STATISTICAL MODELLING   Part IIC, Michaelmas 2021
Practical 3: Linear regression using R   By courtesy of Dr. S. A. Bacallado and Dr. R. Shah
Note: Below is the practical sheet used by previous year's lecturers. Some notations are different from what we used this year: $\tilde{\sigma}$ below is our $\hat{\sigma}$; P below is our hat matrix H.

Data.frame

We have already met lists as a very flexible data structure in R. A data frame is a special type of list where each variable has a name and has the same length. Thus it takes the form of a two-dimensional array, but it differs from a matrix in R as different variables (columns) can be of different types. The **read.csv** function creates data frames from **c**omma-**s**eparated **v**alue files. Download the students' brain size data from my website and store it in the data frame **BrainSize**

```
> file_path <- "https://raw.githubusercontent.com/AJCoca/SM19/master/"
> BrainSize <- read.csv(paste0(file_path, "BrainSize.csv"))
```

The **paste0** command joins strings together and is used here simply for convenience. We can view the first 5 rows of the data by issuing

```
> BrainSize[1:5, ]
```

and the full dataset can be viewed by simply typing in

```
> BrainSize
```

(not advisable for large datasets). The data contains various measures of IQ (Intelligence Quotient) and brain size measurements for 38 psychology students, along with their height, weight and gender. Note how the first column consists of the categories **Male** and **Female**, whereas the other columns are numbers. The **summary** function can be applied to a variety of different objects in R to produce helpful summaries. Try applying it to the the data frame **BrainSize**. We can get a correlation matrix of the numerical variables by performing

```
> cor(BrainSize[, -1])
```

the -1 serving to omit the first column.
Individual variables of a data frame can be accessed in the following way:

```
> BrainSize$PIQ
> BrainSize$Height
```

It can be a bit laborious prepending **BrainSize$** to every variable. The **attach** function adds the variables of given data frame to the R search path so that they can be accessed by simply giving their names. The opposite of **attach** is **detach**. Note it is not advisable to use attach when working with several datasets at once due to potential name conflicts.

```
> search()
 [1] ".GlobalEnv"        "tools:rstudio"     "package:stats"     "package:graphics"
 [5] "package:grDevices" "package:utils"     "package:datasets"  "package:methods"
 [9] "Autoloads"         "package:base"
> attach(BrainSize)
> search()
 [1] ".GlobalEnv"        "BrainSize"         "tools:rstudio"     "package:stats"
 [5] "package:graphics"  "package:grDevices" "package:utils"     "package:datasets"
 [9] "package:methods"   "Autoloads"         "package:base"
> Height
```

The `lm` function

So far, we have manually created projection matrices in R from given design matrices, and written out code to calculate t-statistics etc. As you might expect, R has ready-made functions to do all this for you and much more. The `lm` function is used to fit linear models.

```
> BrainSizeLM1 <- lm(PIQ ~ MRI_Count)
> BrainSizeLM2 <- lm(PIQ ~ MRI_Count + Height)
```

If `BrainSize` were not attached, it can be supplied as a data argument to the `lm` function,

```
> BrainSizeLM1 <- lm(PIQ ~ MRI_Count, data = BrainSize)
```

For both of the linear models fitted above, the response is `PIQ` (performance IQ). The first linear model uses as predictors `MRI_Count` and an intercept term, which is included by default. The second fits a model that also includes `Height` as a predictor. Typing `?formula` provides more information on the syntax.

Output from `lm`

Examining the output of `str(BrainSizeLM1)` shows that the function `lm` creates a list with many components. The easiest way to access the most important information is via the `summary` function. We have annotated the slightly abbreviated output below, using the same notation as in the lecture notes.

```
> summary(BrainSizeLM1)


Residuals:
    Min      1Q  Median      3Q     Max
-40.079 -17.508  -2.096  17.100  41.574
```

These are the main empirical quantiles of the residuals. The output continues as

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.660e+00  4.371e+01   0.107   0.9157
MRI_Count   1.177e-04  4.806e-05   2.448   0.0194 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

The columns of this table contain the following information, respectively: the estimates for the coefficients, i.e. the components of $\hat{\beta}$; the standard errors of the estimates, i.e., $\tilde{\sigma}\sqrt{\{(X^TX)^{-1}\}_{jj}}$; the values of the t-statistic $\hat{\beta}_j/(\tilde{\sigma}\sqrt{\{(X^TX)^{-1}\}_{jj}})$; the probability that a random variable $T \sim t_{n-p}$ is such that $|T|$ exceeds the absolute value of the t-statistic. These latter probabilities are thus p-values for testing the null hypotheses $\beta_j = 0$ against the alternatives $\beta_j \neq 0$ in the normal linear model. The last line above shows the symbols that are displayed on the right of the p-values to easily identify between what significance values each of them is. There is more output:

```
Residual standard error: 21.21 on 36 degrees of freedom
```

This is $\tilde{\sigma}$; the degrees of freedom is $n - p$ since $\tilde{\sigma}^2 \sim \frac{\sigma^2}{n-p}\chi^2_{n-p}$ in the normal linear model.

```
Multiple R-squared:  0.1427,    Adjusted R-squared:  0.1189
```

The R^2 and adjusted R^2 values. Lastly,

```
F-statistic: 5.993 on 1 and 36 DF,  p-value: 0.01937
```

This is the F-statistic

$$\frac{\frac{1}{p-1}\|PY - \bar{Y}1_n\|^2}{\frac{1}{n-p}\|(I-P)Y\|^2}$$

and the corresponding p-value for testing the null hypothesis of the intercept-only model against the alternative of the full model. If we set the significance level at 5%, we should reject the null hypothesis. Try `?summary.lm` for more details about `lm`. How would we print the correlations between the coefficient estimates?

Various other functions extract specific information from the `lm` output e.g. `coef`, `residuals`, `fitted.values`, `hatvalues` (leverage), `cooks.distance` etc. We can visualise our linear model fit with

```
> plot(PIQ ~ MRI_Count)
> abline(BrainSizeLM1)
```

Diagnostic plots

Applying the plot function to the output of `lm` gives four useful diagnostic plots (see `plot.lm` for further details, though the titles and axes labels are fairly self-explanatory). The `par` function is used to set plotting parameters. Here we set up the display so that four plots are produced on the same screen, saving the old parameters in `old_par`. We then reinstate the old parameters after the plot has been produced.

```
> old_par <- par(mfrow = c(2, 2))
> plot(BrainSizeLM1)
> par(old_par)
```

What should we expect these plots to look like if all the assumptions for the normal linear model held? One thing we can do is the following

```
> old_par <- par(mfrow = c(2, 2))
> plot(lm(rnorm(length(MRI_Count)) + fitted.values(BrainSizeLM1) ~ MRI_Count))
> par(old_par)
```

The plots produced will have almost exactly the same "distributions" as those from `BrainSizeLM1` if the normal linear model is correct—make sure you understand why. The only slight difference is that the scale on the y-axis of the first plot will be different. We can repeat the final plot above as many times as we like to get a better idea of what to expect when the assumptions hold.

In our case, it appears there is some trend in the residuals to suggest non-linearity; the Q–Q plot indicates that the tails of the studentised residuals (called the "standardized residuals" by R) are slightly lighter than we'd expect and the "scale–location" plot suggests a slight decrease in the error variance as the fitted values increase.

Let us see if adding a predictor to our model can alleviate the issue of non-linearity.

```
> plot(residuals(BrainSizeLM1) ~ Height)
```

There appears to be a negative relationship between height and the residuals, so we should try adding the variable `Height` to our model. Happily we have already done this in `BrainSizeLM2`. Examine the diagnostic plots for `BrainSizeLM2` to convince yourself that the non-linearity and heteroscedasticity don't appear to be as serious as before. The "Residuals vs Leverage" plot does however show that there is at least one observation with a rather high leverage score that we could consider omitting.

Interpretting the coefficients

The summary information of `BrainSizeLM2`,

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.113e+02  5.587e+01    1.992 0.054243 .
MRI_Count    2.061e-04  5.467e-05    3.769 0.000605 ***
Height      -2.730e+00  9.932e-01   -2.748 0.009403 **
---

Residual standard error: 19.51 on 35 degrees of freedom
Multiple R-squared:  0.2949,    Adjusted R-squared:  0.2546
F-statistic: 7.319 on 2 and 35 DF,  p-value: 0.00221
```

3

shows that `Height` has a negative coefficient: perhaps a slightly worrying fact for the taller ones amongst us! However, there is an important lesson to be learnt here. The way to interpret the negative coefficient is not that taller people have lower IQs, but that for a unit increase in height (in this case an inch), the IQ decreases 2.7 points on average *when keeping the MRI count (brain size) fixed*. Note the correlation matrix obtained from `cor(BrainSize[, -1])` shows a positive correlation between brain size and height, suggesting that taller people naturally have larger brains (perhaps as they are bigger overall). Also, note that though both height and brain size appear to be significant, the R^2 is rather small so there is still a large amount of unexplained variation in the data—we should not read too much into the results!

Cook's distance and leverage

To further understand the ideas behind Cook's distance and leverage, let us look at some artificial datasets.

```
> set.seed(1)  # sets the seed for the random number generator
> x <- c(rnorm(24), 4)
> y <- 1 + x + rnorm(25)
> plot(y ~ x)
> LinMod1 <- lm(y~x)
> abline(LinMod1)
> (lev1 <- hatvalues(LinMod1))
> which(lev1 > 3*2/25)  # do ?which to understand what it does
```

We see that the last observation has a high leverage score according to our rule of thumb threshold of $3p/n$. Compare the values of the F-statistics for the null hypothesis of the intercept-only model against the alternative of the full model, and the R^2 values, for `LinMod1` and a model that excludes the final observation (use the subset option of `lm`).
Examining

```
> pf(cooks.distance(LinMod1), 2, 23, lower.tail = FALSE)
```

shows that the Cook's distance of the observation is rather small. The value of α for which $F_{p,n-p}(\alpha)$ (the upper α point of an $F_{p,n-p}$ distribution) equals this Cook's distance is roughly 0.77: this is above 0.5, the rule of thumb threshold discussed in lectures. Omitting the last observation pushes the estimates for the coefficients almost to the edge of their 23% confidence ellipsoid—not necessarily a big cause for concern.
Now create a new response `y2` as follows:

```
> y2 <- y
> y2[25] <- y[25] - 5
```

Plot the data and add the least squares fit line. Further add a dashed line for the least squares fit without the final observation (setting the `lty` argument of `abline` to 2). Examine the Cook's distances of the observations as above.

Confidence and prediction intervals

Let us now briefly look at another dataset on house prices.

```
> detach(BrainSize)
> (HousePrices <- read.csv(paste0(file_path, "HousePrices.csv")))
```

This data gives the sale prices (in dollars) of various houses in New York along with various factors that are thought to be relevant for predicting sale price. To fit a model of `Sale.price` against all other variables in the data frame `HousePrices`, we can do the following.

```
> HousePricesLM <- lm(Sale.price ~ ., data = HousePrices)
```

Applying `confint` to `HousePricesLM` gives confidence intervals for the coefficients.
The `predict` function (try `?predict.lm`) can be used to give $x^{*T}\hat{\beta}$ for a new observation x^*. The observation x^* must be supplied as a data frame:

```
> newdata <- data.frame("Bedrooms" = 5, "Bathrooms" = 2, "Living.area" = 1400,
+ "Lot.size" = 7000, "Year.built" = 1950, "Property.tax"=9000)
> predict(HousePricesLM, newdata = newdata)
       1
326144.8
```

Using the `interval` option of the `predict` function allows us to get confidence intervals for $x^{*T}\beta$ and a prediction interval for $Y^* \sim N(x^{*T}\beta, \sigma^2)$. Note the former gives you an interval for the expected price of a house with properties x^*, a quantity perhaps useful for a tax assessor. If you want an interval for how much your own house will sell for, the prediction interval is more relevant.