R for Health Data Science

Week 04: Biostatistics Part 1

Sam Stewart

2021-03-12

```r
dat = read.csv("data/framinghamFirst.csv",header=TRUE,
               na.strings=".",stringsAsFactors=FALSE)
dat$BMIGroups = cut(dat$BMI,breaks=c(0,18.5,25,30,Inf),
                    labels=c("Underweight","Normal","Overweight","Obese"))
dat$SEX = factor(dat$SEX,levels=1:2,labels=c("Male","Female"))
dat$DIABETES = factor(dat$DIABETES,levels=0:1,labels=c("No Diabetes","Diabetes"))
dat$HYPERTEN = factor(dat$HYPERTEN,levels=0:1,labels=c("Normotensive","Hypertensive"))
```

1 Introduction

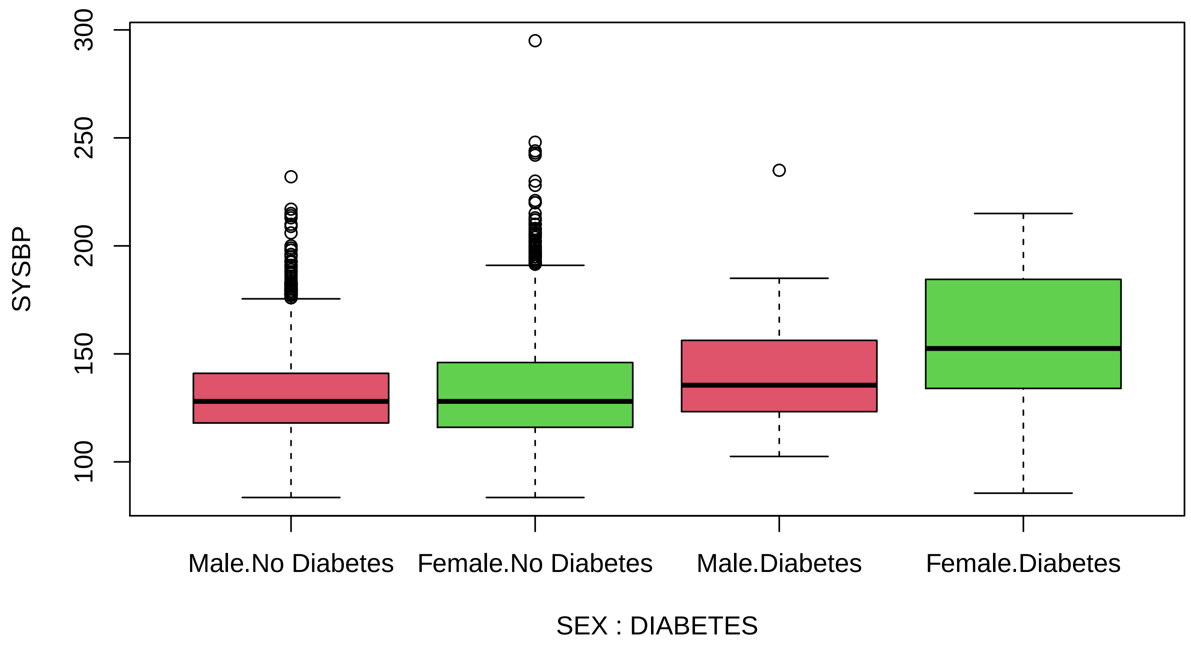
This week we're going to learn some actual statistics - largely the material covered in Biostats I in a typical graduate epi program. We'll cover the following functions

- Hypothesis tests
  - t-tests
  - chi-square tests
  - Fisher's tests
- regression models
  - linear regression
  - ANOVA and post-hoc comparisons
  - logistic regression
  - general linear models (GLM)

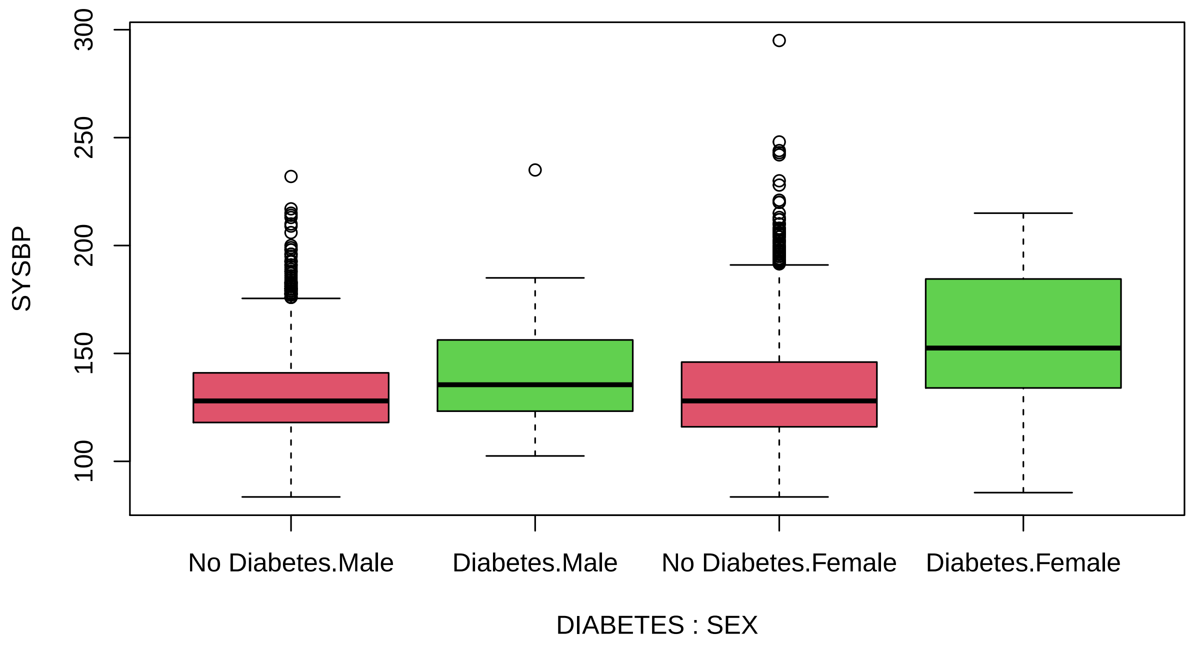
2 Formulas

We've seen formulas once before with boxplots, but they're worth covering again since they'll be used in several functions today. The general structure of a formula is `y~x`, where we're predicting `y` with `x`, or splitting `y` by `x`. Formulas are evaluated from the tilde-out, so in places where order matters they will evaluate in a specific order. Consider the following simple boxplot example.

```r
boxplot(SYSBP~SEX+DIABETES,data=dat,col=2:3)
```



```
boxplot(SYSBP~DIABETES+SEX,data=dat,col=2:3)
```



The order of the boxes is dependent on the structure of the formula. This won't matter in most uses (testing, modeling) but when it does be aware that you can re-arrange the formulas to alter the presentation of the results.

2.1 T-tests

We'll use the same command for 1-sample, 2-sample independent and 2-sample paired t-tests, `t.test`.

Format	Use
`t.test(x,mu=0)`	is a one-sample t-test of the variable `x` against a null value of 0.
`t.test(x,y)`	is an independent 2-sample t-test comparing the values of `x` to the values of `y`
`t.test(x,y,paired=TRUE)`	can be used for a paired t-test, though you could also create a variable called `z=y-x` and then do a 1-sample test on `z`
`t.test(y~x)`	is a 2-sample independent t-test of the variable `y` across the levels of `x`

While the main purpose of the function is to perform p-value testing, the most valuable part is the CIs and effect estimates, so we'll look at how to extract values from the function.

```
#testing Systolic blood pressure against a null of 0
t.test(dat$SYSBP)
```

```
##
##  One Sample t-test
##
## data:  dat$SYSBP
## t = 394.71, df = 4433, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  132.2476 133.5679
## sample estimates:
## mean of x
##  132.9078
```

```
#testing against a null of 130
t.test(dat$SYSBP,mu=130)
```

```
##
##  One Sample t-test
##
## data:  dat$SYSBP
## t = 8.6355, df = 4433, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 130
## 95 percent confidence interval:
##  132.2476 133.5679
## sample estimates:
## mean of x
##  132.9078
```

```
#saving the value and extracting the components
tTest01 = t.test(dat$SYSBP,mu=130)
tTest01
```

```
##
##  One Sample t-test
##
## data:  dat$SYSBP
## t = 8.6355, df = 4433, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 130
## 95 percent confidence interval:
##  132.2476 133.5679
## sample estimates:
## mean of x
##  132.9078
```

```r
names(tTest01)
```

```
##  [1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
##  [6] "null.value"  "stderr"      "alternative" "method"      "data.name"
```

```r
tTest01$statistic
```

```
##        t
## 8.635542
```

```r
tTest01$p.value
```

```
## [1] 8.036623e-18
```

```r
tTest01$estimate
```

```
## mean of x
##  132.9078
```

```r
tTest01$conf.int
```

```
## [1] 132.2476 133.5679
## attr(,"conf.level")
## [1] 0.95
```

You can see how you can extract all the components you might need - we'll see later in reporting how you can use these extractions to produce effective, concise reports.

```r
#testing systolic against diastolic BP
t.test(dat$SYSBP,dat$DIABP)
```

```
##
##  Welch Two Sample t-test
##
## data:  dat$SYSBP and dat$DIABP
## t = 130.32, df = 6798.6, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  49.07475 50.57365
## sample estimates:
## mean of x mean of y
## 132.90776  83.08356
```

```r
#testing systolic between sexes
t.test(dat$SYSBP~dat$SEX)
```

```
##
##  Welch Two Sample t-test
##
## data:  dat$SYSBP by dat$SEX
## t = -3.1622, df = 4430.6, p-value = 0.001577
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
```

```
##  -3.377678 -0.792332
## sample estimates:
##   mean in group Male mean in group Female
##             131.7369             133.8219
```

```
#almost all functions that take equations will let you submit a dataset
t.test(SYSBP~SEX,data=dat)
```

```
##
##  Welch Two Sample t-test
##
## data:  SYSBP by SEX
## t = -3.1622, df = 4430.6, p-value = 0.001577
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.377678 -0.792332
## sample estimates:
##   mean in group Male mean in group Female
##             131.7369             133.8219
```

```
#we can extract the BPs using the tapply function
SBP.sex = tapply(dat$SYSBP,dat$SEX,c)
t.test(SBP.sex$Male,SBP.sex$Female)
```

```
##
##  Welch Two Sample t-test
##
## data:  SBP.sex$Male and SBP.sex$Female
## t = -3.1622, df = 4430.6, p-value = 0.001577
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.377678 -0.792332
## sample estimates:
## mean of x mean of y
##  131.7369  133.8219
```

2.1.1 Aside: `tapply()`

This is the first time we've seen an apply function (I think), they'll become incredibly useful later for organizing your analyses.

In this example we used tapply:

`tapply(X, INDEX, FUN = NULL, ..., default = NA, simplify = TRUE)`

`tapply()` is used when we want to split a vector (`X`) by the levels of another vector (`INDEX`), and apply a function (`FUN`) to each of the split values. In this example we used the function `c()` or the concatenate function, as we just wanted to get all the SYSBP values, stratified by SEX.

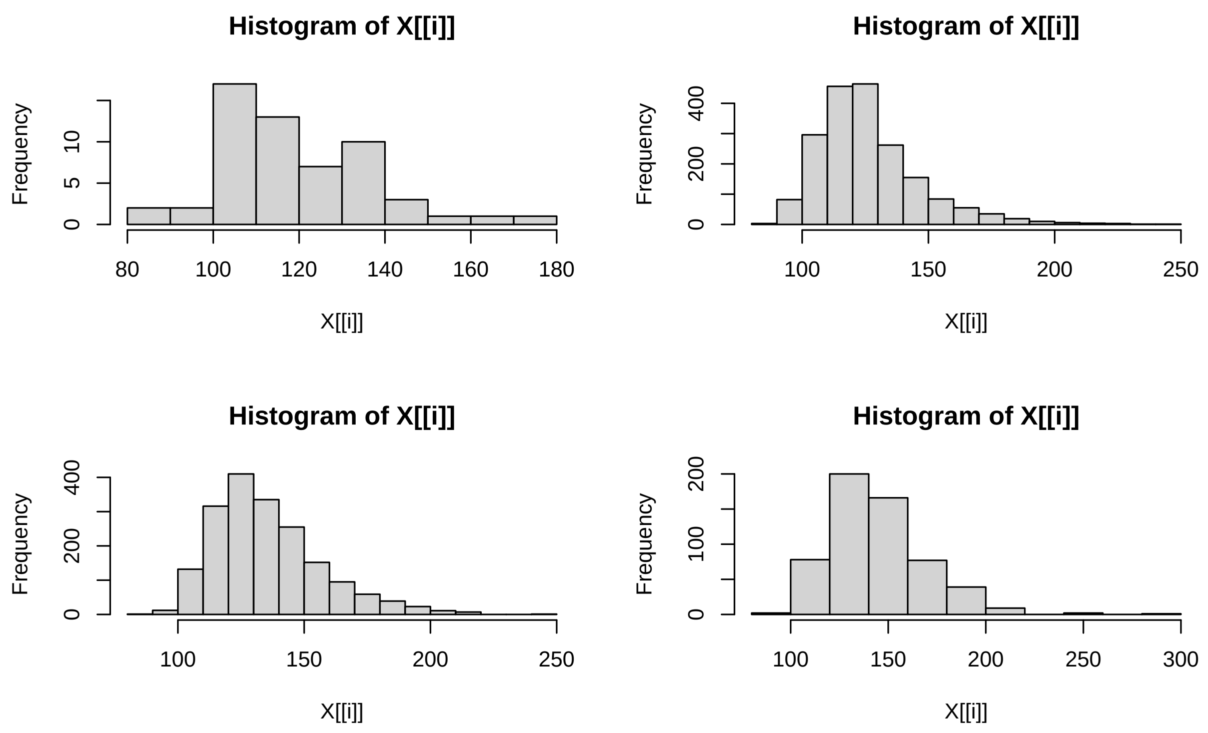
```
#get the average SYSBP for each value of SEX
tapply(dat$SYSBP,dat$SEX,mean)
```

```
##     Male   Female
## 131.7369 133.8219
```

```
#get the average for each BMI group
tapply(dat$SYSBP,dat$BMIGroups,mean)
```

```
## Underweight      Normal  Overweight       Obese
##    119.8070    126.7903    135.8837    145.1490
```

```
#the functions don't need to be numeric
par(mfrow=c(2,2))
temp = tapply(dat$SYSBP,dat$BMIGroups,hist)#this doesn't work well for titles
```



We'll see other apply functions later in the course: `lapply`, `sapply`, `apply` are all valuable in certain situations.

2.2 Chi-square tests

For testing categorical data we use the function `chisq.test()`, or `fisher.test()` if you need the non-parametric version. The typical use case will be to create a 2x2 (or RxC) table using the command `table()`, and then perform the test on the table. The functions will take the vectors themselves, but you tend to want the table anyway.

```
tab01 = table(dat$SEX,dat$HYPERTEN)
chi01 = chisq.test(tab01,correct=FALSE)#I think this matches STATA
chi01 = chisq.test(tab01)#with the Yate's correction
chi01
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  tab01
## X-squared = 2.1207, df = 1, p-value = 0.1453
```

```
names(chi01)
```

```
## [1] "statistic" "parameter" "p.value"   "method"    "data.name" "observed"
## [7] "expected"  "residuals" "stdres"
```

```
chi01$statistic
```

```
## X-squared
##  2.120693
```

```
chi01$p.value
```

```
## [1] 0.1453208
```

```
#this also works, but no table
chisq.test(dat$SEX,dat$HYPERTEN)
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  dat$SEX and dat$HYPERTEN
## X-squared = 2.1207, df = 1, p-value = 0.1453
```

Unlike with `t.test()` there's no effect measure here - the chi-square test doesn't calculate one, so you'll have to get the OR/RR/RD yourself. Fisher's test is nice in that regard - it produces an OR, even though it doesn't use it in the calculation.

```
fisher.test(tab01)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  tab01
## p-value = 0.1411
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.9660691 1.2684553
## sample estimates:
## odds ratio
##    1.10709
```

```
fish01 = fisher.test(tab01)
names(fish01)
```

```
## [1] "p.value"     "conf.int"    "estimate"    "null.value"  "alternative"
## [6] "method"      "data.name"
```

```
fish01$estimate
```

```
## odds ratio
##    1.10709
```

```
fish01$conf.int
```

```
## [1] 0.9660691 1.2684553
## attr(,"conf.level")
## [1] 0.95
```

2.2.1 Getting Effect Measures from Tables

Getting RR/OR/RD isn't in base-R - you can build a function yourself, or you can use the `epiR` library.

```
library(epiR)
test01 = epi.2by2(tab01)
test01
```

```
##                 Outcome +    Outcome -      Total        Inc risk *        Odds
## Exposed +             540         1404       1944              27.8       0.385
## Exposed -             642         1848       2490              25.8       0.347
## Total                1182         3252       4434              26.7       0.363
##
## Point estimates and 95% CIs:
## -------------------------------------------------------------------
## Inc risk ratio                               1.08 (0.98, 1.19)
## Odds ratio                                   1.11 (0.97, 1.27)
## Attrib risk *                                1.99 (-0.64, 4.62)
## Attrib risk in population *                  0.87 (-1.28, 3.03)
## Attrib fraction in exposed (%)               7.18 (-2.36, 15.83)
## Attrib fraction in population (%)            3.28 (-1.14, 7.51)
## -------------------------------------------------------------------
##  Test that OR = 1: chi2(1) = 2.222 Pr>chi2 = 0.14
##  Wald confidence limits
##  CI: confidence interval
##  * Outcomes per 100 population units
```

```
names(test01)
```

```
## [1] "method"     "n.strata"   "conf.level" "res"        "massoc"
## [6] "tab"
```

```
test01$res$RR.crude.wald
```

```
##        est     lower    upper
## 1 1.077362 0.9769162 1.188136
```

I find the function a bit overkill, but it definitely can do everything you need (see the help file for more details, ?epi.2by2).

The one contingency is that it assumes the table has a very specific arrangement - it assumes that the rows are exposed +/-, and the columns are outcomes +/-. While this is a logical arrangement, it is usually the opposite order in R, which tends to sort the rows/columns low-to-high, which means they're usually -/+. You should make sure to get your table in the correct order before you submit it.

```
tab01#wrong arrangement of columns
```

```
##
##           Normotensive Hypertensive
##   Male             540         1404
##   Female           642         1848
```

```
tab01 = tab01[,2:1]#flip the columns
tab01
```

```
##
##           Hypertensive Normotensive
##   Male            1404          540
##   Female          1848          642
```

```
epi.2by2(tab01)
```

```
##                 Outcome +    Outcome -      Total        Inc risk *        Odds
## Exposed +            1404          540       1944              72.2        2.60
## Exposed -            1848          642       2490              74.2        2.88
## Total                3252         1182       4434              73.3        2.75
##
```

```
## Point estimates and 95% CIs:
## -------------------------------------------------------------------
## Inc risk ratio                                 0.97 (0.94, 1.01)
## Odds ratio                                     0.90 (0.79, 1.03)
## Attrib risk *                                  -1.99 (-4.62, 0.64)
## Attrib risk in population *                    -0.87 (-3.03, 1.28)
## Attrib fraction in exposed (%)                 -2.76 (-6.53, 0.87)
## Attrib fraction in population (%)              -1.19 (-2.78, 0.37)
## -------------------------------------------------------------------
##  Test that OR = 1: chi2(1) = 2.222 Pr>chi2 = 0.14
##  Wald confidence limits
##  CI: confidence interval
##  * Outcomes per 100 population units
```

3 Regression

The two main functions we'll look at for regression are `lm` for linear regression and `glm` for general linear regression, which includes logistic. There are a couple of other functions that we'll need, but those two are the core.

3.1 Linear Regression

```r
mod.lm01 = lm(SYSBP~DIABP,data=dat)
mod.lm01
```

```
##
## Call:
## lm(formula = SYSBP ~ DIABP, data = dat)
##
## Coefficients:
## (Intercept)        DIABP
##      11.733        1.458
```

```r
summary(mod.lm01)
```

```
##
## Call:
## lm(formula = SYSBP ~ DIABP, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -40.017  -9.035  -2.083   6.590  89.383
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 11.73340    1.45526   8.063 9.52e-16 ***
## DIABP        1.45846    0.01733  84.138  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 13.91 on 4432 degrees of freedom
## Multiple R-squared:  0.615,  Adjusted R-squared:  0.6149
## F-statistic:  7079 on 1 and 4432 DF,  p-value: < 2.2e-16
```

This builds a linear regression model predicting systolic BP with diastolic BP. The model returns just the coefficients, while the `summary()` returns a detailed description, including coefficient testing.

Both the `lm` object and the `summary` object have components that we might want to access:

```
names(mod.lm01)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
mod.lm01$coeff
```

```
## (Intercept)       DIABP
##   11.733395    1.458464
```

```
sum = summary(mod.lm01)
sum$coefficients
```

```
##              Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 11.733395 1.45526146  8.06274 9.515288e-16
## DIABP        1.458464 0.01733413 84.13825 0.000000e+00
```

```
sum$r.squared
```

```
## [1] 0.6149852
```

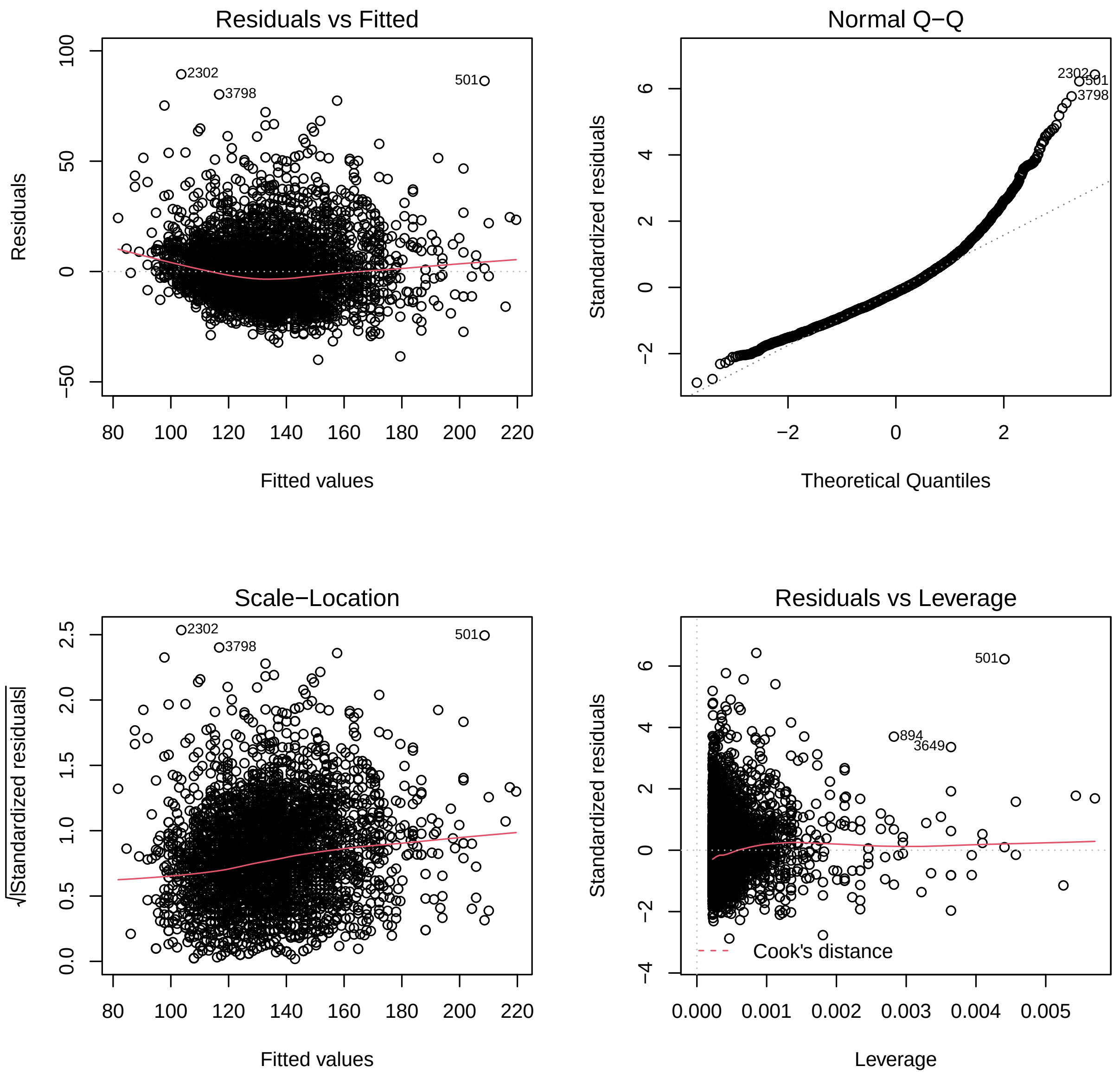
```
sum$fstatistic
```

```
##    value    numdf    dendf
## 7079.246    1.000 4432.000
```

You can extract almost everything you need from either the model or the summary object - the coefficient table from the summary object is the most useful.

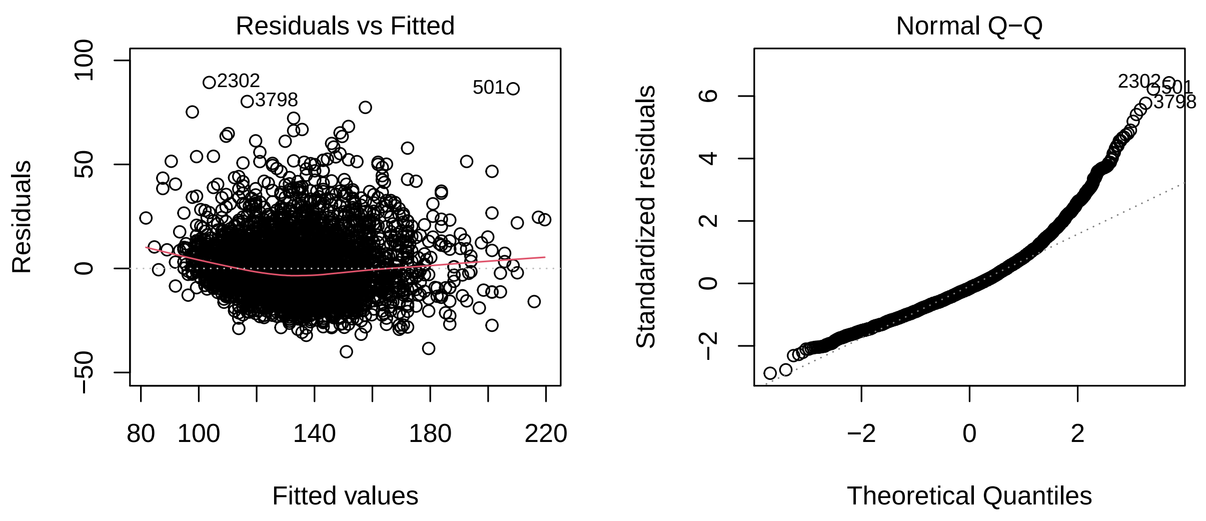
To check our assumptions (which I'm sure you all do religiously and never forget about) we just plot the object - it produces 4 plots, so we'll need to change the plotting space.

```
par(mfrow=c(2,2))
plot(mod.lm01)
```



I only tend to like the first two plots, luckily theres a `which` option to control which plots are produced.

```
par(mfrow=c(1,2))
plot(mod.lm01,which=c(1,2))
```



3.2 Multiple Linear Regression

```
library(car)
mod.lm02 = lm(SYSBP~DIABP+AGE+SEX+BMIGroups+DIABETES+CURSMOKE,data=dat)
car::Anova(mod.lm02)
```

```
## Anova Table (Type II tests)
##
## Response: SYSBP
##            Sum Sq   Df   F value    Pr(>F)
## DIABP     1019149    1 6433.9517 < 2.2e-16 ***
## AGE        118278    1  746.6971 < 2.2e-16 ***
## SEX         14336    1   90.5027 < 2.2e-16 ***
## BMIGroups    1295    3    2.7259   0.04261 *
## DIABETES     5029    1   31.7467 1.866e-08 ***
## CURSMOKE      651    1    4.1075   0.04275 *
## Residuals  697918 4406
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
summary(mod.lm02)
```

```
##
## Call:
## lm(formula = SYSBP ~ DIABP + AGE + SEX + BMIGroups + DIABETES +
##     CURSMOKE, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -35.007  -8.267  -1.320   6.570  81.479
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -18.04142    2.35572  -7.659 2.30e-14 ***
## DIABP                 1.36012    0.01696  80.212  < 2e-16 ***
## AGE                   0.62577    0.02290  27.326  < 2e-16 ***
## SEXFemale             3.78328    0.39768   9.513  < 2e-16 ***
## BMIGroupsNormal       3.65528    1.69434   2.157   0.0310 *
## BMIGroupsOverweight   4.28533    1.70702   2.510   0.0121 *
## BMIGroupsObese        4.46576    1.77033   2.523   0.0117 *
## DIABETESDiabetes      6.64729    1.17976   5.634 1.87e-08 ***
## CURSMOKE              0.81443    0.40185   2.027   0.0428 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.59 on 4406 degrees of freedom
##   (19 observations deleted due to missingness)
## Multiple R-squared:  0.6838, Adjusted R-squared:  0.6832
## F-statistic:  1191 on 8 and 4406 DF,  p-value: < 2.2e-16
```

For multiple regression the formula connects the predictors with `+`. We can get summaries the same way, but if we want to perform ANOVA tests on the individual factors we use the `Anova()` function from the `car` library.

Don't use the `anova()` command from the `base` library - it performs sequential sum of squares, which no-one ever wants. There are many in the R community that consider this a bug, but it's been this way for too long, and R values backward compatibility, so it will never be fixed. I always preface my Anova analyses with the `car::` library specification to make sure I never make a mistake.

```r
confint(mod.lm02)
```

```
##                            2.5 %      97.5 %
## (Intercept)         -22.65981445 -13.4230345
## DIABP                 1.32687470   1.3933615
## AGE                   0.58086970   0.6706615
## SEXFemale             3.00362125   4.5629408
## BMIGroupsNormal       0.33352388   6.9770400
## BMIGroupsOverweight   0.93870522   7.6319589
## BMIGroupsObese        0.99502622   7.9364928
## DIABETESDiabetes      4.33435938   8.9602216
## CURSMOKE              0.02660488   1.6022540
```

The command `conf.int` will produce the confidence intervals on the estimates - you could pull the table yourself from `summary(mod.lm02)$coefficients` and calculate the CI using the values there, but this function is more efficient and less prone to errors.

3.3 Post-hoc comparisons

I won't be teaching anything about the anova analysis itself - there is a command called `aov()` to perform Anova's, but as everyone here knows an Anova is just a linear regression, so I would stick to the `lm()` command.

The one place that `aov()` is needed is post-hoc analyses - the functions for Tukey and Bonferroni post-hoc analyses require an `aov()` object, so we'll have to convert.

```r
mod.lm03 = lm(SYSBP~BMIGroups,data=dat)
#first convert to aov
aov03 = aov(mod.lm03)
#TukeyHSD performs the post-hoc CIs using the Tukey Honest Significant Difference
TukeyHSD(aov03)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = mod.lm03)
##
## $BMIGroups
##                               diff        lwr      upr     p adj
## Normal-Underweight      6.983272 -0.4119735 14.37852 0.0722426
## Overweight-Underweight 16.076640  8.6763611 23.47692 0.0000002
## Obese-Underweight      25.341937 17.6998790 32.98400 0.0000000
## Overweight-Normal       9.093369  7.3037483 10.88299 0.0000000
## Obese-Normal           18.358665 15.7433910 20.97394 0.0000000
## Obese-Overweight        9.265297  6.6358209 11.89477 0.0000000
```

There is a more complete library called `multcomp` if you need to complex post-hoc comparisons - I find the syntax confusing, but if you have complex linear-combinations of coefficients for testing then this library can help.

```
library(multcomp)
CIs = glht(mod.lm03,linfct=mcp(BMIGroups='Tukey'))
summary(CIs,test=adjusted(type='bonferroni'))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: lm(formula = SYSBP ~ BMIGroups, data = dat)
##
## Linear Hypotheses:
##                               Estimate Std. Error t value Pr(>|t|)
## Normal - Underweight == 0       6.9833     2.8775   2.427   0.0916 .
## Overweight - Underweight == 0  16.0766     2.8795   5.583  1.5e-07 ***
## Obese - Underweight == 0       25.3419     2.9736   8.522  < 2e-16 ***
## Overweight - Normal == 0        9.0934     0.6963  13.059  < 2e-16 ***
## Obese - Normal == 0            18.3587     1.0176  18.041  < 2e-16 ***
## Obese - Overweight == 0         9.2653     1.0231   9.056  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- bonferroni method)
```

3.4 Logistic Regressions

Logistic regression will work the same way, the only difference is that we're using a generalized function, so we need to specify that we're working with a binary outcome. To do this we specify the `family=binomial` argument. **Don't forget this argument** - the default is to perform a linear regression, so if you pass a 1/0 variable and don't set the `family` value you get a result that looks correct but is very wrong.

```
mod.log01 = glm(CURSMOKE~GLUCOSE,data=dat,family=binomial)
#THIS IS THE WRONG MODEL SPEC
mod.log01.wrong = glm(CURSMOKE~GLUCOSE,data=dat)
summary(mod.log01)
```

```
##
## Call:
## glm(formula = CURSMOKE ~ GLUCOSE, family = binomial, data = dat)
```

```
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.252  -1.165  -1.040   1.186   1.862
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.382856   0.121807   3.143 0.001671 **
## GLUCOSE     -0.005220   0.001441  -3.623 0.000292 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5594.4  on 4036  degrees of freedom
## Residual deviance: 5579.9  on 4035  degrees of freedom
##   (397 observations deleted due to missingness)
## AIC: 5583.9
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(mod.log01.wrong)
```

```
##
## Call:
## glm(formula = CURSMOKE ~ GLUCOSE, data = dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.5396  -0.4926  -0.4227   0.5050   0.8557
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.587769   0.027605  21.292  < 2e-16 ***
## GLUCOSE     -0.001205   0.000322  -3.742 0.000185 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.2491321)
##
##     Null deviance: 1008.7  on 4036  degrees of freedom
## Residual deviance: 1005.2  on 4035  degrees of freedom
##   (397 observations deleted due to missingness)
## AIC: 5850
##
## Number of Fisher Scoring iterations: 2
```

There's nothing in the output or the model results to clue you into the model being wrong. If your outcome is a factor then it will break, another good argument for using factors. When predicting a factor variable it will take the first level as the control and the second level as the event.

```r
mod.log02 = glm(DIABETES~GLUCOSE,data=dat,family=binomial)
#This will throw an error since DIABETES is a factor variable
#mod.log02 = glm(DIABETES~GLUCOSE,data=dat)
mod.log02
```

```
##
## Call:  glm(formula = DIABETES ~ GLUCOSE, family = binomial, data = dat)
##
## Coefficients:
## (Intercept)      GLUCOSE
##   -11.42035      0.08059
##
## Degrees of Freedom: 4036 Total (i.e. Null);  4035 Residual
##   (397 observations deleted due to missingness)
## Null Deviance:       1052
## Residual Deviance: 503.1     AIC: 507.1
```

```
summary(mod.log02)
```

```
##
## Call:
## glm(formula = DIABETES ~ GLUCOSE, family = binomial, data = dat)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9396  -0.1435  -0.1041  -0.0786   3.9072
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -11.420353   0.610118  -18.72   <2e-16 ***
## GLUCOSE       0.080589   0.005732   14.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1052.16  on 4036  degrees of freedom
## Residual deviance:  503.12  on 4035  degrees of freedom
##   (397 observations deleted due to missingness)
## AIC: 507.12
##
## Number of Fisher Scoring iterations: 8
```

R doesn't naturally produce Odds Ratios, so you'll have to do it yourself.

```
mod.log03 = glm(CURSMOKE~SEX+SYSBP+GLUCOSE+BMIGroups,data=dat,family=binomial)
#extract the coefficients
coef = mod.log03$coefficients
#get the CIs on the coefficients
ci = confint(mod.log03)
#tie them together in a table
out = cbind(OR=coef,ci)
#exponentiate them to get the ORs
out01 = exp(out)

#or in one line
out02 = exp(cbind(OR=mod.log03$coefficients,confint(mod.log03)))
```

This is a place where self-written functions can be useful - I've written a command called `modelOR()` that takes a model and produces the confidence interval itself. A brief version called `getOR()` is given below:

```r
getOR = function(mod){
  out = exp(cbind(OR=mod$coefficients,confint(mod)))
  return(out)
}
out03 = getOR(mod.log03)
```

Home made functions are useful if you're running the same code over and over - in the `function()` you specify the arguments you need, and then between the braces you operate on those arguments. As the full `modelOR()` function below shows at the end of this code, they can get rather complex. Feel free to keep that `modelOR()` function for your own use, I've been using it for years and I *think* all the kinks are worked out.

4 Breakout Exercise

We covered 5 major testing commands today (along with several other helper commands)

- t-test
- chisq.test
- fisher.test
- lm
- glm

Using those commands test the following hypotheses (in whatever manner you see fit):

1. Is there a difference in glucose levels between smokers and non-smokers?
2. Is the difference still present after controlling for sex?
3. Is there an interaction between smoking and sex?
4. Is there an effect of smoking on diabetes status?
5. Is the effect of smoking on diabetes present after controlling for sex?

5 `modelOR()` Code

```r
modelOR = function(model,alpha=0.05,pvalue=FALSE){
  z = qnorm(1-alpha/2,0,1)
  lev = model$xlevels
  lab = names(model$model)[-1]
  dat = model$data
  lreg.coeffs <- coef(summary(model))
  k=2
  rowNames = matrix(rep(vector(length=dim(lreg.coeffs)[1]),4),ncol=4)
  rowNames[1,] = c('(Intercept)','','','')
  for(i in 1:length(lab)){
    if(lab[i]%in%names(lev)){
      # categorical variable
      for(j in 2:length(lev[[(1:length(lev))[names(lev)==lab[i]]]])){
        rowNames[k,] = c(paste(lab[i],":"),lev[[(1:length(lev))[names(lev)==lab[i]]]][j],'vs.',lev[[(1:
        k=k+1
      }
    }
    else{
      rowNames[k,] = c(paste(lab[i],':'),'1','unit','increase')
      k=k+1
    }
  }
  lci <- exp(lreg.coeffs[ ,1] - z * lreg.coeffs[ ,2])
  or <- exp(lreg.coeffs[ ,1])
  uci <- exp(lreg.coeffs[ ,1] + z * lreg.coeffs[ ,2])
  lreg.or <- data.frame(cbind(lci, or, uci))
  orNames = vector(length=dim(rowNames)[1])
  for(i in 1:dim(lreg.or)[1]){
    if(lreg.or[i,2] < 1){
      # invert the odds ratio
      tempLci = 1/lreg.or[i,3]
      tempOR = 1/lreg.or[i,2]
      tempUci = 1/lreg.or[i,1]
      lreg.or[i,] = c(tempLci,tempOR,tempUci)
      # fix the row names
      if(rowNames[i,3]=='vs.'){
        temp = rowNames[i,2]
        rowNames[i,2] = rowNames[i,4]
        rowNames[i,4] = temp
      }
      else{
        rowNames[i,4] = 'decrease'
      }
    }
    orNames[i] = paste(rowNames[i,1],rowNames[i,2],rowNames[i,3],rowNames[i,4])
  }
  names(lreg.or) = c("Lower CI","OR","Upper CI")
  if(pvalue){
    lreg.or[,4] = (summary(model))$coefficients[,4]
    names(lreg.or)[4] = "p-value"
  }
  #The function doesn't handle interaction models well, even though I usually only want to first order
  if(sum(duplicated(orNames))>0){
    orNames[which(duplicated(orNames))] = LETTERS[1:length(which(duplicated(orNames)))]
  }
  row.names(lreg.or) = orNames
  if(pvalue){

  }
  return(lreg.or)
}
```