TEORÍA Y PRAXIS DE MODELOS GENERALIZADOS: INFIRIENDO PATRONES CON EL PAQUETE ESTADÍSTICO R

Ejemplo de análisis con una variable respuesta con elevada carga de ceros.

Curso de la Sociedad de Amigos del Museo Nacional de Ciencias Naturales - CSIC

Luis M. Carrascal

Marzo 2015

CONTENIDO:

Modelo Generalizado Lineal con una respuesta Binomial Negativa

Modelo Generalizado Lineal “Hurdle” asumiendo inflado de ceros

Modelo Generalizado Lineal Logístico, con una respuesta Binomial estimada como frecuencia

Modelo Generalizado Aditivo con una respuesta Binomial estimada como frecuencia

Revisión de los supuestos canónicos de los modelos

Estimas robustas de los parámetros y significación del modelo

Estimas frecuentistas y multimodelo basadas en la Teoría de la Información

Visualización de los efectos parciales de las predictoras sobre la respuesta

Modelos nulos para estima de significación

Sea una variable respuesta que cuantifica la abundancia relativa de una especie durante el invierno en cuadrículas UTM de 10x10 km2. La variable respuesta mide la aparición de la especie en 60 transectos de 15 minutos de duración (más tarde retomaremos esta idea). La especie es el Avión roquero (http://www.seo.org/ave/avion-roquero/). Los datos provienen del Atlas de las Aves Invernantes en España (http://www.magrama.gob.es/es/biodiversidad/publicaciones/atlas_aves_invierno_tcm7-291664.pdf). En esta ocasión vamos a dejar de lado el aspecto de la independencia muestral entre las UTM, por cuestiones meramente prácticas de exposición del modo de trabajar.

Primeramente vamos a cargar todos los paquetes que utilizaremos en la secuencia de análisis que vamos a realizar.

```
library(MuMIn)      ## para calcular AICc
library(car)        ## para obtener VIF's y usar Anova, bootCase
library(MASS)       ## para usar dropterm
library(lmtest)     ## para obtener la p del modelo, en sustitución de anova(nulo, model, test="Chisq")
library(moments)    ## para obtener el sesgo, kurtosis y sus significaciones: kurtosis, anscombe.test, skewness, agostino.test
library(sandwich)   ## para estima robusta, estimando sandwich standard errors
library(robustbase) ## para estimas robustas usando Mallows-Huber quasilikelihhod estimation
library(wle)        ## para estimas robustas usando Weighted Likelihood Estimating Equations
library(psych)      ## para uso del comando describe
library(fit.models) ## para usar el comando leverage
```

Observamos el contenido de nuestro juego de datos (*data frame*) llamado “datos”, con un tamaño muestral de 999.
Con `str(datos)` podemos conocer la naturaleza de las variables.

```
names(datos)
 [1] "ID"          "longitud"    "latitud"     "distmar"     "altmed"      "rangoalt"    "altmax"      "shannon"     "ice"
[10] "fcaducif"    "fesclerof"   "fconif"      "fagroarb"    "furbano"     "fmatorral"   "fagromos"    "fagua"       "fherbaceo"
[19] "tempmin"     "tempmedia"   "precip"      "ibakm2"      "zepakm2"     "ntransectos" "nspp"        "ptyrup"      "turmer"
[28] "erirub"      "stuuni"      "petpet"      "ptyrup01"    "turmer01"    "erirub01"    "stuuni01"    "petpet01"    "ptyrup02"
[37] "turmer03"


str(datos)
'data.frame':    999 obs. of  45 variables:
 $ ID         : int  6 1582 424 868 935 1433 454 425 1398 1925 ...
 $ longitud   : num  -0.05 -2.77 -7.66 -4.58 -5.34 -4.1 -7.53 -7.56 -3.61 -1.47 ...
 $ latitud    : num  40.2 39.4 42.4 37.4 39.9 ...
 $ distmar    : int  15081 204611 85804 83167 296709 50674 70857 98119 127103 124772 ...
 $ altmed     : int  310 713 606 446 342 968 450 521 1017 1444 ...
 $ rangoalt   : int  399 39 1151 330 167 470 123 672 247 352 ...
 $ altmax     : int  532 739 1333 635 445 1242 515 917 1172 1629 ...
 $ shannon    : num  0.93 1.42 2.45 1.25 1.37 2.36 1.54 1.04 1.79 1.7 ...
 $ ice        : num  2.2 1.7 2.6 2.7 1.7 2.8 2.7 2.1 3.1 3.2 ...
 $ fcaducif   : num  -0.57 -0.79 1.45 -0.09 -0.87 2.38 1.48 -0.5 0.76 -0.52 ...
  ……
 $ ptyrup     : int  0 0 1 0 0 0 0 0 0 0 ...
 $ turmer     : int  3 1 30 6 10 25 35 40 24 12 ...
 $ erirub     : int  42 0 38 3 0 9 51 32 7 1 ...
 $ stuuni     : int  13 0 12 11 37 2 14 25 2 0 ...
 $ petpet     : int  0 0 0 0 0 0 0 3 3 1 ...
 $ ptyrup01   : int  0 0 1 0 0 0 0 0 0 0 ...
 $ turmer01   : int  1 1 1 1 1 1 1 1 1 1 ...
  ……
 $ nptyrup    : int  0 0 1 0 0 0 0 0 0 0 ...
 $ npetpet    : int  0 0 0 0 0 0 0 1 3 1 ...
 $ sizepa     : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
 $ zepa01     : num  0 0 0 0 1 0 0 0 0 0 ...
```

Nuestra especie de estudio, y variable respuesta de análisis, es `"ptyrup"`. Es la variable número [26] según podemos ver en la salida del comando `names(datos)`. Si queremos conocer el valor de la variable respuesta en la unidad muestral número 125 (fila dentro del *data frame*), tecleamos cualquiera de las dos siguientes opciones (entre corchetes sólo la fila para una variable que se especifica, o la [fila,variable] si sólo especificamos el juego de datos):

```
datos$ptyrup[125]
[1] 0
datos[125,26]
[1] 0
```

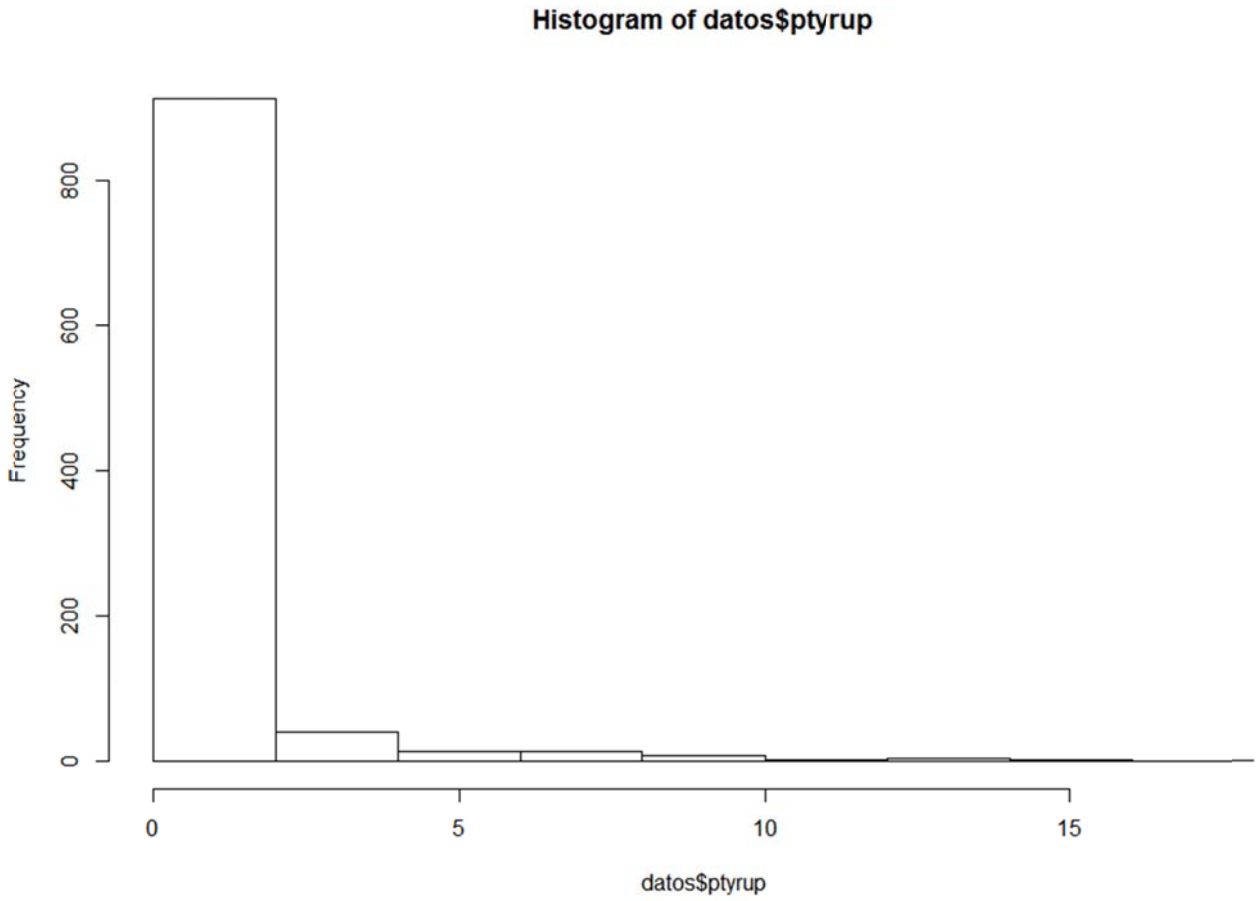
Calculamos la media (mean) y varianza (cuadrado de la desviación típica, sd) de nuestra variable respuesta `"ptyrup"`. Para no repetir la escritura de `datos$ptyrup` subimos en la consola de RStudio con el cursor una posición hasta que nos aparezca `mean(datos$ptyrup)` y cambiamos `mean` por `sd`, añadiendo tras el paréntesis `^2`:

```
mean(datos$ptyrup)
[1] 0.7137137
sd(datos$ptyrup)^2
[1] 4.154431
```

Una distribución de Poisson viene caracterizada por tener números enteros y positivos; se establece con un solo parámetro lambda, que se corresponde con la media de la distribución, que a su vez coincide con la varianza: *lambda* = media = varianza. Este no es el caso de nuestra variable `datos$ptyrup`.

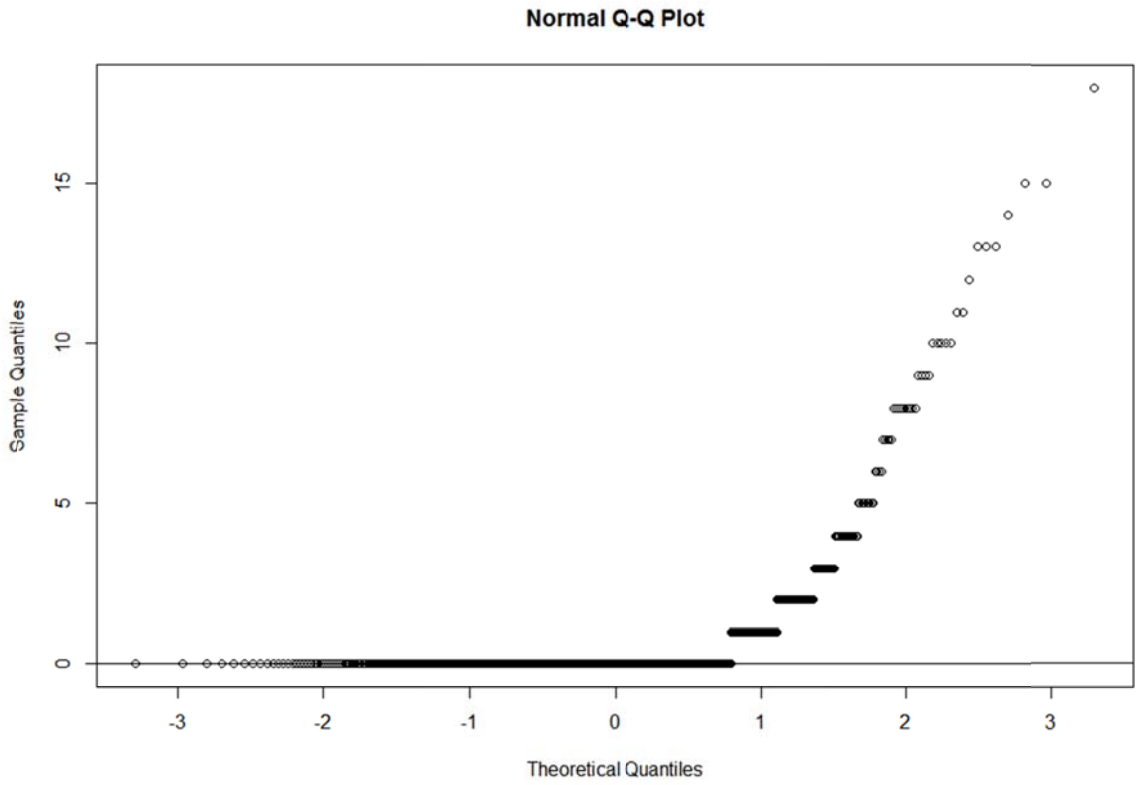
Veamos su histograma:

```
hist(datos$ptyrup)
```



Y ahora su *normal probability plot* (qqplot):

```
qqnorm(datos$ptyrup)
qqline(datos$ptyrup)
```



Hay muchísimos valores ceros.

Una distribución binomial negativa es aquella constituida por valores enteros positivos, cuya varianza viene definida en función de la media (mu) del siguiente modo:

$$\text{Varianza} = \text{mu} + \textit{alfa} * \text{mu}^2$$

En R, el valor alfa se define como *alfa* = 1 / size, de manera que size también es denominado theta (posteriormente lo veremos con el comando `glm.nb`).

Si nuestra variable fuese una binomial negativa, su size debería ser:

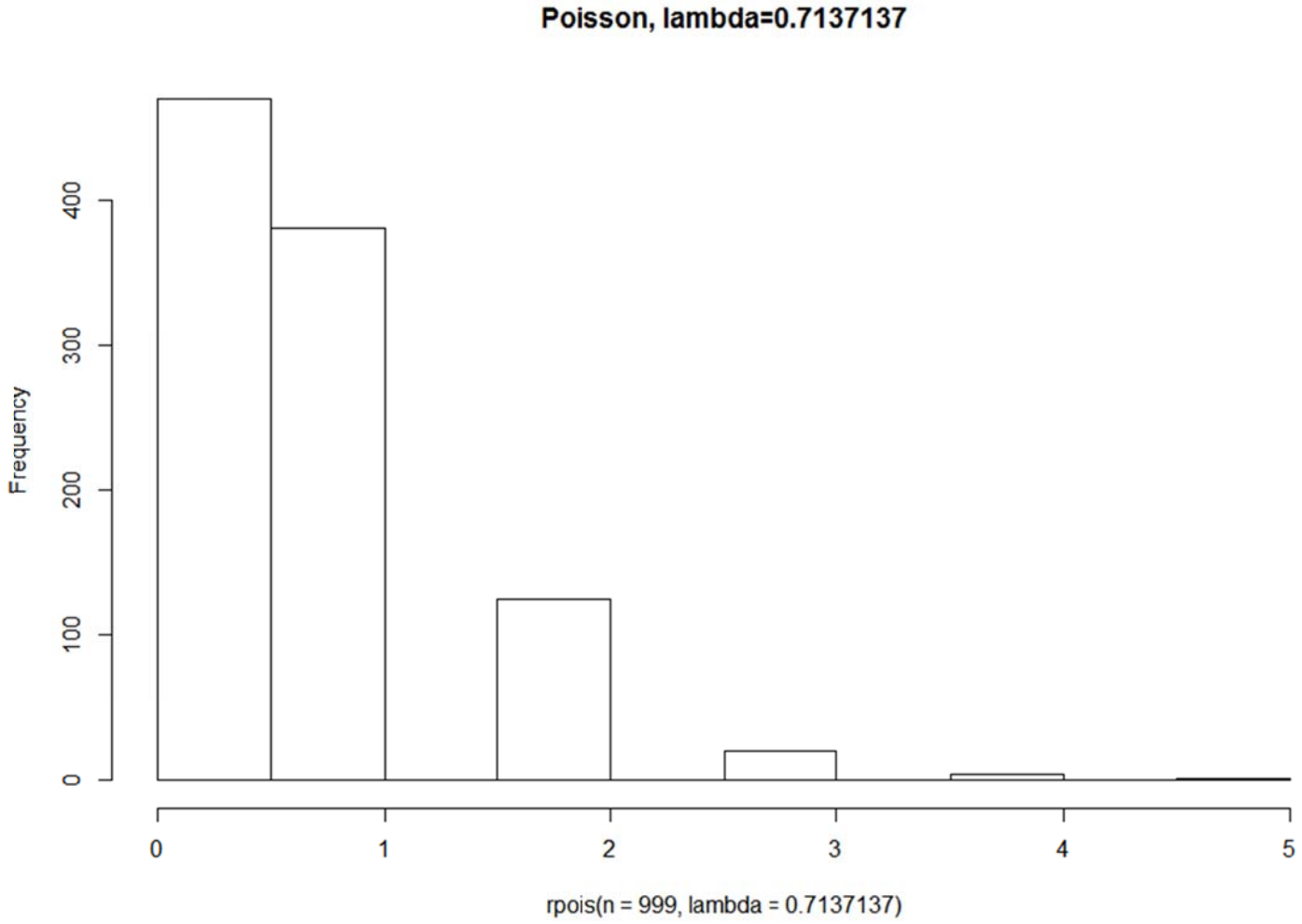
`sd(datos$ptyrup)^2 = 4.154431` = [`mean(datos$ptyrup) = 0.7137137`] + (1 / size) * 0.7137137^2

size = 0.7137137^2 / (4.154431 - 0.7137137) = **0.1480468** … que calculamos en la consola de RStudio:

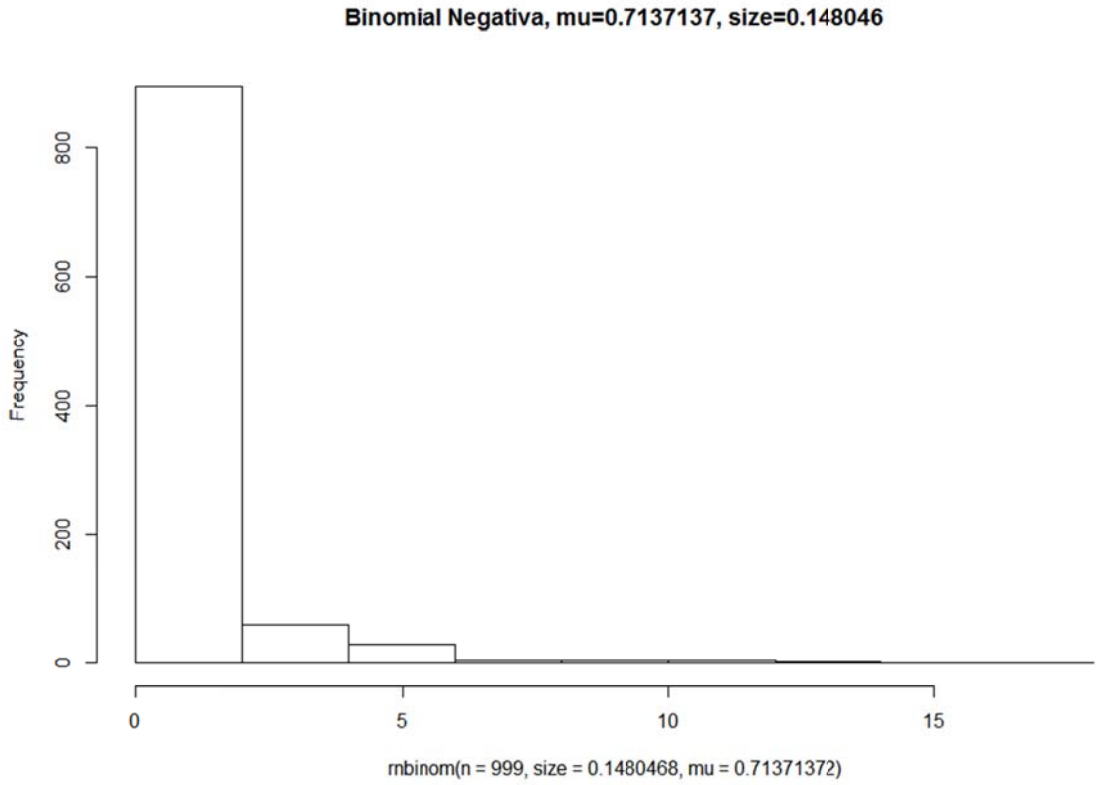
```
(0.71371372^2) / (4.154431 - 0.7137137)
[1] 0.1480468
```

Vamos a generar a continuación dos distribuciones de 999 datos, según una Poisson con media = mu = *lambda* = `0.7137137`, y una Binomial Negativa de media = mu = `0.7137137` y un size = `0.1480468`. Con el argumentao `main` ponemos el título superior a la figura.

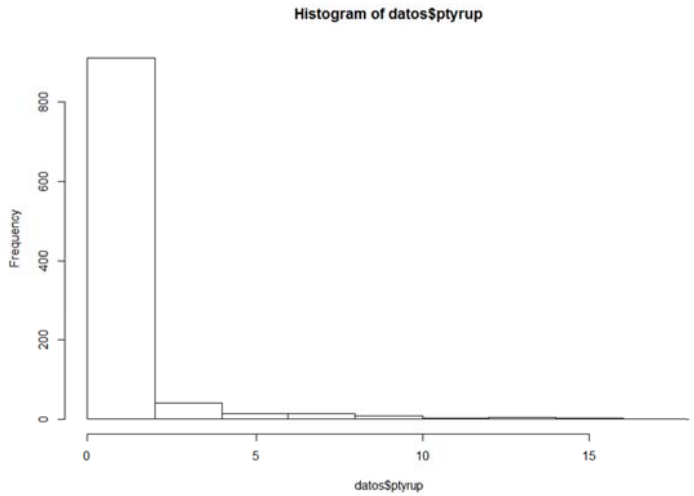
```
hist(rpois(n=999, lambda=0.7137137), main="Poisson, lambda=0.7137137")
```



```
hist(rnbinom(n=999, size=0.1480468, mu=0.71371372), main="Binomial Negativa, mu=0.7137137, size=0.148046")
```



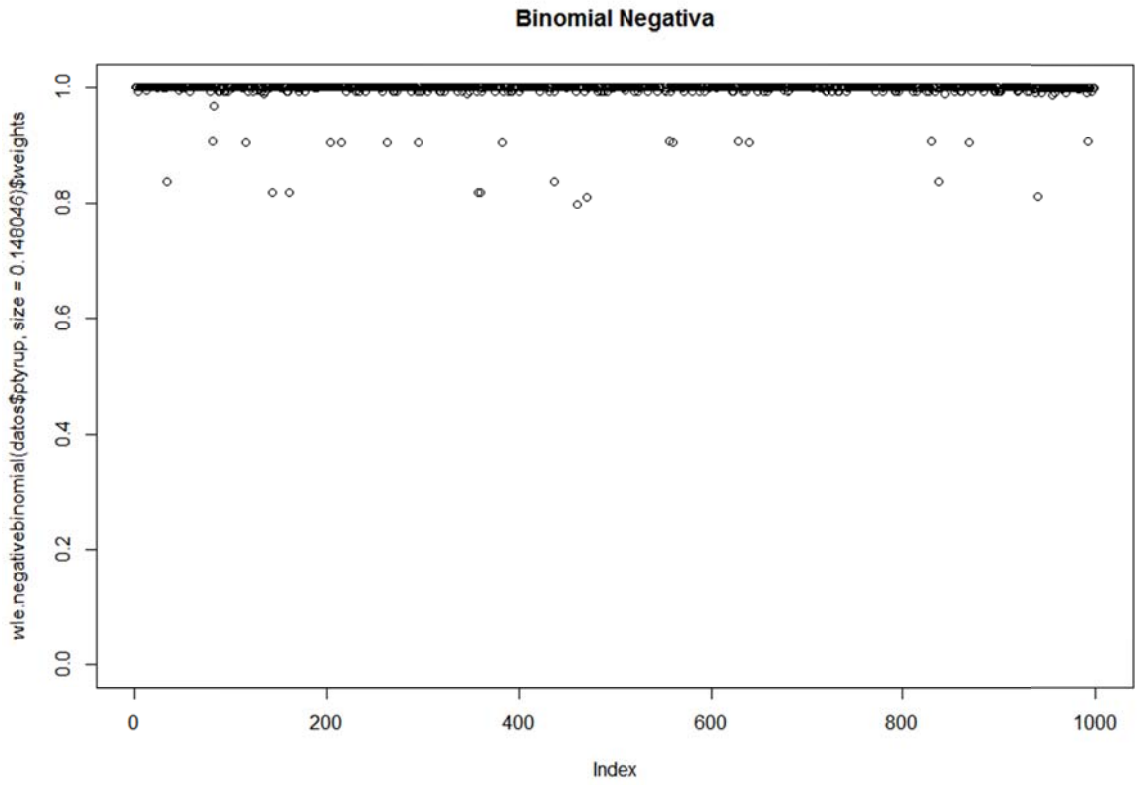

```
hist(datos$ptyrup)
```



Podemos comprobar que esta segunda distribución Binomial Negativa teórica es muy parecida a la obtenida previamente mediante `hist(datos$ptyrup)`.

Valoremos cómo la distribución real de nuestros datos, uno por uno con ellos, se aleja del supuesto teórico de ser una Binomial Negativa "perfecta". Para ello contamos con el comando `wle."distribución que sea"`. Cuanto menores sean los pesos de los puntos (unidades muestrales) menos probable es que ese dato derive de la distribución asumida.

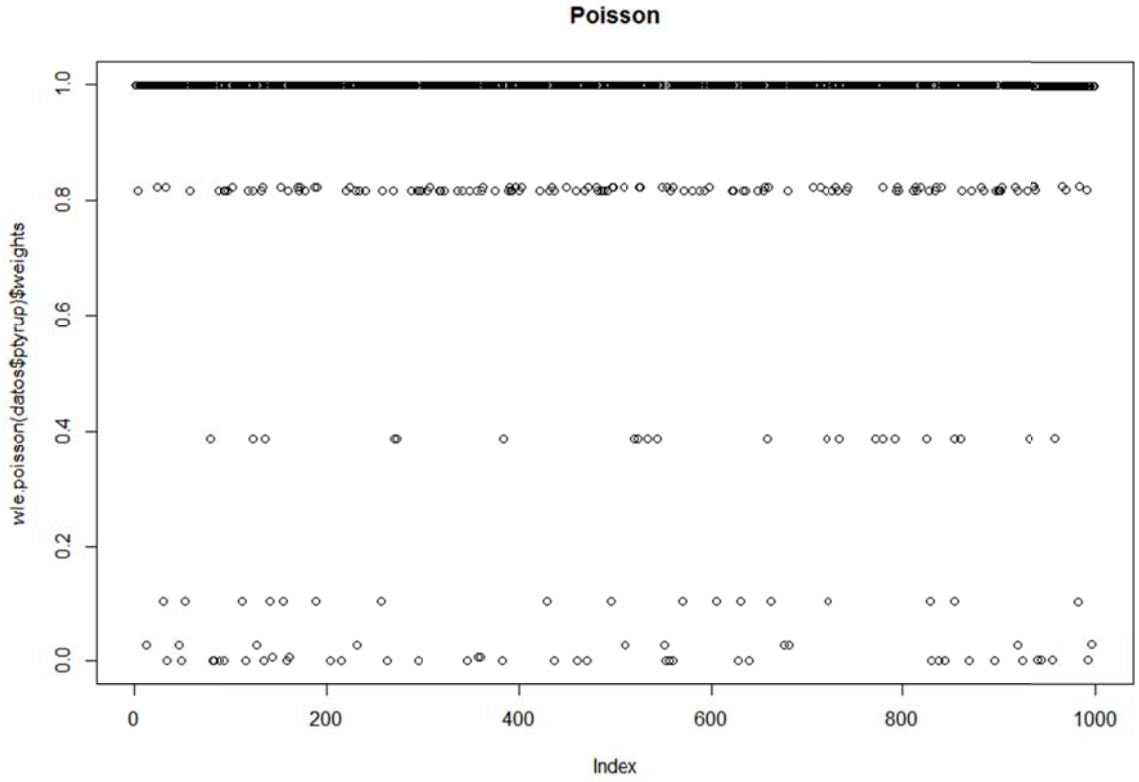
```
plot(wle.negativebinomial(datos$ptyrup, size=0.148046)$weights, ylim=c(0,1) , main="Binomial Negativa")
```



Bueno, esto queda bastante bien.

Sin embargo, si hacemos lo anterior asumiendo una distribución Poisson, tenemos lo siguiente:

```
plot(wle.poisson(datos$ptyrup)$weights, ylim=c(0,1), main="Poisson")
```



¡Bastante peor! ¿Lo veis teniendo en cuenta cuantos datos hay con pesos (*weights* en el eje Y) bajos o nulos?

Definimos ahora la función de variables predictoras que van a explicar la variable respuesta. Utilizaremos, por motivos prácticos de explicación, sólo cinco predictoras continuas que son covariantes. Creamos un texto que lo asignamos a una **fórmula** operativa que denominamos **eqt**.

```
eqt <- as.formula(ptyrup ~ altmed+rangoalt+shannon+tempmin+precip)
```

El objeto `eqt` aparecerá en el panel de RStudio de *Environment*, bajo el epígrafe *values*.

Construimos nuestro **modelo** de Generalizado Lineal asumiendo una distribución de la variable respuesta, y sus errores, Binomial Negativa. Con el comando `glm.nb` del paquete MASS se calcula automáticamente la theta del modelo (size en `rnbinomial`). R trabaja con la varianza definida como mu+[(mu^2)/size].

```
modelo <- glm.nb(eqt, data=datos, link=log)
```

El objeto `modelo` aparecerá en el panel de RStudio de *Environment*, bajo el epígrafe *values*.

El objeto `modelo` tiene contenido, que podemos obtener y conocer mediante:

```
names(modelo)
 [1] "coefficients"      "residuals"         "fitted.values"     "effects"           "R"                 "rank"
 [7] "qr"                "family"            "linear.predictors" "deviance"          "aic"               "null.deviance"
[13] "iter"              "weights"           "prior.weights"     "df.residual"       "df.null"           "y"
[19] "converged"         "boundary"          "terms"             "call"              "model"             "theta"
[25] "SE.theta"          "twologlik"         "xlevels"           "method"            "control"
```

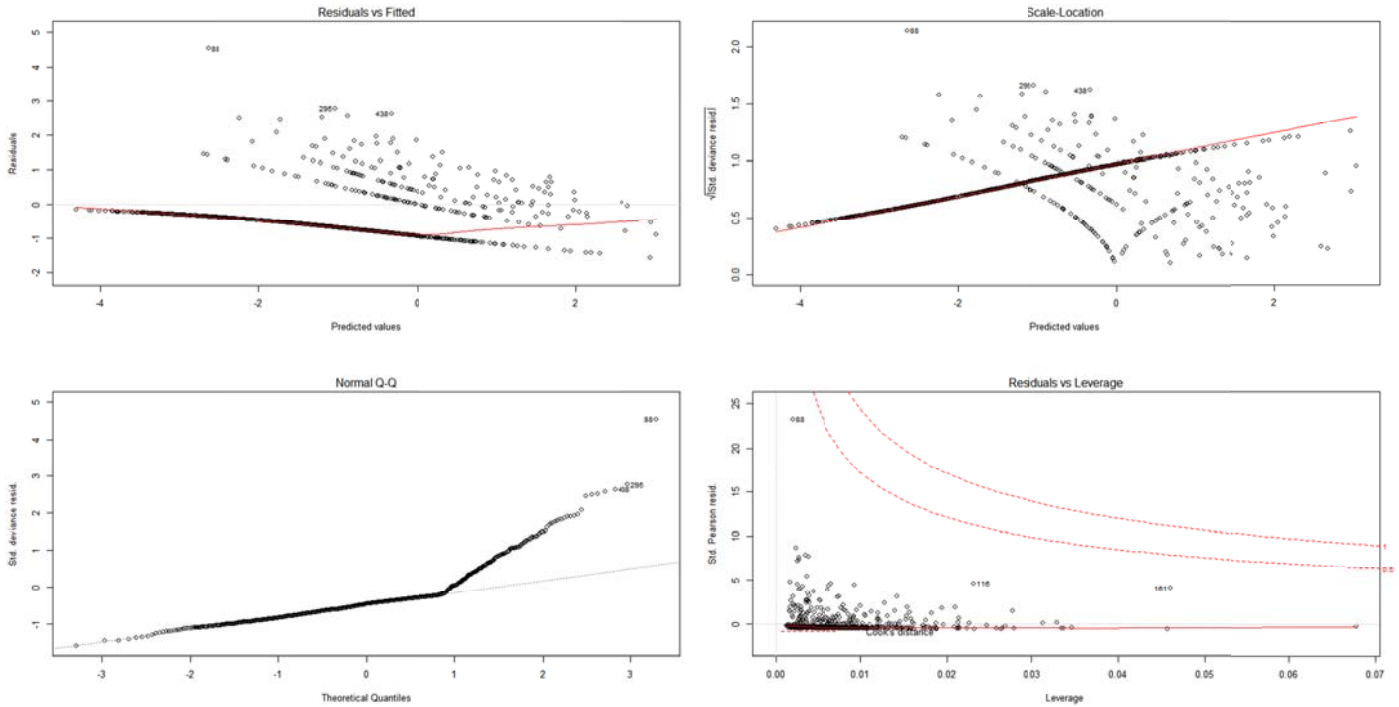
Marco en **rojo** aquellos que más vamos a utilizar.

Revisión de los supuestos canónicos del modelo trabajando con sus RESIDUOS (residuos de devianza).

Los residuos no serán los valores [observados – predichos] en la escala original, sino los [observados – predichos] trabajando con la transformación incluida en la función de vínculo (`link=log`) que linealiza los efectos de la función predictora **g(x)** (establecida al lado derecho de `~` en la ecuación `eqt`).

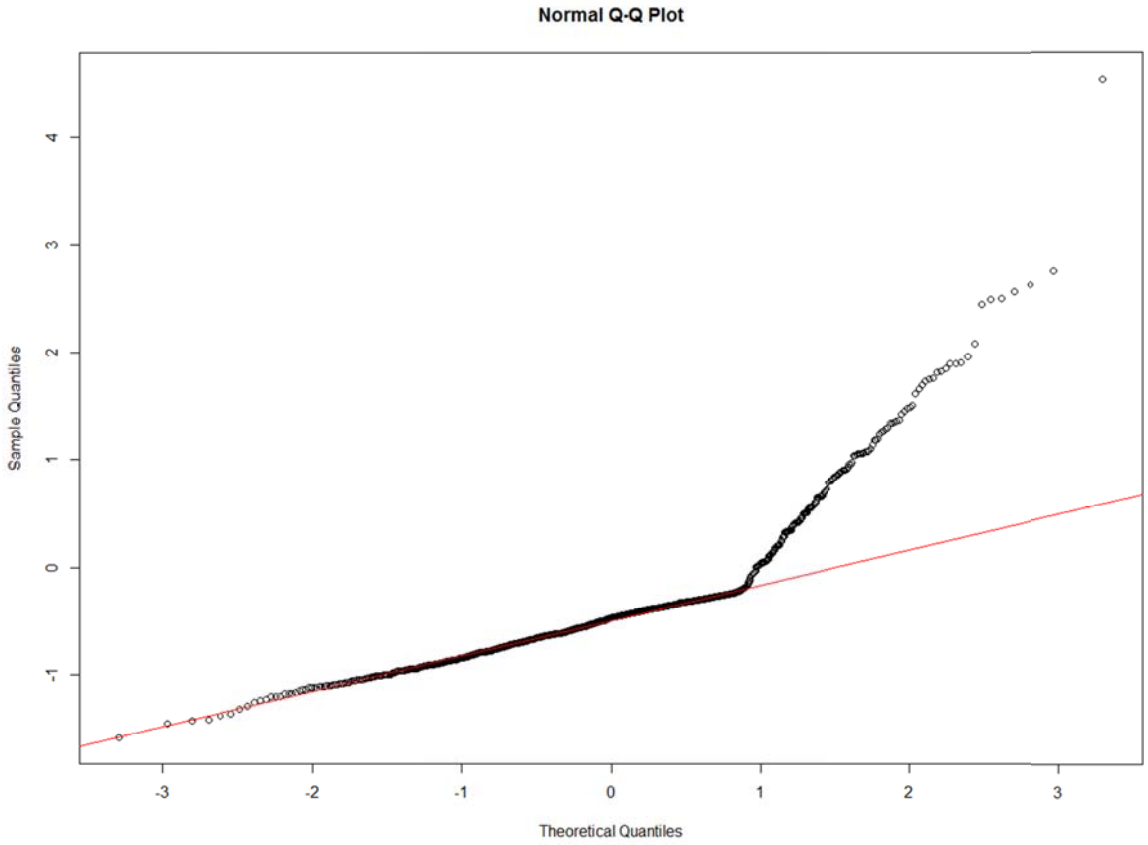
`par(mfcol=c(1,1))` define un panel gráfico de una columna y una fila; `par(mfcol=c(2,2))` define dos paneles gráficos de dos columnas y dos filas.

```
par(mfcol=c(1,1))
par(mfcol=c(2,2))
plot(modelo)
par(mfcol=c(1,1))
```



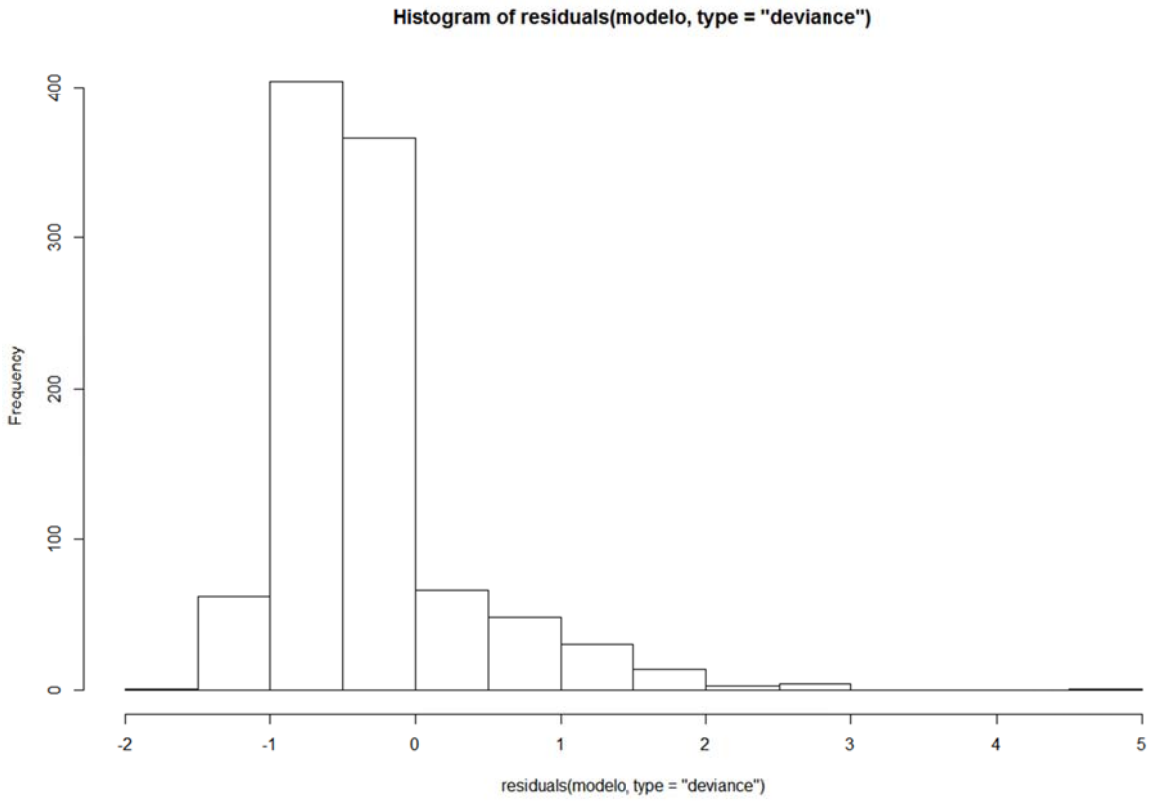
Fijémonos en los dos paneles de la izquierda, que también podemos obtener con estas dos líneas de código:

```
qqnorm(residuals(modelo,type="deviance"))
qqline(residuals(modelo,type="deviance"), col="red")
```



```
hist(residuals(modelo,type="deviance"))
```

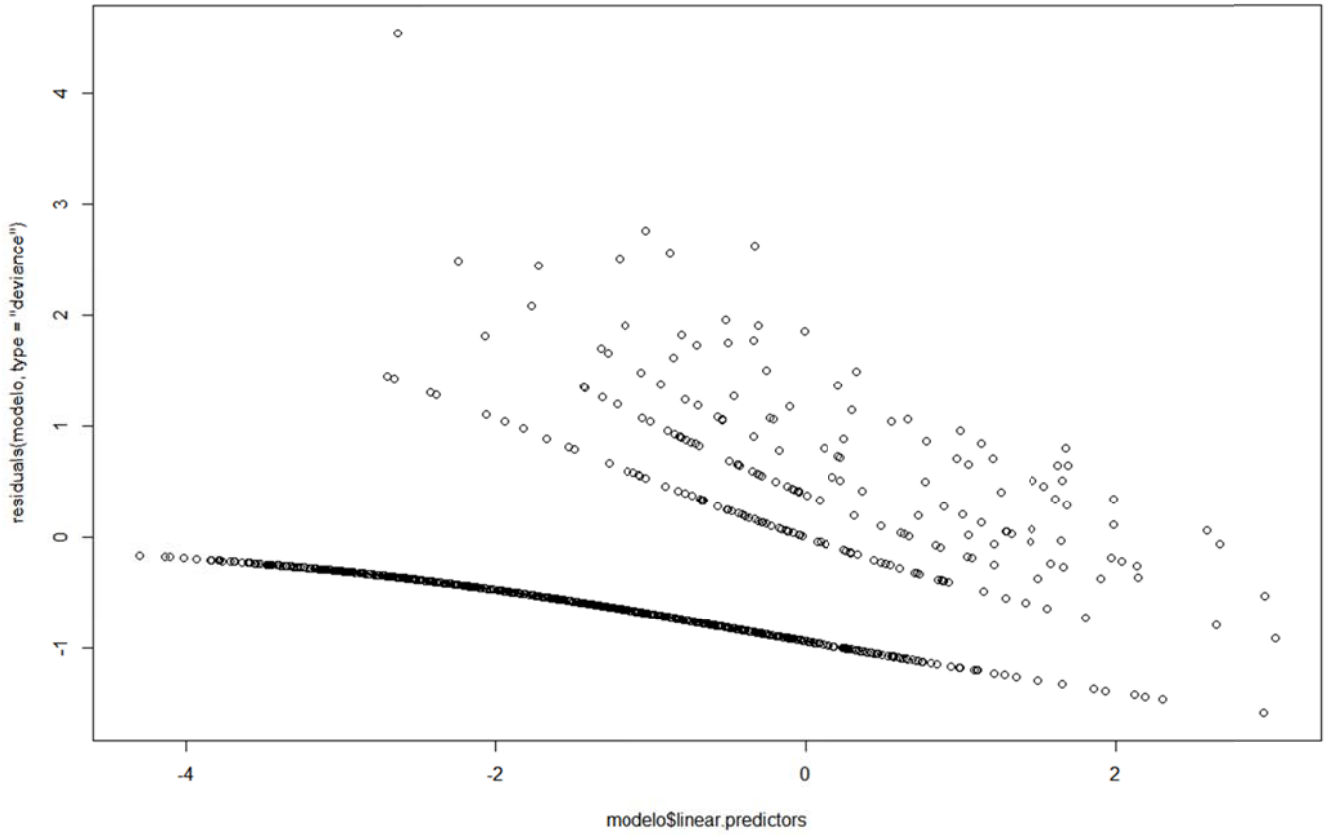
Bueno … esto es un extra.



NORMALIDAD DE LOS RESIDUOS: Los residuos de devianza no están centrados en el cero, hay un predominio de valores residuales negativos, y hay valores extremos positivos. Ya sabemos cómo nuestro modelo se ajusta poco "canónicamente" a los valores de la respuesta (en escala de la *link function* = log).

```
plot(modelo$linear.predictors, residuals(modelo, type="deviance"))
```

Esto es para ver “los cielos estrellados de una noche de verano” o las “bolas dispersas al azar en una mesa de billar”. Los residuos del modelo (en devianza) se deben dispersar al azar a lo largo de las predicciones del modelo en la escala definida por la *link function* (en logaritmo). Para ello utilizamos `modelo$linear.predictors` en vez de `modelo$fitted.values` (que son $e^{g(x)}$).



Mala pinta. Clara prueba de HETEROCEDASTICIDAD en el modelo. Habrá que corregir este problema (estimadores *sandwich*).

La devianza es una medida de la variación residual de un modelo. Mide cuánto se dispersa el valor de la variable respuesta de cada unidad muestral respecto a las predicciones del modelo trabajando con la transformación introducida por la función de vínculo *link function* (en esta ocasión `link=log`).

Calculando el sumatorio de los cuadrados de los valores de devianza de cada unidad muestral se obtiene la devianza residual del modelo.

$$\text{devianza residual del modelo} = \Sigma\ \text{devianza}_i^2$$

para las i unidades muestrales de un total de N (i.e., tamaño muestral).

Veamos el valor de la devianza residual del modelo que está "guardada" en el objeto `modelo`.

```
modelo$deviance
[1] 529.1295
```

Calculemos ahora la devianza residual del modelo como la suma de los cuadrados de las devianzas individuales de cada unidad muestral. Estos valores los obtenemos con el comando `residuals(…)` aplicado al modelo:

```
sum(residuals(modelo, type="deviance")^2)
[1] 529.1295
```

Otro tipo de devianza residual es la de Pearson que se obtiene definiendo el argumento `type="pearson"`.

Esta devianza de Pearson la vamos a utilizar a la hora de estimar el coeficiente de sobredispersión **φ** en los modelos Poisson y Binomiales. Esta medida de devianza nos será de utilidad para convertir una diferencia de devianzas en la F de Fisher cuando corri jamos la sobredispersión de los modelos (lo veremos más adelante con distribuciones GLM efectuados con binomiales).

Si queremos hacer una aproximación más “cuantitativa” y académica al análisis de los residuos podemos estimar su Sesgo y Kurtosis.

```
skewness(residuals(modelo,type="deviance"))          ## sesgo, Ho = 0
[1] 2.180688

agostino.test(residuals(modelo,type="deviance"))     ## D'Agostino test for skewness in normally distributed data
        D'Agostino skewness test
data:  residuals(modelo, type = "deviance")
skew = 2.1807, z = 18.2283, p-value < 2.2e-16
alternative hypothesis: data have a skewness


kurtosis(residuals(modelo,type="deviance"))          ## kurtosis de Pearson, Ho (hipótesis nula) = 3
[1] 10.22145

anscombe.test(residuals(modelo,type="deviance"))     ## Anscombe-Glynn test of kurtosis for normal samples
        Anscombe-Glynn kurtosis test
data:  residuals(modelo, type = "deviance")
kurt = 10.2215, z = 12.3509, p-value < 2.2e-16
alternative hypothesis: kurtosis is not equal to 3
```

La distribución de los residuos está muy sesgada (valor positivo y sesgo por la derecha) y muestra una elevada kurtosis (valor positivo > 3, leptokurtosis o aguzamiento).

El sesgo no altera sustancialmente los valores alfa críticos de significación (e.g., α = 0.05).

Pero las desviaciones de normalidad por kurtosis inflan la significación (más significativos de lo que deberían ser, aumento del Error de tipo I: rechazar la hipótesis nula Ho cuando, de hecho, es cierta) si existe leptokurtosis (kurtosis de Pearson > 3), y al contrario si hay platikurtosis (i.e., aplastamiento mayor que el esperable por normalidad, kurtosis de Pearson < 3, incremento del Error del tipo II: aceptar la hipótesis nula cuando, de hecho, es falsa).

También podemos efectuar un test global de desvío de la normalidad en los residuos de devianza del modelo:

```
shapiro.test(residuals(modelo,type="deviance"))
```

```
        Shapiro-Wilk normality test
data:  residuals(modelo, type = "deviance")
W = 0.808, p-value < 2.2e-16
```

Pues nada, lo ya visto con el qq-plot, el histograma y los exámenes paramétricos de los residuos de devianza utilizando el sesgo y la kurtosis.

En numerosas ocasiones, sólo valorando los *normal probability plots* con `qqnorm(residuals(modelo,type="deviance"))` podremos saber (con buena experiencia acumulada) si nos estamos desviando poco o mucho de los supuestos canónicos de normalidad.

Insistimos: los residuos de devianza del modelo se deben dispersar simétricamente a lo largo de las predicciones del modelo que "funciona" en la escala de medida definida por la *link function*. La densidad de valores residuales debe ser máxima cerca de las predicciones, y debe disminuir de **modo simétrico** al alejarnos de ellas. ¿Y qué es esto que acabamos de describir?

LA DISTRIBUCIÓN NORMAL DE LOS RESIDUOS (de devianza).

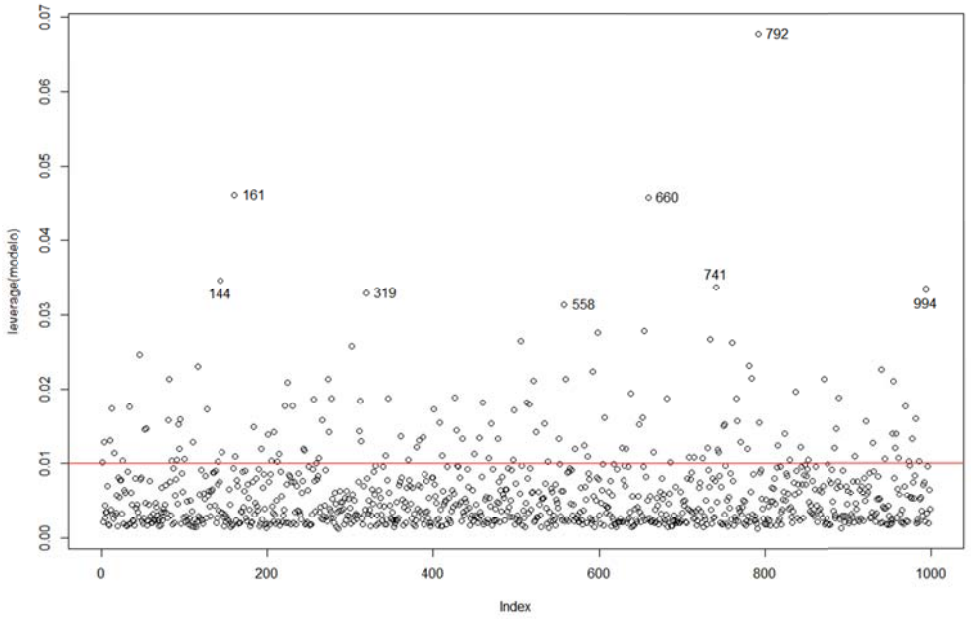
SINTETIZANDO: no tenemos normalidad de los residuos, probablemente estemos inflando el error de Tipo I (los resultados serán más significativos de lo que realmente son) y hay clara heterocedasticidad, con lo cual las estimas de significación van a estar alteradas.

… CAVEAT EMPTOR.

Ahora efectuamos una exploración de los valores residuales de manera individualizada (dato a dato), para saber en qué medida son importantes los valores perdidos e influyentes en nuestros datos.

Empezamos con el ***leverage*** (cómo de extremos son los datos, en función de los valores que toman en las variables predictoras). http://en.wikipedia.org/wiki/Leverage_(statistics). El nivel crítico "aproximado" es 2 * (g.l. del modelo) / (Número de datos)

```
plot(leverage(modelo)); abline(h=2*5/999, col="red")
identify(leverage(modelo))          ## marcamos con el ratón los datos extremos y luego clic en el botón Finish
                                       del panel Plots (esquina superior derecha del panel
```

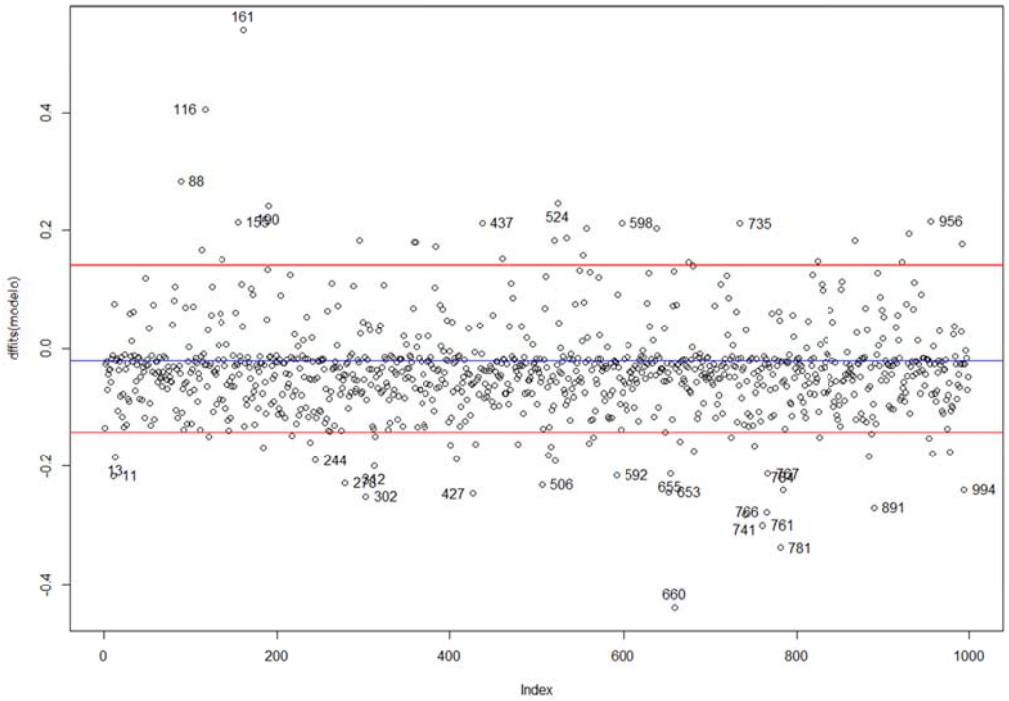


Sólo se marcan "subjetivamente", por ser los datos más extremos, aquellos cuyos valores de leverage son más de tres veces el valor crítico umbral.

En las tres gráficas siguientes, "Index" denota el número de orden (i.e., número de fila) de cada unidad muestral en nuestra matriz de **datos** (*data frame* de trabajo).

Seguimos con los **dffits** (http://en.wikipedia.org/wiki/DFFITS): cómo cambian los residuos de cada observación dependiendo de si ese dato **lo-incluimos-o-no** en el modelo. El nivel crítico "aproximado" es + / - (2 * raíz [(g.l. del modelo) / (Número de datos)])

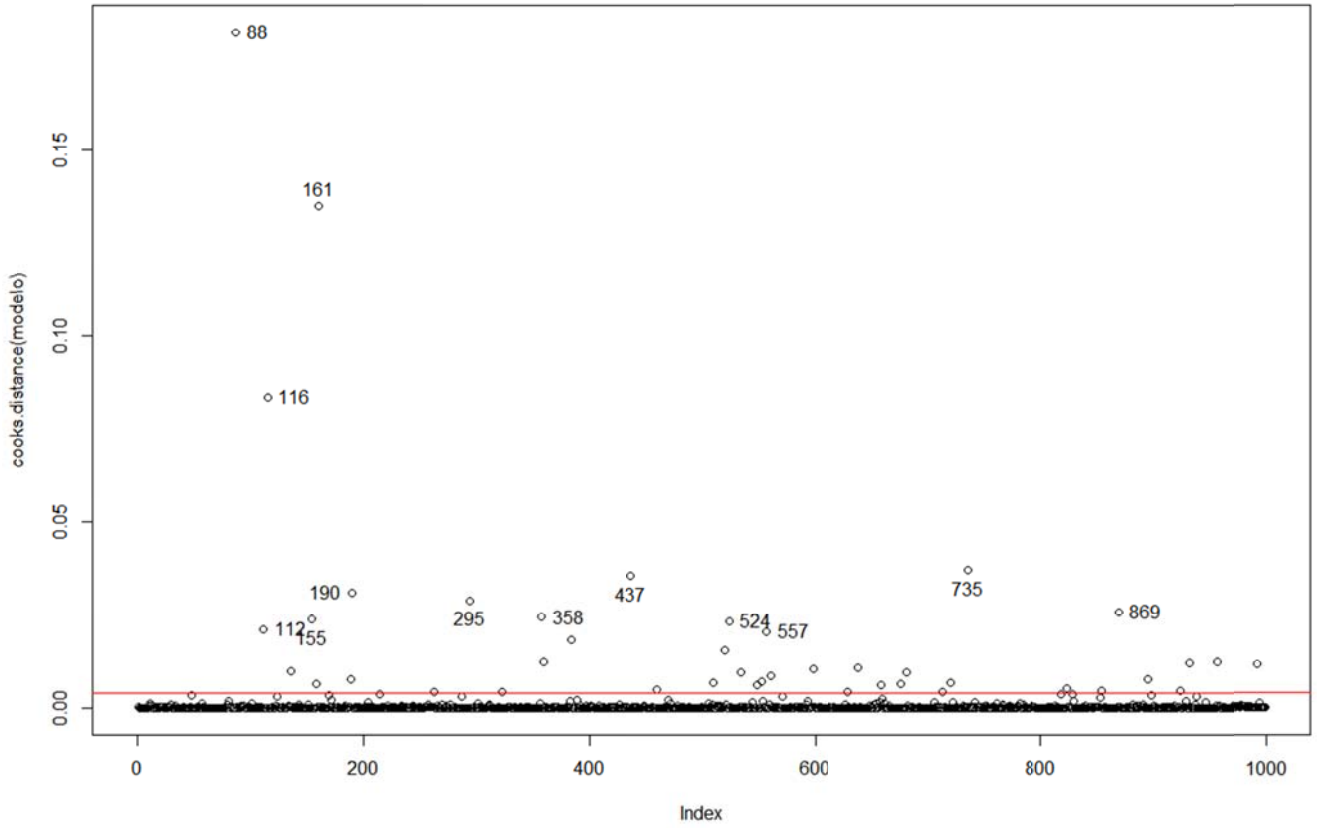
```
plot(dffits(modelo)); abline(h=(2*(5/999)^0.5), col="red"); abline(h=-(2*(5/999)^0.5), col="red"); abline(h=0, col="blue")
identify(dffits(modelo))          ## marcamos con el ratón los datos extremos y luego clic en el botón Finish
```



Fijaos que los **dffits** no están centrados en el CERO, sino por -0.02 (línea azul horizontal). De aquí, el predominio de valores residuales negativos en el histograma previo con `hist(residuals(modelo,type="deviance"))`. También identificamos puntos que denominamos "*outliers*".

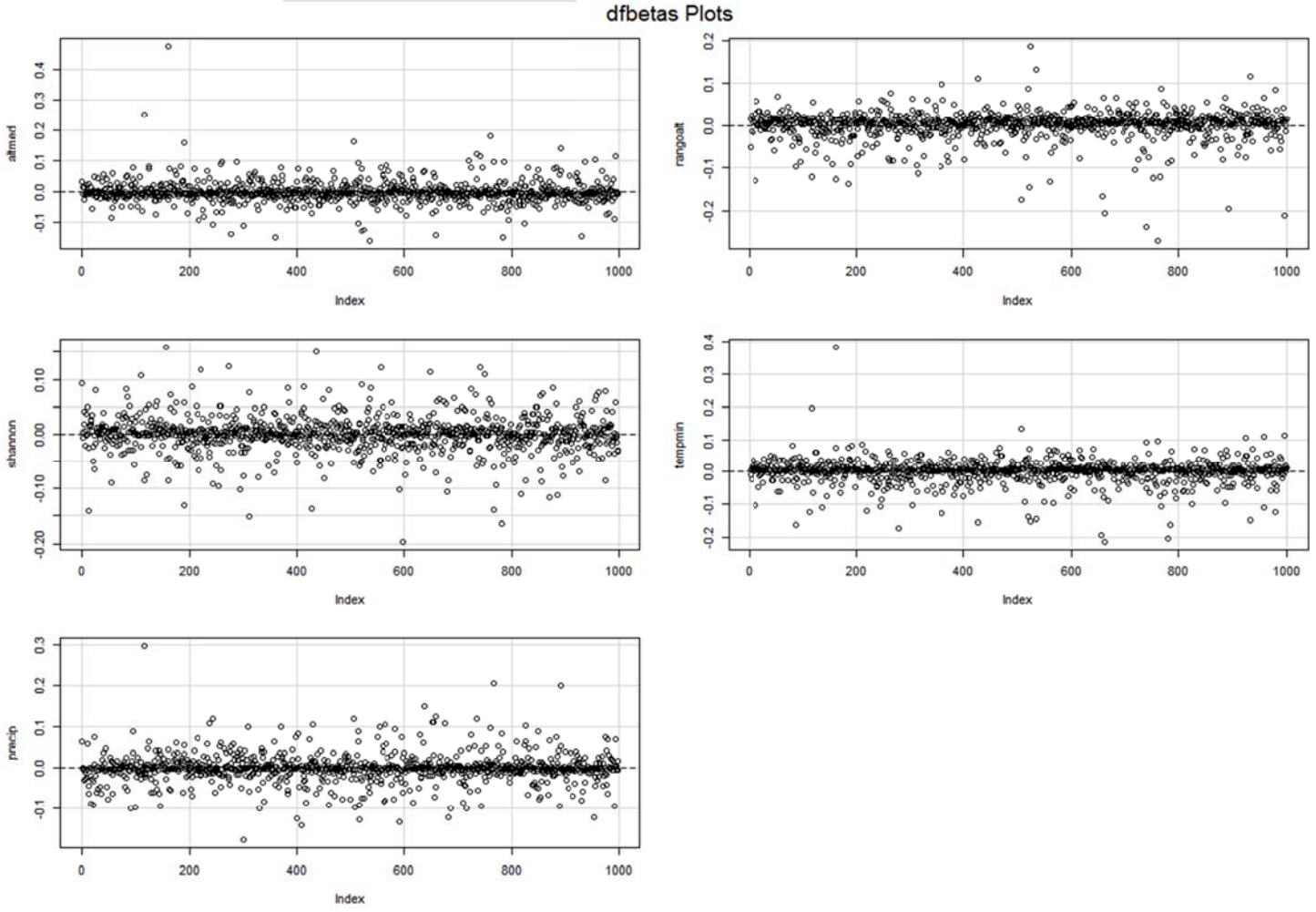
Por último, la **distancia de Cook** (http://en.wikipedia.org/wiki/Cook%27s_distance) combina estos dos aspectos en una sola estima. Un valor "crítico" aproximado sería 4/(número de datos) (i.e., que los valores sean menores que esta cantidad umbral).

```
plot(cooks.distance(modelo)); abline(h=4/999, col="red")
identify(cooks.distance(modelo))
```



¿Y cómo se alterarían los coeficientes de regresión dependiendo de que construyamos el modelo con-y-sin cada uno de los datos?

Veamos las **dfbetas**: `dfbetasPlots(modelo)`



Los valores de **dfbeta** nos indican cómo de influyente es un dato alterando los coeficientes de regresión del modelo.

Nos fijamos en el hecho de que la escala de las Y's no es comparable por su diferente rango de variación. Pero los valores de las **dfbetas** de cada variable identificada en el eje Y SÍ son comparables (ya que se trabaja con coeficientes z-estandarizados).

Vemos "problemas" con algunos datos (pocos, ¡pero existentes!) en las predictoras "altmed", "tempmin" y "precip".

Si no tenemos criterios para quitar esos puntos influyentes y/o perdidos (e.g., por ser datos erróneos a la hora de introducirlos en el *data frame*, por tener problemas de precisión-exactitud durante los muestreos, etc) deberemos tender a efectuar **estimas robustas de nuestro modelo**. Y esto, tanto más cuanto más puntos influyentes-perdidos tengamos en relación a nuestro tamaño muestral total.

¿Y cómo de **independientes entre sí** son nuestras **variables predictoras**?

En teoría … mejor cuanto más independientes sean. Este fenómeno lo parametrizamos mediante el concepto **VIF** (*variance inflation factor*). http://en.wikipedia.org/wiki/Variance_inflation_factor.

VIF = 1 / (1 - R^2), donde R^2 es lo explicado de cada predictora por todas las restantes. NO INCLUIMOS AQUÍ A LA RESPUESTA.

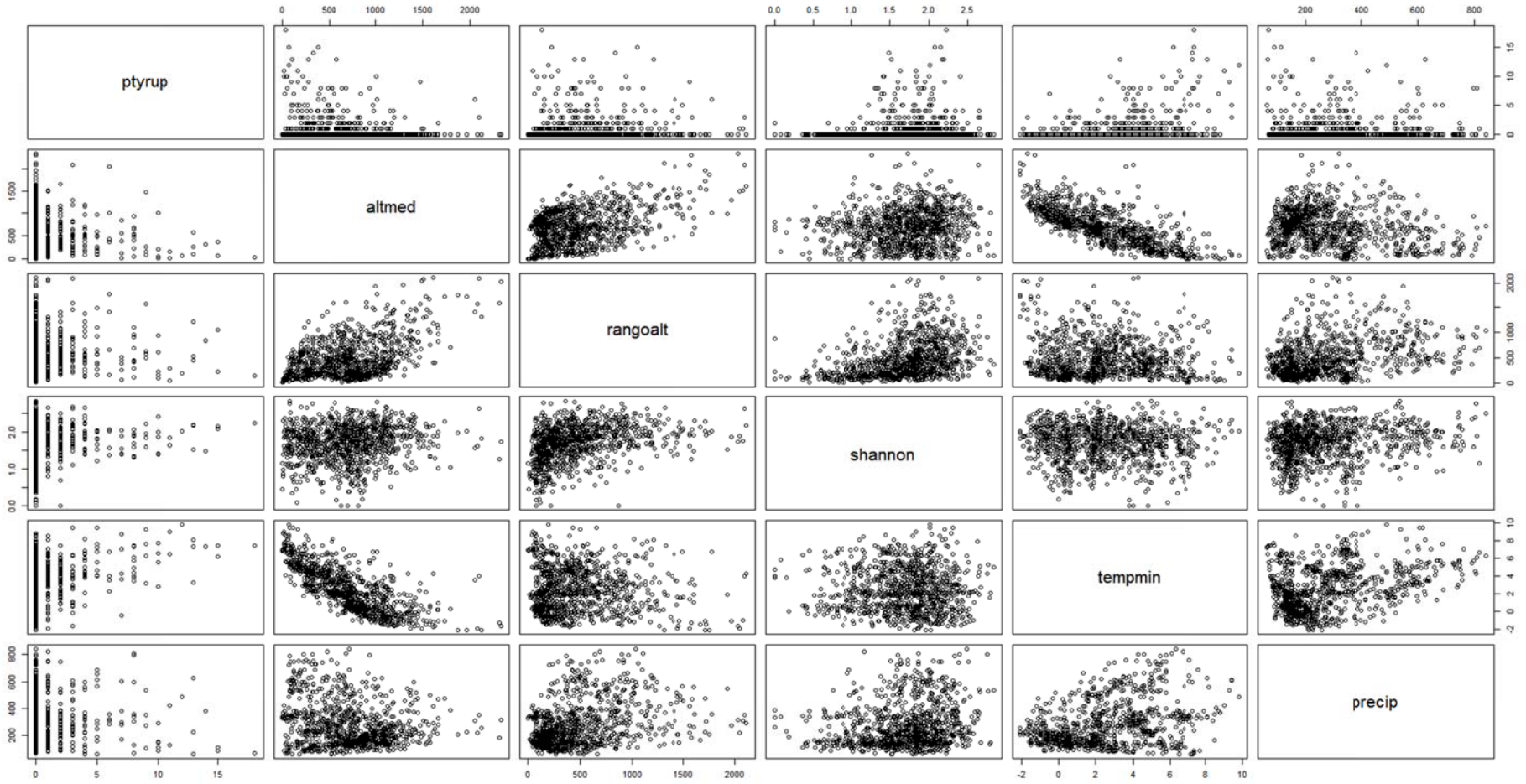
La raíz cuadrada de VIF indica aproximadamente cuántas veces está aumentado el error estándar de una variable predictora, debido a su no independencia (o existencia de colinealidad) con las variables predictoras restantes.

```
vif(modelo)
  altmed rangoalt  shannon  tempmin   precip
3.789212 2.303854 1.139364 2.477226 1.337875

sqrt(3.789212)      ## esas veces se infla el error standard de esa predictora: "altmed"
[1] 1.94659
```

Por tanto, si el error estándar aumenta su amplitud, la existencia de colinealidad va a tender a reducir su significación en el modelo.

Para ver cómo de relacionadas están entre sí las variables predictoras (y la respuesta, en la primera fila):

```
pairs(eqt, data=datos)
```



¡Pues esto es lo que hay! Y con estos mimbres hacemos un cesto … pero a sabiendas de que tenemos algunos problemas de los que protegernos.

Primeramente efectuamos un test de significación del modelo. Esto sería equivalente a un **ómnibus test**, mediante el cual si el modelo global no es significativo (el "todo"), no miramos la significación de sus partes (los efectos de cada predictor o variable explicativa. http://en.wikipedia.org/wiki/Omnibus_test

Para ello podemos contar con el **LRTest** o *Likelihhod ratio test*, que no es más que una diferencia de devianzas residuales del modelo nulo (`ptyrup ~ 1`; aquel que no introduce ninguna predictora; sólo la ordenada en el origen que coincide con la media de la respuesta) y la del modelo de interés (`ptyrup ~ altmed + rangoalt + shannon + tempmin + precip`).

```
lrtest(modelo)
Likelihood ratio test
Model 1: ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
Model 2: ptyrup ~ 1
  #Df  LogLik Df  Chisq Pr(>Chisq)
1   7 -837.58
2   2 -947.18 -5 219.19  < 2.2e-16 ***                ## recordemos que devianza = -2*LogLik; ¡¡haced las cuentas!!
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

También podemos contar con un test equivalente: el **test de Wald**, que opera con una F en vez de con una Chi^2.

```
waldtest(modelo)
Wald test
Model 1: ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
Model 2: ptyrup ~ 1
  Res.Df Df     F    Pr(>F)
1    993
2    998 -5 44.55 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

EN RESUMEN: el modelo es altamente significativo.

Bueno pero … teníamos problema de **heterocedasticidad** en los valores residuales (de devianza) del modelo.

Corrijamos, pués, por ese desvío del supuesto canónico. Recordemos lo que salía en:

```
plot(modelo$linear.predictors, residuals(modelo, type="deviance"))
```

Para ello hacemos uso de una **corrección *sandwich*** a la estima de significación del modelo (que contempla el *bread* & *meat*; consultad las páginas 14, 2 y 8 de http://cran.r-project.org/web/packages/sandwich/sandwich.pdf). Algo muy parecido obtendríamos al considerar sólo una parte del "bocata" (*meat*), haciendo uso de **vcovHC** (para más detalles de este aspecto consultad: http://www.inside-r.org/packages/cran/sandwich/docs/vcovHC).

```
waldtest(modelo, vcov=sandwich)
Wald test
Model 1: ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
Model 2: ptyrup ~ 1
  Res.Df Df     F    Pr(>F)
1    993
2    998 -5 41.92 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
waldtest(modelo, vcov=vcovHC)
Wald test
Model 1: ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
Model 2: ptyrup ~ 1
  Res.Df Df      F    Pr(>F)
1    993
2    998 -5 41.505 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

¡Vale!, teniendo en cuenta el desvío de homocedasticidad … ¡¡NUESTRO MODELO SIGUE SIENDO MUUUY SIGNIFICATIVO!!

¿Y cómo de verosímil es mi modelo representando la realidad contenida en los datos respecto a un modelo nulo que no incluye efectos? Construimos un modelo nulo de nuestra respuesta sobre la marcha:

```
modelo.nulo <- glm.nb(ptyrup ~ 1, data=datos, link=log)
```

Ahora estimamos los valores de Akaike (basados en la teoría de la información) para los dos modelos. Pero ojo, corregimos teniendo en cuenta el tamaño muestral (aunque en este caso, como N=999 es enorme, no haría falta; mirad la parte derecha):

```
AICc(modelo.nulo, modelo)
            df     AICc
modelo.nulo  2 1898.370
modelo       7 1689.279
```

```
AIC(modelo.nulo, modelo)
            df      AIC
modelo.nulo  2 1898.358
modelo       7 1689.166
```

Con los valores **AICc** podemos calcular cuántas veces es mejor el modelo de interés que el modelo nulo. **A menor valor, ¡mejor**!

Para ello trabajamos con AICc. Comenzamos haciendo la diferencia entre los dos modelos (mejor - peor). Luego negativizamos esa diferencia y la dividimos entre dos. Y finalmente calculamos el antilogaritmo de ese valor:

```
exp(-(AICc(modelo)-AICc(modelo.nulo))/2)
[1] 2.532466e+45
```

Nuestro modelo de interés es millones de millones de millones … de veces mejor que el modelo nulo.

¿Cuánto explica mi modelo de la variación observada en mi variable respuesta `datos$ptyrup` ?

Pues es fácil: (DEVIANZA RESIDUAL$_{\text{modelo nulo}}$ - DEVIANZA RESIDUA$_{\text{mi modelo}}$) / DEVIANZA RESIDUAL$_{\text{modelo nulo}}$

```
## DEVIANZA EXPLICADA
d2 <- round((modelo$null.deviance-modelo$deviance)/modelo$null.deviance, 4)
print(c("D2 de MCFadden (%) =", 100*d2), quote=FALSE)                  ## en tanto por cien: 100*
[1] D2 de MCFadden (%) = 36.01
```

Ahora podemos pasar a la estima de la parametrización de los efectos y la estima de sus significaciones.

Para ello podemos hacer uso de dos comandos que dan parecidos resultados. Veamos uno a uno.

Estima simultánea, con estima asintótica de los errores estándar:

```
summary(modelo)              ## perdemos un grado de libertad más por estimar Theta.

Call:
glm.nb(formula = eqt, data = datos, link = log, init.theta = 0.2974642714)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5784  -0.7180  -0.4646  -0.2735   4.5393

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.7169395  0.4904432  -5.540 3.03e-08 ***
altmed      -0.0012181  0.0004057  -3.002  0.00268 **
rangoalt     0.0021294  0.0003095   6.880 5.97e-12 ***
shannon      0.6068703  0.1951244   3.110  0.00187 **
tempmin      0.4000696  0.0536370   7.459 8.73e-14 ***
precip      -0.0022501  0.0005417  -4.154 3.27e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for Negative Binomial(0.2975) family taken to be 1)

    Null deviance: 826.86  on 998  degrees of freedom
Residual deviance: 529.13  on 993  degrees of freedom
AIC: 1689.2

Number of Fisher Scoring iterations: 1
              Theta:  0.2975
          Std. Err.:  0.0346
2 x log-likelihood:  -1675.1660
```

Recodemos que anteriormente Theta = size nos daba `0.1480468` al trabajar sólo con la respuesta `datos$ptyrup`. ¡Parecido!

Estima comparada. Se calcula la contribución de un efecto al modelo como la diferencia en las devianzas residuales de los modelos que no quitan esa variable (`<none>`) y aquel que sí la quita (e.g., `altmed`).

```
dropterm(modelo, test="Chisq", sorted=FALSE)    ## sorted=TRUE ordena los resultados de menor a mayor AIC
Single term deletions
Model:
ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
         Df    AIC    LRT   Pr(Chi)
<none>      1687.2
altmed    1 1695.8 10.630  0.001113 **
rangoalt  1 1728.5 43.389 4.488e-11 ***
shannon   1 1693.9  8.694  0.003192 **
tempmin   1 1747.1 61.896 3.621e-15 ***
precip    1 1700.5 15.379 8.798e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Esta aproximación es **más aconsejable** si tenemos desvíos de los supuestos canónicos del modelo y poco tamaño muestral.

También podemos re-estimar la significación de los coeficientes de regresión teniendo en cuenta la **heterocedasticidad** detectada en los residuos del modelo. Aquí abajo comparamos las estimas con y sin control de la heterocedasticidad. Los coeficientes de regresión no cambian; sólo cambian las estimas de los errores estándar y la significación.
`coeftest(modelo)` da lo mismo que `summary(modelo)` (valoradlo comparativamente con los resultados previos)

```
coeftest(modelo, vcov=sandwich)
z test of coefficients:
               Estimate  Std. Error z value  Pr(>|z|)
(Intercept) -2.71693949  0.51624780 -5.2629 1.418e-07 ***
altmed      -0.00121811  0.00050545 -2.4099  0.015955 *
rangoalt     0.00212936  0.00028452  7.4841 7.203e-14 ***
shannon      0.60687032  0.18965050  3.1999  0.001375 **
tempmin      0.40006959  0.06725394  5.9486 2.704e-09 ***
precip      -0.00225015  0.00049882 -4.5110 6.453e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
coeftest(modelo)
z test of coefficients:
               Estimate  Std. Error z value  Pr(>|z|)
(Intercept) -2.71693949  0.49044317 -5.5398 3.029e-08 ***
altmed      -0.00121811  0.00040574 -3.0022  0.002681 **
rangoalt     0.00212936  0.00030948  6.8804 5.968e-12 ***
shannon      0.60687032  0.19512439  3.1102  0.001870 **
tempmin      0.40006959  0.05363695  7.4588 8.729e-14 ***
precip      -0.00225015  0.00054174 -4.1536 3.273e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

El desvío del supuesto canónico de la homocedasticidad de los residuos altera muy poco las significaciones de los efectos parciales.

Interpretación de los coeficientes de regresión. Debemos considerar que estamos ante modelos multiplicativos, porque la función de vínculo es el logaritmo (`family=negbin(link="log")`). En la regresión de Poisson y Binomial Negativa:

$$\log(Y) = a + b \cdot X$$

Por lo que el log(Y) cambia linealmente en función de las variables predictoras (usando log en base e, o *ln*).

El coeficiente **b** es el cambio esperado en el log(Y) cuando la variable predictora aumenta en una unidad de medida (sean metros de altitud, mm de precipitación, grados de temperatura, etc). Consideremos nuestra tabla de coeficientes:

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.7169395  0.4904432  -5.540 3.03e-08 ***
altmed      -0.0012181  0.0004057  -3.002  0.00268 **
rangoalt     0.0021294  0.0003095   6.880 5.97e-12 ***
shannon      0.6068703  0.1951244   3.110  0.00187 **
tempmin      0.4000696  0.0536370   7.459 8.73e-14 ***
precip      -0.0022501  0.0005417  -4.154 3.27e-05 ***
```

Para el intercepto (`Intercept`): Cuando todas las variables predictoras tienen el valor CERO, entonces log(Y) = `-2.7169`. Por lo que **Y** = exp(`-2.7169`) = 0.0661; o sea, "casi" no aparecen los aviones roqueros (nuestra variable respuesta).

Considerando `shannon 0.6069`, podemos decir que cuando la diversidad de paisajes en 100 km^2 aumenta una unidad, a todo lo demás constante, el logaritmo de la abundancia relativa del avión roquero aumenta en 0.6069 unidades: exp(0.6069) = 1.8 presencias en transectos de 15 minutos de duración (`shannon` se mide en nats, utilizando *ln* para calcularla).

Considerando `tempmin 0.4001`, podemos decir que cuando la temperatura mínima media invernal aumenta 1ºC, a todo lo demás constante, entonces el logaritmo de la abundancia relativa del avión roquero aumenta en 0.4001 unidades: exp(0.4001) = 1.5 presencias en transectos de 15 minutos de duración. Eso es, si `tempmin` aumentase 10 ºC, el avión roquero aumentaría en 15 presencias.

Hacedlo para la altitud media (`altmed`), pero como su coeficiente es muy pequeño, estimadlo para un aumento de 100 m en altitud. Considerad que el coeficiente tiene valor negativo, por lo que la respuesta "disminuye" o "aumenta negativamente".

Coeficientes de regresión beta (β). Los coeficientes de regresión del modelo no son comparables entre sí, ya que las variables predictoras están medidas en distintas unidades (e.g., diversidad en nats, altitud en metros, precipitación en mm, temperatura en ºC, etc). Por tanto, si queremos que la comparación de los coeficientes de regresión nos proporcione una **medida de la magnitud de su efecto**, tenemos que "llevar" las variables predictoras a una misma escala de medida.

Esto lo podemos conseguir mediante la zeta-estandarización de las variables predictoras (las lleva a media = 0 y sd = 1):

$$X_i' = (X_i - media_X) / sd_X$$

Para ello utilizamos el comando `scale(variable predictora)`, creando nuevas variables z-estandarizadas o incluyendo el comando `scale` dentro de una nueva ecuación del modelo. ¡Mejor esta segunda opción! Y finalmente creamos un segundo modelo con esta nueva ecuación, pero manteniendo sus otras características:

```
eqt2 <- as.formula(ptyrup ~ I(scale(altmed))+I(scale(rangoalt))+I(scale(shannon))+I(scale(tempmin))+I(scale(precip)))
modelo2 <- glm.nb(eqt2, data=datos, link=log)
```

Y los nuevos resultados son:

```
coeftest(modelo2, vcov=sandwich)
```

```
z test of coefficients:

                 Estimate Std. Error z value  Pr(>|z|)
(Intercept)      -1.171501   0.123560 -9.4812 < 2.2e-16 ***
I(scale(altmed))  -0.479621   0.199017 -2.4099  0.015955 *
I(scale(rangoalt))  0.801334   0.107071  7.4841 7.203e-14 ***
I(scale(shannon))   0.291670   0.091148  3.1999  0.001375 **
I(scale(tempmin))   0.983901   0.165399  5.9486 2.704e-09 ***
I(scale(precip))  -0.362099   0.080271 -4.5110 6.453e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Ahora los nuevos coeficientes son perfectamente comparables entre sí, estableciendo una medida cuantitativa de la importancia de su efecto explicando la variación observada en la variable respuesta.

tempmin > rangoalt >> altmed > precip > shannon

Son los mismos resultados de significación para las predictoras que hemos obtenido previamente haciendo uso de `coeftest(modelo2, vcov=sandwich)` (¡ no para el intercepto que cambia ! … por motivos obvios).

Una aproximación a la partición de la variabilidad observada en la variable respuesta la podemos obtener mediante el uso de `dropterm(…)` y `lrtest(modelo)`, haciendo uso de los **valores de diferencias de devianzas** (LRT en `dropterm(…)` y Chisq en `lrtest(modelo)`).

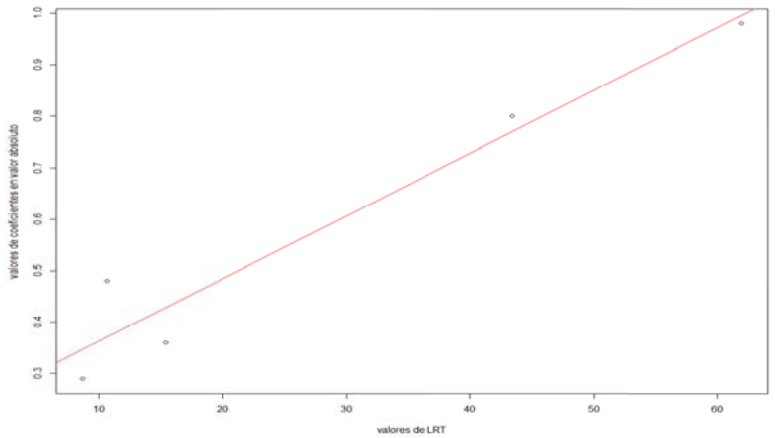
```
dropterm(modelo, test="Chisq", sorted=FALSE)
Single term deletions
Model:
ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
          Df    AIC    LRT   Pr(Chi)
<none>       1687.2
altmed     1 1695.8 10.630  0.001113 **
rangoalt   1 1728.5 43.389 4.488e-11 ***
shannon    1 1693.9  8.694  0.003192 **
tempmin    1 1747.1 61.896 3.621e-15 ***
precip     1 1700.5 15.379 8.798e-05 ***
```

Recordemos lo obtenido previamente

```
coeftest(modelo2, vcov=sandwich)
z test of coefficients:
                    Estimate Std. Error z value  Pr(>|z|)
(Intercept)        -1.171501   0.123560 -9.4812 < 2.2e-16 ***
I(scale(altmed))   -0.479621   0.199017 -2.4099  0.015955 *
I(scale(rangoalt))  0.801334   0.107071  7.4841 7.203e-14 ***
I(scale(shannon))   0.291670   0.091148  3.1999  0.001375 **
I(scale(tempmin))   0.983901   0.165399  5.9486 2.704e-09 ***
I(scale(precip))   -0.362099   0.080271 -4.5110 6.453e-06 ***
```

Valoramos la relación entre los valores de **LRT** y los coeficientes de regresión estandarizados (**beta**) en valor absoluto.

```
plot(c(10.63, 43.39, 8.69, 61.90, 15.38), c(0.48, 0.80, 0.29, 0.98, 0.36),
+ xlab="valores de LRT", ylab="valores de coeficientes en valor absoluto")
abline (lm(c(0.48, 0.80, 0.29, 0.98, 0.36) ~ c(10.63, 43.39, 8.69, 61.90, 15.38)), col="red")
```



Vemos que están muy relacionados ambos valores.

`lrtest(modelo)` da una Chisq = 219.19

Como la suma de LRT es = 139.99

LA SUMA DE LAS CONTRIBUCIONES PARCIALES NO EQUIVALE A LA DEVIANZA RETENIDA POR EL MODELO PORQUE LAS VARIABLES PREDICTORAS **NO SON INDEPENDIENTES ENTRE SÍ**
Lo cual vimos con `vif(modelo)`

¿Y si queremos actualizar nuestro modelo quitando un efecto y añadiendo otro no utilizado previamente?

Podemos crear una nueva ecuación y un nuevo modelo o … mucho mejor actualizar el que ya tenemos. Para ello utilizamos el comando `update`. Pongo varios casos de ejemplo:

```
modelo.nulo <- update(modelo, .~. -tempmin -precip -altmed -rangoalt -shannon)
modelo.sinclima <- update(modelo, .~. -tempmin -precip)
modelo.soloclima <- update(modelo, .~. -altmed -rangoalt -shannon)
modelo.zepas <- update(modelo, .~. +zepakm2)
```

Calculamos ahora su verosimilitud comparada mediante AICc:

```
AICc(modelo.nulo, modelo, modelo.zepas, modelo.sinclima, modelo.soloclima)
                df     AICc
modelo.nulo      2 1898.370
modelo           7 1689.279
modelo.zepas     8 1677.476                Este es el mejor modelo
modelo.sinclima  5 1752.683
modelo.soloclima 4 1751.209
```

¿Por qué el modelo nulo (`modelo.nulo`) tiene dos grados de libertad (`df`) si sólo contiene el intercepto (`ptyrup ~1)`?

Porque los modelos binomiales negativos también efectúan la estima del parámetro theta que permite obtener la varianza de la respuesta a partir de su media. Recordemos: varianza = mu + (1 / theta)*mu^2. A menor theta mayor varianza.

Y esto plantea un pequeño problema: los valores de AICc en los modelos binomiales negativos no son “perfectamente” comparables si no tienen todos ellos exactamente el mismo valor de theta. Pero bueno … ¡es lo que hay!

```
modelo.nulo$theta; modelo$theta; modelo.zepas$theta; modelo.sinclima$theta; modelo.soloclima$theta
[1] 0.1276754
[1] 0.2974643
[1] 0.3089555          Parecidos, pero no idénticos (sobre todo el nulo)
[1] 0.2224345
[1] 0.2368135
```

Otra aproximación sintética a los resultados del modelo es la basada en la Teoría de la Información y en la construcción de una síntesis multimodelo. Esta aproximación multimodelo genera todos los modelos posibles, teniendo en cuenta todas las combinaciones posibles de las variables predictoras. A cada modelo generado se le asigna su valor de Akaike (**AICc**), a partir de ellos se obtienen los **pesos w_i** de los modelos, y se efectúa una media ponderada con esos pesos de los resultados obtenibles utilizando todos los modelos. Esta aproximación genera unos nuevos errores estándar de los coeficientes promediados, que se llaman errores estándar incondicionales. **ESTA FORMA DE TRABAJAR NO ES FRECUENTISTA BASADA EN "p's".**

interesante WEB: http://theses.ulaval.ca/archimede/fichiers/21842/apa.html
repasad este PDF: http://machinelearning102.pbworks.com/w/file/fetch/47699411/aic_reg.pdf
este PDF también está bien: http://avesbiodiv.mncn.csic.es/estadistica/multiinf.pdf

Para ello cargamos la librería `glmulti` (cuidado con no tener instalado *Java* en la misma versión que R y RStudio; 32 o 64 bits).

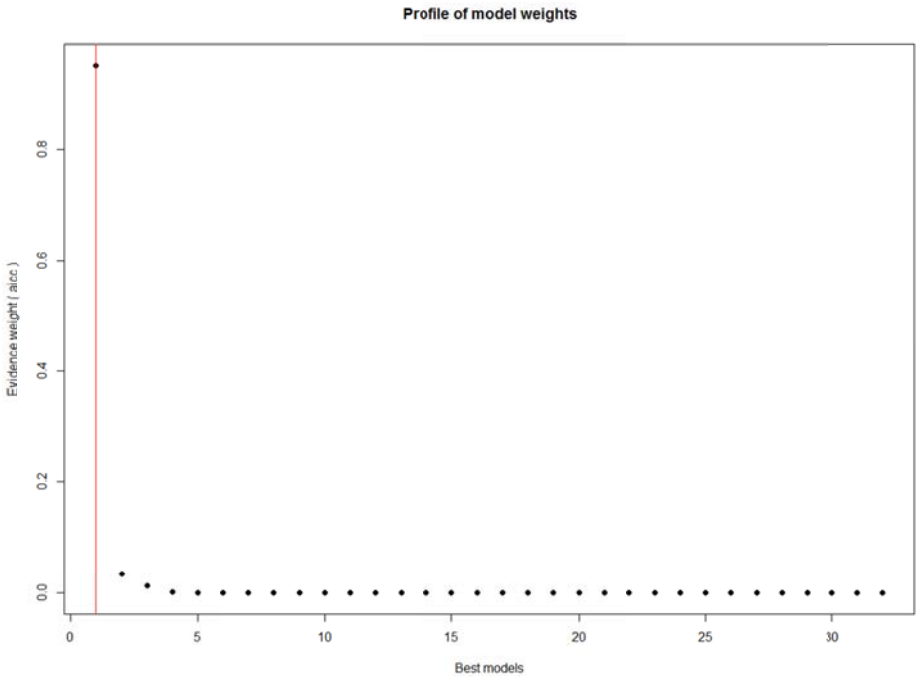
```
library(glmulti)

## con binomiales negativas introducimos la ecuación del modelo (eqt) y no el "modelo" previamente creado
## "h" para exhaustive screening; "g" para "genetic algorithm method"
multimodelo <- glmulti(eqt, data=datos, fitfunction=glm.nb, level=1, confsetsize=100, method="h", crit="aicc")
weightable(multimodelo)

cumsum(weightable(multimodelo)$weights)          ## suma acumulada a partir del mejor modelo (primero [1])
 [1] 0.9517336 0.9856947 0.9985995 0.9998004 0.9998792 0.9999429 0.9999993 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
[13] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
[25] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000

sum(weightable(multimodelo)$weights[1:10])      ## suma de weights de los diez mejores modelos
[1] 1

plot(multimodelo, type="w")    ## para visualizar la evolución de los weights de los modelos (línea roja =cumsum95%)
plot(multimodelo, type="s")    ## para visualizar el peso de las variables
```

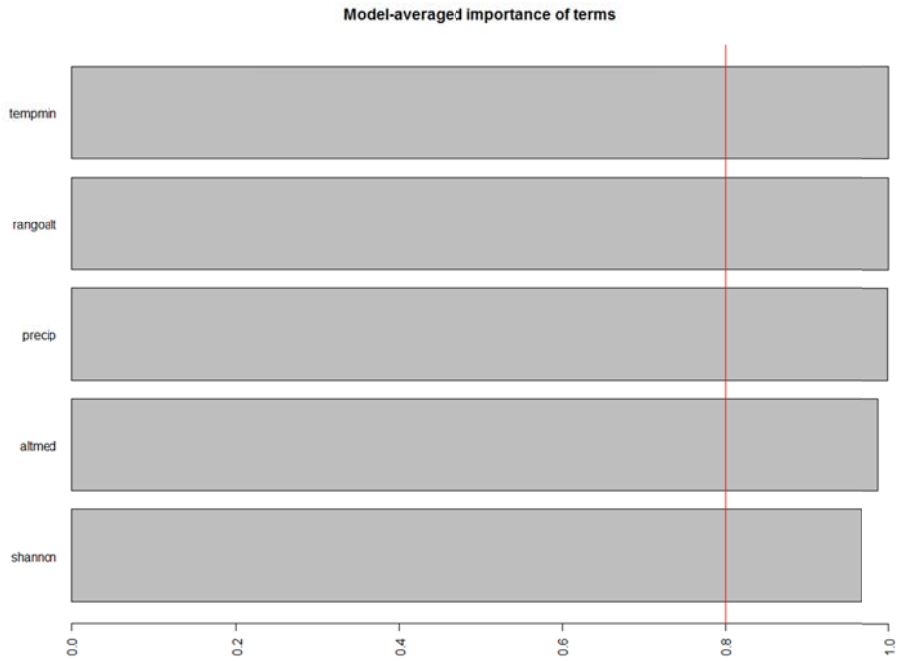
Veamos a continuación la salida textual de todos los modelos posibles con 5 variables predictoras con `weightable(multimodelo)`

Y veamos las dos figuras generadas con las dos últimas líneas de código.

```
weightable(multimodelo)
                                                    model     aicc      weights
1  ptyrup ~ 1 + altmed + rangoalt + shannon + tempmin + precip 1689.279 9.517336e-01    este es el MEJOR
2            ptyrup ~ 1 + altmed + rangoalt + tempmin + precip 1695.945 3.396114e-02
3           ptyrup ~ 1 + rangoalt + shannon + tempmin + precip 1697.880 1.290476e-02
4           ptyrup ~ 1 + altmed + rangoalt + shannon + tempmin 1702.629 1.200954e-03
5                    ptyrup ~ 1 + rangoalt + tempmin + precip 1708.079 7.872543e-05
6                    ptyrup ~ 1 + altmed + rangoalt + tempmin 1708.503 6.368868e-05
7                   ptyrup ~ 1 + rangoalt + shannon + tempmin 1708.746 5.640643e-05
8                             ptyrup ~ 1 + rangoalt + tempmin 1717.430 7.335533e-07
9                     ptyrup ~ 1 + shannon + tempmin + precip 1730.084 1.311163e-09
10                             ptyrup ~ 1 + shannon + tempmin 1730.439 1.098233e-09
11           ptyrup ~ 1 + altmed + shannon + tempmin + precip 1730.639 9.936821e-10
12                    ptyrup ~ 1 + altmed + shannon + tempmin 1731.524 6.383522e-10
13          ptyrup ~ 1 + altmed + rangoalt + shannon + precip 1749.146 9.514827e-14
14                                      ptyrup ~ 1 + tempmin 1749.759 7.002532e-14
15                             ptyrup ~ 1 + altmed + tempmin 1750.448 4.962222e-14
16                    ptyrup ~ 1 + altmed + rangoalt + precip 1751.037 3.696313e-14
17                             ptyrup ~ 1 + tempmin + precip 1751.209 3.392712e-14
18                      ptyrup ~ 1 + altmed + tempmin + precip 1751.608 2.779080e-14
19                    ptyrup ~ 1 + altmed + rangoalt + shannon 1752.683 1.623190e-14
20                             ptyrup ~ 1 + altmed + rangoalt 1754.636 6.114103e-15
21                      ptyrup ~ 1 + altmed + shannon + precip 1840.293 1.535295e-33
22                               ptyrup ~ 1 + altmed + shannon 1841.376 8.931645e-34
23                                ptyrup ~ 1 + altmed + precip 1854.414 1.317493e-36
24                                         ptyrup ~ 1 + altmed 1857.343 3.046820e-37
25                             ptyrup ~ 1 + rangoalt + shannon 1882.107 1.277258e-42
26                    ptyrup ~ 1 + rangoalt + shannon + precip 1883.948 5.089250e-43
27                                      ptyrup ~ 1 + shannon 1887.073 1.066496e-43
28                               ptyrup ~ 1 + shannon + precip 1887.497 8.629155e-44
29                                       ptyrup ~ 1 + rangoalt 1889.502 3.166583e-44
30                             ptyrup ~ 1 + rangoalt + precip 1891.106 1.419637e-44
31                                        ptyrup ~ 1 + precip 1897.356 6.237575e-46
32                                                ptyrup ~ 1 1898.370 3.758130e-46    este es el NULO
```

Realmente, sólo es importante el primer modelo (el saturado, original) que tiene un **peso** `weights` **de 0.952**





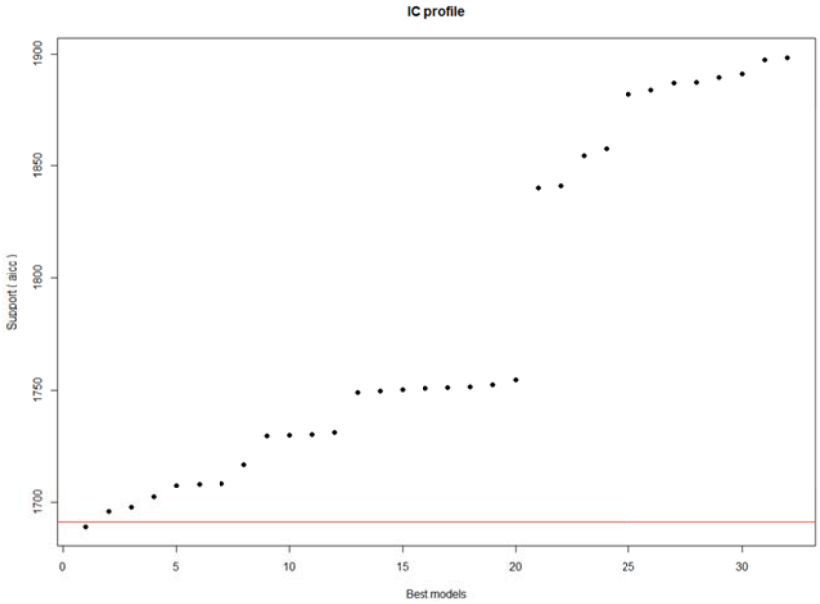
Todas las variables predictoras son importantes, porque las sumas de los pesos en los modelos en los que entran son muy altas.

Y finalmente la tabla numérica con los valores obtenidos en la aproximación multimodelo. No veremos significaciones “p” en ella.

Podemos utilizar el error estándar incondicional de los coeficientes de regresión para saber cuántas veces hay que utilizarlos para alcanzar el valor CERO (hipótesis nula de ausencia de efectos); esta “distancia” sería equivalente a una t de Student.

```
tabla.uncond <- as.data.frame(coef.glmulti(multimodelo, select="all", varweighting="Buckland", icmethod="Burnham", alphaIC=0.05))
names(tabla.uncond)[2] <- "var.uncond"
names(tabla.uncond)[5] <- "-/+ IC95%"
tabla.uncond$stderror <- sqrt(tabla.uncond$var.uncond)
tabla.uncond
```

	Estimate	var.uncond	Nb models	Importance	-/+ IC95%	stderror
shannon	0.587519725	4.405328e-02	16	0.9658957	0.4118220049	0.2098887217
altmed	-0.001208149	1.745843e-07	16	0.9869594	0.0008198919	0.0004178329
precip	-0.002242841	2.965021e-07	16	0.9986782	0.0010685324	0.0005445201
rangoalt	0.002128790	1.008560e-07	16	1.0000000	0.0006231684	0.0003175784
tempmin	0.400787410	2.984559e-03	16	1.0000000	0.1072016797	0.0546311187
(Intercept)	-2.692904314	2.715579e-01	32	1.0000000	1.0224911846	0.5211122153



Vamos ahora a predecir con el modelo los valores esperados en otro juego de datos teniendo en cuenta los valores de las variables predictoras.

Para ello haremos uso del comando `predict`, y tendremos que cargar un nuevo juego de datos con esa información (en esta ocasión llamado "predecir", con N=689). El tipo de la predicción lo establecemos en esta ocasión en la escala de medida original de la variable respuesta. Los datos predichos de la variable respuesta los llamo como quiero: `predecir$ptyrup.nuevo`.

```
predecir$ptyrup.nuevo <- predict(modelo, newdata=predecir, type="response")
```

Como también contamos con la variable respuesta `ptyrup` (medida realmente en el campo) en el juego de datos `predecir`, podemos efectuar un test del poder predictivo del modelo "atinando" en las predicciones de la variable respuesta. Para ello, regresionamos los datos de la variable respuesta, medidos pero no utilizados en la construcción de nuestro `modelo` GLM (`predecir$ptyrup`), con los predichos por él (`predecir$ptyrup.nuevo`).

```
summary(lm(predecir$ptyrup~predecir$ptyrup.nuevo))
```

```
Call:
lm(formula = predecir$ptyrup ~ predecir$ptyrup.nuevo)
Residuals:
     Min       1Q   Median       3Q      Max
-10.7369  -0.5260  -0.2539  -0.1549  22.1621

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)          0.11556    0.09454   1.222    0.222
predecir$ptyrup.nuevo  0.87714    0.04693  18.689   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.219 on 687 degrees of freedom
Multiple R-squared:  0.3371,   Adjusted R-squared:  0.3361
F-statistic: 349.3 on 1 and 687 DF,  p-value: < 2.2e-16
```

Como vemos, los datos observados y predichos en el juego de datos "externo" al modelo están relacionados de modo significativo (R^2 = 33.71%, p<<0.001) y definen la siguiente relación:

OBSERVADO = 0.116 + 0.877 * PREDICHO

en comparación con la perfecta asociación definida por R2 = 100% y **OBSERVADO = 0 + 1 * PREDICHO**

El intercepto (0.116) no difiere significativamente de "cero" (p=0.222).

Y la pendiente de `predecir$ptyrup.nuevo` (**0.877**), aunque significativamente diferente de cero (**0.87714** / 0.04693 = t = 18.689; **p<<<0.001**) … también difiere de UNO (**1**) establecida en la relación marcada en amarillo (i.e., identidad observado – predicho):

```
(1-0.87714) / 0.04693        ## esto es una t de Student de desvío de un coeficiente de regresión del valor 1
[1] 2.617942

2*(1-pt(2.617942, df=687))   ## cálculo de la p para una t de Student, con dos colas
[1] 0.009041038
```

Por tanto, el modelo tiene un "apreciable" grado de predecibilidad (del 33.71%), es significativa (p<<0.001), y tiende a la subestima creciente en valores altos de la respuesta (pendiente **0.877**).

Recordemos que el modelo original explicaba el `D2 de MCFadden (%) = 36.01` de la devianza original. En la escala de la medida de la variable respuesta (i.e., haciendo la anti-transformación de la *link function* log), la R^2 de nuestro modelo es 20.1%:

```
cor(modelo$y, modelo$fitted.values)^2  ## correlación al cuadrado = R^2
[1] 0.200927                           ## llamamos a los valores de la respuesta-y y a los valores predichos en la
                                       ## escala original de medida contenidos en el modelo.
```

¡¡ Pues no está nada mal !! Recordad, esto ha sido para valorar el poder predictivo de un modelo.

Utilizando esta aproximación, podemos visualizar el efecto parcial que una variable predictora de nuestro modelo tendría sobre la variable respuesta … controlando por todas las otras variables predictoras consideradas en el modelo.

Para ello, "nos inventamos" un nuevo conjunto de datos (por ejemplo, 1000 pseudo-unidades muestrales) en donde establezcamos el rango de variación real de la variable predictora de interés que queramos visualizar. A las otras variables predictoras "les decimos" … a todo lo demás igual en las unidades muestrales y les asignamos a cada una de ellas su valor medio observado en la matriz de datos de análisis. Supongamos que queremos visualizar el efecto de la temperatura mínima sobre la abundancia relativa del avión roquero (presencia en 60 transectos de 15 minutos de duración cada uno) … controlando por todas las demás variables. Primero estimamos su rango de variación real en nuestros datos.

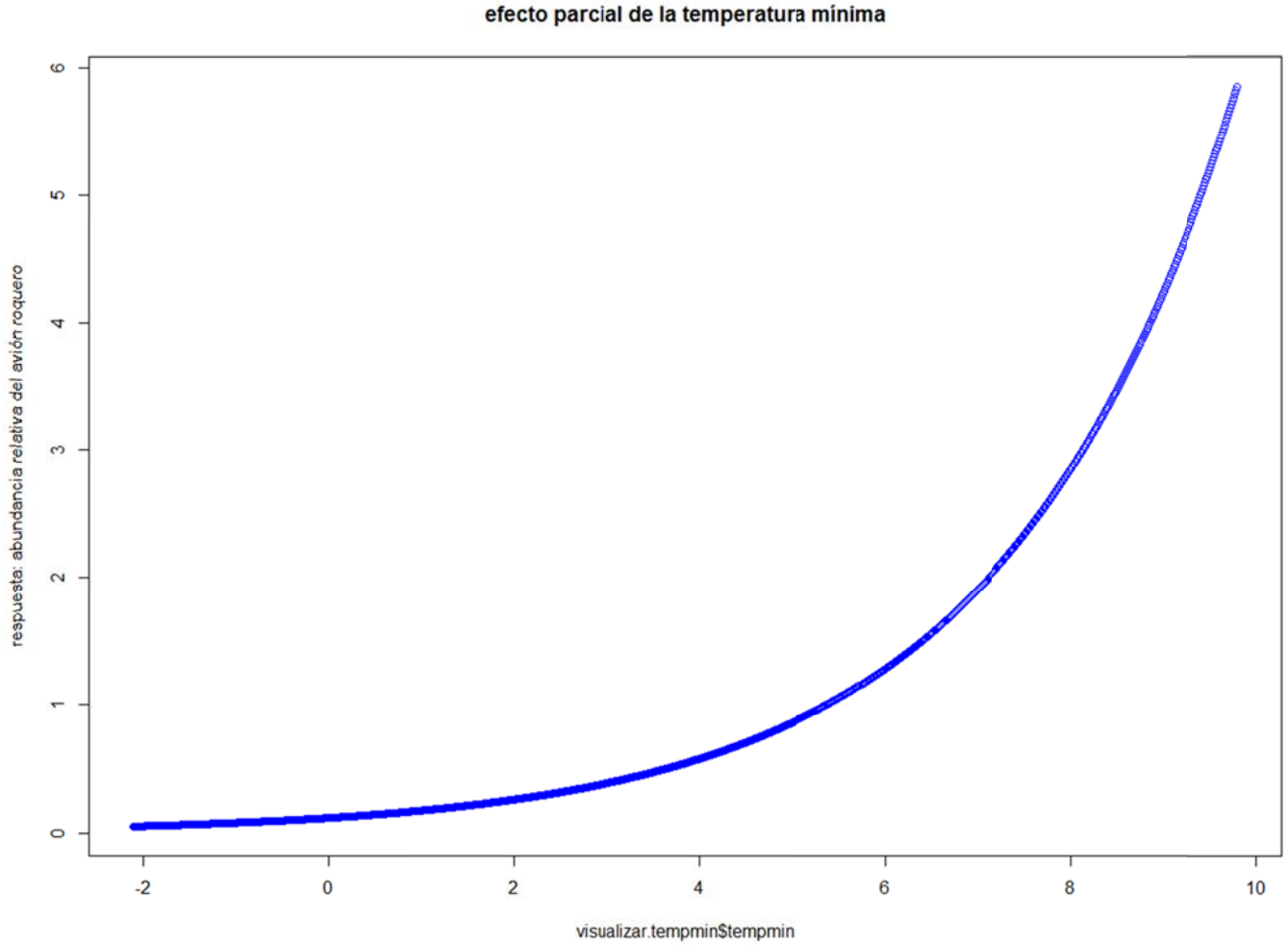
```
min(datos$tempmin)
[1] -2.1
max(datos$tempmin)
[1] 9.8
```

A continuación creamos un nuevo *data frame* "inventado" que contiene mil unidades muestrales (o 757, o … lo que sea, pero muchos datos, para presentar tendencias bien dibujadas), en el cual al resto de las variables les asignamos su media. Utilizaremos EXACTAMENTE los mismos nombres de las variables predictoras originales incluidas en nuestra ecuación del modelo (`eqt`).

```
visualizar.tempmin <- data.frame(tempmin=seq(-2.1, 9.8, length=1000), altmed=mean(datos$altmed),
+ rangoalt=mean(datos$rangoalt), shannon=mean(datos$shannon), precip=mean(datos$precip))
```

Y para terminar, le aplicamos a este nuevo *data frame* "inventado", denominado `visualizar.tempmin`, el modelo original, generando una variable que denominamos "parciales" (`visualizar.tempmin$parciales`), y obtenemos un gráfico:

```
visualizar.tempmin$parciales <- predict(modelo, newdata=visualizar.tempmin, type="response")
plot(visualizar.tempmin$parciales ~ visualizar.tempmin$tempmin, main="efecto parcial de la temperatura mínima",
+ ylab="respuesta: abundancia relativa del avión roquero", col="blue")
```



Como en las exploraciones de los valores residuales "dato-a-dato" habíamos identificado la existencia de puntos influyentes y/o perdidos mediante el uso de ***leverage***, **dffits**, **distancias de Cook** y **dfbetas**, los resultados de nuestro modelo puede que no sean estables y robustos a la existencia de esos datos.

Si tuviésemos fuertes evidencias de que esos datos son erróneos o presentan graves problemas de precisión, certidumbre, etc, podríamos eliminarlos de nuestro modelo, construyendo otro nuevo que no cuente con ellos. Esto es, recalcularíamos nuestro modelo excluyendo datos: por ejemplo, las unidades muestrales con un "Index" (i.e., número de orden en las filas del *data frame* datos) 88, 116, 161, 190, 437, 735 (que son valores muy extremos en la distancia de Cook, `cooks.distance(modelo)`).

Para ello repetimos el modelo pero quitando datos aplicando el argumento `subset` a nuestro `data`. Y comparamos con lo previo.

```
modelo.quitando.datos <- glm.nb(eqt, data=subset(datos[c(-88, -116, -161, -190, -437, -735),]), link=log)
```

```
summary(modelo.quitando.datos)
Call:
glm.nb(formula = eqt, data = subset(datos[c(-88, -116, -161,
    -190, -437, -735), ]), link = log, init.theta = 0.33638841)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6475  -0.7010  -0.4204  -0.2378   3.0678

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.3084597  0.4905285  -4.706 2.53e-06 ***
altmed      -0.0020434  0.0004234  -4.826 1.39e-06 ***
rangoalt     0.0024952  0.0003116   8.006 1.18e-15 ***
shannon      0.6107293  0.1967504   3.104  0.00191 **
tempmin      0.3678653  0.0534839   6.878 6.07e-12 ***
precip      -0.0027135  0.0005352  -5.070 3.97e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(0.3364) family take
n to be 1)

    Null deviance: 863.06  on 992  degrees of freedom
Residual deviance: 507.99  on 987  degrees of freedom
```

```
summary(modelo)
Call:
glm.nb(formula = eqt, data = datos, link = log, init.theta =
0.2974642714)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5784  -0.7180  -0.4646  -0.2735   4.5393

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.7169395  0.4904432  -5.540 3.03e-08 ***
altmed      -0.0012181  0.0004057  -3.002  0.00268 **
rangoalt     0.0021294  0.0003095   6.880 5.97e-12 ***
shannon      0.6068703  0.1951244   3.110  0.00187 **
tempmin      0.4000696  0.0536370   7.459 8.73e-14 ***
precip      -0.0022501  0.0005417  -4.154 3.27e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Negative Binomial(0.2975) family take
n to be 1)

    Null deviance: 826.86  on 998  degrees of freedom
Residual deviance: 529.13  on 993  degrees of freedom
```

Otra manera más objetiva de robustecer nuestros datos es recurrir a una aproximación robusta que tenga en cuenta cómo de "*outliers*" son nuestras observaciones, de manera que se les dé menos peso en el análisis a aquellas observaciones poco consistentes con el patrón global observado.

Para complenderlo, vamos a abordarlo con una aproximación antigua ("*Regression Sensitivity Analysis and Bounded-Influence Estimation*" RSABIE, http://www.nber.org/chapters/c11698.pdf). Es sencillo, construimos un valor de "peso" para cada observación que es inversamente proporcional a su valor de **dffit**: las unidades muestrales con elevado **dffit**, pesarán menos en el modelo que aquellas que introducen menos alteraciones. Y esto lo vamos a hacer a partir del umbral crítico, o preocupante de los dffits (i.e., aquellos que sean mayores que **2*raíz[p/n]**, como ya expusimos al presentar los **dffits**); además corregiremos la suma de esos pesos para todas nuestras 999 observaciones – unidades muestrales, de manera que la suma de los valores de los "pesos" sea igual a 999. `length(modelo$coefficients)` nos da los parámetros incluidos en el modelo. `length(…)` mide la "longitud" de un vector.

```
k.dffits.modelo <- 2*((length(modelo$coefficients)/length(modelo$y))^0.5)
w.dffits.intermedio <- ifelse(abs(dffits(modelo))<k.dffits.modelo,1,k.dffits.modelo/abs(dffits(modelo)))
w.dffits.modelo <- w.dffits.intermedio * (999/sum(w.dffits.intermedio))
sum(w.dffits.modelo)
[1] 999
```

Construimos el nuevo modelo `modelo.RSABIE` con esos pesos `w.dffits.modelo` que introducimos en el argumento `weights`.

```
modelo.RSABIE <- glm.nb(eqt, data=datos, link=log, weights=w.dffits.modelo)
```

Y ahora podemos revisar sus supuestos y resultados del mismo modo que hemos hecho antes (no presento las salidas de los análisis por abreviar)

```
plot(modelo.RSABIE)
plot(dffits(modelo.RSABIE)); plot(cooks.distance(modelo.RSABIE))

lrtest(modelo.RSABIE)
waldtest(modelo.RSABIE, vcov=sandwich)
d2.RSABIE <- round((modelo.RSABIE$null.deviance-modelo.RSABIE$deviance)/modelo.RSABIE$null.deviance, 4)
print(c("D2 de MCFadden (%) =", 100*d2.RSABIE), quote=FALSE)
dropterm(modelo.RSABIE, test="Chisq", sorted=FALSE)
coeftest(modelo.RSABIE, vcov=sandwich)
```

Otra aproximación robusta es efectuar cientos de modelos aplicados a matrices obtenidas por remuestreo con reemplazo de nuestros datos originales (http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/appendix/Appendix-Bootstrapping.pdf http://en.wikipedia.org/wiki/Bootstrapping_(statistics) http://www.dst.unive.it/rsr/BelVenTutorial.pdf ; bootstrapping).

BOOTSTRAPPING DE LOS COEFICIENTES DEL MODELO (con **B** modificamos el número de procesos de remuestreo)

```
boot.modelo <- as.data.frame(bootCase(modelo, f.=coef, B=1000))
```

Y ahora representamos los datos mediante estas líneas de código:

```
## QUE LOS INTERVALOS SIGUIENTES NO INCLUYAN EL VALOR "CERO"
nv <- length(boot.modelo)
mb <- vector(length=nv)
sdb <- vector(length=nv)
tb <- vector(length=nv)
for (i in 1:nv) {
  print(c(colnames(boot.modelo[i]),"=",round(mean(boot.modelo[,i]),4)), quote=FALSE)
  mb[i] <- mean(boot.modelo[,i])
  sdb[i] <- sd(boot.modelo[,i])
  tb[i] <- mean(boot.modelo[,i])/sd(boot.modelo[,i])
  vrc=modelo$coefficients[i]/1
  hist(boot.modelo[,i], breaks=20, xlab=names(boot.modelo[i]), main=c(names(boot.modelo[i]),round(vrc,3)), col.main="red")
  abline(v=modelo$coefficients[i], col="red")                  ## en ROJO parámetros observados en el modelo que se examina
  abline(v=quantile(boot.modelo[,i], 0.025), col="blue")       ## en AZUL los límites a alfa = 0.05 de los valores remuestreados
  abline(v=quantile(boot.modelo[,i], 0.975), col="blue")
  abline(v=0, col="green")                                     ## en VERDE el valor cero indicativo de la Ho (hipótesis nula)
  print(quantile(boot.modelo[,i], c(0.005, 0.01, 0.025, 0.05, 0.95, 0.975, 0.99, 0.995)), quote=FALSE)
  print("-------------------------------------------------", quote=FALSE)
}
```

Que se pueden correr de una “tacada” seleccionándolas en el panel de “scripts” (*Source*) y luego pulsando [ctrl]+[intro].

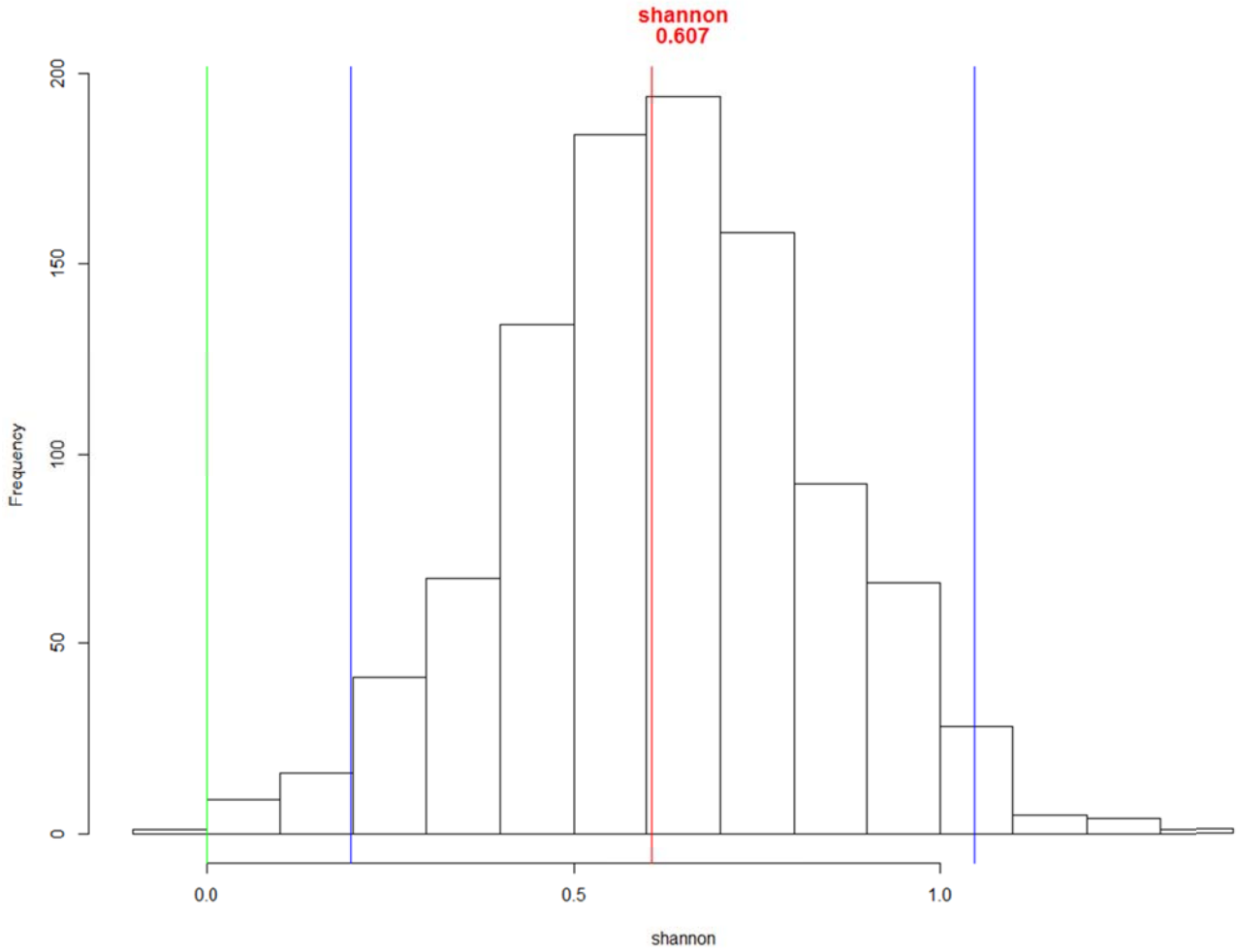
QUE LOS INTERVALOS SIGUIENTES NO INCLUYAN EL VALOR "CERO"

A alfa = 0.05 deberíamos elegir los cuantiles 2.5% y 97.5% para umbrales de significación con dos colas.

```
[1] (Intercept) =           -2.7035
     0.5%        1%      2.5%        5%       95%     97.5%       99%     99.5%
-4.150918 -3.947420 -3.728857 -3.567322 -1.855587 -1.694664 -1.448276 -1.359971
[1] ----------------------------------------------------
[1] altmed  =       -0.0013
         0.5%            1%           2.5%            5%           95%         97.5%           99%         99.5%
-0.0024751982 -0.0024314953 -0.0022144109 -0.0020878984 -0.0006440625 -0.0005260731 -0.0003702498 -0.0003021672
[1] ----------------------------------------------------
[1] rangoalt =        0.0021
       0.5%         1%        2.5%          5%         95%       97.5%         99%       99.5%
0.001392978 0.001447251 0.001520046 0.001635953 0.002685152 0.002764062 0.002938575 0.002970957
[1] ----------------------------------------------------
[1] shannon =       0.6258
      0.5%         1%       2.5%         5%        95%      97.5%        99%      99.5%
0.06398472 0.11091772 0.19707978 0.27597502 0.97283613 1.04679992 1.09849168 1.16848162
[1] ----------------------------------------------------
[1] tempmin =       0.3969
     0.5%        1%       2.5%        5%       95%     97.5%       99%     99.5%
0.2405171 0.2536928 0.2720624 0.2979431 0.4956730 0.5158991 0.5386686 0.5414347
[1] ----------------------------------------------------
[1] precip  =       -0.0023
         0.5%            1%           2.5%            5%           95%         97.5%           99%         99.5%
-0.0036767469 -0.0035079711 -0.0033133952 -0.0031565347 -0.0014522571 -0.0012563849 -0.0010356309 -0.0008862322
[1] ----------------------------------------------------
```

Podemos observar que el valor CERO (cumplimiento de la Ho de ausencia de efecto) no se incluye en el intervalo marcado en AZUL a una p=0.05 con dos colas para la variable `tempmin`.

O visto de modo gráfico (sólo un ejemplo, de las múltiples salidas): AZUL: intervalo al 95%; ROJO: coeficiente real del modelo, VERDE: hipótesis nula de ausencia de efectos, coeficinte = 0). El valor numérico **ROJO** es la media de los 1000 procesos.



O con una tabla de resultados con los valores sintéticos de los 1000 procesos de *bootstrapping*. El 'standard error' de los coeficientes *'bootstrapped'* es equivalente a la **sd** (desviación típica) de los procesos.

```
print(describe(boot.modelo)[,c(1:4,8:9,11:12)], digits=5)
            vars    n     mean      sd      min      max     skew kurtosis
(Intercept)    1 1000 -2.70345 0.52016 -4.42626 -1.12419 -0.02960  0.16041
altmed         2 1000 -0.00129 0.00043 -0.00293 -0.00004 -0.39075  0.21204
rangoalt       3 1000  0.00215 0.00032  0.00118  0.00316  0.08089 -0.20011
shannon        4 1000  0.62580 0.21100 -0.09793  1.30455 -0.03360  0.10700
tempmin        5 1000  0.39693 0.06079  0.22345  0.60059 -0.03218 -0.10797
precip         6 1000 -0.00231 0.00052 -0.00420 -0.00066  0.00108  0.21749
```

Con `[,c(1:4,8:9,11:12)]` hemos seleccionado las columnas de interés (de todas las existentes) en la tabla `boot.modelo`.

Cuyas significaciones, asumiendo distrubuciones gausianas de los valores de coeficientes remuestreados, son:

```
print("valores de p de los coeficientes 'bootstrapped'", quote=FALSE)
[1] valores de p de los coeficientes 'bootstrapped'
for (i in 1:nv) {
   print(c(colnames(boot.modelo[i]), "", 2*pt(-abs(tb[i]), df=999)), digits=5, quote=FALSE)
 }
[1] (Intercept)                                2.4500857846491e-07
[1] altmed                                     0.00313774400066512
[1] rangoalt                                   5.9705653436480e-11
[1] shannon                                    0.00309017176499345
[1] tempmin                                    1.0450438900935e-10
[1] precip                                     1.0858987028576e-05
```

De la misma manera que hemos construido, mediante procesos de re-muestreo aleatorio de los datos, distribuciones de los parámetros de nuestro modelo, con el objeto de obtener estimas robustas (teniendo en cuenta las violaciones de los supuestos canónicos y la existencia de puntos extremos, influyentes y/o perdidos), también podemos utilizar esta aproximación para construir **MODELOS NULOS** que permitan estimar la significación de los coeficientes del modelo.

En este nuevo proceso de aleatorización de nuestros datos originales vamos a introducir una nueva idea: desacoplar la covariación real observada entre la variable respuesta y las variables predictoras.

Para ello, vamos a remuestrear, sin re-emplazo, la variable respuesta según un proceso muy parecido a lo que hacemos al barajar cartas (*shuffling*). De esta manera, cada valor de una unidad muestral en la variable respuesta Y_i, queda desvinculada de su correspondencia o asignación real en la unidad muestral **i** respecto a las **j** variables predictoras **X** (X_{ji}). Esto implica que no debería existir una relación "significativa" entre la variable **Y** y las **X**'s, y por tanto, se cumpliría la hipótesis nula de ausencia de relaciones.

Este es el proceso utilizado para construir tablas de significación (e.g., de la Chi^2, t de Student, F de Fisher, r de correlaciones, etc), para diferentes configuraciones muestrales de número de datos y variables predictoras que definen los grados de libertad. Pero en las tablas de significación, de las que se obtienen los valores de "p" de nuestros modelos, se cumplen exquisitamente las asunciones de independencia, ajuste a supuestos canónicos, ausencia de puntos "*outliers*", etc.

En esta ocasión vamos a hacer algo similar, pero **construyendo la tabla de "significación correcta" para nuestros datos** teniendo en cuenta los sesgos implícitos que hay en ellos, y las violaciones de los supuestos canónicos que conllevan. En esencia, el proceso implica tres pasos: (1) desvincular Y_i de las X_{ji}; (2) construir el modelo (denominado nulo) con los mismos supuestos de familia de distribución de la variable respuesta y la función de vínculo; (3) repetir el proceso 1 y 2 numerosísimas veces para obtener la distribución "nula" de los coeficientes del modelo.

Téngase en cuenta que de esta manera (a) repetamos la existencia de las relaciones realmente observadas entre las variables predictoras, y (b) mantenemos la dimensionalidad de los datos analizados (tamaño muestral y grados de libertad de la función de predictores **g(x)**).

Para ello partimos de la misma ecuación con la que va trabajar nuestro modelo y que hemos definido como:

```
eqt <- as.formula(ptyrup ~ altmed+rangoalt+shannon+tempmin+precip)

modelo <- glm.nb(eqt, data=datos, link=log)

d2 <- round(100*(modelo$null.deviance-modelo$deviance)/modelo$null.deviance,2)     ## para la estima de la devianza explicada
```

Ahora **generaremos 2000 modelos nulos**, cambiando en las siguientes líneas de código los valores “2000” por otros si queremos hacer más o menos (no recomendable): en “**nrow=**” y en “**for (i in 1:2000)**”. Tal y como está escrito el código, tenemos que incluir el nombre de la variable respuesta en YYYYYY y la función de predictoras en XXXXXX (YYYYYY ~ XXXXXX).

En nuestro caso:

```
ptyrup ~ altmed+rangoalt+shannon+tempmin+precip
```

luego XXXXXX es `altmed+rangoalt+shannon+tempmin+precip`
e YYYYYY es `ptyrup`

```
nv <-length(modelo$coefficients)
nula <- matrix(99999, nrow=2000, ncol=nv+1)
for (i in 1:2000)  {
    rr <- sample(datos$YYYYYY)
    mn <- glm.nb(rr~XXXXXX, data=datos, link=log)     ## para otras transformaciones modificar link
    cr <- as.vector(coef(mn))
    d <- deviance(mn)
    dn <- summary(mn)$null.deviance
    dn2 <- 100*(dn-d)/dn
    nula[i,1:nv] <- cr
    nula[i,nv+1] <- dn2
}
```

En el caso práctico que estamos viendo sería:

```
nv <-length(modelo$coefficients)
nula <- matrix(99999, nrow=2000, ncol=nv+1)
for (i in 1:2000)  {
    rr <- sample(datos$ptyrup)
    mn <- glm.nb(rr~altmed+rangoalt+shannon+tempmin+precip, data=datos, link=log)
    cr <- as.vector(coef(mn))
    d <- deviance(mn)
    dn <- summary(mn)$null.deviance
    dn2 <- 100*(dn-d)/dn
    nula[i,1:nv] <- cr
    nula[i,nv+1] <- dn2
}
```

A continuación, seleccionamos las líneas siguientes y las corremos de una “tacada”.

```
## seleccionar las siguientes líneas y correrlas de una sola vez hasta donde se marca ##### HASTA AQUI #####
mnula <- as.data.frame(nula)    ## para convertir en un data.frame "mnula" el atomic vector "nula"
names(mnula)                    ## produce [1] "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8" ...
##
## COMPROBAR QUE LOS VALORES INDICADOS QUEDEN FUERA DE LOS INTERVALOS
nn <- length(mnula)
nombres <- c(modelo$coefficients,d2)
names(nombres)[nn] <- paste("D2%")
for (i in 1:nn) {
  vrc=modelo$coefficients[i]/1
  hist(mnula[,i], breaks=20, xlab=names(nombres[i]), main=c(names(nombres[i]),round(vrc,3)), col.main="red")
  abline(v=modelo$coefficients[i], col="red")
  abline(v=quantile(mnula[,i], 0.025), col="blue")       ## en AZUL los límites a alfa = 0.05 de los valores nulos
  abline(v=quantile(mnula[,i], 0.975), col="blue")
  print(nombres[i])
  print(quantile(mnula[,i], c(0.005, 0.025, 0.05, 0.95, 0.975, 0.995)))
  print("-------------------------------------------------", quote=FALSE)
}
##### HASTA AQUI #####
```

Las líneas de código previas proporcionan los resultados textuales y gráficos de las 2000 simulaciones nulas efectuadas, que dicho sea de paso habrán consumido un tiempo apreciable de cómputo en el ordenador.

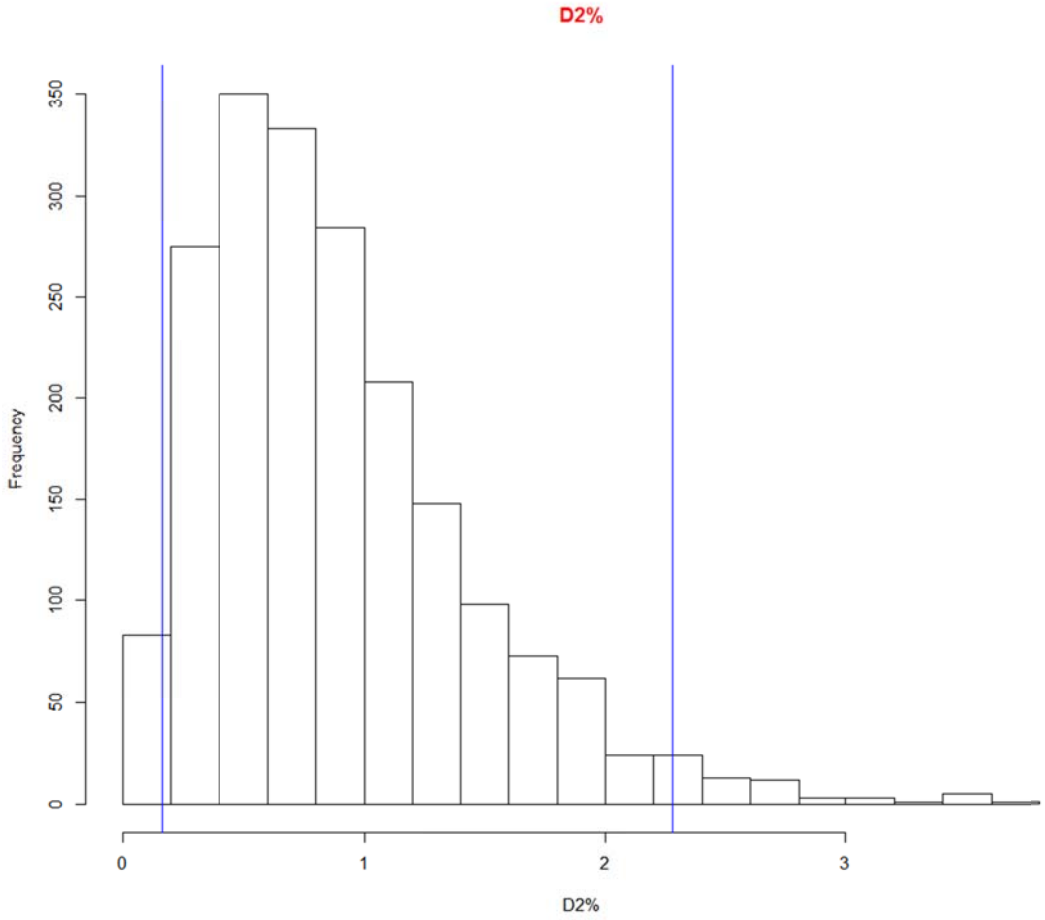
Además de calcular los “coeficientes nulos” del modelo, el procedimiento implementado también ha computado la devianza explicada (índice de McFadden), con lo cual podremos cuantificar qué cantidad de la variabilidad observada en la variable respuesta puede ser explicada por puro azar.

Si los coeficientes de regresión obtenidos en nuestro modelo (`modelo`) son “realmente” significativos, sus valores observados deberán quedar **fuera de los intervalos de “confianza nulos”** generados en los 2000 procesos de aleatorización-modelización. Si quedan a la izquierda de los puntos de corte umbral definidos mediante cuantiles para la alfa establecida (digamos alfa ≈ p = 0.05), entonces la relación será significativamente negativa a esa alfa; si el valor real del coeficiente queda a la derecha de la distribución nula de coeficientes y su cuantil a esa alfa, entonces diremos que existe una relación positiva significativa. Veámoslo a continuación.

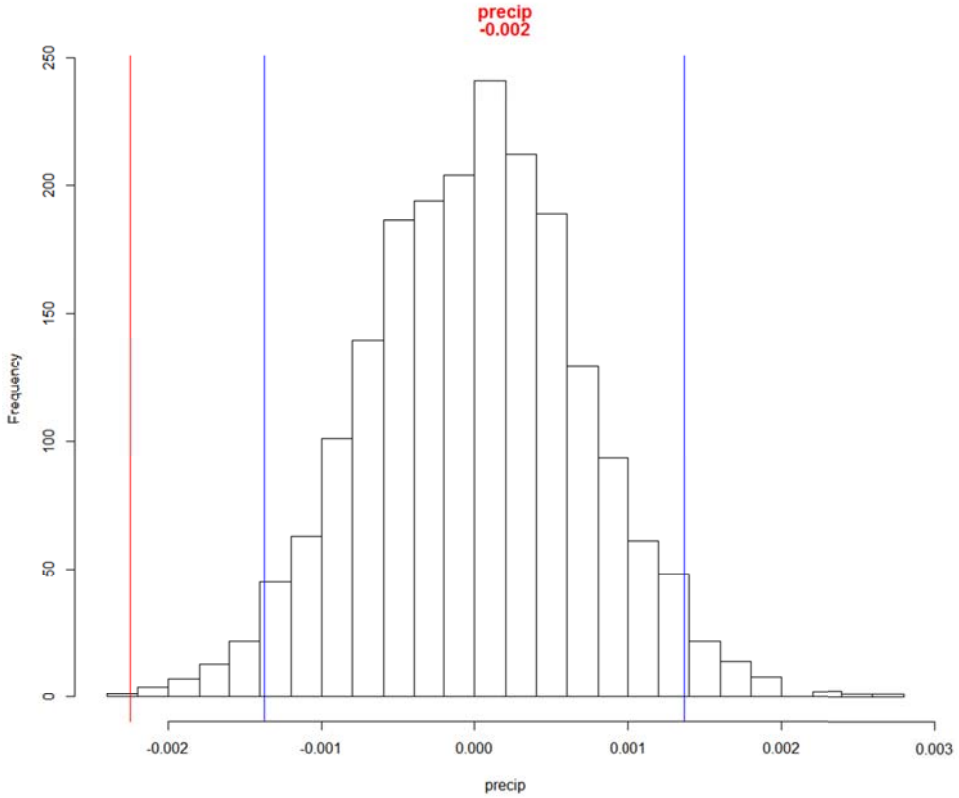
```
## COMPROBAD QUE LOS VALORES INDICADOS QUEDEN FUERA DE LOS INTERVALOS

(Intercept)
  -2.716939
      0.5%       2.5%         5%        95%      97.5%      99.5%
-1.7619034 -1.4197261 -1.2523775  0.5634688  0.7157352  1.0117427
[1] --------------------------------------------------
     altmed
-0.001218111
         0.5%          2.5%          5%          95%         97.5%         99.5%
-0.0012025847 -0.0009404258 -0.0007920339  0.0007675150  0.0009500432  0.0012002300
[1] --------------------------------------------------
   rangoalt
0.002129357
         0.5%          2.5%          5%          95%         97.5%         99.5%
-0.0010492709 -0.0007658210 -0.0006552421  0.0005570060  0.0006892047  0.0009206919
[1] --------------------------------------------------
  shannon
0.6068703
      0.5%       2.5%         5%        95%      97.5%      99.5%
-0.5207681 -0.3696765 -0.3142428  0.3509661  0.4259377  0.5548709
[1] --------------------------------------------------
  tempmin
0.4000696
      0.5%       2.5%         5%        95%      97.5%      99.5%
-0.1914029 -0.1301868 -0.1100756  0.1103365  0.1296641  0.1673672
[1] --------------------------------------------------
     precip
-0.002250147
        0.5%          2.5%          5%          95%         97.5%         99.5%
-0.001904127 -0.001373732 -0.001154112  0.001179926  0.001365615  0.001830063
[1] --------------------------------------------------
  D2%
36.01
     0.5%      2.5%        5%       95%     97.5%     99.5%
0.0666268 0.1644978 0.2197795 1.9373777 2.2770279 2.9785709
[1] --------------------------------------------------
```

El proceso se puede “complicar” un poco más y hacerse más “académico” introduciendo los valores obtenidos en el modelo real dentro de los 2000 valores nulos generados. Pero para entenderlo de manera sencilla, mejor asimilamos primero esta opción.

Esta figura es para la devianza explicada (por el `modelo` con los datos reales) que es del **36.01%**. Este valor queda muy a la derecha de la distribución nula, por lo que el modelo sería mayúsculamente significativo. Las líneas azules verticales denotan los percentiles 2.5% y 97.5%.



Y esto es para el coeficiente de la **precip**itación en el `modelo`, que vale realmente **-0.002** (representado como número y línea vertical roja).



Esta es una manera más "honesta" de obtener significaciones si nuestros datos muestran "problemas".

Y ESTOS SON TODOS LOS PASOS QUE PODEMOS DAR PARA CONSTRUIR MODELOS GENERALIZADOS LINEALES

MODELO GENERALIZADO LINEAL ASUMIENDO QUE EXISTE "INFLADO" DE CEROS.

Pudiera darse la circunstancia de que el fenómeno que estudiamos (distribución invernal de un pajarillo) fuese la consecuencia de dos respuestas combinadas:

(1) ¿qué determina que esté presente? [NO vs. SÍ, o 0 vs. 1]
(2) cuando está presente … ¿qué determina la variación de la abundancia relativa? [en la parte del conteo 1, 2, 3, 4, …]

Resolver estas cuestiones es asumir que en nuestra variable respuesta se mezclan dos tipos de distribuciones: una **binomial** que define los estadíos [**0 vs. ≥1**] y otra de **conteo "truncado"** eliminando el valor ausencia = 0 [**1, 2, 3, 4, 5, …**] (e.g., asumiendo una Poisson o una Binomial Negativa). Y cada una de estas respuestas lleva implícitos sus determinantes.

En vez de efectuar el análisis en dos pasos (i.e., 1º GLM con binomial y 2º GLM con una Poisson) contamos con una herramienta estadística que nos permite hacer los dos análisis combinados en un solo modelo. Esta herramienta se denomina *zero-inflated hurdle regression model*.

Para abordarlo, necesitamos cargar una nueva librería denominada `pscl`.

```
library(pscl)
```

A continuación definimos de nuevo la **ecuación del modelo**, lo cual podemos hacer de dos modos diferentes pero equivalentes:

```
eqt.h <- as.formula(ptyrup ~ altmed+rangoalt+shannon+tempmin+precip)
eqt.h <- as.formula(ptyrup ~ altmed+rangoalt+shannon+tempmin+precip | altmed+rangoalt+shannon+tempmin+precip)
```

En la segunda opción, antes de `|` definimos la parte del conteo, y después la parte denominada *zero hurdle model.* La parte del conteo está condicionada a la parte binomial que analiza [**0 vs. ≥1**]. Si las dos partes del modelo (conteo truncado `|` *zero hurdle*) debieran contener distintas variables predictoras, entonces optamos por la segunda opción de parametrización de la función de predictoras **g(X)**.

Para terminar, **definimos nuestro modelo**. Pero como no tenemos una idea clara de si la parte de conteo se ajusta mejor a una Poisson o a una Binomial Negativa, generamos dos modelos definiendo en el argumento `dist="…"` cada una de las dos distribuciones.

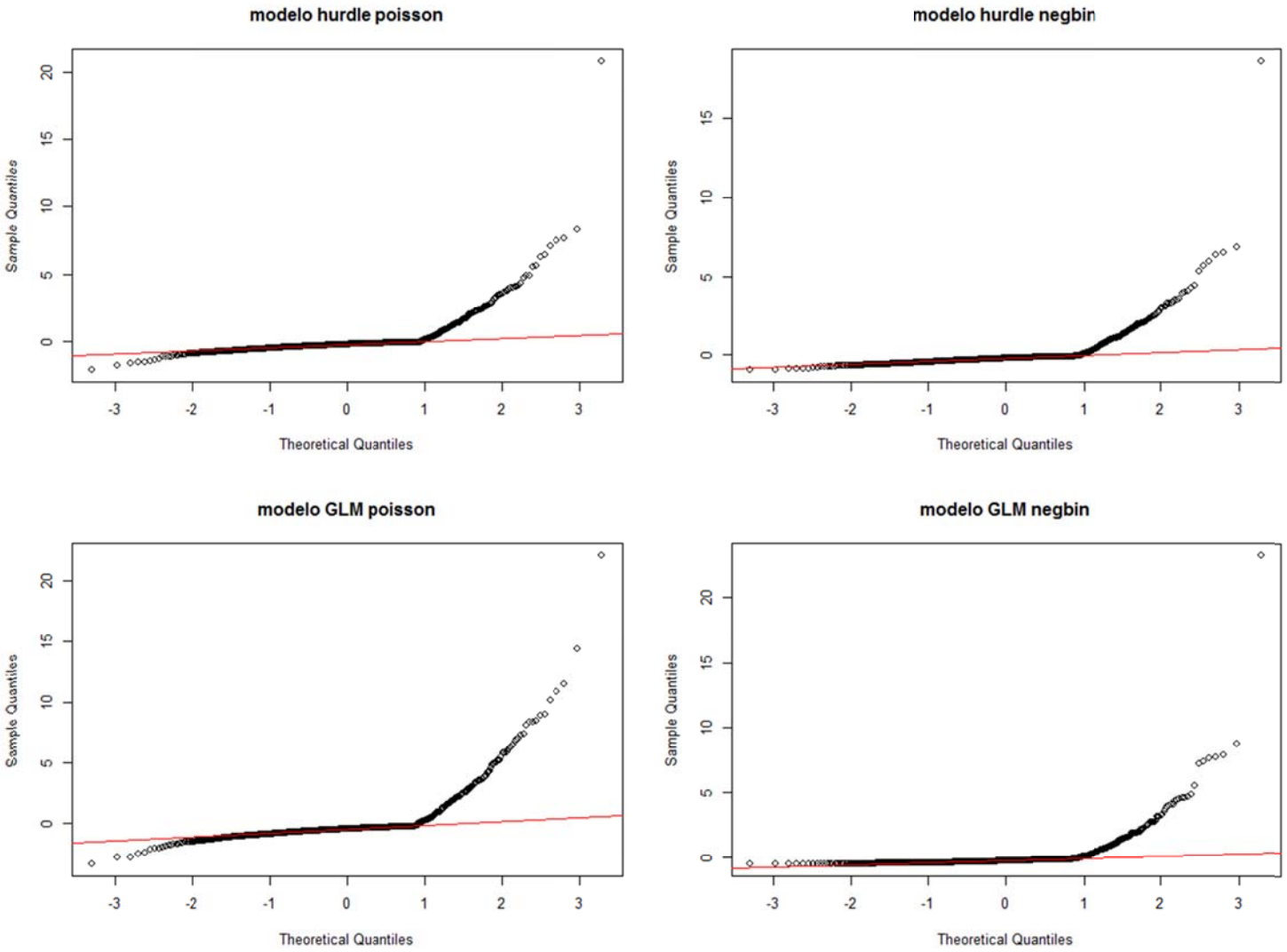
```
## asumiendo una Poisson para la parte de conteo "no cero"
modelo.hp <- hurdle(eqt.h, data=datos, dist="poisson", zero.dist="binomial")

## con una Binomial Negativa para la parte de conteo "no cero"
modelo.hnb <- hurdle(eqt.h, data=datos, dist="negbin", zero.dist="binomial")
```

También vamos a crear, para contraste con el modelo hurdle, dos modelos de "conteos" generalizados lineales asumiendo una distribución Poisson y una Binomial Negativa para la variable respuesta (de hecho, este segundo modelo es el que acabamos de presentar en esta exposición práctica).

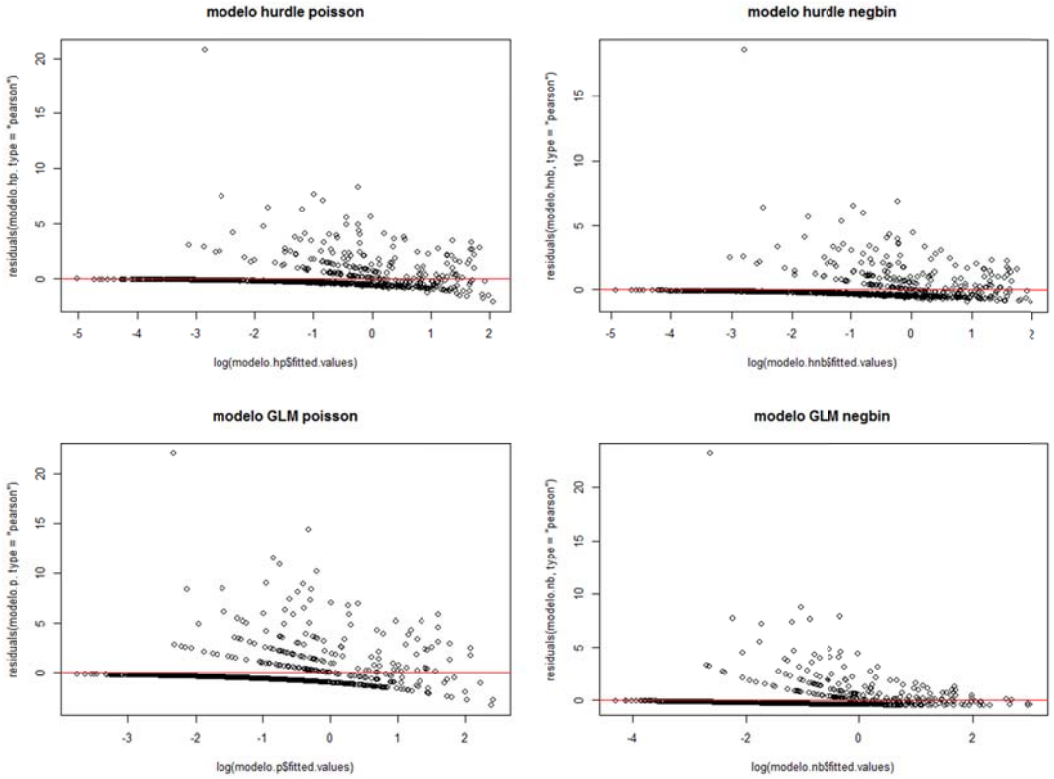
```
modelo.p <- glm(eqt, data=datos, family=poisson(link="log"))      ## modelo de poisson para comparación
modelo.nb <- glm.nb(eqt, data=datos, link=log)                    ## modelo binomial negativo para comparación
```

De nuevo, efectuaríamos todas las **exploraciones de los supuestos canónicos de los modelos** hurdle (`modelo.hp` y `modelo.hnb`) y su comparación con los GLM de conteos. Por ejemplo:

```
## normal probability plots de los residuos de devianza (de Pearson) del modelo (el comando hurdle no proporciona residuos de devianza)
par(mfcol=c(1,1))
par(mfcol=c(2,2))
qqnorm(residuals(modelo.hp, type="pearson"), main="modelo hurdle poisson")
qqline(residuals(modelo.hp, type="pearson"), col="red")
qqnorm(residuals(modelo.p, type="pearson"), main="modelo GLM poisson")
qqline(residuals(modelo.p, type="pearson"), col="red")
qqnorm(residuals(modelo.hnb, type="pearson"), main="modelo hurdle negbin")
qqline(residuals(modelo.hnb, type="pearson"), col="red")
qqnorm(residuals(modelo.nb, type="pearson"), main="modelo GLM negbin")
qqline(residuals(modelo.nb, type="pearson"), col="red")
par(mfcol=c(1,1))
```


modelo hurdle poisson
Sample Quantiles
Theoretical Quantiles
modelo hurdle negbin
Sample Quantiles
Theoretical Quantiles
modelo GLM poisson
Sample Quantiles
Theoretical Quantiles
modelo GLM negbin
Sample Quantiles
Theoretical Quantiles

```
## para la exploración de la heterocedasticidad (homogeneidad de la variación de los residuos a lo largo de las predicciones del modelo)
par(mfcol=c(1,1))
par(mfcol=c(2,2))
plot(log(modelo.hp$fitted.values),residuals(modelo.hp, type="pearson"), main="modelo hurdle poisson")
abline(h=0, col="red")
plot(log(modelo.p$fitted.values),residuals(modelo.p, type="pearson"), main="modelo GLM poisson")
abline(h=0, col="red")
plot(log(modelo.hnb$fitted.values),residuals(modelo.hnb, type="pearson"), main="modelo hurdle negbin")
abline(h=0, col="red")
plot(log(modelo.nb$fitted.values),residuals(modelo.nb, type="pearson"), main="modelo GLM negbin")
abline(h=0, col="red")
par(mfcol=c(1,1))
```



Y así … con otras pruebas vinculadas a los efectos “perdidos” e “influyentes” de los **datos uno-a-uno**.

En general, se observa que los qqplots son “similarmente” malos en los cuatro modelos. Pero respecto a la exploración de la heterocedasticidad de los residuos, los **modelos hurdle** son mejores que los GLM Poisson o GLM Binomial Negativa.

En cualquier caso, contamos con una herramienta para comparar la idoneidad de los modelos. Ésta es el **test de Vuong**. Vamos a comparar en parejas los modelos que comparten la misma familia de distribución de la variable respuesta:

```
# test de modelos poisson (Hurdle vs stándard)
vuong(modelo.p, modelo.hp)          ## Vuong Non-Nested Hypothesis Test
[1] -20.72026
          Raw AIC-corrected BIC-corrected
    -6.844198     -6.844049     -6.332877
[1] 1.282092
[1] -277.3476 -277.3416 -256.6273
Vuong Non-Nested Hypothesis Test-Statistic:
(test-statistic is asymptotically distributed N(0,1) under the
 null that the models are indistinguishible)
-------------------------------------------------------------
              Vuong z-statistic             H_A    p-value
Raw                   -6.844198 model2 > model1 3.8453e-12
AIC-corrected         -6.844049 model2 > model1 3.8493e-12
BIC-corrected         -6.332877 model2 > model1 1.2032e-10

## test de modelos binomiales negativos (Hurdle vs stándard)
vuong(modelo.nb, modelo.hnb)        ## Vuong Non-Nested Hypothesis Test
[1] -20.72026
          Raw AIC-corrected BIC-corrected
   -3.3848982    -3.3841669    -0.8618477
[1] 0.2598284
[1] -27.798090 -27.792084  -7.077826
Vuong Non-Nested Hypothesis Test-Statistic:
(test-statistic is asymptotically distributed N(0,1) under the
 null that the models are indistinguishible)
-------------------------------------------------------------
              Vuong z-statistic             H_A    p-value
Raw                   -3.3848982 model2 > model1 0.00035602
AIC-corrected         -3.3841669 model2 > model1 0.00035697
BIC-corrected         -0.8618477 model2 > model1 0.19438566
```

Claramente son mejores los resultados de los **modelos hurdle** que aquellos de los modelos GLM estándars. Como podemos ver, son mejores los segundos modelos (`model2`) que hemos definido como hurdle (`modelo.hp` y `modelo.hnb`), que los primeros (`model1`) que hemos definido como de conteos (`modelo.p` y `modelo.nb`).

En relación con la **variabilidad explicada en la respuesta por los dos modelos hurdle** … no hay claras diferencias:

```
print(c("R2% obs-pred =",round(100*cor(modelo.hp$y,fitted(modelo.hp))^2,3),"modelo hurdle poisson"), quote=FALSE)
[1] R2% obs-pred =       27.996                modelo hurdle poisson

print(c("R2% obs-pred =",round(100*cor(modelo.hnb$y,fitted(modelo.hnb))^2,3),"modelo hurdle negbin"), quote=FALSE)
[1] R2% obs-pred =       27.444                modelo hurdle negbin
```

Ahora veamos cómo gestionan los modelos hurdle la **sobredispersión en los resultados de los modelos**:

```
## índices de sobredispersión
print(c("sobredispersión modelo poisson =",sum(residuals(modelo.p, type="pearson")^2)/modelo.p0$df.residual), quote=FALSE)
[1] sobredispersión modelo poisson = 3.2164783697132

print(c("sobredispersión hurdle poisson  =",sum(residuals(modelo.hp, type="pearson")^2)/modelo.p$df.residual), quote=FALSE)
[1] sobredispersión hurdle poisson = 1.55652963720692

print(c("sobredispersión modelo negbinom =",sum(residuals(modelo.nb, type="pearson")^2)/modelo.nb0$df.residual), quote=FALSE)
[1] sobredispersión modelo negbinom = 1.52271402520681

print(c("sobredispersión hurdle negbinom =",sum(residuals(modelo.hnb, type="pearson")^2)/modelo.nb$df.residual), quote=FALSE)
[1] sobredispersión hurdle negbinom = 1.09502827641213
```

Claramente es el modelo hurdle-binomial negativo el mejor, conjuntamente con las exploraciones previas (aunque de modo sutil, como podemos ver por comparación mirando los panales gráficos superiores derechos previos).

Seleccionamos el modelo hurdle-binomial negativo el mejor y continuamos sólo con él el resto del análisis.

Primero el ***omnibus test*** general de la significación global del modelo.

```
lrtest(modelo.hnb)
Likelihood ratio test

Model 1: ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
Model 2: ptyrup ~ 1
  #Df  LogLik  Df  Chisq Pr(>Chisq)
1  13 -809.78
2   3 -946.80 -10 274.04  < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

También lo efectuamos mediante el test de Wald para controlar el efecto indeseable de la heterocedasticidad observada en los residuos del modelo.

```
waldtest(modelo.hnb, vcov=sandwich)
Wald test

Model 1: ptyrup ~ altmed + rangoalt + shannon + tempmin + precip
Model 2: ptyrup ~ 1
  Res.Df  Df  Chisq Pr(>Chisq)
1    986
2    996 -10 202.41  < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Globalmente nuestro modelo es muy significativo y ahora ya podemos pasar a considerar los efectos individuales de las variables predictoras (efectos parciales).

```
summary(modelo.hnb)

Call:
hurdle(formula = eqt.h, data = datos, dist = "negbin", zero.dist = "binomial")

Pearson residuals:
    Min      1Q  Median      3Q     Max
-0.9640 -0.3749 -0.2207 -0.1235 18.6547

Count model coefficients (truncated negbin with log link):
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.8718200  0.6283729  -1.387   0.1653
altmed       0.0002805  0.0004390   0.639   0.5228
rangoalt     0.0002855  0.0003594   0.795   0.4269
shannon      0.2266319  0.2599112   0.872   0.3832
tempmin      0.2802575  0.0533204   5.256 1.47e-07 ***
precip      -0.0013692  0.0005541  -2.471   0.0135 *
Log(theta)  -0.1823336  0.3513469  -0.519   0.6038         ## esto es el logaritmo neperiano de Theta

Zero hurdle model coefficients (binomial with logit link):
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.8661538  0.5440286  -5.268 1.38e-07 ***
altmed      -0.0018786  0.0004349  -4.320 1.56e-05 ***
rangoalt     0.0026264  0.0003570   7.358 1.87e-13 ***
shannon      0.5905156  0.2145080   2.753  0.00591 **
tempmin      0.3412101  0.0599982   5.687 1.29e-08 ***
precip      -0.0024783  0.0005907  -4.195 2.73e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Theta: count = 0.8333
Number of iterations in BFGS optimization: 16
Log-likelihood: -809.8 on 13 Df
```

Aparecen dos grupos de coeficientes, unos para la parte del conteo truncada que no considera los ceros (**`truncated negbin with log link`**) y otra para la parte binomial que analiza [**0 vs. ≥1**] (**`binomial with logit link`**). Vemos que en el caso del avión roquero, lo principalmente determinante de su distribución es lo que ocurre a nivel de presencia vs. ausencia (`Zero hurdle model coefficients`). Una vez que está presente la especie (`Count model coefficients`), parece que sólo afecta a su abundancia relativa la temperatura mínima invernal (de modo positivo y muy significativo) y la precipitación (de modo negativo).

Para controlar el problema potencialmente introducido por la heterocedasticidad de los residuos sobre las estimas de significación, podemos efectuar otro test que tiene en cuenta la violación de este supuesto teniendo en cuenta la **corrección sandwich**.

```
## ... si hay heterocedasticidad:
> coeftest(modelo.hnb, vcov=sandwich)

t test of coefficients:

                      Estimate  Std. Error t value  Pr(>|t|)
count_(Intercept) -0.87182004  0.66531411 -1.3104  0.190370
count_altmed       0.00028054  0.00037016  0.7579  0.448688
count_rangoalt     0.00028554  0.00034203  0.8348  0.404013
count_shannon      0.22663188  0.25138181  0.9015  0.367519
count_tempmin      0.28025754  0.04910359  5.7075 1.516e-08 ***
count_precip      -0.00136917  0.00052327 -2.6166  0.009018 **

zero_(Intercept)  -2.86615379  0.53205987 -5.3869 8.966e-08 ***
zero_altmed       -0.00187860  0.00047284 -3.9730 7.614e-05 ***
zero_rangoalt      0.00262639  0.00038372  6.8445 1.347e-11 ***
zero_shannon       0.59051562  0.20512915  2.8788  0.004079 **
zero_tempmin       0.34121011  0.06683824  5.1050 3.968e-07 ***
zero_precip       -0.00247826  0.00059071 -4.1954 2.970e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Como podemos ver por comparación con la salida de `summary(modelo.hnb)`, no cambian los coeficientes de regresión. Sólo cambian … ¡y poco!, las significaciones.

Como existen puntos influyentes y perdidos que no tenemos motivos para eliminar de nuestro análisis, podemos efectuar una estima robusta de nuestros resultados mediante “*bootstrapping*”.

Para ello podemos correr las siguientes líneas de código.

```
## asignar uno de los modelos previos de interés a un nuevo objeto llamado "modelo" con el que las siguientes líneas trabajará.
modelo <- modelo.hnb

## seleccionar SÓLO una de las dos líneas siguientes de acuerdo con el modelo de interés   ¡¡CUIDADO!!
nv <- modelo$df.null-modelo$df.residual+2        ## correr esta línea par hurdle poisson
nv <- modelo$df.null-modelo$df.residual+1        ## correr esta línea par hurdle negative binomial
                                                    (debido al cálculo del parámetro theta)
```

Corremos solamente la segunda línea, porque la parte de conteo en la Binomial Negativa estima un parámetro más (`Log(theta)`). Se generan mil procesos definidos en **`nrow=1000`** y en `for (i in 1:1000)`.

```
## seleccionar desde ESTA LÍNEA hasta ###### HASTA AQUÍ ###### y corredlo con [ctrl]+[enter]
mboot <- matrix(99999, nrow=1000, ncol=nv)
for (i in 1:1000)  {
     iboot <- sample(1:nrow(datos), replace = TRUE)
     bootdatos <- datos[iboot,]
     mn <- hurdle(eqt, data=bootdatos, dist="poisson", zero.dist="binomial")
     cr <- as.vector(coef(mn))
     mboot[i,1:nv] <- cr
 }
##
mmboot <- as.data.frame(mboot)    ## para convertir en un data.frame "mmboot" el atomic vector "mboot"
###### HASTA AQUÍ ######
```

Y a continuación vemos los resultados obtenidos corriendo los dos siguientes grupos de líneas de código que hacen referencia a la parte del conteo y a la parte binomial. Tanto en los resultados textuales (de valores medios remuestreados de los parámetros y la definición de sus cuantiles), como en los gráficos de histogramas, el valor CERO (hipótesis nula de ausencia de efectos para cada coeficiente, Ho = 0) debe quedar fuera de los límites de confianza al 5% de los coeficientes resmuestreados.

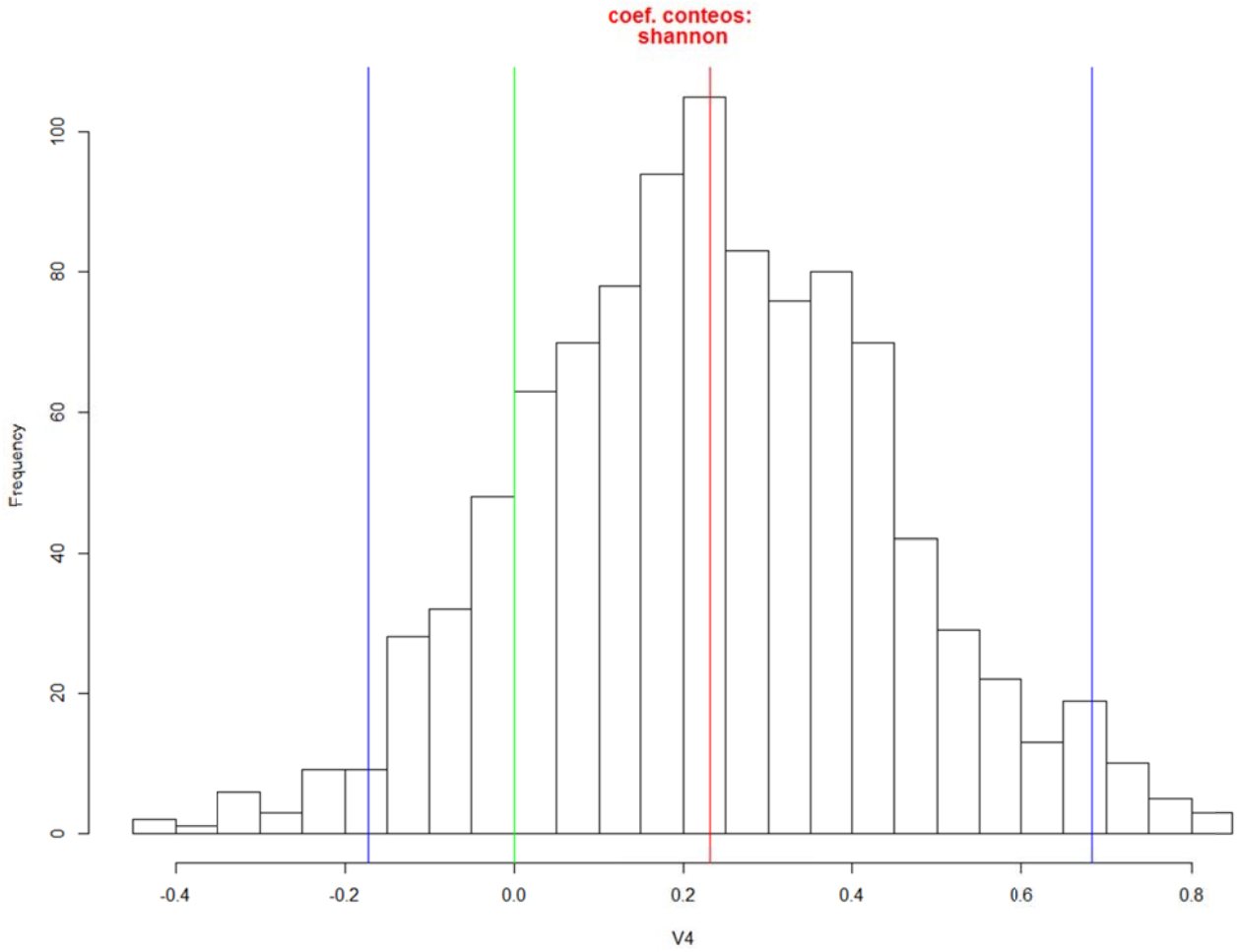
```
## COMPROBAD QUE LOS INTERVALOS NO INCLUYAN EL VALOR CERO
nn <- length(mmboot)

print("COEFICIENTES DE LA PARTE DEL CONTEO")
for (i in 1:(nn/2)) {
   print(c("coef. conteos: ",names(mn$model)[i]), quote=FALSE)
   print(c("media=",mean(mmboot[,i])), quote=FALSE)
   print(quantile(mmboot[,i], c(0.005, 0.01, 0.025, 0.05, 0.95, 0.975, 0.99, 0.995)))
   print("-----------------------------------------------------", quote=FALSE)
   hist(mmboot[,i], breaks=20, xlab=names(mmboot[i]), main=c("coef. conteos: ",names(mn$model)[i]), col.main="red")
   abline(v=mean(mmboot[,i]), col="red")              ## en ROJO la media de los valores remuestreados
   abline(v=quantile(mmboot[,i], 0.025), col="blue")   ## en AZUL los límites a alfa = 0.05 de los valores remuestreados
   abline(v=quantile(mmboot[,i], 0.975), col="blue")
   abline(v=0, col="green")                           ## en VERDE el valor cero indicativo de la Ho (hipótesis nula)
 }
```

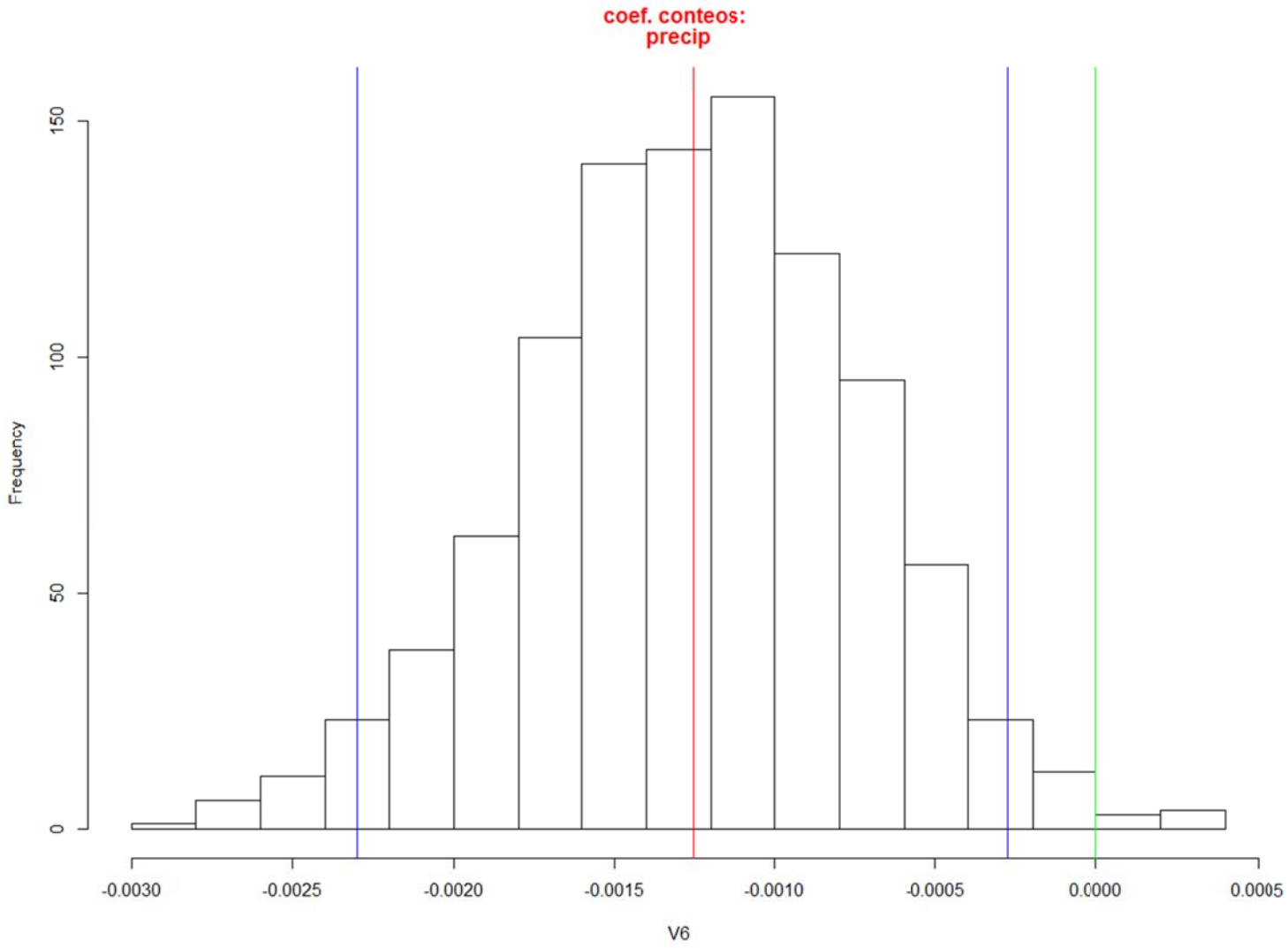
```
[1] "COEFICIENTES DE LA PARTE DEL CONTEO"
[1] coef. conteos:  ptyrup
[1] media=            -0.273229677166351
      0.5%         1%       2.5%         5%        95%      97.5%        99%      99.5%
-1.5103478 -1.3995188 -1.1690232 -1.0500794  0.4474149  0.5694279  0.6823271  0.7900930
[1] -----------------------------------------------------
[1] coef. conteos:  altmed
[1] media=             3.42375039430522e-05
         0.5%          1%         2.5%          5%         95%       97.5%         99%       99.5%
-0.0009175151 -0.0008573834 -0.0006723630 -0.0005291317  0.0005529764  0.0006199495  0.0007526139  0.0008687702
[1] -----------------------------------------------------
[1] coef. conteos:  rangoalt
[1] media=             0.000329363264066105
         0.5%          1%         2.5%          5%         95%       97.5%         99%       99.5%
-2.982685e-04 -2.768187e-04 -1.582090e-04 -8.443112e-05  7.754315e-04  8.753365e-04  1.013545e-03  1.080678e-03
[1] -----------------------------------------------------
[1] coef. conteos:  shannon
[1] media=            0.231719575312683
      0.5%         1%       2.5%         5%        95%      97.5%        99%      99.5%
-0.3362909 -0.2722159 -0.1728024 -0.1135466  0.5998151  0.6836245  0.7345618  0.7707065   el CERO se incluye: NO SIGNIFICATIVO
[1] -----------------------------------------------------
[1] coef. conteos:  tempmin
[1] media=            0.239100847994249
     0.5%        1%       2.5%         5%        95%      97.5%        99%      99.5%
0.1408635 0.1500386 0.1620691 0.1776081 0.3016931 0.3144793 0.3255219 0.3311189        el CERO no se incluye: SÍ SIGNIFICATIVO
[1] -----------------------------------------------------
[1] coef. conteos:  precip
[1] media=            -0.00125385823245497
         0.5%          1%         2.5%          5%         95%       97.5%         99%       99.5%
-2.660667e-03 -2.459294e-03 -2.300031e-03 -2.110693e-03 -4.460321e-04 -2.780161e-04 -8.191649e-05  4.026728e-05
```

```
print("COEFICIENTES DE LA PARTE BINOMIAL")
for (i in (nn/2+1):nn) {
   print(c("coef. binomial: ",names(mn$model)[i-nn/2]), quote=FALSE)
   print(c("media=",mean(mmboot[,i])), quote=FALSE)
   print(quantile(mmboot[,i], c(0.005, 0.01, 0.025, 0.05, 0.95, 0.975, 0.99, 0.995)))
   print("-----------------------------------------------------", quote=FALSE)
   hist(mmboot[,i], breaks=20, xlab=names(mmboot[i]), main=c("coef. binomial: ",names(mn$model)[i-nn/2]), col.main="red")
   abline(v=mean(mmboot[,i]), col="red")                    ## en ROJO la media de los valores remuestreados
   abline(v=quantile(mmboot[,i], 0.025), col="blue")        ## en AZUL los límites a alfa = 0.05 de los valores remuestreados
   abline(v=quantile(mmboot[,i], 0.975), col="blue")
   abline(v=0, col="green")                                 ## en VERDE el valor cero indicativo de la Ho (hipótesis nula)
 }
```

```
[1] "COEFICIENTES DE LA PARTE BINOMIAL"
[1] coef. binomial:  ptyrup
[1] media=            -2.87318080346958
     0.5%        1%       2.5%        5%       95%     97.5%       99%     99.5%
-4.397650 -4.113431 -3.946231 -3.776302 -1.997648 -1.825520 -1.653355 -1.580542
[1] -----------------------------------------------------
[1] coef. binomial:  altmed
[1] media=               -0.00190431624933658
         0.5%            1%          2.5%            5%           95%         97.5%           99%         99.5%
-0.0032284437 -0.0031746012 -0.0029268504 -0.0027763091 -0.0010649420 -0.0008938460 -0.0007308729 -0.0005105686
[1] -----------------------------------------------------
[1] coef. binomial:  rangoalt
[1] media=              0.00262786487617474
       0.5%          1%        2.5%          5%          95%       97.5%         99%       99.5%
0.001618630 0.001810364 0.001923074 0.002031892 0.003299746 0.003406380 0.003495623 0.003529329
[1] -----------------------------------------------------
[1] coef. binomial:  shannon
[1] media=            0.608075310789511
      0.5%         1%       2.5%         5%        95%      97.5%        99%      99.5%
0.09409838 0.11185360 0.17951801 0.25144068 0.95523830 1.03421902 1.10932989 1.16362103
[1] -----------------------------------------------------
[1] coef. binomial:  tempmin
[1] media=            0.341976745083984
     0.5%        1%       2.5%        5%       95%     97.5%       99%     99.5%
0.1699465 0.1867451 0.2112198 0.2368426 0.4585321 0.4795653 0.5088630 0.5220318
[1] -----------------------------------------------------
[1] coef. binomial:  precip
[1] media=               -0.00255031979190563
        0.5%           1%         2.5%           5%          95%        97.5%          99%        99.5%
-0.004095427 -0.003938371 -0.003763861 -0.003521834 -0.001609535 -0.001460947 -0.001300968 -0.001134792
[1] -----------------------------------------------------
```

Ejemplo de dos resultados gráficos con histogramas de efectos NO (primera figura) y Sí (segunda figura) significativos (mirad la raya vertical verde de Ho = 0):




coef. conteos:
precip
Frequency
0
50
100
150
-0.0030
-0.0025
-0.0020
-0.0015
-0.0010
-0.0005
0.0000
0.0005
V6

Y por último, una síntesis tabulada de los resultados numéricos:

```
## SÍNTESIS DE LOS RESULTADOS DEL BOOTSTRAPPING (siguiendo el mismo orden que la salida de summary(modelo...))
## ¡¡¡ OJO !!! En los datos "bootstrapped" el standard error es la sd (desv. típica)
print(describe(mmboot)[c(1:4, 8:9, 11:12)], digits=5)
    vars    n     mean      sd      min      max     skew kurtosis
v1     1 1000 -0.27323 0.44657 -2.03557  1.41500 -0.13826  0.43490
v2     2 1000  0.00003 0.00033 -0.00125  0.00113 -0.29186  0.46378
v3     3 1000  0.00033 0.00027 -0.00054  0.00125  0.22276  0.15577
v4     4 1000  0.23172 0.21433 -0.42897  0.84504  0.05141 -0.01654
v5     5 1000  0.23910 0.03850  0.09460  0.40171  0.03435  0.28544
v6     6 1000 -0.00125 0.00052 -0.00287  0.00035 -0.08268  0.02794
  Aquí se excluye a Log(theta) de la parte del conteo (porque se obtienen SÓLO los coeficientes de las predictoras)
v7     7 1000 -2.87318 0.53523 -4.93494 -1.23919 -0.14741  0.25580
v8     8 1000 -0.00190 0.00052 -0.00390 -0.00019 -0.02700  0.25720
v9     9 1000  0.00263 0.00038  0.00121  0.00369  0.02266  0.01083
v10   10 1000  0.60808 0.21171 -0.03040  1.23094 -0.01601 -0.06964
v11   11 1000  0.34198 0.06762  0.12778  0.56378  0.09701  0.01477
v12   12 1000 -0.00255 0.00059 -0.00485 -0.00036 -0.11318  0.13555
```

Vamos ahora a predecir con el modelo hurdle binomial negativo (`modelo.hnb`) los valores esperados en otro juego de datos teniendo en cuenta los valores de las variables predictoras. Ya lo hemos hecho antes con el `modelo` Binomial Negativo.

Para ello haremos uso del comando `predict`, y tendremos que cargar un nuevo juego de datos con esa información (en esta ocasión llamado “predecir”, con N=689). El tipo de la predicción lo establecemos en esta ocasión en la escala de medida original de la variable respuesta. Los datos predichos de la variable respuesta los llamo, como quiero: `predecir$ptyrup.nuevo.hnb`.

```
predecir$ptyrup.nuevo.hnb <- predict(modelo.hnb, newdata=predecir, type="response")
```

Como también contamos en el juego de datos `predecir` con la variable respuesta `ptyrup` medida realmente en el campo, podemos efectuar un test del poder predictivo del modelo `modelo.hnb`. Para ello, relacionamos los datos de la variable respuesta medidos pero no utilizados en `modelo.hnb` (`predecir$ptyrup`) con los predichos por él (`predecir$ptyrup.nuevo.hnb`).

```
summary(lm(predecir$ptyrup~predecir$ptyrup.nuevo.hnb))

Call:
lm(formula = predecir$ptyrup ~ predecir$ptyrup.nuevo.hnb)
Residuals:
    Min      1Q  Median      3Q     Max
-7.6190 -0.5809 -0.0588  0.1008 20.4305

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)               -0.16317    0.10123  -1.612    0.107
predecir$ptyrup.nuevo.hnb  1.33338    0.07026  18.977   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.208 on 687 degrees of freedom
Multiple R-squared:  0.3439,   Adjusted R-squared:  0.343
F-statistic: 360.1 on 1 and 687 DF,  p-value: < 2.2e-16
```

De nuevo, vemos que el poder predictivo del modelo es aceptable (R^2 = 34.4%), significativo (p<<0.001), pero en esta ocasión el modelo hurdle binomial negativo (`modelo.hnb`) tiende a sobreestimar la abundancia relativa del avión roquero (nuestra variable respuesta, `ptyrup`), porque la pendiente de `predecir$ptyrup.nuevo.hnb` es 1.333, mayor que el valor esperable de 1 si OBSERVADO fuese igual a PREDICHO:

```
(1-1.33338) / 0.07026        ## ésto es una t de Student de desvío de un coeficiente de regresión del valor 1
[1] -4.744947

2*(1-pt(4.744947, df=687))   ## cálculo de la p para una t de Student, con dos colas
[1] 2.538177e-06
```

Reconsideremos el modelo anterior desde la perspectiva de la naturaleza de la variable respuesta.

`datos$ptyrup` es el conteo, sobre un máximo efectuado de 60 transectos de 15 minutos de duración, de presencias de nuestra especie de estudio, la variable respuesta.

La hemos tratado como una **binomial negativa** de acuerdo a sus características numéricas.

Pero también la podríamos haber tratado como una frecuencia, dividiendo cada valor de `datos$ptyrup` por 60.

De este modo, la variable respuesta estaría acotada entre cero y uno (0 – 1).

Esto nos conduce a poder trabajar con otra familia de distribución de la variable respuesta:

La BINOMIAL, que sólo viene caracterizada por un parámetro “p” (probabilidad de que un fenómeno ocurra).

Y nos introduce en los **Modelos Generalizados Lineales Logísticos** que operan con frecuencias y cuya familia y función de vínculo son:

family = binomial(link="logit")

En esta situación, repetiríamos nuestro modelo (ahora llamado `modelo.bf`)

En esta ocasión necesitamos definir con el argumento `weight` la variable que define el denominador para calcular la frecuencia.

También podríamos trabajar con `family = binomial(link="cloglog")` si la distribución de las frecuencias de la variable respuesta está extremadamente sesgada al cero (e.g., promedio=0.1) o al uno (e.g., promedio=0.9).

Creamos la variable `denominador`:

```
denominador <- seq(60,60, length.out=999)      ## el valor más pequeño, el más grande, y el número de datos
[1] 60 60 60 60 60 60 60 60 ...
...
[961] ... 60 60 60 60 60
```

Creamos la fórmula:

```
eqt.bf <- as.formula(ptyrup/denominador ~ altmed+rangoalt+shannon+tempmin+precip)
```

Y construimos el modelo con el comando `glm`:

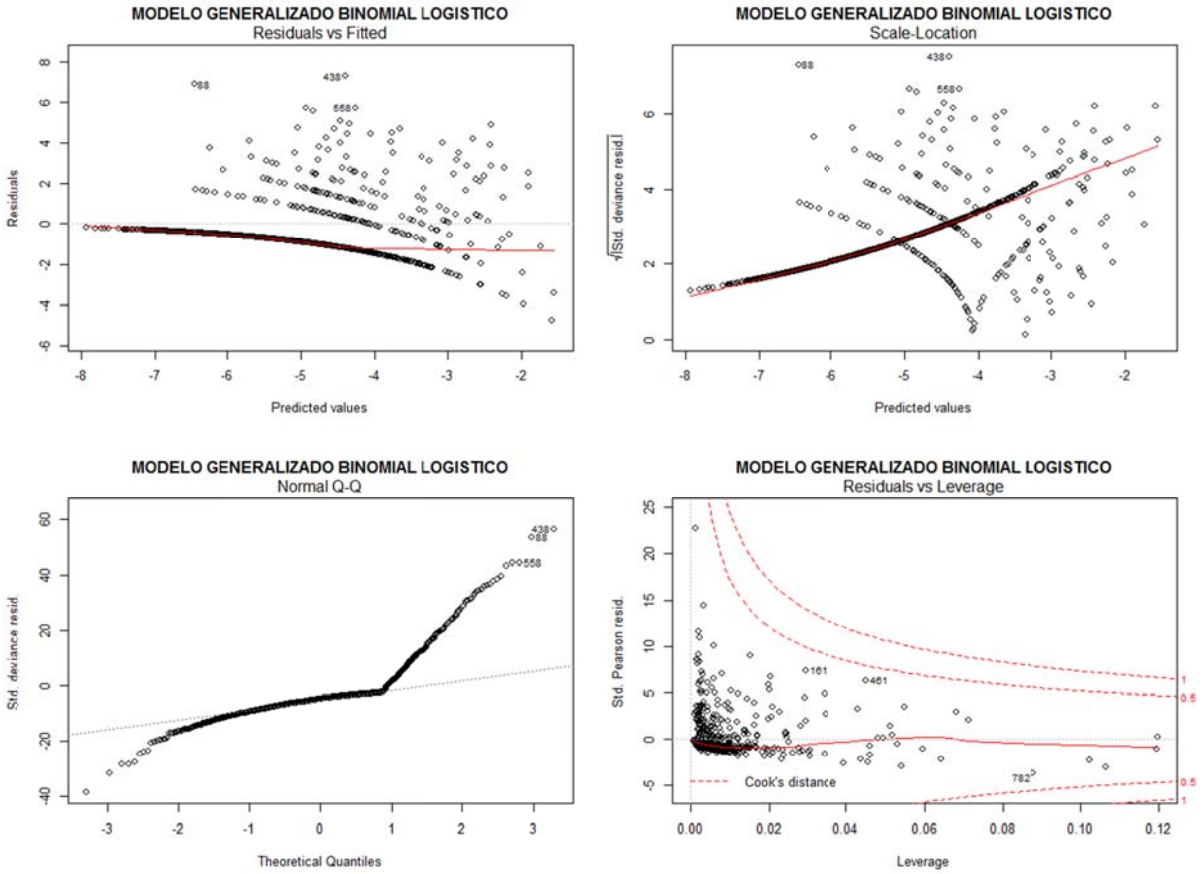
```
modelo.bf <- glm(eqt.bf, weight=denominador, data=datos, family=binomial(link="logit"))
```

Otra manera de construir el modelo sería crear una variable respuesta compuesta por dos variables que se combinan:
Primero pondríamos la variable que define las presencias de la especie de estudio: `ptyrup`
Luego otra variable que define los transectos en los que la especie estuvo ausente: `denominador-ptyrup`
Y ahora construimos el modelo (los dos, `modelo.bf` y `modelo.bf2` producen idénticos resultados):

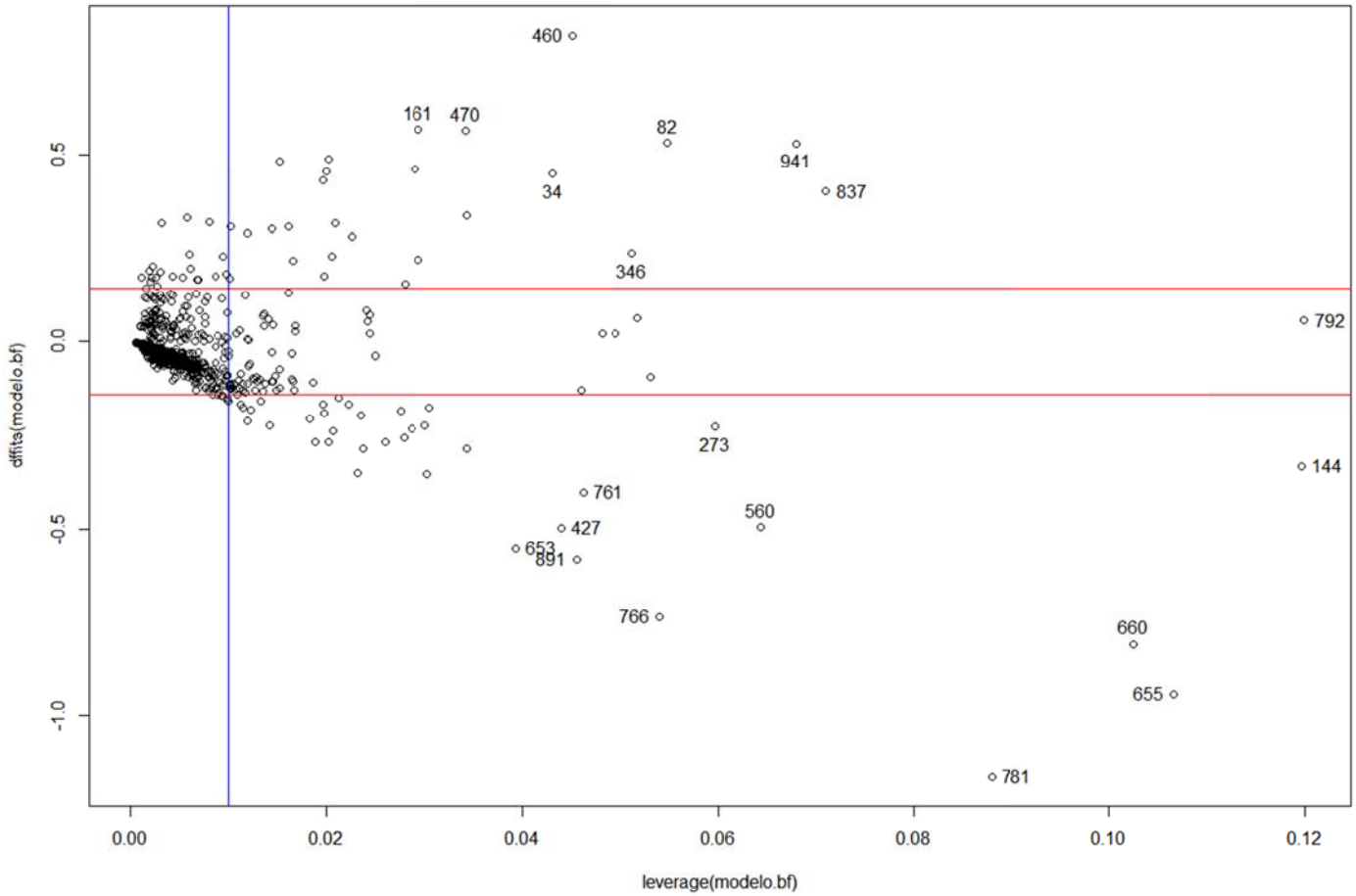
```
eqt.bf2 <- as.formula(cbind(ptyrup, denominador-ptyrup) ~ altmed+rangoalt+shannon+tempmin+precip)
modelo.bf2 <- glm(eqt.bf2, data=datos, family=binomial(link="logit"))
```

Repasemos los supuestos canónicos del modelo. Sólo unas pocas cosas añadiendo algo nuevo que no hemos visto antes con la binomial negativa. Seguimos observando “mala” normalidad de los residuos en devianza, y algo de heterocedasticidad en ellos.

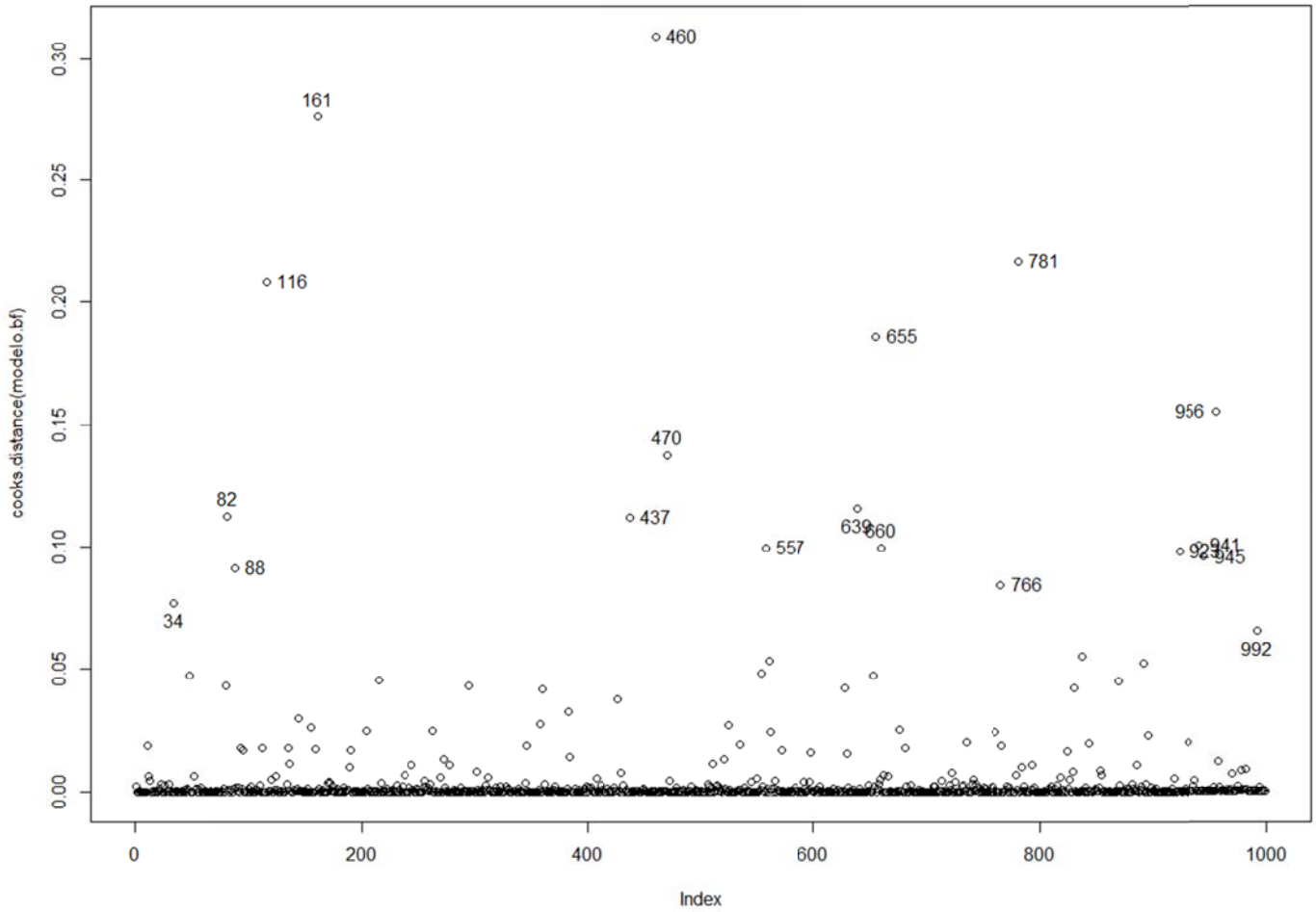
```
par(mfcol=c(1,1))    ## fija un sólo panel gráfico
par(mfcol=c(2,2))    ## fija cuatro paneles según 2 columnas y 2 filas
plot(modelo.bf, main="MODELO GENERALIZADO BINOMIAL LOGISTICO")
par(mfcol=c(1,1))    ## volvemos al modo gráfico de un solo panel
```



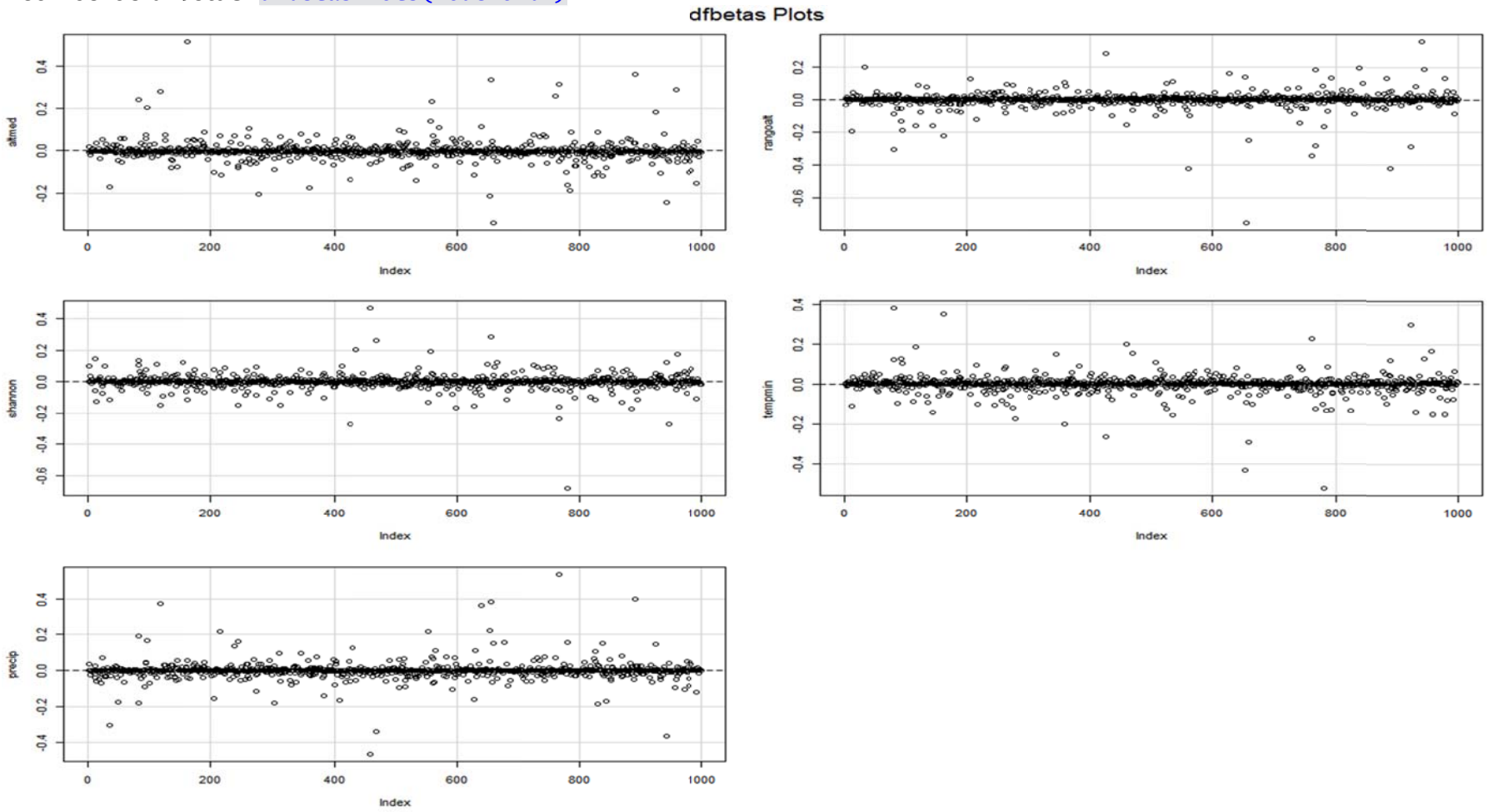
```
plot(dffits(modelo.bf)~leverage(modelo.bf))       ## en la siguiente línea se dan los umbrales  “críticos”
abline(h=(2*(5/999)^0.5), col="red"); abline(h=-(2*(5/999)^0.5), col="red"); abline(v=2*5/999, col="blue")
identify(dffits(modelo.bf)~leverage(modelo.bf))   ## y se marcan los datos más extremos
```



```
plot(cooks.distance(modelo.bf))
identify(cooks.distance(modelo.bf))
```



Veamos las **dfbetas**: `dfbetasPlots(modelo.bf)`



Fijaos que `shannon`, `altmed` y `rangoalt` tienen puntos que alteran mucho la estima de los coeficientes (lo veremos más tarde en las estimas robustas).

Efectuamos la estima de significación global del modelo:

```
lrtest(modelo.bf)
Likelihood ratio test

Model 1: ptyrup/denominador ~ altmed + rangoalt + shannon + tempmin +
    precip
Model 2: ptyrup/denominador ~ 1
  #Df  LogLik Df  Chisq Pr(>Chisq)
1   6 -1173.0
2   1 -1692.3 -5 1038.6  < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Y como también teníamos heterocedasticidad en los residuos en devianza, corregimos este desvío en los supuestos canónicos:

```
waldtest(modelo.bf, vcov=sandwich)
Wald test

Model 1: ptyrup/denominador ~ altmed + rangoalt + shannon + tempmin +
    precip
Model 2: ptyrup/denominador ~ 1
  Res.Df Df      F    Pr(>F)
1    993
2    998 -5 44.158 < 2.2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Estupendo, el modelo es significativo. Habiendo pasado este **ómnibus test**, ya podemos pasar a ver los resultados específicos para cada variable predictora.

Pero … ¡alto! Una variable respuesta Binomial viene caracterizada por un solo parámetro (p – probabilidad), sin definir la varianza.

Podría darse el caso de que los residuos en devianza (de Pearson en esta ocasión para estimar el coeficiente de sobredispersión) manifestasen más dispersión alrededor de las predicciones del modelo (usando la *link function*) que lo “conveniente”.

Para ello, antes de proseguir con las significaciones de las predictoras, estimamos el **coeficiente de sobredispersión (φ)**, que podríamos utilizar para recalcular las significaciones. **φ** debería ser = 1. Este valor **φ** permite convertir las Chi^2 derivadas de las diferencias de devianzas, en valores de F de Fisher.

Si **φ** es mayor que 1, hay **sobredispersión**. En este escenario, "inflamos" las significaciones (i.e., aumentamos la probabilidad de cometer el error de tipo I: rechazar la Ho cuando de hecho es cierta). Si **φ** < 1 hay infradispersión y estaríamos inflando el error de tipo II (aceptar la Ho cuando de hecho es falsa). Hay motivos filosóficos por los que se ha convenido en la necesidad de corregir las estimas de significación cuando **φ** > 1, pero NO cuando **φ** < 1.

Las correcciones teniendo en cuenta la sobredispersión MODIFICAN LAS SIGNIFICACIONES de las predictoras incluidas en el modelo, PERO NO ALTERAN LOS COEFICIENTES de regresión.

Calculamos el coeficiente de sobredispersión en nuestro modelo:

```
phi <- sum((residuals(modelo.bf, type="pearson"))^2)/modelo.bf$df.residual
print(c("Pearson overdispersion=",round(phi,3)), quote=FALSE)
[1] Pearson overdispersion= 3.289
```

φ vale **3.289**, indicando que hay bastante **sobredispersión**.

Al ser así, la petición de los resultados de nuestro modelo `summary(modelo.bf)` no sería realmente válida en sus significaciones, aunque sí en sus coeficientes.

Ante esto podemos hacer dos cosas. Crear un nuevo modelo en el que se incluye la corrección por sobredispersión en la familia binomial, utilizando `family=quasibinomial(link="logit")`, o incluir el valor de `phi` en nuestras estimas de significación.

Primeramente vamos a crear un **modelo "quasi"** que denominaremos `modelo.qbf`.

```
modelo.qbf <- glm(eqt.bf, weight=denominador, data=datos, family=quasibinomial(link="logit"))
```

Veamos comparadamente los resultados:

```
summary(modelo.bf)

Call:
glm(formula = eqt.bf, family = binomial(link = "logit"),
    data = datos,  weights = denominador)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.7313  -0.9651  -0.5806  -0.3508   7.2837

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.4290785  0.2328478 -27.611  < 2e-16 ***
altmed      -0.0009578  0.0001953  -4.905 9.35e-07 ***
rangoalt     0.0015643  0.0001358  11.518  < 2e-16 ***
shannon      0.4945327  0.0994725   4.972 6.64e-07 ***
tempmin      0.3909216  0.0238705  16.377  < 2e-16 ***
precip      -0.0021438  0.0002331  -9.196  < 2e-16 ***
---
Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2806.3  on 998  degrees of freedom
Residual deviance: 1767.7  on 993  degrees of freedom
AIC: 2358

Number of Fisher Scoring iterations: 6
```

```
summary(modelo.qbf)

Call:
glm(formula = eqt.bf, family = quasibinomial(link = "logit"),
    data = datos, weights = denominador)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.7313  -0.9651  -0.5806  -0.3508   7.2837

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -6.4290785  0.4223108 -15.224  < 2e-16 ***
altmed      -0.0009578  0.0003542  -2.704  0.00696 **
rangoalt     0.0015643  0.0002463   6.351 3.26e-10 ***
shannon      0.4945327  0.1804110   2.741  0.00623 **
tempmin      0.3909216  0.0432934   9.030  < 2e-16 ***
precip      -0.0021438  0.0004228  -5.070 4.73e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for quasibinomial family taken to be 3.289)

    Null deviance: 2806.3  on 998  degrees of freedom
Residual deviance: 1767.7  on 993  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 6
```

Como vemos (en rojo) hay sobredispersión en los datos, tal y como estimamos previamente calculando phi = **φ** = **3.289**, por lo que no podemos asumir “`Dispersion parameter for binomial family taken to be 1`”.

También podemos comprobar que **sólo cambian las significaciones** (¡y mucho! aunque sigan siendo muy significativas; valoradlo vosotros mismos comparando los valores de `Pr(>|t|)` y `Pr(>|z|)` para altmed y shannon especialmente. Sin embargo, no cambia nada más.

Otro manera de corregir la sobredispersión y llegar a la misma tabla de resultados que se obtiene aplicando `summary(…)` a un modelo quasibinomial es incluir un argumento dentro de `dispersion=…` aplicado a nuestro modelo original Binomial.

```
summary(modelo.bf, dispersion = 3.289426)      ## cualquiera de estas dos formas vale
summary(modelo.bf, dispersion = phi)           ## porque ϕ ya lo hemos calculado antes con phi

Call:
glm(formula = eqt.bf, family = binomial(link = "logit"), data = datos,
    weights = denominador)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.7313  -0.9651  -0.5806  -0.3508   7.2837

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.4290785  0.4223108 -15.224  < 2e-16 ***
altmed      -0.0009578  0.0003542  -2.704  0.00684 **
rangoalt     0.0015643  0.0002463   6.351 2.14e-10 ***
shannon      0.4945327  0.1804110   2.741  0.00612 **
tempmin      0.3909216  0.0432934   9.030  < 2e-16 ***
precip      -0.0021438  0.0004228  -5.070 3.97e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 3.289425)

    Null deviance: 2806.3  on 998  degrees of freedom   ## lo marcado con …… lo utilizaremos un poco más adelante
Residual deviance: 1767.7  on 993  degrees of freedom
AIC: 2358

Number of Fisher Scoring iterations: 6
```

Otra aproximación es utilizar el comando `dropterm` aplicado al modelo quasibinomial `modelo.qbf`.

Esta aproximación es **más aconsejable** si tenemos desvíos de los supuestos canónicos del modelo y poco tamaño muestral.

```
dropterm(modelo.qbf, test="Chisq", sorted=FALSE)

Single term deletions

Model:
ptyrup/denominador ~ altmed + rangoalt + shannon + tempmin +
    precip
         Df Deviance scaled dev.    Pr(Chi)
<none>        1767.7
altmed    1   1792.5       7.524    0.00609 **
rangoalt  1   1891.5      37.633 8.537e-10 ***
shannon   1   1793.3       7.784    0.00527 **
tempmin   1   2037.8      82.104 < 2.2e-16 ***
precip    1   1856.7      27.035 1.998e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Corrección de heterocedasticidad. Pero como además hay heterocedasticidad en nuestros residuos en devianza, deberíamos corregir este desvío de los supuestos canónicos de los buenos modelos generalizados lineales. Y se lo aplicamos al modelo `modelo.qbf` que ya tiene en cuenta el desvío de la dispersión del supuesto **φ** = 1.

```
coeftest(modelo.qbf, vcov=sandwich)

z test of coefficients:
               Estimate  Std. Error  z value  Pr(>|z|)
(Intercept) -6.42907846  0.45680942 -14.0739 < 2.2e-16 ***
altmed      -0.00095777  0.00042397  -2.2591   0.02388 *
rangoalt     0.00156432  0.00027722   5.6428 1.673e-08 ***
shannon      0.49453269  0.20214080   2.4465   0.01443 *
tempmin      0.39092157  0.04909906   7.9619 1.694e-15 ***
precip      -0.00214378  0.00053395  -4.0149 5.946e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

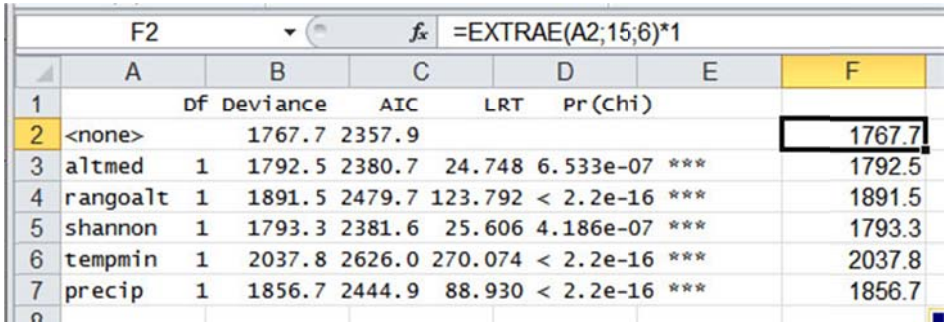
Vemos que los coeficientes no cambian, pero que las significaciones se han hecho más conservadoras.

Comparad, de nuevo, las `Pr(>|z|)` para **altmed** y **shannon**.

Con la salida previa de `dropterm(modelo.qbf, test="Chisq", sorted=FALSE)` podemos crear una tabla de partición de la variabilidad teniendo en cuenta cómo se reparte la devianza entre diferentes efectos y modelos. También vamos a considerar las devianzas obtenidas en `summary(modelo.bf, dispersion = phi)` y marcadas previamente con color amarillo (……).

Recordemos que efectuar un modelo con `family=binomial(link="logit")` o `family=quasibinomial(link="logit")` , sólo cambia las significaciones, pero no las devianzas ni los coeficientes de regresión de los predictores.

Podemos pegar en MS-Excel la salida de `dropterm(…)` y al texto importado aplicarle la función "=EXTRAE(…)*1" para convertir texto en números. Fijaos en la siguiente captura de pantalla (tened cuidado, que a lo peor mi salida textual de RStudio tiene un número diferente de espacios en blanco o decimales que la vuestra):



	A	B	C	D	E	F
1		Df	Deviance	AIC	LRT	Pr(Chi)
2	<none>		1767.7	2357.9		1767.7
3	altmed	1	1792.5	2380.7	24.748	6.533e-07 *** 1792.5
4	rangoalt	1	1891.5	2479.7	123.792	< 2.2e-16 *** 1891.5
5	shannon	1	1793.3	2381.6	25.606	4.186e-07 *** 1793.3
6	tempmin	1	2037.8	2626.0	270.074	< 2.2e-16 *** 2037.8
7	precip	1	1856.7	2444.9	88.930	< 2.2e-16 *** 1856.7

Las devianzas con las que vamos a trabajar son DEVIANZAS RESIDUALES. Para convertirlas en **DEVIANZAS RETENIDAS** hay que efectuar unos pequeños cálculos muy sencillos. Recordad que la **devianza nula** sale al final de `summary(…)`.

Para la devianza retenida por el MODELO: devianza nula – devianza del modelo = 2806.3 - 1767.7 = 1038.6

Para las retenidas por los EFECTOS DE LOS PREDICTORES (en la tabla de `dropterm(…)`) es respecto a la devianza del modelo:
por ejemplo, para "**altmed**" es altmed - <none> = 1792.5 - 1767.7 = 24.8 (esto mismo aparece en la columna LRT)

La CONCOMITANCIA es la diferencia entre la devianza retenida por el MODELO y la suma de las contribuciones de los EFECTOS DE LOS PREDICTORES.

Y para concluir, los **porcentajes de la devianza explicada** son el resultado de dividir las devianzas retenidas entre la devianza nula del modelo, y multiplicar ese valor por 100.

	Devianzas Residuales	Devianzas Retenidas	% devianza explicada
altmed	1792.5	24.8	**0.9**
rangoalt	1891.5	123.8	**4.4**
shannon	1793.3	25.6	**0.9**
tempmin	2037.8	270.1	**9.6**
precip	1856.7	89.0	**3.2**
MODELO	1767.7	1038.6	**37.0**
SUMA EFECTOS PRINCIPALES		533.3	**19.0**
CONCOMITANCIAS		505.3	**18.0**
NULO	2806.3		

O sea, que nuestro **modelo** explica un **37.0%** de la variabilidad existente en la frecuencia de aparición del avión roquero. De esa cantidad, el **19.0%** es atribuible a los **efectos parciales puros de las cinco variables predictoras**, mientras que un **18.0%** está asociado a la contribución de las relaciones complejas entre predictores debida a la correlación (i.e., no independencia) que existe entre ellas (**concomitancias**).

Las variables predictoras que de **modo parcial "exclusivo"** más contribuyen a explicar la variación en la variable respuesta son tempmin (**9.6%**), seguida de rangoalt (**4.4%**) y precip (**3.2%**). Para la altmed y shannon su contribución exclusiva no llega al 1%.

¡Y eso que todas las variables eran muy significativas!

Como hemos detectado puntos perdidos e influyentes en las exploraciones de los **residuos dato-a-dato**, sería conveniente efectuar estimas robustas del modelo, de manera que su parametrización y significación no sea sensible a esos datos “extremos”.

Para ello podemos contar con la aproximación de promediar múltiples modelos generados mediante el remuestreo-con-remplazo de la matriz original de datos (***bootstrapping***).

Mediante la siguiente línea de código efectuamos 1000 *bootstraps* (podemos hacer más cambiando `B=1000`).

```
boot.modelo <- as.data.frame(bootCase(modelo.bf, f.=coef, B=1000))
```

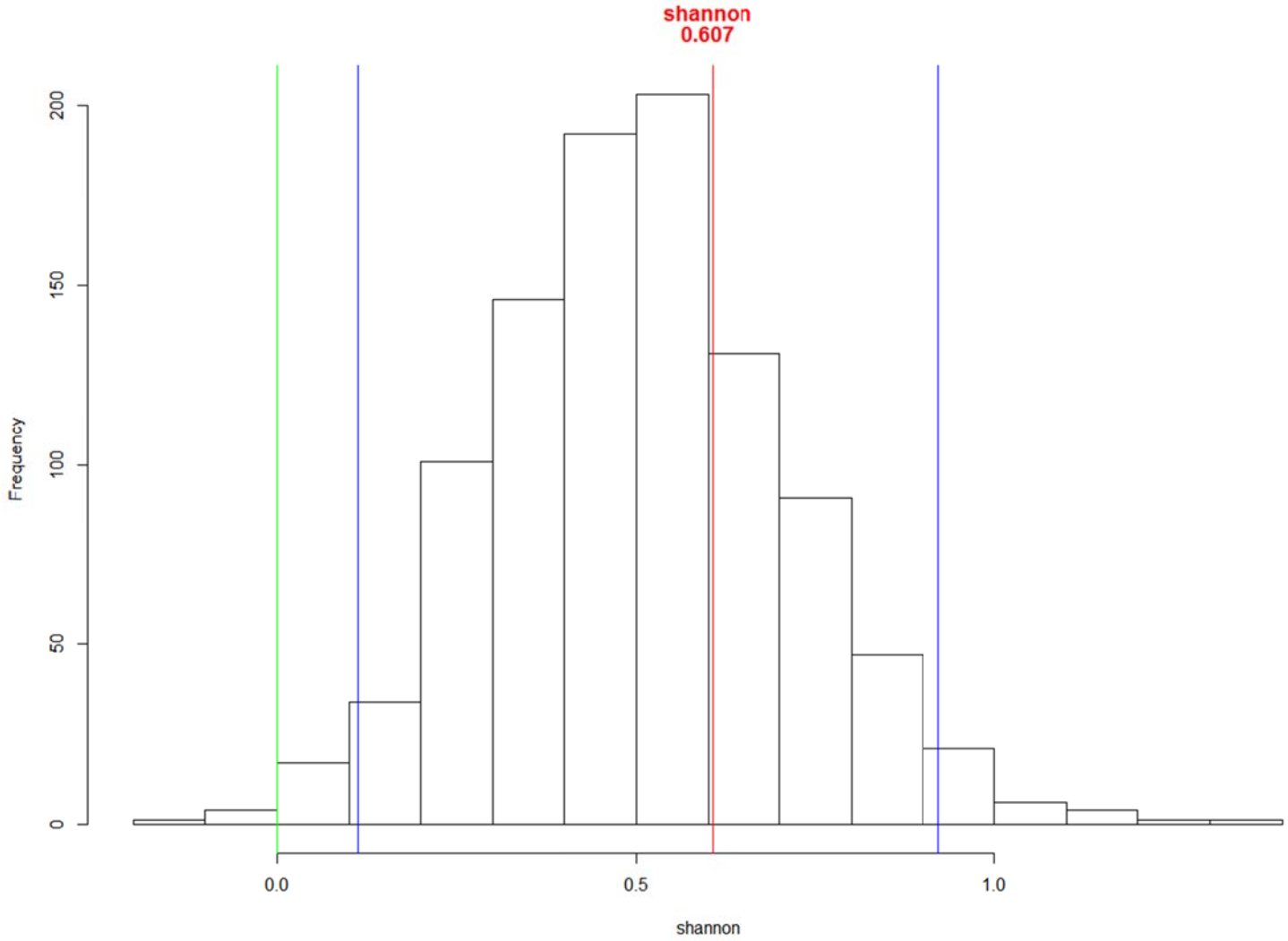
Sus resultados promediados aparecen aquí abajo. Recordad que el 'standard error' de los coeficientes '*bootstrapped*' es equivalente a la **sd** de los `B=1000` modelos generados.

```
print(describe(boot.modelo)[,c(1:4,8:9,11:12)], digits=5)
            vars    n     mean       sd       min      max     skew kurtosis
(Intercept)    1 1000 -6.44098 0.47074 -8.26133 -4.70442 -0.07980  0.29462
altmed         2 1000 -0.00098 0.00044 -0.00250  0.00019 -0.02981 -0.00952
rangoalt       3 1000  0.00158 0.00029  0.00080  0.00248  0.05077 -0.18430
shannon        4 1000  0.50701 0.20545 -0.10538  1.35208  0.20013  0.32001
tempmin        5 1000  0.38957 0.05022  0.22644  0.54926 -0.06889 -0.02030
precip         6 1000 -0.00218 0.00053 -0.00373 -0.00018  0.24812  0.13191


print("valores de p de los coeficientes 'bootstrapped'", quote=FALSE)
[1] valores de p de los coeficientes 'bootstrapped'
for (i in 1:nv) {
   print(c(colnames(boot.modelo[i]), "", 2*pt(-abs(tb[i]), df=999)), digits=5, quote=FALSE)
 }
[1] (Intercept)                          3.45696638820634e-39
[1] altmed                               0.0278819997195345
[1] rangoalt                             4.38991276669911e-08
[1] shannon                              0.0137637686849482
[1] tempmin                              2.14243475444452e-14
[1] precip                               4.20812550788689e-05
```

También podemos verlo (textualmente con cuantiles y de modo gráfico) con las siguientes líneas de código que trabajan con el objeto generado por el comando `bootCase(modelo.bf, f.=coef, B=1000)`.

```
## QUE LOS INTERVALOS SIGUIENTES NO INCLUYAN EL VALOR "CERO"
nv <- length(boot.modelo); mb <- vector(length=nv)
sdb <- vector(length=nv); tb <- vector(length=nv)
for (i in 1:nv) {
   print(c(colnames(boot.modelo[i]),"=",round(mean(boot.modelo[,i]),4)), quote=FALSE)
   mb[i] <- mean(boot.modelo[,i])
   sdb[i] <- sd(boot.modelo[,i])
   tb[i] <- mean(boot.modelo[,i])/sd(boot.modelo[,i])
   vrc=modelo$coefficients[i]/1
   hist(boot.modelo[,i], breaks=20, xlab=names(boot.modelo[i]), main=c(names(boot.modelo[i]),round(vrc,3)), col.main="red")
   abline(v=modelo$coefficients[i], col="red")                ## en ROJO parámetros observados en el modelo que se examina
   abline(v=quantile(boot.modelo[,i], 0.025), col="blue")     ## en AZUL los límites a alfa = 0.05 de los valores remuestreados
   abline(v=quantile(boot.modelo[,i], 0.975), col="blue")
   abline(v=0, col="green")                                   ## en VERDE el valor cero indicativo de la Ho (hipótesis nula)
   print(quantile(boot.modelo[,i], c(0.005, 0.025, 0.05, 0.95, 0.975, 0.995)), quote=FALSE)
   print("-----------------------------------------------------", quote=FALSE)
 }
```

```
[1] (Intercept) =           -6.441
     0.5%      2.5%        5%       95%     97.5%     99.5%
-7.741299 -7.371553 -7.229998 -5.703278 -5.582155 -5.190778
[1] -----------------------------------------------------
[1] altmed =      -0.001
         0.5%          2.5%            5%           95%         97.5%         99.5%
-0.0022165326 -0.0018399622 -0.0016853782 -0.0002632102 -0.0001197907  0.0001520839
[1] -----------------------------------------------------
[1] rangoalt =        0.0016
        0.5%         2.5%           5%           95%        97.5%        99.5%
0.0008894063 0.0010409220 0.0011000274 0.0020467649 0.0021303850 0.0023137447
[1] -----------------------------------------------------
[1] shannon =       0.507
       0.5%        2.5%          5%          95%       97.5%       99.5%
0.003269489 0.111374500 0.192221382 0.847573318 0.922617031 1.116427677
[1] -----------------------------------------------------
[1] tempmin =       0.3896
     0.5%      2.5%        5%       95%     97.5%     99.5%
0.2541182 0.2906202 0.3061930 0.4748624 0.4838110 0.5106176
[1] -----------------------------------------------------
[1] precip  =       -0.0022
       0.5%         2.5%           5%           95%        97.5%        99.5%
-0.003428890 -0.003183341 -0.002999487 -0.001276408 -0.001041311 -0.000802869
[1] -----------------------------------------------------
```


shannon
0.607
Frequency
0
50
100
150
200
0.0
0.5
1.0
shannon

Otra aproximación a la obtención de **modelos robustos** es la que tiene en cuenta las propiedades de las observaciones (i.e., datos) considerando su *leverage* y/o sus valores residuales (e.g., dffits). En estas aproximaciones, las observaciones con mayores valores de leverage o dffits "pesan" menos en la construcción del modelo que aquellas con valores menos "extremos" por ser datos influyentes o perdidos (*outliers*). Por ejemplo, podemos utilizar el comando `glmrob` del paquete `robustbase`.

```
library(robustbase)
```

En esta ocasión, con una distribución binomial de frecuencias, es más recomendable hacer uso del modo: cbind(conteo.si, conteo.no) que no hace uso de weights=... (definiendo el denominador de la frecuencia).

```
eqt.bf2 <- as.formula(cbind(ptyrup, denominador-ptyrup) ~ altmed+rangoalt+shannon+tempmin+precip)
modelo.bf2 <- glm(eqt.bf2, data=datos, family=binomial(link="logit"))
```

`weights.on.x = "hat"` "downweights" por el leverage de cada valo. `Mqle` es el método Mallows-Hubber quasi-liquelihood.

```
modelo.r <- glmrob(eqt.bf2, data=datos, family=binomial(link="logit"), weights.on.x="hat", method="Mqle")
```

En estas líneas podemos ver cómo han cambiado los valores de los coeficientes de regresión de nuestro modelo original (`modelo.bf2`) y del nuevo "robusto":

```
round(modelo.bf2$coefficients, 5)
(Intercept)      altmed    rangoalt     shannon     tempmin      precip
   -6.42908    -0.00096     0.00156     0.49453     0.39092    -0.00214
> round(modelo.r$coefficients, 5)
(Intercept)      altmed    rangoalt     shannon     tempmin      precip
   -7.08801    -0.00148     0.00212     0.59095     0.38438    -0.00248
```

Como vemos son parecidos, aunque hay importantes cambios en las variables predictoras `shannon`, `altmed` y `rangoalt`. Visualizar esto último en relación con la salida de las dfbetas que hemos visto antes en (no representado aquí):

```
dfbetasPlots(modelo.bf)    ## o dfbetasPlots(modelo.bf2) que acabamos de crear con cbind(…)
```

El resumen de los efectos parciales de las variables en la estima robusta viene dada por:

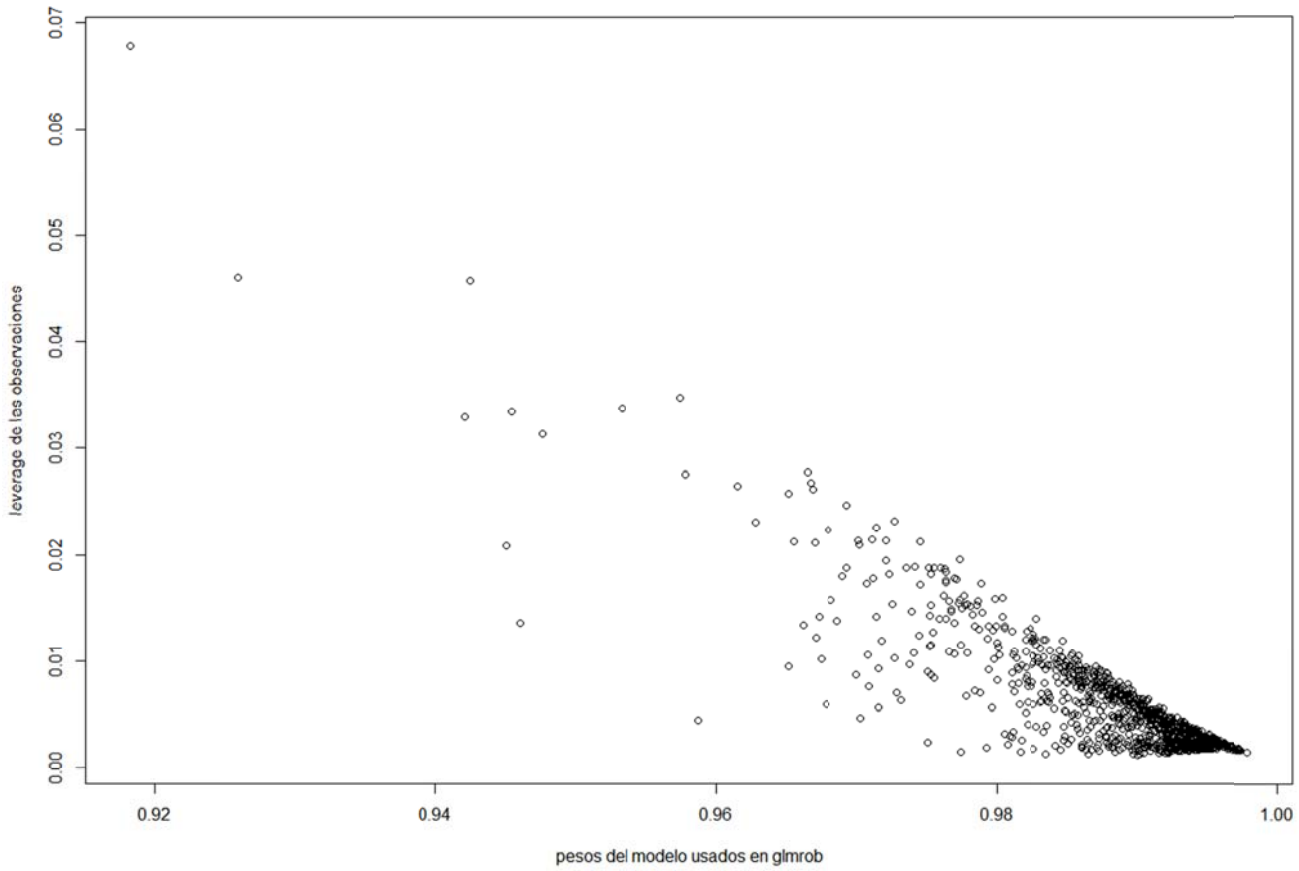
```
summary(modelo.r)
```

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -7.0880056  0.3204095 -22.122  < 2e-16 ***
altmed      -0.0014844  0.0002619  -5.667 1.46e-08 ***
rangoalt     0.0021216  0.0001751  12.119  < 2e-16 ***
shannon      0.5909515  0.1412073   4.185 2.85e-05 ***
tempmin      0.3843779  0.0314404  12.226  < 2e-16 ***
precip      -0.0024774  0.0003062  -8.091 5.90e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Robustness weights w.r * w.x:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.03868 0.98150 0.98900 0.90550 0.99290 0.99790

Number of observations: 999
Fitted by method ‘Mqle’  (in 11 iterations)

(Dispersion parameter for binomial family taken to be 1)

No deviance values available
Algorithmic parameters:
   acc    tcc
0.0001 1.3450
maxit
   50
test.acc
  "coef"
```

Los modelos generados con el comando `glmrob` del paquete `robustbase` no nos permiten hacer muchas cosas más, con lo cual nos quedamos con los anteriores resultados sin ahondar más en otros aspectos (i.e., sobredispersión, devianza explicada, etc).

Para terminar, podemos explorar con qué se han relacionado los pesos que ha utilizado `glmrob` para generar el modelo mediante el uso del argumento `weights.on.x="hat"`.

```
plot(modelo.r$w.x, leverage(modelo), ylab="leverage de las observaciones", xlab="pesos del modelo usados en glmrob")
```



También podemos hacer un análisis del éxito predictivo de nuestro modelo prediciendo cuál será la variación de la variable respuesta en otro juego de datos no utilizado para construir nuestro modelo. Para ello, antes creamos una variable llamada `denominador` en nuestro juego de datos `predecir` que define que siempre hay 60 transectos como denominador.

```
predecir$denominador <- seq(60,60, length.out=689)

predecir$ptyrup.nuevo.bf <- predict(modelo.bf, newdata=predecir, type="response")
```

Como también contamos en el juego de datos `predecir` con la variable respuesta `ptyrup` medida realmente en el campo, podemos efectuar un test del poder predictivo del modelo `modelo.bf`. Para ello, relacionamos los datos de la variable respuesta medidos, pero no utilizados en `modelo.bf` (`predecir$ptyrup`), con los predichos por él (`predecir$ptyrup.nuevo.bf`).

```
summary(lm(I(predecir$ptyrup/predecir$denominador) ~ predecir$ptyrup.nuevo.bf))   ## I(…) indica que calculamos algo

Call:
lm(formula = I(predecir$ptyrup/predecir$denominador) ~ predecir$ptyrup.nuevo.bf)
Residuals:
     Min       1Q   Median       3Q      Max
-0.14171 -0.00908 -0.00239  0.00036  0.35041
Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              -0.001899   0.001632  -1.164    0.245
predecir$ptyrup.nuevo.bf  1.267703   0.064458  19.667   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03634 on 687 degrees of freedom
Multiple R-squared:  0.3602,   Adjusted R-squared:  0.3593
F-statistic: 386.8 on 1 and 687 DF,  p-value: < 2.2e-16
```

De nuevo, vemos que el poder predictivo del modelo es aceptable (R^2 = 36.0%), significativo (p<<0.001), pero en esta ocasión el modelo GLM binomial de frecuencia (`modelo.bf`) tiende a sobreestimar la abundancia relativa del avión roquero (nuestra variable respuesta, `ptyrup`), porque la pendiente de `predecir$ptyrup.nuevo.bf` es **1.268**, mayor que el valor esperable de **1** si OBSERVADO fuese igual a PREDICHO (**OBSERVADO = 0 + 1 * PREDICHO**):

```
(1-1.267706) / 0.064458      ## esto es una t de Student de desvío de un coeficiente de regresión del valor 1
[1] -4.153185

2*(1-pt(4.153185, df=687))   ## cálculo de la p para una t de Student, con dos colas
[1] 3.69182e-05
```

Y sólo constatar que la ordenada en el origen de la ecuación no difiere de cero (mirad en los resultados previos).

```
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                -0.001899   0.001632  -1.164    0.245
```

SIMPLIFICACIÓN DE MODELOS: Los modelos construidos pueden ser reducidos mediante aproximaciones basadas en el valor AIC de Akaike usando el comando `stepAIC` del paquete `MASS`. Si en nuestro modelo saturado existen variables que le aportan complejidad pero no mejoran sustancialmente la representación del contenido informativo de la matriz original de datos, entonces esas variables se identifican como candidatas a ser eliminadas.

```
library(MASS)          ## para el comando stepAIC
```

Para ello establecemos la lista de nuestras variables predictoras incluidas en la ecuación `eqt.bf` con la que hemos generado el `modelo.bf`. Por ejemplo, llamamos a esa lista `formula`.

```
formula <- ~ altmed+rangoalt+shannon+tempmin+precip              ## sí, hay que incluir ~ ; no lo olvidéis
```

A continuación corremos la siguiente línea de código (poniendo `trace=FALSE` no se muestran los resultados parciales de los pasos dados para reducir el modelo original saturado)

```
modelo.reducido <- stepAIC(modelo.bf, scope = list(upper = formula, lower = ~1), direction="both", k=2, trace=TRUE)
```

En esta occasion, `stepAIC(modelo.bf, …)` identifica que no es conveniente eliminar ninguna variable del modelo original. El modelo reducido resultaría en una versión peor en cuanto a la reperesentación del contenido informativo de la matriz de datos.

```
> modelo.reducido <- stepAIC(modelo.bf, scope =list(upper = formula, lower = ~1), direction="both", k=2, trace=TRUE)
Start:  AIC=2357.95
ptyrup/denominador ~ altmed + rangoalt + shannon + tempmin +  precip

           Df Deviance    AIC
<none>          1767.7 2357.9            ## valor de AIC si no quitamos ninguna variable predictora
- altmed    1   1792.5 2380.7            ## AIC del modelo SIN la variable indicada (altmed)
- shannon   1   1793.3 2381.6            ## claramente, cualquier configuración quitando una variable es peor
- precip    1   1856.7 2444.9            ##    (mayor valor de AIC) que el modelo saturado que no quita ninguna
- rangoalt  1   1891.5 2479.7            ## en otras ocasiones salen numerosas tablas de resultados parciales como
- tempmin   1   2037.8 2626.0            ##    ésta quitando una variable en cada paso (cuando AIC del modelo
                                         ##    sin esa variable es menor que el AIC de <none>)
```

Y con la línea siguiente obtenemos los resultados de nuestro modelo simplificado-reducido (que en esta ocasión es el mismo que el original saturado).

```
> summary(modelo.reducido)
Call:
glm(formula = ptyrup/denominador ~ altmed + rangoalt + shannon +
    tempmin + precip, family = binomial(link = "logit"), data = datos,
    weights = denominador)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-4.7313  -0.9651  -0.5806  -0.3508   7.2837

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -6.4290785  0.2328478 -27.611  < 2e-16 ***
altmed      -0.0009578  0.0001953  -4.905 9.35e-07 ***
rangoalt     0.0015643  0.0001358  11.518  < 2e-16 ***
shannon      0.4945327  0.0994725   4.972 6.64e-07 ***
tempmin      0.3909216  0.0238705  16.377  < 2e-16 ***
precip      -0.0021438  0.0002331  -9.196  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
 (Dispersion parameter for binomial family taken to be 1)
    Null deviance: 2806.3  on 998  degrees of freedom
Residual deviance: 1767.7  on 993  degrees of freedom
AIC: 2358
Number of Fisher Scoring iterations: 6
```

Y ya para terminar, podemos efectuar una estima "promedio" del modelo basado en la Teoría de la Información que tiene en cuenta la aproximación de AIC de Akaike. Esta es una versión "no frecuentista" (no establecida sobre las significaciones, p's). Está basada en criterios de evidencia de la proximidad de un modelo (generalizado lineal en este caso) a la información contenida en nuestros datos. Esta aproximación "*multimodel inference*" basada en el valor AIC de Akaike la vamos a llevar a cabo con el paquete `glmulti`.

```
library(glmulti)       ## para el comando glmulti
```

En aras de facilitar el trabajo nos restringimos al uso del modo `cbind(conteo.si, conteo.no)` que no hace uso de `weights=...` definiendo el denominador de la frecuencia. Esto es, vamos a utilizar las versiones `eqt.bf2` y `modelo.bf2` creados previamente. El argumento `method="h"` se refiere a la búsqueda exhaustiva de todos los modelos posibles teniendo en cuenta todas las combinaciones entre variables predictoras. Y `confsetsize=100` indica que aunque estime todos los modelos posibles retiene "sólo" los 100 primeros con menores valores de AICc; en nuestro caso no se alcanza este límite porque sólo hay 5 variables predictoras.

```
Multimodelo.bf2 <- glmulti(modelo.bf2, family=binomial(link="logit"), level=1, confsetsize=100, method="h", crit="aicc")

Initialization...
TASK: Exhaustive screening of candidate set.
Fitting...

After 50 models:
Best model: cbind(ptyrup,denominador-ptyrup)~1+altmed+rangoalt+shannon+tempmin+precip
Crit= 2358.03753120908
Mean crit= 2781.33133272704
Completed.
```

Con la siguiente instrucción vemos todos los modelos generados, las variables predictoras que incluyen, los valores de AICc de cada modelo y sus pesos. Resulta claro que el mejor modelo (el 1, con AICc = 2358.038) es el mejor de todos los posibles, ya que el siguiente tiene un valor de AICc mayor en 22.72 unidades. Por ello, el peso del primer modelo es 0.99998.

```
weightable(multimodelo.bf2)
```

	model	aicc	weights
1	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt + shannon + tempmin + precip	2358.038	**9.999808e-01**
2	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt + shannon + tempmin + precip	2380.761	1.162937e-05
3	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt + tempmin + precip	2381.620	7.571474e-06
4	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt + tempmin + precip	2404.939	6.537160e-11
5	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt + shannon + tempmin	2444.943	1.345017e-19
6	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt + shannon + tempmin	2452.785	2.666446e-21
7	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt + tempmin	2459.538	9.110493e-23
8	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt + tempmin	2469.198	7.275836e-25
9	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + shannon + tempmin + precip	2479.805	3.617732e-27
10	cbind(ptyrup, denominador - ptyrup) ~ 1 + shannon + tempmin + precip	2493.149	4.581159e-30
11	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + shannon + tempmin	2516.015	4.962650e-35
12	cbind(ptyrup, denominador - ptyrup) ~ 1 + shannon + tempmin	2527.548	1.553608e-37
13	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + tempmin + precip	2537.833	9.075692e-40
14	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + tempmin	2560.688	9.886081e-45
15	cbind(ptyrup, denominador - ptyrup) ~ 1 + tempmin + precip	2567.210	3.790861e-46
16	cbind(ptyrup, denominador - ptyrup) ~ 1 + tempmin	2585.051	5.064578e-50
17	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt + shannon + precip	2626.088	6.217859e-59
18	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt + precip	2644.833	5.286694e-63
19	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt + shannon	2708.431	8.186073e-77
20	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + rangoalt	2717.847	7.386814e-79
21	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + shannon + precip	2963.131	4.033161e-132
22	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + shannon	2966.111	9.090845e-133
23	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed	3055.367	3.774430e-152
24	cbind(ptyrup, denominador - ptyrup) ~ 1 + altmed + precip	3057.307	1.430576e-152
25	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt + shannon + precip	3297.053	1.245773e-204
26	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt + shannon	3298.072	7.483266e-205
27	cbind(ptyrup, denominador - ptyrup) ~ 1 + shannon + precip	3321.024	7.765569e-210
28	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt + precip	3321.377	6.507845e-210
29	cbind(ptyrup, denominador - ptyrup) ~ 1 + rangoalt	3324.283	1.522049e-210
30	cbind(ptyrup, denominador - ptyrup) ~ 1 + shannon	3330.299	7.518283e-212
31	cbind(ptyrup, denominador - ptyrup) ~ 1 + precip	3365.744	1.511382e-219
32	cbind(ptyrup, denominador - ptyrup) ~ 1	3386.518	4.658549e-224

Consistentemente, la aproximación previa de reducción-simplificación del modelo mediante `stepAIC(modelo.bf, …)` nos indicaba que no era aconsejable eliminar ninguna variable del modelo original “saturado”.

Y a continuación, obtenemos el resultado multimodelo y los errores estándar incondicionales de cada variable predictora con la ponderación de los coeficientes de regresión teniendo en cuenta los pesos de cada modelo..

```
tabla.uncond <- as.data.frame(coef.glmulti(multimodelo.bf2, select="all", varweighting="Buckland", icmethod="Burnham", alphaIC=0.05))
names(tabla.uncond)[2] <- "var.uncond"
names(tabla.uncond)[5] <- "-/+ IC95%"
tabla.uncond$stderror <- sqrt(tabla.uncond$var.uncond)
tabla.uncond
```

```
                Estimate   var.uncond Nb models Importance    -/+ IC95%     stderror
altmed      -0.0009577555 3.813390e-08        16  0.9999884 0.0003832069 0.0001952790
shannon      0.4945289496 9.895372e-03        16  0.9999924 0.1952062889 0.0994754843
(Intercept) -6.4290803494 5.422245e-02        32  1.0000000 0.4569485277 0.2328571344
rangoalt     0.0015643146 1.844614e-08        16  1.0000000 0.0002665204 0.0001358166
tempmin      0.3909225869 5.698380e-04        16  1.0000000 0.0468439415 0.0238712790
precip      -0.0021437785 5.434364e-08        16  1.0000000 0.0004574589 0.0002331172
```

Con los valores de `Importance` podemos percatarnos de que todas las variables predictoras son imprescindibles en el modelo para representar la máxima cantidad de contenido informativo en la variable respuesta Binomial, frecuencia de ocurrencia del avión roquero.

También podemos ver que las estimas de los coeficientes de regresión de las variables predictoras (`Estimate`, primera columna numérica) distan de la hipótesis de efecto nulo (Ho: coeficientes = 0) muchas unidades de errores estándar (`stderror`, sexta columna numérica). ¡Bastantes más de 4! Recordemos que para más 150 unidades muestrales, `Estimate/stderror` es aproximadamente 2 a un alfa=0.05.

```
abs(tabla.uncond[1]/tabla.uncond[6])
             Estimate
altmed       4.904549
shannon      4.971365
(Intercept) 27.609549
rangoalt    11.517849
tempmin     16.376273
precip       9.196140
```

Los modelos repasados hasta ahora se describen como Modelos Generalizados **LINEALES**. Asumen diferentes distribuciones de la variable respuesta y sus errores, pero todos ellos tienen en común que presuponen la

EXISTENCIA DE RELACIONES LINEALES ENTRE LA VARIABLE RESPUESTA Y LAS PREDICTORAS

De no ser así, los efectos de las variables explicativas predictoras estarán estimados de manera sesgada. El no cumplimiento de este requisito puede manifestarse en la exploración de los residuos de los modelos a través de la violación de la normalidad, o la manifestación de patrones curvos en la relación entre los residuos del modelo y sus predicciones (aplicando la *link function*).

Si esto es así, deberíamos proceder a la estima de las relaciones curvas de las variables predictoras con la respuesta. Para ello contamos con los Modelos Generalizados **ADITIVOS** (GAM).

Para entender cómo se manifiesta este problema vamos a "inventarnos" unos datos para simular su efecto, y establezcamos con ellos un modelo sencillo del tipo **Y es-función-de X**.

Generemos una variable predictora **X** (`pred.x`) con 250 datos, media = 15 y desviación estándar = 10:

```
pred.x <- rnorm(n=250, mean=15, sd=10)
```

Ahora una variable respuesta **Y** (`resp.y`) que va a ser función de la variable predictora según una relación polinomial pre-establecida por la siguiente función: Y = 25 + 3*X – 0.07*X^2. Y además, a la respuesta **Y** le vamos a sumar "**ruido**" (`ruido`) según un generador de números al azar de media = 0 y desviación típica = 8.

```
ruido <- rnorm(n=250, mean=0, sd=8)
resp.y <- 75 + 3*pred.x + -0.07*(pred.x^2) + ruido
```

Y para terminar construyamos un modelo de regresión generalizado lineal.

```
modelo.azar <- glm (resp.y ~ pred.x, family=gaussian(link="identity"))
```

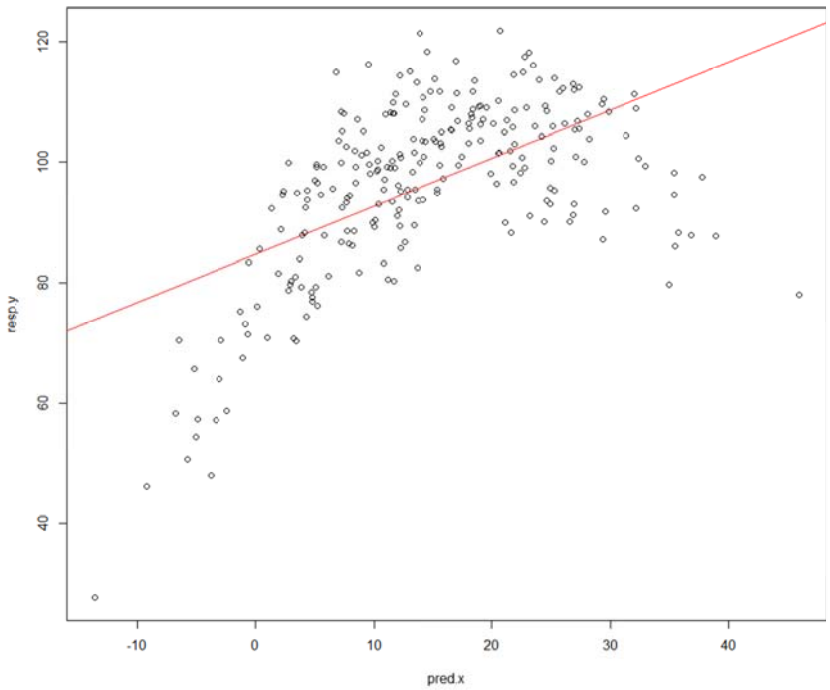
Veamos lo que resulta. ¡OJO! A cada uno le saldrá una cosa parecida … pero diferente. ¡Estamos simulando!

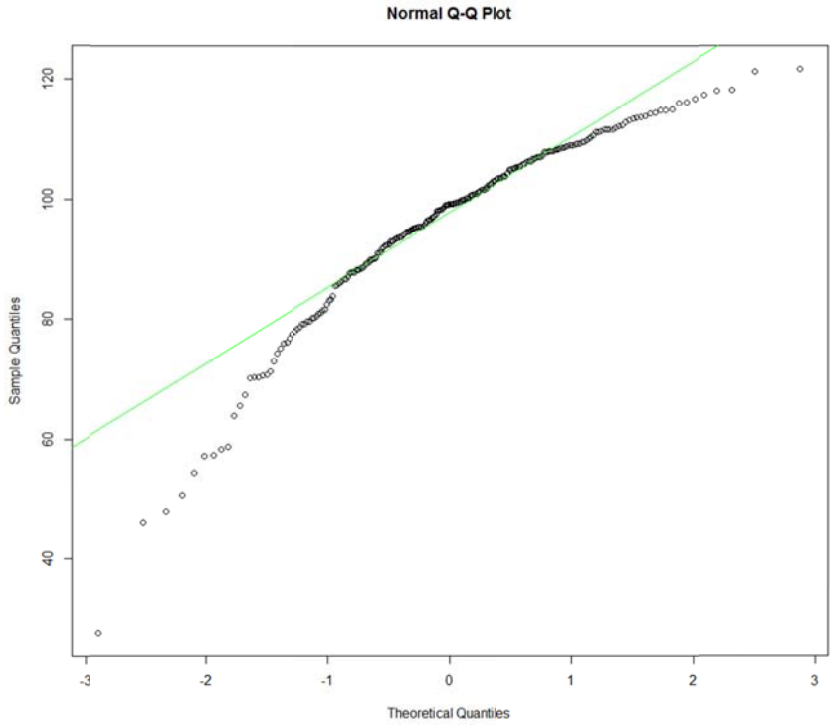
DATOS ORIGINALES

```
plot(pred.x, resp.y); abline(lm(resp.y ~ pred.x), col="red")      ## relación X con Y
qqnorm(resp.y); qqline(resp.y, col="green")                       ## qqplot de la respuesta original
```

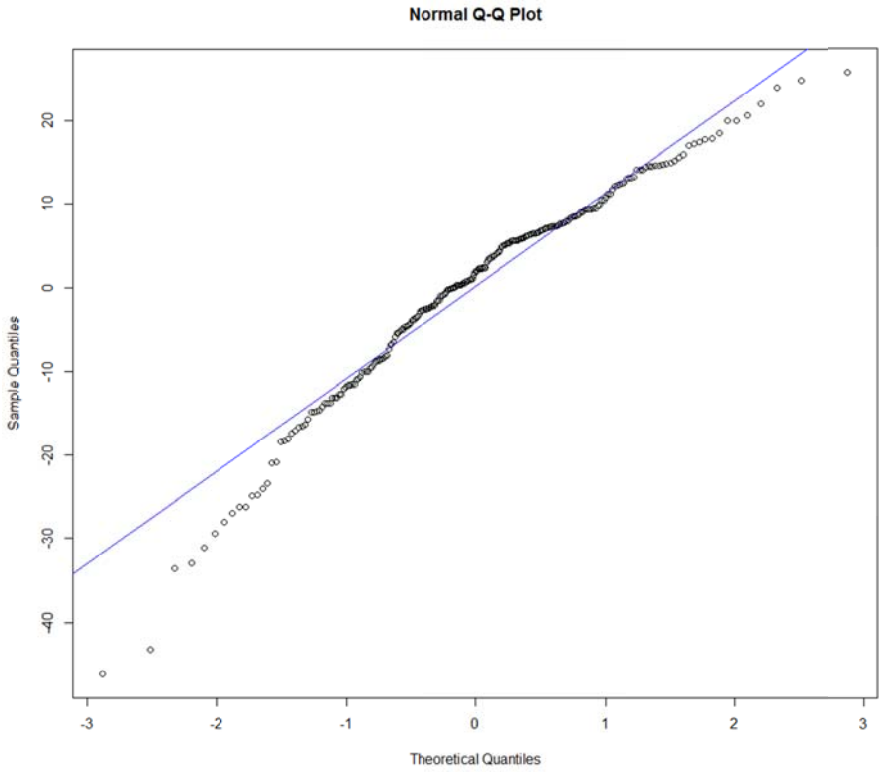
RESIDUOS DEL MODELO modelo.azar

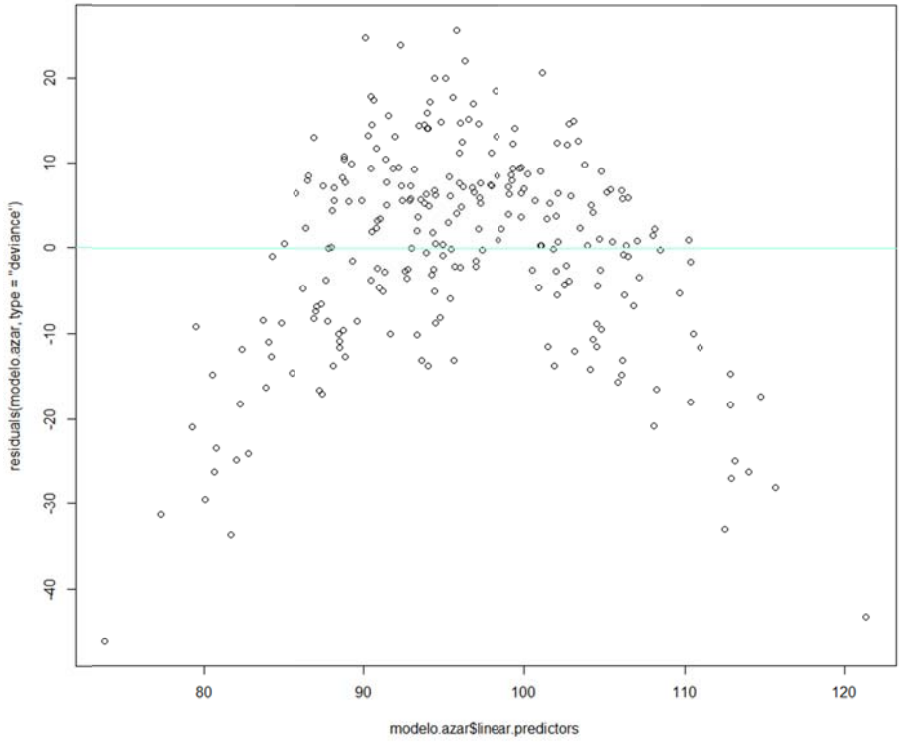
```
qqnorm(residuals(modelo.azar, type="deviance"))                   ## esto ya lo hemos visto muchas veces
qqline(residuals(modelo.azar, type="deviance"), col="blue")

plot(modelo.azar$linear.predictors, residuals(modelo.azar, type="deviance"))
abline(h=0, col="aquamarine")                                     ## esto también lo hemos visto muchas veces
```





Los residuos del modelo se representan aquí abajo.

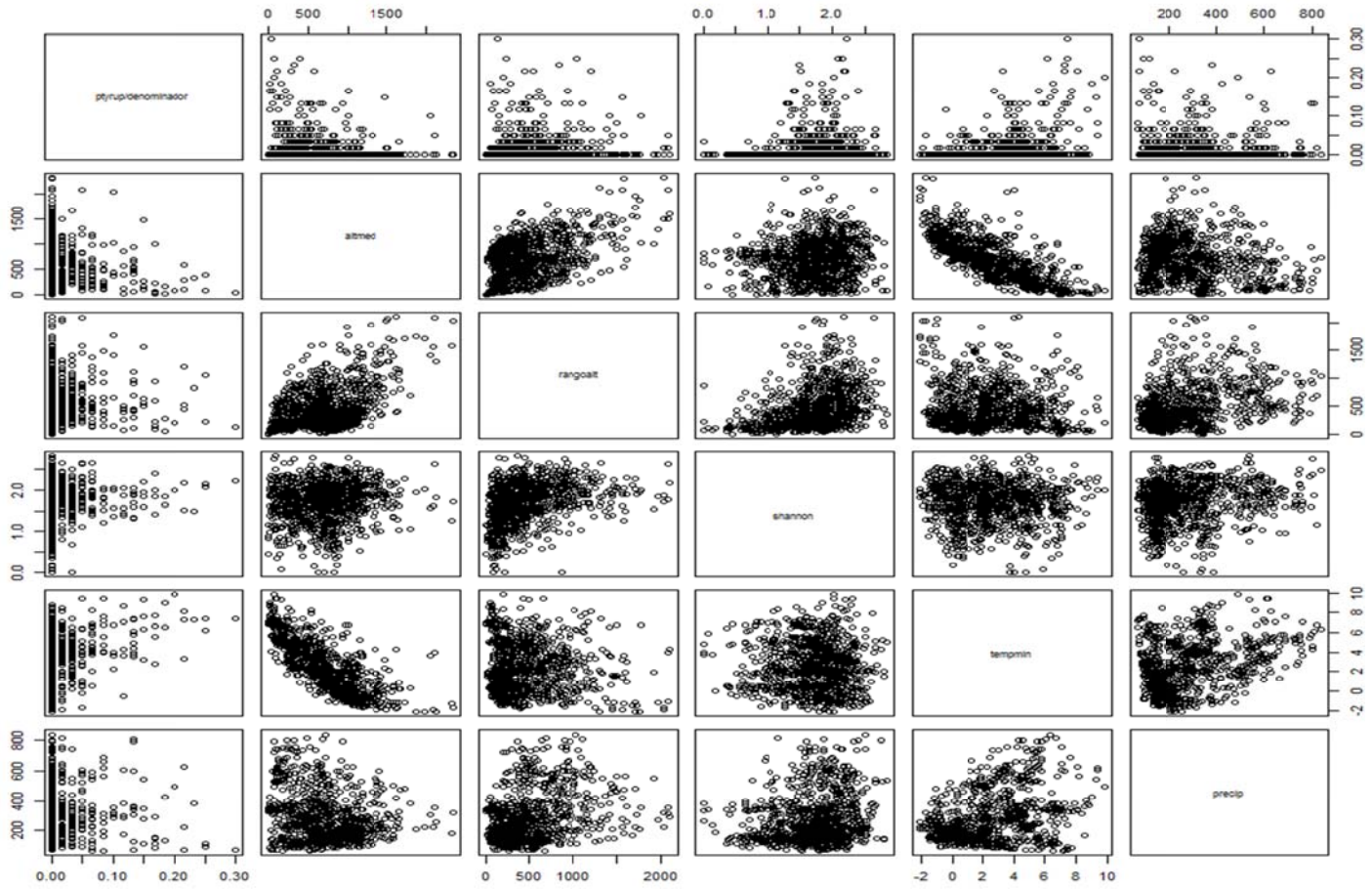




Esto que véis aquí (o patrones similares de desvío de la normalidad y de los “cielos estrellados de una noche de verano”) es lo que siempre deberíais evitar en los análisis con Modelos Generalizados Lineales. ¡Sobre todo lo ilustrado en el gráfico de la derecha!

Y ahora veamos con la siguiente figura (primera fila) las relaciones dibujadas entre la respuesta y las predictoras.

```
pairs(eqt.bf, data=datos)
```



Detectamos claramente relaciones no lineales, más bien relaciones cuadráticas de máximo.

Para abordar este problema vamos a trabajar con **Modelos Generalizados ADITIVOS (GAM)** operando con el paquete `mgcv`.

```
library(mgcv)      ## este es el paquete para efectuar modelos GAM
```

Con los modelos GAM podemos asumir **diferentes tipos de distribuciones de la variable respuesta** al igual que con los modelos GLM previamente presentados (family=): poisson(link="log"); binomiales negativas: nb(); gaussian(link="identity"); binomial(link="logit"); quasipoisson(link="log"); quasibinomial(link="logit").

Los **términos no lineales** se pueden definir de diferentes modos; consultad el capítulo 5 de:
http://reseau-mexico.fr/sites/reseau-mexico.fr/files/igam.pdf (paginas 224, 226)

k define la complejidad máxima alcanzable; k por defecto es 10 (si no se escribe)

por defecto, al utilizar **s(...)** la base del spline (**bs**) es *thin plate* (**tp**)
http://en.wikipedia.org/wiki/Thin_plate_spline para *thin plate splines*.
ejemplos: s(altmed, bs="tp", k=10) s(altmed, bs="tp", k=5)

bs="**cr**": *cubic regression splines*
ejemplos: s(altmed, bs="cr", k=10) s(altmed, bs="cr", k=5)

ti(...) para "*interaction splines*" cuando las dos predictoras se miden en la misma escala; suelen requerir un valor de k más alto
ejemplos: ti(variable_1, variable_2, bs="tp", k=15) podemos modificar bs="tp" por bs="cr"

te(…) para "*tension product spline*" cuando las dos predictoras no se miden en la misma escala ¡ el más versátil !
ejemplos: te(variable_1, variable_2, bs="tp", k=15)

Y a continuación vamos a seguir los mismos pasos dados en la definición de modelos GLM:

Ecuación del modelo **eqt.gam** estableciendo la variable respuesta como una Binomial que se mide en frecuencia (recordad que previamente hemos creado una variable `denominador` que establecía en 60 el número de transectos de 15 minutos realizados en cada unidad muestral UTM de 10x10 km^2).

```
eqt.gam <- as.formula(ptyrup/denominador ~ s(altmed, bs="tp", k=10) +s (rangoalt, bs="tp", k=10) + s(shannon, bs="tp
", k=10) + s(tempmin, bs="tp", k=10) + s(precip, bs="tp", k=10))
```

Construcción del modelo GAM con el comando `gam` del paquete `mgcv`. Como asumimos una distribución de la variable respuesta Binomial medida en frecuencia, y la hemos establecido como un cociente (`ptyrup/denominador`) tenemos que introducir dentro del comando `gam(…)` el argumento `weight=denominador`.

```
modelo.gam <- gam(eqt.gam, data=datos, binomial(link="logit"), weight=denominador, method="GCV.Cp", gamma=1.4, scale=0)
```

En esta línea de código hemos definido que los cálculos numéricos se efectúen aplicando el método `method="GCV.Cp"`, "*general cross validation for unknown scale parameter*". Otra opción es mediante el método de *Restricted Estimation Maximum Likelihood*, lo cual requeriría sustituir `method="GCV.Cp"` por `method="REML", optimizer=c("outer","newton")`. Este es el método implementado por defecto para las distribuciones Binomiales Negativas y en los modelos GAM mixtos.

Con el argumento `gamma`: *"The argument gamma=1.4, forces each model effective degree of freedom to count as 1.4 degrees of freedom in the GCV score, which forces models to be a little smoother than they might otherwise be, and is an ad hoc way of avoiding overfitting*". Esto es, con el parámetro gamma penalizamos aun más la parametrización curva del modelo, suavizándolo un poco más. Se suele recomendar operar con este parámetro gamma de "modo global" en todo el modelo, en vez construir otros probando con menores **k'**s.

Por ultimo, añadiendo `scale=-1` se corrige por sobredispersión en variables respuestas BINOMIALES o POISSON. ¡¡¡ hacedlo sólo con estas distribuciones !!!

Y ahora hagamos un repaso de los **supuestos canónicos del modelo** GAM creado.

```
shapiro.test(residuals(modelo.gam,type="deviance"))
        Shapiro-Wilk normality test
data:  residuals(modelo.gam, type = "deviance")
W = 0.8152, p-value < 2.2e-16
```

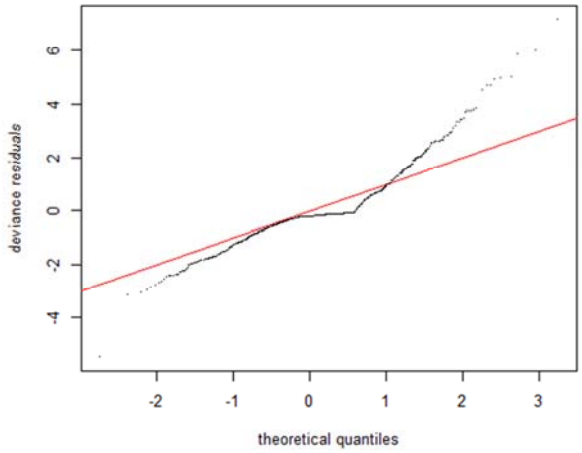
Los residuos del modelo se desvían del supuesto canónico de la normalidad.

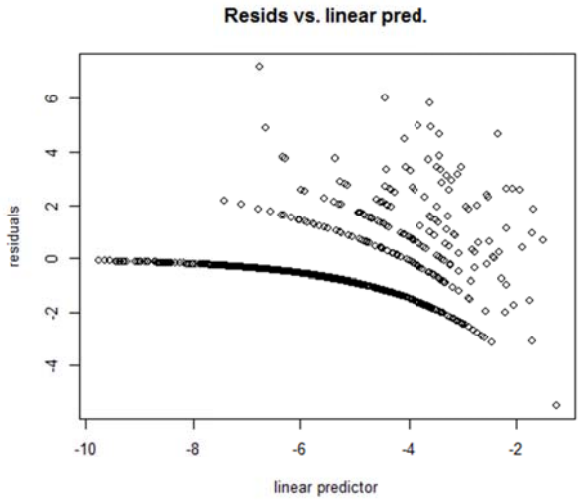
```
par(mfcol=c(1,1))
gam.check(modelo.gam, k.rep=1000)       ## k.rep=1000 denota 1000 procesos de remuestreo para las estimas

Method: UBRE   Optimizer: outer newton
full convergence after 9 iterations.
Gradient range [-2.082345e-08,2.160076e-06]
(score 0.5875627 & scale 1).
Hessian positive definite, eigenvalue range [0.001225933,0.002146431].
Model rank =  46 / 46

Basis dimension (k) checking results. Low p-value (k-index<1) may
indicate that k is too low, especially if edf is close to k'.

               k'   edf k-index p-value
s(altmed)   9.000 6.577   0.949    0.80
s(rangoalt) 9.000 6.544   0.930    0.54
s(shannon)  9.000 7.518   0.915    0.34
s(tempmin)  9.000 8.006   0.950    0.77
s(precip)   9.000 6.789   0.942    0.71
```

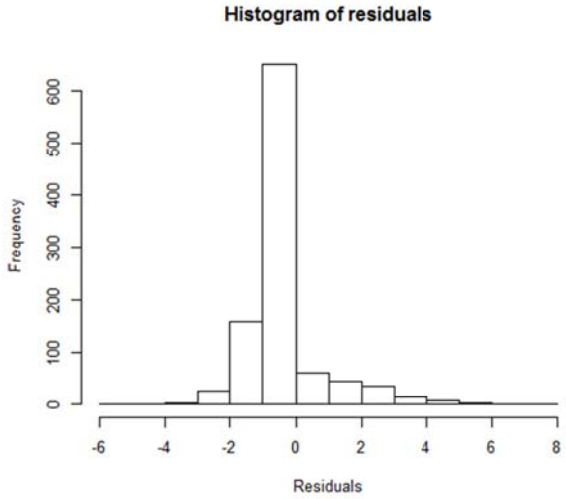
Test de complejidad de los splines: Los resultados anteriores muestran si es necesario elevar el parámetro de complejidad **k** de los splines. **k'** es el máximo **k** obtenible menos uno (k'=k-1); **edf** son los grados de libertad efectivos; en los términos **te** o **ti** es k'=k*k-1.

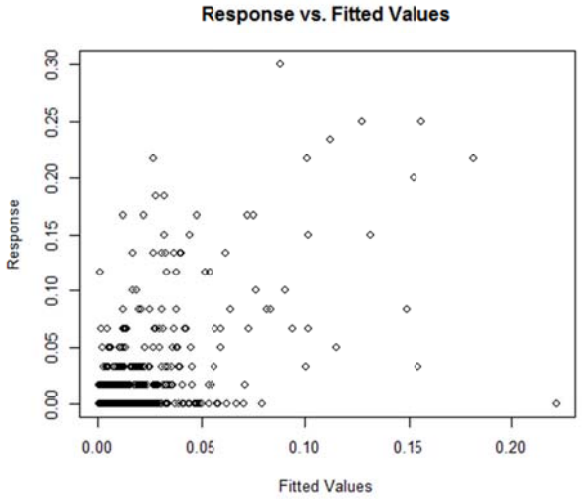
Lo ideal es que edf sea menor que k'; si no es así ... volvemos a construir la ecuación **eqt.gam** elevando k (sobre todo si p<0.01)

En esta ocasión, la complejidad máxima asumida de k'=k-1=9 no ha sido alcanzada, habiendo sido penalizada a valores menores.


deviance residuals
theoretical quantiles


Resids vs. linear pred.
residuals
linear predictor


Histogram of residuals
Frequency
Residuals


Response vs. Fitted Values
Response
Fitted Values

El panel superior izquierdo es el *normal probability plot* de los residuos en devianza del modelo GAM. En comparación con la salidad de `plot(modelo.bf)` esta prueba canónica muestra mejor resultado en el modelo GAM que en el GLM. El panel inferior izquierdo representa el histograma de los residuos de devianza en vez del *normal probability plot*.

El panel superior derecho representa la relación entre los residuos en devianza del modelo y sus predicciones (operando con la transformación implícita en la *link function*). Observamos, de nuevo, que manifiesta “un poco” menos de heterocedasticidad que lo observado en `plot(modelo.bf)`, aunque la existencia de muchos valores de frecuencia nula en la variable respuesta genera un corte curvo abrupto en la parte inferior del *plot*.

Y para terminar con esta exploración gráfica del modelo GAM, en el panel inferior derecho se representa la relación entre valores observados y predichos por el modelo en la escala original de medida de la variable respuesta (frecuencia de aparición acotada entre cero y uno; i.e., no aplicando la transformación implícita en `link="logit"`).

Estimamos el **coeficiente de sobredispersión**, para corregir por este valor si es >1 sólo en las distribuciones binomiales y poisson

```
print(c("coef dispersión modelo GCV=", sum(residuals(modelo.gam, type="pearson")^2)/modelo.gam$df.residual), quote=FALSE)
[1] coef dispersión modelo GCV= 2.92626471897711
```

Como es bastante mayor que 1-UNO, deberíamos recalcular el modelo trabajando con una familia quasibinomial, o con `scale=-1` si no ha habido claros problemas de heterocedasticidad en los residuos en devianza del modelo GAM (i.e., no vamos a necesitar corregir por desvíos en el supuesto de heterocedasticidad). En esta ocasión vamos a ser “tiquismiquis” y vamos a crear un nuevo modelo haciendo uso de la *pseudo-familia* quasibinomial.

```
modelo.gam.qb <- gam(eqt.gam, data=datos, quasibinomial(link="logit"), weight=denominador, method="GCV.Cp", gamma=1.4, scale=0)
```

Ahora efectuemos el **ómnibus test** de significación global del modelo. Para ello vamos a utilizar el primer modelo (modelo.gam), reservando el segundo que hemos creado *quasibinomial* (modelo.gam.qb) para la estima de las significaciones de las predictoras.

```
lrtest(modelo.gam)
Likelihood ratio test
Model 1: ptyrup/denominador ~ s(altmed, bs = "tp", k = 10) + s(rangoalt,
    bs = "tp", k = 10) + s(shannon, bs = "tp", k = 10) + s(tempmin,
    bs = "tp", k = 10) + s(precip, bs = "tp", k = 10)
Model 2: ptyrup/denominador ~ 1                              ## este es el modelo nulo que no introduce efectos
     #Df  LogLik      Df  Chisq Pr(>Chisq)
1 36.434 -1031.1
2  1.000 -1692.3 -35.434 1322.3  < 2.2e-16 ***
```

Como hay un aparente desvío de la homocedasticidad, corrijamos su efecto; comparando además, cuál es su influencia “sin corregir” y “corrigiendo” (con y sin `vcov=sandwich`).

```
waldtest(modelo.gam)
Wald test
Model 1: ptyrup/denominador ~ s(altmed, bs = "tp", k = 10) + s(rangoalt,
    bs = "tp", k = 10) + s(shannon, bs = "tp", k = 10) + s(tempmin,
    bs = "tp", k = 10) + s(precip, bs = "tp", k = 10)
Model 2: ptyrup/denominador ~ 1                              ## este es el modelo nulo que no introduce efectos
  Res.Df  Df     F    Pr(>F)
1 962.57
2 998.00 -45 22.41 < 2.2e-16 ***

waldtest(modelo.gam, vcov=sandwich)
Wald test
Model 1: ptyrup/denominador ~ s(altmed, bs = "tp", k = 10) + s(rangoalt,
    bs = "tp", k = 10) + s(shannon, bs = "tp", k = 10) + s(tempmin,
    bs = "tp", k = 10) + s(precip, bs = "tp", k = 10)
Model 2: ptyrup/denominador ~ 1                              ## este es el modelo nulo que no introduce efectos
  Res.Df  Df      F    Pr(>F)
1 962.57
2 998.00 -45 11.255 < 2.2e-16 ***
```

En resumen, el modelo es muy significativo, y ahora ya podemos proceder con seguridad con la estima de los efectos de las predictoras. El efecto de la heterocedasticidad es aparentemente importante porque cambia bastante la F del test de Wald.

Resultados del modelo teniendo en cuenta los efectos que incluye. Con el modelo *quasibinomial*, por ser la sobredispersión alta.

```
summary(modelo.gam.qb)        ## modelo quasibinomial corregido por la sobredispersión
Family: quasibinomial
Link function: logit

Formula:
ptyrup/denominador ~ s(altmed, bs = "tp", k = 10) + s(rangoalt,
    bs = "tp", k = 10) + s(shannon, bs = "tp", k = 10) + s(tempmin,
    bs = "tp", k = 10) + s(precip, bs = "tp", k = 10)

Parametric coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -5.428      0.135   -40.2   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
             edf Ref.df      F  p-value
s(altmed)  2.923  3.683  3.755  0.00663 **
s(rangoalt) 5.595  6.733  7.474 1.98e-08 ***
s(shannon)  6.853  7.778  3.282  0.00124 **
s(tempmin)  7.658  8.472 10.771 3.76e-15 ***
s(precip)   5.644  6.796  4.963 2.20e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.339   Deviance explained = 46.2%
GCV = 1.6442  Scale est. = 2.8861    n = 999
```

El modelo explica el **46.2% de la** variabilidad en la variable respuesta estimada en **devianza**. Algo más que en los modelos previos GLM. Si llevamos a la variable respuesta a su escala original de medida (frecuencia de aparición sin aplicarle la *link function* logit), y corregimos la variabilidad explicada por el número de grados de libertad (**edf** que cuantifican la curvilinealidad de las relaciones), haciendo uso de la R^2, obtenemos que el modelo explica un **33.9% de la varianza**.

A continuación, por comparación con los resultados del modelo original (`modelo.gam`), podemos valorar cuál ha sido el efecto de la corrección por sobredispersión (que era muy alta: 2.926) sobre la complejidad de los splines y la significación de los efectos.

```
summary(modelo.gam)
Family: binomial
Link function: logit

Formula:
ptyrup/denominador ~ s(altmed, bs = "tp", k = 10) + s(rangoalt,
    bs = "tp", k = 10) + s(shannon, bs = "tp", k = 10) + s(tempmin,
    bs = "tp", k = 10) + s(precip, bs = "tp", k = 10)

Parametric coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -5.48216    0.08269   -66.3   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Approximate significance of smooth terms:
              edf Ref.df Chi.sq  p-value
s(altmed)   6.577  7.672  57.49 1.37e-09 ***
s(rangoalt) 6.544  7.666 142.46  < 2e-16 ***
s(shannon)  7.518  8.297  78.88 1.63e-13 ***
s(tempmin)  8.006  8.687 233.77  < 2e-16 ***
s(precip)   6.789  7.890  98.27  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

R-sq.(adj) =  0.341   Deviance explained = 47.1%
UBRE = 0.58756  Scale est. = 1         n = 999
```

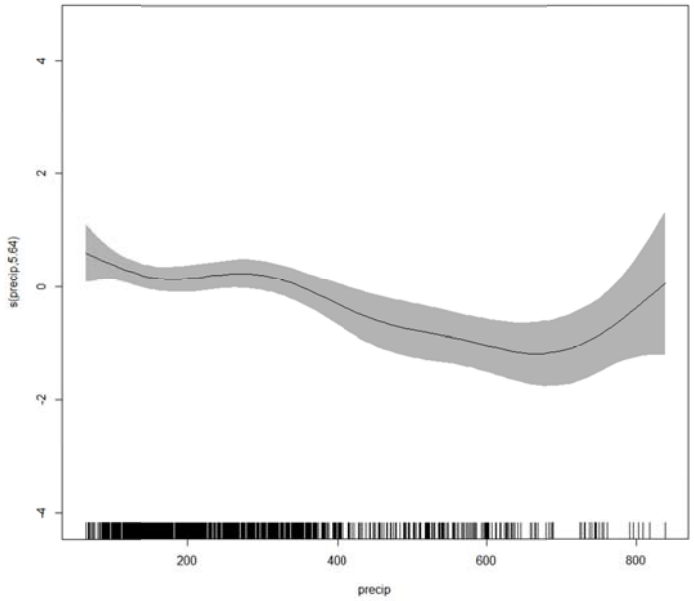
Al haber corregido por la sobredispersión estimada en 2.926, vemos que la complejidad de los *splines* se ha simplificado sustancialemnete. En cuanto a la variabilidad explicada (tanto en devianza como en R^2 ajustada) obtenemos valores muy similares en el modelo *binomial* y *quasibinomial*.

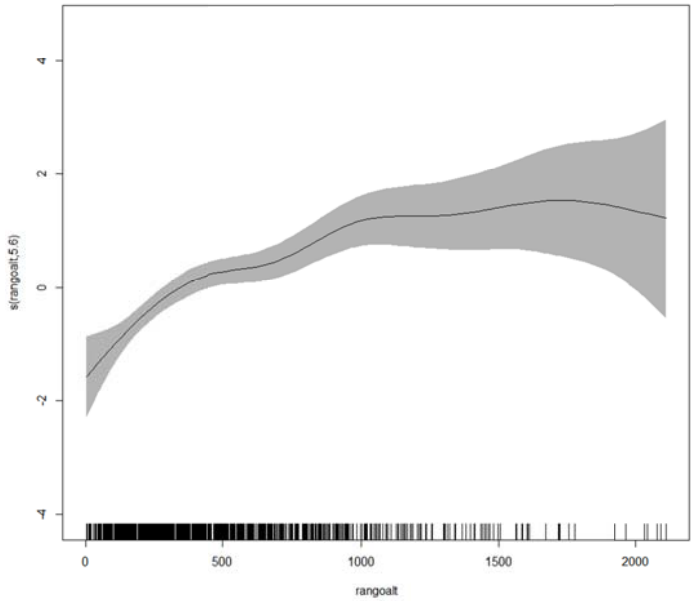
Como es así, ¿por qué complicarnos con un modelo complejo cuando explicamos casi lo mismo de la variabilidad en la variable respuesta? Por este motivo, nos vamos a quedar con el modelo *quasibinomial* para explorar visualmente los efectos de las predictoras (`modelo.gam.qb`).

Y decimos "explorar visualmente los efectos de las predictoras" porque los Modelos Generalizados ADITIVOS no estiman coeficientes de regresión numéricos "sencillos" como obteníamos en los modelo GLM. Son estimas "no paramétricas" de efectos que sólo podemos verbalizar viéndolos con los *partial residual plots.* Y al ser *residual plots* representan los valores residuales en devianza. Las salidas gráficas podemos verlas simultáneamente con muchos paneles para todos los efectos juntos (… muy pequeñitas), o de modo más claro de-uno-en-uno. Optemos por esta segunda opción.

```
par(mfcol=c(1,1))
plot(modelo.gam.qb, shade=T, shade.col="gray70", all.terms=T)    ## y damos [Intro] cada vez para ver un Nuevo plot
```

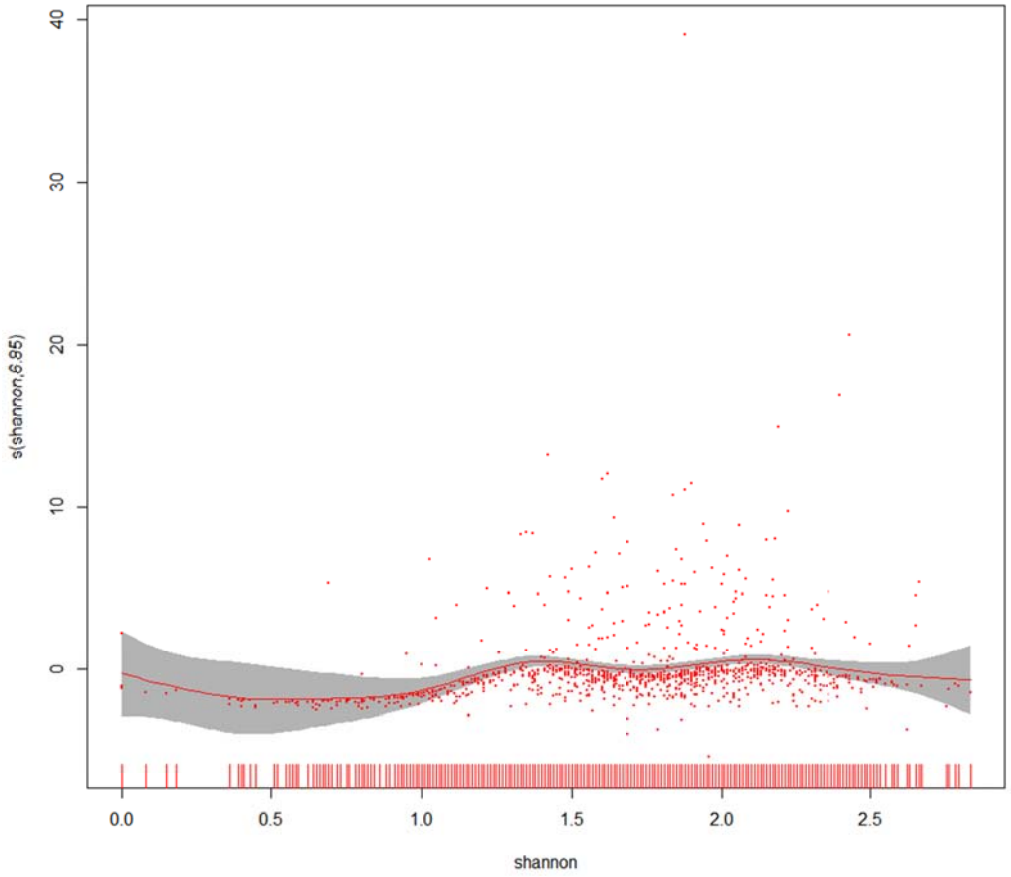
Por ejemplo, para precip y rangoalt. Lo que vemos es la variación residual en la respuesta (frecuencia de aparición del avión roquero) teniendo en cuenta el efecto de la **precip**itación (que tiene una complejidad definida por el *thin plate spline* de **edf = 5.64**)





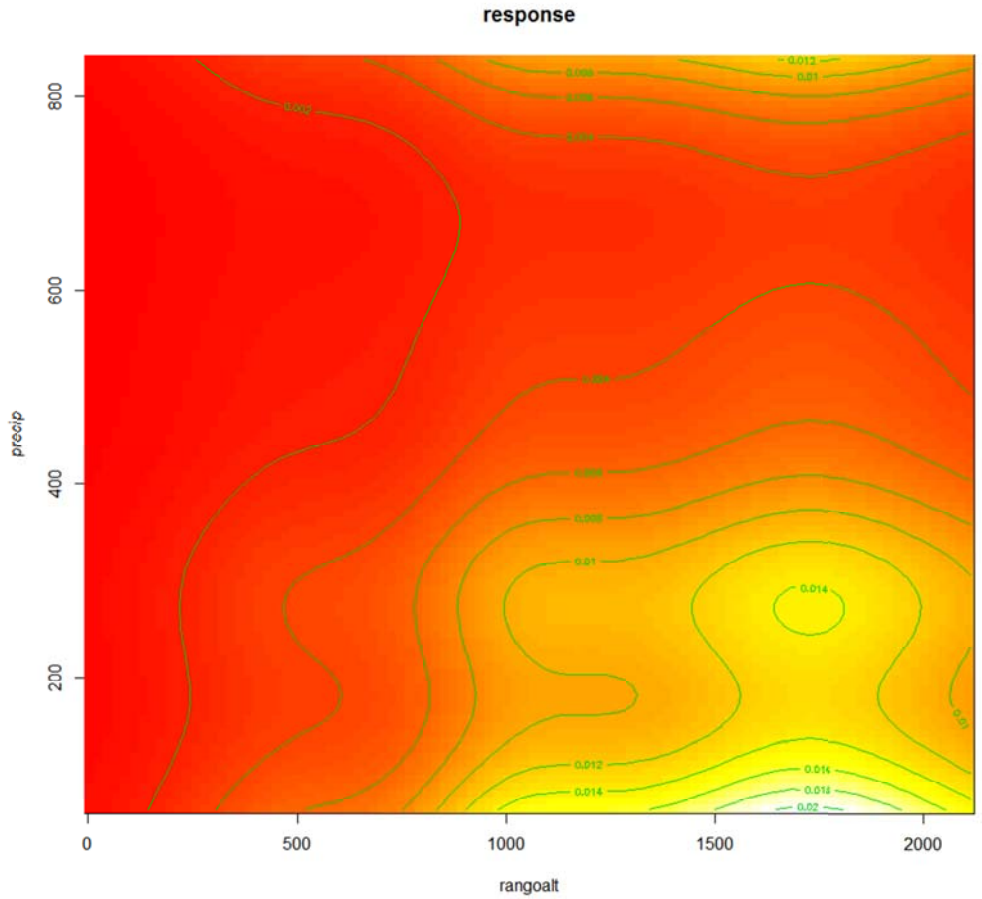
Otra posibilidad gráfica es representar, además, los puntos de las unidades muestrales. ¡Ah! Las rayitas verticales sobre el eje de las X's indican dónde se posicionan las 999 unidades muestrales en el rango de variación de las variables predictoras.

```
plot(modelo.gam.qb, shade=T, shade.col="gray70", all.terms=T, residuals=T, pch=19, cex=0.1, scheme=1, col='red')
```



Como los efectos de la montañosidad (rangoalt) y la precipitación (precip) son muy importantes cuantitativamente en la distribución invernal del avión roquero (valoradlo en `summary(modelo.gam)` como significación y devianza retenida por esos efectos medida por la `Chi.sq`; y de modo similar pero, con el valor de la F, en `summary(modelo.gam.qb)`), podemos crear plots de interacciones.

```
vis.gam(modelo.gam.qb, view=c("rangoalt", "precip"), type = "response", zlim=c(0,0.02), plot.type = "contour", n.grid=100)
```



Debemos añadir en `view=c(…)` las variables de interés “entrecomilladas”, y en `zlim=c(…)` el rango de variación de la respuesta que queramos visualizar. Tened en cuenta que el rango de variación en esta ocasión viene dado por una frecuencia escalada entre cero-y-uno (`type="response"`), que estamos representando un efecto parcial “controlando por las otras variables consideradas en `modelo.gam.qb`, y que la variable respuesta en la escala original toma predominantemente valores muy pequeños (de ahí `zlim=c(0,0.02)`). A color más claro, más abundancia del avión roquero en el plano definido por la montañosidad (rangoalt) y la precipitación (precip).

Hasta ahora hemos analizado el poder explicativo de nuestro modelo GAM. Valoremos ahora su poder predictivo aplicado a otro juego de datos no empleado en su construcción, como ya hemos hecho en las tres ocasiones previas con GLM Binomial Negativa, Hurdle Binomial Negativa y GLM Binomial frecuencial.

Para ello, antes creamos una variable llamada `denominador` en nuestro juego de datos `predecir` que define que siempre hay 60 transectos como denominador.

```
predecir$denominador <- seq(60,60, length.out=689)
predecir$ptyrup.nuevo.gam.qb <- predict(modelo.gam.qb, newdata=predecir, type="response")
```

Como también contamos en el juego de datos `predecir` con la variable respuesta `ptyrup` medida realmente en el campo, podemos efectuar un test del poder predictivo del modelo `modelo.gam.qb`. Para ello, relacionamos los datos de la variable respuesta medidos pero no utilizados en `modelo.gam.qb` (`predecir$ptyrup`) con los predichos por él (`predecir$ptyrup.nuevo.gam.qb`).

```
summary(lm(I(predecir$ptyrup/predecir$denominador) ~ predecir$ptyrup.nuevo.gam.qb))      ## I(…) indica que calculamos algo
Call:
lm(formula = I(predecir$ptyrup/predecir$denominador) ~ predecir$ptyrup.nuevo.gam.qb)
Residuals:
     Min       1Q   Median       3Q      Max
-0.19162 -0.00937 -0.00467 -0.00311  0.38063
Coefficients:
                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                   0.002470   0.001663   1.485    0.138
predecir$ptyrup.nuevo.gam.qb 0.971417   0.059745  16.259   <2e-16 ***
```

```
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.03861 on 687 degrees of freedom
Multiple R-squared:  0.2779,   Adjusted R-squared:  0.2768
F-statistic: 264.4 on 1 and 687 DF,  p-value: < 2.2e-16
```

Vemos que el poder predictivo del modelo es algo flojo (**R^2 = 27.8%**), significativo (**p<<0.001**), y el modelo GAM (`modelo.gam.qb`) no tiende a sobreestimar la abundancia relativa del avión roquero (nuestra variable respuesta, `ptyrup`), porque la pendiente de `predecir$ptyrup.nuevo.gam.qb` es **0.971**, no diferente del valor esperable de **1** si OBSERVADO fuese igual a PREDICHO (**OBSERVADO = 0 + 1 * PREDICHO**):

```
(1-0.971417) / 0.059745      ## esto es una t de Student de desvío de un coeficiente de regresión del valor 1
[1] 0.4784166

2*(1-pt(0.4784166, df=687))   ## cálculo de la p para una t de Student, con dos colas
[1] 0.6325059
```

Y sólo constatar que la ordenada en el origen de la ecuación no difiere de cero (mirad en los resultados previos).

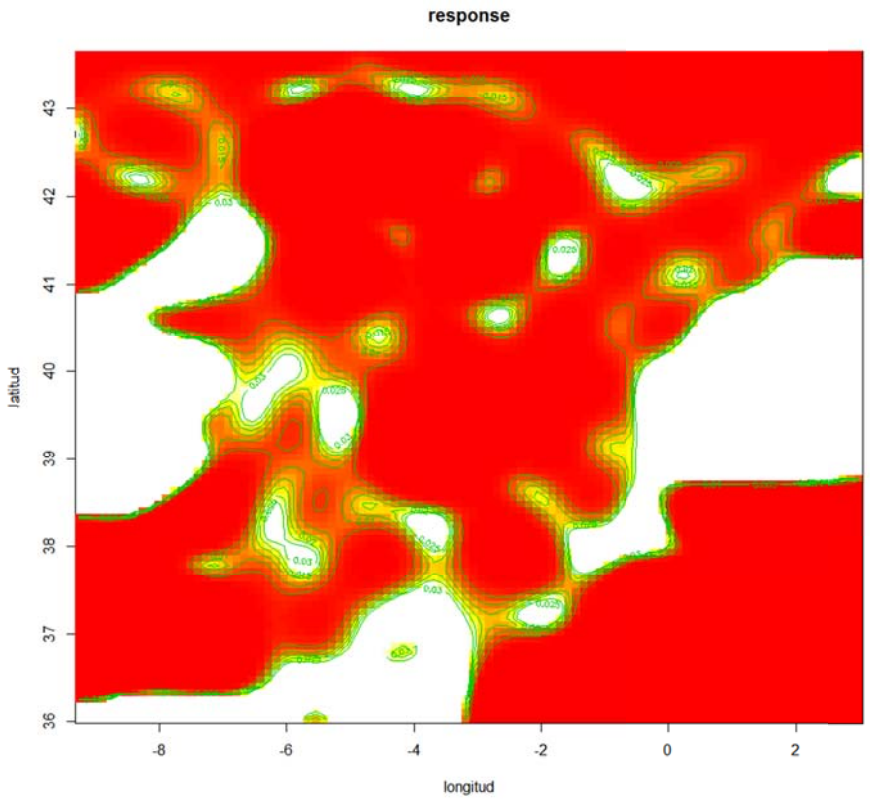
```
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                 0.002470   0.001663   1.485    0.138
```

Esto es, el modelo GAM (`modelo.gam.qb`) es más preciso en sus predicciones porque cumple de mejor manera con el postulado **OBSERVADO = 0 + 1 * PREDICHO**, pero tiene un poder predictivo (R^2 observado-predicho) un poco menor (comparadlo con el valor previo de `R-sq.(adj) =  0.339` en el `modelo.gam.qb`).

Construyamos otro modelo diferente, más sencillo, que incluya un efecto “geografía”. Tal efecto no lo “capturamos” con una variable concreta, si no con un “constructo” definido por dos variables distintas pero vinculadas al concepto “geografía”: la latitud y la longitud. Para ello debemos definir un plano de interacción [latitud x longitud], al cual le vamos a aplicar un *tensión spline* utilizando **te(…)** en vez de **s(…)** que hemos utilizado hasta ahora. Este *tensión espline* usando **te(…)** gestiona las anisotropías (medida de las variables predictoras en distinta escala o unidades).

Para ello calculamos un nuevo modelo (usando la pseudo-familia *quasibinomial* porque ya hemos visto que existe bastante sobredispersión). Y nos saltamos … todos los pasos de análisis subsiguientes por no ser repetitivos. Realmente esto lo hacemos para ver la representación de un *tensión spline*. Primero la ecuación y el modelo; luego la representación. ¡Allá vamos!

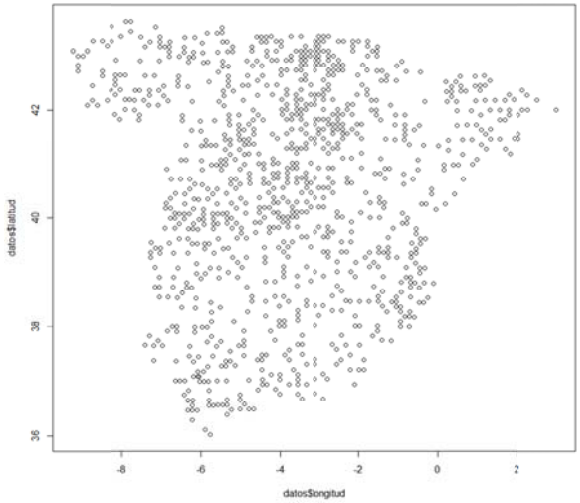
```
eqt.gam.geogr <- as.formula(ptyrup/denominador ~ te(longitud, latitud, bs="tp", k=15))
modelo.gam.qb.geogr <- gam(eqt.gam.geogr, data=datos, quasibinomial(link="logit"), weight=denominador, method="GCV.Cp", gamma=1.4, scale=0)

vis.gam(modelo.gam.qb.geogr, view=c("longitud", "latitud"), type = "response", zlim=c(0,0.03), plot.type = "contour", n.grid=100)
```



Las zonas ROJAS denotan ausencia, mientras que los tonos claros son indicativos de máxima abundancia.

Tengamos en cuenta que se efectúan predicciones fuera del ámbito analizado (mar al este y Portugal al oeste).

```
plot(datos$longitud, datos$latitud)
```



Además, vamos a generar el modelo anterior sin hacer uso de la pseudo-familia *quasibinomial*, porque queremos obtener su valor de Akaike (AICc). Y esto ¿por qué? Porque no se puede estimar AIC con pseudo-familias "quasi".

```
modelo.gam.geogr <- gam(eqt.gam.geogr, data=datos, binomial(link="logit"), weight=denominador, method="GCV.Cp", gamma=1.4, scale=0)
```

Actualizamos el modelo GAM con el que hemos estado trabajando hasta ahora, añadiéndole el término *tensión spline*. Para ello vamos a hacer uso del comando `update`. Pero como en el caso inmediatamente previo, lo vamos a hacer sin aplicar la pseudo-familia *quasibinomial*, porque queremos obtener su valor de Akaike (AICc).

```
modelo.gam.total <- update (modelo.gam, .~. +te(longitud, latitud, bs="tp", k=15))
```

Ya casi terminamos. Vamos a actualizar el modelo GAM original dejando sólo los efectos de variables climáticas, eliminando las otras. Volvemos a utilizar el comando `update`, para crear un modelo GAM de sólo clima:

```
modelo.gam.clima <- update (modelo.gam, .~. -s(altmed, bs="tp", k=10) -s(rangoalt, bs="tp", k = 10) -s(shannon, bs="tp", k=10))
```

Y rescatemos nuestro modelo GLM Binomial de frecuencia (`modelo.bf`) y el original GAM (`modelo.gam`).

Comparemos ahora los cuatro modelos (que trabajan con los mismos datos, la misma escala de medida de la variable respuesta, la misma familia y *link function* definida para la respuesta, y modelos Generalizados utilizando *máximum likelihood*). Para ello recurrimos a los valores de AIC de Akaike, pero teniendo en cuenta el tamaño muestral y los grados de libertad retenidos por los efectos incluidos en los modelos: valores de **AICc**, en vez de AIC.

```
AICc(modelo.bf, modelo.gam.clima, modelo.gam.geogr, modelo.gam, modelo.gam.total)
                      df     AICc
modelo.bf           6.00000 2358.038
modelo.gam.clima   16.56778 2452.471
modelo.gam.geogr  167.13107 1837.574
modelo.gam         36.43429 2137.881
modelo.gam.total  197.24163 1675.073
```

Claramente, el modelo GAM completo que introduce las cinco variables predictoras y el término *tensión spline* "geografía" (`modelo.gam.total`) es el mejor modelo (por tener menor valor de AICc).

Teniendo en cuenta las cinco variables predictoras en común, el modelo GAM es mejor que el modelo GLM en 220.2 unidades de AICc (¡eso es una enormidad!). Recordad [1 / exp(-220.2/2)] = 6.5e+47 veces mejor `modelo.gam` que `modelo.bf`.

El modelo de sólo clima (`modelo.gam.clima`) es enormemente peor que el de sólo geografía (`modelo.gam.geogr`).

El modelo de sólo geografía (`modelo.gam.geogr`) es sustancialmente mejor que el modelo ambiental original (`modelo.gam`).

Y el modelo completo (`modelo.gam.total`) que incluye la posición geográfica es mejor el modelo ambiental original (`modelo.gam`).

De este último aspecto también podemos hacer el test de Vuong para compararlos (comando `vuong` del paquete `pscl`):

```
vuong(modelo.gam, modelo.gam.geogr)                              ## en el orden: modelo1, modelo2
NA or numerical zeros or ones encountered in fitted probabilities
dropping these 213 cases, but proceed with caution
[1] -596.6926
          Raw AIC-corrected BIC-corrected
    -6.116025     -5.427366   1798.246913
[1] 0.01179548
[1]  -2.022535  -1.794799 594.670098
Vuong Non-Nested Hypothesis Test-Statistic:
(test-statistic is asymptotically distributed N(0,1) under the
 null that the models are indistinguishible)
-------------------------------------------------------------
              Vuong z-statistic             H_A    p-value
Raw                   -6.116025 model2 > model1 4.7969e-10
AIC-corrected         -5.427366 model2 > model1 2.8596e-08
```

¡¡ Y ESTO ES TODO !! ESPERO QUE OS HAYA SIDO DE UTILIDAD ESTE REPASO PRÁCTICO.

¡¡ YA PODÉIS VOLAR SOLOS !!

… como el avión **R**oquero