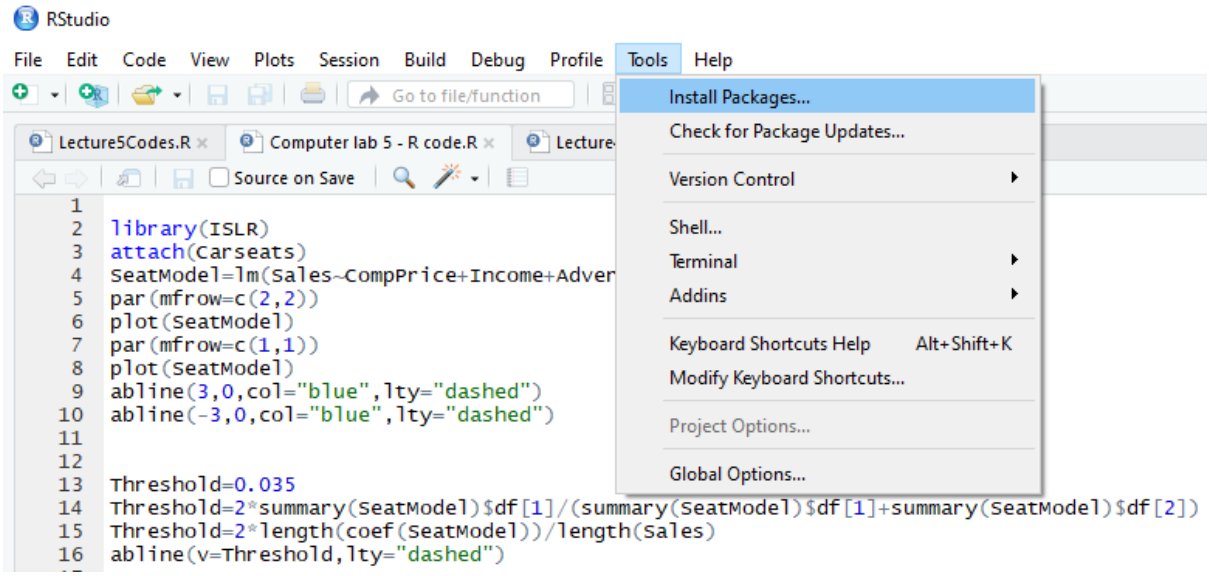
R code examples for computer lab 5 in MMS075, Feb 19, 2020

1. Re-do exercise 2 in exercise class 5a using a computer. That is, load the **Carseats** library and define a multiple linear regression model for predicting sales of car seats based on the variables **CompPrice**, **Income**, **Advertising**, **Price** and **ShelveLoc**, and plot the model. Once you have the Residuals vs Leverage plot on the screen, do the following steps to identify outliers and high leverage points:

   NOTE: Unfortunately, there is a mistake in the first line: Carseats is not a library but rather a dataset in the ISLR library! Therefore, the ISLR library needs to be loaded before proceeding:
   library(ISLR)

   If the above line returns an error message, it may be that you do not have ISLR installed. In that case, you can go to Tools > Install packages in RStudio (see figure below) to install it. Do not forget to type **library(ISLR)** again after the installation!



   After loading the ISLR library, we can attach the dataset Carseats – this will allow R to refer to its columns directly by column name, e.g. as Sales instead of Carseats$Sales.
   attach(Carseats)

   Once we have done this, we define the model:
   SeatModel=lm(Sales~CompPrice+Income+Advertising+Price+ShelveLoc)

   We now plot the model; the following command returns the four default plots for a regression model.
   plot(SeatModel)

   a. Draw a horizontal line, for example a blue dashed line, at values -3 and +3 for an easier identification of outliers;
      abline(3,0,col="blue",lty="dashed")
      abline(-3,0,col="blue",lty="dashed")

b. Compute the threshold of 2(p+1)/n recommended to identify high leverage points, and create a variable called **Threshold** containing this value. One way to specify the numerator and denominator in the threshold is to use the **$df** part of the summary, as follows: assuming that the model is named SeatModel, we have that **summary(SeatModel)$df[1]** equals p+1 and **summary(SeatModel)$df[1]+summary(SeatModel)$df[2]** equals n.

Threshold=2*summary(SeatModel)$df[1]/(summary(SeatModel)$df[1]+summary(SeatModel)$df[2])

The above command line relies on the fact that $df contains the degrees of freedom for the F-test for model significance, which are p+1 and n-(p+1); therefore, the numerator in the threshold 2(p+1)/n is 2*(the first degree of freedom) while the denominator is the sum of the first and second degrees of freedom.
An alternative way, utilizing that there are exactly p+1 coefficients of the model and therefore the length of the coefficient vector is p+1, and that all elements of Sales are included in the sample and therefore the length of Sales is exactly 400:
Threshold=2*length(coef(SeatModel))/length(Sales)

c. Draw a vertical line at the value computed in part b as follows:
abline(v=Threshold,lty="dashed")

Based on this plot, what can you conclude about outliers and high leverage points?
We conclude that there are two outliers and a couple of high leverage points, but none of the high leverage points is an outlier.

2. Consider the model defined Exercise 14 on page 125 of ISL, with the following commands:

```
set.seed(1)
x1=runif(100)
x2=0.5*x1+rnorm(100)/10
y=2+2*x1+0.3*x2+rnorm(100)
Model1=lm(y~x1+x2)
```

a. During the exercise class, we computed the value inflation factor (VIF) value for this model based on the correlation. This time, compute the VIF values for Model1 with the **vif** function that is included in a library called **car**!

```
library(car)
```

If the above line returns an error message, it may be that you do not have car installed. In that case, you can go to Tools > Install packages in RStudio (see figure in exercise 1) to install it. Do not forget to type **library(car)** again after the installation!

```
vif(Model1)
```

b. Repeat the same procedure with using another number in the first command line instead of 1 (e.g. use **set.seed(87)**). Has the result changed? Re-do it again with set.seed(1) and confirm that you get back the original results.

```
set.seed(87)
x1=runif(100)
x2=0.5*x1+rnorm(100)/10
y=2+2*x1+0.3*x2+rnorm(100)
Model1=lm(y~x1+x2)
vif(Model1)
```

We see that the results have changed somewhat. The reason is that after setting a different seed in the random number generator, the random numbers used when defining the variables were different, which has slightly changed the VIF value in the resulting model.

```
set.seed(1)
x1=runif(100)
x2=0.5*x1+rnorm(100)/10
y=2+2*x1+0.3*x2+rnorm(100)
Model1=lm(y~x1+x2)
vif(Model1)
```

We got back the original results. This shows that setting a given seed in the random number generator will always give the same random numbers for the same commands – this way, we can reproduce results that involve randomness. This can be useful when talking to managers/customers/colleagues!

c. Use the **vif** function on the model defined in exercise 1 and check the results. Then remove the predictor ShelveLoc from the model and use **vif** again. Why is there a difference in the format of the outputs?

```
vif(SeatModel)
SeatModel=update(SeatModel,~.-ShelveLoc)
vif(SeatModel)
```

The first command returns generalized VIF values, because while a categorical variable, ShelveLoc, is included in the model, the usual variance inflation factor values cannot be computed. However, after the second command line, updating the model in such a way that it uses all predictors except ShelveLoc, all predictors in the model are quantitative. Therefore, the third command line will return the usual VIF values for this updated model.

Note: instead of using the update function, we could also have redefined the model by writing out all predictors except ShelveLoc:

```
SeatModel=lm(Sales~CompPrice+Income+Advertising+Price)
```

3. This exercise will do the optimization of the budget assignment in the advertising example using a computer. Download Advertising.csv from http://faculty.marshall.usc.edu/gareth-james/ISL/data.html and save it on the Desktop (i.e. in a computer-specific folder). Import the by using the menu in RStudio choosing File > Import Dataset > From Text (base)… Rename it as **AdvertisingData** while importing it to avoid a clash with the variable called Advertising in exercise 1. Attach the dataset for easier reference of its columns:

attach(AdvertisingData)

We define a model with sales as response and TV, radio and their interaction as predictors:
AdModelInt=lm(sales~TV*radio)

Assume that there is a fixed total budget of $100 000 and we want to find the optimal way to divide this amount between TV and radio advertisements. This can be done by simply making predictions for all possible divisions, considering all 100 001 possibilities ranging from $0 to $100 000 on TV, as follows:

a. Define a variable called **TVBudget** containing a sequence of numbers from 0 to 100000 using either the **seq** or the **seq.int** function with appropriate arguments.
TVBudget=seq(from=0, to=100000)

Note: it is a slightly unpleasant feature of this sequence that the 1st element of the sequence, i.e. TVBudget[1], corresponds to a TV budget of $0, and generally, the i-th element of the sequence, i.e. TVBudget[i] corresponds to a TV budget of i-1 dollars.

b. Since the variables '**TV**' and '**radio**' in the linear regression model are defined in $1000, divide each element of TVBudget by 1000 using the following command:
TVBudget=TVBudget/1000

c. Make a prediction of sales for each value of TVBudget using the pred function:
Predictions=predict(AdModelInt,data.frame(TV=TVBudget, radio=100-TVBudget))

d. Check where Predictions has its maximum value and what the corresponding prediction is:
which.max(Predictions)
max(Predictions)

Note: the first command line returns the index within the Predictions vector that has the maximum value. This index corresponds to the same index within the TVBudget sequence; as noted in part a), the corresponding budget in $ is the index minus 1. The second command line returns the maximum value within the Predictions vector (which we could also get by writing **Predictions[which.max(Predictions)]**). This gives the estimated value for the sales variable which is measured in 1000 sold units; therefore, we will need to multiply the result from the second command line by 1000 when answering the next question.

Describe the optimal division of the total budget and the maximum estimate for sales in words and in terms of $, respectively sold units instead of variable values!
The above command lines give the following outputs:

```
> which.max(Predictions)
45510
45510
> max(Predictions)
[1] 11.88644
```

Therefore, according to the note at the beginning of part d, a TV budget of $45 509 gives the highest estimate for the number of sold units, namely 11 886 units.

Now repeat and appropriately modify the above steps to find the optimal budget distribution if the available total budget is $50 000 and also for a total budget of $200 000.

```
> TVBudget=seq(from=0,to=50000)
> TVBudget=TVBudget/1000
> Predictions=predict(AdModelInt,data.frame(TV=TVBudget, radio=50-TVBudget))
> which.max(Predictions)
20510
20510
> max(Predictions)
[1] 8.65023
> TVBudget=seq(from=0,to=200000)
> TVBudget=TVBudget/1000
> Predictions=predict(AdModelInt,data.frame(TV=TVBudget, radio=200-TVBudget))
> which.max(Predictions)
95510
95510
> max(Predictions)
[1] 22.43322
```

4. Consider the logistic regression model predicting the probability of credit card default based on balance, as discussed in the lecture. The data that this model is based on can be found in dataset '**Default**' the **ISLR** library, and you may want to attach the dataset before proceeding to the further steps.

library(ISLR)
attach(Default)

Explore the dependence of default probability on balance via the following tasks:

a. Understand the **Default** dataset better by checking its summary. How many people are there in the dataset in total? How many have been defaulted? What is the percentage of students in the data?

```
> summary(Default)
 default    student       balance           income
 No :9667   No :7056   Min.   :   0.0   Min.   :  772
 Yes: 333   Yes:2944   1st Qu.: 481.7   1st Qu.:21340
                       Median : 823.6   Median :34553
                       Mean   : 835.4   Mean   :33517
                       3rd Qu.:1166.3   3rd Qu.:43808
                       Max.   :2654.3   Max.   :73554
```

The total number of people in the dataset can be determined by summing the No and Yes values for default: 9667+333 = 10000. Of these, 333 have been defaulted. The percentage of students in the data is 29.44%.

b. Define the model in R using the following command:
glm(default~balance,family="binomial",data=Default)
In this command, glm stands for generalized linear model and family=binomial is needed to specify that we want to use logistic regression.

c. Display a summary of the model to see the coefficient estimates.
One way to do this (which will be useful for part d as well) is to give a name to this model and then ask for a summary:
BalanceModel=glm(default~balance,family="binomial",data=Default)
summary(BalanceModel)

Another way would be to write the following command, asking for the summary directly at the time of model definition:
summary(glm(default~balance,family="binomial",data=Default))
This is a good way if all we want with this model is to display a summary. However, if we plan further steps with the model, then it is better to give it a name, i.e. define a variable containing this model which can then be used in future steps, see part d.

d. Make a prediction for the following balance values: $1000, $1500, $2000, $2500. This can be done by using the **predict** function, including the usual arguments (i.e. model name, data frame to perform the prediction for) plus an additional argument writing **type="response"**. For your own understanding, check what you get as result if you do not include the **type="response"** argument!
predict(BalanceModel,data.frame(balance=1000),type="response")
predict(BalanceModel,data.frame(balance=1500),type="response")
predict(BalanceModel,data.frame(balance=2000),type="response")
predict(BalanceModel,data.frame(balance=2500),type="response")

If we do not use **type="response"** in the above commands, we get back the expression that is on the right side of the logistic regression equation, for the given value of the predictor. In other words, in that case, we get the log-odds of default at the given balance value instead of the probability of default.

e. Plot a function with balance on the x-axis and the predicted probability of default on the y-axis. This can be done using the curve command, as follows:
curve(predict(BalanceModel,data.frame(balance=x),type="response"), from = 500, to = 3000,xlab="Balance ($)",ylab="Estimated probability of default",cex.lab=1.5,cex.axis=1.2)

In this command, the first argument of curve is an expression that depends on x, and the from and to arguments specify the range of x-values that will be considered in the construction of the curve. The arguments **xlab** and **ylab** specify the axis labels and **cex.lab** sets the size of these labels. Finally, **cex.axis** sets font size for the numbers on the x- and the y-axis.

5. Let us now try predicting the probability of credit card default based on student status. Use the **glm** command as in exercise 5 to fit a model and display its summary.
StudentModel=glm(default~student,family="binomial",data=Default)
summary(StudentModel)

a. What are the coefficients of the model? Does being a student increase or decrease the probability of being defaulted?
The coefficients can be seen in the R output below:

```
Coefficients:
             Estimate
(Intercept) -3.50413
studentYes   0.40489
```

We see that the dummy variable representing student status "Yes" has positive coefficient estimate. This means that being a student increases the probability of being defaulted.

b. Compute the estimated probability of defaulting for students and also for non-students!

```
> predict(StudentModel,data.frame(student="Yes"),type="response")
         1
0.04313859
> predict(StudentModel,data.frame(student="No"),type="response")
         1
0.02919501
```

This output also confirms the conclusion in part a), i.e. that the probability of defaulting is higher for students than it is for non-students.

c. Can you plot a curve similar to the one prepared in part e of exercise 5?
No, that is not possible here. In that case, balance was a numerical variable that took values along a numerical scale which could then be considered on the x-axis. Student is a binary categorical variable, hence there are only 2 values where the probability of being defaulted could be displayed.

6. Finally, include all variables in the model and answer the following questions:
   a. Which variables are significant in this model and which are not?
   AllModel=glm(default~student+income+balance,family="binomial",data=Default)
   summary(AllModel)

   This gives the following output (only the relevant part is copied):

```
Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -1.087e+01  4.923e-01 -22.080  < 2e-16 ***
studentYes  -6.468e-01  2.363e-01  -2.738  0.00619 **
income       3.033e-06  8.203e-06   0.370  0.71152
balance      5.737e-03  2.319e-04  24.738  < 2e-16 ***
```

   Looking at the last column containing the p-values (or the stars next to that column), we conclude that student and balance are significant variables while income is not significant.

   b. What are the coefficients of the model? Does being a student increase or decrease the probability of being defaulted? Is there any difference compared to part 5a?
   You can look at the summary shown in part a) to see the coefficients. Remember, if the scientific notation of numbers disturbs you, it is possible to change it to the more usual way of numbers by options(scipen=999) and display the summary again:
   options(scipen=999)
   summary(AllModel)

   After these commands, the summary looks like this:

```
Coefficients:
                 Estimate    Std. Error z value             Pr(>|z|)
(Intercept) -10.869045196   0.492255516 -22.080 < 0.0000000000000002 ***
studentYes   -0.646775807   0.236252529  -2.738              0.00619 **
income        0.000003033   0.000008203   0.370              0.71152
balance       0.005736505   0.000231895  24.738 < 0.0000000000000002 ***
```

We can see the coefficients in the Estimate column. In particular, the coefficient of student is negative here, indicating that when income and balance are in the model, then being a student *decreases* the probability of defaulting. In other words, as we learned from 5a, if we only know that somebody is a student, that increases the probability of defaulting; however, if we first fix a balance and income level and look only at people having this fixed level of balance and income, then among those people, students have a lower probability of being defaulted. We will discuss this again in Lecture 6a.

c. Compute the estimated probability of defaulting for a non-student with credit card balance of $1000 and annual income of $30 000!
predict(AllModel,data.frame(balance=1000, income=30000, student="No"),type="response")
The estimated probability of defaulting in this case is 0.0064, i.e. 0.64%.

d. Estimate the probability of defaulting for a student with the same credit card balance and annual income values as in part c!
predict(AllModel,data.frame(balance=1000, income=30000, student="Yes"),type="response")
The estimated probability of defaulting for a student with the same credit card balance and annual income values as in part c is 0.0034, i.e. 0.34%. This is indeed lower than the estimate in part c).

7. If you would like to ask something or give feedback, feel free to talk to me or enter your question/feedback at www.menti.com, using the code 71 36 17.