TP 1 - Herramientas de Modelado Estadístico

Jésica Charaf e Ignacio Spiousas

30 de junio de 2024

Regresión ordinal

Vamos a usar un conjunto de datos correspondiente a una encuesta con escala de tipo Likert (es decir, se pide al encuestado marcar un entero entre 1 y 5, donde 1 = Totalmente en desacuerdo y 5 = Totalmente de acuerdo). La encuesta consiste de 44 preguntas muy variadas, como ”Disfruto de bailar” o ”Creo que un desastre climático podría llegar a ser divertido”. Para los individuos encuestados, se tienen también otras variables extra-encuesta que pueden ser de interés, como por ejemplo edad, género, religión, etc. Los datos están `encuesta.csv` y el archivo `codebook.txt` contiene una descripción de las preguntas y las variables extra-encuesta.

Preliminares

Lo primero que vamos a hacer es ver los datos. Para esto vamos a echar mano a una función muy útil del paquete *{modelsummary}* que nos permite ver rápidamente varias características de los mismos[1].

```
# Cargamos los datos
cuestionario_tbl <- read_tsv(here("modelado_estadístico/TP1/data/data.csv"))

# Creamos una versión long del dataset
cuestionario_tbl_pivoted <- cuestionario_tbl |>
  pivot_longer(
    cols = starts_with("Q"),
    names_to = "Pregunta",
    values_to = "Respuesta",
  )

# Usamos la función datasummary_skim de modelsummary para ver los datos
```

[1] No vamos a mirar las respuestas de cada pregunta por separado porque esto genera una tabla muy larga y que no es adecuada para el formato de entrega del informe. En su lugar observaremos una versión *long* de los datos, es decir, las respuestas de **todas** las preguntas en busca de irregularidades.

```r
datasummary_skim(cuestionario_tbl_pivoted,
                 type = "numeric",
                 title = "Descripción de los datos.") |>
  theme_tt("placement", latex_float = "H")
```

	Unique	Missing Pct.	Mean	SD	Min	Median	Max	Histogram
introelapse	5702	0	898.5	48 469.6	0.0	9.0	15 456 301.0	
testelapse	5611	0	957.3	48 968.2	89.0	247.0	15 122 938.0	
IPC	86	0	3.1	10.6	1.0	1.0	162.0	
source	6	0	0.9	1.2	0.0	0.0	5.0	
engnat	3	0	1.2	0.4	0.0	1.0	2.0	
age	174	0	7111.2	3 808 882.0	13.0	20.0	2 147 483 647.0	
education	5	0	2.2	0.8	0.0	2.0	4.0	
gender	4	0	1.6	0.6	0.0	2.0	3.0	
orientation	6	0	1.9	1.2	0.0	1.0	5.0	
race	8	0	5.0	1.9	0.0	6.0	7.0	
religion	8	0	2.3	2.1	0.0	1.0	7.0	
hand	4	0	1.2	0.5	0.0	1.0	3.0	
Respuesta	6	0	3.0	1.6	0.0	3.0	5.0	

Se puede ver (en *Missing Pct.*) que no hay datos faltantes en ninguna pregunta ni en la edad, pero sí tenemos dos problemas: **1)** Que hay respuestas con valor 0 en las preguntas *Q1* a *Q44* (las respuestas toman 6 valores distintos y el mínimo es 0) y, **2)** hay edades imposiblemente altas. Entonces, inicialmente vamos a filtrar los datos para quedarnos con las edades menores a 90 años. También filtraremos todas las respuestas iguales a cero. Esto es potencialmente problemático ya que, por ejemplo, al ajustar un modelo con **Q4**, sólo nos importaría quitar las respuestas cero de esa pregunta. El motivo por el que lo hacemos de esta manera es que el *split* entre *train* y *test* lo hacemos una vez al comienzo del análisis y creemos que es más correcto hacerlo ya con el *dataset* final. Por otro lado, hacer este filtrado elimina sólo el 8.22% de los datos.

```r
# Nos quedamos con edades menores a 90 y respuestas mayores a 0
cuestionario_tbl <- cuestionario_tbl |>
  # Filtramos todas las columnas que empiezan con Q
  filter(if_all(starts_with('Q'), function(x) x > 0)) |>
  filter(age <= 90)
```

```r
# Volvemos a generar la versiôn long de los datos
cuestionario_tbl_pivoted <- cuestionario_tbl |>
  pivot_longer(
    cols = starts_with("Q"),
    names_to = "Pregunta",
    values_to = "Respuesta",
  )
```

Una variable que nos interesa en detalle es la edad, ya que la usaremos como predictor en varios de los modelos más adelante. Vamos a mirar el histograma y la densidad de la variable edad para ver cómo se distribuye (Figura 1).

```r
cuestionario_tbl |>
  ggplot(aes(x = age)) +
  geom_histogram(aes(y = after_stat(density)), fill = "steelblue", alpha = .4) +
  geom_density(color = "steelblue", alpha = 0, bw = 1, linewidth = 1) +
  labs(x = "Edad", y = "Densidad") +
  theme_bw()
```

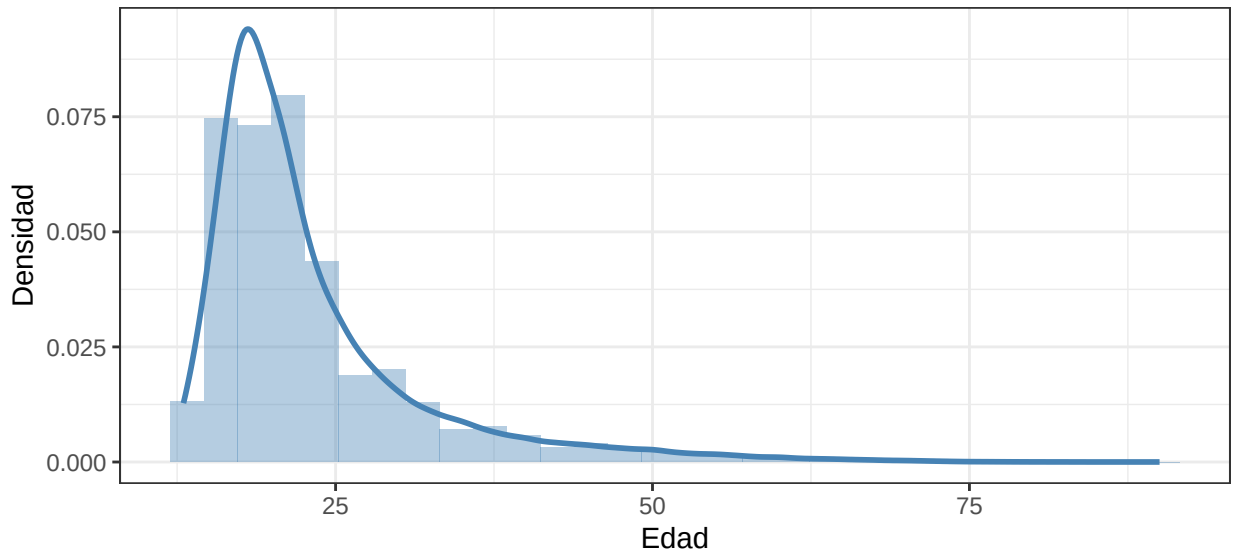

Figure 1: Histograma y densidad de la variable edad.

Lo que se observa es que la gran mayoría de las observaciones corresponden a sujetos de menos de 35 años. Esto, teniendo en cuenta que queremos usar a la edad como predictor, puede ser problemático (spoiler: **ES** problemático).

1. Dividir al conjunto de datos en data de entrenamiento y testeo.

Para dividir los datos en un conjunto de entrenamiento y otro de testeo vamos a utilizar la función `initial_split` del paquete *{tidymodels}*. Esta función nos permite hacer una división estratificada por edad (para evitar que haya una representación desigual de edades en cada conjunto). Vamos a dejar dos tercios de los datos en el conjunto de entrenamiento y un tercio en el de testeo.

```
# Vamos a usar la función initial_split de tidymodels
cuestionario_splits <- initial_split(cuestionario_tbl, prop = 2/3, strata = age)
train_data <- training(cuestionario_splits)
test_data <- testing(cuestionario_splits)
```

2. Leer las preguntas y elegir alguna que parezca interesante. Llamaremos Q a esta pregunta.

En este punto tomamos una vía alternativa y lo que consideramos para elegir la pregunta de interés fue su relación con la edad, ya que esta variable será la principal predictora de los modelos del resto del trabajo. Para analizar esto vamos a visualizar los promedios de respuesta por edad para todas las preguntas (Figura 2).

A partir de esta figura decidimos seleccionar la pregunta **Q6** ("Me da vergüenza que la gente lea cosas que he escrito") que es una de las que muestra un comportamiento que varía según la edad.

```
cuestionario_tbl_pivoted |>
    mutate(Pregunta = as_factor(Pregunta)) |> # Para que me haga el facet_wrap
  #                                           ordenado
  ggplot(aes(x = age,
             y = Respuesta)) +
  stat_summary(color = "steelblue", alpha = .5, size = .1, fun.data = "mean_se") +
  facet_wrap(~ Pregunta, labeller = label_both) +
  labs(x = "Edad") +
  theme_bw() +
  theme(strip.background = element_rect(colour="white", fill="white"))
```

Ahora que ya seleccionamos la pregunta podemos ver cómo es la distribución de edades para cada valor de respuesta. Esto lo haremos utilizando tanto un boxplot como una estimación no paramétrica de la densidad (Figura 3). Podemos observar que los valores centrales de las edades disminuyen ligeramente a medida que aumenta el valor de la respuesta.

```
cuestionario_tbl |>
  dplyr::select(c("age", "Q6")) |>
  mutate(Q6 = as_factor(Q6)) |> # Para que me haga el facet_wrap ordenado
```

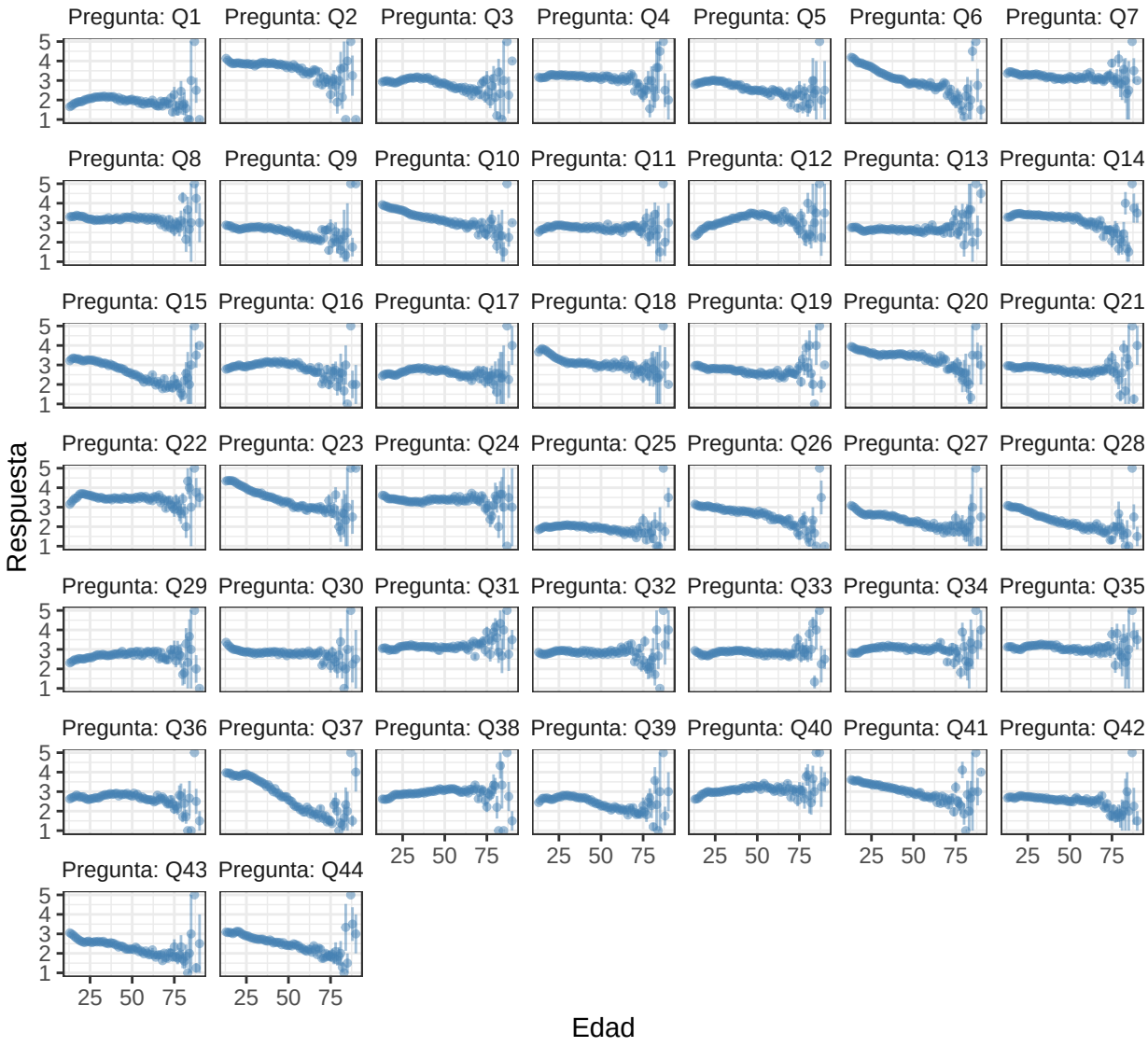

Figure 2: Promedio de las respuestas en función de la edad para cada pregunta.

```r
ggplot(aes(x = Q6,
           y = age,
           group = Q6)) +
ggdist::stat_halfeye(
  fill = "steelblue",
  alpha = .7,
  adjust = 3,
  width = .7,
  .width = 0,
  justification = -.3,
  point_colour = NA) +
geom_boxplot(color = "steelblue",
             fill = "steelblue",
             alpha = .4,
             width = .25,
             outlier.shape = NA) +
labs(x = "Respuesta a Q6", y = "Edad") +
theme_bw()
```

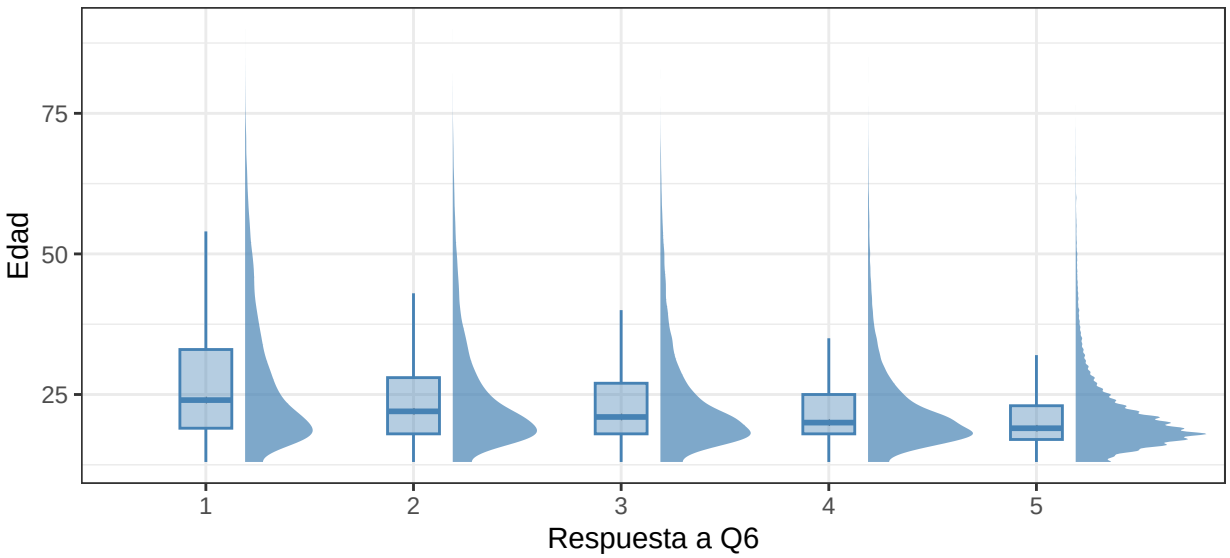

Figure 3: Distribución de las edades según el tipo de respuesta.

3. Supongamos que queremos modelar la respuesta Q en función de la edad y el género. ¿Cuál sería el problema teórico de usar una regresión lineal para esto? ¿Cuál sería el problema de usar una regresión multinomial en este problema?

Si usamos regresión lineal para modelar la respuesta Q en función de la edad y el género, para empezar tendríamos un problema de rangos ya que las predicciones van a variar en todos los reales y nuestra variable de respuesta toma números enteros entre 1 y 5. Por otra parte, las diferencias o saltos entre una clase y otra no necesariamente son equivalentes y en el modelo de regresión lineal no podemos regular este aspecto ya que se considera el valor numérico y la distancia que hay entre las clases es siempre la misma.

En el caso de la regresión multinomial, si bien nos permite abordar el problema de diferentes clases, la principal desventaja es que nuestras clases están ordenadas y el modelo de regresión multinomial no tiene en cuenta ninguna relación de orden entre las categorías, son indistintas entre sí, con lo cual se estaría perdiendo esta información.

4. Leer (en Wikipedia, por ejemplo) acerca de Regresión Ordinal. Explicar, en tus propias palabras, en qué consiste este modelo.

El modelo de Regresión Ordinal se utiliza cuando la variable de respuesta es ordinal, es decir, donde los posibles resultados son categorías que están ordenadas en cierta escala.

Si consideramos a Y como nuestra variable de respuesta que toma valores en una escala de $1, ..., K$, el enfoque se basa en modelar la probabilidad de que Y sea menor o igual que k, $P(Y \leq k)$, donde k representa a cada una de las diferentes clases. Luego, en función de estas probabilidades se predice la clase.

Para esto se consideran $K-1$ puntos de corte $\alpha_1 < \alpha_2 < ... < \alpha_{K-1}$ que son parámetros a ajustar en el modelo. De esta manera, a partir de las covariables X, podemos pensar que la $P(Y \leq k)$ se modela como la probabilidad de que $X^t\beta \leq \alpha_k$, donde β es un vector de parámetros a ajustar también en el modelo de regresión. Esto quiere decir que, dependiendo de dónde se ubica $X^t\beta$ respecto de los puntos de corte, va a ser cómo se clasifique la variable de respuesta en la categoría correspondiente.

En forma general, el modelo se puede escribir de la siguiente manera:

$$P(Y \leq k|X) = F(\alpha_k - X^t\beta)$$

donde la F dependerá de la distribución que se asuma, pero típicamente se utiliza la función logística o la función de distribución de una normal estándar.

5. Usando el paquete `MASS` y la función `polr`, aplicar el modelo de regresión ordinal para predecir Q en función de la edad.

Creamos un dataset que contenga sólo las columnas `Q6` y `age` (por comodidad) y ajustamos el modelo tomando como función F la función de distribución de una normal estándar (probit).

```
train_data_fit <- train_data |>
  dplyr::select(c("age", "Q6")) |>
  mutate(Q6 = as.factor(Q6))
ajuste_ord_Q6 <- polr(Q6 ~ age, data = train_data_fit, method = "probit")
```

Vamos a analizar cómo predice el modelo en una grilla de edades desde el mínimo al máximo de los datos con paso 1.

```
pred_ord_Q6 <- tibble(age = min(cuestionario_tbl$age):max(cuestionario_tbl$age),
                Prediccion = predict(ajuste_ord_Q6,
  newdata = tibble(age = min(cuestionario_tbl$age):max(cuestionario_tbl$age))))

pred_ord_Q6 |> group_by(Prediccion) |>
  summarise(Cantidad = n()) |>
  tt(width = 0.4, caption = "Cantidad de predicciones por clase.") |>
  theme_tt("placement", latex_float = "H")
```

Table 1: Cantidad de predicciones por clase.

Prediccion	Cantidad
1	47
4	6
5	25

Realizamos una tabla en la que se muestra la cantidad de predicciones que caen en cada clase y podemos ver que el modelo predice sólo las clases 1, 4 y 5.

Algo más que podemos observar son las probabilidades de pertenecer a cada clase en función de la edad. Para eso vamos a predecir usando `type = 'p'`.

```
pred_P_ord_Q6 <-
  tibble(age = min(cuestionario_tbl$age):max(cuestionario_tbl$age)) |>
  bind_cols(as_tibble(predict(ajuste_ord_Q6,
    newdata = tibble(age = min(cuestionario_tbl$age):max(cuestionario_tbl$age)),
           type = "p"))) |>
```

```
  pivot_longer(cols = -age, values_to = "probabilidad", names_to = "clase")
```

```
pred_P_ord_Q6 |>
  ggplot(aes(x = age,
             y = probabilidad)) +
  geom_line(data = pred_ord_Q6,
            aes(x = age, y = .6, color = Prediccion),
            linewidth = 2, show.legend = FALSE) +
  geom_text(data = tibble(x = c(26, 42, 67.5), y = .63,
                    label = paste0("Clase = ", c(5, 4, 1))),
                    aes(x = x, y = y, label = label)) +
  geom_line(aes(color = clase)) +
  scale_color_brewer(palette = "Dark2") +
  labs(x = "Edad", y = "Probabilidad", color = "Clase") +
  theme_bw() +
  theme(legend.position = "top")
```

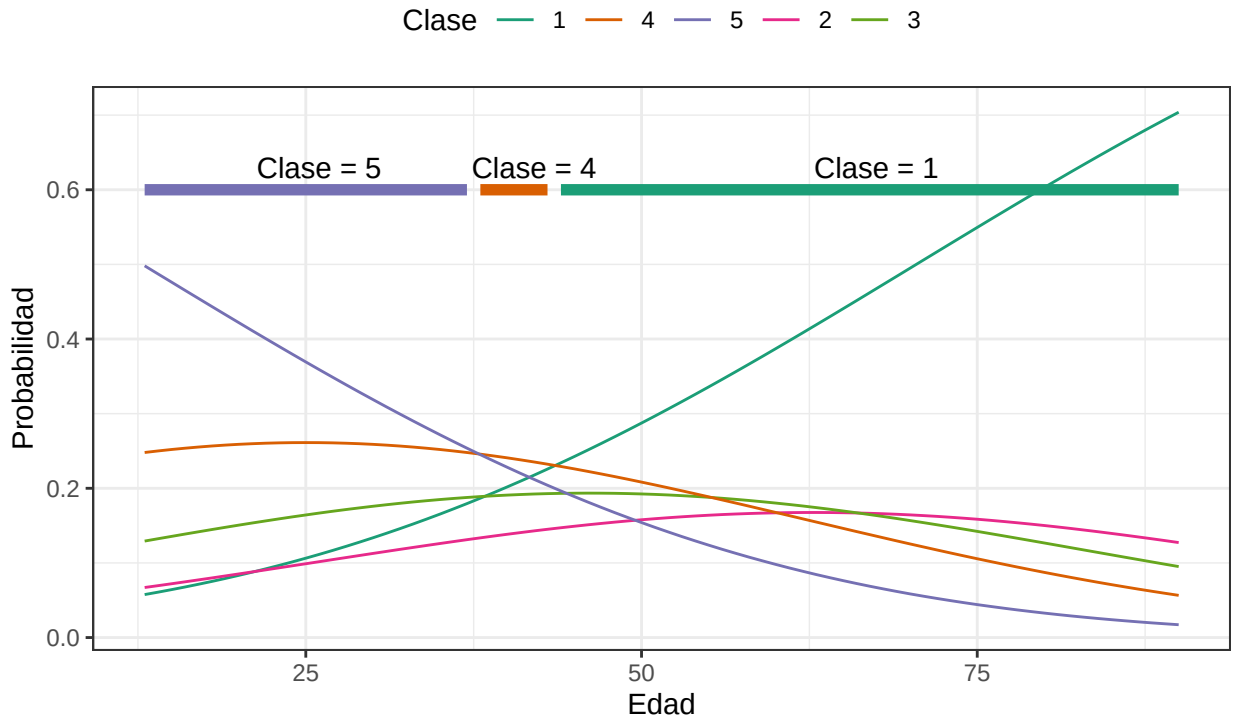

Figure 4: Probabilidad de cada clase de respuesta en función de la edad.

En la Figura 4 vemos en distintos colores las curvas obtenidas al estimar la probabilidad de cada tipo de respuesta en función de las edades. Allí se puede observar cómo resulta la predicción de la

clase en las diferentes regiones de edades, según cuál es la probabilidad que "gana". Por ejemplo, en el primer tramo (aproximadamente hasta los 37 años) la probabilidad estimada de que un sujeto responda "5" supera a las probabilidades del resto de las respuestas, con lo cual para todas esas edades la predicción corresponderá a la clase 5.

6. Estimar la probabilidad de que a una persona de 25 años esté al menos de acuerdo con la frase "me gustan las armas" (pregunta 9).

En este caso nos vamos a quedar sólo con las columnas `Q9` y `age` y ajustar el modelo.

```
train_data_fit <- train_data |>
  dplyr::select(c("age", "Q9")) |>
  mutate(Q9 = as.factor(Q9))
ajuste_ord_Q9 <- polr(Q9 ~ age, data = train_data_fit, method = "probit")

data_25 <- tibble(age=25)
pred_25 <- predict(ajuste_ord_Q9, newdata = data_25, type = "p")
proba_acuerdo <- pred_25[4]+pred_25[5]
```

En lugar de predecir la respuesta para una persona de 25 años, predecimos la probabilidad de que su respuesta sea cada una de las opciones posibles. Entonces, para calcular la probabilidad de que esté al menos de acuerdo debemos sumar las probabilidades de que su respuesta sea "4" o "5" (es decir, las respuestas mayores a "neutral"). De este modo, la probabilidad de que una persona de 25 años esté al menos de acuerdo con "Me gustan las armas" es de 0.34.

7. Para la pregunta Q, definamos la siguiente función de pérdida:

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

donde y_i es la respuesta del individuo i a la pregunta Q y $\hat{y}_i$ es la correspondiente predicción. Notar que tanto y_i como $\hat{y}_i$ son números enteros entre 1 y 5. Implementar esta función de pérdida en R.

Llamaremos a la función de pérdida `loss_abs`, debido a que se trata del valor absoluto de la diferencia entre la estimación del modelo y el valor real.

```
loss_abs <- function(y_hat, y) {
  mean(abs(y - y_hat))
}
```

8. Implementar un modelo lineal que prediga la respuesta a la pregunta Q en función de la edad. Este modelo tendrá predicciones $\hat{y}_i$ que pertenecen a toda la recta real. Para hacerlo comparable con el modelo de regresión ordinal, tomar como predicción final al número entero entre 1 y 5 más cercano a y_i.

Lo primero que vamos a hacer es ajustar un modelo lineal.

```
train_data_fit <- train_data |>
  dplyr::select(c("age", "Q6"))
ajuste_lin_Q6 <- lm(Q6 ~ age, data = train_data_fit)
```

A continuación, vamos a crear una función que nos permita redondear al entero más cercano pero con dos parámetros extras (`xmin` y `xmax`) que nos permiten limitar el rango de enteros que nos puede devolver la función.

```
round_Q <- function(x, xmin, xmax) {
  x <- round(x)
  for (i in 1:length(x)) {
    if (x[i] < xmin) {
      x[i] <- xmin
    } else if (x[i] > xmax) {
      x[i] <- xmax
    }
  }
  x
}
```

Ahora vamos a predecir con el modelo lineal la respuesta en función de la edad. Para eso consideramos la misma grilla de edades del punto 5.

```
pred_lm_Q6 <- tibble(age = min(cuestionario_tbl$age):max(cuestionario_tbl$age)) |>
  # Encuentro el entero más cercano entre 1 y 5
  rowwise() |>
  mutate(lm = round_Q(predict(ajuste_lin_Q6, newdata = tibble(age = age)), 1, 5),
         polr = as.numeric(predict(ajuste_ord_Q6, newdata = tibble(age = age))))
```

```
pred_lm_Q6 |>
  pivot_longer(cols = -age, names_to = "Metodo", values_to = "Clase") |>
  ggplot(aes(x = age, y = Clase, color = Metodo)) +
  geom_line(linewidth = 2) +
  scale_color_brewer(palette = "Dark2") +
  labs(x = "Edad", y = "Clase predicha", color = "Método") +
```

```
  theme_bw() +
  theme(legend.position = "top")
```

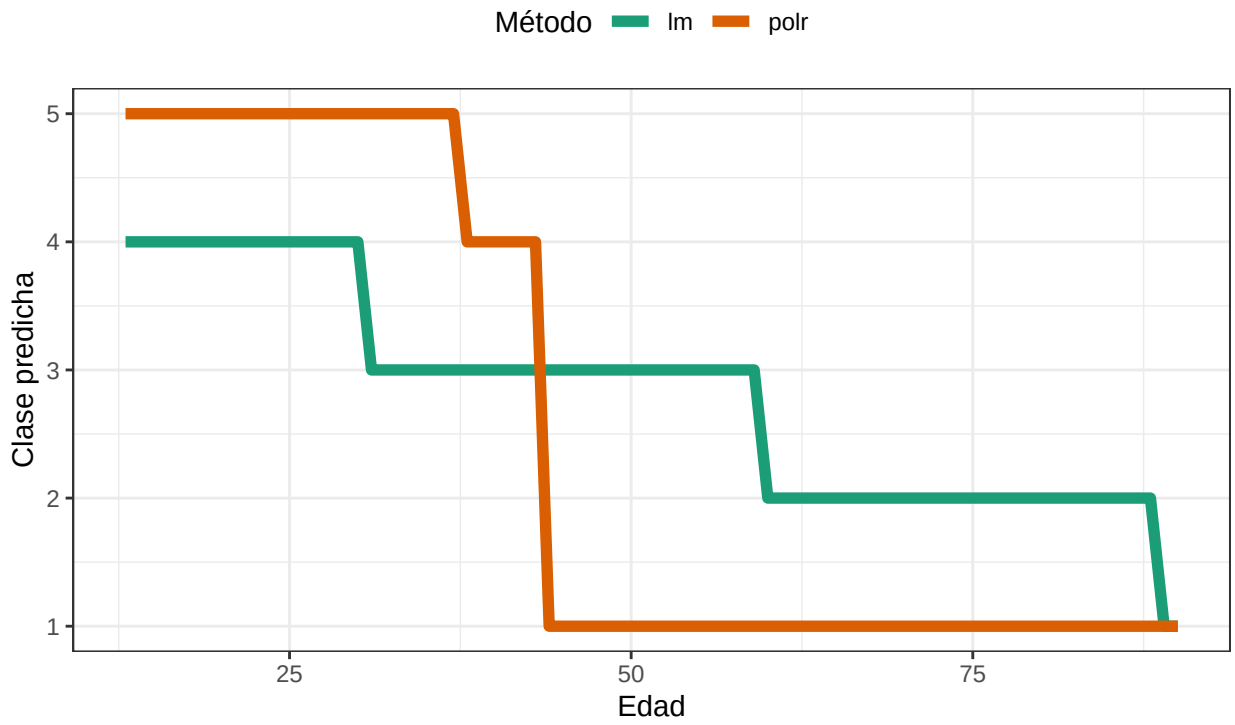

Figure 5: Comparación de las predicciones de las respuestas con el modelo regresión lineal y el modelo de regresión ordinal.

En la Figura 5 se muestran las predicciones de las respuestas en función de la edad utilizando el modelo de regresión lineal en comparación con los resultados obtenidos para el modelo de regresión ordinal. Se puede observar que las respuestas predichas en general difieren, pero lo que tienen en común es que en las edades más chicas comienzan con respuestas altas y van decreciendo a medida que aumenta la edad.

9. Comparar el valor de la pérdida L para el modelo de regresión ordinal y el modelo de regresión lineal (modificado) del item anterior, aplicando ambos. Decidir cuál de los dos es preferible (recordar entrenar los modelos en el conjunto de entrenamiento y evaluarlo en la data de testeo).

A partir de los dos modelos ajustados con los datos de entrenamiento, predecimos las respuestas en nuestro conjunto de testeo y evaluamos la función de pérdida implementada en el punto 7.

```
# Predecimos para ambos modelos con el conjunto de test
pred_test_data <- test_data |>
```

```
  dplyr::select(c("age", "Q6")) |>
  mutate(lm = round_Q(predict(ajuste_lin_Q6, newdata = tibble(age = age)), 1, 5),
         polr = as.numeric(predict(ajuste_ord_Q6, newdata = tibble(age = age))))
```

```
# Calculamos la loss para cada predicción
pred_test_data |>
  pivot_longer(cols = c("lm", "polr"),
               names_to = "Metodo",
               values_to = "Clase") |>
  group_by(Metodo) |>
  summarise(loss = round(loss_abs(Clase, Q6), 3)) |>
  tt(width = 0.4, caption = "Pérdida para cada tipo de modelo.") |>
  theme_tt("placement", latex_float = "H")
```

Table 2: Pérdida para cada tipo de modelo.

Metodo	loss
lm	1.035
polr	1.234

Basándonos en el valor de pérdida L, resulta preferible el modelo de regresión lineal que es el que tiene menor valor.

10. Vamos a intentar predecir la respuesta a la pregunta Q únicamente con la edad. Sea β el coeficiente correspondiente a la edad. Vamos a plantear un modelo bayesiano usando la funciónn `stan_polr` del paquete `rstanarm`, en donde le vamos a imponer una distribución a priori a β. Mostrar cómo dicha elección de la priori afecta la probabilidad a posteriori de dicho parámetro. Es decir, mostrar un conjunto de distribuciones a priori que induzcan probablilidades a posteriori muy distintas. (Comparar estas posterioris en un gráfico sería lo ideal).

A continuación vamos a ajustar unos modelos similares a los que ajustamos anteriormente (para **Q6**) pero utilizando un modelo bayesiano con la función `stan_polr`. La distribución a priori que nos permite agregar el modelo es una distribución Beta sobre el R^2. Sólo se puede agregar un parámetro de esta distribución (la media o la mediana). De esta forma, de algún modo le estamos diciendo al modelo qué porcentaje de varianza esperamos que explique.

Para estudiar la sensibilidad de la estimación del parámetro β a este prior vamos a ajustar[2] dos

[2]El ajuste de los modelos aparece comentado en el RMD ya que su tiempo de procesamiento es considerablemente

modelos con medias de R^2 extremas (0.00001 y 0.999) y ver su efecto en la estimación puntual y la distribución a posteriori de β.

```
# Filtramos los datos de training
train_data_fit <- train_data |>
  dplyr::select(c("age", "Q6")) |>
  mutate(Q6 = as.factor(Q6))

# Ajustamos un modelos bayesiano con prior R2(.00001, "mean")
# ajuste_ord_bayesiano_Q6_Rchico <-
#   stan_polr(Q6 ~ age, data = train_data_fit, method = "probit",
#             prior = R2(.00001, "mean"), seed = 12345,
#             algorithm = "fullrank")
# saveRDS(ajuste_ord_bayesiano_Q6_Rchico,
#         file = here("modelado_estadístico/TP1/modelos/ajuste_ord_bayesiano_Q6_Rchico.rds"))

# Ajustamos un modelos bayesiano con prior R2(.999, "mean")
#ajuste_ord_bayesiano_Q6_Rgrande <-
#  stan_polr(Q6 ~ age, data = train_data_fit, method = "probit",
#            prior = R2(.999, "mean"), seed = 12345,
#            algorithm = "fullrank") # for speed only
#saveRDS(ajuste_ord_bayesiano_Q6_Rgrande,
#        file = here("modelado_estadístico/TP1/modelos/ajuste_ord_bayesiano_Q6_Rgrande.rds"))

# Cargo los modelos que previamente ajusté
ajuste_ord_bayesiano_Q6_Rchico <-
  readRDS(here("modelado_estadístico/TP1/modelos/ajuste_ord_bayesiano_Q6_Rchico.rds"))
ajuste_ord_bayesiano_Q6_Rgrande <-
  readRDS(here("modelado_estadístico/TP1/modelos/ajuste_ord_bayesiano_Q6_Rgrande.rds"))
```

```
modelsummary(list("R grande" = ajuste_ord_bayesiano_Q6_Rgrande,
                  "R chico" = ajuste_ord_bayesiano_Q6_Rchico),
             fmt = 4,
             title = "Parámetros estimados para un R2 chico y grande",
             width = .5) |>
  theme_tt("placement", latex_float = "H")
```

largo. Los mismos se encuentran guardados en archivos.

Table 3: Parámetros estimados para un R2 chico y grande

	R grande	R chico
age	−0.0276	−0.0174
1|2	−1.9286	−1.6924
2|3	−1.4982	−1.2659
3|4	−1.0178	−0.7776
4|5	−0.3653	−0.1230
Num.Obs.	194 826	194 826

Al igual que en los modelos `polr` de `MASS`, cada modelo ajusta un parámetro para el β y 4 para los umbrales que separan las categorías. En la Tabla 3 se muestra la comparación de los parámetros estimados con `stan_polr` para un R^2 chico y otro grande.

```
# Una figura comparando las posterioris de beta_age
as_tibble(as.matrix(ajuste_ord_bayesiano_Q6_Rchico)) |>
  mutate(R = "Chico") |>
  bind_rows(as_tibble(as.matrix(ajuste_ord_bayesiano_Q6_Rgrande)) |>
              mutate(R = "Grande")) |>
  ggplot(aes(x = age, fill = R)) +
  geom_density(color = NA, alpha = .6) +
  scale_fill_brewer(palette = "Accent") +
  geom_vline(xintercept = 0, linetype = "dashed") +
  labs(x = "Beta", y = NULL) +
  theme_bw() +
  theme(legend.position = "top")
```

Se esperaría que, al ser el prior de R^2 extremadamente bajo, esto tenga un efecto de regularización en los coeficientes del ajuste, llevando los mismos a cero. Sin embargo, aunque hay un movimiento considerable hacia el cero (estimaciones puntuales de -0.0276 a -0.0174), en la Figura 6 no se observa que se alcance el cero. Esto podría deberse a que, al ser tan grande el n de nuestros datos de ajuste (194826), en la distribución a posteriori tiene mucho más peso relativo la parte dependiente de los datos (máxima verosimilitud) que la distribución a priori. Para poner a prueba esto podemos ajustar los mismos modelos pero con una muestra aleatoria de 5000 datos de entrenamiento.

```
# Ajustamos los mismos modelos pero con una muestra aleatoria de 5000 filas de
# los datos de entrenamiento
#set.seed(12345)
```

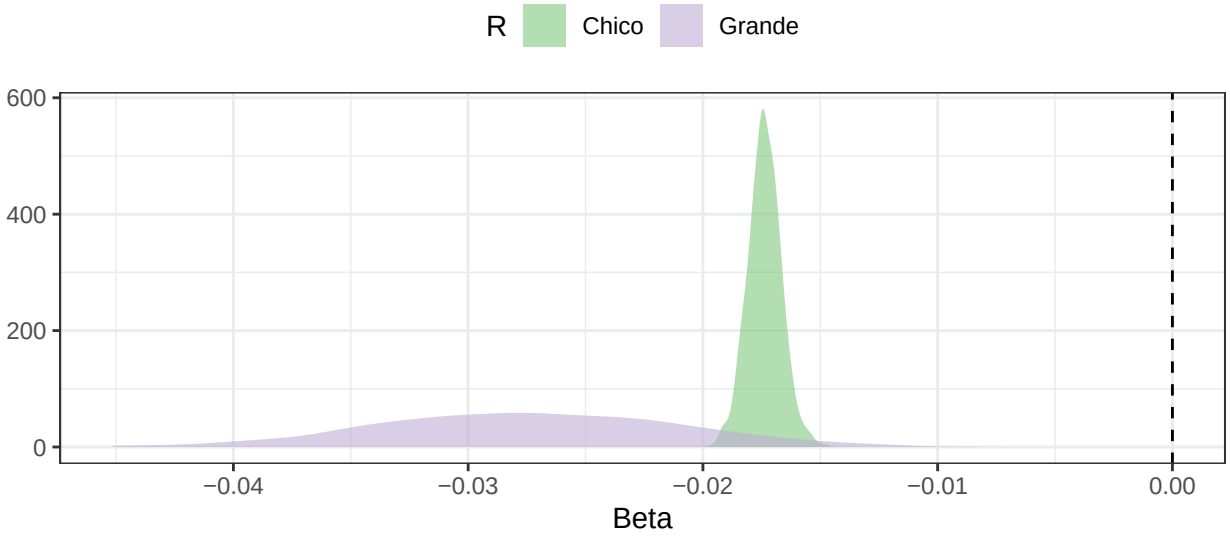

Figure 6: Comparación de la posteriori del beta para un modelo con prior R2 chico y grande.

```
#train_data_fit_short <- sample_n(train_data_fit, size = 5000)

# Volvemos a ajustar los modelos
# ajuste_ord_bayesiano_Q6_Rchico_menosdatos <-
#   stan_polr(Q6 ~ age, data = train_data_fit_short,
#             method = "probit",
#             prior = R2(.00001, "mean"), seed = 12345,
#             algorithm = "fullrank")
# saveRDS(ajuste_ord_bayesiano_Q6_Rchico_menosdatos,
#          file = here("modelado_estadístico/TP1/modelos/
#                      ajuste_ord_bayesiano_Q6_Rchico_menosdatos.rds"))
#
# ajuste_ord_bayesiano_Q6_Rgrande_menosdatos <-
#   stan_polr(Q6 ~ age, data = train_data_fit_short,
#             method = "probit",
#             prior = R2(.99, "mean"), seed = 12345,
#             algorithm = "fullrank")
# saveRDS(ajuste_ord_bayesiano_Q6_Rgrande_menosdatos,
#         file = here("modelado_estadístico/TP1/modelos/
#                     ajuste_ord_bayesiano_Q6_Rgrande_menosdatos.rds"))

# Cargo los modelos que previamente ajusté
```

```
ajuste_ord_bayesiano_Q6_Rchico_menosdatos <-
  readRDS(here(paste0("modelado_estadístico/TP1/modelos/",
                "ajuste_ord_bayesiano_Q6_Rchico_menosdatos.rds")))
ajuste_ord_bayesiano_Q6_Rgrande_menosdatos <-
  readRDS(here(paste0("modelado_estadístico/TP1/modelos/",
                "ajuste_ord_bayesiano_Q6_Rgrande_menosdatos.rds")))
```

```
# Un summary de los modelos
modelsummary(list("R grande" = ajuste_ord_bayesiano_Q6_Rgrande_menosdatos,
                  "R chico" = ajuste_ord_bayesiano_Q6_Rchico_menosdatos),
             fmt = 4,
             title = "Parámetros estimados para un R2 chico y grande con n=5000",
             width = .5) |>
  theme_tt("placement", latex_float = "H")
```

Table 4: Parámetros estimados para un R2 chico y grande con n=5000

	R grande	R chico
age	−0.0267	−0.0027
1|2	−1.9154	−1.3207
2|3	−1.4904	−0.8969
3|4	−0.9676	−0.4148
4|5	−0.3214	0.1685
Num.Obs.	5000	5000

En la Tabla 4 se muestra la comparación de los parámetros estimados con `stan_polr` para un R^2 chico y otro grande pero ahora con una muestra de tamaño más chica.

En este caso podemos ver que efectivamente el β vale muy cercano a cero para el modelo con el prior R2 pequeño. Esto también puede verse en la distribución a posteriori mostrada en la Figura 7.

```
# Una figura comparando las posterioris de beta_age
as_tibble(as.matrix(ajuste_ord_bayesiano_Q6_Rchico_menosdatos)) |>
  mutate(R = "Chico") |>
  bind_rows(as_tibble(as.matrix(ajuste_ord_bayesiano_Q6_Rgrande_menosdatos)) |>
              mutate(R = "Grande")) |>
  ggplot(aes(x = age, fill = R)) +
```

```
  geom_density(color = NA, alpha = .6) +
  scale_fill_brewer(palette = "Accent") +
  geom_vline(xintercept = 0, linetype = "dashed") +
  labs(x = "Beta", y = NULL) +
  theme_bw() +
  theme(legend.position = "top")
```

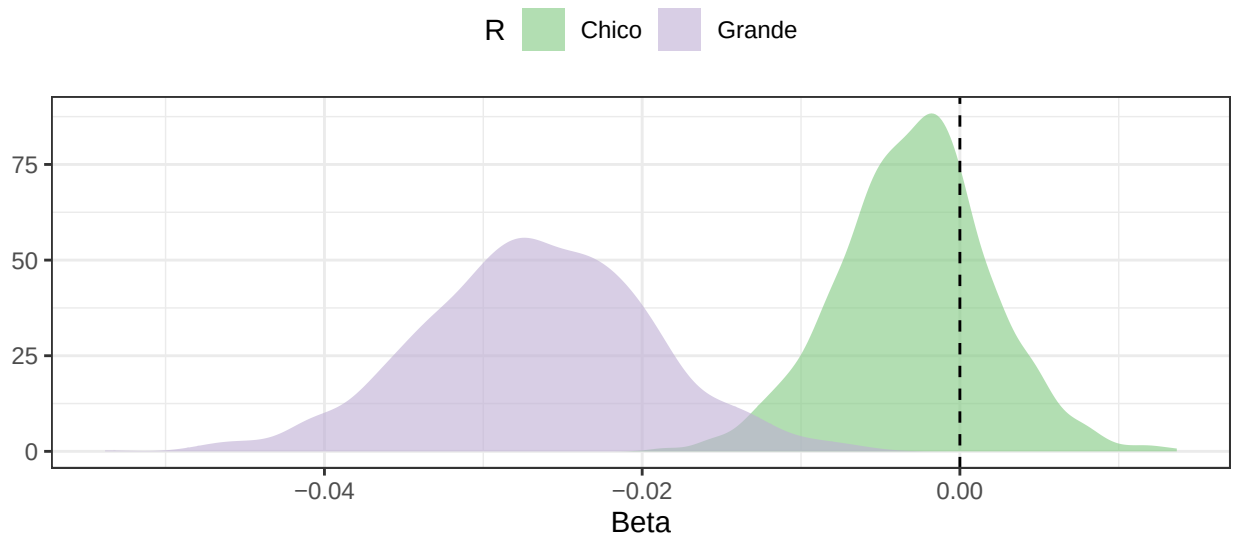

Figure 7: Comparación de la posteriori del beta para un modelo con prior R2 chico y grande para una muestra al azar de 2000 filas de los datos de entrenamiento.

11. Implementar un modelo de regresión ordinal bayesiano únicamente usando el paquete `rstan`.

Para realizar este punto, por una cuestión de practicidad, vamos a tomar una muestra más chica dentro del conjunto de entrenamiento para agilizar el tiempo de corrida del modelo. Vamos a considerar nuevamente 5000 observaciones elegidas al azar dentro del conjunto de train.

En el archivo `punto11.stan` se encuentran todas las especificaciones del modelo. En este caso consideramos como función F la función logística y pusimos una distribución a priori para el parámetro β $\mathcal{N}(0, 10)$.

```
N <- 5000 # Tamanio de la muestra
K <- 5

# train_data_fit <- train_data |>
#   dplyr::select(c("age", "Q6"))
# set.seed(12345)
```

```
# train_data_fit_short <- sample_n(train_data_fit, size = N)
# y <- train_data_fit_short$Q6
# x <- train_data_fit_short$age
# Ajustamos el modelo
#ajuste_ord_stan <- stan(file = "modelado_estadístico/TP1/modelos/punto11.stan",
#                        data = c('N','K','y','x'), iter = 1000)
# Guardamos el modelo
# saveRDS(ajuste_ord_stan, file = here("modelado_estadístico/TP1/modelos/ajuste_ord_stan.rds"))

# Cargamos el modelo pre ajustado
ajuste_ord_stan <- readRDS(here("modelado_estadístico/TP1/modelos/ajuste_ord_stan.rds"))
```

```
modelsummary(list("Stan" = ajuste_ord_stan),
             fmt = 4,
             title = "Parámetros estimados con rstan.",
             width = .5) |>
  theme_tt("placement", latex_float = "H")
```

Table 5: Parámetros estimados con rstan.

	Stan
beta	−0.0453
c[1]	−3.3019
c[2]	−2.4852
c[3]	−1.6509
c[4]	−0.6235

```
as_tibble(as.matrix(ajuste_ord_stan)) |>
  select(-lp__) |>
  pivot_longer(cols = everything(),
               names_to = "param",
               values_to = "value") |>
  ggplot(aes(x = value)) +
  geom_density(fill = "steelblue", alpha = .6) +
  facet_wrap(~param, scales = "free") +
  labs(x = "Estimaciones", y = NULL) +
  theme_bw() +
  theme(legend.position = "top") +
```

```
  theme(strip.background = element_rect(colour="white", fill="white"))
```

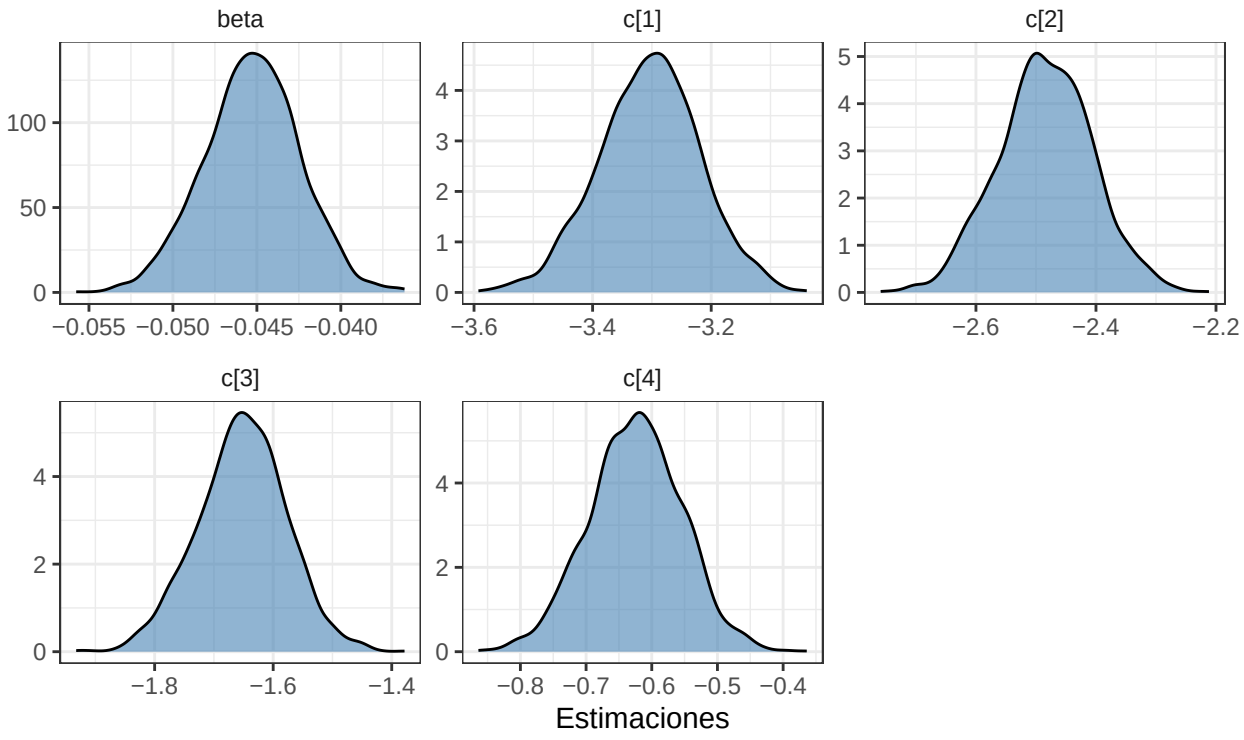

Figure 8: Distribucion a posteriori de los parámetros del modelos ajustado utilizando Stan.

En la Figura 8 podemos ver las distribuciones a posteriori de los diferentes parámetros del modelo y en la Tabla 5 las estimaciones puntuales. El valor estimado para el parámetro β es -0.0453 y podemos observar que difiere de los valores obtenidos con los modelos del punto anterior, pero hay que tener en cuenta que en este caso la función del modelo es la logística y la diferencia puede deberse a eso. Para tener una comparación, ajustamos el modelo logístico con `polr`.

```
train_data_fit <- train_data |>
  dplyr::select(c("age", "Q6")) |>
  mutate(Q6 = as.factor(Q6))
ajuste_ord_Q6_l <- polr(Q6 ~ age, data = train_data_fit)
```

La estimación del parámetro β con `polr` es -0.0462. Ahora vemos que los resultados se parecen y esto probablemente se deba a que la priori que pusimos no es muy informativa.