Lab Notes

Chapter 6

OpenIntro Biostatistics

Overview

1. Examining Scatterplots
   - *OI Biostat* Section 6.1
2. Introduction to Least Squares Regression
   - *OI Biostat* Sections 6.2 and 6.3.1
3. Understanding R^2
   - *OI Biostat* Section 6.3.2
4. Categorical Predictors with Two Levels and Inference in Regression
   - *OI Biostat* Sections 6.3.3 and 6.4

Lab 1 introduces the idea of using a straight line to summarize data that exhibit an approximately linear relationship and the mechanics of fitting and interpreting a line of best fit.

Lab 2 formally introduces the statistical model for least squares regression and discusses the residual plots used to assess the assumptions for linear regression.

Lab 3 explores the idea behind the quantity R^2 by sampling observations according to a population regression model with known parameters.

Lab 4 discusses the use of binary categorical predictor variables and the extension of statistical inference to a regression context.

Lab 1: Examining Scatterplots

Fitting and Plotting a Least Squares Model

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

```
lm(y ~ x, data)
```

where the first argument specifies the variables used in the model; in this example, the model regresses a response variable `y` against an explanatory variable `x`. The second argument is used only when the dataframe name is not already specified in the first argument. Running the function creates an *object* (of class '`lm`') that contains several components, such as the model coefficients. The model coefficients are directly displayed upon running `lm()`, while other components can be accessed through either the `$` notation or specific functions like `summary()`.

The following example shows fitting a linear model that predicts BMI from age (in years) using data from `nhanes.samp.adult.500`, a sample of individuals 21 years of age or older from the NHANES data. The first use of `lm()` specifies the name of the dataframe using the `$` notation with each variable name, while the second uses the `data` argument to indicate that both variables are in the `nhanes.samp.adult.500` dataframe.

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

#fitting linear model
lm(nhanes.samp.adult.500$BMI ~ nhanes.samp.adult.500$Age)
```

```
##
## Call:
## lm(formula = nhanes.samp.adult.500$BMI ~ nhanes.samp.adult.500$Age)
##
## Coefficients:
##                (Intercept)  nhanes.samp.adult.500$Age
##                   28.40113                    0.01982
```

```
#equivalently...
lm(BMI ~ Age, data = nhanes.samp.adult.500)
```

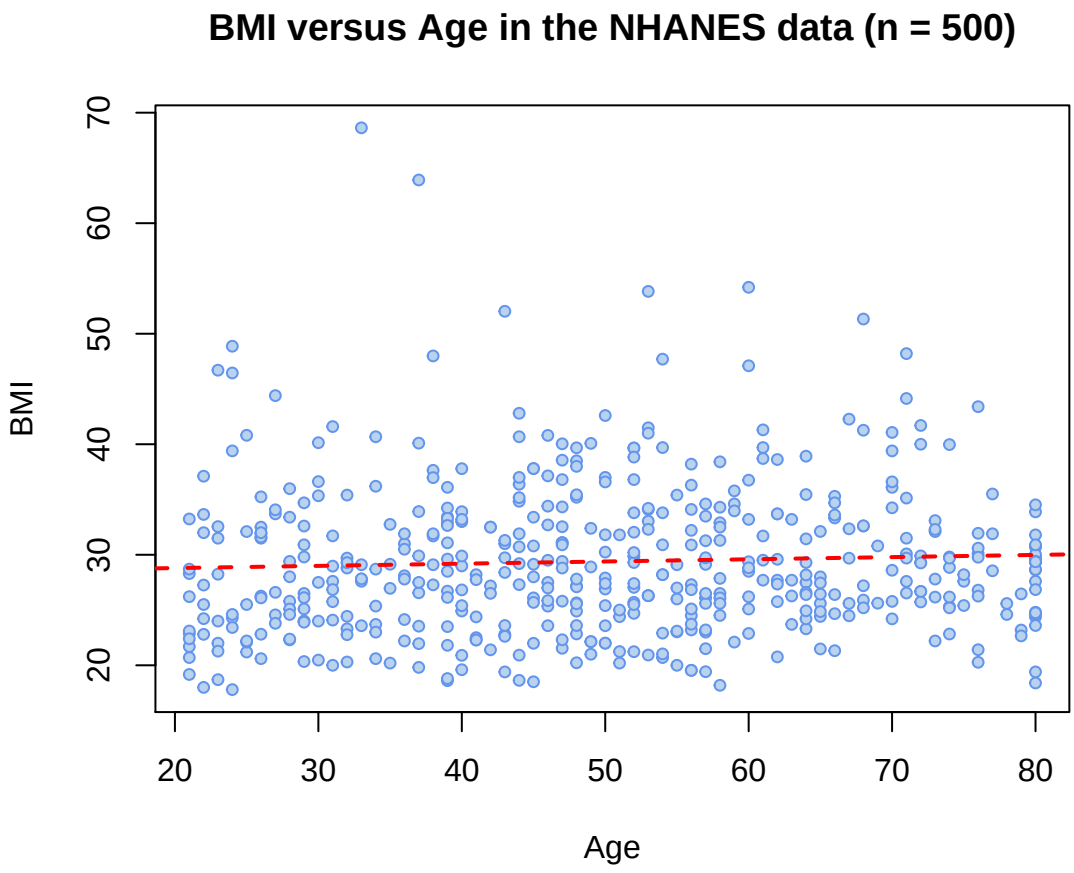
```
##
## Call:
## lm(formula = BMI ~ Age, data = nhanes.samp.adult.500)
##
## Coefficients:
## (Intercept)          Age
##    28.40113      0.01982
```

To add the least squares regression line to a scatterplot, use the **abline()** function on the model. The abline() function was introduced in the Chapter 4 Lab Notes (Lab 1).

The following example shows a scatterplot with a least squares regression line. Additional plot options have been specified to add a plot title (main), change the plotting symbol (pch) so that an outline color (col) and fill color (bg) can be specified, and reduce the size of the dots (cex).

```
#plot the data
plot(BMI ~ Age, data = nhanes.samp.adult.500,
     main = "BMI versus Age in the NHANES data (n = 500)",
     pch = 21, col = "cornflowerblue", bg = "slategray2",
     cex = 0.75)

#add least squares line
abline(lm(BMI ~ Age, data = nhanes.samp.adult.500),
       col = "red", lty = 2, lwd = 2)
```



Lab 2: Introduction to Least Squares Regression

Extracting Residuals and Predicted Values from a Model Fit

The main type of residual plot used in *OpenIntro Biostatistics* is a scatterplot in which the residuals are plotted on the vertical axis against predicted values from the model on the horizontal axis. Predicted values can also be referred to as 'fitted' values.

The residuals can be extracted from a model object using either the **`residuals()`** function (which can be abbreviated as `resid()`) or the `$` notation.

The following example demonstrates extracting residuals from the model of BMI versus age in the sample of 500 adults from the NHANES data, then printing out the first five residual values. It can be convenient to assign a model a specific name then refer to the model name in subsequent functions, rather than repeat the call to `lm()`.

```
#name the model object
model.BMIvsAge = lm(BMI ~ Age, data = nhanes.samp.adult.500)

#extract residuals with residuals()
residuals = residuals(model.BMIvsAge)
residuals[1:5]
```

```
##        5514        7882        2619        8361        8725
## -1.49196704  0.06748322 -3.96270002 -3.15599844 -2.49196704
```

```
#alternatively... extract residuals with $residuals
residuals = model.BMIvsAge$residuals
residuals[1:5]
```

```
##        5514        7882        2619        8361        8725
## -1.49196704  0.06748322 -3.96270002 -3.15599844 -2.49196704
```

The predicted values can be extracted from a model object using either the **`predict()`** function, **`fitted()`** function, or the `$` notation.

The following example demonstrates extracting predicted values from the model of BMI versus age in `nhanes.samp.adult.500`, then printing out the first five predicted values.

```
#extract predicted values with predict()
predicted = predict(model.BMIvsAge)
predicted[1:5]
```

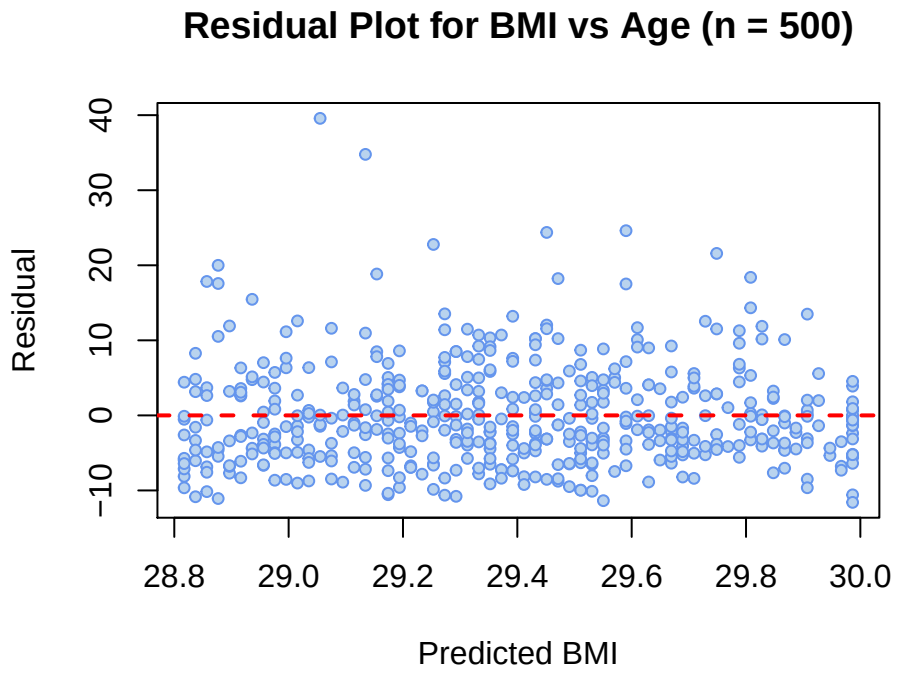
```
##     5514     7882     2619     8361     8725
## 29.39197 29.33252 29.31270 28.95600 29.39197
```

```
#alternatively... extract predicted values with $fitted.values
predicted = model.BMIvsAge$fitted.values
predicted[1:5]
```

```
##     5514     7882     2619     8361     8725
## 29.39197 29.33252 29.31270 28.95600 29.39197
```

The following example demonstrates a residual plot for the model regressing BMI on age in `nhanes.samp.adult.500`.

```
plot(residuals ~ predicted,
     main = "Residual Plot for BMI vs Age (n = 500)",
     xlab = "Predicted BMI", ylab = "Residual",
     pch = 21, col = "cornflowerblue", bg = "slategray2",
     cex = 0.75)
abline(h = 0, lty = 2, lwd = 2, col = "red")
```

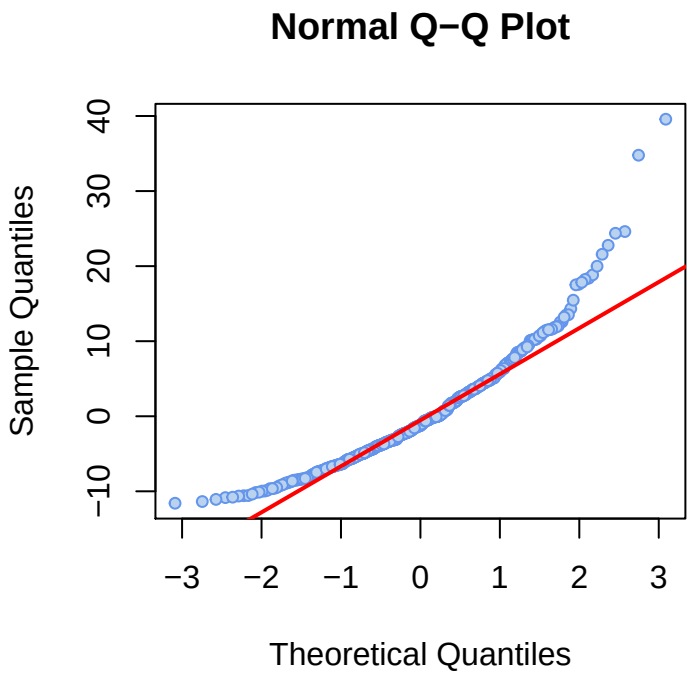


Normal Probability Plot of Residuals

The **qqnorm()** function produces a normal quantile-quantile plot of a set of values while **qqline()** adds a diagonal line through the first and third quartiles.

The following example demonstrates a Q-Q plot of residuals from the model regressing BMI on age in `nhanes.samp.adult.500`.

```
#normal probability plot of residuals
qqnorm(residuals,
     pch = 21, col = "cornflowerblue", bg = "slategray2", cex = 0.75)
qqline(residuals,
       col = "red", lwd = 2)
```



Lab 3: Understanding R^2

Extracting R^2 from a Model Fit

The use of `summary()` on a model object will be discussed in the next section. The 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 R^2 from the model regressing BMI on age in `nhanes.samp.adult.500`.

```
#print R-squared value
summary(model.BMIvsAge)$r.squared
```

```
## [1] 0.00237723
```

Removing Objects from the RStudio Environment

To remove a specific object from the environment, use **`rm()`** on the name of the object. For example, to remove the previously created model object `model.BMIvsAge`, run

```
rm(model.BMIvsAge)
```

To clear all objects from the environment, run **`rm(list = ls())`**; this is equivalent to clicking the broom icon in the Environment tab.

Lab 4: Categorical Predictors and Inference in Regression

Creating Factor Variables

The **factor()** function has the following generic structure:

```
factor(x, levels, labels)
```

where x is a vector of data (usually with a small number of distinct values), levels is a vector of the unique values that x might have taken, and labels is a character vector of labels for the levels (in the same order as levels).

The following example shows the creation of a factor variable DM.factor based on the integer vector DM in prevend.sample. The variable DM takes on values of either 0 or 1, where 0 corresponds to absence of diabetes and 1 corresponds to presence of diabetes.

```
#load the data
data("prevend")
set.seed(5011)
prevend.sample = prevend[sample(1:nrow(prevend), 500, replace = FALSE), ]

#create DM.factor
DM.factor = factor(prevend.sample$DM, levels = c(0, 1),
                   labels = c("Absent", "Present"))
```

To overwrite the variable DM in prevend.sample with DM.factor, assign DM.factor to DM. For clarity of logic, the assignment operator <- is used rather than the equivalent = symbol; think of the factor version of the variable is being assigned *to* the existing variable name DM in prevend.sample.

```
#overwrite DM with DM.factor
prevend.sample$DM <- DM.factor

#confirm the overwrite is successful
summary(prevend.sample$DM)
```

```
##  Absent Present
##     467      33
```

Directly Converting a Variable in a Dataframe to a Factor

Note that the variable DM.factor was not part of the prevend.sample dataframe, even if it was created from a variable in prevend.sample. To specify that a created variable should be placed in a dataframe, specify the name of the dataframe with $ when using factor(). The variable will be added as the last variable in the dataframe.

The following example shows a factor version of the variable Gender being added to prevend.sample.

```
#create Gender.factor in prevend.sample
prevend.sample$Gender.factor = factor(prevend.sample$Gender, levels = c(0, 1),
                                      labels = c("Male", "Female"))
```

```r
#view first five rows and last three columns of prevend.sample
prevend.sample[1:5, 30:32]
```

```
##      Match_1 Match_2 Gender.factor
## 2266     816     113        Female
## 3235     727     242        Female
## 1068      -1      -1          Male
## 3422     838      -1        Female
## 3570      -1     276          Male
```

Using the same $ syntax, the factor version of the variable can be directly assigned to the original variable in the dataframe, essentially "converting" it from an integer vector to a factor with a single command:

```r
#convert Gender to a factor variable
prevend.sample$Gender = factor(prevend.sample$Gender, levels = c(0, 1),
                                       labels = c("Male", "Female"))

#confirm that Gender is now a factor
summary(prevend.sample$Gender)
```

```
##   Male Female
##    265    235
```

Working with Linear Models

Model Summary

Applying the `summary()` command to a model fit outputs a list of information about the model, including the coefficient estimates, standard errors, t-statistics, and p-values. The R^2 value is labeled 'Multiple R-squared'.

The following example shows the summary for the model regressing BMI on age in `nhanes.samp.adult.500`. An equivalent syntax would be to use the name of the model as defined previously, `summary(model.BMIvsAge)`.

```r
summary(lm(BMI ~ Age, data = nhanes.samp.adult.500))
```

```
##
## Call:
## lm(formula = BMI ~ Age, data = nhanes.samp.adult.500)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.586  -4.668  -1.235   3.610  39.575
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 28.40113    0.96172  29.531   <2e-16 ***
```

```
## Age          0.01982    0.01825   1.086    0.278
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.815 on 495 degrees of freedom
##   (3 observations deleted due to missingness)
## Multiple R-squared:  0.002377,   Adjusted R-squared:  0.0003618
## F-statistic:  1.18 on 1 and 495 DF,  p-value: 0.278
```

Extracting Components of a Model Fit and Summary

The coefficients of a model fit can be extracted using either the **coef()** function or the $ syntax.

```
#extract coefficients with coef()
coef(model.BMIvsAge)
```

```
## (Intercept)         Age
## 28.40112932  0.01981675
```

```
#alternatively... extract coefficients with $coefficients
model.BMIvsAge$coefficients
```

```
## (Intercept)         Age
## 28.40112932  0.01981675
```

Similarly, the coefficients of a model summary can also be extracted; this will not only output the estimates of the coefficients, but also the associated standard errors, t-statistics, and p-values.

```
#extract coefficients with coef()
coef(summary(model.BMIvsAge))
```

```
##                Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) 28.40112932 0.96172389 29.531480 2.851707e-111
## Age          0.01981675 0.01824641  1.086063  2.779797e-01
```

```
#alternatively... extract coefficients with $coefficients
summary(model.BMIvsAge)$coefficients
```

```
##                Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) 28.40112932 0.96172389 29.531480 2.851707e-111
## Age          0.01981675 0.01824641  1.086063  2.779797e-01
```

Square bracket notation can be used to isolate specific information from the coefficients.

```
#extract the estimate of age from the model fit
coef(model.BMIvsAge)[2]
```

```
##        Age
## 0.01981675
```

```
#extract the standard error of age with coef( ) syntax
coef(summary(model.BMIvsAge))[2, 2]
```

```
## [1] 0.01824641
```

```
#extract the t-statistic of age with $coefficients syntax
summary(model.BMIvsAge)$coefficients[2, 3]
```

```
## [1] 1.086063
```

Letting R Do the Work: Confidence Intervals

To calculate confidence intervals for the parameters in a regression model, use `confint()`. The **`confint()`** function has the following generic structure:

```
confint(object, parm, level = 0.95)
```

where `object` is the name of the fitted model, `parm` is an optional argument specifiying which parameters to calculate confidence intervals for, and `level` is the confidence level. The function outputs lower and upper confidence limits for each parameter. By default, the function calculates 95% confidence intervals for all parameters.

The following example shows the calculation of 95% confidence intervals for both β_0 and β_1 and a 90% confidence interval for only β_1, from the model regressing BMI on age in `nhanes.samp.adult.500`.

```
#output CIs for all parameters (intercept and slope)
confint(model.BMIvsAge)
```

```
##                   2.5 %      97.5 %
## (Intercept) 26.51156501 30.29069363
## Age         -0.01603322  0.05566672
```

```
#output CI only for slope parameter
confint(model.BMIvsAge, parm = "Age", level = 0.90)
```

```
##            5 %       95 %
## Age -0.0102522 0.04988571
```

Letting R Do the Work: Predicted Values

The `predict()` function has been previously used to extract the fitted values from a model object; i.e., the specific predicted y values for all x-values observed in the data.

The `predict()` function can also 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()`.

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

```
#BMI ~ Age in nhanes.samp.adult.500
predict(model.BMIvsAge, newdata = data.frame(Age = 60))
```

```
##        1
## 29.59013
```

```
#confirm answer from solving 28.40 + 0.0198(60)
coef(model.BMIvsAge)
```

```
## (Intercept)         Age
## 28.40112932  0.01981675
```

```
coef(model.BMIvsAge)[1] + coef(model.BMIvsAge)[2]*60
```

```
## (Intercept)
##    29.59013
```

To use `predict()` with a categorical predictor variable (stored as a factor), enter the name of the factor level. The following example shows calculating $\widehat{BMI}$ for a male individual in the model regressing BMI on gender in `prevend.sample`, then checking the result by printing the model intercept.

```
#BMI ~ Gender in prevend.sample
predict(lm(BMI ~ Gender, data = prevend.sample),
        newdata = data.frame(Gender = "Male"))
```

```
##        1
## 26.84449
```

```
#confirm answer from checking intercept
coef(lm(BMI ~ Gender, data = prevend.sample))
```

```
##  (Intercept) GenderFemale
##   26.8444875    0.1163529
```