UNIVERSIDAD TORCUATO DI TELLA

Profesores: Ezequiel Merovich, Julián Regatky, Alfonso Gauna

Laboratorio para el Ánalisis de Datos

Examen Domiciliario

Grupo:

Tomás Leonel Temudio (20Q141)

Lautaro Castares (20N174)

Gonzalo Tajada (19N636)

Fecha de entrega: 9 de noviembre, 2022

Contents

1 Ejercicio 1

1.1 Inciso 1

Para realizar este inciso lo primero que hacemos es crear un variable que va a ser común para todas las distintas observaciones que hacemos en el inciso. Esta es la variable *"simulaciones = 1000"*. Una vez definida vamos a comenzar con las distintas observaciones, voy a explicar solo lo hecho con un *"n = 30"* ya que para el resto de las observaciones el código es análogo sólo con variantes en los nombres de las variables.

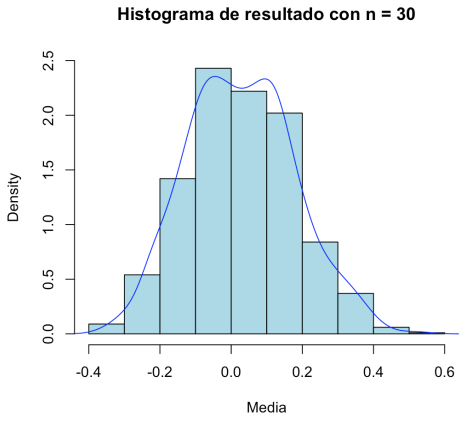
Para comenzar creamos una variable llamada *"n1"* que va a representar la cantidad de observaciones, en este caso es igual a 30. Luego generamos una variable llamada *"muestra_30"* que consiste de una muestra de 30 número aleatorios generados con una distribución normal de media 0 y desvío estándar 1. También creamos una variable llamada *"resultado_30"* que contiene un vector vacío que luego se va a ir completando.

Continuamos con un for loop que va desde 1 hasta la cantidad de simulaciones(1000). En este loop vamos a ir rellenando por índice la variable *"resultado_30"*, esta se va a ir completando con la media de un sample que utiliza los datos de la variable *"muestra_30"*.

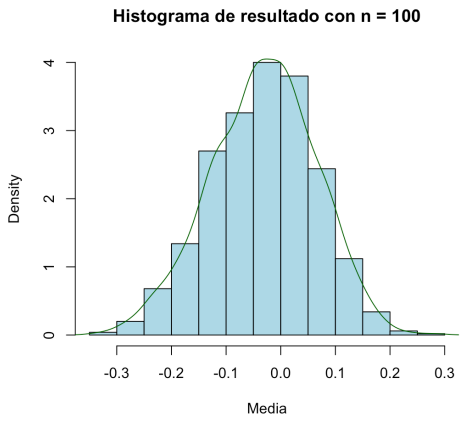
Para ir finalizando, creamos dos variables llamadas *"media_30"* y *"sd_30"* las cuales contienen la media y el desvío estándar de la variables *"resultado_30"*. Por último creamos un histograma que nos muestra gráficamente cómo se distribuye la muestra calculada. Además le agregamos la línea que muestra la estimación de densidad de Kernel para cada observación.

Al comparar los resultados obtenidos con las diferentes observaciones, podemos concluir que a medida que aumenta el número de observaciones que utilizamos la muestra se asemeja cada vez más a una distribución normal. [1]

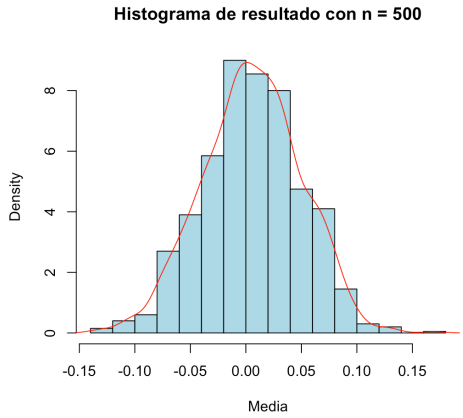
[1] En la página de abajo podemos ver los histogramas y la tabla de datos obtenida.



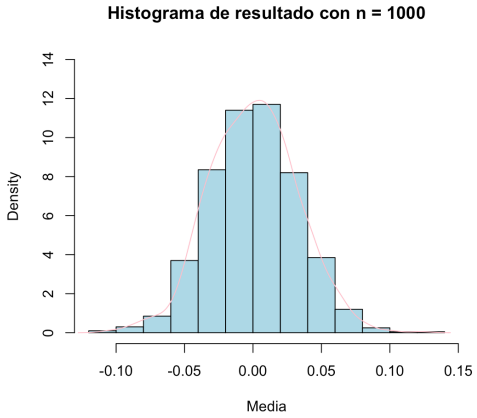
(a) n = 30



(b) n = 100



(c) n = 500



(d) n = 1000

Figure 1: Gráficos para las distintas observaciones

n	Media	Desvío estandar
30	0.0280	0.1513
100	-0.0318	0.0962
500	0.0056	0.0443
1000	0.0004	0.0320

Table 1: Resumen de resultados

2 Ejercicio 2

2.1 Inciso 1

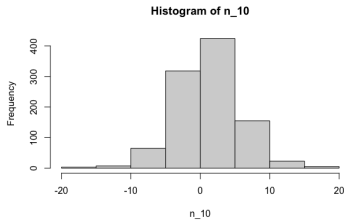
Para realizar el ejercicio se utilizó paquetes y librerías no vistos con anterioridad en clase ("pacman" y "MonteCarlo") . Para poder realizar el loop con anterioridad se tuvo que crear la data con las variables (X e Y), el error (U) y guardar todo en tabla *"data.table"*. Un vez creada la data, se continúa con la regresión mediante la función *"feols"*, se extrae los datos de los coeficientes para después agruparlos en una lista y finaliza con la creación de la función.

Con la función creada, procedemos al armado del loop. Para esto creamos una grilla *"list"* con los distintos tamaños de muestras, el intercepto y la pendiente. Con todo esto, corremos y guardamos el loop con la función *"MonteCarlo"*, especificamos el número de repeticiones por test, para después extraer los resultados y almacenarlos como valores separados.

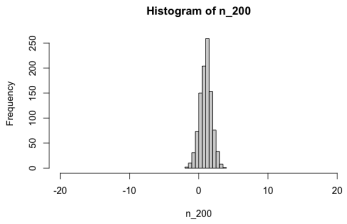
Con las regresiones hechas y almacenadas procedemos, continuamos con los requisitos del inciso. Dichos requisitos consisten en hacer tests de hipótesis para los distintos montecarlos con dos hipótesis nulas distintas (H0 = 1, H0 = 0.5), para después obtener las medias, el error tipo 1 (p.valor) y la potencia del test. Los objetivos de estos test son que a pesar de caer en la aleatoriedad de las pruebas, n_20 podría tener mejores números que n_100, se compruebe la veracidad de modelo de Montecarlo y al testear los valores con muestras más grandes estos tiendan a mejores resultados.

Por fortuna los resultados fueron los esperados. Para el primer test donde nuestra hipótesis nula era H0 = 1 (El número que pretendíamos de la pendiente) encontramos una media que con el crecer de las muestras convergia más a 1, un valor t (si es alto rechazo la hipótesis nula) cada vez más bajo y un p.valor (si es alto no rechazo la H0) cada vez más alto. Finalmente con el último test encontramos resultados todavía más claros, con un p.valor decreciente y un valor T creciente a medida que crecía la muestra, resultados

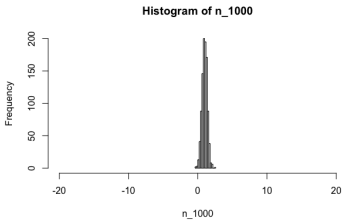
que se buscaban.



(a) n = 10



(b) n = 200



(c) n = 1000

Figure 2: Gráficos para las distintas observaciones

	nombre_filas	Sample_10	Sample_20	Sample_100	Sample_200	Sample_500	Sample_1000
1	mean	1.1486900	0.8593968	0.92967982	1.0275516	1.0277593	1.0016326
2	p	0.2983584	0.1425104	0.07381991	0.3035248	0.1031636	0.8930616
3	t	1.0405023	-1.4676635	-1.78960711	1.0294344	1.6311986	0.1344654

(a) Tabla de datos opcion 1

	nombre_filas	Sample_10b	Sample_20b	Sample_100b	Sample_200b	Sample_500b	Sample_1000b
1	mean	0.959987155	0.72172363	9.468272e-01	1.017653e+00	9.960752e-01	9.954560e-01
2	p	0.001399943	0.01690385	5.411834e-30	3.116830e-69	2.700860e-138	1.844308e-217
3	t	3.203644386	2.39280789	1.175933e+01	1.904529e+01	2.953226e+01	4.117579e+01

(b) Tabla de datos opcion 2

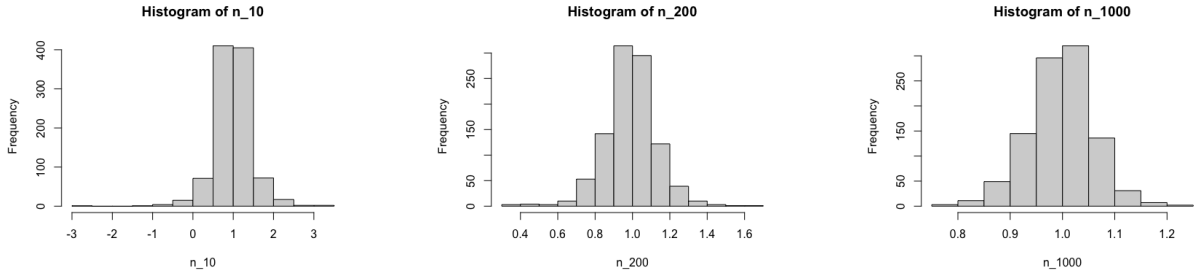
Figure 3: Tablas de datos para cada sample

2.2 Inciso 2

Para este segundo inciso se siguió con el mismo procedimiento del inciso anterior pero a diferencia de la regresión del punto 1, esta presenta problemas de heterocedasticidad en el error de la regresión lo que podría afectar los test de hipótesis dado el sesgo del error estándar.

Por fortuna, cuando se aplica monte carlo para pruebas con heterocedasticidad afecta nuestras estimaciones de la varianza de beta1 (Pudimos comprobar viendo los histogramas) pero no afecta la insesgadez y por lo tanto va a afectar la inferencia estadística.

Los resultados de los test de hipótesis no son exactamente los mismos pero si se puede obtener las mismas conclusiones que los test del inciso anterior.



(a) n = 10 (b) n = 200 (c) n = 1000

Figure 4: Gráficos para las distintas observaciones

	nombre_filas	Sample_10	Sample_20	Sample_100	Sample_200	Sample_500	Sample_1000
1	mean	0.9879734	0.9877530	1.0056521	0.988459491	0.9978087	0.99655795
2	p	0.3957683	0.2798713	0.3662531	0.008500239	0.4351206	0.07738615
3	t	−0.8495689	−1.0811950	0.9039249	−2.636754492	−0.7807737	−1.76788011

(a) Tabla de datos opcion 1

	nombre_filas	Sample_10b	Sample_20b	Sample_100b	Sample_200b	Sample_500b	Sample_1000b
1	mean	9.879734e−01	9.877530e−01	1.005652	0.9884595	0.9978087	0.996558
2	p	3.466030e−172	6.963895e−230	0.000000	0.0000000	0.0000000	0.000000
3	t	3.447072e+01	4.305998e+01	80.867319	111.6023388	177.3736033	255.038477

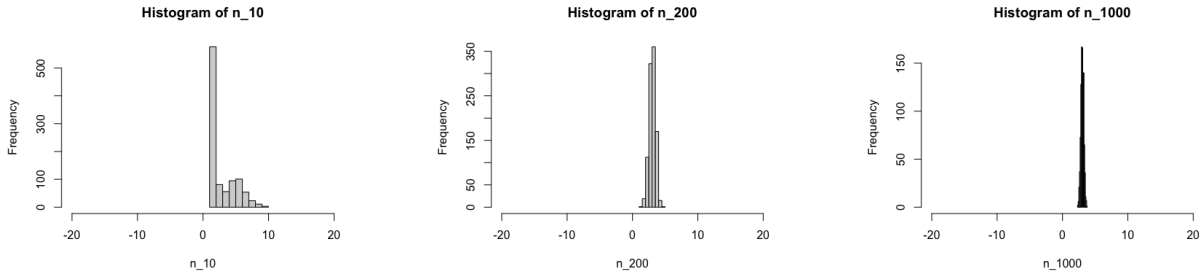
(b) Tabla de datos opcion 2

Figure 5: Tablas de datos para cada sample

2.3 Inciso 3

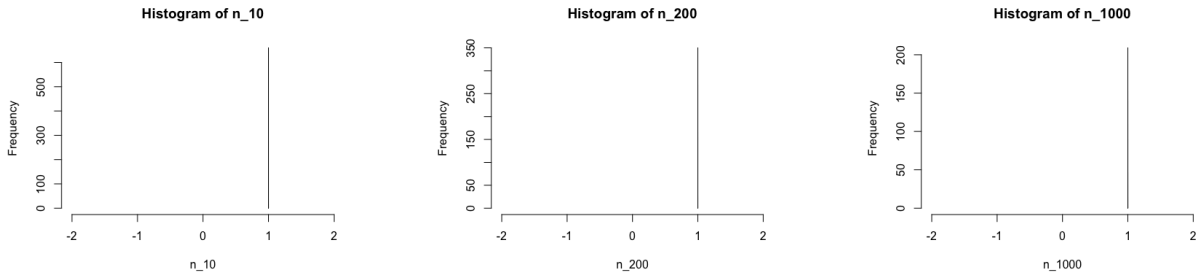
Para el último inciso, se pide comprobar que el método de monte carlo también es efectivo para errores sin distribución normal. Para eso corrimos las regresiones y llegamos a la conclusión viendo los histogramas que el método de Montecarlo era igual de efectivo.

Esto lo sabemos gracias a la teoría, la cual precisa que no importa el tipo de error, el método de monte carlo es muy efectivo para llegar a los valores verdaderos.



(a) n = 10 (b) n = 200 (c) n = 1000

Figure 6: Gráficos para las distintas observaciones



(a) n = 10 (b) n = 200 (c) n = 1000

Figure 7: Gráficos para las distintas observaciones

3 Ejercicio 3

3.1 Inciso 1

Comenzamos el inciso creando una variable llamada *"files"* en donde utilizamos la funcion `list.files` para crear una lista con los nombres de los archivos a partir de los cuales vamos a iterar. Aquí solo nos quedamos con los archivos que comienzan con *"US"* excluyendo así al archivo llamado *"Data_1900_2019stations"*. El siguiente paso es crear un data frame formado de solo una columna de fechas que van desde 1900/1/1 hasta 1970/12/31. En nuestro caso este data frame se llama *"dates"* y es el que vamos a ir completando con los valores de lluvia por estación en cada iteración. Antes de comenzar con el for loop creamos una variable llamada *"id"* que va a estar vacía y se va a ir completando con el ID de cada estación a medida que avanza el loop.

Iniciando el for loop, vamos a iterar a lo largo de los 749 archivos que poseemos. Aquí es donde utilizamos la variable *"files"* que contiene los nombres de cada uno de los archivos. El primer paso del loop es generar un dataframe llamado *"data"* en donde se utiliza la función `read.csv`. Lo que hacemos con esto es leer los datos para el i-ésimo archivo y almacenarlo en *"data"*. Una vez hecho esto, vamos a tener dentro del data frame *"data"* a

la información en su formato original. Por lo tanto vamos a comenzar a modificarla para lograr hacer el `left_join` al data frame *"dates"*.

Lo primero que hacemos es seleccionar las variables que nos interesan y pasamos el data frame a formato long. De esta forma logramos que en cada fila represente un día que posee su información de lluvia correspondiente. Una vez que tenemos esto listo, con la función mutate creamos una columna de formato Date llamada *"date"* y eliminamos las columnas *"ID"*, *"year"*, *"month"*. Que la columna *"date"* tenga el formato Date nos permite nos permite unirla al data frame *"dates"*.

Para concluir con el loop, utilizamos un `left_join` donde vamos a unir los data frames *"dates"* y *"data"* utilizando la columna *"date"* como parámetro de unión. De esta forma obtenemos un data frame que posee 25,932 (una fila para cada día) y 750 columnas. Cada columna representa una estación.

3.2 Inciso 2

Para este inciso, el proceso es el mismo que en el inciso anterior. La diferencia se encuentra en que ahora solo nos quedamos con la información de las flags. Estas son las columnas que comienzan con "xxQ". Vamos a obtener un data frame final de las misma dimensiones que en el inciso anterior. La diferencia se encuentra en la información que posee cada data frame. El del inciso 1 posee datos de la lluvia por día en cada estación mientras que el de este inciso posee información de las flags por día en cada estación.

3.3 Inciso 3

Aquí comenzamos tomando el data frame *"dates"* y lo pasamos a formato long para tener en cada fila la combinación de fecha, estación y valor de lluvia. Esto lo guardamos en un nuevo data frame llamado *"unified_dates"*. Lo mismo haremos con *"flags"* y lo guardamos en *"unified_flags"*. Antes de proceder a realizar el `left_join` transformamos

la columna de *"rain"* del data frame *"unified_flags"* a formato integer ya que sino sería imposible realizar el `left_join` entre ambos data frames.

Una vez hecho esto generamos el data frame *"unified_dataframe"* donde utilizamos un `left_join` entre *"unified_dates"* y *"unified_flags"* usando las columnas *"station"* y *"date"* como parámetros de unión. Una vez unidos ambos data frames vamos a pararnos en la columna *"flags* y utilizamos un ifelse que cuando los valores de flags no son un `NA` los reemplaza con el valor original de rain.

3.4 Inciso 4

Para este inciso lo que hacemos es generar un data frame llamado *"max_observation"*. Dentro de este vamos a utilizar la función `arrange(desc())` aplicada a la columna *"rain"*. Esto nos permite ordenar de mayor a menor la cantidad de lluvia para cada combinación de fecha y estación. Luego vamos a utilizar `slice(1:10)` para quedarnos con los 10 mayores valores de lluvia.

3.5 Inciso 5

A partir de acá lamentablemente no pudimos continuar debido a que tuvimos muchos inconvenientes con el inciso 3. Nos ocurría que al querer unir ambos data frames en uno nuestras computadoras no mostraban un error de memoria ram exhausta. Esto lo hablamos con Alfonso para ver si había una solución. Logramos llegar a destrabarlo el mismo domingo de la entrega y no nos dio el tiempo para continuar con el resto de los incisos.

4 Ejercicio 4

4.1 Inciso 1

Simplemente siguiendo la fórmula de la suma de residuos cuadrados y usando 4 variables que representen a, b, x e y, es posible definir la función *'RSS_funcion'*.

4.2 Inciso 2

Usamos la función read.table para importar el archivo hibbs.dat y subsecuentemente, lo guardamos en la sección de Data de nuestro environment.
Luego, definimos la regresión *'regresionej4'*, siendo vote la variable dependiente y growth la variable explicativa. Realizamos un summary de la regresión y obtenemos un β_0 (constante) igual a 46.2476 y una β_1 igual a 3.0605.

4.3 Inciso 3

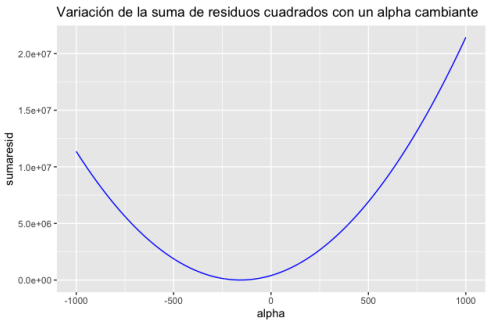
De acuerdo a la consigna, aquí usaremos $\beta = 3.0605$; x = growth; y = vote; α cambiante y la fórmula de la suma de residuos cuadrados obtenida en el inciso 1. El parámetro α lo creamos mediante una secuencia numérica que va desde -1000 hasta 1000.

Para evitar problemas a la hora de correr nuestro ejercicio definimos un límite muy alto en la cantidad de observaciones que tiene permitido observar R. Luego creamos el data frame *'grafico'*, donde una columna representará los valores de alpha y la otra su respectiva suma de residuos cuadrados. Esta última columna comenzará vacía y se irá llenando a medida que corra el for loop. Los valores se irán agregando en la posición $\alpha + 1001$. Esto se debe a que, al comenzar en $\alpha = -1000$, nos asegura que vayan a parar desde la fila 1 hasta la 2001.

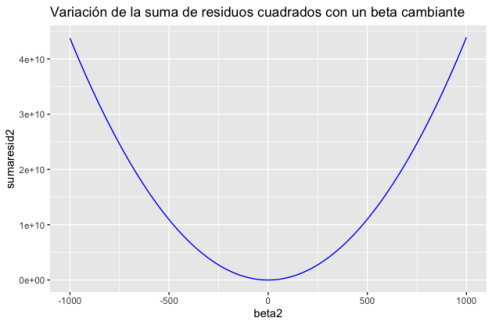
Para hacer el gráfico, usamos las librerías ggplot2 y dplyr. En el primer caso, seteamos los valores de α para que se mueva a través del eje de abscisas, mientras que en el eje de

ordenadas se verán reflejados los valores de la suma de residuos cuadrados. Nos queda una parábola centrada en un valor ligeramente negativo, cuando debería centrarse alrededor de $\alpha = 46$ (por definición de mínimos cuadrados clásicos). Esto se puede deber a un error de parte nuestra a la hora de estimar la suma de residuos o de leer los datos.

Esta sección del inciso la hacemos de la misma forma, solo que ahora es α el valor fijo y es β quien varía. Es decir, ahora definimos a beta como una secuencia de -1000 a 1000, mientras que fijamos el parámetro $\alpha = 46.2476$. Aquí el gráfico muestra que la suma de residuos al cuadrado se minimiza en un β aproximadamente igual a 0, lo cual cuadra con los resultados de nuestra regresión ($\beta = 3.0605$).



(a) Variacion de la suma de residuos cuadrados con un α cambiante

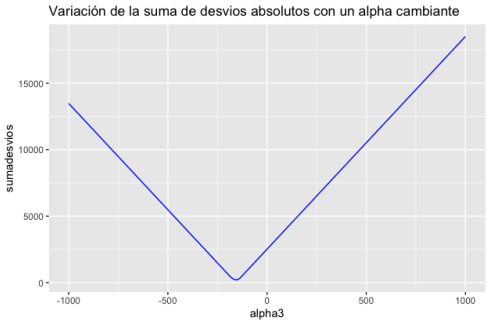


(b) Variacion de la suma de residuos cuadrados con un β cambiante

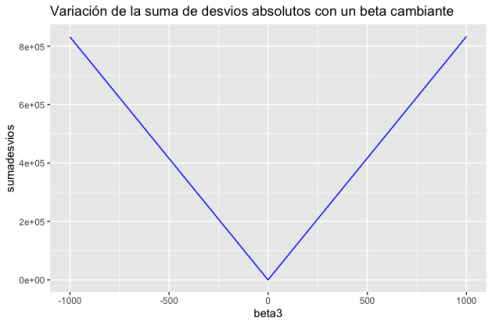
Figure 8: Gráficos para las variaciones de la sumas de residuos al cuadrado

4.4 Inciso 4

En este último inciso creamos una función para medir la suma de desvíos absolutos, que es similar a la creada en el inciso 1, sólo que ahora tomamos el valor absoluto y no elevamos la suma al cuadrado. El proceso en sí es prácticamente idéntico, solo que ahora aplicamos la función *'desvios_funcion'* y obtenemos un gráfico en forma de V. Nuevamente, en el caso del α cambiante, se minimizan los desvíos con un α ligeramente negativo, mientras que con un β cambiante, el gráfico muestra que los desvíos se minimizan con un β casi igual a 0, lo que cuadra con los resultados obtenidos en el inciso 2.



(a) Variacion de desvios absolutos con un α cambiante



(b) Variacion de desvios absolutos con un β cambiante

Figure 9: Gráficos para las variaciones de los desvios absolutos

5 Ejercicio 5

5.1 Inciso 1

Para este inciso, lo primero que hacemos es leer los datos desde el archivo *"earnings.csv"* y lo guardamos en un data frame. Para comenzar con el forward stepwise selection lo primero que hacemos es un modelo llamado M0 donde regresamos la variable *"earnk"* contra una constante.

Ahora procedemos a explicar los pasos y la intuición para el modelo "M1". Lo haremos solo para este modelo ya que para el resto es análogo.

Para comenzar, definimos un vector numérico vacío llamado *"residuos"* en donde vamos a acumular los residuos en cada iteración del for loop. Luego creamos un data frame llamado *"data_wo_earnk"* en el cual almacenamos los mismos datos que en el data frame *"Data"* solo que extraemos la columna que representa los datos del ingreso. Esto lo hacemos para que la iteración sea a través de 13 variables y no se incluya el ingreso. El paso siguiente es comenzar con el for loop en donde vamos a iterar a través de las 13 columnas de *"data_wo_earnk"* y en cada iteración guardamos el nombre de la i-ésima hilera en *"variable"*. Esto nos permite realizar una regresión en donde la variable explicada es *"earnk"* y la explicativa es *"variable"*. Esta última en cada iteración toma los valores de una columna distinta de las 13 que nos quedaron en *"data_wo_earnk"*. Una vez hecho esto, pasamos a calcular la suma de los residuos al cuadrado con la función `deviance` y los almacenamos en la variable *"residuos"* creada antes de comenzar el for loop.

Cuando ya obtenemos los residuos para cada iteración, encontramos cuál fue la variable con el menor residuo y guardamos el nombre en la variable *"name_column_i"*.

Para continuar con el proceso, como ya tenemos conocimiento de la variable con menor suma de residuos al cuadrado del modelo 1, la excluimos del nuevo data frame que creamos para el modelo 2. Este data frame se llama *"data_wo_earnk2"* y está conformado por la información de *"data_wo_earnk"* menos la columna perteneciente a la columna *"father_education"* que fue la variables con menor residuo del modelo 1. El for loop para el modelo 2 posee la misma lógica que el anterior solo que ahora agregamos a la regresión a la columna *"father_education"* como variable explicativa constantes e iteramos entre las 12 variables restantes. De esta forma continuamos el proceso, extrayendo en cada paso la variables con menor suma de residuos al cuadrado y agregandola como constante en la regresion del modelo siguiente.

Una vez que tenemos todos los modelos generados calculamos el AIC para todos los

modelos con la función `AIC` y lo almacenamos en el data frame *"AIC_forward"*. Una vez hecho esto, encontramos cual es el modelo con menor AIC. En nuestro caso este fue el modelo M8. Esto nos dice que este modelo es el que mejor se adapta a los datos.

5.2 Inciso 2

En este inciso se nos pedía repetir el inciso anterior, pero ahora realizando un algoritmo de backwards stepwise. Lo primero que debemos hacer es crear una regresión donde 'earnk' sea la variable dependiente y todas las columnas del data frame 'data' sean las variables independientes. Una vez hecho esto, calculamos su suma de residuos cuadrados y su AIC, luego buscamos aquella fila con el mayor RSS y nos deshacemos de ese parámetro en el data frame original 'data'. Repetimos este proceso 8 veces hasta que llegamos al caso en que únicamente nos quedan las variables que refieren a educación, tense y angry, las cuales tienen la misma suma de residuos cuadrados. Siendo sinceros, el script no nos resultó simple y, por lo tanto, no pudimos expresar de manera concreta lo que pedía este inciso. Probablemente haya una forma más eficaz de resolverlo.