Introduction to R for statistics Part 2.

Reading data, graphing and linear models

George Savva (QIB)

January 2020

In this example we will:

* create a dataset using data stored in three separate Excel sheets
* create new variables based on existing variables
* identify and remove some obvious outliers and mistakes
* create bar charts, box plots and scatter plots of the data using the ggplot2 package
* estimate and diagnose linear models using the lm() function

# Loading data from Excel, merging, cleaning and coding.

First I made a new project in RStudio, and made sure that the data file for this exercise (walkingdata2.xlsx) was in the root directory of the project.

Then I could just refer to the data by its filename and R would know where to look.

Do this now!

## Loading data

Our data for this example is in three separate sheets in the same workbook, but it needs some fixing before analysis.

Rather than change the data in the workbook we keep the raw data the way it is and do all of our processing in R.

We’ll use the read\_excel() function in the readxl package to load the data. If you didn’t install readxl on day 1 you might need to do it now.

Look at the help file for the read\_excel() function to understand what the options do, what other options exist and what their default values are. Then we are ready to load the data.

Note I always specify sheet and range when using read\_excel() even though it may not be necessary, because this is safer and makes my code more readable.

library(readxl)  
treated <- read\_excel(path="walkingspeed2.xlsx",   
 sheet="treated",   
 range="A1:B68")  
  
control <- read\_excel(path="walkingspeed2.xlsx",   
 sheet="control",   
 range="A1:B70")  
  
meta <- read\_excel(path="walkingspeed2.xlsx",   
 sheet="meta",   
 range="A1:D139")

Keep in mind the goal here: We want to end up with one tidy dataframe that includes all the outcomes and meta data; that has one row per participant and one column per variable.

## Appending data using rbind()

So we’ll need to rbind the data from the treated and control patients together, then merge this with the meta data.

First we want to ‘rbind’ the rows from the treated and control patients so that they are in the same data frame. But becuase the column names are not exactly the same this won’t work:

# This gives an error  
outcomes <- rbind(treated, control)

To look at the names of a data frame, use the ‘names()’ function.

names(treated)

## [1] "patid" "time"

names(control)

## [1] "patid" "walktime"

The names() function returns a character vector. So by replacing it with another character vector we can replace the names in our data frames. We *could* do this:

names(treated) <- c("patid", "time")  
names(control) <- names(treated)

But we need to be sure we have the order right. Here it’s probably fine because we only have two columns, but for a more complex dataset it would be easy to get the column order wrong. It’s safer to rename the specific column that we need to rename:

# How does this command work?  
names(control)[names(control)=="walktime"] <- "time"

Make sure you understand how this command worked, and check that it did what you expected!

names(control)

## [1] "patid" "time"

*Now* we can rbind the two frames together:

outcomes <- rbind(treated, control, make.row.names=TRUE)  
str(outcomes)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 136 obs. of 2 variables:  
## $ patid: num 1 3 5 7 9 11 13 15 17 19 ...  
## $ time : num 1.9 2.93 2.2 2.14 1.72 ...

## Adding a new variable

Now, although we have all our data in a single frame, we can no longer tell how each patient was treated! We should have added a variable to each data frame before rbinding to say what the treatment was.

Lets make a new column called ‘treatment’ that will have two levels, ‘treated’ and ‘control’. You add a new variable to a dataframe as follows:

treated$treated = "Treated"

Now lets add the same column to the control file, and create our outcomes data frame again.

control$treated = "Control"  
outcomes <- rbind(treated, control)

We can check that our new data frame has the treated variable and that it looks correct:

head(outcomes)

## # A tibble: 6 x 3  
## patid time treated  
## <dbl> <dbl> <chr>   
## 1 1 1.90 Treated  
## 2 3 2.93 Treated  
## 3 5 2.20 Treated  
## 4 7 2.14 Treated  
## 5 9 1.72 Treated  
## 6 11 2.15 Treated

str(outcomes)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 136 obs. of 3 variables:  
## $ patid : num 1 3 5 7 9 11 13 15 17 19 ...  
## $ time : num 1.9 2.93 2.2 2.14 1.72 ...  
## $ treated: chr "Treated" "Treated" "Treated" "Treated" ...

table(outcomes$treated)

##   
## Control Treated   
## 69 67

## Adding new columns using merge()

Now we have our outcomes data, but we can’t analyse it against patient characteristics because that information is in a different data frame.

We need to add columns to our data frame for the patient meta-data. This is in the ‘meta’ data frame that we created earlier. Note that the outcomes and meta data frames are linked by the patient id number, but the columns have different names in each sheet.

We could rename the columns of meta as we did above, or we can tell the ‘merge’ function that we want to merge dataframes on a key that is named differently in each parent set. The former approach seems better to me:

names(meta)[1] <- "patid"  
  
# Look at the help for the merge command to find out how this works:  
alldata <- merge(outcomes, meta, by="patid", all=TRUE)

Now look at the ‘alldata’ data frame to check it has everything we need ready for analysis.

Next we’ll go on to plotting and modelling with this dataset.

# 2. Plotting using ggplot2, transforming and fixing outliers

Following on from loading and reshaping data, we’ll create some plots using ggplot2 to do some descriptive analysis.

First we’ll load the libraries that we are going to need. You might need to install them if they are not already installed on your system:

library(ggplot2)  
library(ggpubr)

## Warning: package 'ggpubr' was built under R version 3.6.2

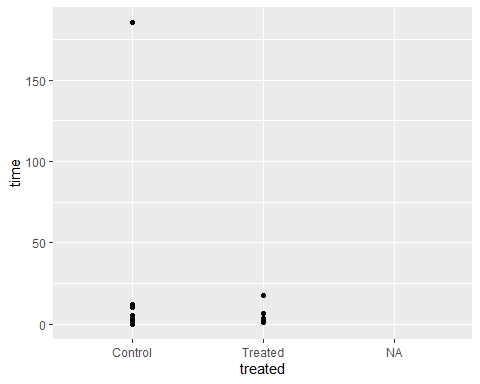
## Loading required package: magrittr

library(ggbeeswarm)

## Warning: package 'ggbeeswarm' was built under R version 3.6.2

First make a simple scatter plot:

ggplot(alldata, aes(x=treated, y=time)) + geom\_point()



Note the missing catgory! We should investigate these, and remove any rows with missing data if appropriate. There are lots of ways we could do this!

ggplot2 comes with the ‘remove\_missing’ function, that takes away any row with any missing data

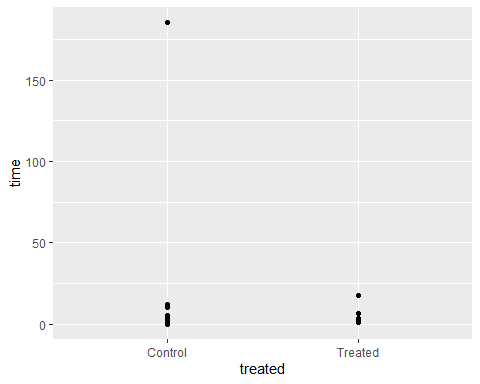
ggplot(remove\_missing(alldata), aes(x=treated, y=time)) + geom\_point()

Or we could run ‘remove\_missing’ first to create a new dataset without these rows, then pass that to ggplot.

alldata2 <- remove\_missing(alldata)

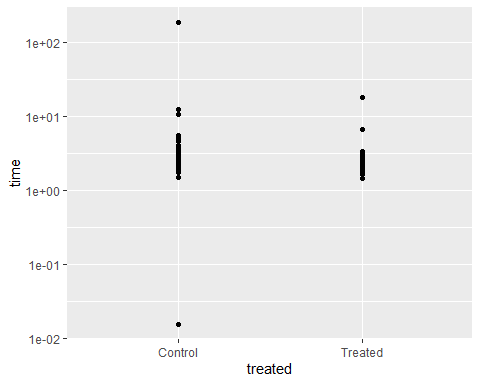
Another method is to use the base function subset(). Subset() returns the rows of a dataset that meet the condition specified in its second argument. Here we will ask for the rows of ‘alldata’ where ‘treated’ is not NA, and where time is not NA.

alldata <- subset(alldata, !is.na(treated) & !is.na(time))  
ggplot(alldata, aes(x=treated, y=time)) + geom\_point()



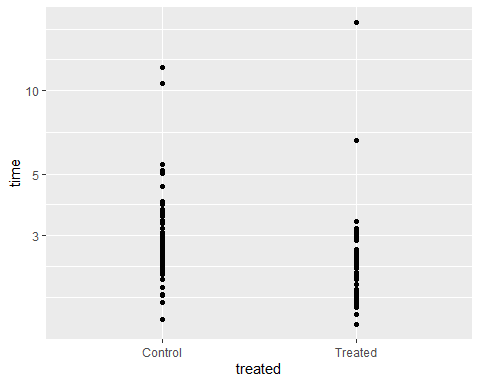
The plot above suggests an outlier. We should use our knowledge of the experiment and our research question to decide what do with this point. But it is often the case that what seems like an outlier on one scale might not look like an outlier on another. Switching to a log-scale for the y-axis we now see a second possible problem observation:

ggplot(alldata, aes(x=treated, y=time)) + geom\_point() + scale\_y\_log10()



Our experiment is a walking time experiment. It is not feasible that anybody completed this task in less than a second, or to take more than a couple of minutes, so we can safely remove these observations by setting their values to NA.

alldata$time[alldata$time>100] <- NA  
alldata$time[alldata$time<0.1] <- NA  
ggplot(alldata, aes(x=treated, y=time)) + geom\_point() + scale\_y\_log10()

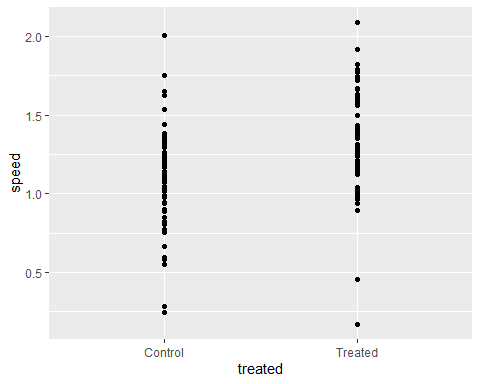


Even on a log scale some of the data points look unusual. Another possible transformation is the inverse transformation. This would turn our ‘time’ variable into a ‘speed’ variable.

Our ‘time’ variable corresponds to the time taken to complete a 3m walk, so to get the speed over the walk we’ll divide 3 by the time taken.

We can make a new column in our dataframe, then plot this instead:

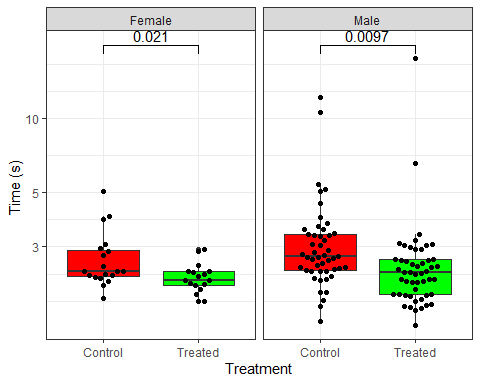
# What does this do?  
alldata$speed <- 3 / alldata$time  
  
ggplot(alldata, aes(x=treated, y=speed)) + geom\_point()



## Embellishing the graph with ggplot2

See the associated powerpoint to understand the development of this plot!

ggplot(alldata, aes(x=treated, y=time, fill=treated)) +  
 geom\_boxplot() +  
 geom\_beeswarm(cex=3) +  
 facet\_wrap(~sex, labeller=labeller(sex=c(M="Male",F="Female"))) +  
 scale\_fill\_manual(values=c("red","green")) +  
 scale\_y\_log10() +  
 stat\_compare\_means(method="t.test", comparisons = list(c("Control","Treated"))) +  
 labs(x="Treatment", y="Time (s) ", fill="Treatment") +  
 theme\_bw() + theme(legend.position = "none")

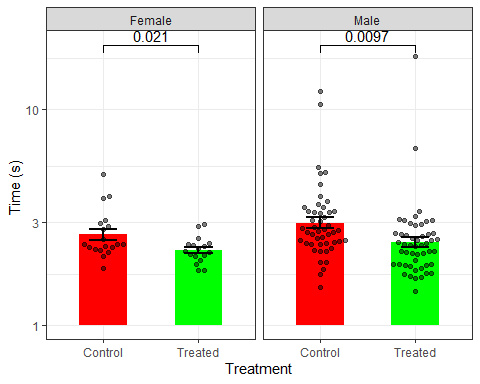


## Plotting summary statistics

You might want to plot only summary statistics, or place summaries along with points on the same graph.

You can do this with ‘stat\_summary()’ as follows:

ggplot(alldata, aes(x=treated, y=time, fill=treated)) +  
 stat\_summary( geom="bar", fun.y=mean,width=0.5) +  
 stat\_summary( geom="errorbar", fun.data=mean\_se,width=0.3 ,size=1) +   
 geom\_beeswarm(cex=3,alpha=0.5) +  
 facet\_wrap(~sex, labeller=labeller(sex=c(M="Male",F="Female"))) +  
 scale\_fill\_manual(values=c("red","green")) +  
 scale\_y\_log10() +  
 stat\_compare\_means(method="t.test", comparisons = list(c("Control","Treated"))) +  
 labs(x="Treatment", y="Time (s) ", fill="Treatment") +  
 theme\_bw() + theme(legend.position = "none")



# 3. Linear Modelling using lm()

Linear modelling is the basis of modern statistical inference.

Most statistical procedures can be framed as linear models.

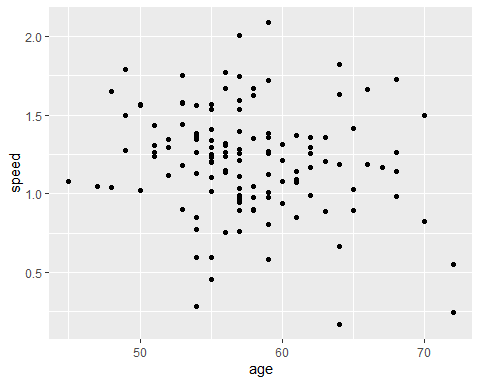
The R function that estimates linear models is lm(). To use this function you need to supply (i) a dataset, and (ii) a model expressed as a formula.

Most of R’s modelling commands can be thought of as extensions of lm(), so it is worth learning this way to do analysis. Here we’ll see how it replaces the t-test, ANOVA etc with linear regression modelling. If you want to learn Bayesian statistics this is also an essential starting point.

Suppose we want to test the relationships we were plotting in the graphing exericise above. First we would like to understand the relationship between time taken to complete our walking speed task, the age of the participant, and whether they were treated.

First lets plot the relationship between walking speed and age:

ggplot(data = alldata, aes(x=age, y=speed)) + geom\_point()



The command below estimates a simple linear model testing the relationship between walking speed and age.

lm( data=alldata, formula = speed ~ age )

##   
## Call:  
## lm(formula = speed ~ age, data = alldata)  
##   
## Coefficients:  
## (Intercept) age   
## 1.899 -0.012

This has estimated the model and returned the model coefficients, but doesn’t show us very much other useful output. To get the full output we can save the resulting object which contains all of the information we want.

model1 <- lm( data=alldata, formula = speed ~ age )

What class is model1?

We could see the structure of the object with str(). But it is better to use specific functions to extract the information we want rather than to try to get the information out the object directly.

Eg, if we run the ‘summary’ function on an object of class ‘lm’, we see a more traditional regression model output

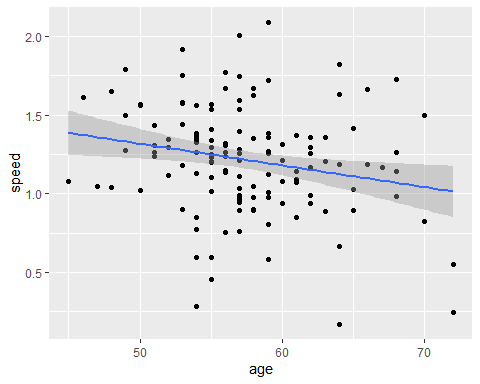
model1 <- lm( data=alldata, formula = speed ~ age )  
summary(model1)

##   
## Call:  
## lm(formula = speed ~ age, data = alldata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.9682 -0.2055 0.0147 0.1801 0.8992   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.899208 0.312333 6.081 1.21e-08 \*\*\*  
## age -0.011995 0.005405 -2.219 0.0282 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.328 on 132 degrees of freedom  
## (2 observations deleted due to missingness)  
## Multiple R-squared: 0.03597, Adjusted R-squared: 0.02866   
## F-statistic: 4.925 on 1 and 132 DF, p-value: 0.02818

**Question:** How do you interpret these model coefficients? Does walking speed depend on age? By how much?

We can add a similar model to our ggplot to look the fit and get a rough indication of how appropriate it is likely to be:

ggplot(data = alldata, aes(x=age, y=speed)) + geom\_point() + stat\_smooth(method="lm")



## Regression diagnostics

There are certain assumptions for a linear regression model like this to be valid. These are:

1. The error variance is normally distributed
2. The error variance is equal for all points
3. There is a linear relationship between our variables to begin with!
4. The errors are independent of each other

The first three conditions can be tested from the data, but the fourth cannot, we need to understand how the data arose to understand this.

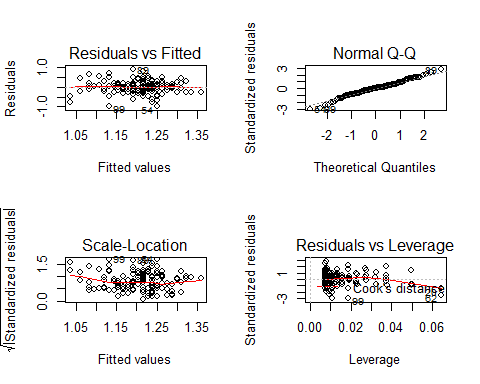
*If the errors are not independent, (for example if more than one point came from the same patient or biological replicate, or from the same block in block randomised study) then we would need to build a more complex model, but this is beyond the scope of todays work.*

Often a visual inspection of the model in this way is enough to satisfy us that conditions 1-3 are met. We might also be interested in whether any one individual has high *leverage* over the regression line, which can happen if one has a predictor variable that is a long way from all the others.

R can also generate diagnostic plots from the model to help us judge whether the diagnostic conditions are met.

Base R will plot these diagnostics if a lm object is passed to the ‘plot’ function:

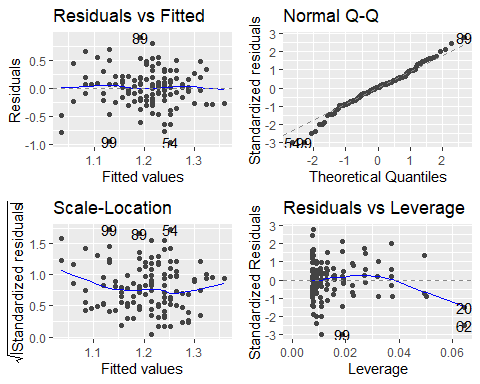
par(mfrow=c(2,2))  
plot(model1)



But there is a new package called ggfortify that enables lots of different types of object to be directly plotted using ggplot:

library(ggfortify)

autoplot(model1)



I have no concerns about my model from a validity point of view.

## Categorical predictors and interaction terms

Now lets test whether speed depends on treatment.

We can do this using a linear model in exactly the same way as we did above:

model2 <- lm(data=alldata, speed ~ treated)  
summary(model2)

##   
## Call:  
## lm(formula = speed ~ treated, data = alldata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.14741 -0.18711 0.00576 0.20296 0.90771   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.10042 0.03856 28.54 < 2e-16 \*\*\*  
## treatedTreated 0.21704 0.05453 3.98 0.000113 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.3156 on 132 degrees of freedom  
## (2 observations deleted due to missingness)  
## Multiple R-squared: 0.1072, Adjusted R-squared: 0.1004   
## F-statistic: 15.84 on 1 and 132 DF, p-value: 0.000113

Does treatment improve walking speed? By how much?

Note: This is exactly the same calculation as an unpaired t-test. If you can use lm() there is no need to ever run the t.test() command. In my view its easier to use lm because the interface and the output is cleaner. (Having said that, there is no easy way to do the Welsh’s T-test for unequal variances using lm().)

## Factor variables

In the last example we entered a ‘character’ object (treated) into our regression model. R realised this and treated it as if the different levels represented categories of a categorical variable. This was the correct thing to do in this case.

If we had a numerical variable reprenting categories we would have to explictly change the type of variable to be ‘factor’, otherwise the levels would be treated as if they represented quantities.

For example, our ‘department’ variable was read as a numeric, even though it represents unordered categories. It would not make sense to fit:

model2 <- lm(data=alldata, speed ~ department)  
summary(model2)

##   
## Call:  
## lm(formula = speed ~ department, data = alldata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -1.01194 -0.16968 0.01625 0.17640 0.90875   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.08941 0.07386 14.749 <2e-16 \*\*\*  
## department 0.04629 0.02639 1.755 0.0817 .   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.3302 on 132 degrees of freedom  
## (2 observations deleted due to missingness)  
## Multiple R-squared: 0.02279, Adjusted R-squared: 0.01539   
## F-statistic: 3.078 on 1 and 132 DF, p-value: 0.08166

Because this suggests that each increment in the ‘department’ variable increases walking speed by some amount. Which is clearly nonsense!

We need to convert department to a factor before we can enter it into the model. We can use the ‘factor()’ function for this.

alldata$departmentF <- factor(alldata$department)  
str(alldata)

## 'data.frame': 136 obs. of 8 variables:  
## $ patid : num 1 2 3 4 5 6 7 8 9 10 ...  
## $ time : num 1.9 3.54 2.93 1.82 2.2 ...  
## $ treated : chr "Treated" "Control" "Treated" "Control" ...  
## $ sex : chr "M" "M" "M" "M" ...  
## $ age : num 53 61 65 48 62 62 57 57 57 55 ...  
## $ department : num 3 3 1 2 2 4 2 4 3 2 ...  
## $ speed : num 1.581 0.848 1.025 1.649 1.361 ...  
## $ departmentF: Factor w/ 4 levels "1","2","3","4": 3 3 1 2 2 4 2 4 3 2...

Now lm() will realise that ‘departmentF’ is a factor and will treat it appropriately:

model2 <- lm(data=alldata, speed ~ departmentF)  
summary(model2)

##   
## Call:  
## lm(formula = speed ~ departmentF, data = alldata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.96084 -0.16065 0.01791 0.18318 0.98574   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.15951 0.06090 19.040 <2e-16 \*\*\*  
## departmentF2 -0.05451 0.08473 -0.643 0.521   
## departmentF3 0.13468 0.07958 1.692 0.093 .   
## departmentF4 0.08460 0.08348 1.013 0.313   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.328 on 130 degrees of freedom  
## (2 observations deleted due to missingness)  
## Multiple R-squared: 0.05056, Adjusted R-squared: 0.02864   
## F-statistic: 2.307 on 3 and 130 DF, p-value: 0.07961

Just as the regression on treatment was equivalent to a t-test, note that the F-statistic line at the bottom of the model corresponds to an ANOVA of walking speed across groups. We can see the full ANOVA table using the anova() command on the lm() object.

There are several functions and packages that will allow you to perform post-hoc pairwise tests between the levels. These are:

* pairwise.t.test()
* TukeyHSD()
* emmeans package

You can explore these if it is something you need to do. Emmeans is a useful package to know as it allows you to easily generate contrasts between levels and to estimate marginal means for levels of categorical variables, but I won’t go into this now!

# Note R generates type I ANOVAs. This is important in multivariate regressions with unbalanced designs. See the 'car' package for type II and type III ANOVAs, (if you really want to use ANOVA).  
anova(model2)

## Analysis of Variance Table  
##   
## Response: speed  
## Df Sum Sq Mean Sq F value Pr(>F)   
## departmentF 3 0.7445 0.24817 2.3074 0.07961 .  
## Residuals 130 13.9824 0.10756   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

## Multivariable models

We easily extend our lm() to estimate a multivariable model:

model3 <- lm(data=alldata, speed ~ age + sex + treated + departmentF)  
summary(model3)

##   
## Call:  
## lm(formula = speed ~ age + sex + treated + departmentF, data = alldata)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.96742 -0.14740 -0.00492 0.17228 0.88998   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.789684 0.304875 5.870 3.57e-08 \*\*\*  
## age -0.012658 0.005582 -2.268 0.025032 \*   
## sexM -0.019685 0.065663 -0.300 0.764829   
## treatedTreated 0.213568 0.053577 3.986 0.000113 \*\*\*  
## departmentF2 -0.035994 0.079767 -0.451 0.652592   
## departmentF3 0.126202 0.075693 1.667 0.097921 .   
## departmentF4 0.101425 0.078332 1.295 0.197736   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.3062 on 127 degrees of freedom  
## (2 observations deleted due to missingness)  
## Multiple R-squared: 0.1916, Adjusted R-squared: 0.1534   
## F-statistic: 5.018 on 6 and 127 DF, p-value: 0.0001201

Finally, there are several packages available for making outputs more visually appealing. These include the ‘table1’ package for descriptive tables and ‘sjPlot’ package that includes the ‘tab\_model’ function for displaying regression models.

Although its harder to visualise this complex model graphically, we can still produce the diagnostic graphics that we made earlier with ggfortify. The sjPlot package also includes various functions for visualising linear models.

library(table1)

## Warning: package 'table1' was built under R version 3.6.2

##   
## Attaching package: 'table1'

## The following objects are masked from 'package:base':  
##   
## units, units<-

table1(data=alldata, ~ age + sex + departmentF + speed + time | treated)

|  | **Control (n=69)** | **Treated (n=67)** | **Overall (n=136)** |
| --- | --- | --- | --- |
| **age** |  |  |  |
| Mean (SD) | 57.3 (5.16) | 57.8 (5.36) | 57.6 (5.24) |
| Median [Min, Max] | 57.0 [45.0, 72.0] | 57.0 [48.0, 70.0] | 57.0 [45.0, 72.0] |
| **sex** |  |  |  |
| F | 19 (27.5%) | 17 (25.4%) | 36 (26.5%) |
| M | 50 (72.5%) | 50 (74.6%) | 100 (73.5%) |
| **departmentF** |  |  |  |
| 1 | 16 (23.2%) | 13 (19.4%) | 29 (21.3%) |
| 2 | 16 (23.2%) | 15 (22.4%) | 31 (22.8%) |
| 3 | 17 (24.6%) | 25 (37.3%) | 42 (30.9%) |
| 4 | 20 (29.0%) | 14 (20.9%) | 34 (25.0%) |
| **speed** |  |  |  |
| Mean (SD) | 1.10 (0.320) | 1.32 (0.312) | 1.21 (0.333) |
| Median [Min, Max] | 1.12 [0.247, 2.01] | 1.29 [0.170, 2.09] | 1.24 [0.170, 2.09] |
| Missing | 2 (2.9%) | 0 (0%) | 2 (1.5%) |
| **time** |  |  |  |
| Mean (SD) | 3.12 (1.68) | 2.60 (1.99) | 2.86 (1.85) |
| Median [Min, Max] | 2.69 [1.49, 12.2] | 2.32 [1.43, 17.6] | 2.43 [1.43, 17.6] |
| Missing | 2 (2.9%) | 0 (0%) | 2 (1.5%) |

library(sjPlot)

tab\_model(model3)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **speed** | | |
| *Predictors* | *Estimates* | *CI* | *p* |
| (Intercept) | 1.79 | 1.19 – 2.39 | **<0.001** |
| age | -0.01 | -0.02 – -0.00 | **0.025** |
| sex [M] | -0.02 | -0.15 – 0.11 | 0.765 |
| treated [Treated] | 0.21 | 0.11 – 0.32 | **<0.001** |
| departmentF [2] | -0.04 | -0.19 – 0.12 | 0.653 |
| departmentF [3] | 0.13 | -0.02 – 0.28 | 0.098 |
| departmentF [4] | 0.10 | -0.05 – 0.26 | 0.198 |
| Observations | 134 | | |
| R2 / R2 adjusted | 0.192 / 0.153 | | |