this is not appropriate because sigma has a natural lower bound of 0 but a Gaussian allows any positive or negative value. Thus, the Laplace approximation does not keep the lower bound, and therefore we need to use a transformation from sigma to varphi to respect the lower bound of 0, so that if the value of sigma needed to be negative, it could.

2b)

Consider a general transformation, or link function, applied to the unknown σ:

$$\varphi = g\left(\sigma\right)$$

The inverse link function which allows "back-transforming" from φ to σ is:

$$\sigma = g^{-1}\left(\varphi\right)$$

PROBLEM Write out the joint-posterior between the unknown μ and unknown transformed parameter φ by accounting for the probability change-of-variables formula. You must keep the prior on μ and σ general for now, and make use of the general inverse link function notation. Denote the likelihood as a Gaussian likelihood, as shown in the Problem 2 description.

SOLUTION

$$p\left(\mu, \varphi \mid \mathbf{x}\right) \propto \prod_{n=1}^{N} \left(\mathrm{normal}\left(x_n \mid \mu, g^{-1}\left(\varphi\right)\right)\right) \times p\left(\mu\right) \times p\left(g^{-1}\left(\varphi\right)\right) \times \left(\left| \frac{d}{d\varphi}\left(g^{-1}\left(\varphi\right)\right)\right|\right)$$

2c)

After discussions with the engineering teams at the manufacturing company, it was decided that a normal prior on μ and an Exponential prior on σ are appropriate. The normal prior is specified in terms of a prior mean, μ_0, and prior standard deviation, τ_0. The Exponential prior is specified in terms of a rate parameter, λ. The prior distributions are written out for you below.

$$p\left(\mu\right) = \mathrm{normal}\left(\mu \mid \mu_0, \tau_0\right)$$

$$p\left(\sigma\right) = \mathrm{Exp}\left(\sigma \mid \lambda\right)$$

Because you will be applying the Laplace approximation, you decide to use a log-transformation as the link function:

$$\varphi = \log\left(\sigma\right)$$

The engineering team feels fairly confident that the measurement process is rather noisy. They feel the noise is obscuring a mean figure of merit value that is actually negative. For that reason, it is decided to set the hyperparameters to be $\mu_0 = -1/2$, $\tau_0 = 1$, and $\lambda = 1/3$. You are first interested in how their prior beliefs change due to the first 5 observations.

You will define a function to calculate the un-normalized log-posterior for the unknown μ and φ. You will follow the format discussed in lecture, and so the hyperparameters and measurements will be supplied within a list. The list of the required information is defined for you in the code chunk below. The hyperparameter values are specified, as well as the first 5 observations are stored in the variable `xobs`. The list is assigned to the object `info_inform_N05` to denote the prior on the unknown mean is informative and the first 5 observations are used.

```
info_inform_N05 <- list(
  xobs = df_01$x[1:5],
  mu_0 = -0.5,
  tau_0 = 1,
  sigma_rate = 1/3
)
```

PROBLEM You must complete the code chunk below in order to define the `my_logpost()` function. This function is in terms of the unknown mean μ and unknown transformed parameter φ. Therefore you MUST account for the change-of-variables transformation. The first argument to `my_logpost()` is a vector, `unknowns`, which contains all of the unknown parameters to the model. The second argument `my_info`, is the list of required information. The unknown mean and unknown transformed parameter are extracted from `unknowns`, and are assigned to the `lik_mu` and `lik_varphi` variables, respectively. You must complete the rest of the code chunk. Comments provide hints for what calculations you must perform.

You ARE allowed to use built-in `R` functions to evaluate densities.

HINT: the `info_inform_N05` list of information provides to you the names of the hyperparameter and observation variables to use in the `my_logpost()` function. This way all students will use the same nomenclature for evaluating the log-posterior.

HINT: The second code chunk below provides three test cases for you. If you correctly coded the `my_logpost()` function, the first test case which sets `unknowns = c(0, 0)` will yield a value of `-76.75337`. The second test case which sets `unknowns = c(-1, -1)` yields a value of `-545.4938`. The third test case which sets `unknowns = c(1, 2)` yields a value of `-19.49936`. If you do not get three values very close to those printed here, there is a typo in your function.

```
my_logpost <- function(unknowns, my_info)
{
  # extrack the unknowns
  lik_mu <- unknowns[1]
  lik_varphi <- unknowns[2]

  # backtransform to sigma
  lik_sigma <-  exp(lik_varphi)

  # calculate the log-likelihood
  log_lik <- sum(dnorm(x=my_info$xobs, mean = lik_mu, sd= lik_sigma, log = TRUE))

  # calculate the log-prior on each parameter
  log_prior_mu <- dnorm(x = lik_mu, mean = my_info$mu_0, sd = my_info$tau_0, log = TRUE)

  log_prior_sigma <- dexp(x = lik_sigma, rate = my_info$sigma_rate, log = TRUE)

  # calculate the log-derivative adjustment due to the
  # change-of-variables transformation
  log_deriv_adjust <- lik_varphi

  # sum all together
  log_lik + log_prior_mu + log_prior_sigma + log_deriv_adjust
}
```

SOLUTION Check test cases.

```
my_logpost(c(0, 0), info_inform_N05)
```

```
## [1] -76.75337
```

```
my_logpost(c(-1, -1), info_inform_N05)
```

```
## [1] -545.4938
```

```
my_logpost(c(1, 2), info_inform_N05)
```

```
## [1] -19.49936
```

2d)

Because this problem consists of just two unknowns, we can graphically explore the true log-posterior surface. We did this in lecture by studying in both the original (μ, σ) space, as well as the transformed and unbounded (μ, φ) space. However, in this homework assignment, you will continue to focus on the transformed parameters μ and φ.

In order to visualize the log-posterior surface, you must define a grid of parameter values that will be applied to the `my_logpost()` function. A simple way to create a full-factorial grid of combinations is with the `expand.grid()` function. The basic syntax of the `expand.grid()` function is shown in the code chunk below for two variables, `x1` and `x2`. The `x1` variable is a vector of just two values `c(1, 2)`, and the variable `x2` is a vector of 3 values `1:3`. As shown in the code chunk output printed to the screen the `expand.grid()` function produces all 6 combinations of these two variables. The variables are stored as columns, while their combinations correspond to a row within the generated object. The `expand.grid()` function takes care of the "book keeping" for us, to allow varying `x2` for all values of `x1`.

```
expand.grid(x1 = c(1, 2),
            x2 = 1:3,
            KEEP.OUT.ATTRS = FALSE,
            stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tbl_df()
```

```
## # A tibble: 6 x 2
##      x1    x2
##   <dbl> <int>
## 1     1     1
## 2     2     1
## 3     1     2
## 4     2     2
## 5     1     3
## 6     2     3
```

You will now make use of the `expand.grid()` function to create a grid of candidate combinations between μ and φ.

PROBLEM Complete the code chunk below. The variables `mu_lwr` and `mu_upr` correspond to the lower and upper bounds in the grid for the unknown μ parameter. Specifiy the lower bound to be the 0.01 quantile (1st percentile) based on the prior on μ. Specify the upper bound to the 0.99 quantile (99th percentile) based on the prior on μ. The variables `varphi_lwr` and `varphi_upr` are the lower and upper bounds in the grid for the unknown φ parameter. Specify the lower bound to be the log of the 0.01 quantile (1st percentile) based on the prior on σ. Specificy the upper bound to be the log of the 0.99 quantile(99th percentile) based on the prior on σ.

Create the grid of candidate values, `param_grid`, by setting the input vectors to the `expand.grid()` function to be 201 evenly spaced points between the defined lower bound and upper bounds on the paraemters.

HINT: remember that probability density functions in `R` each have their own specific quantile functions...

```


### bounds on mu
mu_lwr <- qnorm(0.01, mean = info_inform_N05$mu_0, sd= info_inform_N05$tau_0)
```

```r
mu_upr <- qnorm(0.99, mean = info_inform_N05$mu_0, sd= info_inform_N05$tau_0)

### bounds on varphi
varphi_lwr <- log(qexp(0.01, rate = info_inform_N05$sigma_rate))
varphi_upr <- log(qexp(0.99, rate = info_inform_N05$sigma_rate))

### create the grid
param_grid <- expand.grid(mu = seq(from = mu_lwr, to = mu_upr, length.out = 201) ,
                          varphi = seq(from = varphi_lwr, to= varphi_upr, length.out = 201),
                          KEEP.OUT.ATTRS = FALSE,
                          stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tbl_df()
```

SOLUTION

2e)

The `my_logpost()` function accepts a vector as the first input argument, `unknowns`. Thus, you cannot simply pass in the separate columns of the `param_grid` data set. To overcome this, you will define a "wrapper" function, which manages the call to the log-posterior function. The wrapper, `eval_logpost()` is started for you in the code chunk below. The first argument to `eval_logpost()` is a value for the unknown mean, the second argument is a value to the unknown φ parameter, the third argument is the desired log-posterior function, and the fourth argument is the list of required information.

This problem tests that you understand how to call a function, and how the input arguments to the log-posterior function are structured. You will need to understand that structure in order to perform the Laplace Approximation later on.

PROBLEM Complete the code chunk below such that the user supplied `my_func` function has the `mu_val` and `varphi_val` variables combined into a vector with the correct order.

Once you complete the `eval_logpost()` function, the second code chunk below uses the `purrr` package to calculate the log-posterior over all combinations in the grid. The result is stored in a vector `log_post_inform_N05`.

```r
eval_logpost <- function(mu_val, varphi_val, my_func, my_info)
{
  my_func( c(mu_val, varphi_val), my_info)
}
```

```r
log_post_inform_N05 <- purrr::map2_dbl(param_grid$mu,
                                       param_grid$varphi,
                                       eval_logpost,
                                       my_func = my_logpost,
                                       my_info = info_inform_N05)
```

SOLUTION

2f)

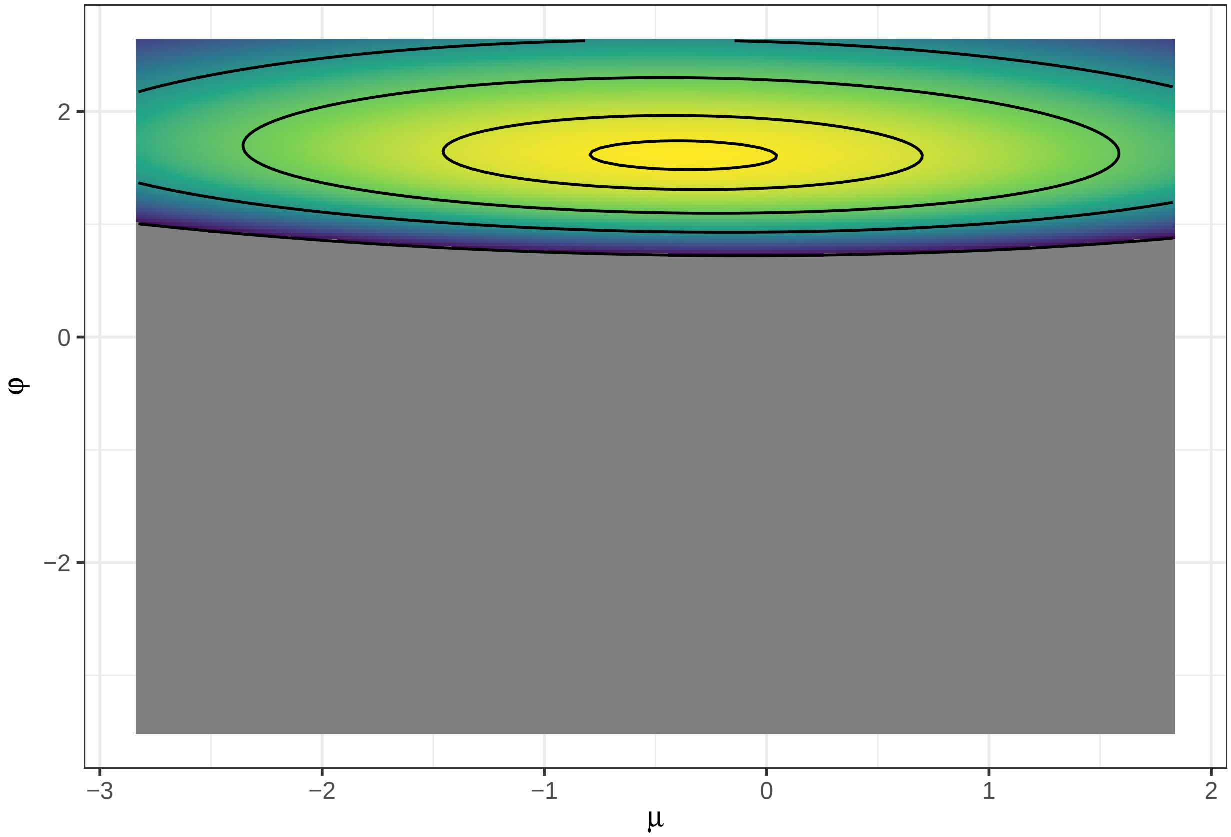
The code chunk below defines the `viz_logpost_surface()` function for you. It generates the log-posterior surface contour plots in the style presented in lecture. You are required to interpret the log-posterior surface and describe the most probable values and uncertainty in the parameters.

```r
viz_logpost_surface <- function(log_post_result, grid_values)
{
  gg <- grid_values %>%
    mutate(log_dens = log_post_result) %>%
    mutate(log_dens_2 = log_dens - max(log_dens)) %>%
    ggplot(mapping = aes(x = mu, y = varphi)) +
    geom_raster(mapping = aes(fill = log_dens_2)) +
    stat_contour(mapping = aes(z = log_dens_2),
                 breaks = log(c(0.01/100, 0.01, 0.1, 0.5, 0.9)),
                 color = "black") +
    scale_fill_viridis_c(guide = FALSE, option = "viridis",
                         limits = log(c(0.01/100, 1.0))) +
    labs(x = expression(mu), y = expression(varphi)) +
    theme_bw()

  print(gg)
}
```

PROBLEM Call the `viz_logpost_surface()` function by setting the `log_post_result` argument equal to the `log_post_inform_N05` vector, and the `grid_values` argument equal to the `param_grid` object.

Which values of the transformed unbounded φ parameter are completely ruled out, even after 5 observations? Which values for the unknown mean, μ, appear to be the most probable? Is the posterior uncertainty on μ small compared to the prior uncertainty?

```r
viz_logpost_surface(log_post_result = log_post_inform_N05, grid_values = param_grid)
```



SOLUTION

Any value less than approximately 1 for the transformed unbounded varphi parameter are ruled out in the case with five observations. This is because any value of mu with a varphi value less than 1 is greyed out in the graph above. Also, values between -.5 and 0.5 appear to be the most probable for the unknown mean mu, as the yellow filled section of the graph has a width around this range. With just 5 observations, the posterior uncertainty is much larger than the prior uncertainty.

2g)

The log-posterior visualization in Problem 2f) is based just on the first 5 observations. You must now repeat the analysis with 45 and all 405 observations. The code chunk below defines the two lists of required information, assuming the informative prior on μ. You will use these two lists instead of the `info_inform_N05` to recreate the log-posterior surface contour visualization.

```
info_inform_N45 <- list(
  xobs = df_01$x[1:45],
  mu_0 = -0.5,
  tau_0 = 1,
  sigma_rate = 1/3
)

info_inform_N405 <- list(
  xobs = df_01$x,
  mu_0 = -0.5,
  tau_0 = 1,
  sigma_rate = 1/3
)
```

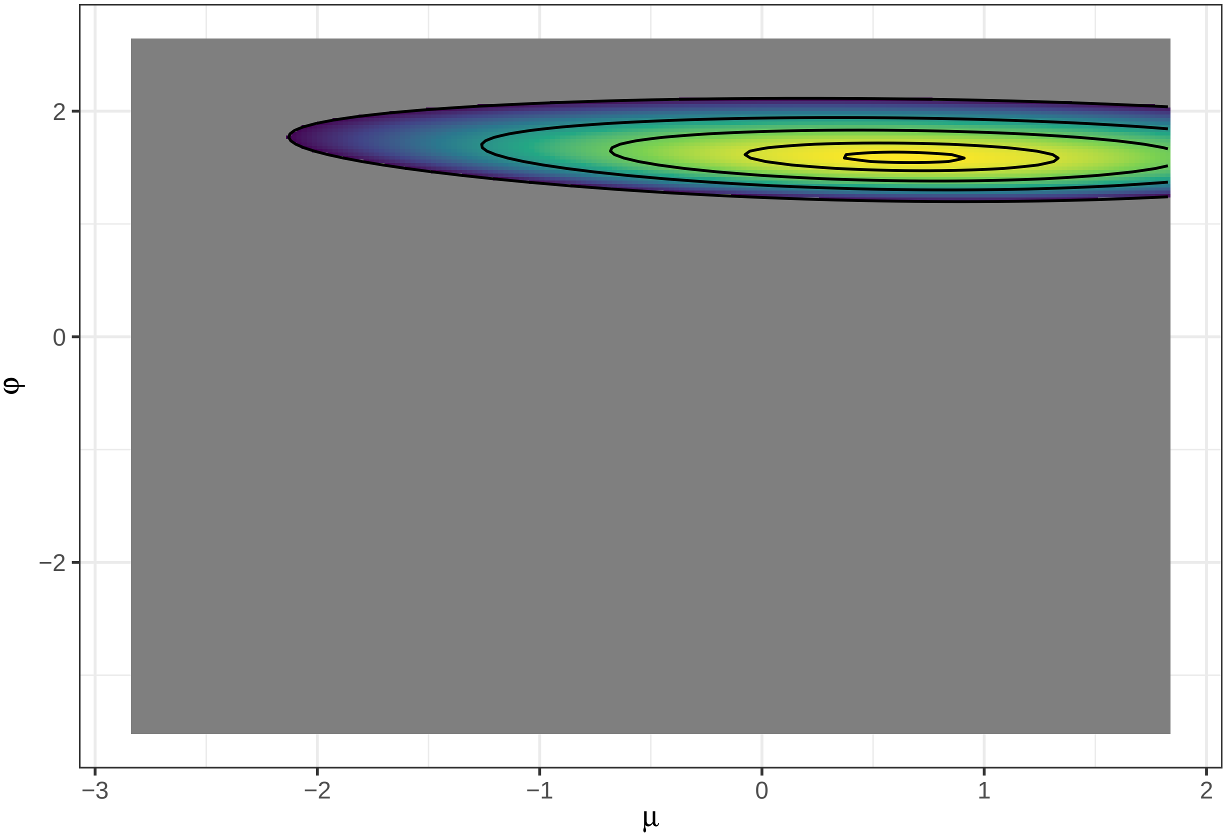
PROBLEM Complete the two code chunks below. The first code chunk requires you to complete the `purrr::map2_dbl()` call, which evaluates the log-posterior for all combinations of the parameters. By careful that you use the correct list of required information. The `log_post_inform_N45` object corresponds to the 45 sample case, while the `log_post_info_N405` object corresponds to using all of the samples.

The second and third code chunks require that you call the `viz_logpost_surface()` function for the posterior based on 45 samples, and the posterior based on all 405 samples, respectively.

How does the log-posterior surface change as the sample size increases? Does the most probable values of the two parameters change as the sample size increases? Does the uncertainty in the parameters decrease?

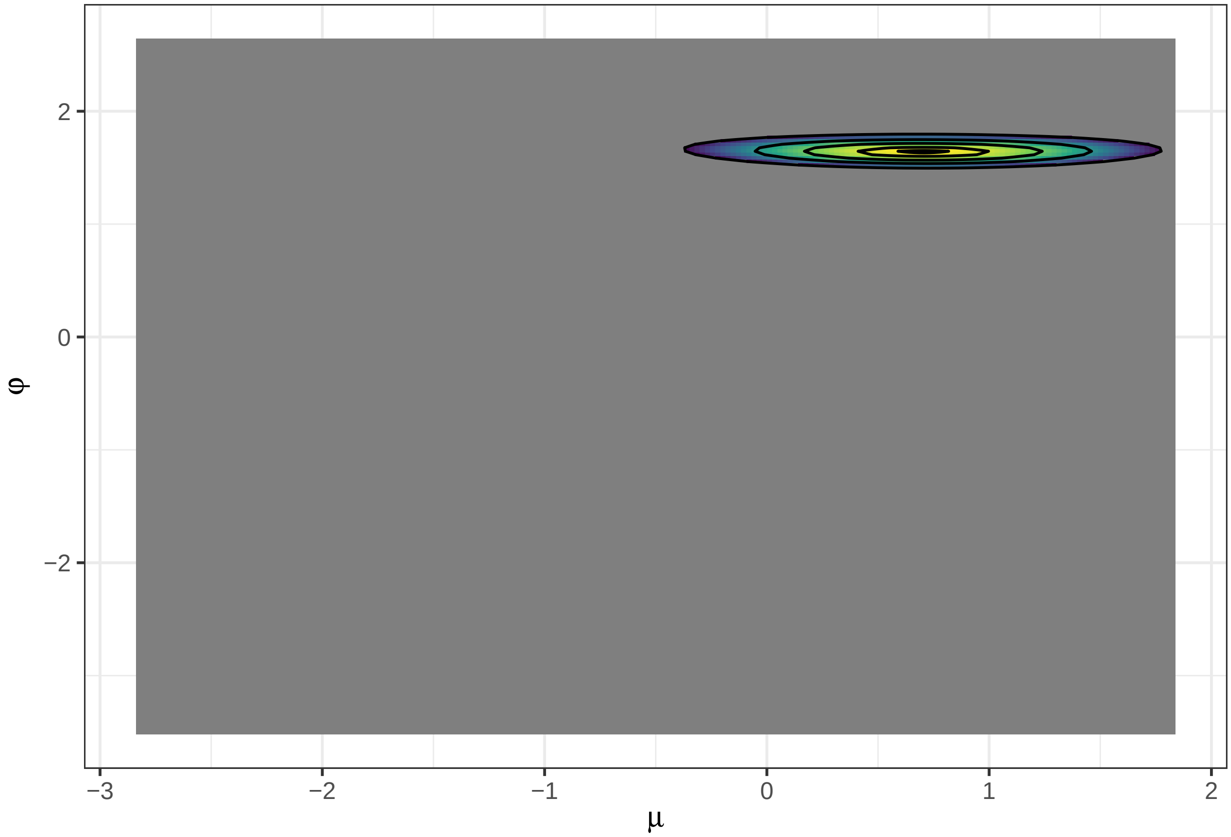
```
### check the arguments from param_grid!!!
log_post_inform_N45 <- purrr::map2_dbl(param_grid$mu,
                                       param_grid$varphi,
                                       eval_logpost,
                                       my_func = my_logpost,
                                       my_info = info_inform_N45)

### check the arguments from param_grid!!!
log_post_inform_N405 <- purrr::map2_dbl(param_grid$mu,
                                        param_grid$varphi,
                                        eval_logpost,
                                        my_func = my_logpost,
                                        my_info = info_inform_N405)
```

```
viz_logpost_surface(log_post_result = log_post_inform_N45, grid_values = param_grid)
```



SOLUTION

```
viz_logpost_surface(log_post_result = log_post_inform_N405, grid_values = param_grid)
```



The log posterior surface becomes much more restricted as the sample size increases. With just 5 observations, there was a large combination of values for mu and varphi that could be plausible. However, as the sample size increased to 45 and eventually 405, the combination of values for mu and varphi that could be plausible significantly declined. As the sample size increased the most probable values for varphi changed from being greater than 1.5 to closer to 1.5 to being very much 1.75. Similarly, the values for mu went from a range between -0.5 to 0.5 with the most probable value being 0 to being close to 0.5 to finally being 0.6. Thus, as the sample size increases, the uncertainty in the parameters decrease.

Problem 3

It's now time to work with the Laplace Approximation. The first step is to find the posterior mode, or the maximum a posteriori (MAP) estimate. An iterative optimization scheme is required to do so for most of the posteriors you will use in this course. As described in lecture, you will use the `optim()` function to perform the optimization.

3a)

The code chunk below defines two different initial guesses for the unknowns. You will try out both initial guesses and compare the optimization results. You will focus on the 5 observation case.

```
init_guess_01 <- c(-1, -1)
init_guess_02 <- c(2, 2)
```

PROBLEM Complete the two code chunks below. The first code chunk finds the posterior mode (the MAP) based on the first initial guess `init_guess_01` and the second code chunk uses the second initial guess `init_guess_02`. You must fill in the arguments to the `optim()` call in order to approximate the posterior based on 5 observations.

To receive full credit you must: specify the initial guesses correctly, specify the function to be optimized correctly, specify the gradient evaluation correctly, correctly pass in the list of required information for 5 samples, tell `optim()` to return the hessian matrix, and ensure that `optim()` is trying to maximize the log-posterior rather than attempting to minimize it.

```
map_res_01 <- optim(init_guess_01,
                    my_logpost,
                    gr = NULL,
                    info_inform_N05,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
map_res_01
```

SOLUTION

```
## $par
## [1] -0.374417  1.607401
##
## $value
## [1] -17.54464
##
## $counts
## function gradient
##       22       10
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]        [,2]
## [1,] -1.2008166  -0.2511661
## [2,] -0.2511661 -12.9898380
```

```
map_res_02 <- optim(init_guess_02,
                    my_logpost,
                    gr = NULL,
                    info_inform_N05,
                    method = "BFGS",
                    hessian = TRUE,
                    control = list(fnscale = -1, maxit = 1001))
map_res_02
```

```
## $par
## [1] -0.3744223  1.6074010
##
## $value
## [1] -17.54464
##
## $counts
## function gradient
##       23        8
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]        [,2]
## [1,] -1.2008164  -0.2511679
## [2,] -0.2511679 -12.9898294
```

3b)

You tried two different starting guesses. . . will you get the same optimized results?

PROBLEM Compare the two optimization results from Problem 3a). Are the identified optimal parameter values the same? Are the Hessian matrices the same? Was anything different?

What about the log-posterior surface gave you a hint about how the two results would compare?

SOLUTION The identified optimal parameters are the same for both cases, as with the hessian matricies. The only apparent difference was in the counts section where the values of the function and gradient differ slightly. Since the log-posterior surface had a clear maximum for the parameters, it was clear that no matter what the initial guess was, the final result would provide the same value for the optimized parameters.

3c)

Finding the mode is the first step in the Laplace Approximation. The second step uses the negative inverse of the Hessian matrix as the approximate posterior covariance matrix. You will use a function, `my_laplace()`, to perform the complete Laplace Approximation. This way, this one function is all that's needed in order to perform all steps of the Laplace Approximation.

PROBLEM Complete the code chunk below. The `my_laplace()` function is adapted from the `laplace()` function from the `LearnBayes` package. Fill in the missing pieces to double check that you understand which portions of the optimization result correspond to the mode and which are used to approximate the posterior covariance matrix.

```
my_laplace <- function(start_guess, logpost_func, ...)
{
  # code adapted from the `LearnBayes`` function `laplace()`
  fit <- optim(start_guess,
               logpost_func,
               gr = NULL,
               ...,
               method = "BFGS",
               hessian = TRUE,
               control = list(fnscale = -1, maxit = 1001))

  mode <- fit$par
  h <- -solve(fit$hessian)
  p <- length(mode)
  # we will discuss what int means in a few weeks...
  int <- p/2 * log(2 * pi) + 0.5 * log(det(h)) + logpost_func(mode, ...)
  list(mode = mode,
       var_matrix = h,
       log_evidence = int,
       converge = ifelse(fit$convergence == 0,
                         "YES",
                         "NO"),
       iter_counts = fit$counts[1])
}
```

SOLUTION

3d)

You now have all of the pieces in place to perform the Laplace Approximation. Execute the Laplace Approximation for the 5 sample, 45 sample, and 405 sample cases.

PROBLEM Call the `my_laplace()` function in order to perform the Laplace Approximation based on the 3 sets of observations you studied with the log-posterior surface visualizations. Check that each converged.

```
laplace_res_inform_N05 <- my_laplace(init_guess_02, my_logpost, info_inform_N05)

laplace_res_inform_N45 <- my_laplace(init_guess_02, my_logpost, info_inform_N45)

laplace_res_inform_N405 <- my_laplace(init_guess_02, my_logpost, info_inform_N405)

### check if they eached converged
laplace_res_inform_N05
```

SOLUTION

```
## $mode
## [1] -0.3744223  1.6074010
##
## $var_matrix
##             [,1]        [,2]
## [1,]  0.83614844 -0.01616755
## [2,] -0.01616755  0.07729592
##
## $log_evidence
## [1] -17.07832
##
## $converge
## [1] "YES"
##
## $iter_counts
## function
##       23
```

```
laplace_res_inform_N45
```

```
## $mode
## [1] 0.6360899 1.5897039
##
## $var_matrix
##             [,1]        [,2]
## [1,]  0.35500136 -0.00868253
## [2,] -0.00868253  0.01097634
##
## $log_evidence
## [1] -139.359
##
## $converge
## [1] "YES"
##
## $iter_counts
## function
##       27
```

```
laplace_res_inform_N405
```

```
## $mode
## [1] 0.705152 1.638622
##
## $var_matrix
##               [,1]          [,2]
## [1,]  0.0614461494 -0.0001813897
## [2,] -0.0001813897  0.0012302057
##
## $log_evidence
## [1] -1244.418
##
## $converge
## [1] "YES"
```

```
##
## $iter_counts
## function
##       15
```

All three converged.

3e)

The MVN approximate posteriors that you solved for in Problem 3d) are in the (μ, φ) space. In order to help the manufacturing company, you will need to back-transform from φ to σ, while accounting for any potential posterior correlation with μ. A simple way to do so is through random sampling. The MVN approximate posteriors are a known type of distribution, a Multi-Variate Normal (MVN). You are therefore able to call a random number generator which uses the specified mean vector and specified covariance matrix to generate random samples from a MVN. Back-transforming from φ to σ is accomplished by simply using the inverse link function.

The code chunk below defines the `generate_post_samples()` function for you. The user provides the Laplace Approximation result as the first argument, and the number of samples to make as the second argument. The `MASS::mvrnorm()` function is used to generate the posterior samples. Almost all pieces of the function are provided to you, except you **must** complete the back-transformation from φ to σ by using the correct inverse link function.

PROBLEM Complete the code chunk below by using the correct inverse link function to back-transform from φ to σ.

```r
generate_post_samples <- function(mvn_info, N)
{
  MASS::mvrnorm(n = N,
                mu = mvn_info$mode,
                Sigma = mvn_info$var_matrix) %>%
    as.data.frame() %>% tbl_df() %>%
    purrr::set_names(c("mu", "varphi")) %>%
    mutate(sigma = exp(varphi)) ### backtransform to sigma!!
}
```

SOLUTION

3f)

1e4 posterior samples based on the 3 different observation sample sizes are generated for you in the code chunk below.

```r
set.seed(200121)
post_samples_inform_N05 <- generate_post_samples(laplace_res_inform_N05, N = 1e4)
post_samples_inform_N45 <- generate_post_samples(laplace_res_inform_N45, N = 1e4)
post_samples_inform_N405 <- generate_post_samples(laplace_res_inform_N405, N = 1e4)
```

You will summarize the posterior samples and visualize the posterior marginal histograms.

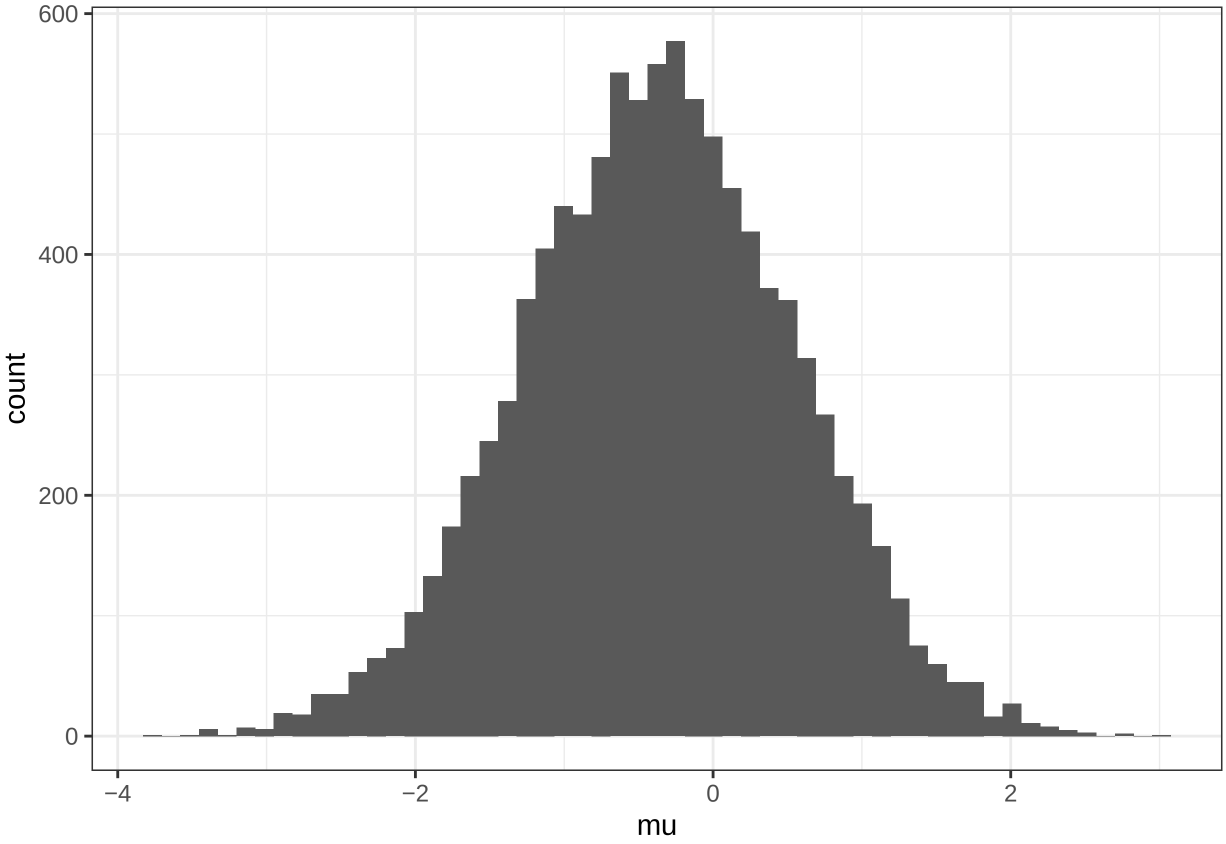
PROBLEM **Use the summary() function to summarize the posterior samples based on each of the three observation sample sizes. What are the posterior mean values on μ and σ as the number of observations increase? Visualize the posterior histograms on μ and σ based on each of the three observation sample sizes.**

```
summary(post_samples_inform_N05)
```

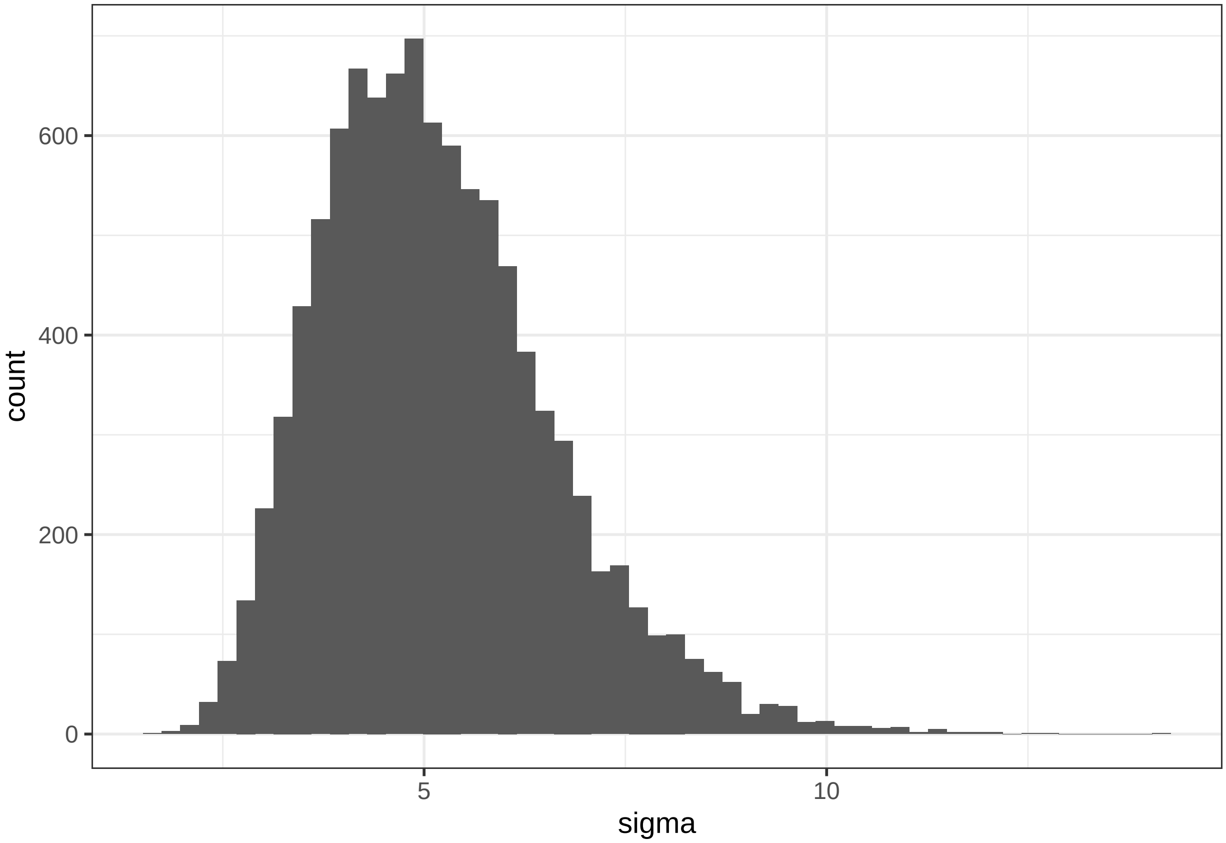
SOLUTION

```
##        mu              varphi           sigma
##  Min.   :-3.7249   Min.   :0.473   Min.   : 1.605
##  1st Qu.:-0.9947   1st Qu.:1.414   1st Qu.: 4.114
##  Median :-0.3631   Median :1.606   Median : 4.984
##  Mean   :-0.3697   Mean   :1.606   Mean   : 5.175
##  3rd Qu.: 0.2546   3rd Qu.:1.794   3rd Qu.: 6.015
##  Max.   : 3.0565   Max.   :2.649   Max.   :14.140
```

```
post_samples_inform_N05 %>%
  ggplot(mapping = aes(x= mu)) +
  geom_histogram(bins = 55) +
  theme_bw()
```



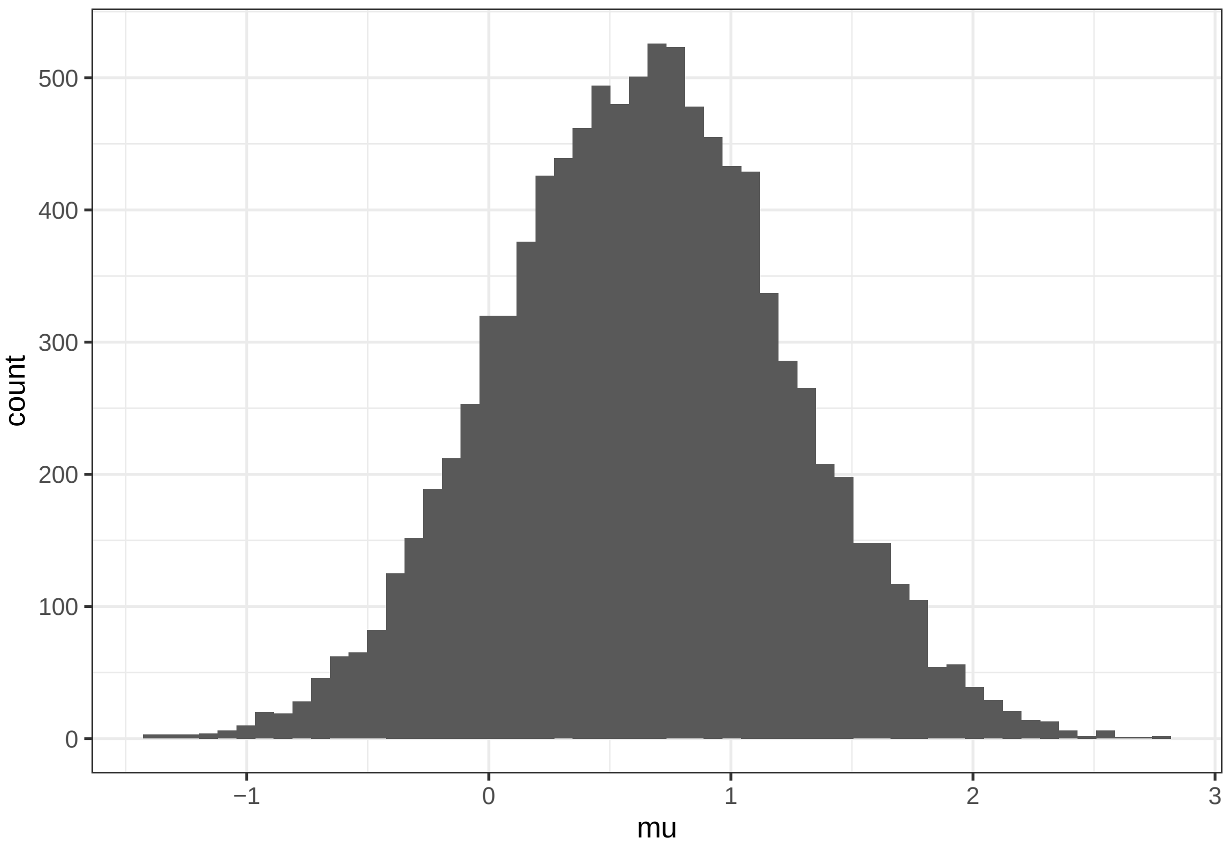
```
post_samples_inform_N05 %>%
  ggplot(mapping = aes(x= sigma)) +
  geom_histogram(bins = 55) +
  theme_bw()
```



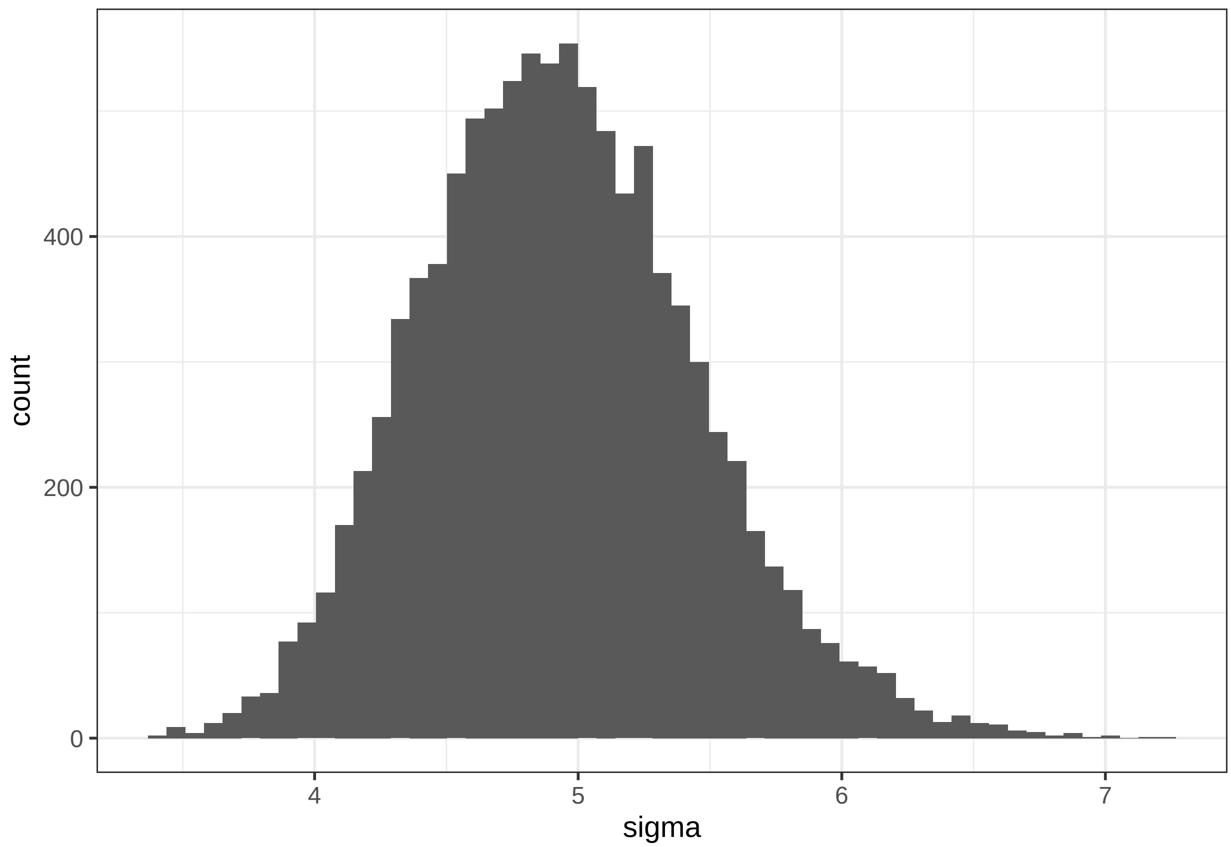
```
summary(post_samples_inform_N45)
```

```
##        mu              varphi          sigma
##  Min.   :-1.3960   Min.   :1.225   Min.   :3.403
##  1st Qu.: 0.2297   1st Qu.:1.518   1st Qu.:4.564
##  Median : 0.6409   Median :1.591   Median :4.906
##  Mean   : 0.6389   Mean   :1.589   Mean   :4.927
##  3rd Qu.: 1.0397   3rd Qu.:1.660   3rd Qu.:5.261
##  Max.   : 2.7729   Max.   :1.978   Max.   :7.231
```

```
post_samples_inform_N45 %>%
  ggplot(mapping = aes(x= mu)) +
  geom_histogram(bins = 55) +
  theme_bw()
```



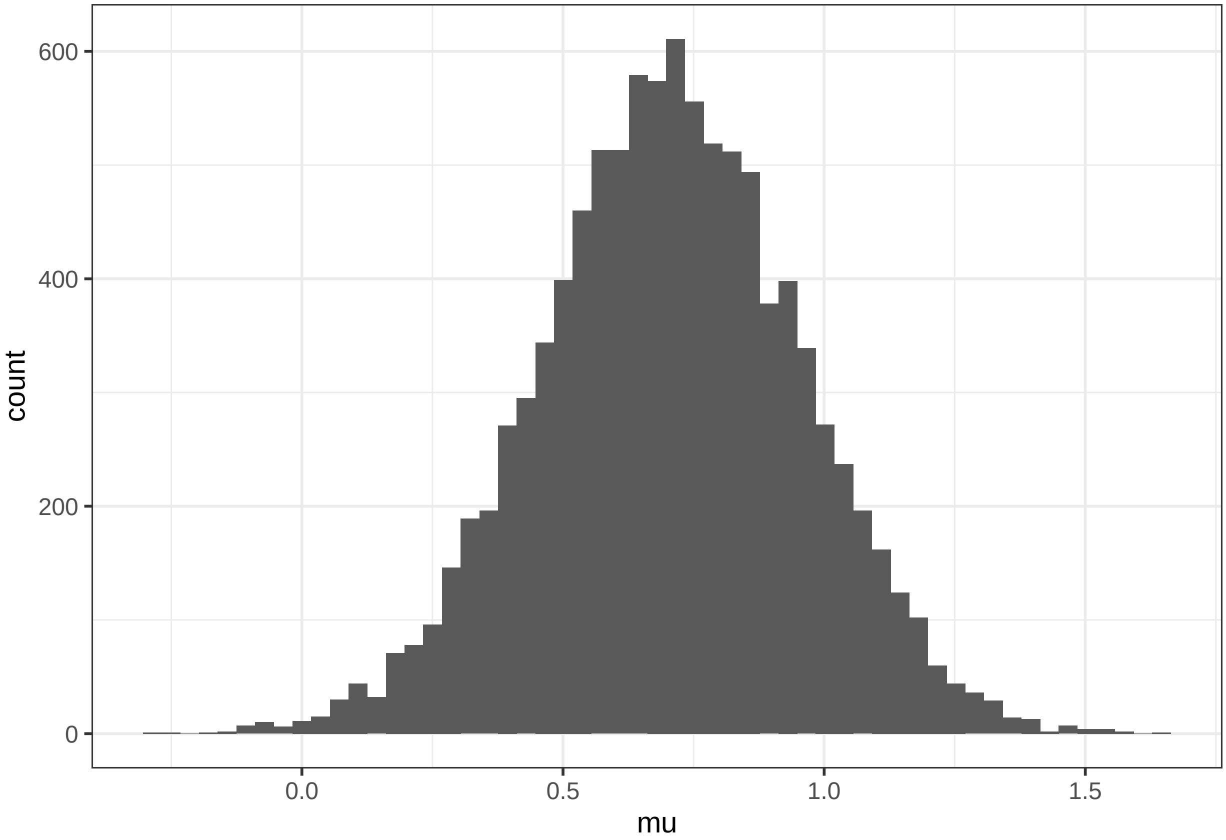
```
post_samples_inform_N45 %>%
  ggplot(mapping = aes(x= sigma)) +
  geom_histogram(bins = 55) +
  theme_bw()
```



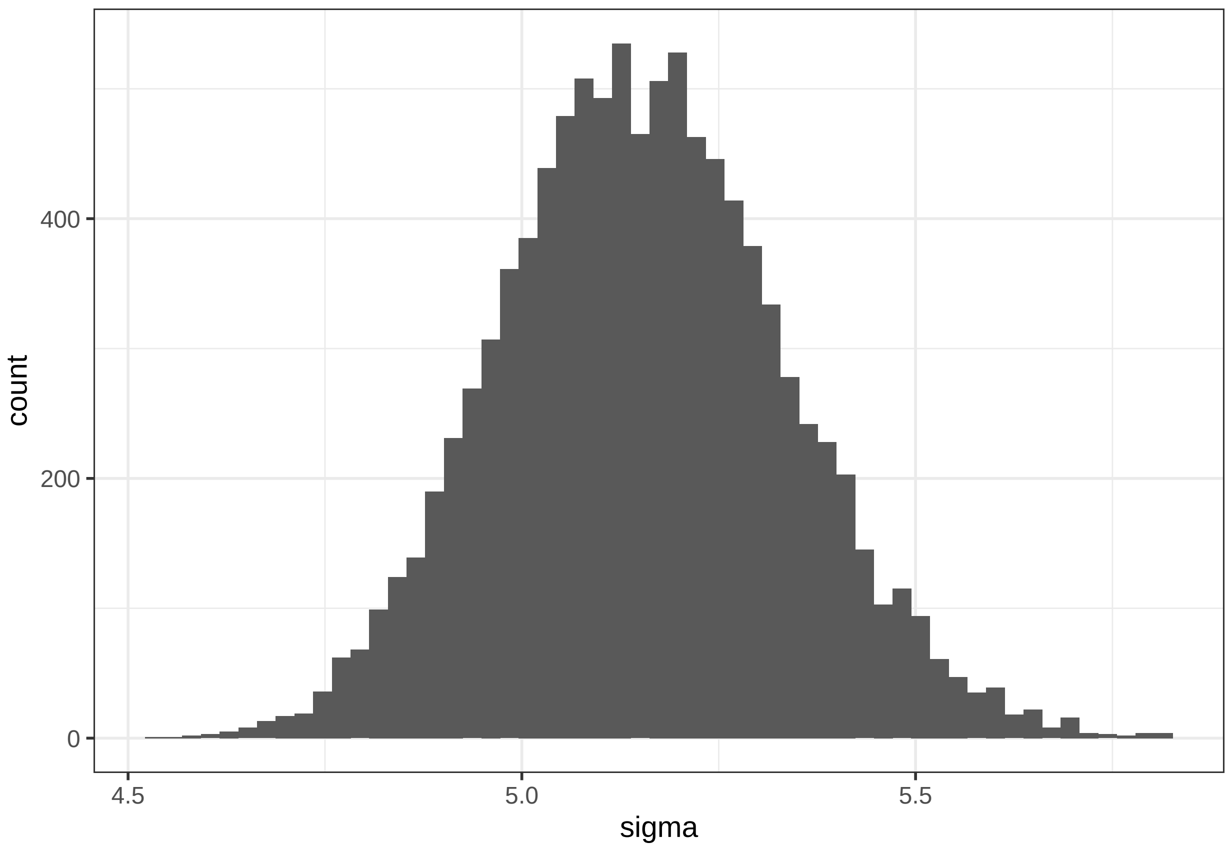
```
summary(post_samples_inform_N405)
```

```
##        mu               varphi           sigma      
##  Min.   :-0.2738   Min.   :1.513   Min.   :4.538  
##  1st Qu.: 0.5391   1st Qu.:1.615   1st Qu.:5.030  
##  Median : 0.7045   Median :1.639   Median :5.150  
##  Mean   : 0.7043   Mean   :1.639   Mean   :5.154  
##  3rd Qu.: 0.8710   3rd Qu.:1.663   3rd Qu.:5.275  
##  Max.   : 1.6586   Max.   :1.761   Max.   :5.820
```

```
post_samples_inform_N405 %>%
  ggplot(mapping = aes(x= mu)) +
  geom_histogram(bins = 55) +
  theme_bw()
```



```
post_samples_inform_N405 %>%
  ggplot(mapping = aes(x= sigma)) +
  geom_histogram(bins = 55) +
  theme_bw()
```



3g)

Another benefit of working with posterior samples rather than the posterior density is that it is relatively straight forward to answer potentially complex questions. At the start of this project, the engineering teams felt that mean value of the figure of merit was negative. You can now tell them the probability of that occuring. Additionally, the engineering teams felt that the measurement process was rather noisy. You can provide uncertainty intervals on the unknown σ. Depending on the amount of uncertainty about the noise, that might help them decide if company should invest in more precise measurement equipment.

PROBLEM What is the posterior probability that μ is positive for each of the three observation sample sizes? What are the 0.05 and 0.95 quantiles on σ for each of the three observation sample sizes?

```
temporary1 <-filter(post_samples_inform_N05, mu >=0)
count(temporary1)/count(post_samples_inform_N05)
```

SOLUTION

```
##       n
## 1 0.343
```

```
quantile(post_samples_inform_N05$sigma, c(0.05, 0.95))
```

```
##       5%      95%
## 3.153686 7.846135
```

```
temporary2 <-filter(post_samples_inform_N45, mu >=0)
count(temporary2)/count(post_samples_inform_N45)
```

```
##        n
## 1 0.8565
```

```
quantile(post_samples_inform_N45$sigma, c(0.05, 0.95))
```

```
##       5%      95%
## 4.123962 5.828306
```

```
temporary3 <-filter(post_samples_inform_N405, mu >=0)
count(temporary3)/count(post_samples_inform_N405)
```

```
##       n
## 1 0.997
```

```
quantile(post_samples_inform_N405$sigma, c(0.05, 0.95))
```

```
##       5%      95%
## 4.860401 5.462160
```

Problem 4

In this problem you get first hand experience working with the lm() function in R. You will make use of that function to fit simple to complex models on low and high noise data sets. You will then use cross-validation to assess if the models are overfitting to a training set.

Two data sets are in for you in the code chunks below. Both consist of an input x and a continuous response y. The two data sets are synthetic. Both come from the same underlying true functional form. The true functional form is not given to you. You are given random observations generated around that true functional form. The df_02_low data set corresponds to a random observations with low noise, while the df_02_high data set corresponds to random observations with high noise.

```
df_02_low <- readr::read_csv("https://raw.githubusercontent.com/jyurko/CS_1675_Spring_2020/master/hw_da
                             col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   x = col_double(),
##   y = col_double()
## )
```

```
df_02_high <- readr::read_csv("https://raw.githubusercontent.com/jyurko/CS_1675_Spring_2020/master/hw_da
                              col_names = TRUE)
```

```
## Parsed with column specification:
## cols(
##   x = col_double(),
##   y = col_double()
## )
```

Show a glimpse of each data set below.

```
df_02_low %>% glimpse()
```

```
## Observations: 30
## Variables: 2
## $ x <dbl> 1.18689051, 0.74887469, 0.48150224, -0.63588243, -0.10975984, -0.84...
## $ y <dbl> -1.18839785, -0.22018125, 0.52781651, -1.66456080, 0.04543771, -2.5...
```

```
df_02_high %>% glimpse()
```

```
## Observations: 30
## Variables: 2
## $ x <dbl> 1.18689051, 0.74887469, 0.48150224, -0.63588243, -0.10975984, -0.84...
## $ y <dbl> 1.3047004, 6.4105645, -2.5582175, 5.6122242, 0.6981725, -1.3924964,...
```

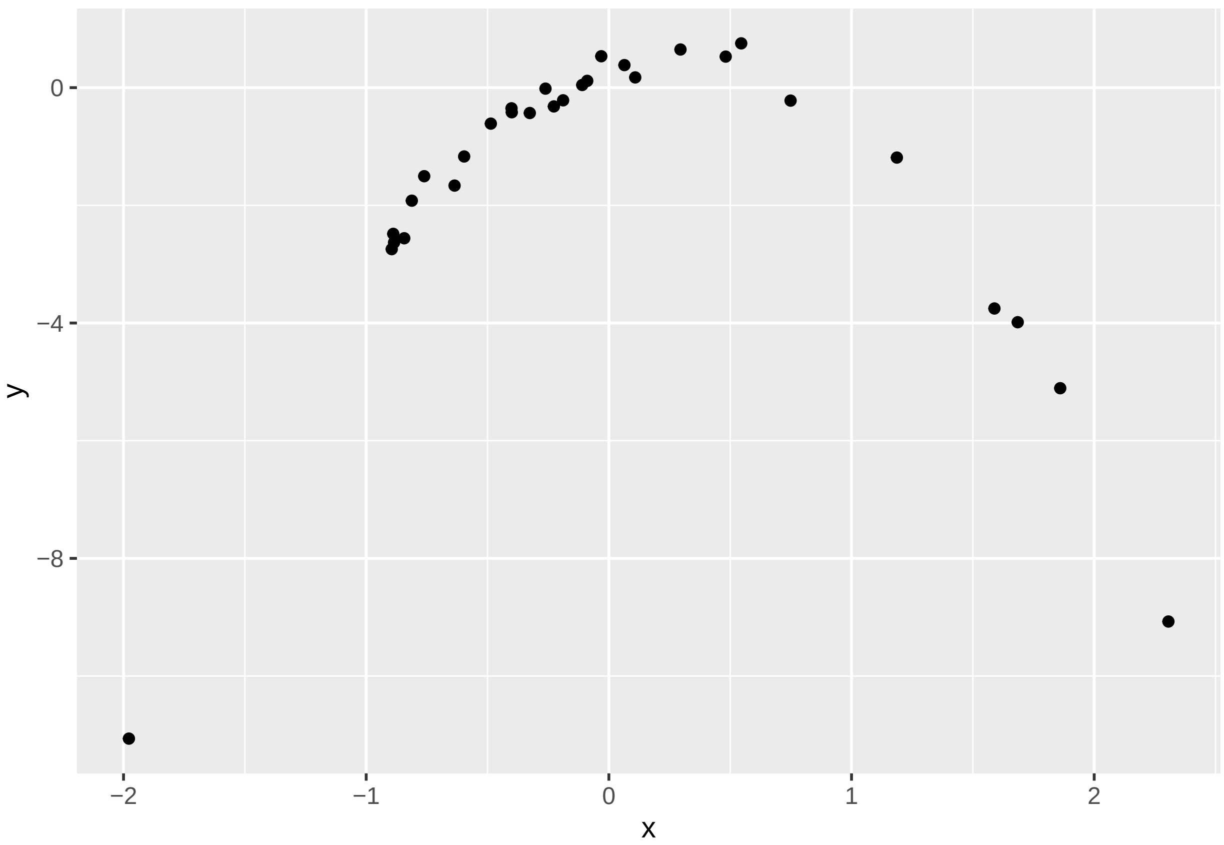
4a)

Create scatter plots in `ggplot2` between the response `y` and the input `x`.

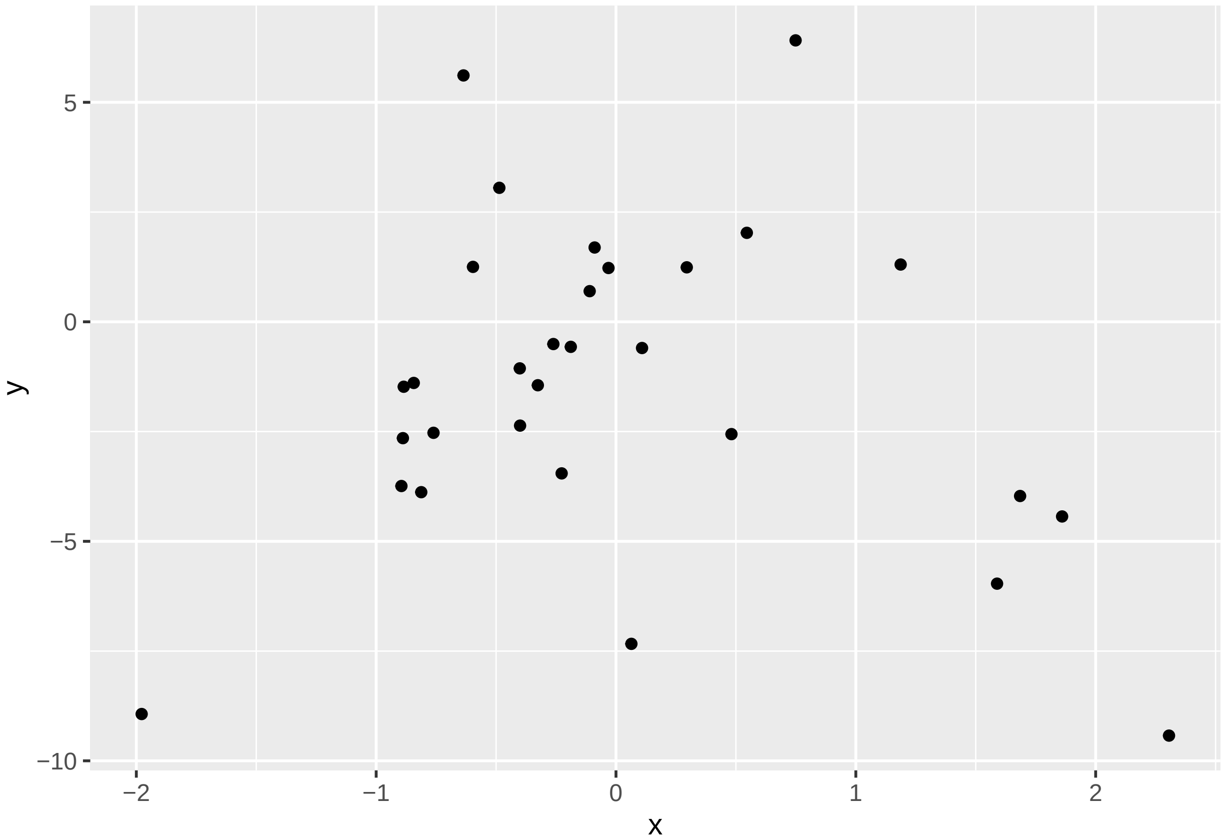
PROBLEM Use `ggplot()` with `geom_point()` to visualize scatter plots between the response and the input for both the low and high noise data sets. Since you know that `df_02_low` corresponds to low noise, can you make a guess about the true functional form that generated the data?

SOLUTION Your solution here.

```
df_02_low %>%
  ggplot(mapping = aes(x=x, y=y)) +
  geom_point()
```



```
df_02_high %>%
  ggplot(mapping = aes(x=x, y=y)) +
  geom_point()
```



The funcitonal form that has low noise is a polynomial function. Therefore I believe that the true funcitonal form of the low noise data set is a polynomial.

4b)

The `lm()` function in `R` is quite flexible. You can use a formula interface to specify models of various functional relationships between the response and the inputs. To get practice working with the formula interface you will create three models for the low noise case and three models for the high noise case. You will specificy a linear relationship between the response and the input, a quadratic relationship, and an 8th order polynomial.

There are many ways the polynomials can be created. For this assignment though, you will use the `I()` function to specify the polynomial terms. This approach will seem quite tedious, but the point is to get practice working with the formula interface. We will worry about efficiency later in the semester. The formula interface for stating the response, `y`, has a cubic relationship with the input `x` is just:

```
y ~ x + I(x^2) + I(x^3)
```

The `~` operator reads as "is a function of". Thus, the term to the left of `~` is viewed as a response, and the expression to the right of `~` is considered to be the features or predictors. With the formula specified the only other required argument to `lm()` is the data object. Thus, when using the formula interface, the syntax to creating a model with `lm()` is:

```
<model object> <- lm(<output> ~ <input expression>, data = <data object>)
```

PROBLEM Create a linear relationship, quadratic relationship, and 8th order polynomial model for the low and high noise cases. Use the formula interface to define the relationships.

SOLUTION The low noise case models are given below.

```
mod_low_linear <- lm(y ~ x, df_02_low)

mod_low_quad <- lm(y ~ x + I(x^2), df_02_low)

mod_low_8th <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), df_02_low)
```

The high noise case models are given in the code chunk below.

```
mod_high_linear <- lm(y ~ x, df_02_high)

mod_high_quad <- lm(y ~ x + I(x^2), df_02_high)

mod_high_8th <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), df_02_high)
```

4c)

There are many different approaches to inspect the results of `lm()`. A straightforward way is to use the `summary()` function to display a print out of each model's fit to the screen. Here, you will use `summary()` to read the R-squared performance metric associated with each model.

PROBLEM Use the `summary()` function to print out the summary of each model's fit. Which of the three relationships had the highest R-squared for the low noise case? Which of the three relationships had the highest R-squared for the high noise case?

```
summary(mod_low_linear)
```

SOLUTION

```
## 
## Call:
## lm(formula = y ~ x, data = df_02_low)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
## -9.936 -1.103  1.037  1.683  2.578 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)   
## (Intercept)  -1.6746     0.5071  -3.302  0.00263 **
## x            -0.2770     0.5388  -0.514  0.61118   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.778 on 28 degrees of freedom
## Multiple R-squared:  0.009353,	Adjusted R-squared:  -0.02603 
## F-statistic: 0.2644 on 1 and 28 DF,  p-value: 0.6112
```

```r
summary(mod_low_quad)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2), data = df_02_low)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.30058 -0.14804  0.02846  0.15243  0.41576
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.35622    0.04892   7.282 7.83e-08 ***
## x            1.24544    0.04696  26.521  < 2e-16 ***
## I(x^2)      -2.29565    0.03343 -68.674  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2134 on 27 degrees of freedom
## Multiple R-squared:  0.9944, Adjusted R-squared:  0.9939
## F-statistic:  2380 on 2 and 27 DF,  p-value: < 2.2e-16
```

```r
summary(mod_low_8th)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
##     I(x^7) + I(x^8), data = df_02_low)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.39944 -0.10282 -0.00119  0.13125  0.33509
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.27145    0.08218   3.303 0.003386 **
## x            1.41395    0.32837   4.306 0.000313 ***
## I(x^2)      -1.33091    0.57404  -2.319 0.030592 *
## I(x^3)      -0.58241    1.25036  -0.466 0.646155
## I(x^4)      -1.16003    0.71125  -1.631 0.117801
## I(x^5)       0.74587    1.28784   0.579 0.568642
## I(x^6)       0.16919    0.38967   0.434 0.668574
## I(x^7)      -0.15448    0.25577  -0.604 0.552340
## I(x^8)       0.01739    0.09003   0.193 0.848694
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.224 on 21 degrees of freedom
## Multiple R-squared:  0.9952, Adjusted R-squared:  0.9933
## F-statistic: 540.8 on 8 and 21 DF,  p-value: < 2.2e-16
```

```r
summary(mod_high_linear)
```

```
##
## Call:
## lm(formula = y ~ x, data = df_02_high)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -8.3898 -2.0073 -0.2306  2.6104  8.2148
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.4582     0.6852  -2.128   0.0423 *
## x            -0.4620     0.7280  -0.635   0.5308
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.753 on 28 degrees of freedom
## Multiple R-squared:  0.01418,    Adjusted R-squared:  -0.02103
## F-statistic: 0.4028 on 1 and 28 DF,  p-value: 0.5308
```

```r
summary(mod_high_quad)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2), data = df_02_high)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.7275 -1.2161 -0.2529  0.8867  6.6574
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.3436     0.6620   0.519 0.607993
## x             0.8888     0.6356   1.398 0.173391
## I(x^2)       -2.0368     0.4524  -4.502 0.000116 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.889 on 27 degrees of freedom
## Multiple R-squared:  0.4369, Adjusted R-squared:  0.3952
## F-statistic: 10.47 on 2 and 27 DF,  p-value: 0.0004296
```

```r
summary(mod_high_8th)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) +
##     I(x^7) + I(x^8), data = df_02_high)
##
## Residuals:
```

```
##     Min      1Q  Median      3Q     Max 
## -5.4166 -1.3881  0.0054  1.4579  4.7423 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)   
## (Intercept)  -1.8768     0.9666  -1.942   0.0657 . 
## x            -1.8605     3.8622  -0.482   0.6350   
## I(x^2)       18.2790     6.7517   2.707   0.0132 * 
## I(x^3)       10.1191    14.7063   0.688   0.4989   
## I(x^4)      -24.8491     8.3655  -2.970   0.0073 **
## I(x^5)       -3.2517    15.1472  -0.215   0.8321   
## I(x^6)        7.9959     4.5831   1.745   0.0957 . 
## I(x^7)        0.2307     3.0083   0.077   0.9396   
## I(x^8)       -0.7406     1.0589  -0.699   0.4920   
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 2.634 on 21 degrees of freedom
## Multiple R-squared:  0.6357, Adjusted R-squared:  0.4969 
## F-statistic: 4.581 on 8 and 21 DF,  p-value: 0.002403
```

For the low noise case the quadratic and the 8th order polynomial funciton had similarly high r squared values, however for the high noise case the 8th order polynomial function had the highest r squared values.

4d)

As mentioned previously, there are many methods in `R` to extract the performance of a `lm()` model. The `modelr` package, which is within the `tidyverse`, includes some useful functions for calculating the Root Mean Squared Error (RMSE) and R-squared values. The syntax for calculating the RMSE with the `modelr::rmse()` function is:

`modelr::rmse(<model object>, <data set>)`

The data supplied to `modelr::rmse()` can be a new data set, as long as the response is included in the data set. `modelr::rmse()` manages the book keeping for making predictions, comparing those predictions to the observed responses, calculating the errors, and summarizing. We will be discussing all of these steps later in the semester. For now, it's practical to know a function to quickly compute an important quantity such as RMSE.

We will use the `modelr` package for the remainder of Problem 4, so the code chunk below loads it into the current session.

```
library(modelr)
```

PROBLEM Use the `modelr::rmse()` function to calculate the RMSE for each of the models on their corresponding training sets. Thus, calculate the RMSE of the low noise case models with respect to the low noise data set. Calculate the RMSE of the high noise case models with respect to the high noise data set.

SOLUTION The low noise case RMSEs are calculated below.

```
modelr::rmse(mod_low_linear, df_02_low)
```

```
## [1] 2.683545
```

```
modelr::rmse(mod_low_quad, df_02_low )
```

```
## [1] 0.2024683
```

```
modelr::rmse(mod_low_8th, df_02_low )
```

```
## [1] 0.1873949
```

The high noise case RMSEs are calculated below.

```
modelr::rmse(mod_high_linear, df_02_high)
```

```
## [1] 3.625747
```

```
modelr::rmse(mod_high_quad, df_02_high)
```

```
## [1] 2.740299
```

```
modelr::rmse(mod_high_8th, df_02_high)
```

```
## [1] 2.204083
```

4e)

The previous performance metrics were calculated based on the training set alone. We know that only considering the training set performance metrics such as RMSE and R-squared will prefer overfit models, which simply chase the noise in the training data. To assess if a model is overfit, we can breakup a data set into multiple training and test splits via cross-validation. For the remainder of this problem, you will work with 5-fold cross-validation to get practice working with multiple data splits.

You will not have create the data splits on your own. In fact, the data splits are created for you in the code chunk below, for the low noise case. The `modelr::crossv_kfold()` function has two main input arguments, `data` and `k`. The `data` argument is the data set we wish to performance k-fold cross-validation on. The `k` argument is the number of folds to create. There is a third argument, `id`, which allows the user to name the fold "ID" labels, which by default are named `".id"`. As previously stated, the code chunk below creates the 5-fold data splits for the low noise case for you.

```
set.seed(23413)
cv_info_k05_low <- modelr::crossv_kfold(df_02_low, k = 5)
```

The contents of the `cv_info_k05_low` object are printed for you below. As you can see, `cv_info_k05_low` contains 3 columns, `train`, `test`, and `.id`. Although the object appears to be a data frame or tibble, the contents are not like most data frames. The `train` and `test` columns are actually lists which contain complex data objects. These complex objects are pointer-like in that they store how to access the training and test data sets from the original data set. In this way, the resampled object can be more memory efficient than just storing the actual data splits themselves. The `.id` column is just an ID, and so each row in `cv_info_k05_low` is a particular fold.

```
cv_info_k05_low
```

```
## # A tibble: 5 x 3
##   train        test         .id
##   <named list> <named list> <chr>
## 1 <resample>   <resample>   1
## 2 <resample>   <resample>   2
## 3 <resample>   <resample>   3
## 4 <resample>   <resample>   4
## 5 <resample>   <resample>   5
```

There are several ways to access the data sets directly. You can convert the resampled objects into integers, which provide the row indices associated with the train or test splits for each fold. To access the indices you need to use the [[]] format since you are selecting an element in a list. For example, to access the row indices for all rows selected in the first fold's training set:

```
as.integer(cv_info_k05_low$train[[1]])
```

```
##  [1]  2  3  4  5  6  8  9 10 11 13 14 15 16 17 18 21 22 23 24 25 26 27 29 30
```

Likewise to access the row indices for the third fold's test set:

```
as.integer(cv_info_k05_low$test[[3]])
```

```
## [1]  4 11 17 21 26 30
```

By storing pointers, the resample objects are rather memory efficient. We can make use of functional programming techniques to quickly and efficiently train and test models across all folds. In this assignment, though, you will turn the resampled object into separate data sets. Although not memory efficient, doing so allows you to work with each fold directly.

PROBLEM Convert each training and test split within each fold to a separate data set. To do so, use the as.data.frame() function instead of the as.integer() function. The object names in the code chunks below denote training or test and the particular fold to assign. You only have to work with the resampled object based on the low noise data set.

SOLUTION The fold training sets should be specified in the code chunk below.

```
low_noise_train_fold_01 <- as.data.frame(cv_info_k05_low$train[[1]])
low_noise_train_fold_02 <- as.data.frame(cv_info_k05_low$train[[2]])
low_noise_train_fold_03 <- as.data.frame(cv_info_k05_low$train[[3]])
low_noise_train_fold_04 <- as.data.frame(cv_info_k05_low$train[[4]])
low_noise_train_fold_05 <- as.data.frame(cv_info_k05_low$train[[5]])
```

The fold test sets should be specified in the code chunk below.

```
low_noise_test_fold_01 <-as.data.frame(cv_info_k05_low$test[[1]])
low_noise_test_fold_02 <-as.data.frame(cv_info_k05_low$test[[2]])
low_noise_test_fold_03 <-as.data.frame(cv_info_k05_low$test[[3]])
low_noise_test_fold_04 <- as.data.frame(cv_info_k05_low$test[[4]])
low_noise_test_fold_05 <- as.data.frame(cv_info_k05_low$test[[5]])
```

4f)

With the training and test splits available, now it's time to train the models on each training fold split. You can ignore the linear relationship model for the remainder of the assignment, and focus just on the quadratic relationship and 8th order polynomial.

PROBLEM Fit or train the quadratic relationship and 8th order polynomial using each fold's training split. Use the formula interface the define the relationship in each `lm()` call.

SOLUTION The quadratic relationship fits should be specified below.

```
mod_low_quad_fold_01 <- lm(y ~ x + I(x^2), low_noise_train_fold_01)

mod_low_quad_fold_02 <-lm(y ~ x + I(x^2), low_noise_train_fold_02)

mod_low_quad_fold_03 <- lm(y ~ x + I(x^2), low_noise_train_fold_03)

mod_low_quad_fold_04 <- lm(y ~ x + I(x^2), low_noise_train_fold_04)

mod_low_quad_fold_05 <- lm(y ~ x + I(x^2), low_noise_train_fold_05)
```

The 8th order polynomial fits should be specified below.

```
mod_low_8th_fold_01 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_nois

mod_low_8th_fold_02 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_nois

mod_low_8th_fold_03 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_nois

mod_low_8th_fold_04 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_nois

mod_low_8th_fold_05 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_nois
```

4g)

Let's compare the RMSE of the quadratic relationship to the 8th order polynomial, within each training fold. In this way, we can check that even after splitting the data, comparing models based on their training data still favors more complex models.

PROBLEM Calculate the RMSE for the quadratic relationship, using each fold's model fit, relative to the training splits. Thus, you should calculate 5 quantities, and store the 5 quantities in a vector named `cv_low_quad_fold_train_rmse`. Perform the analogous operation for the 8th order polynomial fits, and store in the vector named `cv_low_8th_fold_train_rmse`.

Calculate the average RMSE across the 5-folds for both relationships. Which relationship has the lowest average RMSE?

```r
cv_low_quad_fold_train_rmse <- c(modelr::rmse(mod_low_quad_fold_01, df_02_low),
modelr::rmse(mod_low_quad_fold_02, df_02_low),
modelr::rmse(mod_low_quad_fold_03, df_02_low),
modelr::rmse(mod_low_quad_fold_04, df_02_low),
modelr::rmse(mod_low_quad_fold_05, df_02_low))
```

```r
cv_low_8th_fold_train_rmse <- c(modelr::rmse(mod_low_8th_fold_01, df_02_low),
modelr::rmse(mod_low_8th_fold_02, df_02_low),
modelr::rmse(mod_low_8th_fold_03, df_02_low),
modelr::rmse(mod_low_8th_fold_04, df_02_low),
modelr::rmse(mod_low_8th_fold_05, df_02_low))
```

```r
mean(cv_low_quad_fold_train_rmse)
```

SOLUTION

```
## [1] 0.2049777
```

```r
mean(cv_low_8th_fold_train_rmse)
```

```
## [1] 3.363602
```

The quadratic relationship has the lowest average RMSE between the quadratic relationship and 8th order polynomial relationship for the low noise training datasets.

4h)

Now, it's time to compare the quadratic and 8th order polynomial performance in the 5 test splits. Repeat the steps you performed in Problem 4g), but this time with the test splits, instead of the training splits.

PROBLEM Calculate the test split RMSEs for each fold for the quadratic and 8th order polynomial. Assign the results to `cv_low_quad_fold_test_rmse` and `cv_low_8th_fold_test_rmse`, for the quadratic and 8th order polynomial, respectively. Which relationship has the lowest average cross-validation RMSE on the test splits?

```r
mod_low_quad_fold_001 <- lm(y ~ x + I(x^2), low_noise_test_fold_01)

mod_low_quad_fold_002 <-lm(y ~ x + I(x^2), low_noise_test_fold_02)

mod_low_quad_fold_003 <- lm(y ~ x + I(x^2), low_noise_test_fold_03)

mod_low_quad_fold_004 <- lm(y ~ x + I(x^2), low_noise_test_fold_04)

mod_low_quad_fold_005 <- lm(y ~ x + I(x^2), low_noise_test_fold_05)
```

SOLUTION The 8th order polynomial fits should be specified below.

```
mod_low_8th_fold_001 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_no

mod_low_8th_fold_002 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_no

mod_low_8th_fold_003 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_no

mod_low_8th_fold_004 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_no

mod_low_8th_fold_005 <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8), low_no
```

```
cv_low_quad_fold_test_rmse <- c(modelr::rmse(mod_low_quad_fold_001, df_02_low),
modelr::rmse(mod_low_quad_fold_002, df_02_low),
modelr::rmse(mod_low_quad_fold_003, df_02_low),
modelr::rmse(mod_low_quad_fold_004, df_02_low),
modelr::rmse(mod_low_quad_fold_005, df_02_low))
```

```
cv_low_8th_fold_test_rmse <- c(modelr::rmse(mod_low_8th_fold_001, df_02_low),
modelr::rmse(mod_low_8th_fold_002, df_02_low),
modelr::rmse(mod_low_8th_fold_003, df_02_low),
modelr::rmse(mod_low_8th_fold_004, df_02_low),
modelr::rmse(mod_low_8th_fold_005, df_02_low))
```

```
## Warning in predict.lm(model, data): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(model, data): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(model, data): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(model, data): prediction from a rank-deficient fit may be
## misleading

## Warning in predict.lm(model, data): prediction from a rank-deficient fit may be
## misleading
```

```
mean(cv_low_quad_fold_test_rmse)
```

```
## [1] 0.5163533
```

```
mean(cv_low_8th_fold_test_rmse)
```

```
## [1] 6323.189
```