Overview

This homework assignment reviews important aspects of training linear models focusing on the relationship between complexity and model performance. You will fit non-Bayesian and Bayesian linear models of varying levels of complexity. You will compare their performance and examine their predictive trends. Unlike earlier assignments you will examine **both** types of uncertainty in addition to the trend. This way you gain experience with the relationship between complexity, training set performance, confidence intervals, and prediction intervals.

This assignment also requires you to fit the Bayesian models with various prior standard deviations. In this way, you learn about the prior's role on coefficient estimates and predictive trends. These tasks introduce to the concept that the prior **regularizes** coefficients.

Lastly, you are introduced to non-Bayesian regularization with Lasso regression via the `glmnet` package. If you do not have `glmnet` installed **PLEASE** download it before starting the assignment.

IMPORTANT: The RMarkdown assumes you have downloaded the data set (CSV file) to the same directory you saved the template Rmarkdown file. If you do not have the CSV files in the correct location, the data will not be loaded correctly.

IMPORTANT!!!

Certain code chunks are created for you. Each code chunk has `eval=FALSE` set in the chunk options. You **MUST** change it to be `eval=TRUE` in order for the code chunks to be evaluated when rendering the document.

You are free to add more code chunks if you would like.

Load packages

This assignment will use packages from the `tidyverse` suite as well as the `coefplot` package. Those packages are imported for you below.

```
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.3     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(coefplot)
```

This assignment also uses the `splines` and `MASS` packages. Both are installed with base `R` and so you do not need to download any additional packages to complete the assignment.

The last question in the assignment uses the `glmnet` package. As stated previously, please download and install `glmnet` if you do not currently have it.

Problem 01

You will fit and compare **6 models** of varying complexity using **non-Bayesian methods**. The unknown parameters will be be estimated by finding their Maximum Likelihood Estimates (MLE). You are allowed to use the `lm()` function for this problem.

The data are loaded in the code chunk and a glimpse is shown for you below. There are 2 continuous inputs, `x1` and `x2`, and a continuous response `y`.

```
hw_file_path <- 'hw08_data.csv'

df <- readr::read_csv(hw_file_path, col_names = TRUE)
```

```
## Rows: 100 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (3): x1, x2, y
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
df %>% glimpse()
```

```
## Rows: 100
## Columns: 3
## $ x1 <dbl> -0.30923281, 0.63127211, -0.68276690, 0.26930562, 0.37252021, 1.296~
## $ x2 <dbl> 0.308779853, -0.547919793, 2.166449412, 1.209703658, 0.785485991, -~
## $ y  <dbl> 0.43636596, 1.37562976, -0.84366730, -0.43080811, 0.77456951, 1.361~
```

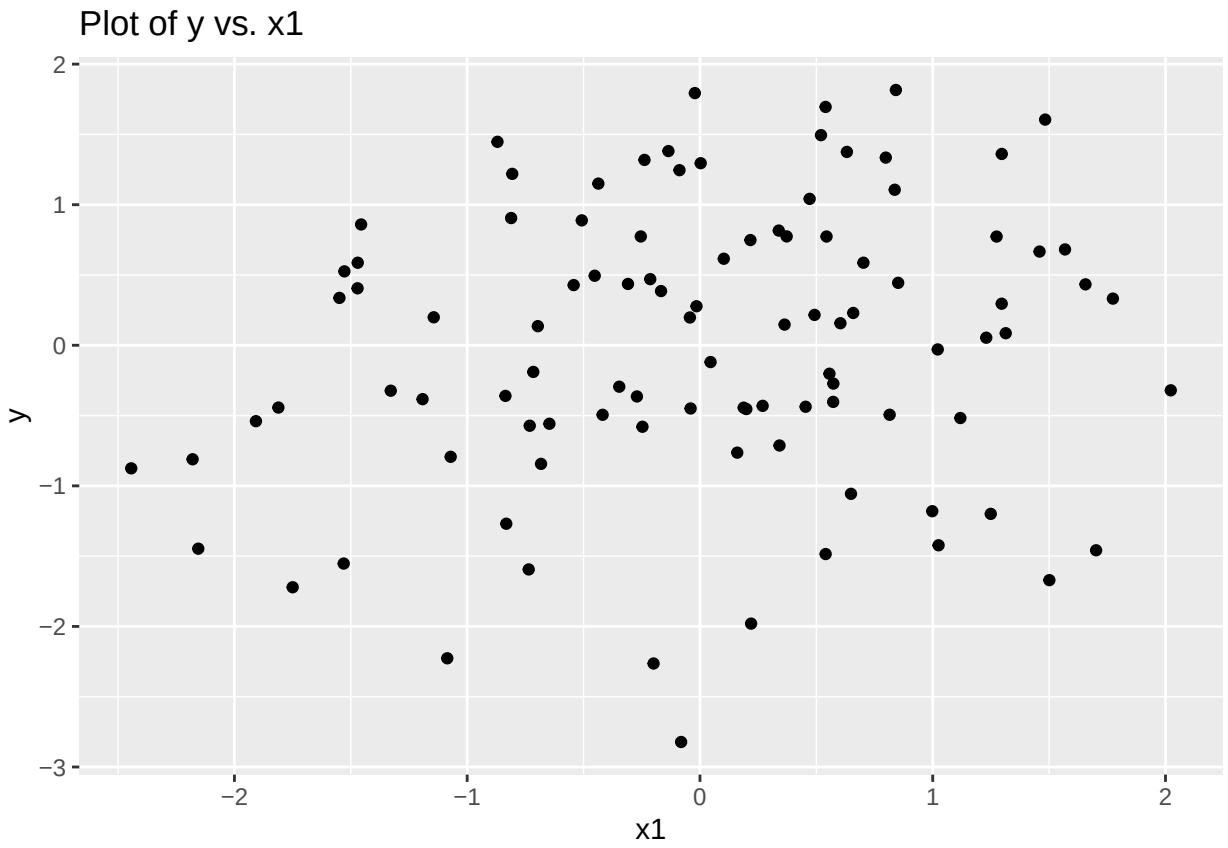
1a)

Create a scatter plot between the response, `y`, and each input using `ggplot()`.

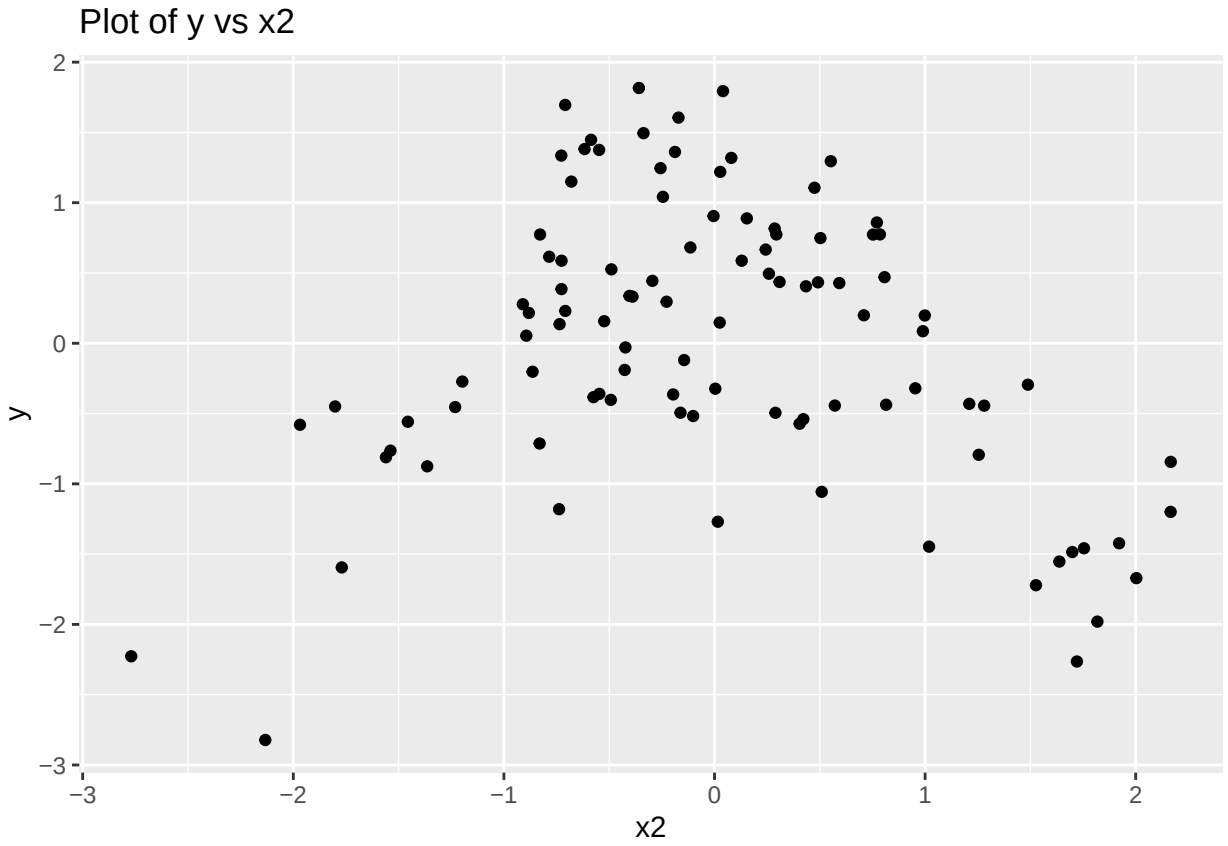
Based on the visualizations, do you think there are trends between either input and the response?

SOLUTION Based on my visualizations on the plots below, I do not think there are trends between either input and the response.

```
# Load necessary libraries
ggplot(df, aes(x=x1, y=y)) +
  geom_point() +
  ggtitle("Plot of y vs. x1") +
  xlab("x1") +
  ylab("y")
```




```
ggplot(df, aes(x = x2, y=y))+
  geom_point() +
  ggtitle("Plot of y vs x2") +
  xlab("x2") +
  ylab("y")
```



1b)

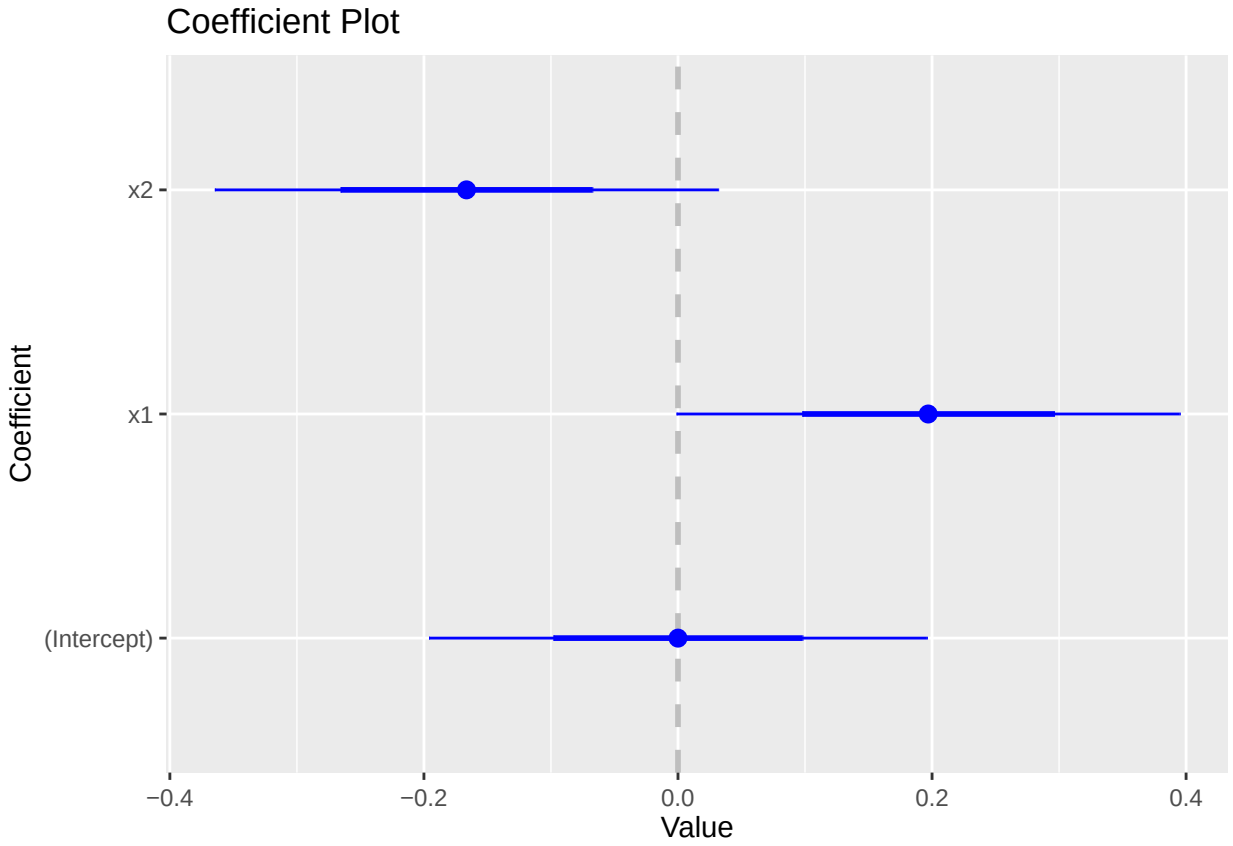
You will fit multiple models of varying complexity in this problem. You will start with *linear additive features* which *add* the effect of one input with the other. Your model therefore *controls* for both inputs.

Fit a model with linear additive features to predict the response, `y`. Use the formula interface and the `lm()` function to fit the model. Assign the result to the `mod01` object.

Visualize the coefficient summaries with the `coefplot()` function. Are any of the features statistically significant?

SOLUTION Since the 3 confidence intervals contain 0 in the plot below, we can conclude that none of the features is statistically significant.

```
mod01 <- lm(y ~ x1 + x2, data=df)

coefplot(mod01)
```



1c)

As discussed in lecture, we can derive features from inputs. We have worked with polynomial features and spline-based features in previous assignments. Features can also be derived as the products between different inputs. A feature calculated as the **product** of multiple inputs is usually referred to as the **interaction** between those inputs.

In the formula interface, a product of two inputs is denoted by the `:`. And so if we want to include just the multiplication of `x1` and `x2` in a model we would type, `x1:x2`. We can then include **main-effect** terms by including the additive features within the formula. Thus, the formula for a model with additive features and the interaction between `x1` and `x2` is:

```
y ~ x1 + x2 + x1:x2
```

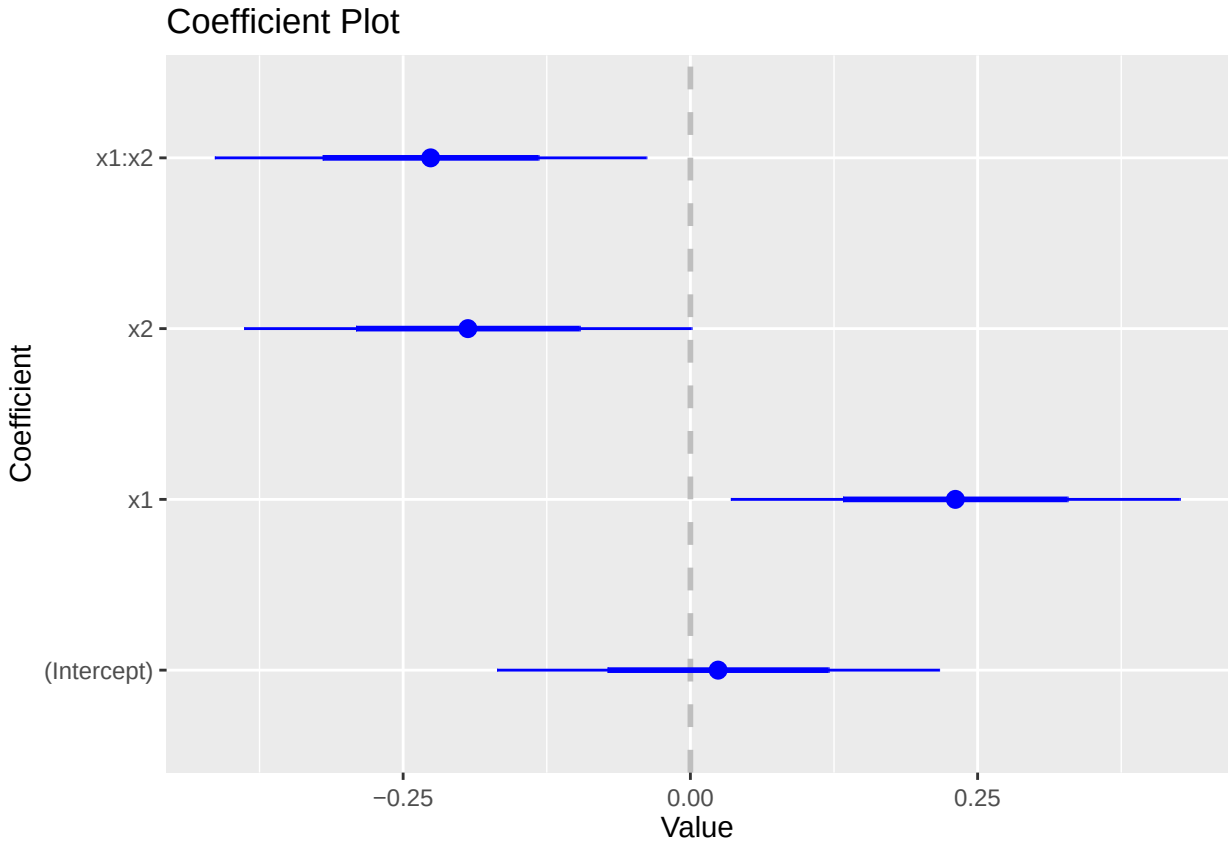
However, the formula interface provides a short-cut to create main effects and interaction features. In the formula interface, the `*` operator will generate all main-effects and all interactions for us.

Fit a model with all main-effect and all-interaction features between `x1` and `x2` using the short-cut `*` operator within the formula interface. Assign the result to the `mod02` object.

Visualize the coefficient summaries with the `coefplot()` function. How many features are present in the model? Are any of the features statistically significant?

SOLUTION As it can be seen in the plot below, there are 4 features in the model and the features `x1` and `x1:x2` are statistically significant because their confidence intervals does not contain 0.

```
mod02 <- lm(y ~ x1 * x2, data=df)

coefplot(mod02)
```



1d)

The `*` operator will interact more than just inputs. We can interact expressions or groups of features together. To interact one group of features by another group of features, we just need to enclose each group within parenthesis, `()`, and separate them by the `*` operator. The line of code below shows how this works with the `<expression 1>` and `<expression 2>` as place holders for any expression we want to use.

```
(<expression 1>) * (<expression 2>)
```

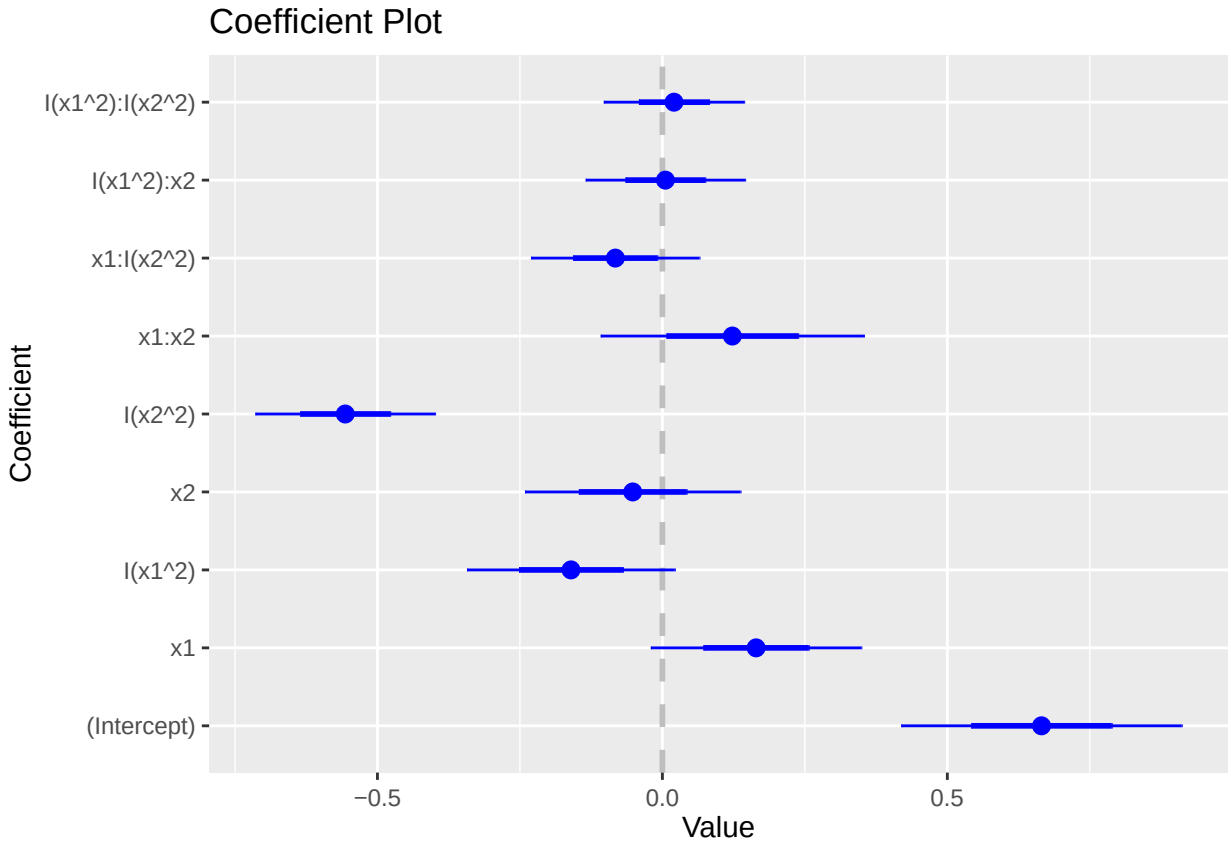
Fit a model which interacts linear and quadratic features from `x1` with linear and quadratic features from `x2`. Assign the result to the `mod03` object.

Visualize the coefficient summaries with the `coefplot()` function. How many features are present in the model? Are any of the features statistically significant?

HINT: Remember to use the `I()` function when typing polynomials in the formula interface.

SOLUTION There are 9 features in the model. 2 features are statistically significant; `intercept` and `x2^2`, because their confidence intervals do not contain 0.

```
mod03 <- lm(y ~ (x1 + I(x1^2)) * (x2 + I(x2^2)), data =df)

coefplot(mod03)
```



1e)

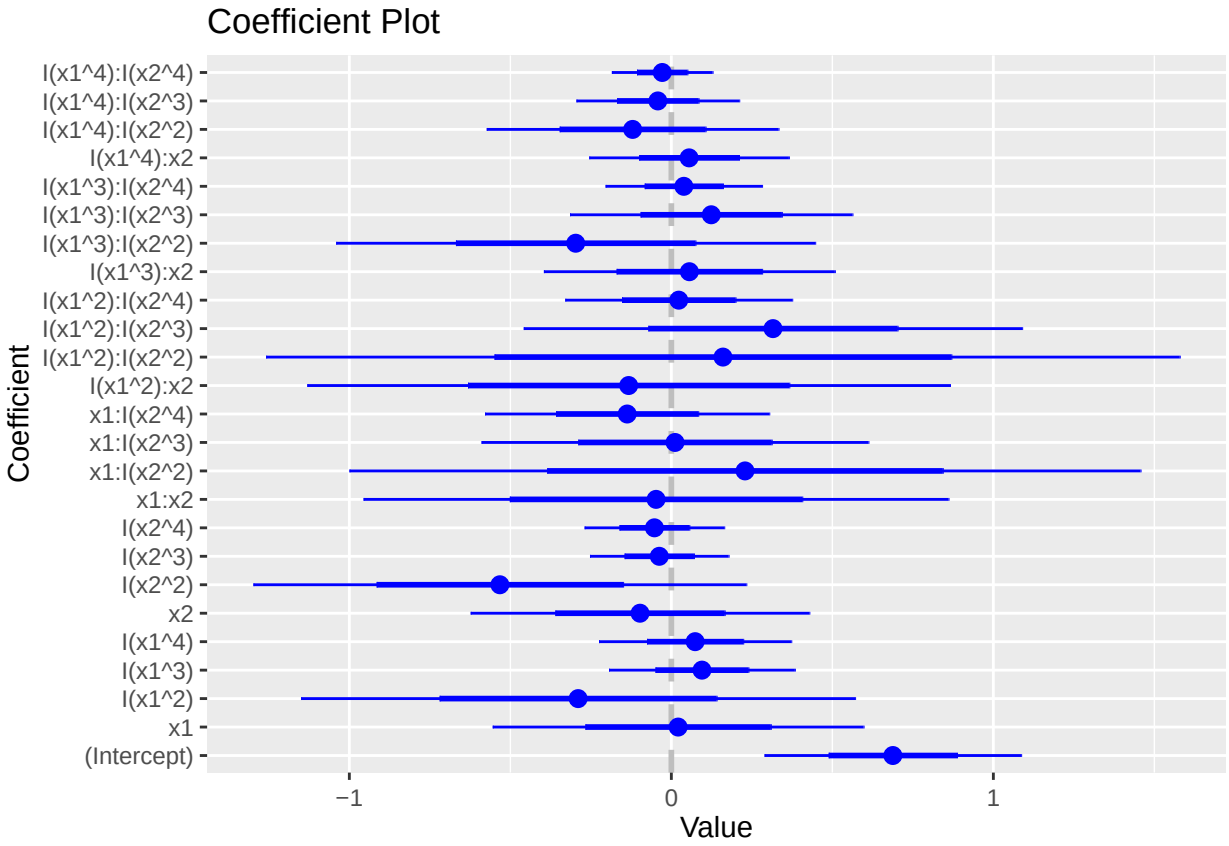
Let's now try a more complicated model.

Fit a model which interacts linear, quadratic, cubic, and quartic (4th degree) polynomial features from `x1` with linear, quadratic, cubic, and quartic (4th degree) polynomial features from `x2`. Assign the result to the `mod04` object.

Visualize the coefficient summaries with the `coefplot()` function. Are any of the features statistically significant?

SOLUTION There are 25 features in the model. Only 1 feature is statistically significant; `intercept`, because its confidence interval does not contain 0.

```
mod04 <- lm(y ~ (x1 + I(x1^2) + I(x1^3) + I(x1^4)) * (x2 + I(x2^2) + I(x2^3) + I(x2^4)), data =df)

coefplot(mod04)
```



1f)

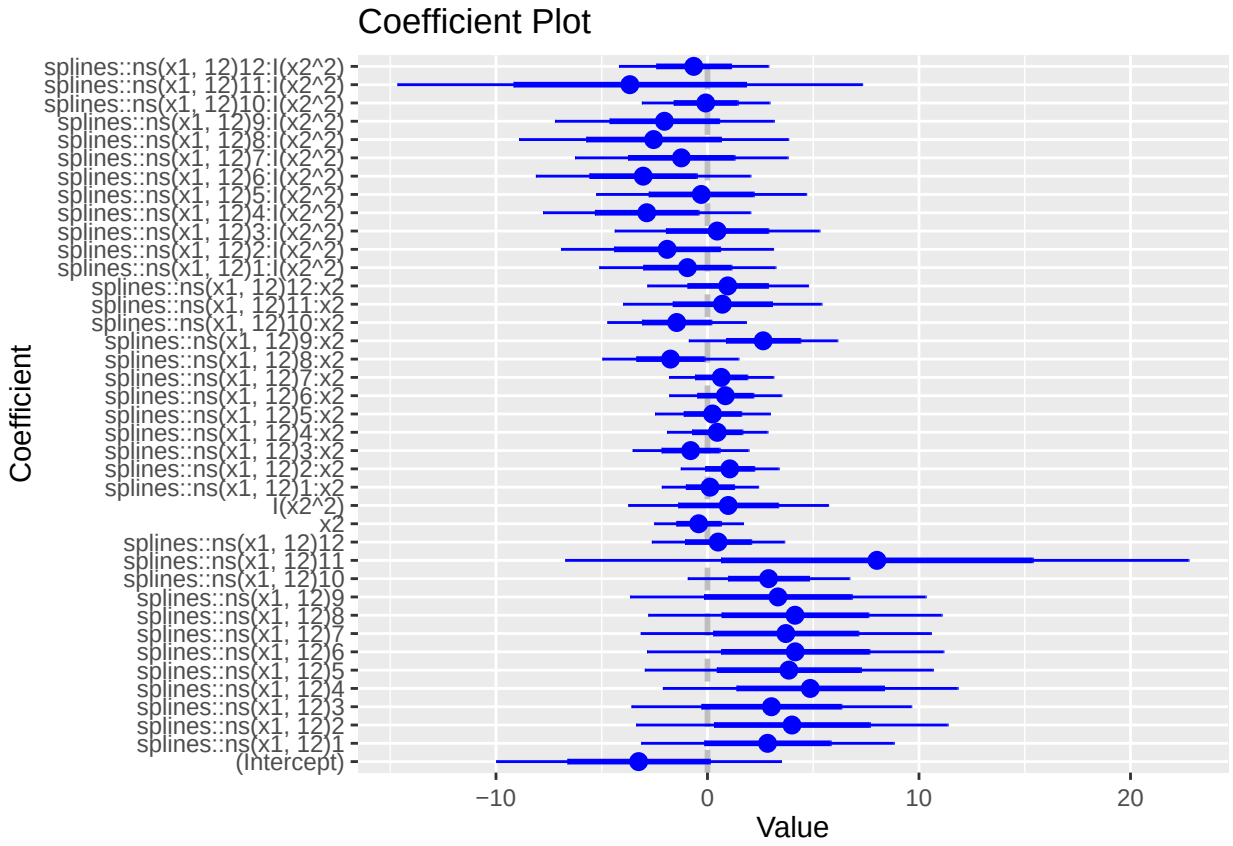
Let's try using spline based features. We will use a high degree-of-freedom natural spline applied to `x1` and interact those features with polynomial features derived from `x2`.

Fit a model which interacts a 12 degree-of-freedom natural (DOF) spline from `x1` with linear and quadrtic polyonomial features from `x2`. Assign the result to `mod05`.

Visualize the coefficient summaries with the `coefplot()` function. Are any of the features statistically significant?

SOLUTION When we see the plot below, we can conclude that none of the features is statistically significant.

```
mod05 <- lm(y ~ splines::ns(x1, 12) * (x2 + I(x2^2)), data=df)

coefplot(mod05)
```



1g)

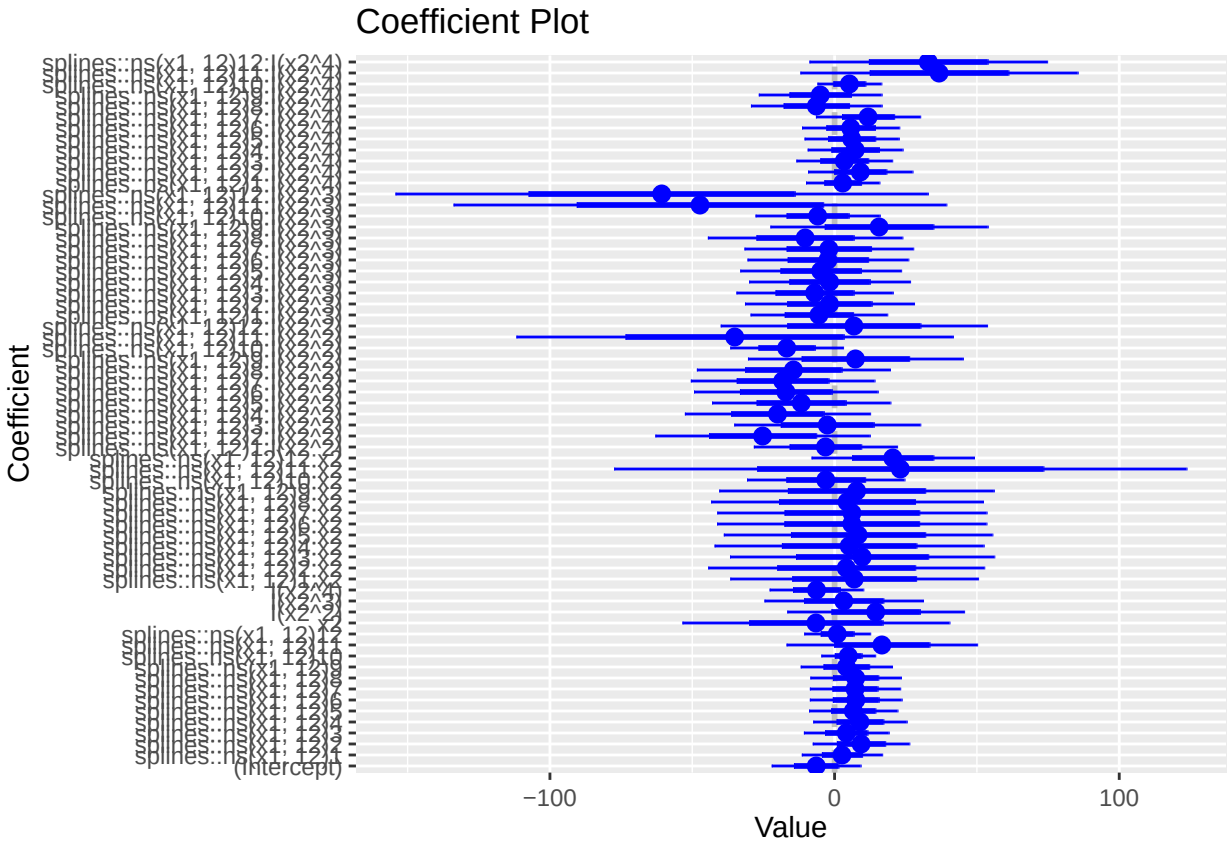
Let's fit one final model.

Fit a model which interacts a 12 degree-of-freedom natural spline from `x1` with linear, quadrtic, cubic, and quartic (4th degree) polyonomial features from `x2`. Assign the result to `mod05`.

Visualize the coefficient summaries with the `coefplot()` function. Are any of the features statistically significant?

SOLUTION Based on the plot below, we can conclude that none of the features is statistically significant.

```
mod06 <- lm(y ~ splines::ns(x1, 12) * (x2 + I(x2^2) + I(x2^3) + I(x2^4)), data=df)

coefplot(mod06)
```



1h)

Now that you have fit multiple models of varying complexity, it is time to identify the best performing model.

Identify the best model considering training set only performance metrics. Which model is best according to R-squared? Which model is best according to AIC? Which model is best according to BIC?

HINT: The `brooom::glance()` function can be helpful here. The `broom` package is installed with `tidyverse` and so you should have it already.

SOLUTION Based on the values below, we can conclude that;

1) The best model according to R-squared is `mod06`, because it has the biggest R-squared value.
2) The best model according to AIC is `mod03`, because it has the smallest AIC value.
3) The best model according to BIC is `mod03`, because it has the smallest BIC value.

```
summary_mod01 <- broom::glance(mod01)
summary_mod02 <- broom::glance(mod02)
summary_mod03 <- broom::glance(mod03)
summary_mod04 <- broom::glance(mod04)
summary_mod05 <- broom::glance(mod05)
summary_mod06 <- broom::glance(mod06)
summary_mod01
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
```

```
## 1    0.0594        0.0401 0.980      3.07  0.0512     2  -138.  285.  295.
## # i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
summary_mod02
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.113        0.0853 0.956      4.08 0.00900     3  -135.  281.  294.
## # i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
summary_mod03
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.547         0.507 0.702      13.7 7.25e-13     8  -102.  224.  250.
## # i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
summary_mod04
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic     p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>       <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.599         0.470 0.728      4.66 0.000000151    24  -95.7  243.  311.
## # i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
summary_mod05
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic    p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>      <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.699         0.512 0.699      3.73 0.00000235    38  -81.3  243.  347.
## # i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
summary_mod06
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.782         0.383 0.785      1.96  0.0164    64  -65.3  263.  434.
## # i 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

Problem 02

Now that you know which model is best, let's visualize the predictive trends from the six models. This will help us better understand their performance and behavior.

2a)

You will define a prediction or visualization test grid. This grid will allow you to visualize behavior with respect to `x1` for multiple values of `x2`.

Create a grid of input values where `x1` consists of 101 evenly spaced points between -3.2 and 3.2 and `x2` is 9 evenly spaced points between -3 and 3. The `expand.grid()` function is started for you and the data type conversion is provided to force the result to be a `tibble`.

```

```

```
viz_grid <- expand.grid(x1 = seq(-3.2, 3.2, length.out = 101),
                        x2 = seq(-3, 3, length.out = 9),
                        KEEP.OUT.ATTRS = FALSE,
                        stringsAsFactors = FALSE) %>%
  as.data.frame() %>% tibble::as_tibble()
```

SOLUTION

2b)

You will make predictions for each of the models and visualize their trends. A function, `tidy_predict()`, is created for you which assembles the predicted mean trend, the confidence interval, and the prediction interval into a `tibble` for you. The result include the input values to streamline making the visualizations.

```
tidy_predict <- function(mod, xnew)
{
  pred_df <- predict(mod, xnew, interval = "confidence") %>%
    as.data.frame() %>% tibble::as_tibble() %>%
    dplyr::select(pred = fit, ci_lwr = lwr, ci_upr = upr) %>%
    bind_cols(predict(mod, xnew, interval = 'prediction') %>%
                as.data.frame() %>% tibble::as_tibble() %>%
                dplyr::select(pred_lwr = lwr, pred_upr = upr))

  xnew %>% bind_cols(pred_df)
}
```

The first argument to the `tidy_predict()` function is a `lm()` model object and the second argument is new or test dataframe of inputs. When working with `lm()` and its `predict()` method, the functions will create the test design matrix consistent with the training design basis. It does so via the model object's formula which is contained within the `lm()` model object. The `lm()` object therefore takes care of the heavy lifting for us!

Make predictions with each of the six models you fit in Problem 01 using the visualization grid, `viz_grid`. The predictions should be assigned to the variables `pred_lm_01` through `pred_lm_06` where the number is consistent with the model number fit previously.

```
pred_lm_01 <- tidy_predict(mod01, viz_grid)

pred_lm_02 <- tidy_predict(mod02, viz_grid)

pred_lm_03 <- tidy_predict(mod03, viz_grid)

pred_lm_04 <- tidy_predict(mod04, viz_grid)

pred_lm_05 <- tidy_predict(mod05, viz_grid)

pred_lm_06 <- tidy_predict(mod06, viz_grid)
```

SOLUTION

2c)

You will now visualize the predictive trends and the confidence and prediction intervals for each model. The `pred` column in of each `pred_lm_` objects is the predictive mean trend. The `ci_lwr` and `ci_upr` columns are

the lower and upper bounds of the confidence interval, respectively. The `pred_lwr` and `pred_upr` columns are the lower and upper bounds of the prediction interval, respectively.

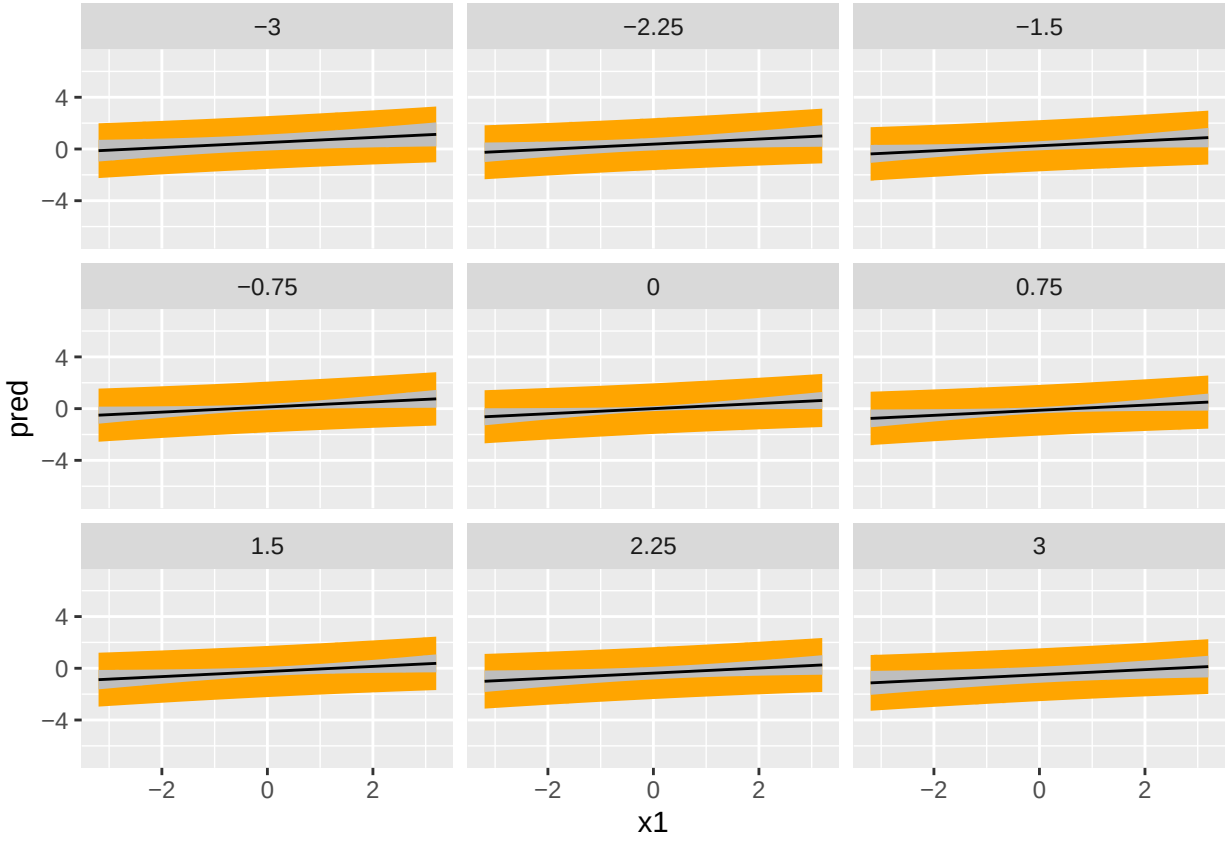
You will use `ggplot()` to visualize the predictions. You will use `geom_line()` to visualize the mean trend and `geom_ribbon()` to visualize the uncertainty intervals.

Visualize the predictions of each model on the visualization grid. Pipe the `pred_lm_` object to `ggplot()` and map the `x1` variable to the x-aesthetic. Add three geometric object layers. The first and second layers are each `geom_ribbon()` and the third layer is `geom_line()`. In the `geom_line()` layer map the `pred` variable to the `y` aesthetic. In the first `geom_ribbon()` layer, map `pred_lwr` and `pred_upr` to the `ymin` and `ymax` aesthetics, respectively. Hard code the `fill` to be orange in the first `geom_ribbon()` layer (outside the `aes()` call). In the second `geom_ribbon()` layer, map `ci_lwr` and `ci_upr` to the `ymin` and `ymax` aesthetics, respectively. Hard code the `fill` to be grey in the second `geom_ribbon()` layer (outside the `aes()` call). Include `facet_wrap()` with the facets with controlled by the `x2` variable.

To help compare the visualizations across models include a `coord_cartesian()` layer with the `ylim` argument set to `c(-7,7)`.

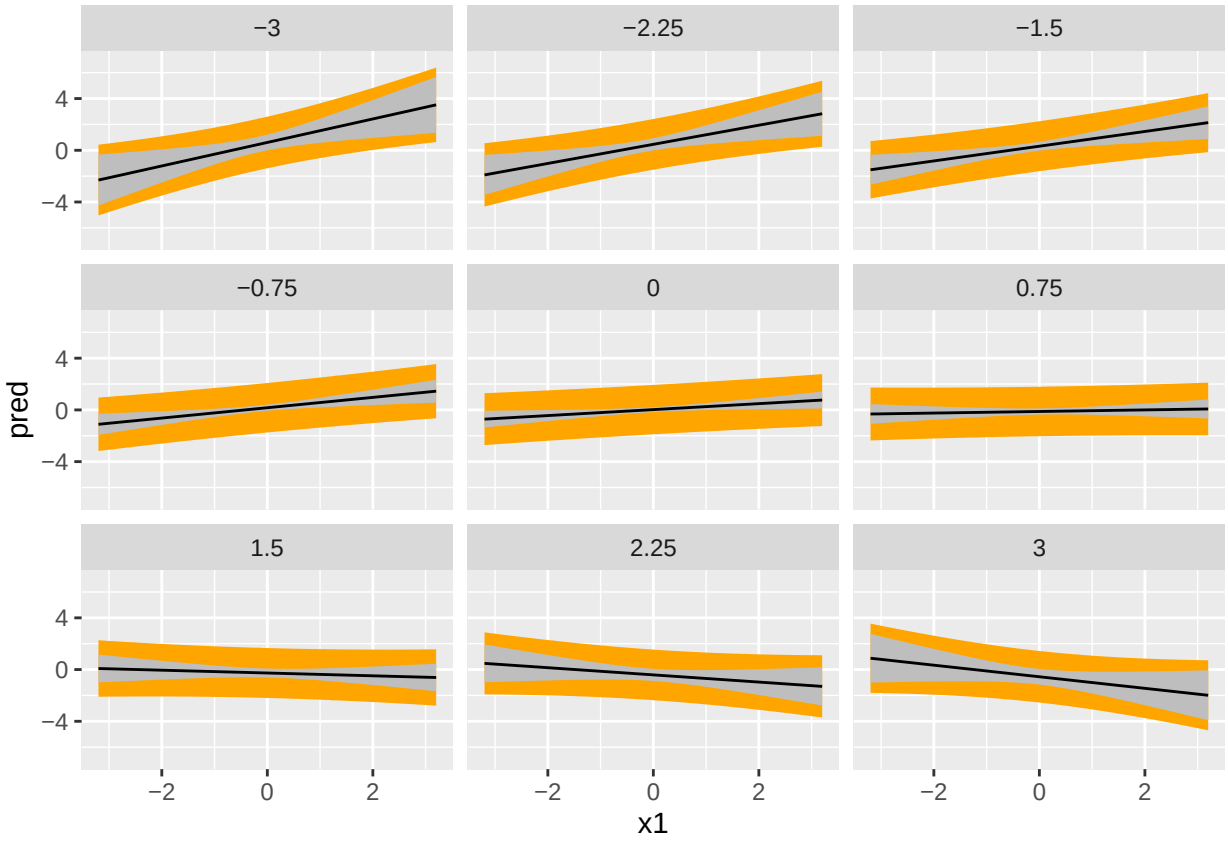
Each model's prediction visualization should be created in a separate code chunk.

```
pred_lm_01 %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=pred_lwr, ymax=pred_upr), fill="orange")+
  geom_ribbon(aes(ymin=ci_lwr, ymax=ci_upr), fill="grey")+
  geom_line(aes(y=pred)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```

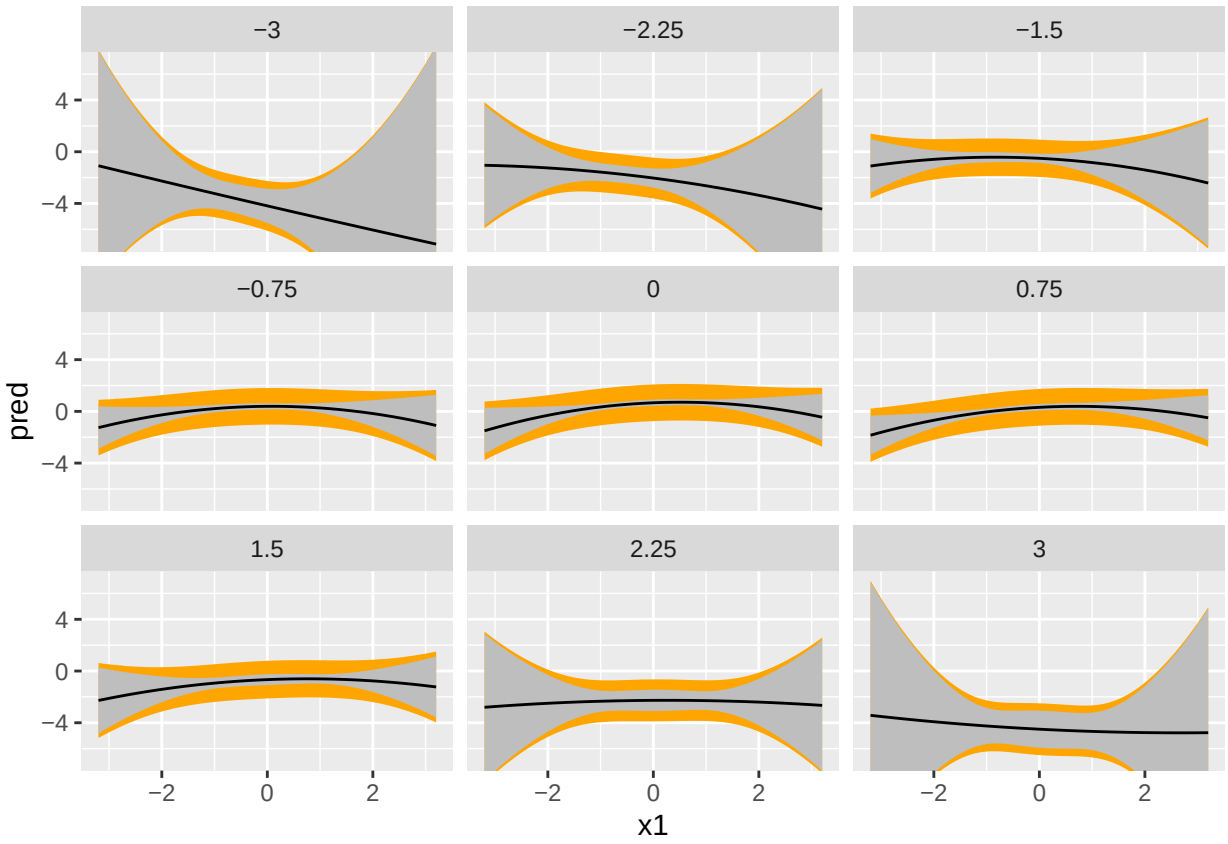


SOLUTION

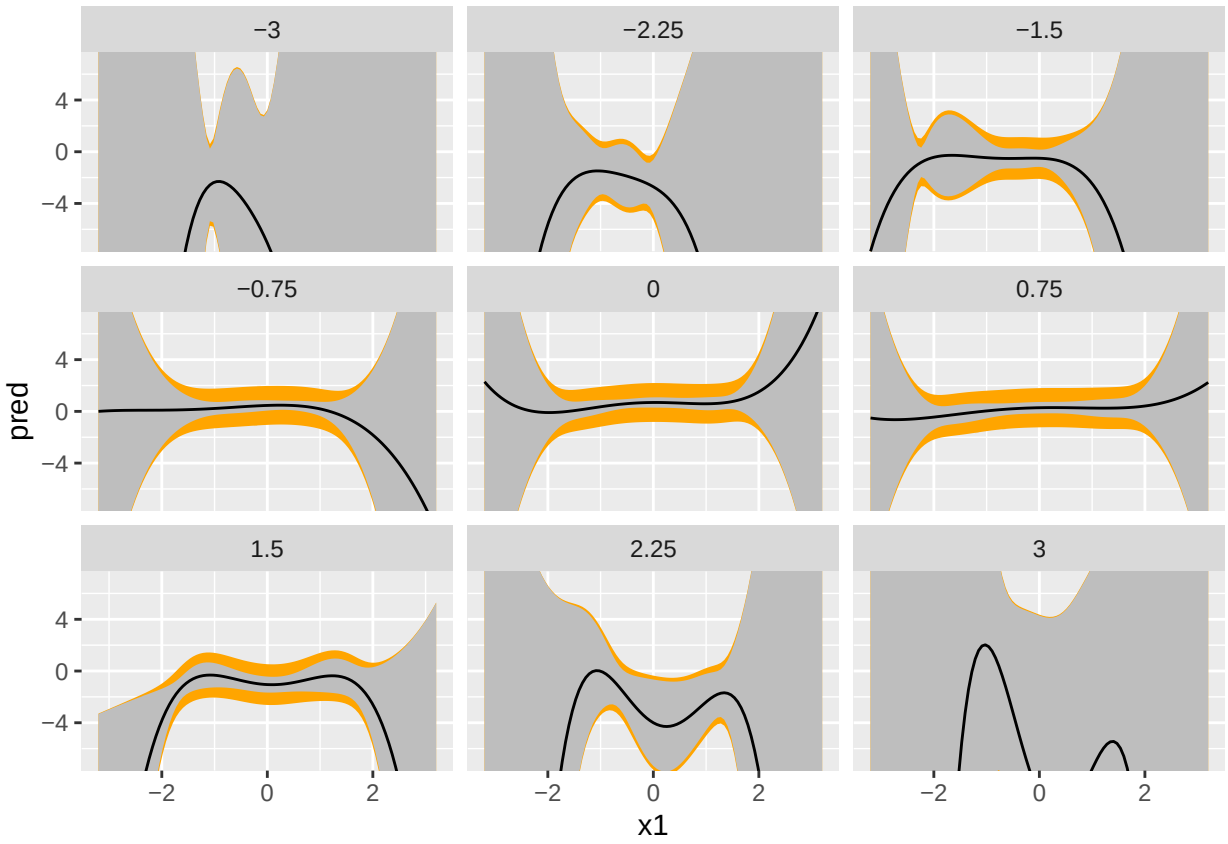
```
pred_lm_02 %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=pred_lwr, ymax=pred_upr), fill="orange")+
  geom_ribbon(aes(ymin=ci_lwr, ymax=ci_upr), fill="grey")+
  geom_line(aes(y=pred)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```



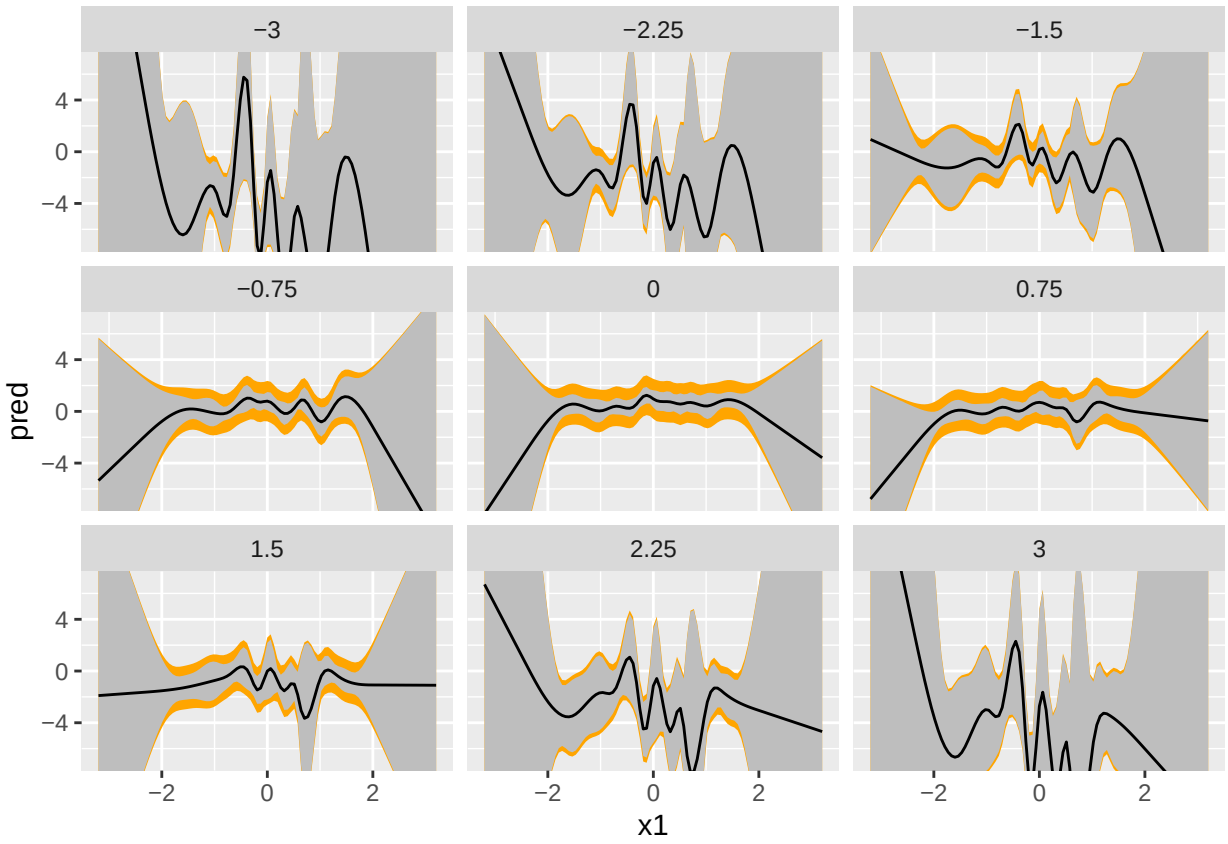

```
pred_lm_03 %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=pred_lwr, ymax=pred_upr), fill="orange")+
  geom_ribbon(aes(ymin=ci_lwr, ymax=ci_upr), fill="grey")+
  geom_line(aes(y=pred)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```



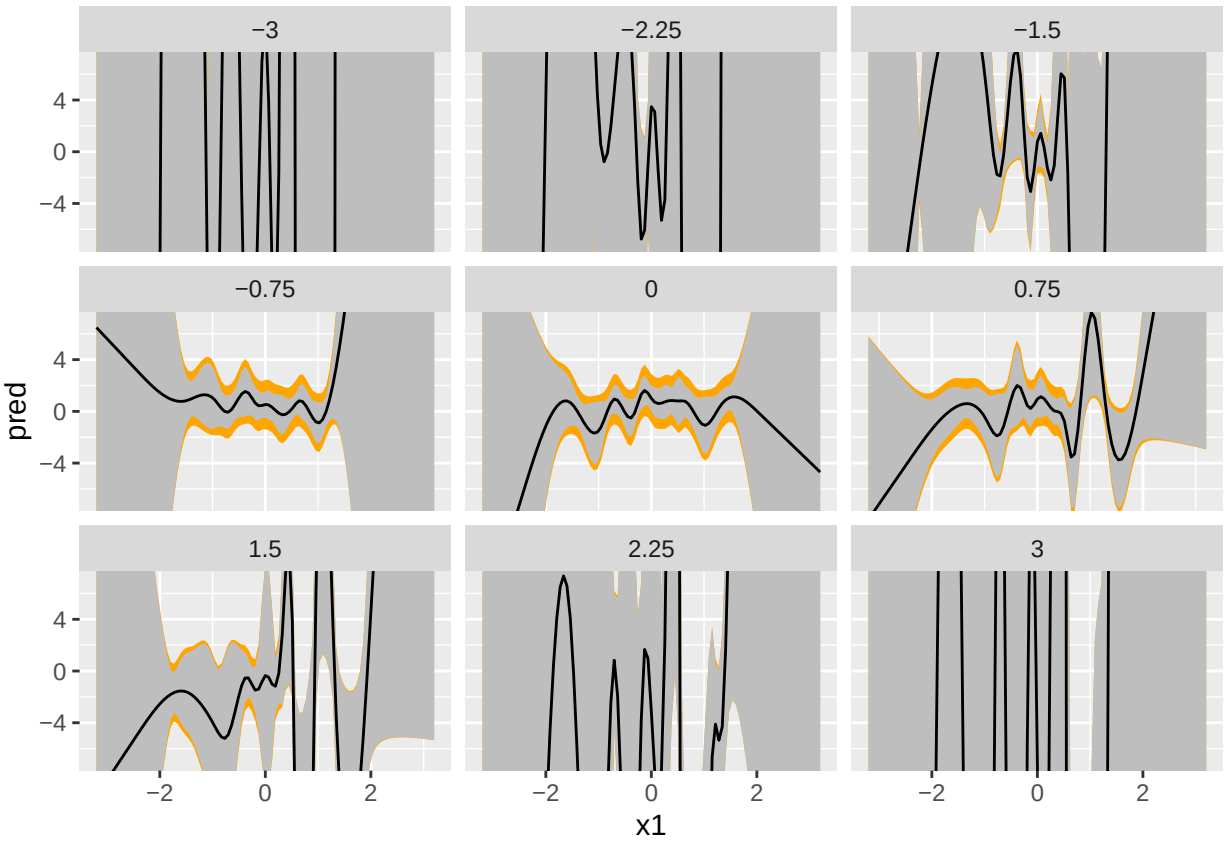

```
pred_lm_04 %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=pred_lwr, ymax=pred_upr), fill="orange")+
  geom_ribbon(aes(ymin=ci_lwr, ymax=ci_upr), fill="grey")+
  geom_line(aes(y=pred)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```




```
pred_lm_05 %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=pred_lwr, ymax=pred_upr), fill="orange")+
  geom_ribbon(aes(ymin=ci_lwr, ymax=ci_upr), fill="grey")+
  geom_line(aes(y=pred)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```




```
pred_lm_06 %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=pred_lwr, ymax=pred_upr), fill="orange")+
  geom_ribbon(aes(ymin=ci_lwr, ymax=ci_upr), fill="grey")+
  geom_line(aes(y=pred)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```



2d)

Do you feel the predictions are consistent with the model performance rankings based on AIC/BIC? What is the defining characteristic of the models considered to be the worst by AIC/BIC?

SOLUTION What do you think?

1) Based on the AIC/BIC results in Problem 1h), the best model was Model 3. However, when we see the plots in this question, Model 1 and Model 2 seems to perform better because their confidence and prediction intervals are narrower.
2) Models 4 5 and 6 were the worst performing models according to the AIC/BIC results in Problem 1h). The plots in this question imply similar results because their confidence and prediction intervals are wider.
3) When you observe models that have a lot of terms (polynomial terms, or many spline knots) but don't offer a substantial improvement in the explained variance of the response variable, these models are likely candidates for being penalized by AIC/BIC and considered "worse."

Problem 03

Now that you have fit non-Bayesian linear models with maximum likelihood estimation, it is time to use Bayesian models to understand the influence of the prior on the model behavior.

Regardless of your answers in Problem 02 you will only work with model 3 and model 6 in this problem.

3a)

You will perform the Bayesian analysis using the Laplace Approximation just as you did in the previous assignment. You will define the log-posterior function just as you did in the previous assignment and so before doing so you must create the list of required information. This list will include the observed response, the design matrix, and the prior specification. You will use independent Gaussian priors on the regression parameters with a shared prior mean and shared prior standard deviation. You will use an Exponential prior on the unknown likelihood noise (the σ parameter).

Complete the two code chunks below. In the first, create the design matrix following `mod03`'s formula, and assign the object to the `X03` variable. Complete the `info_03_weak` list by assigning the response to `yobs` and the design matrix to `design_matrix`. Specify the shared prior mean, `mu_beta`, to be 0, the shared prior standard deviation, `tau_beta`, as 50, and the rate parameter on the noise, `sigma_rate`, to be 1.

Complete the second code chunk with the same prior specification. The second code chunk however requires that you create the design matrix associated with `mod06`'s formula and assign the object to the `X06` variable. Assign `X06` to the `design_matrix` field of the `info_06_weak` list.

```
X03 <- model.matrix (y ~ (x1 + I(x1^2)) * (x2 + I(x2^2)), data = df)

info_03_weak <- list(
  yobs = df %>% pull(y) %>% as.matrix(),
  design_matrix = X03,
  mu_beta = 0,
  tau_beta = 50,
  sigma_rate = 1
)
```

```
X06 <- model.matrix (y ~ splines::ns(x1, 12) * (x2 + I(x2^2) + I(x2^3) + I(x2^4)), data = df)

info_06_weak <- list(
  yobs = df %>% pull(y) %>% as.matrix(),
  design_matrix = X06,
  mu_beta = 0,
  tau_beta = 50,
  sigma_rate = 1
)
```

SOLUTION

3b)

You will now define the log-posterior function `lm_logpost()`. You will continue to use the log-transformation on σ, and so you will actually define the log-posterior in terms of the mean trend β-parameters and the unbounded noise parameter, $\varphi = \log[\sigma]$.

The comments in the code chunk below tell you what you need to fill in. The unknown parameters to learn are contained within the first input argument, `unknowns`. You will assume that the unknown β-parameters are listed before the unknown φ parameter in the `unknowns` vector. You must specify the number of β parameters programmatically to allow scaling up your function to an arbitrary number of unknowns. You will assume that all variables contained in the `my_info` list (the second argument to `lm_logpost()`) are the same fields in the `info_03_weak` list you defined in Problem 3a).

Define the log-posterior function by completing the code chunk below. You must calculate

the mean trend, `mu`, using matrix math between the design matrix and the unknown β column vector.

HINT: This function should look very famaliar...

```
lm_logpost <- function(unknowns, my_info)
{
  # specify the number of unknown beta parameters
  length_beta <- length(unknowns)-1

  # extract the beta parameters from the `unknowns` vector
  beta_v <- unknowns[1:length_beta]

  # extract the unbounded noise parameter, varphi
  lik_varphi <- unknowns[length(unknowns)]

  # back-transform from varphi to sigma
  lik_sigma <- exp(lik_varphi)

  # extract design matrix
  X <- my_info$design_matrix

  # calculate the linear predictor
  mu <- X%*%beta_v

  # evaluate the log-likelihood
  log_lik <- sum(dnorm(x = my_info$yobs,
                       mean = mu,
                       sd = lik_sigma,
                       log = TRUE))

  # evaluate the log-prior
  log_prior_beta <- sum(dnorm(x = beta_v,
                              mean = my_info$mu_beta,
                              sd = my_info$tau_beta,
                              log = TRUE))

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

  # add the mean trend prior and noise prior together
  log_prior <- log_prior_beta+log_prior_sigma

  # account for the transformation
  log_derive_adjust <- lik_varphi

  # sum together
  log_lik+log_prior+log_derive_adjust
}
```

SOLUTION

3c)

The `my_laplace()` function is defined for you in the code chunk below. This function executes the laplace approximation and returns the object consisting of the posterior mode, posterior covariance matrix, and the log-evidence.

```
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$par
  post_var_matrix <- -solve(fit$hessian)
  p <- length(mode)
  int <- p/2 * log(2 * pi) + 0.5 * log(det(post_var_matrix)) + logpost_func(mode, ...)
  # package all of the results into a list
  list(mode = mode,
       var_matrix = post_var_matrix,
       log_evidence = int,
       converge = ifelse(fit$convergence == 0,
                         "YES",
                         "NO"),
       iter_counts = as.numeric(fit$counts[1]))
}
```

Execute the Laplace Approximation for the model 3 formulation and the model 6 formulation. Assign the model 3 result to the `laplace_03_weak` object, and assign the model 6 result to the `laplace_06_weak` object. Check that the optimization scheme converged.

```
num_beta_params03 <- ncol(X03)

init_beta03 <- rnorm(num_beta_params03)

init_varphi03 <- log(rexp(1))

laplace_03 <- my_laplace(c(init_beta03, init_varphi03), lm_logpost, info_03_weak)

laplace_03$converge
```

SOLUTION

```
## [1] "YES"
```

```
num_beta_params06 <- ncol(X06)

init_beta06 <- rnorm(num_beta_params06)

init_varphi06 <- log(rexp(1))
```

```
  laplace_06 <- my_laplace(c(init_beta06, init_varphi06), lm_logpost, info_06_weak)

  laplace_06$converge
```

```
## [1] "YES"
```

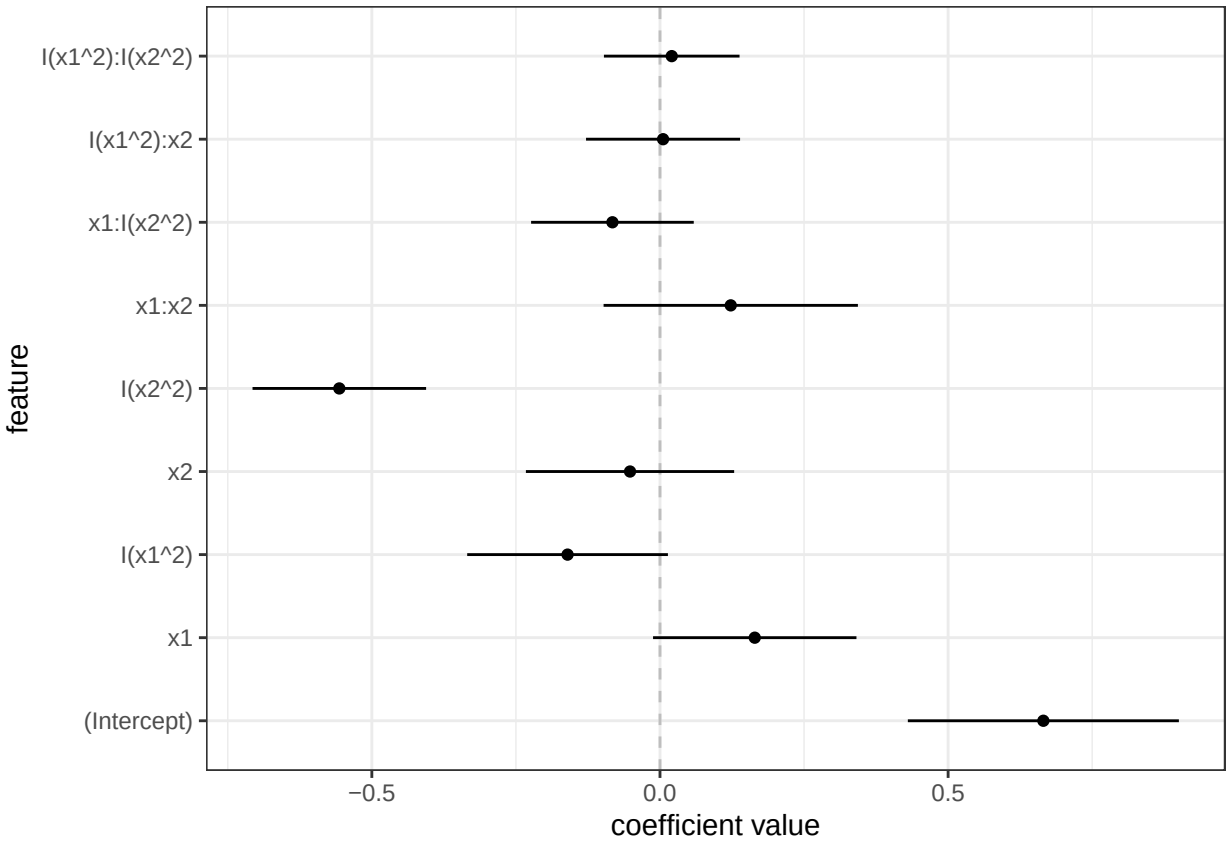
3d)

A function is defined for you in the code chunk below. This function creates a coefficient summary plot in the style of the `coefplot()` function, but uses the Bayesian results from the Laplace Approximation. The first argument is the vector of posterior means, and the second argument is the vector of posterior standard deviations. The third argument is the name of the feature associated with each coefficient.

```
viz_post_coefs <- function(post_means, post_sds, xnames)
{
  tibble::tibble(
    mu = post_means,
    sd = post_sds,
    x = xnames
  ) %>%
    mutate(x = factor(x, levels = xnames)) %>%
    ggplot(mapping = aes(x = x)) +
    geom_hline(yintercept = 0, color = 'grey', linetype = 'dashed') +
    geom_point(mapping = aes(y = mu)) +
    geom_linerange(mapping = aes(ymin = mu - 2 * sd,
                                 ymax = mu + 2 * sd,
                                 group = x)) +
    labs(x = 'feature', y = 'coefficient value') +
    coord_flip() +
    theme_bw()
}
```

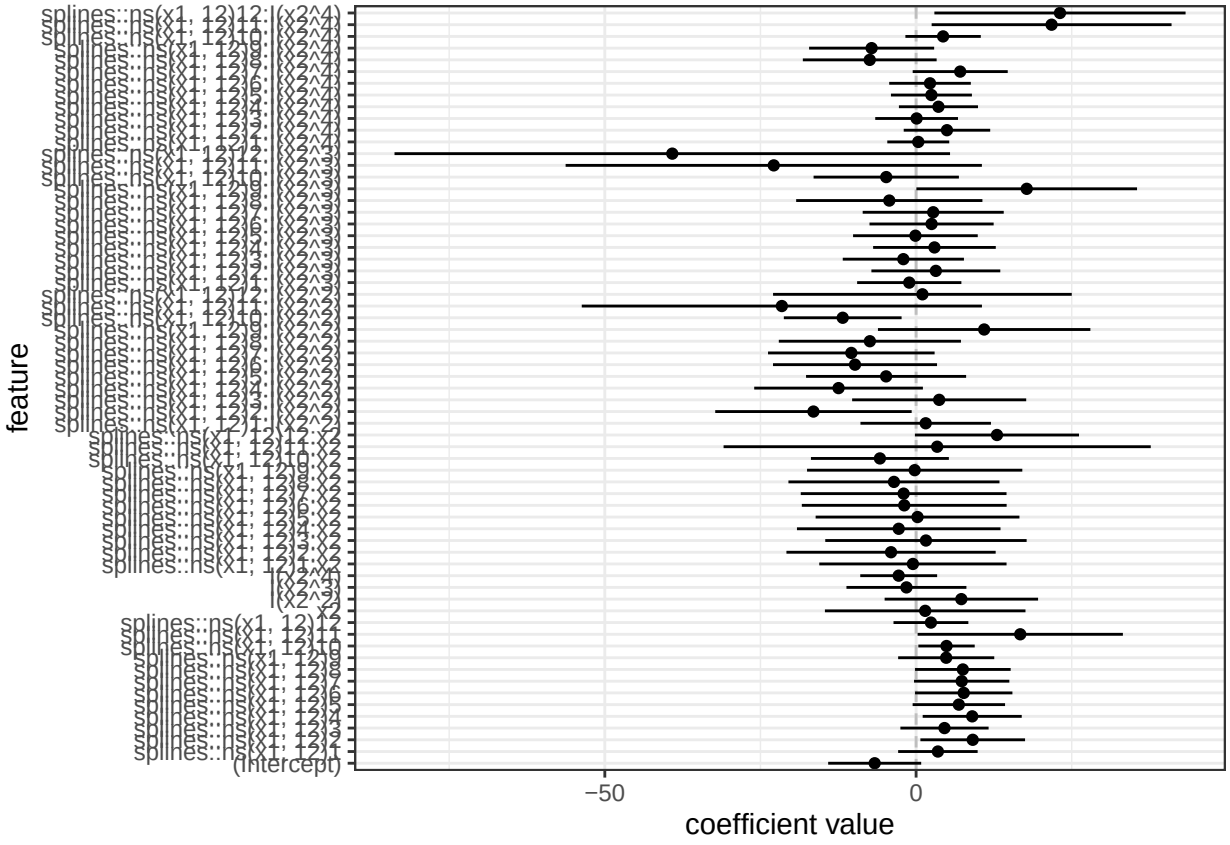
Create the posterior summary visualization figure for model 3 and model 6. You must provide the posterior means and standard deviations of the regression coefficients (the β parameters). Do NOT include the φ parameter. The feature names associated with the coefficients can be extracted from the design matrix using the `colnames()` function.

```
post_means_03 <- laplace_03$mode
post_sds_03 <- sqrt(diag(laplace_03$var_matrix))
feature_names_03 <- colnames(X03)

viz_post_coefs(post_means_03[-length(post_means_03)], post_sds_03[-length(post_sds_03)], feature_names_
```



SOLUTION

```
post_means_06 <- laplace_06$mode
post_sds_06 <- sqrt(diag(laplace_06$var_matrix))
feature_names_06 <- colnames(X06)

viz_post_coefs(post_means_06[-length(post_means_06)], post_sds_06[-length(post_sds_06)], feature_names_
```



3e)

Use the Bayes Factor to identify the better of the models.

```
exp(laplace_03$log_evidence)/exp(laplace_06$log_evidence)
```

SOLUTION

```
## [1] 1.473105e+88
```

The above code calculates the Bayes Factor of the models. Since the Bayes Factor is much greater than 1 it is expected that Model 3 is more plausible than Model 6.

3f)

You fit the Bayesian models assuming a diffuse or *weak* prior. Let's now try a more informative or *strong* prior by reducing the prior standard deviation on the regression coefficients from 50 to 1. The prior mean will still be zero.

Complete the first code chunk below, which defines the list of required information for both the model 3 and model 6 formulations using the strong prior on the regression coefficients. All other information, data and the σ prior, are the same as before.

Run the Laplace Approximation using the strong prior for both the model 3 and model 6 formulations. Assign the results to `laplace_03_strong` and `laplace_06_strong`.

Confirm that the optimizations converged for both laplace approximation results.

SOLUTION Define the lists of required information for the strong prior.

```
info_03_strong <- list(
  yobs = df %>% pull(y) %>% as.matrix(),
  design_matrix = X03,
  mu_beta = 0,
  tau_beta = 1,
  sigma_rate = 1
)

info_06_strong <- list(
  yobs = df %>% pull(y) %>% as.matrix(),
  design_matrix = X06,
  mu_beta = 0,
  tau_beta = 1,
  sigma_rate = 1
)
```

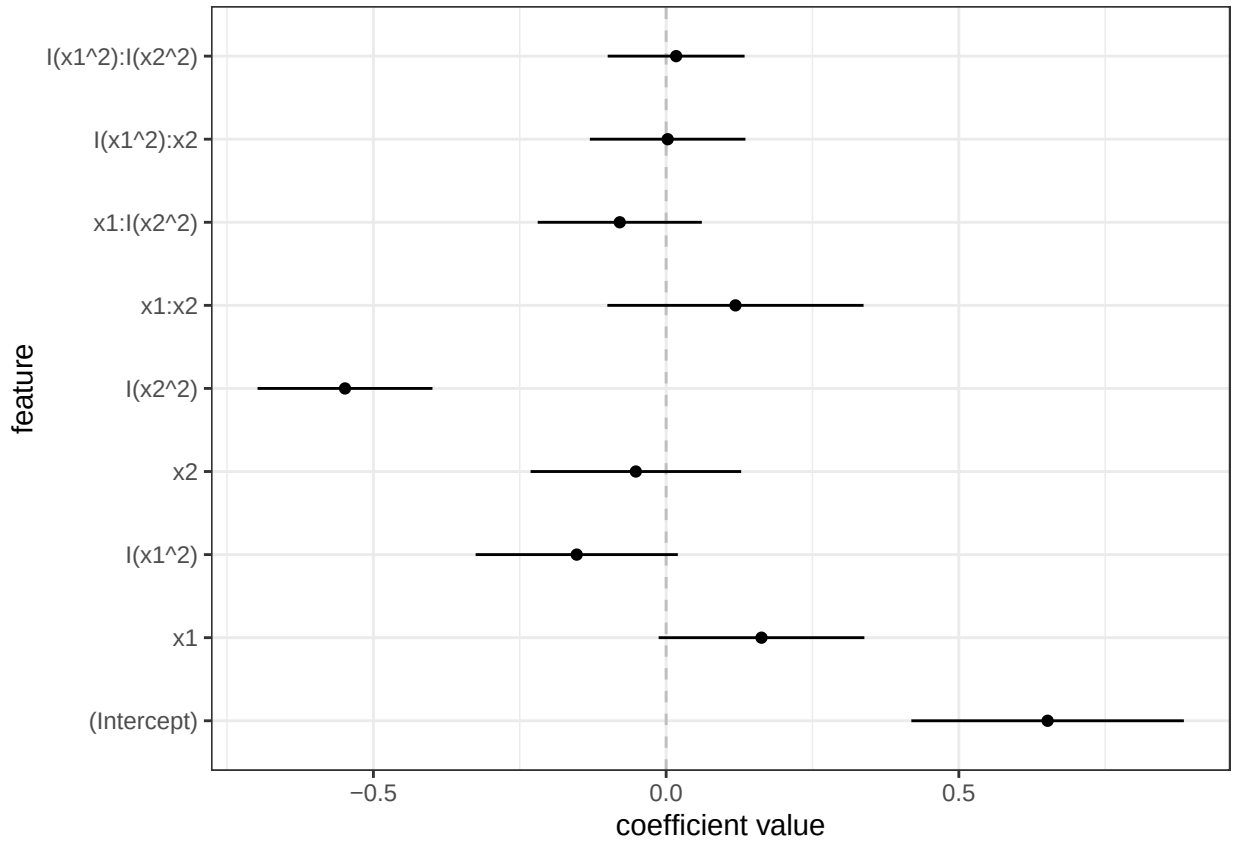
Execute the Laplace Approximation.

```
laplace_03_strong <- my_laplace(c(init_beta03, init_varphi03), lm_logpost, info_03_strong)
laplace_06_strong <- my_laplace(c(init_beta06, init_varphi06), lm_logpost, info_06_strong)
```

3g)

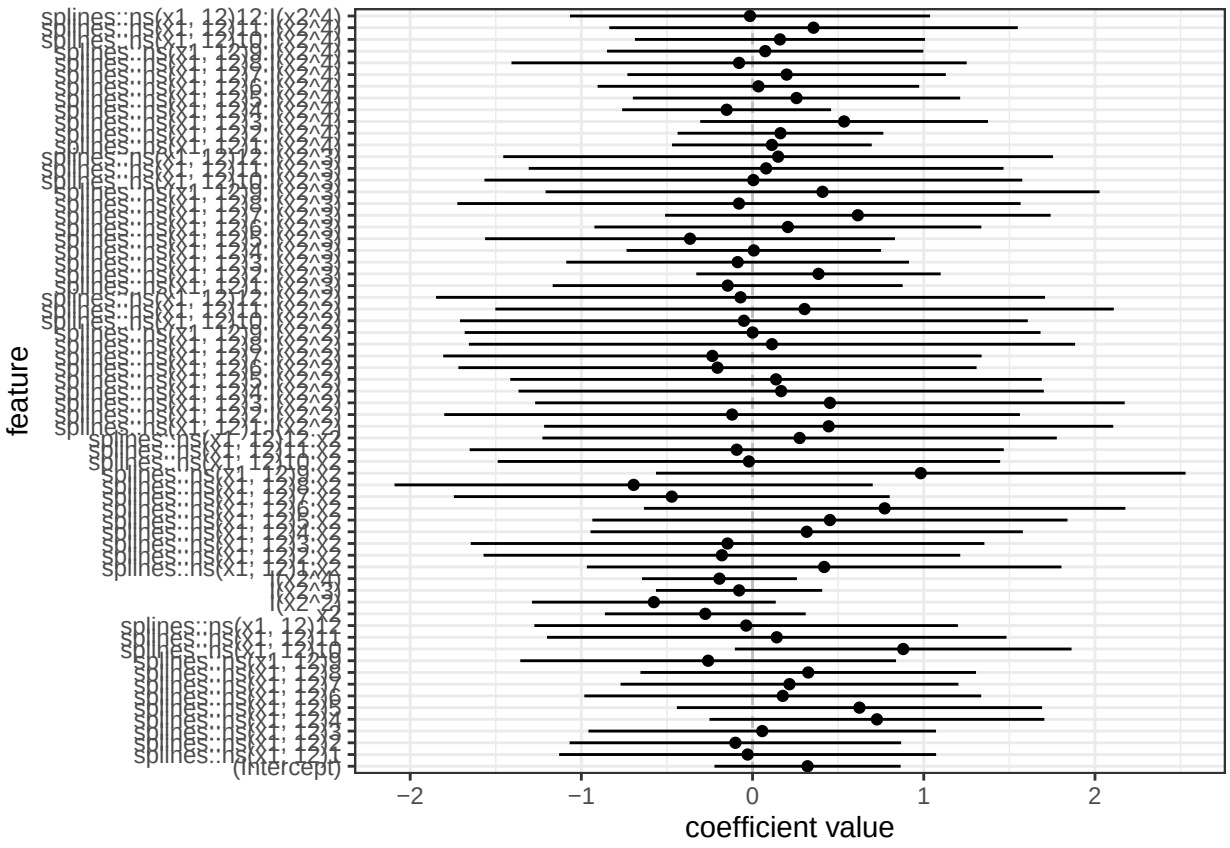
Use the `viz_post_coefs()` function to visualize the posterior coefficient summaries for model 3 and model 6, based on the strong prior specification.

```
post_means_03_strong <- laplace_03_strong$mode
post_sds_03_strong <- sqrt(diag(laplace_03_strong$var_matrix))

viz_post_coefs(post_means_03_strong[-length(post_means_03_strong)], post_sds_03_strong[-length(post_sds
```



SOLUTION

```
post_means_06_strong <- laplace_06_strong$mode
post_sds_06_strong <- sqrt(diag(laplace_06_strong$var_matrix))

viz_post_coefs(post_means_06_strong[-length(post_means_06_strong)], post_sds_06_strong[-length(post_sds
```



3h)

You will fit one more set of Bayesian models with a very strong prior on the regression coefficients. The prior standard deviation will be equal to 1/50.

Complete the first code chunk below, which defines the list of required information for both the model 3 and model 6 formulations using the very strong prior on the regression coefficients. All other information, data and the σ prior, are the same as before.

Run the Laplace Approximation using the strong prior for both the model 3 and model 6 formulations. Assign the results to `laplace_03_very_strong` and `laplace_06_very_strong`.

Confirm that the optimizations converged for both laplace approximation results.

```
info_03_very_strong <- list(
  yobs = df %>% pull(y) %>% as.matrix(),
  design_matrix = X03,
  mu_beta = 0,
  tau_beta = 1/50,
  sigma_rate = 1
)

info_06_very_strong <- list(
  yobs = df %>% pull(y) %>% as.matrix(),
  design_matrix = X06,
  mu_beta = 0,
  tau_beta = 1/50,
```

```
  sigma_rate = 1
)
```

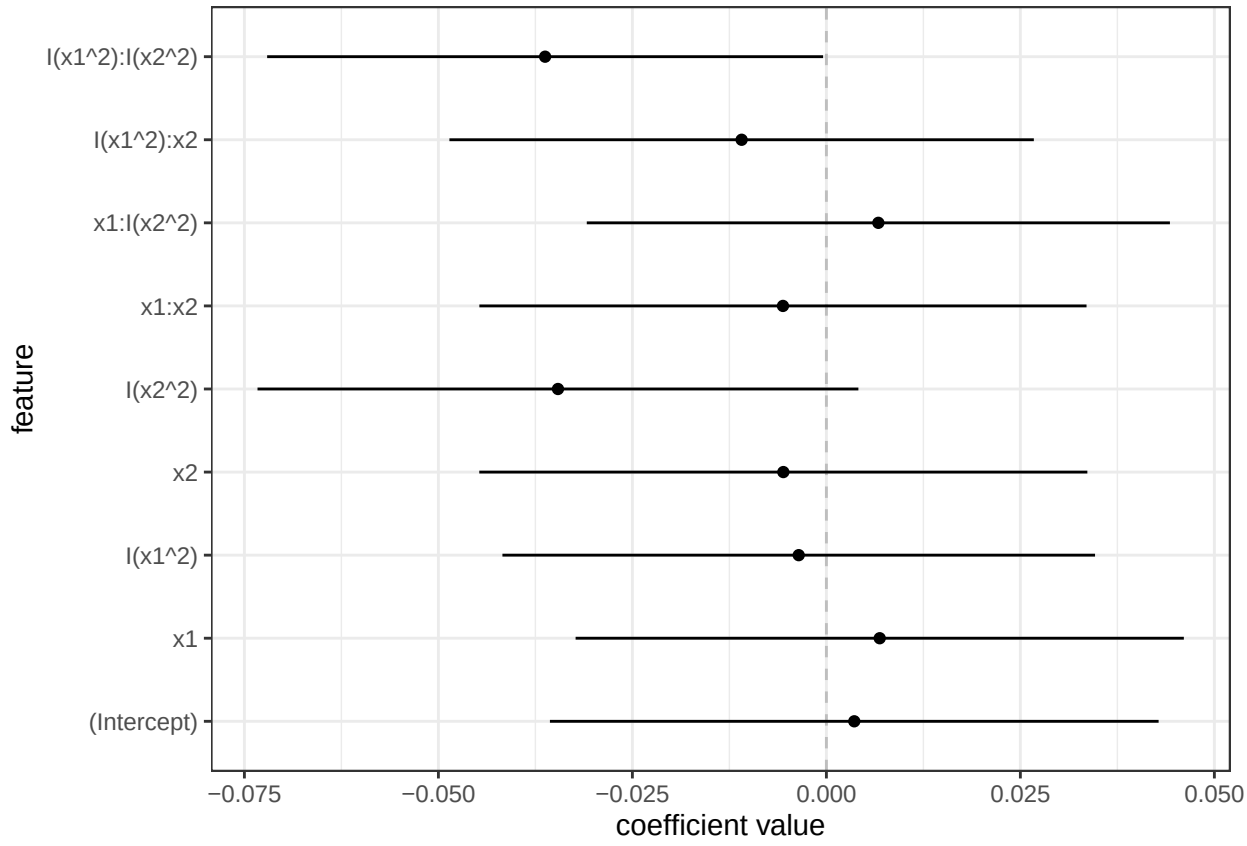
SOLUTION Execute the Laplace Approximation.

```
laplace_03_very_strong <- my_laplace(c(init_beta03, init_varphi03), lm_logpost, info_03_very_strong)
laplace_06_very_strong <- my_laplace(c(init_beta06, init_varphi06), lm_logpost, info_06_very_strong)
```

3i)

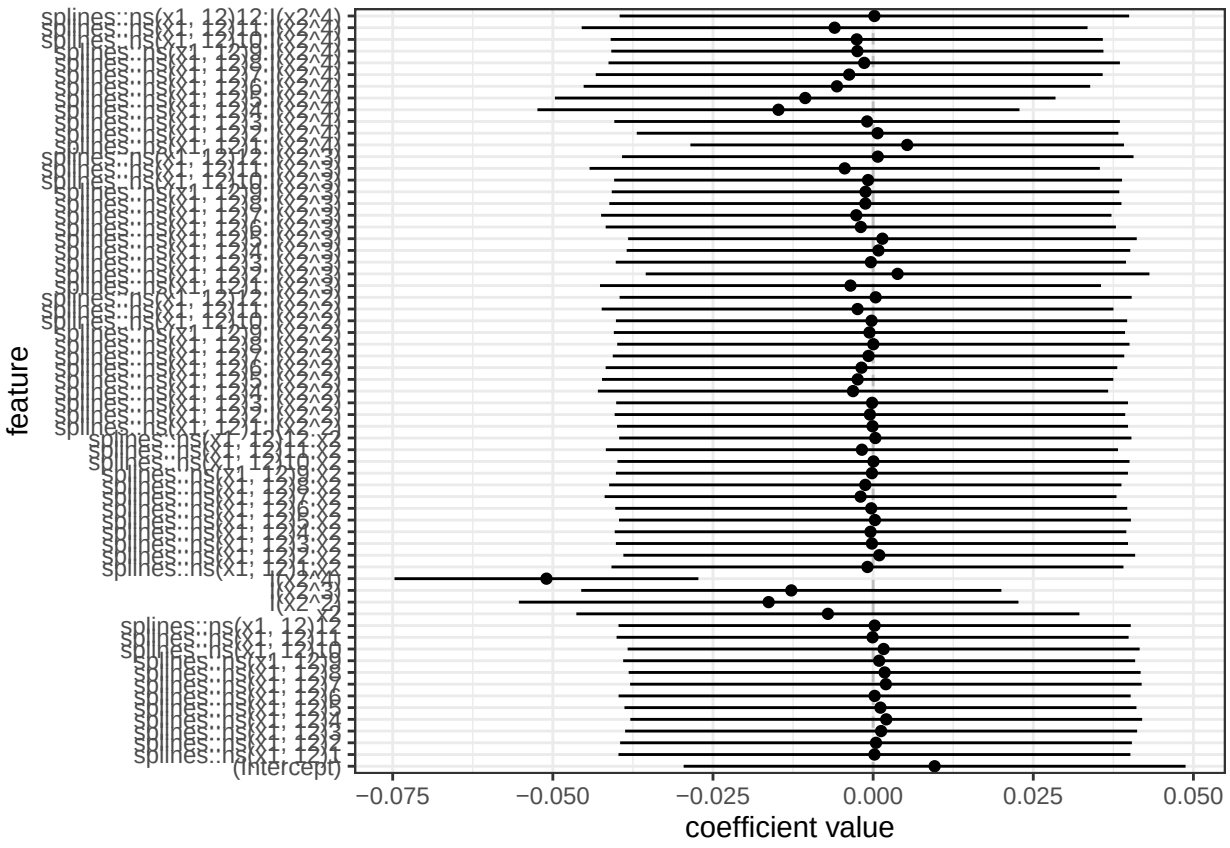
Use the `viz_post_coefs()` function to visualize the posterior coefficient summaries for model 3 and model 6, based on the very strong prior specification.

```
post_means_03_very_strong <- laplace_03_very_strong$mode
post_sds_03_very_strong <- sqrt(diag(laplace_03_very_strong$var_matrix))

viz_post_coefs(post_means_03_very_strong[-length(post_means_03_very_strong)], post_sds_03_very_strong[-
```



SOLUTION

```
post_means_06_very_strong <- laplace_06_very_strong$mode
post_sds_06_very_strong <- sqrt(diag(laplace_06_very_strong$var_matrix))

viz_post_coefs(post_means_06_very_strong[-length(post_means_06_very_strong)], post_sds_06_very_strong[-
```



3j)

Describe the influence of the regression coefficient prior standard deviation on the coefficient posterior distributions.

SOLUTION What do you think?

1) When `tau_beta` is decreased from 50 to 1, it did not effect Model-3 much, however, the confidence intervals in Model-6 decreased and posterior values got closer to 0.

2) When `tau_beta` is decreased from 1 to 1/50, it affected both models. In Model-3 and Model-6, the confidence intervals of the features decreased and posterior values got closer to 0.

3k)

You previously compared the two models using the Bayes Factor based on the weak prior specification.

Compare the performance of the two models with Bayes Factors again, but considering the results based on the strong and very strong priors. Does the prior influence which model is considered to be better?

```
exp(laplace_03_strong$log_evidence)/exp(laplace_06_strong$log_evidence)
```

SOLUTIOn

```
## [1] 2.67001e+13
```

```
exp(laplace_03_very_strong$log_evidence)/exp(laplace_06_very_strong$log_evidence)
```

```
## [1] 0.0002438025
```

1) The first result above gives the Bayes Factor based on the strong priors, while, the second result above gives the Bayes Factor based on the very strong priors.

2) Based on the results we can conclude that strong priors did not effect the determination of the better approach. However, very strong prior determines the better approach differently by showing Model-6 as the better method.

Problem 04

You examined the behavior of the coefficient posterior based on the influence of the prior. Let's now consider the prior's influence by examining the posterior predictive distributions.

4a)

You will make posterior predictions following the approach from the previous assignment. Posterior samples are generated and those samples are used to calculate the posterior samples of the mean trend and generate random posterior samples of the response around the mean. In the previous assignment, you made posterior predictions in order to calculate errors. In this assignment, you will not calculate errors, instead you will summarize the posterior predictions of the mean and of the random response.

The `generate_lm_post_samples()` function is defined for you below. It uses the `MASS::mvrnorm()` function generate posterior samples from the Laplace Approximation's MVN distribution.

```
generate_lm_post_samples <- function(mvn_result, length_beta, num_samples)
{
  MASS::mvrnorm(n = num_samples,
                mu = mvn_result$mode,
                Sigma = mvn_result$var_matrix) %>%
    as.data.frame() %>% tibble::as_tibble() %>%
    purrr::set_names(c(sprintf("beta_%02d", 0:(length_beta-1)), "varphi")) %>%
    mutate(sigma = exp(varphi))
}
```

The code chunk below starts the `post_lm_pred_samples()` function. This function generates posterior mean trend predictions and posterior predictions of the response. The first argument, `Xnew`, is a potentially new or test design matrix that we wish to make predictions at. The second argument, `Bmat`, is a matrix of posterior samples of the β-parameters, and the third argument, `sigma_vector`, is a vector of posterior samples of the likelihood noise. The `Xnew` matrix has rows equal to the number of predictions points, `M`, and the `Bmat` matrix has rows equal to the number of posterior samples `S`.

You must complete the function by performing the necessary matrix math to calculate the matrix of posterior mean trend predictions, `Umat`, and the matrix of posterior response predictions, `Ymat`. You must also complete missing arguments to the definition of the `Rmat` and `Zmat` matrices. The `Rmat` matrix replicates the posterior likelihood noise samples the correct number of times. The `Zmat` matrix is the matrix of randomly generated standard normal values. You must correctly specify the required number of rows to the `Rmat` and `Zmat` matrices.

The `post_lm_pred_samples()` returns the `Umat` and `Ymat` matrices contained within a list.

Perform the necessary matrix math to calculate the matrix of posterior predicted mean trends `Umat` and posterior predicted responses `Ymat`. You must specify the number of required rows to create the `Rmat` and `Zmat` matrices.

HINT: The following code chunk should look famaliar...

```
post_lm_pred_samples <- function(Xnew, Bmat, sigma_vector)
{
  # number of new prediction locations
  M <- nrow(Xnew)
  # number of posterior samples
  S <- nrow(Bmat)

  # matrix of linear predictors
  Umat <- Xnew %*% t(Bmat)

  # assmeble matrix of sigma samples, set the number of rows
  Rmat <- matrix(rep(sigma_vector, M), M, byrow = TRUE)

  # generate standard normal and assemble into matrix
  # set the number of rows
  Zmat <- matrix(rnorm(M*S), M, byrow = TRUE)

  # calculate the random observation predictions
  Ymat <- Umat + Rmat * Zmat

  # package together
  list(Umat = Umat, Ymat = Ymat)
}
```

SOLUTION

4b)

Since this assignment is focused on visualizing the predictions, we will summarize the posterior predictions to focus on the posterior means and the middle 95% uncertainty intervals. The code chunk below is defined for you which serves as a useful wrapper function to call `post_lm_pred_samples()`.

```
make_post_lm_pred <- function(Xnew, post)
{
  Bmat <- post %>% select(starts_with("beta_")) %>% as.matrix()

  sigma_vector <- post %>% pull(sigma)

  post_lm_pred_samples(Xnew, Bmat, sigma_vector)
}
```

The code chunk below defines a function `summarize_lm_pred_from_laplace()` which manages the actions necessary to summarize posterior predictions. The first argument, `mvn_result`, is the Laplace Approximation object. The second object is the test design matrix, `Xtest`, and the third argument, `num_samples`, is the number of posterior samples to make.

You must complete the code chunk below which summarizes the posterior predictions. This function takes care of most of the coding for you. You do not have to worry about the generation of the posterior samples OR calculating the posterior quantiles associated with the middle 95% uncertainty interval. You must calculate the posterior average by deciding on whether you should use `colMeans()` or `rowMeans()` to calculate the average across all posterior samples per prediction location.

Follow the comments in the code chunk below to complete the definition of the summarize_lm_pred_from_laplace() function. You must calculate the average posterior mean trend and the average posterior response.

```r
summarize_lm_pred_from_laplace <- function(mvn_result, Xtest, num_samples)
{
  # generate posterior samples of the beta parameters
  post <- generate_lm_post_samples(mvn_result, ncol(Xtest), num_samples)

  # make posterior predictions on the test set
  pred_test <- make_post_lm_pred(Xtest, post)

  # calculate summary statistics on the predicted mean and response
  # summarize over the posterior samples

  # posterior mean, should you summarize along rows (rowMeans) or
  # summarize down columns (colMeans) ???
  mu_avg <- rowMeans(pred_test$Umat)
  y_avg <- rowMeans(pred_test$Ymat)

  # posterior quantiles for the middle 95% uncertainty intervals
  mu_lwr <- apply(pred_test$Umat, 1, stats::quantile, probs = 0.025)
  mu_upr <- apply(pred_test$Umat, 1, stats::quantile, probs = 0.975)
  y_lwr <- apply(pred_test$Ymat, 1, stats::quantile, probs = 0.025)
  y_upr <- apply(pred_test$Ymat, 1, stats::quantile, probs = 0.975)

  # book keeping
  tibble::tibble(
    mu_avg = mu_avg,
    mu_lwr = mu_lwr,
    mu_upr = mu_upr,
    y_avg = y_avg,
    y_lwr = y_lwr,
    y_upr = y_upr
  ) %>%
    tibble::rowid_to_column("pred_id")
}
```

SOLUTION

4c)

When you made predictions in Problem 02, the `lm()` object handled making the test design matrix. However, since we have programmed the Bayesian modeling approach from scratch we need to create the test design matrix manually.

Create the test design matrix based on the visualization grid, `viz_grid`, using the model 3 formulation. Assign the result to the `X03_test` object.

Call the `summarize_lm_pred_from_laplace()` function to summarize the posterior predictions from the model 3 formulation for the weak, strong, and very strong prior specifications. Use 5000 posterior samples for each case. Assign the results from the weak prior to `post_pred_summary_viz_03_weak`, the results from the strong prior to `post_pred_summary_viz_03_strong`, and the results from the very strong prior to `post_pred_summary_viz_03_very_strong`.

```r
X03_test <- model.matrix( ~ (x1 + I(x1^2)) * (x2 + I(x2^2)), viz_grid)
```

```
post_pred_summary_viz_03_weak <- summarize_lm_pred_from_laplace(laplace_03, X03_test, 5000)

post_pred_summary_viz_03_strong <- summarize_lm_pred_from_laplace(laplace_03_strong, X03_test, 5000)

post_pred_summary_viz_03_very_strong <- summarize_lm_pred_from_laplace(laplace_03_very_strong, X03_test
```

SOLUTION

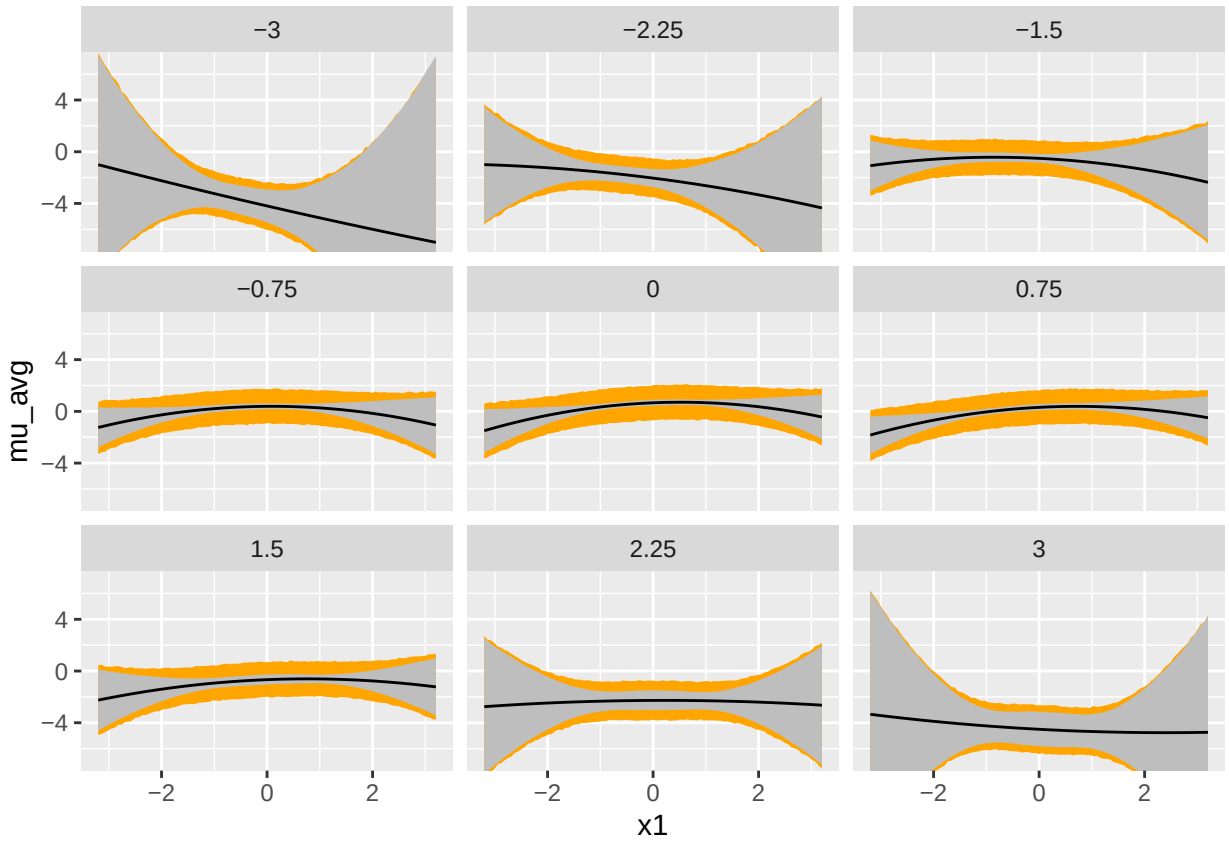
4d)

You will now visualize the posterior predictions from the model 3 Bayesian models associated with the weak, strong, and very strong priors. The `viz_grid` object is joined to the prediction dataframes assuming you have used the correct variable names!

Visualize the predicted means, confidence intervals, and prediction intervals in the style of those that you created in Problem 02. The confidence interval bounds are `mu_lwr` and `mu_upr` columns and the prediction interval bounds are the `y_lwr` and `y_upr` columns, respectively. The posterior predicted mean of the mean is `mu_avg`.

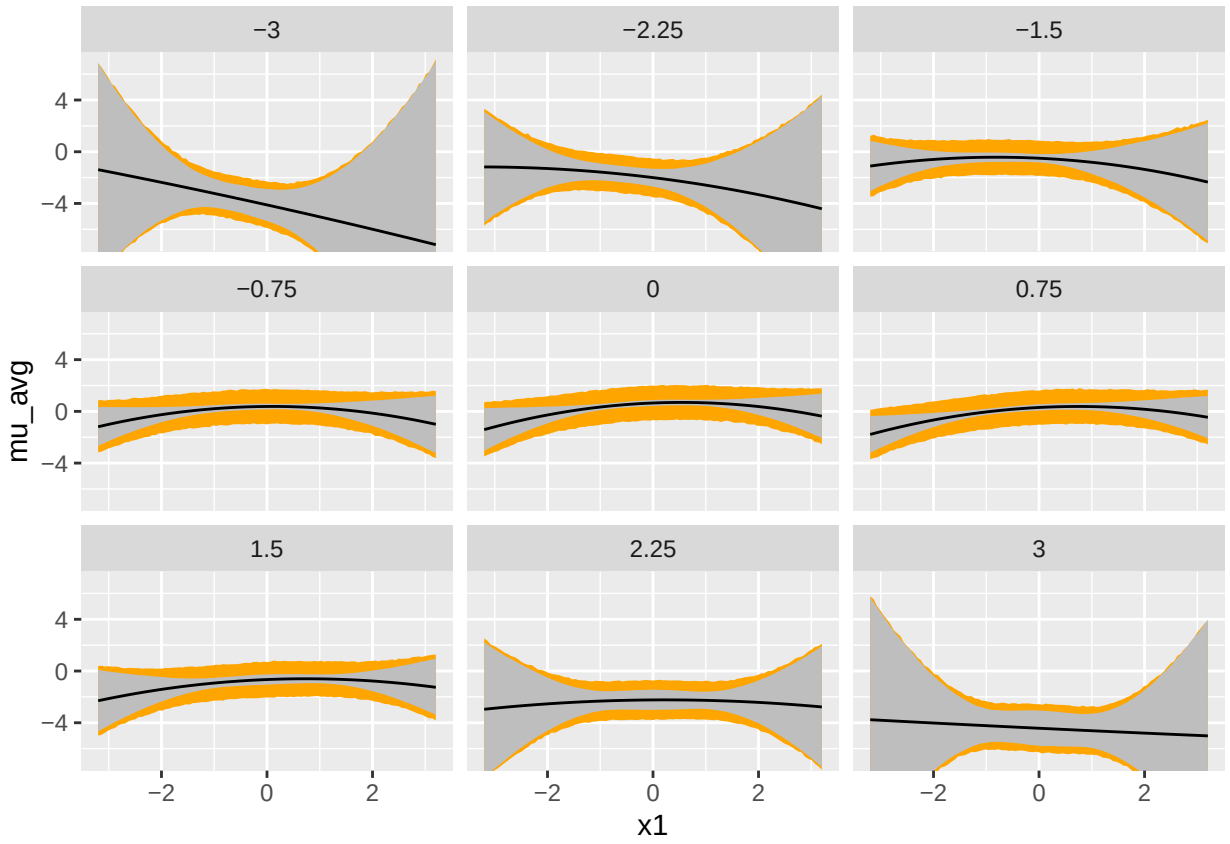
Pipe the result of the joined dataframe into `ggplot()` and make appropriate aesthetics and layers to visualize the predictions with the `x1` variable mapped to the `x` aesthetic and the `x2` variable used as a facet variable.

```
post_pred_summary_viz_03_weak %>%
  left_join(viz_grid %>% tibble::rowid_to_column("pred_id"),
            by = 'pred_id') %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=y_lwr, ymax=y_upr), fill="orange")+
  geom_ribbon(aes(ymin=mu_lwr, ymax=mu_upr), fill="grey")+
  geom_line(aes(y=mu_avg)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```

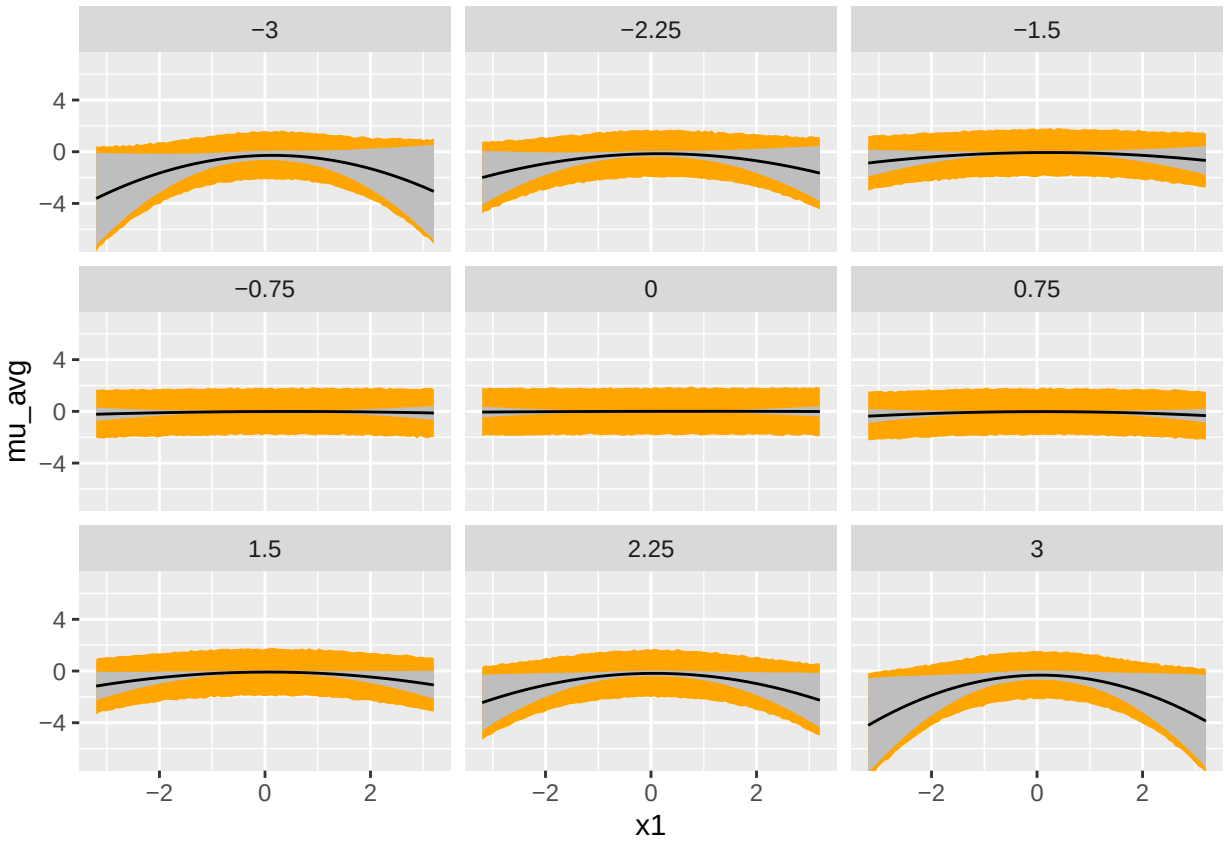


SOLUTION

```
post_pred_summary_viz_03_strong %>%
  left_join(viz_grid %>% tibble::rowid_to_column("pred_id"),
            by = 'pred_id')%>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=y_lwr, ymax=y_upr), fill="orange")+
  geom_ribbon(aes(ymin=mu_lwr, ymax=mu_upr), fill="grey")+
  geom_line(aes(y=mu_avg)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```




```
post_pred_summary_viz_03_very_strong %>%
  left_join(viz_grid %>% tibble::rowid_to_column("pred_id"),
            by = 'pred_id')%>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=y_lwr, ymax=y_upr), fill="orange")+
  geom_ribbon(aes(ymin=mu_lwr, ymax=mu_upr), fill="grey")+
  geom_line(aes(y=mu_avg)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```



4e)

In order to make posterior predictions for the model 6 formulation you must create a test design matrix consistent with the training set basis. The code chunk below creates a helper function which extracts the interior and boundary knots of a natural spline associated with the training set for you. The first argument, `J`, is the degrees-of-freedom (DOF) of the spline, the second argument, `train_data`, is the training data set. The third argument `xname` is the name of the variable you are applying the spline to. The `xname` argument **must** be provided as a character string.

```
make_splines_training_knots <- function(J, train_data, xname)
{
  # extract the input from the training set
  x <- train_data %>% select(all_of(xname)) %>% pull()

  # create the training basis
  train_basis <- splines::ns(x, df = J)

  # extract the knots
  interior_knots <- as.vector(attributes(train_basis)$knots)

  boundary_knots <- as.vector(attributes(train_basis)$Boundary.knots)

  # book keeping
  list(interior_knots = interior_knots,
       boundary_knots = boundary_knots)
}
```

Create the test design matrix based on the visualization grid, `viz_grid`, using the model 6

formulation. Assign the result to the `X06_test` object. Use the `make_splines_training_knots()` function to get the interior and boundary knots associated with the training set for the `x1` variable to create the test design matrix.

Call the `summarize_lm_pred_from_laplace()` function to summarize the posterior predictions from the model 6 formulation for the weak, strong, and very strong prior specifications. Use 5000 posterior samples for each case. Assign the results from the weak prior to `post_pred_summary_viz_06_weak`, the results from the strong prior to `post_pred_summary_viz_06_strong`, and the results from the very strong prior to `post_pred_summary_viz_06_very_strong`.

HINT: The `make_spline_training_knots()` function returns a list! The fields or elements of the list can be accessed via the `$` operator.

```
tr_knot_06 <- make_splines_training_knots(12,viz_grid,"x1")

X06_test <- model.matrix( ~ splines::ns(x1, knots = tr_knot_06$interior_knots, Boundary.knots = tr_knot_

post_pred_summary_viz_06_weak <- summarize_lm_pred_from_laplace(laplace_06, X06_test, 5000)

post_pred_summary_viz_06_strong <- summarize_lm_pred_from_laplace(laplace_06_strong, X06_test, 5000)

post_pred_summary_viz_06_very_strong <- summarize_lm_pred_from_laplace(laplace_06_very_strong, X06_test
```

SOLUTION

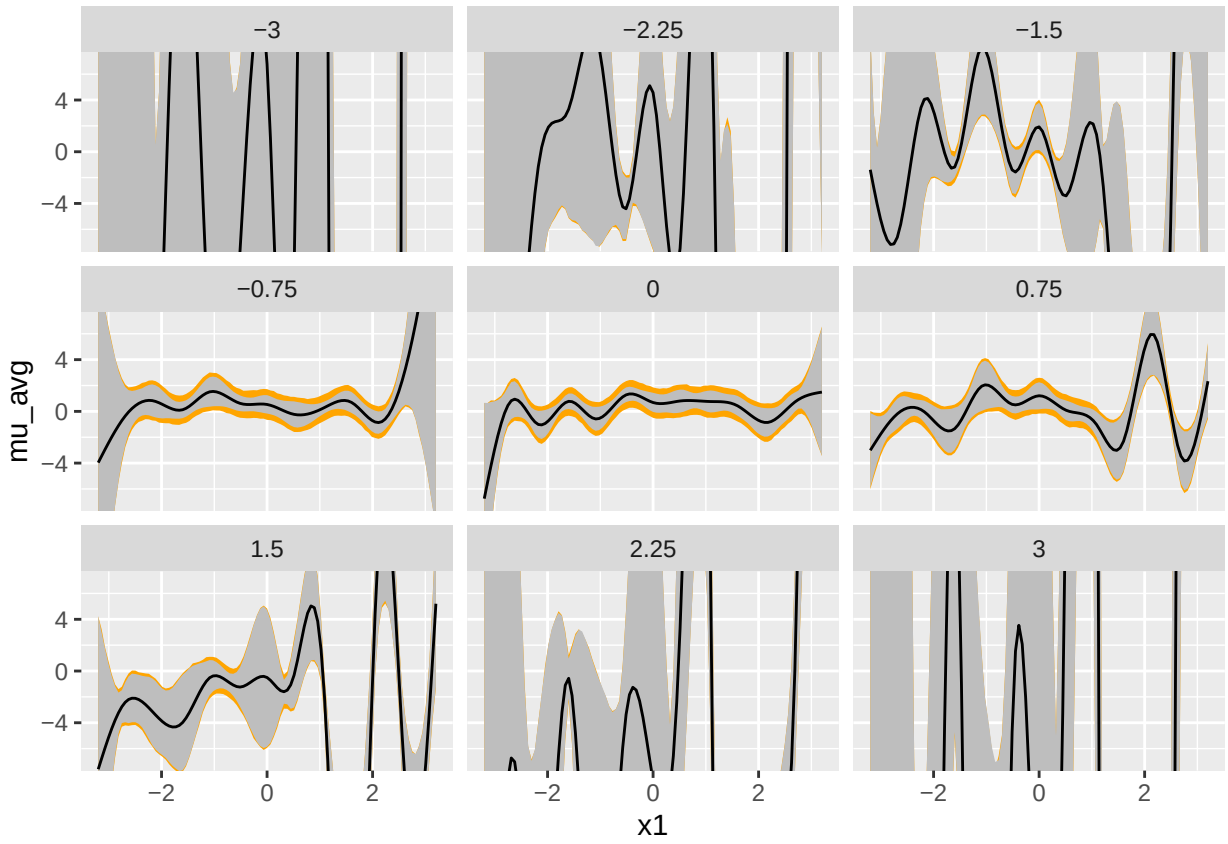
4f)

You will now visualize the posterior predictions from the model 6 Bayesian models associated with the weak, strong, and very strong priors. The `viz_grid` object is joined to the prediction dataframes assuming you have used the correct variable names!

Visualize the predicted means, confidence intervals, and prediction intervals in the style of those that you created in Problem 02. The confidence interval bounds are `mu_lwr` and `mu_upr` columns and the prediction interval bounds are the `y_lwr` and `y_upr` columns, respectively. The posterior predicted mean of the mean is `mu_avg`.

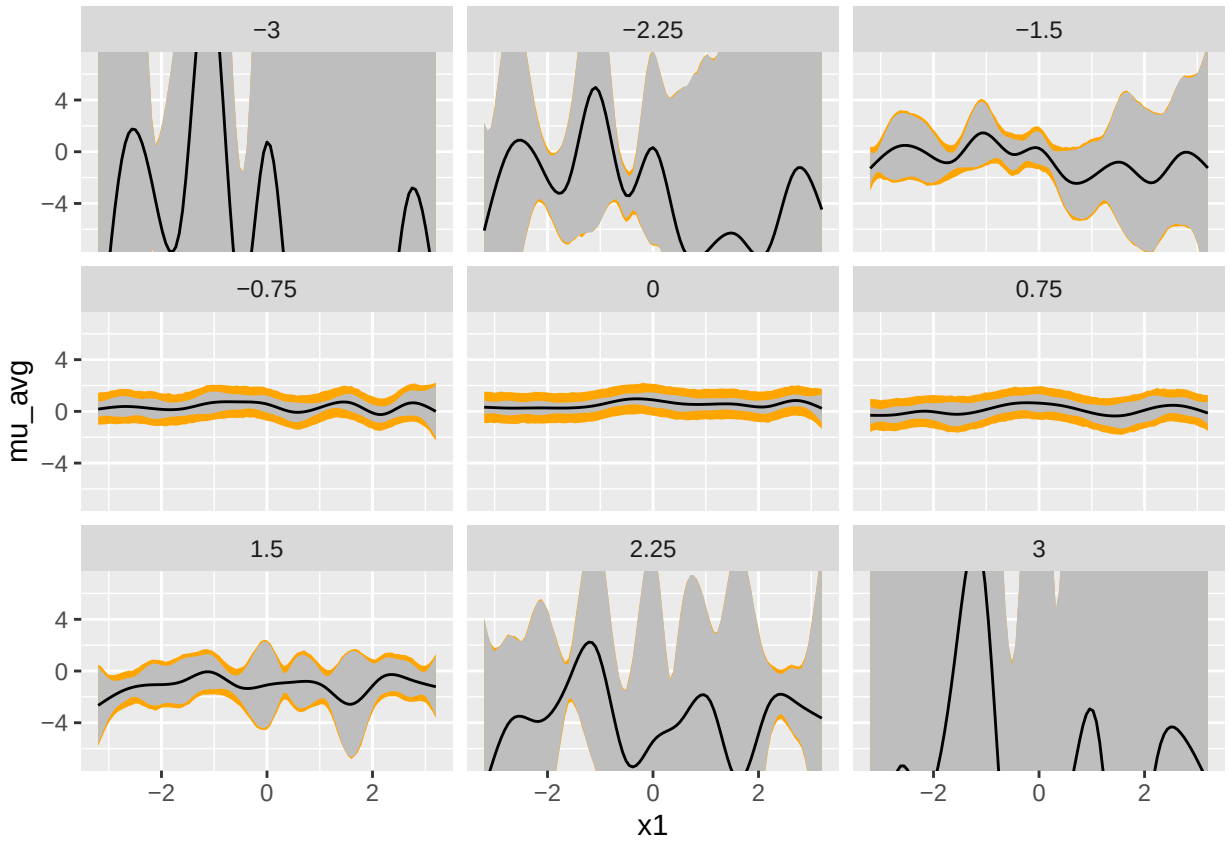
Pipe the result of the joined dataframe into `ggplot()` and make appropriate aesthetics and layers to visualize the predictions with the `x1` variable mapped to the `x` aesthetic and the `x2` variable used as a facet variable.

```
post_pred_summary_viz_06_weak %>%
  left_join(viz_grid %>% tibble::rowid_to_column("pred_id"),
            by = 'pred_id') %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=y_lwr, ymax=y_upr), fill="orange")+
  geom_ribbon(aes(ymin=mu_lwr, ymax=mu_upr), fill="grey")+
  geom_line(aes(y=mu_avg)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```

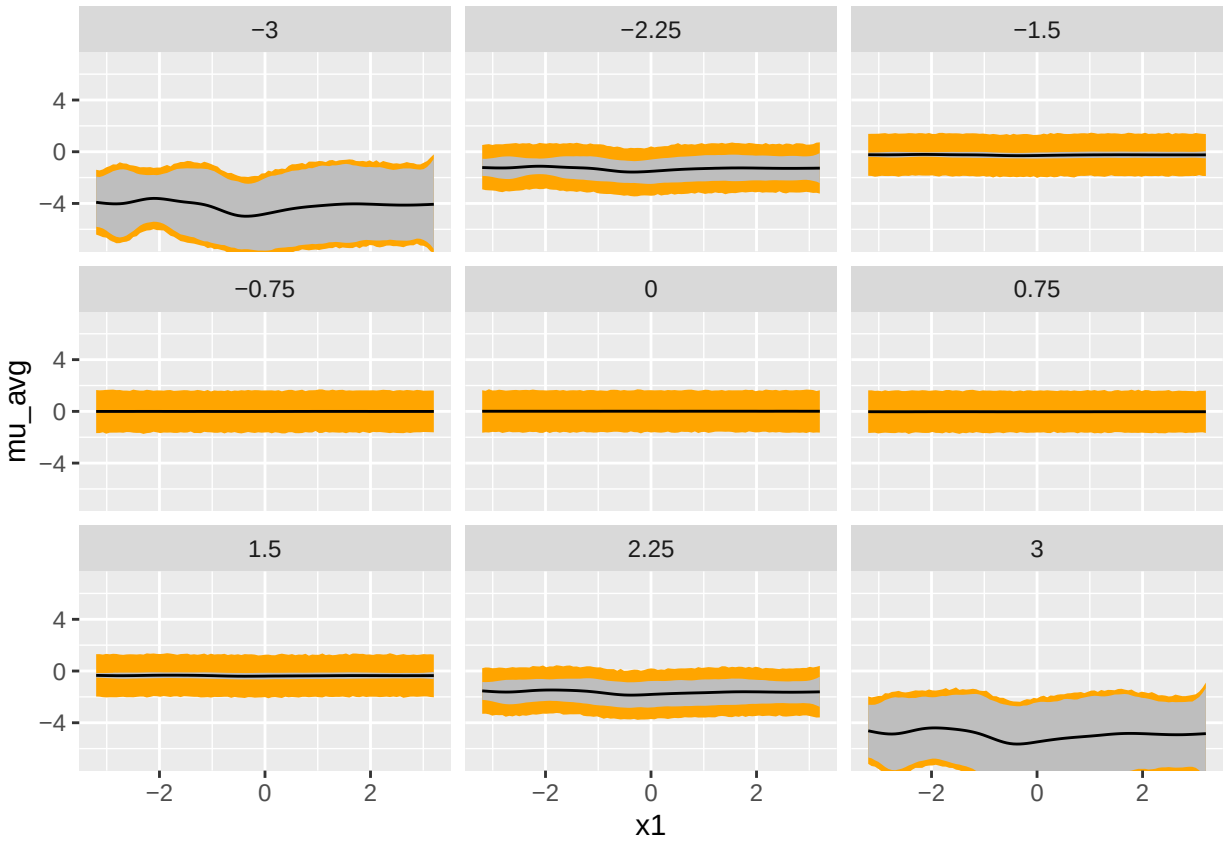


SOLUTION

```
post_pred_summary_viz_06_strong %>%
  left_join(viz_grid %>% tibble::rowid_to_column("pred_id"),
            by = 'pred_id') %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=y_lwr, ymax=y_upr), fill="orange")+
  geom_ribbon(aes(ymin=mu_lwr, ymax=mu_upr), fill="grey")+
  geom_line(aes(y=mu_avg)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```




```
post_pred_summary_viz_06_very_strong %>%
  left_join(viz_grid %>% tibble::rowid_to_column("pred_id"),
            by = 'pred_id') %>%
  ggplot(aes(x=x1))+
  geom_ribbon(aes(ymin=y_lwr, ymax=y_upr), fill="orange")+
  geom_ribbon(aes(ymin=mu_lwr, ymax=mu_upr), fill="grey")+
  geom_line(aes(y=mu_avg)) +
  facet_wrap(~x2) +
  coord_cartesian(ylim=c(-7,7))
```



4g)

Describe the behavior of the predictions as the prior standard deviation decreased. Are the posterior predictions consistent with the behavior of the posterior coefficients?

SOLUTION What do you think?

1) Based on the results in Problem 4d), in Model-3, the confidence interval and prediction interval do not change much as prior standard deviation changed from 50 to 1. However, as the standard deviation changed from 1 to 1/50, we observe that both intervals reduce and the portion of the confidence interval gets smaller.

2) Based on the results in Problem 4f), in Model-6, the confidence interval and prediction interval do reduce and mean trend becomes less wiggle as prior standard deviation changed from 50 to 1. Also, as the standard deviation changed from 1 to 1/50, we observe that both intervals reduce significantly and mean trend becomes more stable (almost constant).

3) As the prior standard deviation decreases, it implies a stronger belief that the coefficients are close to zero. Therefore, the regularization effect of the prior becomes stronger. This results in the model becoming less flexible and potentially more biased, but with reduced variance.

4) Also, as observed in Model-6, with a decreasing prior standard deviation, the model's predictions become smoother. Large fluctuations, wiggles, or overly complex patterns in the predictions get suppressed, leading to general trend-like predictions.

Problem 05

Now that you have worked with Bayesian models with the prior *regularizing* the coefficients, you will consider non-Bayesian regularization methods. You will work with the `glmnet` package in this problem which takes

care of all fitting and visualization for you.

The code chunk below loads in `glmnet` and so you must have `glmnet` installed before running this code chunk. **IMPORANT**: the `eval` flag is set to FALSE below. Once you download `glmnet` set `eval=TRUE`.

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

5a)

`glmnet` does not work with the formula interface. And so you must create the training design matrix. However, `glmnet` prefers the the intercept column of ones to **not** be included in the design matrix. To support that you must define new design matrices. These matrices will use the same formulation but you must remove the intercept column. This is easy to do with the formula interface and the `model.matrix()` function. Include `-1` in the formula and `model.matrix()` will not include the intercept. The code chunk below demonstrates removing the intercept column for a model with linear additive features.

```
model.matrix( y ~ x1 + x2 - 1, data = df) %>% head()
```

```
##           x1         x2
## 1 -0.3092328  0.3087799
## 2  0.6312721 -0.5479198
## 3 -0.6827669  2.1664494
## 4  0.2693056  1.2097037
## 5  0.3725202  0.7854860
## 6  1.2966439 -0.1877231
```

Create the design matrices for `glmnet` for the model 3 and model 6 formulations. Remove the intercept column for both and assign the results to `X03_glmnet` and `X06_glmnet`.

```
X03_glmnet <- model.matrix(y ~ (x1 + I(x1^2)) * (x2 + I(x2^2)) - 1, data = df)

X06_glmnet <- model.matrix (y ~ splines::ns(x1, 12) * (x2 + I(x2^2) + I(x2^3) + I(x2^4)) - 1, data = df)
```

SOLUTION

5b)

By default `glmnet` uses the **lasso** penalty. Fit a Lasso model by calling `glmnet()`. The first argument to `glmnet()` is the design matrix and the second argument is a regular vector for the response.

Train a Lasso model for the model 3 and model 6 formulations, assign the results to `lasso_03` and `lasso_06`, respectively.

```
yobs = df %>% pull(y) %>% as.matrix()

lasso_03 <- glmnet(X03_glmnet, yobs)
```

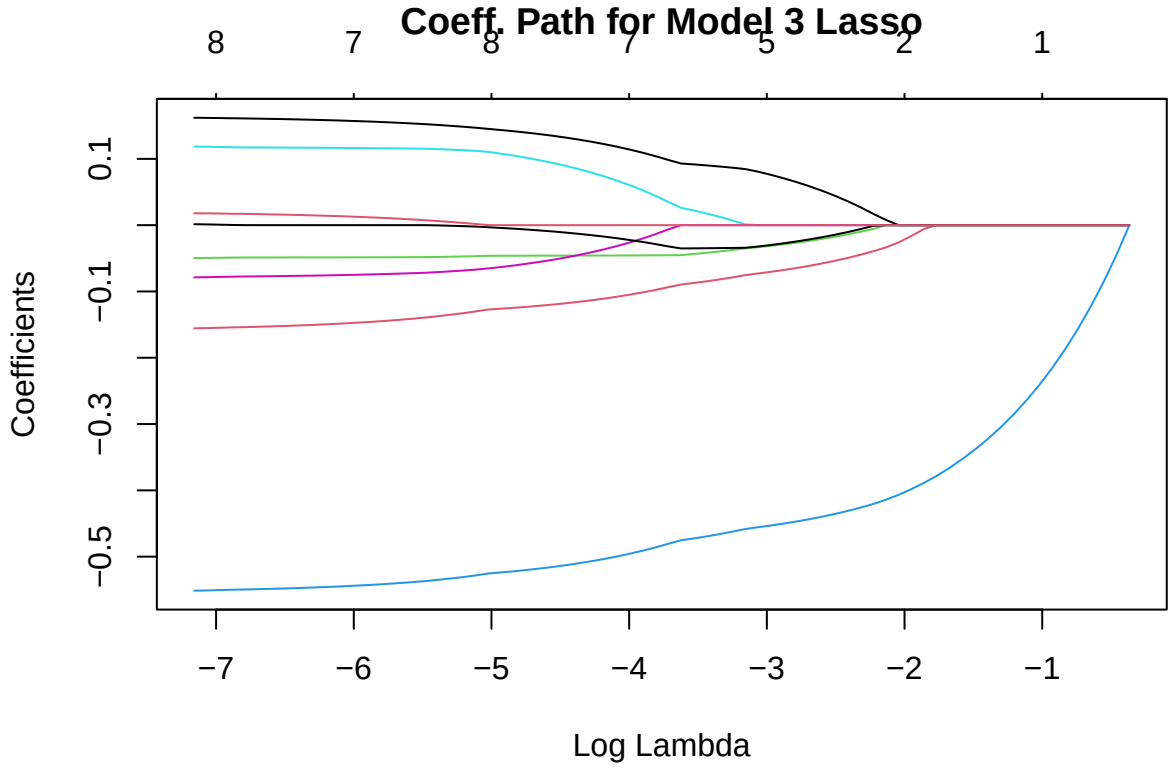
```
lasso_06 <- glmnet(X06_glmnet, yobs)
```

SOLUTION

5c)

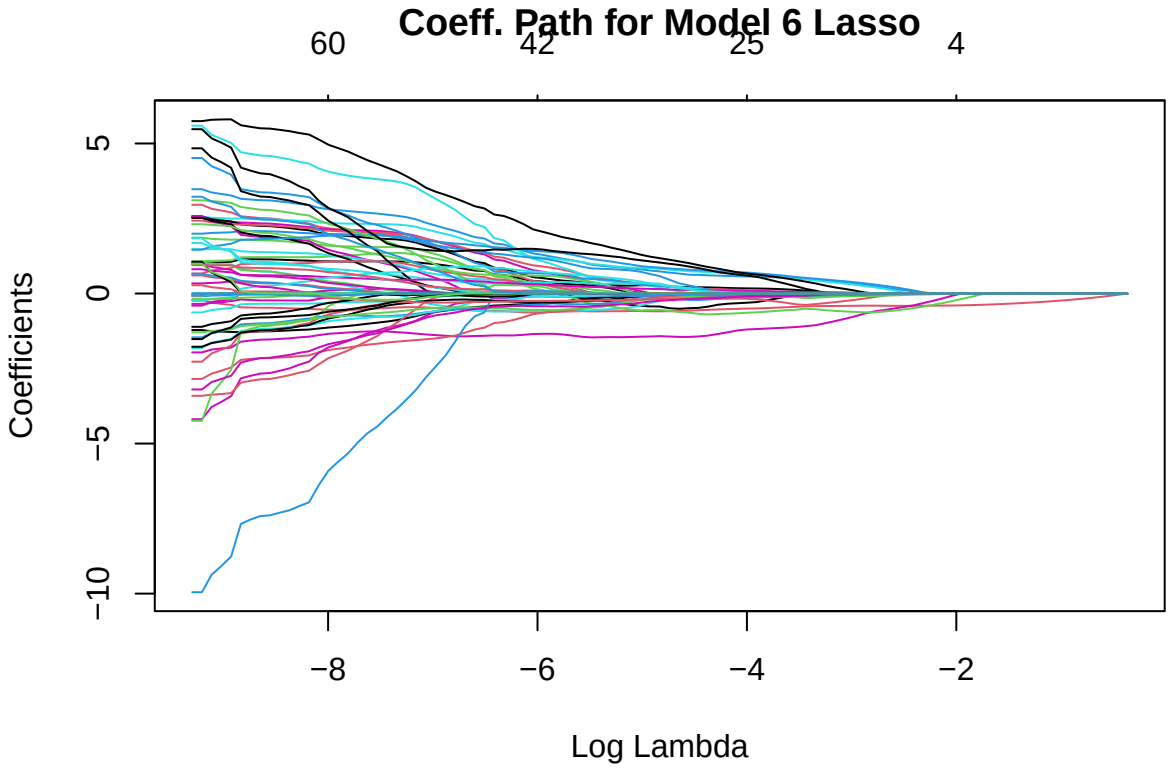
Plot the coefficient path for each Lasso model by calling the `plot()` function on the `glmnet` model object. Specify the `xvar` argument to be `'lambda'` in the `plot()` call.

```
plot(lasso_03, xvar='lambda', main="Coeff. Path for Model 3 Lasso")
```



SOLUTION

```
plot(lasso_06, xvar='lambda', main="Coeff. Path for Model 6 Lasso")
```



5d)

Now that you have visualized the coefficient path, it's time to identify the best `'lambda'` value to use! The `cv.glmnet()` function will by default use 10-fold cross-validation to tune `'lambda'`. The first argument to `cv.glmnet()` is the design matrix and the second argument is the regular vector for the response.

Tune the Lasso regularization strength with cross-validation using the `cv.glmnet()` function for each model formulation. Assign the model 3 result to `lasso_03_cv_tune` and assign the model 6 result to `lasso_06_cv_tune`. Also specify the `alpha` argument to be 1 to make sure the Lasso penalty is applied in the `cv.glmnet()` call.

HINT: The random seed was assigned for you in two separate code chunks below. This will help ensure you can reproduce the cross-validation results.

```
### the random seed is set for you
set.seed(812312)
### tune the model 3 formulation
lasso_03_cv_tune <- cv.glmnet(X03_glmnet, yobs, alpha=1)
lasso_03_cv_tune
```

SOLUTION

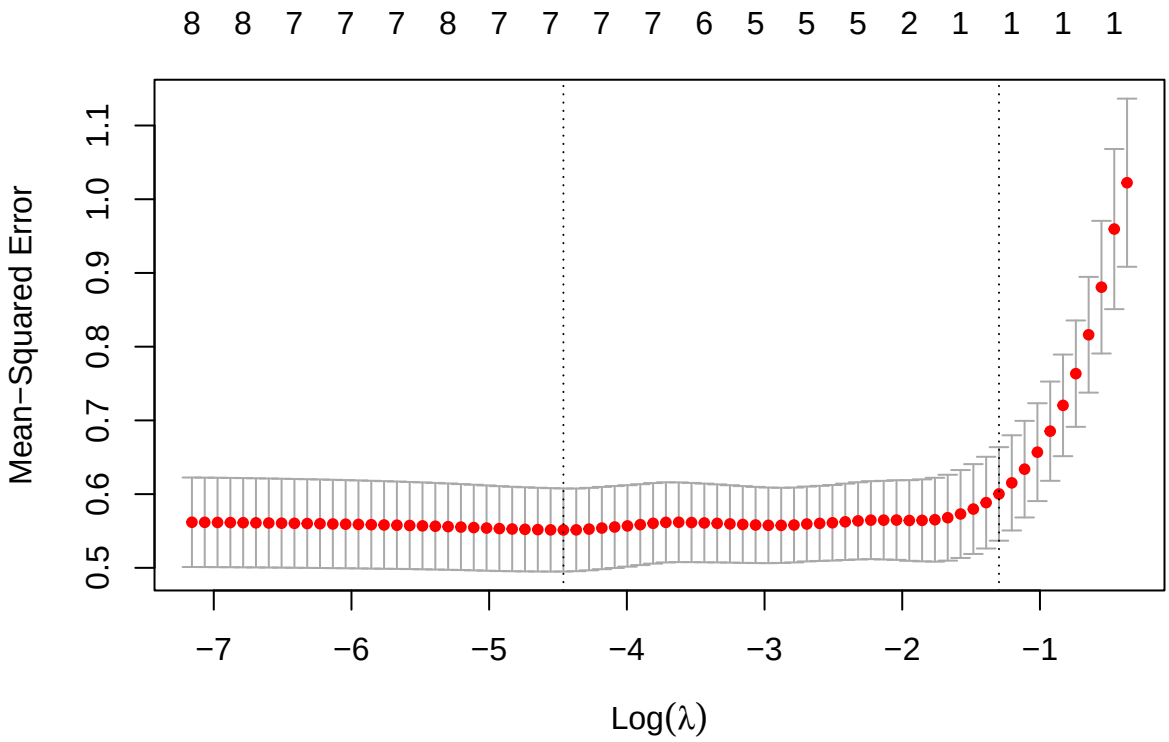
```
##
## Call:  cv.glmnet(x = X03_glmnet, y = yobs, alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Index Measure      SE Nonzero
## min 0.01156    45  0.5514 0.05634       7
## 1se 0.27322    11  0.6003 0.06345       1
```

```
### the random seed is set for you
set.seed(812312)
### tune the model 6 formulation
lasso_06_cv_tune <- cv.glmnet(X06_glmnet, yobs, alpha=1)
```

5e)

Plot the cross-validation results using the default plot method for each cross-validation result. How many coefficients are remaining after tuning?

```
plot(lasso_03_cv_tune)
```

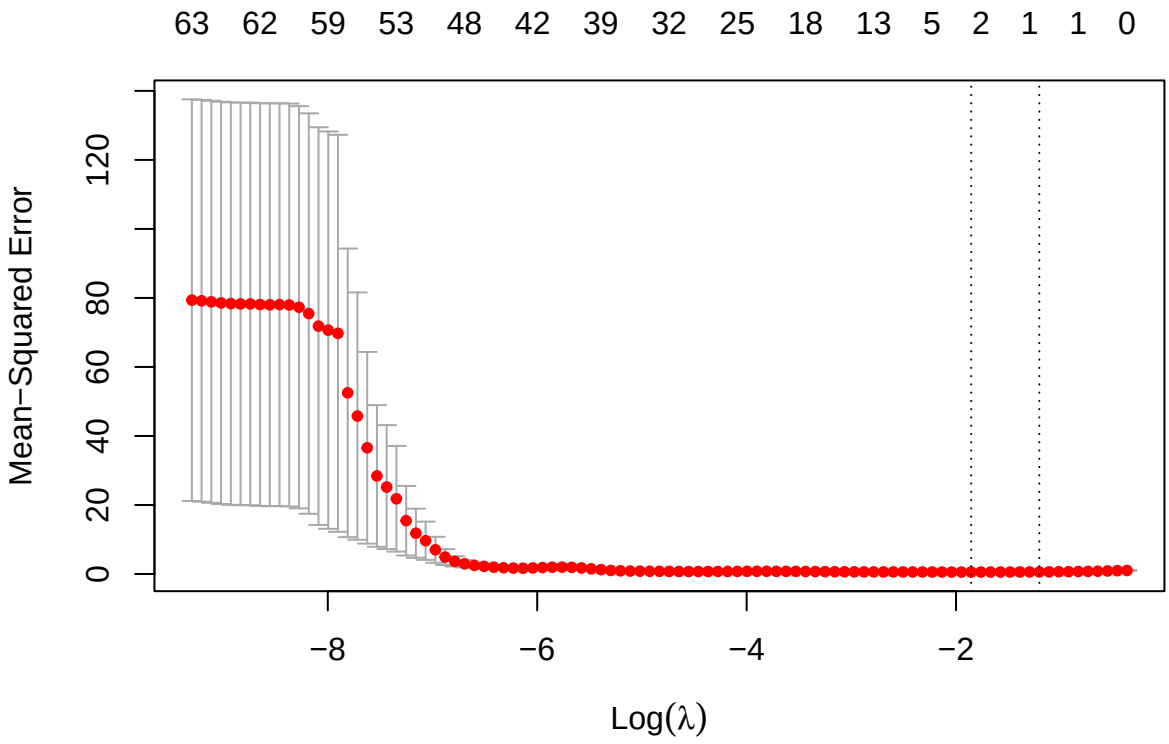


SOLUTION

```
best_lambda_03 <- lasso_03_cv_tune$lambda.min

coef_03 <- coef(lasso_03, s = best_lambda_03)

nonzero_coef_03 <- sum(coef_03 != 0)

cat("Number of non-zero coefficients for model 3 formulation:", nonzero_coef_03, "\n")
```

```
## Number of non-zero coefficients for model 3 formulation: 8
```

```
plot(lasso_06_cv_tune)
```




```
best_lambda_06 <- lasso_06_cv_tune$lambda.min

coef_06 <- coef(lasso_06, s = best_lambda_06)

nonzero_coef_06 <- sum(coef_06 != 0)

cat("Number of non-zero coefficients for model 6 formulation:", nonzero_coef_06, "\n")
```

```
## Number of non-zero coefficients for model 6 formulation: 3
```

5f)

Which features have NOT been "turned off" by the Lasso penalty? Use the `coef()` function to display the lasso model cross-validation results to show the tuned penalized regression coefficients for each model.
Are the final tuned models different from each other?

SOLUTION Yes, the final tuned models are different from each other. Here are the lasso model cross-validation results that show the tuned penalized regression coefficients for each model:

```
coef_03_tuned <- coef(lasso_03_cv_tune, s = "lambda.min")
print("Coefficients for Model 3 Formulation:")
```

```
## [1] "Coefficients for Model 3 Formulation:"
```

```
print(coef_03_tuned[coef_03_tuned[, 1] != 0, , drop = FALSE])
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                       s1
## (Intercept)  0.61129800
## x1           0.13209891
## I(x1^2)     -0.11778618
```

```
## x2          -0.04595605
## I(x2^2)     -0.51289193
## x1:x2        0.08947501
## x1:I(x2^2)  -0.04880905
## I(x1^2):x2  -0.01119924
```

```
coef_06_tuned <- coef(lasso_06_cv_tune, s = "lambda.min")
print("Coefficients for Model 6 Formulation:")
```

```
## [1] "Coefficients for Model 6 Formulation:"
```

```
print(coef_06_tuned[coef_06_tuned[, 1] != 0, , drop = FALSE])
```

```
## 3 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)                 0.3862601
## I(x2^2)                    -0.3865314
## splines::ns(x1, 12)11:x2 -0.1091537
```