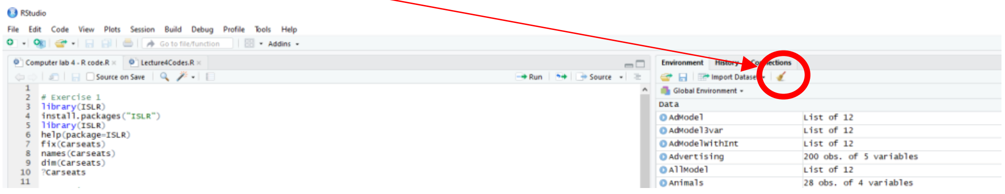
Computer lab 4 in MMS075, Feb 12, 2020

Solutions in terms of an R code example and detailed explanations

1. We can start the lab by removing all variables that we could possibly find from earlier sessions to ensure that we can start from scratch and previously defined variables do not interfere with new code that we may want to enter. We can do that in two ways; either using a command:
   rm(list=ls())
   or by clicking the tiny broom icon in RStudio (see picture below).



```
# Exercise 1
library(ISLR)
install.packages("ISLR")
library(ISLR)
help(package=ISLR)
fix(Carseats)
names(Carseats)
dim(Carseats)
?Carseats
```

```
Data
AdModel          List of 12
AdModel3var      List of 12
AdModelWithInt   List of 12
Advertising      200 obs. of 5 variables
AllModel         List of 12
Animals          28 obs. of 4 variables
```

   We can do this later at any time when we feel that we want to start from scratch.

2. As usual, download the advertising example in ISL 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)…

   This time, we will not change the name of the dataset while importing it, which means that it will be called Advertising.

3. We can ask for a summary of the data which gives us some basic measures for each variable:
   summary(Advertising)
   Which variable has the largest mean and median values? Which value has the widest range of values?
   TV has the larges mean and median (which are directly given in the output) as well as the widest range of values. The range is the difference between the maximum and minimum values; this can be computed using the summary output or using the following command:
   max(Advertising$TV)-min(Advertising$TV)

4. When interpreting the summary, it may have been distracting that it included the index variable X. Therefore, we first remove that variable, ask for a summary again and also ask for a visualization of the data using "pairs":
   Advertising=Advertising[,-1]
   summary(Advertising)
   pairs(Advertising)

   The command "pairs" has created scatter plots for each pair of variables in the dataset. Do all predictors seem to be linearly related to the response "sales"? Do you see any signs of nonlinear relationship?
   The bottom row of the plot shows the scatter plots with each predictor on the x-axis and the response, sales, on the y-axis. The plot sales vs TV looks a bit curvy for small values of TV.

Note that the plots below and above the diagonal are reflections of each other across the line y=x, because they show the same set of points, but with the x and y axes switched. Note also that the scatter plots between two predictor variables can also be useful; for example, if there is strong correlation between two predictors, we may spot that by looking at the scatter plots.

5. During the lecture, we have seen that interaction terms are relevant for predicting sales, and we now learn how to include interaction terms in linear regression in R. We first attach the dataset so the we can avoid writing "Advertising$" when referring to variables in this data frame:

   ```
   attach(Advertising)
   ```

   We then look at the model with TV and radio as predictors and specify that we want to include their interaction term as well:

   ```
   AdModelInt=lm(sales~TV+radio+TV:radio)
   ```

   Note: I named this model as AdModelInt because it models the effect of ads and includes an interaction term. You should feel free to choose another name that you like more – all you need to do is to change AdModelInt in the command line above with the name that you choose. You just need to remember to use that name in all subsequent command lines when you refer to this model.

   We can look at the summary of the model and see that it indeed includes the interaction term:

   ```
   summary(AdModelInt)
   ```

   Let us now try another command to define a model and look at the summary:

   ```
   AdModelInt2=lm(sales~TV*radio)
   summary(AdModelInt2)
   ```

   Check that this output is exactly the same as the one above. It should be, because TV*radio is a shorthand for "include TV, radio and their interaction term", i.e. it does exactly the same thing as writing TV+radio+TV:radio.

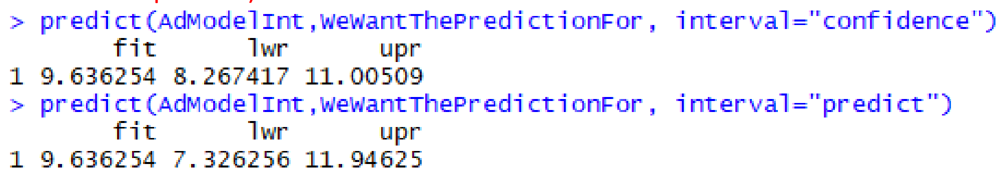
   Now, however, we do not need two identical models, so we remove the one that we defined later:

   ```
   rm(AdModelInt2)
   ```

6. Revisit exercise 2 from yesterday, with the addition of quantifying uncertainty: using AdModelInt, predict the number of sold units and specify confidence and prediction intervals for the following advertisement budget distributions:
   a) TV budget: $0, radio budget: $100 000;
      We will need to use the "predict" function for this on the model AdModelInt and we need to put the values that we want to get the predictions for into a data frame. Let's do this first. We usually create a data frame called newdata, but that's just a name; I will now use a different name when creating the data frame:

      ```
      WeWantThePredictionFor = data.frame(TV=0,radio=100)
      ```

Having the data frame available, we can use the predict function, with interval=”confidence” for confidence interval (which gives an interval for the average sales value with this advertisement budget, i.e. if the management decides to use this budget for many products, this is what they can expect for their average sales) and interval=”predict” for prediction interval (which gives an interval for a specific case, e.g. if the management now decides to use this advertisement for a specific product, then this interval gives an idea of what they can expect about the sales of that product).



```
> predict(AdModelInt,WeWantThePredictionFor, interval="confidence")
       fit      lwr      upr
1 9.636254 8.267417 11.00509
> predict(AdModelInt,WeWantThePredictionFor, interval="predict")
       fit      lwr      upr
1 9.636254 7.326256 11.94625
```

These outputs give the same estimate on the value of 9.636 on the sales variable, with a 95% confidence interval of 8.267 ≤ sales ≤ 11.005 and a wider prediction interval of 7.326 ≤ sales ≤ 11.946. Remembering that the variable sales corresponds to 1000 sold units, we predict 9636 sold units with this advertisement budget, with a confidence interval of 8267 ≤ sold units ≤ 11005 and a prediction interval of 7326 ≤ sold units ≤ 11946.

Note: instead of first creating a data frame variable and using that variable within the predict function, we could also have specified within predict the data frame that we want the predictions for; using the two lines below gives the same outputs:
predict(AdModelInt,data.frame(TV=0,radio=100), interval="confidence")
predict(AdModelInt,data.frame(TV=0,radio=100), interval="predict")

Note2: The estimated numbers of sold units here and in the other parts of this exercise differ slightly from the answers provided in “Exercise class 4 - exercises with solutions.pdf”. What is the reason for this difference? Think about this and if you are not sure, ask me!

b) TV budget: $100 000, radio budget: $0;
predict(AdModelInt,data.frame(TV=100,radio=0), interval="confidence")
predict(AdModelInt,data.frame(TV=100,radio=0), interval="predict")
Based on the output from these commands, we predict 8660 sold units with this advertisement budget, with a confidence interval of 8385 ≤ sold units ≤ 8936 and a prediction interval of 6779 ≤ sold units ≤ 10514.

c) TV budget: $50 000, radio budget: $50 000;
predict(AdModelInt,data.frame(TV=50,radio=50), interval="confidence")
predict(AdModelInt,data.frame(TV=50,radio=50), interval="predict")
Based on the output from these commands, we predict 11865 sold units with this advertisement budget, with a confidence interval of 11454 ≤ sold units ≤ 12275 and a prediction interval of 9959 ≤ sold units ≤ 13770.

d) TV budget: $50 000, radio budget: $50 000, newspaper budget: $30 000.

We can either directly note that newspaper budget is not used in the model and therefore there won’t be any difference compared to part c), or we can include a newspaper term in the defined data frame that won’t be used by R anyway:
predict(AdModelInt,data.frame(TV=50,radio=50,newspaper=30), interval="confidence")
predict(AdModelInt,data.frame(TV=50,radio=50,newspaper=30), interval="predict")
gives the exact same output as the commands in part c).

Furthermore, estimate the effect of:

e) $1000 increase of TV advertisement on sold units if the radio budget is $10 000;
We will make two separate predictions, both with radio=10 and changing the value of the TV variable by 1 between the predictions. Taking their difference will quantify the effect of increasing TV advertisement budget for the given radio budget.
FirstPredictorValues=data.frame(TV=0,radio=10)
SecondPredictorValues=data.frame(TV=1,radio=10)
predict(AdModelInt, SecondPredictorValues)-predict(AdModelInt, FirstPredictorValues)
This gives a difference of 0.03 in the prediction of the sales variable; therefore, the effect of $1000 increase of TV advertisement on sold units at a radio budget of $10 000 is an increase of 30 units.

f) $1000 increase of TV advertisement on sold units if the radio budget is $100 000;
FirstPredictorValues=data.frame(TV=0,radio=100)
SecondPredictorValues=data.frame(TV=1,radio=100)
predict(AdModelInt, SecondPredictorValues)-predict(AdModelInt, FirstPredictorValues)
This gives a difference of 0.128 in the prediction of the sales variable; therefore, the effect of $1000 increase of TV advertisement on sold units at a radio budget of $10 000 is an increase of 128 units.

g) $1000 increase of radio advertisement on sold units if the TV budget is $50 000.
FirstPredictorValues=data.frame(TV=50,radio=0)
SecondPredictorValues=data.frame(TV=50,radio=1)
predict(AdModelInt, SecondPredictorValues)-predict(AdModelInt, FirstPredictorValues)
This gives a difference of 0.083 in the prediction of the sales variable; therefore, the effect of $1000 increase of radio advertisement on sold units at a TV budget of $50 000 is an increase of 83 units.

Finally, predict the number of sold units for:
TV budget: $50 000, radio budget: $51 000.
predict(AdModelInt,data.frame(TV=50,radio=51))
gives 11.948 as a prediction of the sales variable, which corresponds to a prediction of 11948 sold units. We could also get this by summing the prediction of 11865 for TV budget: $50 000, radio budget: $50 000 computed in part c) and the increase of 83 units as the estimated effect of $1000 increase in radio advertisement on sold units at a TV budget of $50 000 computed in part g).

7. We now want to look at potential problems with this model. As the starting point, we can use the default graphs provided by R for linear models:

   ```
   plot(AdModelInt)
   ```

   This command will show you 4 different plots and you need to press Enter to switch between them. If you want to see them all together, you can divide the plotting screen as we did last time, using the “**par(mfrow=c(2,2))**” command before plotting the model.

   ```
   par(mfrow=c(2,2))
   plot(AdModelInt)
   ```

   The first plot shown is the residual plot, having the predicted values on the x-axis and the residuals on the y-axis. The last one showing standardized residuals vs leverage is used for identifying high leverage points.

   Now, before going further with plotting, it makes sense to switch back to the usual plotting screen so that we get one larger plot each time instead of four smaller plots. Previously, with par(mfrow=c(2,2)), we asked R to divide the plotting screen in two vertical and two horizontal parts. If we want to have a single plot, we need one vertical part and one horizontal part. Therefore, to switch back, we can type:

   ```
   par(mfrow=c(1,1))
   ```

   From now on, any new plots will be shown as a single, big plot, as we will soon see below.

   For finding outliers, we prefer to look at studentized residuals that are very similar to standardized residuals but are not the same thing. Therefore, we produce a studentized residual plot as follows:

   ```
   plot(predict(AdModelInt),rstudent(AdModelInt), xlab=”Predicted sales (1000
   units)”,ylab=”Studentized residuals”,cex=1.5,pch=20)
   ```

   Note: in the studentized residual plot, we have the predicted values for all observations in the data on the x-axis, and the studentized residuals for all observations in the data on the y-axis. This is why we can just use the model name in predict: without specifying the data frame that it should give predictions for, it will give the predicted values for all observations that the model was created with. Similarly, rstudent with only as the name of the model as argument will return the studentized residuals for all observations that the model was created with. Consequently, predict(AdModelInt) and rstudent(AdModelInt) return vectors of the same size which can then be plotted against each other. The other arguments of the function just make the plot look better: xlab and ylab specify the axis labels, cex determines the size of plotted points (size 1 is the default, and cex=1.5 results in 50% bigger points) and pch determines the point type to be plotted – it can take values 1, 2, …, 20, and you should experiment with this to find the point type that you like most.

   If we include lines at y-values of -3 and 3, that will make it even easier to spot outliers:

   ```
   abline(-3,0,lty="dashed",col="red")
   abline(3,0,lty="dashed",col="red")
   ```

Note: this makes sense, because points with studentized residuals below -3 or above +3 indicate large deviation from the prediction, which is what being an outlier means. Why exactly these values? The studentized residuals result from dividing the residuals by their estimated standard error, so they should have approximately a t distribution. If the degree of freedom is sufficiently large, then the t distribution is very close to standard normal. For standard normal distribution, 99% of values are between -3 and +3, hence those that are outside this interval indeed indicate large deviations.

Note2: The argument "lty" means line type, and "col" means color.

We may also want to produce our own residual plot and leverage plot as well:

```
plot(predict(AdModelInt),residuals(AdModelInt), xlab="Predicted sales (1000 units)",ylab="Residuals",cex=1.5,pch=20)
plot(hatvalues(AdModelInt),rstudent(AdModelInt), xlab="Leverage",ylab="Studentized residuals",cex=1.5,pch=20)
```

Note: the creation of the residual plot is analogous to the creation of the studentized residual plot described above, except that "residuals" is used for the y-axis instead of "rstudent".

For the leverage plot, the x-axis contains the values of the leverage statistic for each observation, i.e. the measure of how unusual predictor values belong to this point. These can be accessed by the "hatvalues" function (and the corresponding formula for simple linear regression is equation (3.37) in ISL).

It is nice to set a threshold for how large the leverage statistic should be to justify calling an observation a high leverage point. In Bruce, P. & Bruce, A. (2017), Practical Statistics for Data Scientists, points with leverage above 2(p+1)/n are called high leverage points. In this case, p=3, because we have TV, radio and their interaction term as predictors, and n=200, because there were 200 observations in Advertising, hence the threshold is 0.04. We can draw a vertical line to indicate this:

```
abline(v=2*(3+1)/200,lty="dashed",col="blue")
```

Furthermore, if we want to spot high leverage points and outliers in the same graph, we can note that we have studentized residuals on the y-axis, so points with y-axis values below -3 or above +3 are outliers. We can include two horizontal lines to indicate this:

```
abline(-3,0,lty="dashed",col="red")
abline(3,0,lty="dashed",col="red")
```

This plot shows that in this case, there are two outliers of which one is also a high leverage point.

Give titles to these figures by using the title("…") command!
This command always makes a title for the plot that was created last. Therefore, if we want to set a title for both of these figures, we need to re-create the first one, set a title, re-create the second one and set the title for that one. To make sure that we see both of them, we will split the plotting screen in two horizontal parts; however, we set it back to the usual one-part structure at the end to make it better for future plots.

```
par(mfrow=c(2,1))
plot(predict(AdModelInt),residuals(AdModelInt), xlab="Predicted sales (1000
units)",ylab="Residuals",cex=1.5,pch=20)
title("Residual plot")
plot(hatvalues(AdModelInt),rstudent(AdModelInt), xlab="Leverage",ylab="Studentized
residuals",cex=1.5,pch=20)
abline(v=2*(3+1)/200,lty="dashed",col="blue")
abline(-3,0,lty="dashed",col="red")
abline(3,0,lty="dashed",col="red")
title("Studentized residuals vs Leverage")
par(mfrow=c(1,1))
```

8. Revisit Exercise 14 on page 125 of ISL that was considered at the end of the exercise class yesterday. If you produce the required plots and summaries yourself instead of using the ones provided by me, then you will get a much better understanding of the relevant concepts. Additionally, you can experiment with other options and check whatever you find interesting about that model.
   We start with the set.seed command that allows us to get the same random numbers each time we run the code below (this way we can get reproducible results that we can discuss, otherwise the results would differ slightly each time and maybe we would not see the intended interesting behavior).
   set.seed(1)

   We then generate 100 random numbers between 0 and 1 and include them in a variable called x1:
   x1=runif(100)

   We then generate another predictor called x2 which is takes half of the value of x1, plus a random normal error term divided by 10:
   x2=0.5*x1+rnorm(100)/10

   Finally, we make a response variable y as a linear combination of x1 and x2 plus a random normal error term:
   y=2+2*x1+0.3*x2+rnorm(100)

   (a): What we did here is that we defined a linear model where we KNOW the true coefficients, because we created them:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon,$$

   where $\beta_0 = 2$, $\beta_1 = 2$, $\beta_2 = 0.3$ and ε has normal distribution with mean=0 and variance $\sigma^2 = 1$.
   We will see in the other parts how well the true coefficients are approximated by the estimated coefficients.

   (b): The correlation can be checked using the “cor” command, and we can also plot them against each other:
   cor(x1,x2)
   plot(x1,x2)

These variables are strongly correlated with each other, which is not surprising considering that, except for some small random error, x2 is constant times x1.

(c) Fitting a least squares regression to predict y using x1 and x2 can be done with the lm command, and we can check the summary of the model:
summary(lm(y~x1+x2))

This gives the estimated coefficients, so we can learn the following:
$\hat{\beta}_0 = 2.13$, $\hat{\beta}_1 = 1.43$, $\hat{\beta}_2 = 1.01$ which are not too close to the true coefficients $\beta_0 = 2$, $\beta_1 = 2$, $\beta_2 = 0.3$, except for the intercept. The residual standard error of RSE $= 1.06$ is reasonably close to the standard deviation $\sigma = 1$ of the error term.

We can reject the null hypothesis
$H_0 : \beta_1 = 0$
because the p-value for the t-test of the significance of x1 is 0.0487 < 0.05. This means that, even in the presence of x2 in the model, x1 is a significant predictor of the response y.

We cannot reject the null hypothesis
$H_0 : \beta_2 = 0$
because the p-value for the t-test of the significance of x2 is 0.3752 > 0.05. This means that in the presence of x1 in the model, x2 is not a significant predictor of the response y.

(d) Fitting a least squares regression to predict y using only x1 can be done with the lm command, and we can check the summary of the model:
summary(lm(y~x1))
In this case, the estimated coefficient of 1.97 for x1 is reasonably close to the true coefficient 2. We can reject the null hypothesis
$H_0 : \beta_1 = 0$
because the p-value for the t-test of the significance of x1 is 0.00000266, which is very small compared to the threshold of 0.05. This means that if we do not include other variables in the model, then x1 is a highly significant predictor of the response y.

(e) Fitting a least squares regression to predict y using only x2 can be done with the lm command, and we can check the summary of the model:
summary(lm(y~x2))
In this case, the estimated coefficient of 2.9 for x2 is way higher than its true coefficient 0.3. We can reject the null hypothesis
$H_0 : \beta_2 = 0$
because the p-value for the t-test of the significance of x2 is 0.0000137, which is very small compared to the threshold of 0.05. This means that if we do not include other variables in the model, then x2 is a highly significant predictor of the response y.

(f) The results do not contradict each other, because different questions are addressed in the different parts. In (d) and (e) we see that the variables alone help predicting the response. However, the much higher p-values in (c) indicate that once we have one of x1 or x2 in the model, the addition of the other one helps substantially less to predict y. This is not so surprising given that the two variables are strongly correlated, so by knowing the value of one variable, we already know a lot about the value of the other variable, and adding the other variable to the model would not provide so much extra information that could be used to predict y.

(g) We add the mis-measured observation as instructed:
x1=c(x1,0.1)
x2=c(x2,0.8)
y=c(y,6)

Note that this code means: redefine the variable x1 to be the concatenation of x1 and a new value, 0.1. Then redefine x2 as the concatenation of x2 and 0.8. Finally, redefine y as the concatenation of y and a new value 6. In other words, we have extended the length of each variable by 1 and from a model perspective, we have added an observation to the ones so far that has an x1 value of 0.1 and an x2 value of 0.8, and a response value of 6.

After adding the new observations, we can check how the models have changed by again fitting the models and looking at their summaries one by one. Let's start with the two-predictor model:
TwoModel=lm(y~x1+x2)
summary(TwoModel)

Checking the output shows that adding the extra observation has substantially changed the results. The new estimated coefficient of x2 is much higher than its coefficient in the original model and x2 has become significant (at p-value=0.00614). Conversely, the new estimated coefficient of x1 is much smaller than the original one and in the new model, x1 is not significant anymore in the presence of x2 (p-value=0.3646). There are no big changes in the overall significance of the model.

Now we check the one-predictor model with x1:
x1Model=lm(y~x1)
summary(x1Model)

We see that the coefficient of x1 has slightly decreased, the R^2 value has decreased from 0.21 to 0.16, but x1 is still a highly significant predictor of y when it is considered alone.

We also check the one-predictor model with x2:
x2Model=lm(y~x2)
summary(x2Model)

Compared to the original model, the coefficient of x2 has slightly increased, the R^2 value has increased from 0.18 to 0.21 and x2 is a highly significant predictor of y when it is considered alone.

To address the question of whether the new point is an outlier and/or a high leverage point in each model, we could construct the studentized residuals vs leverage graphs like we did in Exercise 7 for each model and we could see whether there are any outliers or high leverage points. Instead, we just check the studentized residual value and the leverage statistic for the newly added point which is observation 101 in the rstudent and hatvalues vectors:

```
rstudent(TwoModel)[101]
hatvalues(TwoModel)[101]
rstudent(x1Model)[101]
hatvalues(x1Model)[101]
rstudent(x2Model)[101]
hatvalues(x2Model)[101]
```

This gives that the new observation is an outlier in x1Model (with a studentized residual of 3.44), but not in the other two models.

The threshold for being a high leverage point in TwoModel is 2*(2+1) / 101 = 0.059 and for x1Model and x2Model it is 2*(1+1) / 101 = 0.04. This means that the new observation is a massive high leverage point in TwoModel (with leverage statistic of 0.41), not a high leverage value in x1Model (with leverage statistic of 0.033) and a high leverage value in x2Model (with leverage statistic of 0.1).

To understand why this is so, we can plot x2 against x1 and color the new point red:

```
plot(x1,x2,cex=2,pch=20)
points(x1[101],x2[101],col="red",cex=2,pch=20)
```

This plot shows that the new point does not have an unusual x1 value, has a somewhat unusual x2 value and a very unusual combination of (x1,x2) values.

9. We check polynomial regression again, this time on a different dataset called Wage that is included in the ISLR library.

   ```
   library(ISLR)
   attach(Wage)
   ```

   For a better understanding of the underlying data, we ask for the description of this dataset:

   ```
   ?Wage
   ```

   We want to investigate the dependence of wage on age, so we first plot these variables:

   ```
   plot(age,wage)
   ```

   It looks like wage does not linearly depend on age. We can try to get a better fit with polynomials. We could try to include different powers of age as predictors; the following examples, taken from Section 7.8.1 of ISL, consider polynomials of degree 4:

   ```
   fit=lm(wage~poly(age,4),data=Wage)
   ```

   This command created a regression model using a somewhat complex method (called orthogonal polynomial regression) to fit a polynomial of age of degree 4 on the observations. This is a good method and, in fact, has certain statistical and numerical advantages compared to the polynomial regression method as presented in the lecture; see, for example, this question in a statistical forum. However, the results are difficult to interpret. We will see below that the results from this method and the polynomial regression that uses age, (age)^2, (age)^3 and (age)^4 as predictors, called raw polynomial regression, are essentially identical in this example. Generally, unless you often need to use polynomial regression and need a detailed understanding of the matter, I recommend that you use raw polynomial regression (which will be presented below) to ensure a proper understanding of the model and the results.

```
coef(summary(fit))
fit2=lm(wage~poly(age,4,raw=T),data=Wage)
```

Including the argument “raw=T” which could also be written as “raw=TRUE” instructs R to fit a raw polynomial regression model using simply age, $(age)^2$, $(age)^3$ and $(age)^4$ as predictors rather than fitting an orthogonal polynomial regression that one would get without using this argument or by writing “raw=FALSE”.

```
coef(summary(fit2))
fit2a=lm(wage~age+I(age^2)+I(age^3)+I(age^4),data=Wage)
coef(fit2a)
```

Defining the model fit2a where age, $(age)^2$, $(age)^3$ and $(age)^4$ are explicitly specified as predictors and comparing the coefficients with those of fit2 can help to convince us that fit2 is indeed exactly the same model as fit2a and is just a more convenient way of writing the same thing.

Compare the results from the different outputs! The difference comes from the different ways of defining the predictors. The models fit2 and fit2a really use age, age^2, age^3 and age^4 as predictors while fit defines a degree 4 polynomial of age in a slightly different way.

Make an even more detailed comparison by looking at the full summaries:

```
summary(fit)
summary(fit2)
```

You will see that the general properties of the models are exactly the same. We will see that they also give the same predictions; copy the code below to make a nice plot using the first model:

Note: From this point on, we will slightly deviate from the code given in ISL. The reason for this is that the models defined above are called fit, fit2 and fit2a and in the code below we will need to use the $fit attributes of predictions which has caused confusion in the class. Generally, using “fit” and its variants seems like bad practice to me and I would recommend using more specific names for the models we create. Therefore, new model names have been defined below and the code has been rewritten to include the new model names. Additionally, we simplify the code to avoid the usage of the “cbind” and “matlines” functions, because one can perfectly well make the plot without those with a code that may be easier to interpret. However, if you are interested in the details of using these functions, type ?cbind and ?matlines for more information.

```
OrthModel=fit
RawModel=fit2
RawModel2=fit2a
rm(fit)
rm(fit2)
```

rm(fit2a)

agelims=range(age)
This command defined a vector called agelims that includes the minimum and maximum values of age in the data.

age.grid=seq(from=agelims[1],to=agelims[2])
This command created a variable called age.grid as a sequence of numbers from the minimum age to the maximum age, i.e. 18, 19, 20, …, 79, 80.

preds=predict(OrthModel,data.frame(age=age.grid),se=TRUE)
Here we make predictions using OrthModel for all age values in age.grid and ask for standard errors as well.

par(mfrow=c(1,2),mar=c(4.5,4.5,1,1),oma=c(0,0,4,0))
The mar and oma arguments are used to control the margins of the subplots.

plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
In this command, the argument xlim sets the range of values to be plotted on the x-axis.

title("Degree-4 Polynomial",outer=T)
Here, the outer=T argument specifies that the title is applied to all subplots together in the plotting screen.

Now we plot the estimates as well as the confidence interval limits:
lines(age.grid,preds$fit,lwd=2,col="blue")
lines(age.grid,preds$fit+2*preds$se.fit,lwd=1,lty=3,col="blue")
lines(age.grid,preds$fit-2*preds$se.fit,lwd=1,lty=3,col="blue")

Now try adding a similar plot using the second model (i.e. fit2)!
This can be done by similar commands as above; note that the second model has been renamed from fit2 to RawModel. As agelims and age.grid is the same as before, we don't need to repeat the corresponding lines. However, we want to make some new predictions, this time with RawModel:
preds2=predict(RawModel,data.frame(age=age.grid),se=TRUE)

The plotting screen is already split in two parts, so we don't need to repeat the par(mfrow= …) command here. However, we again need to plot the wage vs age points for the new plot:
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")

As we already have a title that applies to the whole plot, we don't need to repeat that command either. Therefore, all that is left is to plot the prediction curve and the confidence interval limits for the new predictions:
lines(age.grid,preds$fit,lwd=2,col="blue")
lines(age.grid,preds$fit+2*preds$se.fit,lwd=1,lty=3,col="blue")
lines(age.grid,preds$fit-2*preds$se.fit,lwd=1,lty=3,col="blue")

The plots created this way look identical, so RawModel seems to work as well in this case as OrthModel.

We can also check that the difference in predictions is essentially zero:

```
max(abs(preds$fit-preds2$fit))
```

This command takes the difference of the fitted values in preds and preds2 (i.e. the prediction values for all age values using OrthModel and RawModel respectively), takes the absolute value of that difference (so that negative differences are made into positive) and asks for the maximum of the absolute values. In other words, this command quantifies the biggest possible difference between the predictions when using OrthModel and RawModel. As you see from the output, the largest difference in predictions is extremely small.

10. If you are done with the previous exercises, do exercises 9 and 13 in Section 3.7 of ISL (pages 122-125). The Auto dataset is included in the ISLR library, so it does not need to be downloaded separately.
    The solutions of these exercises are not provided here. Both exercises are similar to those whose solutions are discussed above, in particular Exercises 7-8 above.

11. 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 69 36 52.