Lab Notes

Chapter 7

OpenIntro Biostatistics

Overview

1. Introduction to Multiple Regression
   - *OI Biostat* Sections 7.1 and 7.2
2. Evaluating Model Fit
   - *OI Biostat* Section 7.3
3. Categorical Predictors with Several Levels and Inference in Regression
   - *OI Biostat* Sections 7.4 - 7.6 and 7.9
4. Interaction
   - *OI Biostat* Section 7.7
5. Model Selection for Explanatory Models
   - *OI Biostat* Section 7.8

Lab 1 introduces the multiple regression model in the context of estimating an association between a response variable and primary predictor of interest while adjusting for possible confounding variables.

Lab 2 discusses the use of residual plots to check assumptions for multiple regression and introduces adjusted R^2.

Lab 3 extends on the topics introduced in Chapter 6, Lab 4 by discussing categorical predictors with more than two levels and generalizing inference in regression to the setting where there are several slope parameters.

Lab 4 introduces the concept of a statistical interaction, specifically in the case of an interaction between a categorical variable and a numerical variable.

Lab 5 discusses explanatory modeling, in which the goal is to construct a model that explains the observed variation in the response variable. This is an application of multiple regression distinct from that presented in Lab 1.

Lab 1: Introduction to Multiple Regression

Working with Several Predictors

The **lm()** function is used to fit linear models. It has the following generic structure:

```
lm(y ~ x1 + x2, data)
```

where the first argument specifies the variables used in the model; in this example, the model regresses a response variable y against two explanatory variables x1 and x2. Additional predictor variables can be added to the model formula with the + symbol.

The following example shows fitting a linear model that predicts BMI from age (in years) and gender using data from `nhanes.samp.adult.500`, a sample of individuals 21 years of age or older from the NHANES data.

```
#load the data
library(oibiostat)
data("nhanes.samp.adult.500")

#fitting linear model
lm(BMI ~ Age + Gender, data = nhanes.samp.adult.500)
```

```
##
## Call:
## lm(formula = BMI ~ Age + Gender, data = nhanes.samp.adult.500)
##
## Coefficients:
## (Intercept)          Age   Gendermale
##     28.8087       0.0206      -0.9571
```

Letting R do the Work: Predicted Values

The `predict()` function can be used to evaluate the regression equation for specific x-values, or in other words, to calculate $\hat{y}$ values for values of x that were not necessarily observed. To use `predict()` in this way, specify the x-values according to the following generic syntax:

```
predict(object, newdata = data.frame( ))
```

where `object` is the name of the fitted model, and the name of the predictor variable and value at which to evaluate the equation are specified within `newdata = data.frame()`.

In a model with several variables, values for all variables in the model must be specified to calculate a prediction.

The following example shows calculating $\widehat{BMI}$ for a male individual 60 years of age using the model regressing BMI on age and gender in `nhanes.samp.adult.500`, then checking the result by explicitly solving the regression equation.

```
#BMI ~ Age + Gender in nhanes.samp.adult.500
model.BMIvsAgeGender = lm(BMI ~ Age + Gender, data = nhanes.samp.adult.500)
predict(model.BMIvsAgeGender, newdata = data.frame(Age = 60, Gender = "male"))
```

```
##     1
## 29.09
```

```
#confirm answer from solving 28.81 + 0.02(60) - 0.95(1)
coef(model.BMIvsAgeGender)[1] + coef(model.BMIvsAgeGender)[2]*60 +
  coef(model.BMIvsAgeGender)[3]*1
```

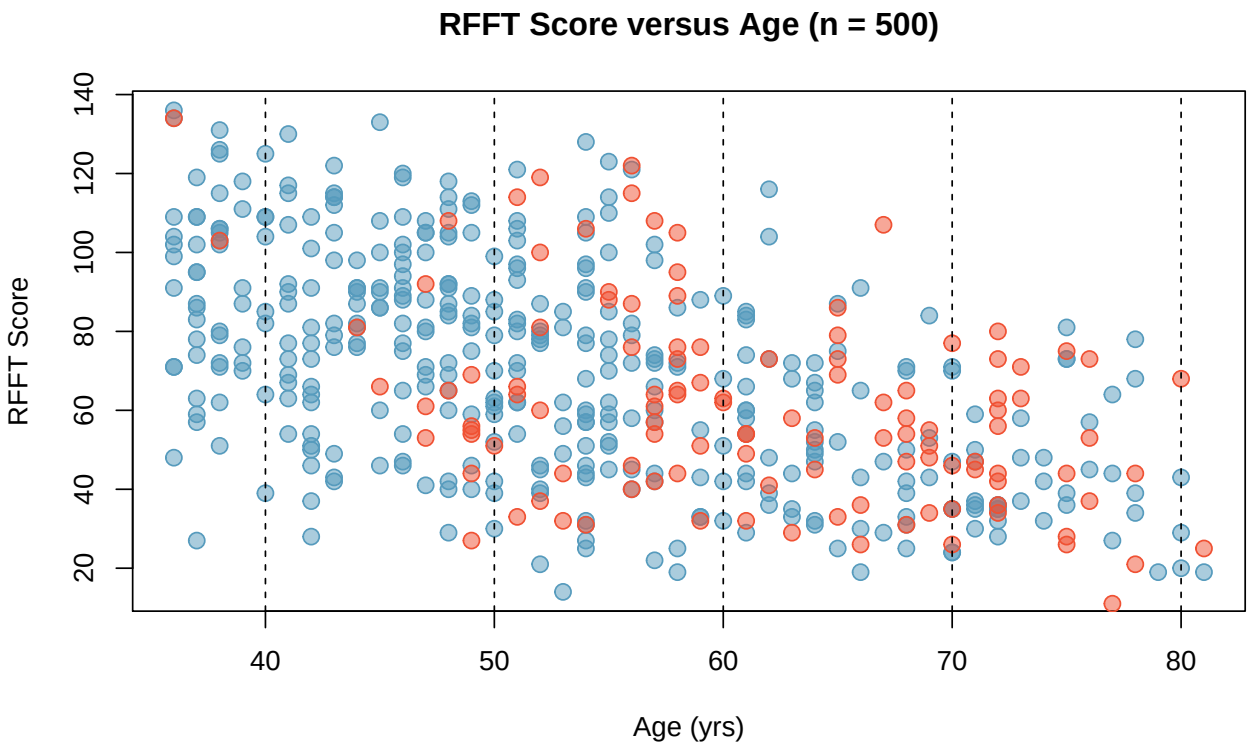
```
## (Intercept)
##       29.09
```

Plotting Points According to a Condition

The following plot was generated by using `plot( )` to plot a set of points with the associated plot axes; in this example, the cases representing statin non-users were plotted first. Afterwards, the **`points( )`** function was used to overlay a set of points onto the existing plot; these points in red represent statin users. The `points( )` function takes the same arguments as `plot( )`.

Refer to the notes for Lab 3 for details about using `COL` to specify colors.

```
#load the data
library(openintro)
library(oibiostat)
data(prevend.samp)
data(COL)

#create statin.use logical
statin.use = (prevend.samp$Statin == 1)

#plot blue points
plot(prevend.samp$Age[statin.use == FALSE], prevend.samp$RFFT[statin.use == FALSE],
     pch = 21, bg = COL[1, 3], col = COL[1], cex = 1.3,
     xlab = "Age (yrs)", ylab = "RFFT Score",
     main = "RFFT Score versus Age (n = 500)")

#plot red points
points(prevend.samp$Age[statin.use == TRUE], prevend.samp$RFFT[statin.use == TRUE],
     pch = 21, bg = COL[4, 3], col = COL[4], cex = 1.3)

#draw vertical lines
abline(v = 40, lty = 2)
abline(v = 50, lty = 2)
abline(v = 60, lty = 2)
abline(v = 70, lty = 2)
abline(v = 80, lty = 2)
```


RFFT Score versus Age (n = 500)
RFFT Score
20
40
60
80
100
120
140
40
50
60
70
80
Age (yrs)

Lab 2: Evaluating Model Fit

Extracting Adjusted R^2 from a Model Fit

The adjusted R^2 of a model fit can be extracted directly from a model summary with the use of the `$` syntax. For example, the following syntax prints the adjusted R^2 from the model regressing BMI on age and gender in `nhanes.samp.adult.500`.

```
#print adjusted R-squared value
summary(model.BMIvsAgeGender)$adj.r.squared
```

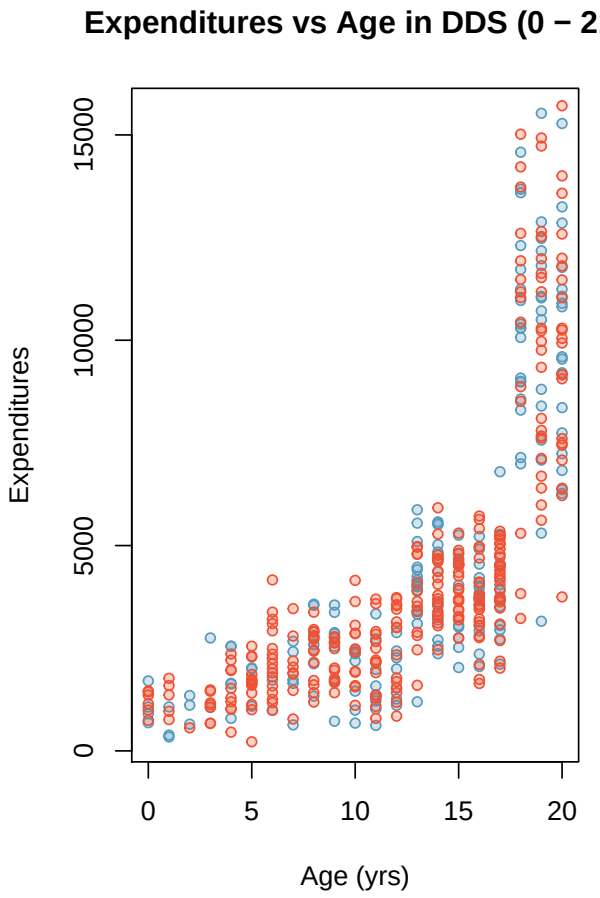
```
## [1] 0.0032735
```

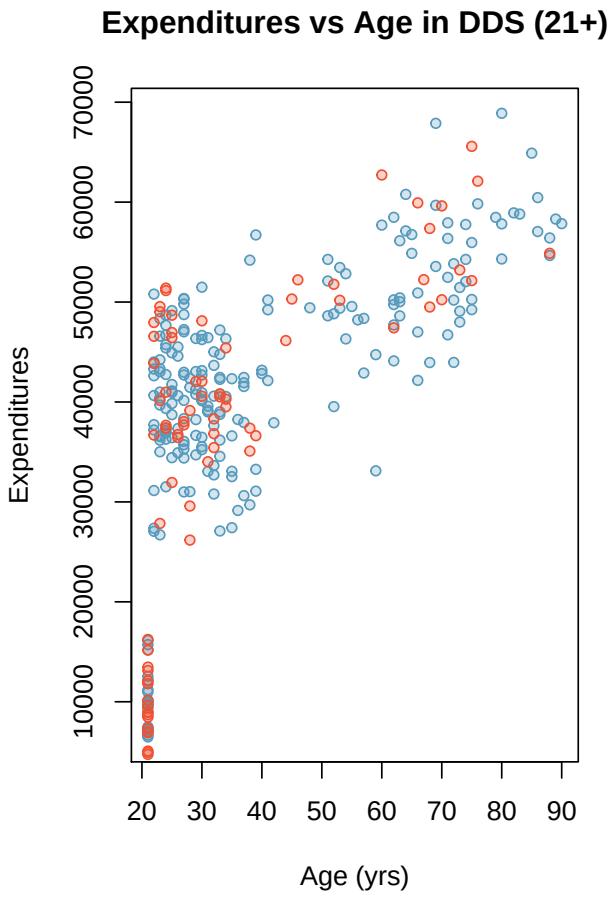
Plotting Points According to Several Conditions

The following plots were generated by defining logical variables for ethnicity and logical variables for age. The logical operator `&` was used to succinctly specify which data values should appear in each plot, and specify the display color.

```
#create hispanic and white.not.hisp logicals
hispanic = (dds.subset$ethnicity == "Hispanic")
white.not.hisp = (dds.subset$ethnicity == "White not Hispanic")

#create age logicals
younger = (dds.subset$age < 21)
older = (dds.subset$age >= 21)

par(mfrow = c(1, 2))

#plot blue points, white not hispanic
plot(expenditures[white.not.hisp & younger] ~ age[white.not.hisp & younger],
     data = dds.subset,
     pch = 21, bg = COL[1, 4], col = COL[1], cex = 0.8,
     xlab = "Age (yrs)", ylab = "Expenditures",
     main = "Expenditures vs Age in DDS (0 - 21)")

#plot red points, hispanic
points(expenditures[hispanic & younger] ~ age[hispanic & younger],
       data = dds.subset, pch = 21, bg = COL[4, 4], col = COL[4],
       cex = 0.8)

#plot blue points, white not hispanic
plot(expenditures[white.not.hisp & older] ~ age[white.not.hisp & older],
     data = dds.subset,
     pch = 21, bg = COL[1, 4], col = COL[1], cex = 0.8,
     xlab = "Age (yrs)", ylab = "Expenditures",
     main = "Expenditures vs Age in DDS (21+)")
```

```
#plot red points, hispanic
points(expenditures[hispanic & older] ~ age[hispanic & older],
       data = dds.subset, pch = 21, bg = COL[4, 4], col = COL[4],
       cex = 0.8)
```





Lab 3: Categorical Predictors with Several Levels and Inference in Regression

Color Palettes with RColorBrewer

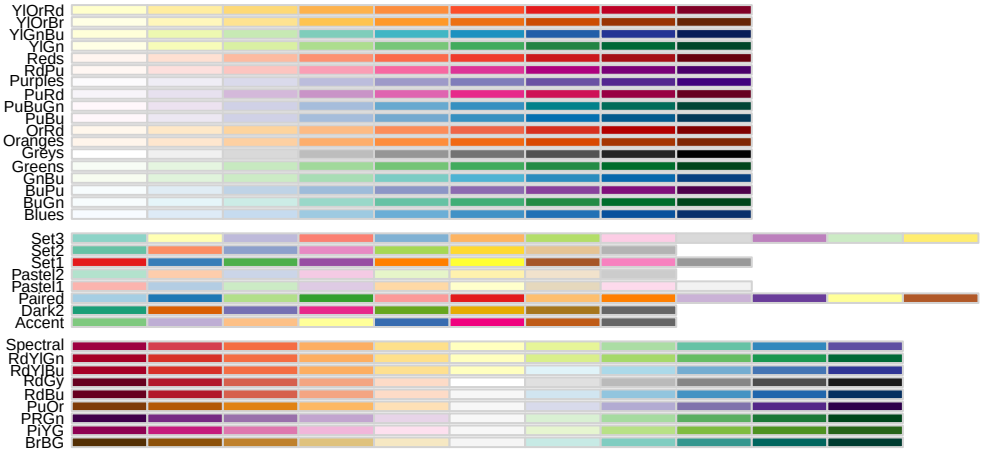
The `RColorBrewer` package is a helpful tool for selecting colors. The package provides three types of palettes: sequential palettes, diverging palettes, and qualitative palettes.

- Sequential palettes are ideal for ordered data that progress across a range from low to high.
- Diverging palettes place equal emphasis on extremes at both ends of the data range.
- Qualitative palettes are useful for unordered categorical data.

The **`brewer.pal()`** function has the generic structure

```
brewer.pal(n, name)
```

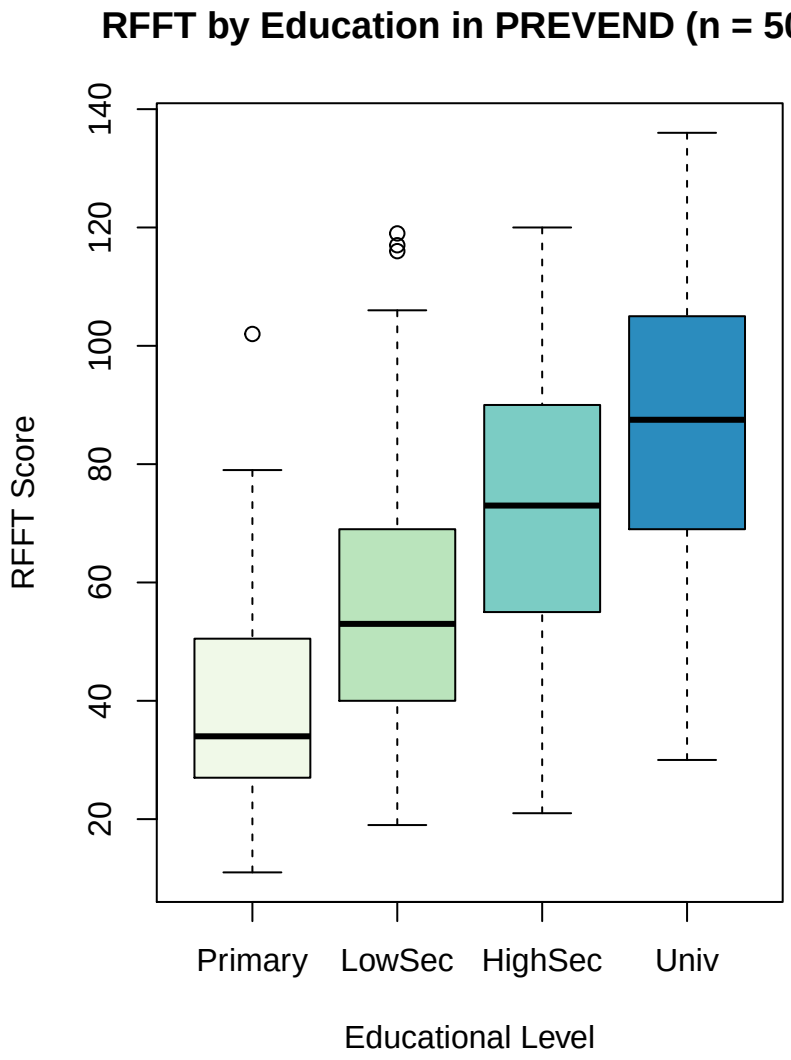
where the first argument specifies the number of colors and the second specifies the name of the palette. The possible palette choices are shown below.



The following example shows a figure created using the `GnBu` palette, one of the sequential palettes. A color scheme progressing from a light shade to a dark shade highlights the ordinal nature of educational level and adds interpretive value to the visual.

```
#load RColorBrewer package
library(RColorBrewer)

#load data
data("prevend.samp")

#convert Education to a factor
prevend.samp$Education = factor(prevend.samp$Education,
                                levels = c(0, 1, 2, 3),
                                labels = c("Primary", "LowerSecond",
                                           "HigherSecond", "Univ"))
```

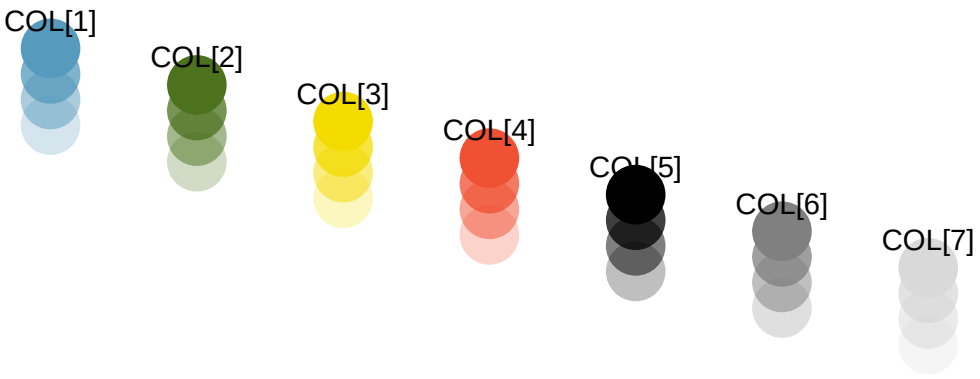
```r
#create plot
plot(RFFT ~ Education, data = prevend.samp,
     xlab = "Educational Level", ylab = "RFFT Score",
     main = "RFFT by Education in PREVEND (n = 500)",
     names = c("Primary", "LowSec", "HighSec", "Univ"),
     col = brewer.pal(4, "GnBu"))
```



Using Colors from OpenIntro

The core colors used for the *OpenIntro Statistics* textbook are accessible as the `COL` dataset in the `openintro` package.

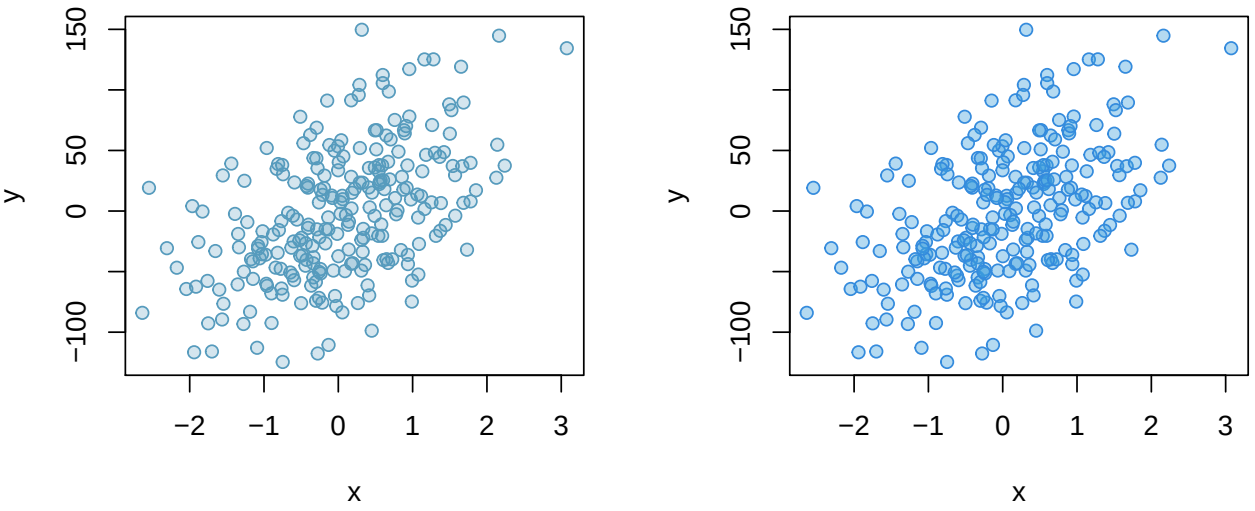
Each of the seven colors shown below have four levels of transparency, which can be useful when making scatterplots that display a large number of observations.



The colors can be accessed with `COL[]`. A single number within the square brackets identifies each color. For example, `COL[1]` corresponds to blue. A second number within the square brackets can be optionally specified to change the transparency, where 4 is the most transparent and 1 is opaque.

For example, the plot below on the left is made using the *OpenIntro* color package. The dots are outlined in an opaque blue (`col = COL[1]`) while the fill color is in a transparent blue (`col = COL[1, 4]`).

```
#simulate data
x <- rnorm(250)
error <- rnorm(250, 0, 50)
y <- 1.3*x^2 + 25*x + error

par(mfrow = c(1, 2))

#plot with openintro colors
plot(y ~ x,
     pch = 21, col = COL[1], bg = COL[1, 4])
```



The plot on the right is made using `rgb( )`, a function for specifying color using the RGB color codes. The **`rgb( )`** function has the generic structure

```
rgb(red, green, blue, max, alpha)
```

where red, blue, green, and alpha can be entered as integers if max is set to 255. The setting alpha is used to specify transparency; a value closer to 0 is more transparent. By default, a color specified will be opaque.

```
#plot with rgb( )
plot(y ~ x, pch = 21,
     col = rgb(52, 139, 221, max = 255),
     bg = rgb(62, 160, 221, max = 255, alpha = 100))
```

Lab 4: Interaction

To fit a model with an interaction term and its main effects, use the `*` symbol in `lm()`. The following example shows fitting a linear model that predicts BMI from age, gender, and the interaction between age and gender in `nhanes.samp.adult.500`.

```
lm(BMI ~ Age*Gender, data = nhanes.samp.adult.500)
```

```
##
## Call:
## lm(formula = BMI ~ Age * Gender, data = nhanes.samp.adult.500)
##
## Coefficients:
##    (Intercept)             Age      Gendermale  Age:Gendermale
##       28.69400         0.02296         -0.69234        -0.00529
```

This is equivalent to explicitly specifying the main effects and the interaction term individually with the `:` symbol.

```
lm(BMI ~ Age + Gender + Age:Gender, data = nhanes.samp.adult.500)
```

```
##
## Call:
## lm(formula = BMI ~ Age + Gender + Age:Gender, data = nhanes.samp.adult.500)
##
## Coefficients:
##    (Intercept)             Age      Gendermale  Age:Gendermale
##       28.69400         0.02296         -0.69234        -0.00529
```

It is also possible to use `()^2`, which indicates to R that all main effects and "second-order" interaction terms should be fit. Interaction terms involving more than two terms are not discussed in this course, but this syntax could be useful in such a setting.

```
lm(BMI ~ (Age + Gender)^2, data = nhanes.samp.adult.500)
```

```
##
## Call:
## lm(formula = BMI ~ (Age + Gender)^2, data = nhanes.samp.adult.500)
##
## Coefficients:
##    (Intercept)             Age      Gendermale  Age:Gendermale
##       28.69400         0.02296         -0.69234        -0.00529
```

Lab 5: Model Selection for Explanatory Models

Scatterplot Matrices

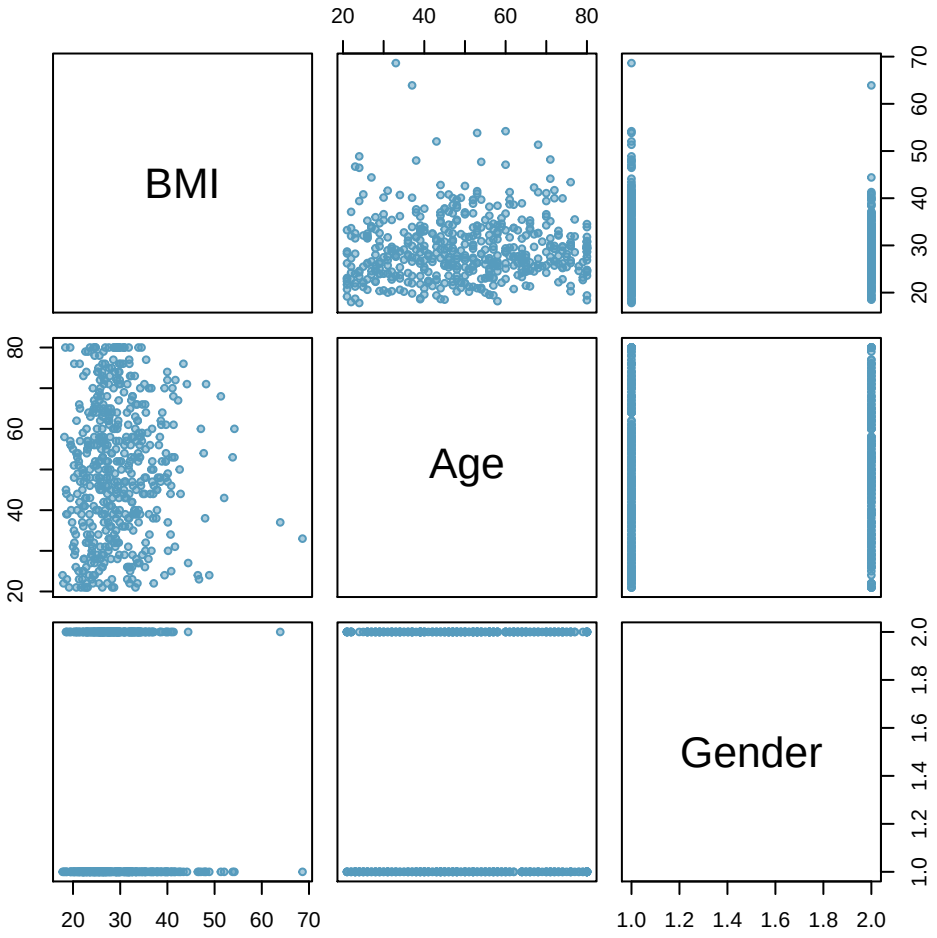
The **pairs()** function creates a matrix of scatterplots. It has the following generic structure

```
pairs(formula, data = )
```

where `formula` is in the form `~ x + y + z`, with each term giving a separate variable in the pairs plot and `data` specifies the data frame for the variables. The function can also take the same graphical parameters that the `plot()` function uses.

The following example shows a scatterplot matrix for the variables BMI, age, and gender in `nhanes.samp.adult.500`.

```
pairs(~ BMI + Age + Gender, data = nhanes.samp.adult.500,
      pch = 21, cex = 0.7, bg = COL[1, 3], col = COL[1])
```



Correlation Matrices

The `cor()` function was previously introduced to calculate the correlation between two variables. When applied to a matrix, the function will output a correlation matrix of all pairwise correlations.

The subset() command has an argument, select, useful for pulling out the variables of interest prior to generating a correlation matrix.

In the following examples, the variables BMI, height, and weight are subsetted from nhanes.samp.adult.500 and stored as a matrix from which a correlation matrix is generated.

```
#subset nhanes.samp.adult.500
nhanes.subset = subset(nhanes.samp.adult.500, select = c(BMI, Height, Weight))

#generate corelation matrix
cor(nhanes.subset, use = "complete.obs")
```

```
##             BMI   Height  Weight
## BMI     1.00000 -0.08178 0.86797
## Height -0.08178  1.00000 0.41023
## Weight  0.86797  0.41023 1.00000
```

Collapsing Factor Levels

The factor() function can also be used to collapse levels of a factor.

The following example shows the re-defining of the levels of grazing.intensity; the variable initially has five levels (light, less than average, average, moderately heavy, heavy). The first four levels can be combined into a single level NotHeavy, while the level heavy is renamed Heavy.

```
#load data
data("forest.birds")

#view levels of grazing.intensity
levels(forest.birds$grazing.intensity)
```

```
## [1] "light"             "less than average" "average"
## [4] "moderately heavy"  "heavy"
```

```
#create the grazing.binary variable
forest.birds$grazing.binary = forest.birds$grazing.intensity

#redefine the levels of grazing.binary
levels(forest.birds$grazing.binary) = list("NotHeavy" = c("light", "less than average",
                                                          "average", "moderately heavy"),
                                           "Heavy" = "heavy")

#view levels of grazing.binary
levels(forest.birds$grazing.binary)
```

```
## [1] "NotHeavy" "Heavy"
```

```
#compare tables
table(forest.birds$grazing.intensity)
```

```
##
```

```
##              light less than average           average  moderately heavy
##                 13                   8                15                 7
##              heavy
##                 13
```

```r
table(forest.birds$grazing.binary)
```

```
##
## NotHeavy    Heavy
##       43       13
```