chapter4 - Regression and Prediction

LuisF

2/6/2020

Regresión Logística Simple

Modela la relación entre ña magnitud de una variable 'x' y una 'y'. Por ejemplo, si x aumenta y aumenta y visceversa. La correlación entre dos variables mide lo mismo, sin embargo, la regresión incluye la cuantificación de la naturaleza de esa relación, además de la fuerza con la que se co-re-lacionan.

Formula:

- Y: variable dependiente o lo que vas a predecir
- x: variable independiente
- b0: intercepto, es el valor de 'y' cuando el valor de las variables independientes es igual a 0. ¿Qué pasa si tienes un modelo con x's que no puede ser 0 (como la edad)?
  - En ese caso puedes 'centrar' a los predictores y en ese caso, tu valor para centrar se convierte en el intercepto
  - Leyendo un poco sobre este concepto, es inusual que te pueda servir para interpretar los datos. Generalmente, solo funciona para que tu ecuación de regresión prediga bien los valores. sin embargo, hay varios escenarios en donde se puede interpretar (qué pasa con tu modelo si tienes x-categóricas o x-dummies?)
- b1: The slope of a regression line (b) represents the rate of change in y as x changes. Because y is dependent on x, the slope describes the predicted values of y given x. When using the ordinary least squares method, one of the most common linear regressions, slope, is found by calculating b as the covariance of x and y, divided by the sum of squares (variance) of x,

```
df <- read.table('D:/Documentos/R_Projects/PSDS/datos_practical_statistics/LungDisease.csv', sep = ",",
head(df)
```

```
##   PEFR Exposure
## 1  390        0
## 2  410        0
## 3  430        0
## 4  460        0
## 5  420        1
## 6  280        2
```

```
model <- lm(PEFR    ~   Exposure,   data=df)
model #'linear model' te arroja el valor del intercepto y la pendiente.
```

```
##
## Call:
## lm(formula = PEFR ~ Exposure, data = df)
##
## Coefficients:
## (Intercept)     Exposure
##     424.583       -4.185
```

```
#predict(model) #los valores de y-hat
#residuals(model) #residuales: valor original - valor predicción
```

Mínimos cuadrados

Método que minimiza los residuales (Residual Sum of Squares) y permite encontrar b0 y b1.

Comparando los resultados de la función lm y mínimos cuadrados:

Mínimos cuadrados pagina web:

```
#https://miprofe.com/minimos-cuadrados/
head(df)
```

```
##   PEFR Exposure
## 1  390        0
## 2  410        0
## 3  430        0
## 4  460        0
## 5  420        1
## 6  280        2
```

```
df$xy <- df$PEFR * df$Exposure
df$x2 <- df$PEFR^2

b1 <- (sum(df$xy) - nrow(df) * mean(df$PEFR) * mean(df$Exposure)) / (sum(df$x2) - nrow(df)*mean(df$PEFR
b0 <- mean(df$Exposure) - b1*mean(df$PEFR)
```

Mínimos cuadrados formula libro:

```
head(df)
```

```
##   PEFR Exposure  xy     x2
## 1  390        0   0 152100
## 2  410        0   0 168100
## 3  430        0   0 184900
## 4  460        0   0 211600
## 5  420        1 420 176400
## 6  280        2 560  78400
```

```
df$residual <- df$Exposure - mean(df$Exposure)
df$xresid <- df$PEFR - mean(df$PEFR)
df$xresid2 <- df$xresid^2

sum(df$residual)*sum(df$xresid)/sum(df$xresid2)
```

```
## [1] -1.884493e-31
```

"To explain or to predict" https://projecteuclid.org/download/pdfview_1/euclid.ss/1294167961

Regresión lineal múltiple

King County Housing Data

```
house <- read.table('D:/Documentos/R_Projects/PSDS/datos_practical_statistics/house_sales.csv')
head(house)
```

```
##   DocumentDate SalePrice PropertyID  PropertyType         ym zhvi_px  zhvi_idx
## 1   2014-09-16    280000    1000102     Multiplex 2014-09-01  405100 0.9308364
```

```
## 2   2006-06-16   1000000    1200013 Single Family 2006-06-01  404400 0.9292279
## 3   2007-01-29    745000    1200019 Single Family 2007-01-01  425600 0.9779412
## 4   2008-02-25    425000    2800016 Single Family 2008-02-01  418400 0.9613971
## 5   2013-03-29    240000    2800024 Single Family 2013-03-01  351600 0.8079044
## 6   2009-03-30    349900    3600090     Townhouse 2009-03-01  369800 0.8497243
##   AdjSalePrice NbrLivingUnits SqFtLot SqFtTotLiving SqFtFinBasement Bathrooms
## 1       300805              2    9373          2400              0      3.00
## 2      1076162              1   20156          3764           1452      3.75
## 3       761805              1   26036          2060            900      1.75
## 4       442065              1    8618          3200           1640      3.75
## 5       297065              1    8620          1720              0      1.75
## 6       411781              1    1012           930              0      1.50
##   Bedrooms BldgGrade YrBuilt YrRenovated TrafficNoise LandVal ImpsVal ZipCode
## 1        6         7    1991           0            0   70000  229000   98002
## 2        4        10    2005           0            0  203000  590000   98166
## 3        4         8    1947           0            0  183000  275000   98166
## 4        5         7    1966           0            0  104000  229000   98168
## 5        4         7    1948           0            0  104000  205000   98168
## 6        2         8    2008           0            0  170000  207000   98144
##   NewConstruction
## 1           FALSE
## 2            TRUE
## 3           FALSE
## 4           FALSE
## 5           FALSE
## 6            TRUE
```

```
head(house[, c("AdjSalePrice",  "SqFtTotLiving",    "SqFtLot",  "Bathrooms", "Bedrooms",    "BldgGrade"
```

```
##   AdjSalePrice SqFtTotLiving SqFtLot Bathrooms Bedrooms BldgGrade
## 1       300805          2400    9373      3.00        6         7
## 2      1076162          3764   20156      3.75        4        10
## 3       761805          2060   26036      1.75        4         8
## 4       442065          3200    8618      3.75        5         7
## 5       297065          1720    8620      1.75        4         7
## 6       411781           930    1012      1.50        2         8
```

```
house_lm    <-  lm(AdjSalePrice ~   SqFtTotLiving   +   SqFtLot +   Bathrooms   +
house_lm
```

```
##
## Call:
## lm(formula = AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
##     Bedrooms + BldgGrade, data = house, na.action = na.omit)
##
## Coefficients:
##   (Intercept)  SqFtTotLiving        SqFtLot      Bathrooms       Bedrooms
##    -5.219e+05      2.288e+02     -6.051e-02     -1.944e+04     -4.778e+04
##     BldgGrade
##     1.061e+05
```

Evaluando el modelo

La más importante medida para evaluar un modelo de regresión es el RMSE (Root Mean Squared Error)/Error cuadrático medio. Que la formula es la raíz cuadrada de los "residuals" al cuadrado entre "n":

```r
head(house$AdjSalePrice - predict(house_lm) == residuals(house_lm)) #aunque no sean exactamente iguales
```

```
##     1     2     3     4     5     6
## FALSE FALSE FALSE FALSE FALSE FALSE
```

```r
head(mean(house$AdjSalePrice - predict(house_lm) -  residuals(house_lm))) #practicamente lo son...
```

```
## [1] -1.904798e-07
```

```r
#entonces es lo mismo hacer esto:
house$residuales <- house$AdjSalePrice - predict(house_lm)
#que esto:
#house$residuales <- residuals(house_lm)

#error cuadrático medio
house$residuales2 <- house$residuales^2
rmse <- sqrt(sum(house$residuales2)/nrow(house))
head(rmse)
```

```
## [1] 261209.7
```

Similar al error cuadrático medio esta el error estandard, el cual, en lugar de utilizar 'n' como denominador, utiliza grados de confianza:

```r
#RSE(Residual standard error)
rse <- sqrt(sum(house$residuales2)/nrow(house)-1)
rse
```

```
## [1] 261209.7
```

```r
summary(house_lm) #el "Residual standard error que arroja la función de lm es = a 261,200 vs 261,209 de
```

```
##
## Call:
## lm(formula = AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
##     Bedrooms + BldgGrade, data = house, na.action = na.omit)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1199508  -118879   -20982    87414  9472982
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.219e+05  1.565e+04 -33.349  < 2e-16 ***
## SqFtTotLiving  2.288e+02  3.898e+00  58.699  < 2e-16 ***
## SqFtLot       -6.051e-02  6.118e-02  -0.989    0.323
## Bathrooms     -1.944e+04  3.625e+03  -5.362 8.32e-08 ***
## Bedrooms      -4.778e+04  2.489e+03 -19.194  < 2e-16 ***
## BldgGrade      1.061e+05  2.396e+03  44.287  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 261200 on 22683 degrees of freedom
## Multiple R-squared:  0.5407, Adjusted R-squared:  0.5406
## F-statistic:  5340 on 5 and 22683 DF,  p-value: < 2.2e-16
```

El autor menciona que en la práctica no hay diferencia para estos dos parámetros...

Otra medida de evaluación es el 'coeficiente de determinación' o R-cuadrado, el cual mide la proporción de variabilidad entre lo predicho y los valores reales:

```
#house$residuales2 #se utilizara esta info, y...
house$difmean2 <- (house$AdjSalePrice - mean(house$AdjSalePrice))^2
#r-cuadrado
r2 <- 1 - (sum(house$residuales2)/sum(house$difmean2))
r2
```

```
## [1] 0.5406642
```

```
summary(house_lm) #el r2 de la fórmula tambien utiliza grados de confianza, y dio muy similar al que ca
```

```
##
## Call:
## lm(formula = AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
##     Bedrooms + BldgGrade, data = house, na.action = na.omit)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -1199508  -118879   -20982    87414  9472982
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.219e+05  1.565e+04 -33.349  < 2e-16 ***
## SqFtTotLiving  2.288e+02  3.898e+00  58.699  < 2e-16 ***
## SqFtLot       -6.051e-02  6.118e-02  -0.989    0.323
## Bathrooms     -1.944e+04  3.625e+03  -5.362 8.32e-08 ***
## Bedrooms      -4.778e+04  2.489e+03 -19.194  < 2e-16 ***
## BldgGrade      1.061e+05  2.396e+03  44.287  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 261200 on 22683 degrees of freedom
## Multiple R-squared:  0.5407, Adjusted R-squared:  0.5406
## F-statistic:  5340 on 5 and 22683 DF,  p-value: < 2.2e-16
```

Cross-Validation/Validación Cruzada

Ya ves que cuando entrenas un modelo lo haces solo con un % de los datos, para después correr el modelo en el % restante y evaluarlo bajo datos que no había visto. La idea de este método es utilizar distintos sampleos para entrenar y evaluar, y no solo quedarte con una corrida por asi decirlo. Es importante recalcar que, cuando vayas a realizar la 2da, 3er o 'n' corrida, los datos que decidiste dejar fuera para luego evaluar con ellos, no pueden ser elegidos para la siguiente corrida.

Model selection and Step-wise regression

En este apartado te exponen el AIC(Akaike's Information Criteria), la cual es una métrica para evaluar el comportamiento de tu modelo de REGRESION: entre menor sea este valor, mejor es el modelo (existen otras métricas aparte de esta como por ejemplo BIC or Bayesian y Mallows CP, sin embargo, menciona el autor que no tienen mucha diferencia para el campo de ciencia de datos).

Ahora, irte creando modelo tras modelo, cambiando variables al tanteo, puede ser tedioso, y por otro lado, utilizar el 'subset regresion' el cual es un metodo para probar todos los posibles modelos, puede ser muy demandante para un set de datos 'big'. Para esto, se utiliza 'setpwise regression', el cual es un método que añade y dropea varialbes para encontrar el menor AIC:

```
#creas un modelo con más variables que el anterior
house_full  <-  lm(AdjSalePrice ~   SqFtTotLiving   +   SqFtLot + Bathrooms + Bedrooms  +   BldgGrade

#library(MASS)
#step   <-  stepAIC(house_full, direction="both")
#step

#?stepAIC
#Notas de la función stepAIC:
#forward direction: empieza con 0 variables y va agregando 1 x 1 con base a su R2. Termina de evaluar c

#Ya metiendo un poco más en la confiabilidad de este método, me encontre con esto:
#https://stats.stackexchange.com/questions/20836/algorithms-for-automatic-model-selection/20856#20856
#aca, al final de cuentas encontre un comentario que menciona el uso de 'caret', una librería en R para

#En resumen puedo decir que utilizar estos métodos automatizados para encontrar un modelo óptimo, tiene
```

Weighted Regresssion

No me metí a fondo como funciona esto, pero la función 'lm' tiene un parámetro que se llama 'weight', el cual puedes utilizar para dar más relevancia a ciertos datos. Por ejemplo, si quieres predecir el precio de una casa, los datos que contengan precios más actuales pueden que den más relevancia para un modelo actualizado al día de hoy.

Prediction using Regression

Para la ciencia de datos, la regresión lineal se utiliza principalmente para la predicción de un valor

Los peligros de Extrapolar

Un modelo de regresión solo tendra sentido cuando le metas datos que esten dentro del rango de los datos utilizados para su entrenamiento. Es decir, si quieres predecir el precio de una casa para variables x = a 0, el precio no tendra sentido, ya que no hay casas que cuesten 0 pesos en tu dataset (eso se esperaría)

Intervalos de confianza y predicción

En lugar de utilizar una métrica como la media para obtener intervalos de confianza, que te permiten medir la variabilidad del modelo, utilizas los coeficientes del modelo para crear estos intervalos, utilizando bootstrap (sampleos con reemplazo). Además de hacerlo de esta manera, también puedes hacerlo mediante el 'individual data point error', en donde corres tu módelo y tomas un 'x' data point de tus residuals y lo sumas a tu valor predicho. Haces esto 10000 (n) veces y sacas percentiles (5 - 95)

Variables categóricas

¿Cómo convertirlas a dummy?

```
house <- read.table('D:/Documentos/R_Projects/PSDS/datos_practical_statistics/house_sales.csv')
#head(house)
#house[, c("AdjSalePrice",  "SqFtTotLiving",    "SqFtLot",  "Bathrooms", "Bedrooms",    "BldgGrade")]
#valuecounts:
aggregate(data.frame(count = house$PropertyType),       # Apply aggregate function
          list(value = house$PropertyType),
          length)
```

```
##           value count
## 1     Multiplex   257
```

```
## 2 Single Family 20722
## 3     Townhouse  1710
```

```
#Entonces podemos ver que hay 3 valores únicos para "tipo de propiedad"

##Ahora convirtiendo a dummy:

#De esta manera lo puedes hacer manual
prop_type_dummies <- model.matrix(~PropertyType -1, data=house)
head(prop_type_dummies)
```

```
##   PropertyTypeMultiplex PropertyTypeSingle Family PropertyTypeTownhouse
## 1                     1                         0                     0
## 2                     0                         1                     0
## 3                     0                         1                     0
## 4                     0                         1                     0
## 5                     0                         1                     0
## 6                     0                         0                     1
```

```
#?model.matrix

#Pero, la función 'lm' lo hace automáticamente y omite el primer nivel de cada variable categórica, por
lm(AdjSalePrice ~   SqFtTotLiving   +   SqFtLot +   Bathrooms   + Bedrooms  +       BldgGrade   +   Pro
```

```
##
## Call:
## lm(formula = AdjSalePrice ~ SqFtTotLiving + SqFtLot + Bathrooms +
##     Bedrooms + BldgGrade + PropertyType, data = house)
##
## Coefficients:
##               (Intercept)              SqFtTotLiving
##                -4.469e+05                  2.234e+02
##                   SqFtLot                  Bathrooms
##                -7.041e-02                 -1.597e+04
##                  Bedrooms                  BldgGrade
##                -5.090e+04                  1.094e+05
## PropertyTypeSingle Family      PropertyTypeTownhouse
##                -8.469e+04                 -1.151e+05
```

```
#si te fijas en los coeficientes que arroja el modelo, el factor "Multiplex" no aparece.
```

¿Qué pasa cuando tienes muchos niveles en un factor?

Como por ejemplo, varios códigos postales:

```
zip_groups <- house %>%
                mutate(resid    =   residuals(house_lm))    %>%
                group_by(ZipCode)   %>%
                summarise(med_resid =   median(resid),
                          cnt   =   n())    %>%
                arrange(med_resid)  %>%
                mutate(cum_cnt  =   cumsum(cnt),
                       ZipGroup =   ntile(cum_cnt,  5))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```r
house <- house %>%
      left_join(select(zip_groups, ZipCode, ZipGroup),by='ZipCode')
head(house)
```

```
##   DocumentDate SalePrice PropertyID  PropertyType         ym zhvi_px  zhvi_idx
## 1   2014-09-16    280000    1000102     Multiplex 2014-09-01  405100 0.9308364
## 2   2006-06-16   1000000    1200013 Single Family 2006-06-01  404400 0.9292279
## 3   2007-01-29    745000    1200019 Single Family 2007-01-01  425600 0.9779412
## 4   2008-02-25    425000    2800016 Single Family 2008-02-01  418400 0.9613971
## 5   2013-03-29    240000    2800024 Single Family 2013-03-01  351600 0.8079044
## 6   2009-03-30    349900    3600090     Townhouse 2009-03-01  369800 0.8497243
##   AdjSalePrice NbrLivingUnits SqFtLot SqFtTotLiving SqFtFinBasement Bathrooms
## 1       300805              2    9373          2400              0      3.00
## 2      1076162              1   20156          3764           1452      3.75
## 3       761805              1   26036          2060            900      1.75
## 4       442065              1    8618          3200           1640      3.75
## 5       297065              1    8620          1720              0      1.75
## 6       411781              1    1012           930              0      1.50
##   Bedrooms BldgGrade YrBuilt YrRenovated TrafficNoise LandVal ImpsVal ZipCode
## 1        6         7    1991           0            0   70000  229000   98002
## 2        4        10    2005           0            0  203000  590000   98166
## 3        4         8    1947           0            0  183000  275000   98166
## 4        5         7    1966           0            0  104000  229000   98168
## 5        4         7    1948           0            0  104000  205000   98168
## 6        2         8    2008           0            0  170000  207000   98144
##   NewConstruction ZipGroup
## 1           FALSE        3
## 2            TRUE        3
## 3           FALSE        3
## 4           FALSE        3
## 5           FALSE        3
## 6            TRUE        4
```

```r
#?cumsum
```

```r
#Justificación de la fórmula para crear los grupos anteriores:

#La idea es usar los residuales (los cuales son errores en los precios de venta(y)) y ver qué códigos p
### Quisiera saber cuál seria la diferencia de hacer esta agrupación directamente con los precios reale
house %>%
  group_by(ZipCode) %>%
  mutate(me = mean(SalePrice),
            counte = n())
```

```
## # A tibble: 22,689 x 25
## # Groups:   ZipCode [82]
##    DocumentDate SalePrice PropertyID PropertyType ym    zhvi_px zhvi_idx
##    <chr>            <int>      <dbl> <chr>        <chr>   <int>    <dbl>
##  1 2014-09-16      280000    1000102 Multiplex    2014~  405100    0.931
##  2 2006-06-16     1000000    1200013 Single Fami~ 2006~  404400    0.929
##  3 2007-01-29      745000    1200019 Single Fami~ 2007~  425600    0.978
##  4 2008-02-25      425000    2800016 Single Fami~ 2008~  418400    0.961
##  5 2013-03-29      240000    2800024 Single Fami~ 2013~  351600    0.808
##  6 2009-03-30      349900    3600090 Townhouse    2009~  369800    0.850
##  7 2013-08-28      327500    3800004 Single Fami~ 2013~  374300    0.860
```

```
##  8 2007-05-24      347000    3800009 Single Fami~ 2007~  432100    0.993
##  9 2006-09-22      220400    6600055 Single Fami~ 2006~  414800    0.953
## 10 2006-08-22      437500    7200080 Multiplex    2006~  411100    0.945
## # ... with 22,679 more rows, and 18 more variables: AdjSalePrice <dbl>,
## #   NbrLivingUnits <int>, SqFtLot <int>, SqFtTotLiving <int>,
## #   SqFtFinBasement <int>, Bathrooms <dbl>, Bedrooms <int>, BldgGrade <int>,
## #   YrBuilt <int>, YrRenovated <int>, TrafficNoise <int>, LandVal <int>,
## #   ImpsVal <int>, ZipCode <int>, NewConstruction <lgl>, ZipGroup <int>,
## #   me <dbl>, counte <int>
```

```
#Lo que hace esta parte del código es ir acumulando los conteos individuales de los códigos postales.
?ntile
```

```
## starting httpd help server ... done
```

```
#¿Cuántos valores quedaorn para cada grupo de zipcode?
aggregate(data.frame(count = house$ZipGroup),       # Apply aggregate function
          list(value = house$ZipGroup),
          length)
```

```
##   value count
## 1     1  4978
## 2     2  3724
## 3     3  4242
## 4     4  4411
## 5     5  5334
```

Interpretando la ecuación de regresión

Correlación entre los predictores

En el libro se muestra un ejemplo en código del efecto que puede tener correr un modelo con varias variables correlacionadas. Se explica con el tema de el 'precio de las casas'. Cuando tu estas prediciendo el precio de una casa, es lógico pensasr que entre más grande sea una casa, más cara sera. Para este caso, el modelo 'stepwise' que se corrio tiene la variables de 'cuartos', 'baños' y las de 'metros cuadrados'. Todas estas variables hacen referencia al tamaño de una casa, por lógica, ya que entre más 'x' más grande será la casa. Entonces, para este modelo stepwise el coeficiente de 'baños' dio negativo, yendo en contra de la lógica común, y es que el hecho de tener varias variables correlacionadas entre sí, te va impedir tener un modelo fácil de interpretar con la realidad.

Para esto, el autor saca de la ecuación 'n-1' variables correlacionadas y deja 1, que siguiendo la lógica de 'entre más grande más cara', el coeficiente tiene que dar positivo.

- Nota: el tema de Multicolinealidad es cuando hay muchos predictores correlacionados o, de plano, predictores repetidos (como por ejemplo todos los niveles de un one-hot-encoder, en lugar de n-1)

Variables 'confusas' (confounding variables)

El libro menciona el ejemplo de cuando se corre el modelo sin utilizar la variable de grupos de zipcode, en el cual, los coeficientes para 'metros cuadrados', 'baños y 'cuartos, dan negativo, algo fuera del sentido común. Pero, al momento de agregar los zipcodes, estas variables ahora pasan a ser positivas. El concepto de varibles confusas hace alusión a aquellas variables que se sabe por lógica o sentido común que tienen un efecto positivo en 'y', pero en tu modelo salen negativas. La pregunta sería, ¿qué diferencia hace esta variable de ubicación en el modelo para que ahora si las tomará positivas? Sin duda alguna la variable de ubicación influye mucho en el precio de una casa, pero, es que acaso los algoritmos también tienen sentido común?

Testing the Assumptions: Regression Diagnostics

Estas indicaciones no sirven para medir la precisión de una predicción, pero dan visión para ello.

Outliers

Se menciona del outlier de regresión: tu 'y-hat' es mucho menor o mayor que tu 'y-real'. Puedes detectar estos outliers mediante el 'residual estandarizado', el cual es = residual/error estándar de los residuales. Este residual estándar se interpreta como 'el número de errores estándar alejados de la línea de regresión'.

Definiendo los conceptos que participan para el residual estandarizado: - standarized residual: la desviación estándar de los puntos con respecto a la línea de regresión. O, número de errores estándar alejados de la línea de regresión. - Error estándar: se utiliza bajo una muestra para estimar a la población. Es la desviación estándar de una estadístico(media, mediana...).

```
house_98105 <- house[house$ZipCode == 98105,]
lm_98105    <-  lm(AdjSalePrice ~   SqFtTotLiving   +   SqFtLot +   Bathrooms   +   Bedrooms + BldgGrad

#obtenemos los residuales estándar:

#mediante fórmula precargada:
sresid  <-  rstandard(lm_98105)
idx <-  order(sresid)
sresid[idx[1]]
```

```
##     20431
## -4.326732
```

```
#a manopla:
# residual/error estándar de los residuales

#varianza
sum((house_98105$AdjSalePrice - mean(house_98105$AdjSalePrice))^2) / (nrow(house_98105)-1)
```

```
## [1] 153243517179
```

```
var(house_98105$AdjSalePrice)
```

```
## [1] 153243517179
```

```
#desviación estándar
sqrt(sum((house_98105$AdjSalePrice - mean(house_98105$AdjSalePrice))^2) / (nrow(house_98105)-1))
```

```
## [1] 391463.3
```

```
sd(house_98105$AdjSalePrice)
```

```
## [1] 391463.3
```