Applications of Machine Learning - 101

Alfonso R. Reyes

2019-05-15

Contents

Chapter 1

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 2

PCA: prcomp vs princomp

http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-ana

2.1 General methods for principal component analysis

There are two general methods to perform PCA in R :

- Spectral decomposition which examines the covariances / correlations between variables
- Singular value decomposition which examines the covariances / correlations between individuals

The function princomp() uses the spectral decomposition approach. The functions prcomp() and PCA()[FactoMineR] use the singular value decomposition (SVD).

2.2 prcomp() and princomp() functions

The simplified format of these 2 functions are :

```
prcomp(x, scale = FALSE)
princomp(x, cor = FALSE, scores = TRUE)
```

1. Arguments for prcomp():
   `x`: a numeric matrix or data frame
   `scale`: a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place

2. Arguments for princomp():
   `x`: a numeric matrix or data frame `cor`: a logical value. If TRUE, the data will be centered and scaled before the analysis `scores`: a logical value. If TRUE, the coordinates on each principal component are calculated

2.3 factoextra

```r
# install.packages("factoextra")
```

```r
library(factoextra)
```

2.4 demo dataset

We'll use the data sets `decathlon2` [in factoextra], which has been already described at: PCA - Data format.

Briefly, it contains:

- Active individuals (rows 1 to 23) and active variables (columns 1 to 10), which are used to perform the principal component analysis
- Supplementary individuals (rows 24 to 27) and supplementary variables (columns 11 to 13), which coordinates will be predicted using the PCA information and parameters obtained with active individuals/variables.

```
library("factoextra")
data(decathlon2)
decathlon2.active <- decathlon2[1:23, 1:10]
head(decathlon2.active[, 1:6])
```

```
#>           X100m Long.jump Shot.put High.jump X400m X110m.hurdle
#> SEBRLE    11.04      7.58    14.83      2.07 49.81        14.69
#> CLAY      10.76      7.40    14.26      1.86 49.37        14.05
#> BERNARD   11.02      7.23    14.25      1.92 48.93        14.99
#> YURKOV    11.34      7.09    15.19      2.10 50.42        15.31
#> ZSIVOCZKY 11.13      7.30    13.48      2.01 48.62        14.17
#> McMULLEN  10.83      7.31    13.76      2.13 49.91        14.38
```

```
decathlon2.supplementary <- decathlon2[24:27, 1:10]
head(decathlon2.supplementary[, 1:6])
```

```
#>         X100m Long.jump Shot.put High.jump X400m X110m.hurdle
#> KARPOV  11.02      7.30    14.77      2.04 48.37        14.09
#> WARNERS 11.11      7.60    14.31      1.98 48.68        14.23
#> Nool    10.80      7.53    14.26      1.88 48.81        14.80
#> Drews   10.87      7.38    13.07      1.88 48.51        14.01
```

2.5 Compute PCA in R using prcomp()

In this section we'll provide an easy-to-use R code to compute and visualize PCA in R using the prcomp() function and the factoextra package.

‘. Load factoextra for visualization

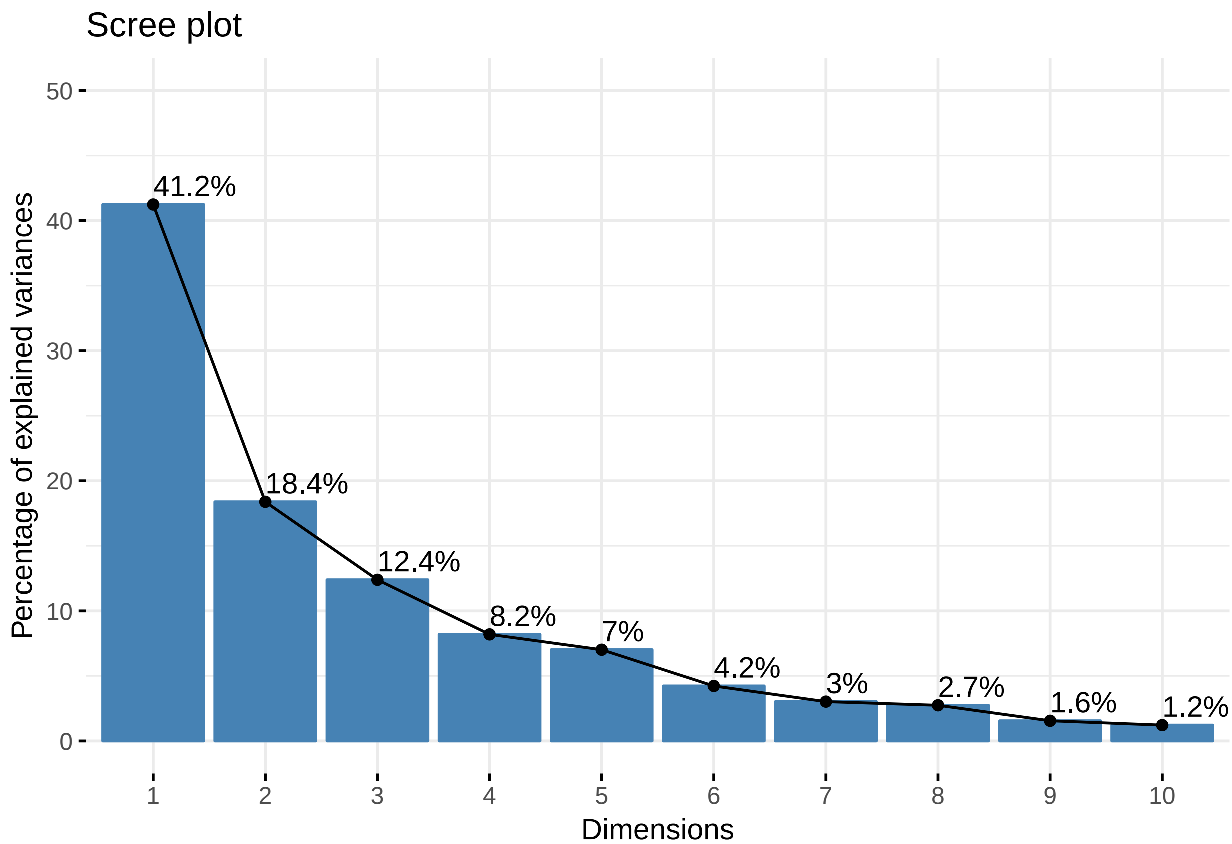
```
library(factoextra)
```

2. compute PCA

```
# compute PCA
res.pca <- prcomp(decathlon2.active, scale = TRUE)
```

3. Visualize eigenvalues (scree plot). Show the percentage of variances explained by each principal component.

```
# Visualize eigenvalues (scree plot).
fviz_eig(res.pca, addlabels = TRUE, ylim = c(0, 50))
```

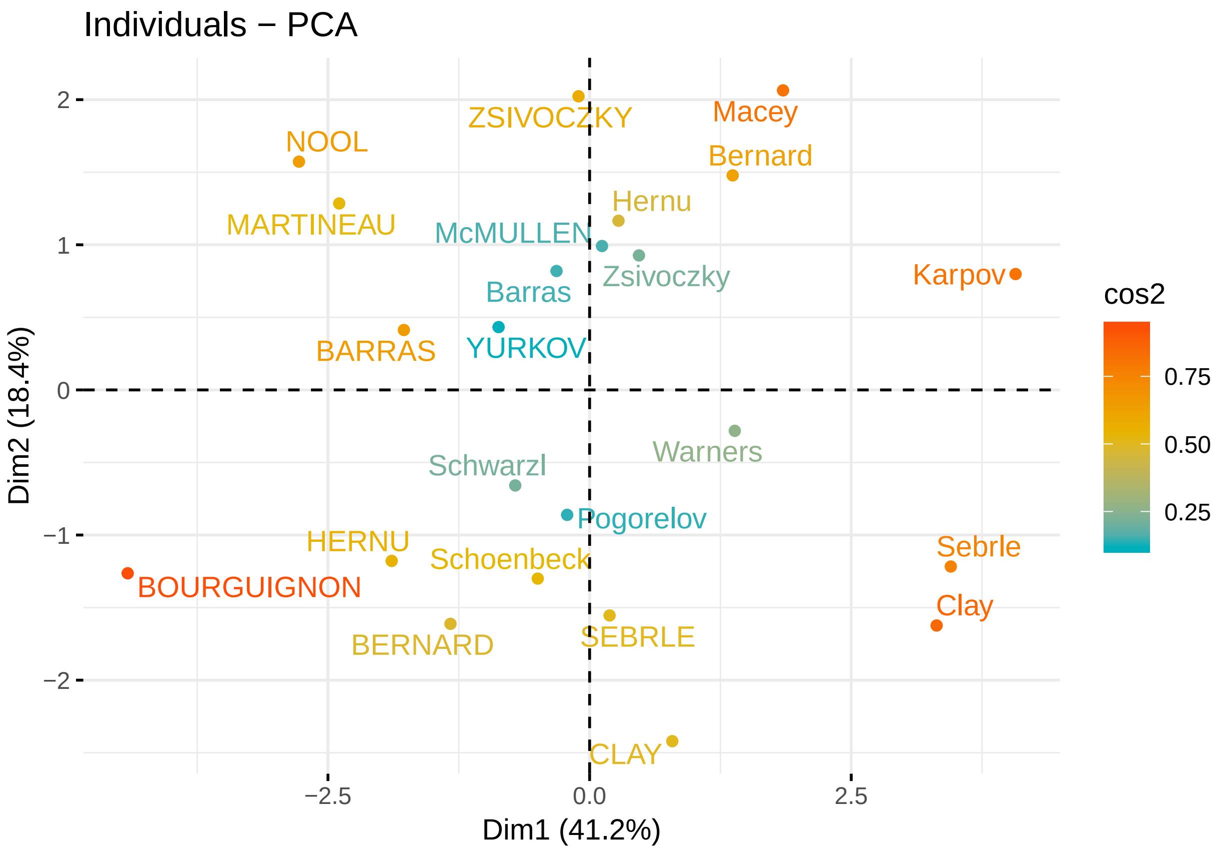


From the plot above, we might want to stop at the fifth principal component. 87% of the information (variances) contained in the data are retained by the first five principal components.

2.6 Plots: quality and contribution

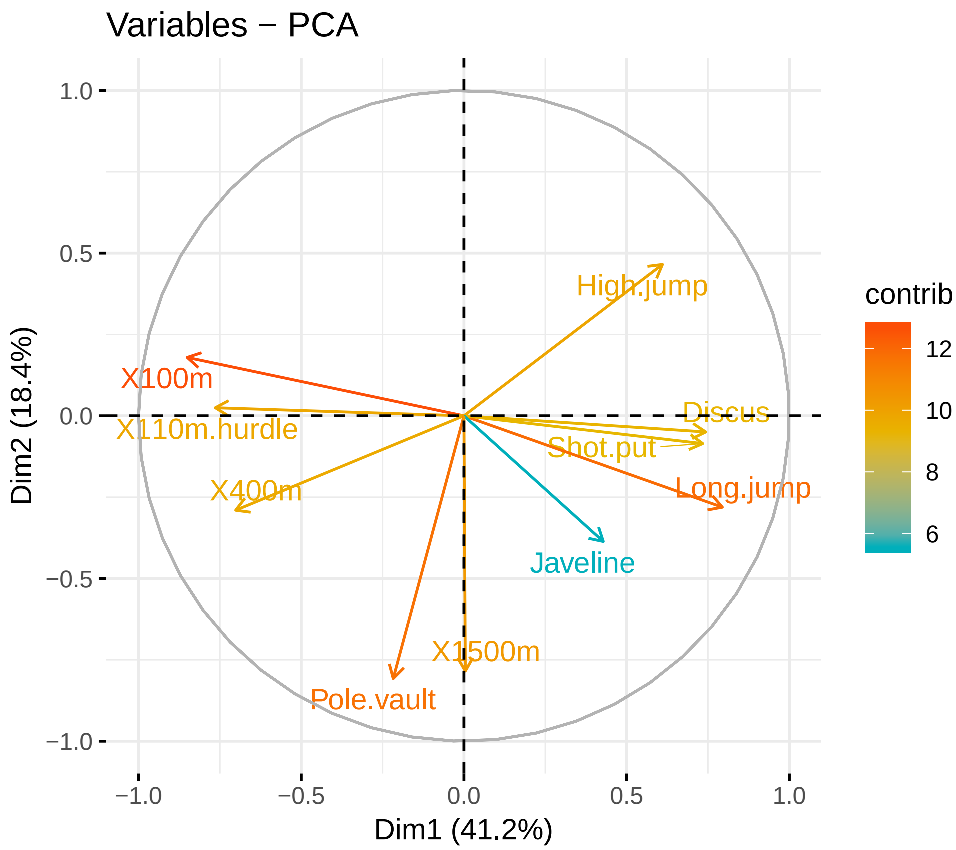
4. Graph of individuals. Individuals with a similar profile are grouped together.

```
# Graph of individuals.
fviz_pca_ind(res.pca,
             col.ind = "cos2", # Color by the quality of representation
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
             )
```



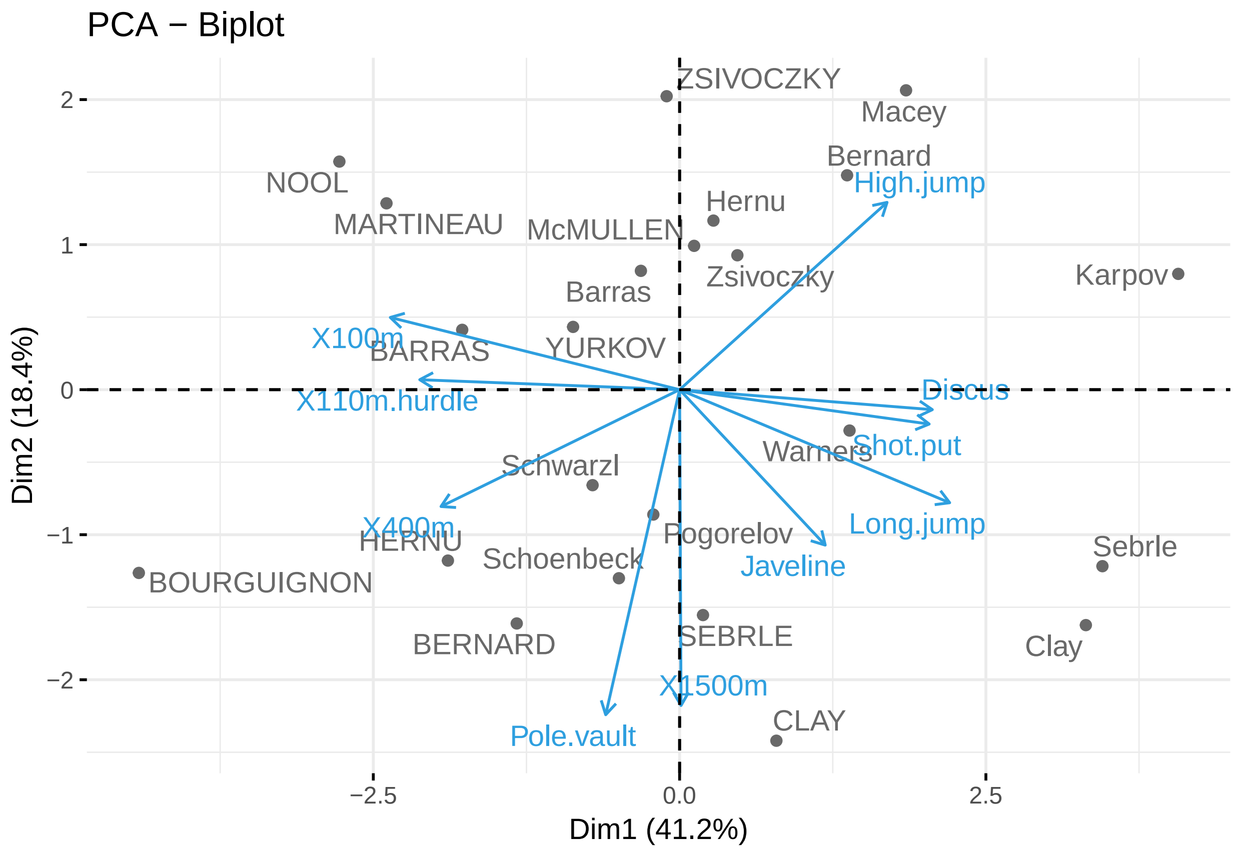
5. Graph of variables. Positive correlated variables point to the same side of the plot. Negative correlated variables point to opposite sides of the graph.

```
# Graph of variables.
fviz_pca_var(res.pca,
             col.var = "contrib", # Color by contributions to the PC
             gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
             repel = TRUE     # Avoid text overlapping
             )
```



6. Biplot of individuals and variables

```
# Biplot of individuals and variables
fviz_pca_biplot(res.pca, repel = TRUE,
                col.var = "#2E9FDF", # Variables color
                col.ind = "#696969"  # Individuals color
                )
```



2.7 Access to the PCA results

```r
library(factoextra)
# Eigenvalues
eig.val <- get_eigenvalue(res.pca)
eig.val
```

```
#>         eigenvalue variance.percent cumulative.variance.percent
#> Dim.1    4.1242133        41.242133                    41.24213
#> Dim.2    1.8385309        18.385309                    59.62744
#> Dim.3    1.2391403        12.391403                    72.01885
#> Dim.4    0.8194402         8.194402                    80.21325
#> Dim.5    0.7015528         7.015528                    87.22878
#> Dim.6    0.4228828         4.228828                    91.45760
#> Dim.7    0.3025817         3.025817                    94.48342
#> Dim.8    0.2744700         2.744700                    97.22812
#> Dim.9    0.1552169         1.552169                    98.78029
#> Dim.10   0.1219710         1.219710                   100.00000
```

```r
# Results for Variables
res.var <- get_pca_var(res.pca)
res.var$coord          # Coordinates
```

```
#>                    Dim.1       Dim.2       Dim.3       Dim.4      Dim.5
#> X100m       -0.850625692  0.17939806 -0.30155643  0.03357320 -0.1944440
```

```
#> Long.jump     0.794180641 -0.28085695  0.19054653 -0.11538956  0.2331567
#> Shot.put      0.733912733 -0.08540412 -0.51759781  0.12846837 -0.2488129
#> High.jump     0.610083985  0.46521415 -0.33008517  0.14455012  0.4027002
#> X400m        -0.701603377 -0.29017826 -0.28353292  0.43082552  0.1039085
#> X110m.hurdle -0.764125197  0.02474081 -0.44888733 -0.01689589  0.2242200
#> Discus        0.743209016 -0.04966086 -0.17652518  0.39500915 -0.4082391
#> Pole.vault   -0.217268042 -0.80745110 -0.09405773 -0.33898477 -0.2216853
#> Javeline      0.428226639 -0.38610928 -0.60412432 -0.33173454  0.1978128
#> X1500m        0.004278487 -0.78448019  0.21947068  0.44800961  0.2632527
#>                     Dim.6        Dim.7        Dim.8       Dim.9
#> X100m         0.035374780 -0.091336386 -0.104716925 -0.30306448
#> Long.jump    -0.033727883 -0.154330810 -0.397380703 -0.05158951
#> Shot.put     -0.239789034 -0.009886612  0.024359049  0.04778655
#> High.jump    -0.284644846  0.028157465  0.084405578 -0.11213822
#> X400m        -0.049289996  0.286106008 -0.233552216  0.08216041
#> X110m.hurdle  0.002632395 -0.370072158 -0.008344682  0.16176025
#> Discus        0.198544870 -0.142725641 -0.039559255  0.01336209
#> Pole.vault   -0.327464549 -0.010393176  0.032914942 -0.02576874
#> Javeline      0.362097598  0.133564318  0.052841099 -0.04045397
#> X1500m        0.042050151 -0.111367083  0.194469730 -0.10224014
#>                    Dim.10
#> X100m         0.044417974
#> Long.jump     0.029719453
#> Shot.put      0.217451948
#> High.jump    -0.133566774
#> X400m        -0.034170673
#> X110m.hurdle -0.015629914
#> Discus       -0.172590426
#> Pole.vault   -0.137211339
#> Javeline     -0.003854347
#> X1500m        0.062834809
```

```
res.var$contrib        # Contributions to the PCs
```

```
#>                     Dim.1      Dim.2      Dim.3       Dim.4     Dim.5
#> X100m        1.754429e+01  1.7505098  7.3386590  0.13755240  5.389252
#> Long.jump    1.529317e+01  4.2904162  2.9300944  1.62485936  7.748815
#> Shot.put     1.306014e+01  0.3967224 21.6204325  2.01407269  8.824401
#> High.jump    9.024811e+00 11.7715838  8.7928883  2.54987951 23.115504
#> X400m        1.193554e+01  4.5799296  6.4876363 22.65090599  1.539012
#> X110m.hurdle 1.415754e+01  0.0332933 16.2612611  0.03483735  7.166193
#> Discus       1.339309e+01  0.1341398  2.5147385 19.04132022 23.755756
#> Pole.vault   1.144592e+00 35.4618611  0.7139512 14.02307063  7.005084
#> Javeline     4.446377e+00  8.1086683 29.4531777 13.42963254  5.577615
#> X1500m       4.438531e-04 33.4728757  3.8871610 24.49386930  9.878367
#>                     Dim.6       Dim.7       Dim.8      Dim.9      Dim.10
#> X100m         0.295915322  2.75705260  3.99520353 59.1740009  1.61756139
#> Long.jump     0.269003613  7.87159392 57.53322220  1.7146826  0.72414393
#> Shot.put     13.596858744  0.03230371  0.21618512  1.4712015 38.76768578
#> High.jump    19.159607001  0.26202607  2.59565787  8.1015517 14.62649091
#> X400m         0.574509906 27.05274658 19.87344405  4.3489667  0.95730504
#> X110m.hurdle  0.001638634 45.26163460  0.02537025 16.8579392  0.20028870
#> Discus        9.321746508  6.73226823  0.57016606  0.1150295 24.42174410
#> Pole.vault   25.357622290  0.03569883  0.39472201  0.4278065 15.43559151
#> Javeline     31.004964393  5.89573984  1.01729950  1.0543458  0.01217993
```

```
#> X1500m        0.418133591  4.09893563 13.77872941  6.7344755  3.23700871
```

```r
res.var$cos2           # Quality of representation
```

```
#>                    Dim.1        Dim.2       Dim.3        Dim.4      Dim.5
#> X100m        7.235641e-01 0.0321836641 0.090936280 0.0011271597 0.03780845
#> Long.jump    6.307229e-01 0.0788806285 0.036307981 0.0133147506 0.05436203
#> Shot.put     5.386279e-01 0.0072938636 0.267907488 0.0165041211 0.06190783
#> High.jump    3.722025e-01 0.2164242070 0.108956221 0.0208947375 0.16216747
#> X400m        4.922473e-01 0.0842034209 0.080390914 0.1856106269 0.01079698
#> X110m.hurdle 5.838873e-01 0.0006121077 0.201499837 0.0002854712 0.05027463
#> Discus       5.523596e-01 0.0024662013 0.031161138 0.1560322304 0.16665918
#> Pole.vault   4.720540e-02 0.6519772763 0.008846856 0.1149106765 0.04914437
#> Javeline     1.833781e-01 0.1490803723 0.364966189 0.1100478063 0.03912992
#> X1500m       1.830545e-05 0.6154091638 0.048167378 0.2007126089 0.06930197
#>                    Dim.6        Dim.7        Dim.8        Dim.9
#> X100m        1.251375e-03 0.0083423353 1.096563e-02 0.0918480768
#> Long.jump    1.137570e-03 0.0238179990 1.579114e-01 0.0026614779
#> Shot.put     5.749878e-02 0.0000977451 5.933633e-04 0.0022835540
#> High.jump    8.102269e-02 0.0007928428 7.124302e-03 0.0125749811
#> X400m        2.429504e-03 0.0818566479 5.454664e-02 0.0067503333
#> X110m.hurdle 6.929502e-06 0.1369534023 6.963371e-05 0.0261663784
#> Discus       3.942007e-02 0.0203706085 1.564935e-03 0.0001785453
#> Pole.vault   1.072330e-01 0.0001080181 1.083393e-03 0.0006640282
#> Javeline     1.311147e-01 0.0178394271 2.792182e-03 0.0016365234
#> X1500m       1.768215e-03 0.0124026272 3.781848e-02 0.0104530472
#>                   Dim.10
#> X100m        1.972956e-03
#> Long.jump    8.832459e-04
#> Shot.put     4.728535e-02
#> High.jump    1.784008e-02
#> X400m        1.167635e-03
#> X110m.hurdle 2.442942e-04
#> Discus       2.978746e-02
#> Pole.vault   1.882695e-02
#> Javeline     1.485599e-05
#> X1500m       3.948213e-03
```

```r
# Results for individuals
res.ind <- get_pca_ind(res.pca)
res.ind$coord          # Coordinates
```

```
#>                  Dim.1      Dim.2       Dim.3       Dim.4         Dim.5
#> SEBRLE       0.1912074 -1.5541282 -0.62836882  0.08205241  1.1426139415
#> CLAY         0.7901217 -2.4204156  1.35688701  1.26984296 -0.8068483724
#> BERNARD     -1.3292592 -1.6118687 -0.19614996 -1.92092203  0.0823428202
#> YURKOV      -0.8694134  0.4328779 -2.47398223  0.69723814  0.3988584116
#> ZSIVOCZKY   -0.1057450  2.0233632  1.30493117 -0.09929630 -0.1970241089
#> McMULLEN     0.1185550  0.9916237  0.84355824  1.31215266  1.5858708644
#> MARTINEAU   -2.3923532  1.2849234 -0.89816842  0.37309771 -2.2433515889
#> HERNU       -1.8910497 -1.1784614 -0.15641037  0.89130068 -0.1267412520
#> BARRAS      -1.7744575  0.4125321  0.65817750  0.22872866 -0.2338366980
#> NOOL        -2.7770058  1.5726757  0.60724821 -1.55548081  1.4241839810
#> BOURGUIGNON -4.4137335 -1.2635770 -0.01003734  0.66675478  0.4191518468
#> Sebrle       3.4514485 -1.2169193 -1.67816711 -0.80870696 -0.0250530746
#> Clay         3.3162243 -1.6232908 -0.61840443 -0.31679906  0.5691645854
```

```
#> Karpov       4.0703560  0.7983510  1.01501662  0.31336354 -0.7974259553
#> Macey        1.8484623  2.0638828 -0.97928455  0.58469073 -0.0002157834
#> Warners      1.3873514 -0.2819083  1.99969621 -1.01959817 -0.0405401497
#> Zsivoczky    0.4715533  0.9267436 -1.72815525 -0.18483138  0.4073029909
#> Hernu        0.2763118  1.1657260  0.17056375 -0.84869401 -0.6894795441
#> Bernard      1.3672590  1.4780354  0.83137913  0.74531557  0.8598016482
#> Schwarzl    -0.7102777 -0.6584251  1.04075176 -0.92717510 -0.2887568007
#> Pogorelov   -0.2143524 -0.8610557  0.29761010  1.35560294 -0.0150531057
#> Schoenbeck  -0.4953166 -1.3000530  0.10300360 -0.24927712 -0.6452257128
#> Barras      -0.3158867  0.8193681 -0.86169481 -0.58935985 -0.7797389436
#>                  Dim.6       Dim.7        Dim.8        Dim.9      Dim.10
#> SEBRLE      -0.46389755 -0.20796012  0.043460568 -0.659344137  0.03273238
#> CLAY         1.30420016 -0.21291866  0.617240611 -0.060125359 -0.31716015
#> BERNARD     -0.40062867 -0.40643754  0.703856040  0.170083313 -0.09908142
#> YURKOV       0.10286344 -0.32487448  0.114996135 -0.109524039 -0.11969720
#> ZSIVOCZKY    0.89554111  0.08825624 -0.202341299 -0.523103099 -0.34842265
#> McMULLEN     0.18657283  0.47828432  0.293089967 -0.105623196 -0.39317797
#> MARTINEAU   -0.45666350 -0.29975522 -0.291628488 -0.223417655 -0.61640509
#> HERNU        0.43623496 -0.56609980 -1.529404317  0.006184409  0.55368016
#> BARRAS       0.09026010  0.21594095  0.682583078 -0.669282042  0.53085420
#> NOOL         0.49716399 -0.53205687 -0.433385655 -0.115777808 -0.09622142
#> BOURGUIGNON -0.08200220 -0.59833739  0.563619921  0.525814030  0.05855882
#> Sebrle      -0.08279306  0.01016177 -0.030585843 -0.847210682  0.21970353
#> Clay         0.77715960  0.25750851 -0.580638301  0.409776590 -0.61601933
#> Karpov      -0.32958134 -1.36365568  0.345306381  0.193055107  0.21721852
#> Macey       -0.19728082 -0.26927772 -0.363219506  0.368260269  0.21249474
#> Warners     -0.55673300 -0.26739400 -0.109470797  0.180283071  0.24208420
#> Zsivoczky   -0.11383190  0.03991159  0.538039776  0.585966156 -0.14271715
#> Hernu       -0.33168404  0.44308686  0.247293566  0.066908586 -0.20868256
#> Bernard     -0.32806564  0.36357920  0.006165316  0.279488675  0.32067773
#> Schwarzl    -0.68891640  0.56568604 -0.687053339 -0.008358849 -0.30211546
#> Pogorelov   -1.59379599  0.78370119 -0.037623661 -0.130531397 -0.03697576
#> Schoenbeck   0.16172381  0.85752368 -0.255850722  0.564222295  0.29680481
#> Barras       1.17415412  0.94512710  0.365550568  0.102255763  0.61186706
```

```
res.ind$contrib        # Contributions to the PCs
```

```
#>                  Dim.1      Dim.2        Dim.3       Dim.4        Dim.5
#> SEBRLE       0.03854254  5.7118249 1.385418e+00  0.03572215 8.091161e+00
#> CLAY         0.65814114 13.8541889 6.460097e+00  8.55568792 4.034555e+00
#> BERNARD      1.86273218  6.1441319 1.349983e-01 19.57827284 4.202070e-02
#> YURKOV       0.79686310  0.4431309 2.147558e+01  2.57939100 9.859373e-01
#> ZSIVOCZKY    0.01178829  9.6816398 5.974848e+00  0.05231437 2.405750e-01
#> McMULLEN     0.01481737  2.3253860 2.496789e+00  9.13531719 1.558646e+01
#> MARTINEAU    6.03367104  3.9044125 2.830527e+00  0.73858431 3.118936e+01
#> HERNU        3.76996156  3.2842176 8.583863e-02  4.21505626 9.955149e-02
#> BARRAS       3.31942012  0.4024544 1.519980e+00  0.27758505 3.388731e-01
#> NOOL         8.12988880  5.8489726 1.293851e+00 12.83761115 1.257025e+01
#> BOURGUIGNON 20.53729577  3.7757623 3.534995e-04  2.35877858 1.088816e+00
#> Sebrle      12.55838616  3.5020697 9.881482e+00  3.47006223 3.889859e-03
#> Clay        11.59361384  6.2315181 1.341828e+00  0.53250375 2.007648e+00
#> Karpov      17.46609555  1.5072627 3.614914e+00  0.52101693 3.940874e+00
#> Macey        3.60207087 10.0732890 3.364879e+00  1.81387486 2.885677e-07
#> Warners      2.02910262  0.1879390 1.403071e+01  5.51585696 1.018550e-02
#> Zsivoczky    0.23441891  2.0310492 1.047894e+01  0.18126182 1.028128e+00
```

```
#> Hernu        0.08048777  3.2136178 1.020764e-01  3.82170515 2.946148e+00
#> Bernard      1.97075488  5.1661961 2.425213e+00  2.94737426 4.581507e+00
#> Schwarzl     0.53184785  1.0252129 3.800546e+00  4.56119277 5.167449e-01
#> Pogorelov    0.04843819  1.7533304 3.107757e-01  9.75034337 1.404313e-03
#> Schoenbeck   0.25864068  3.9969003 3.722687e-02  0.32970059 2.580092e+00
#> Barras       0.10519467  1.5876667 2.605305e+00  1.84296038 3.767994e+00
#>                   Dim.6       Dim.7        Dim.8        Dim.9      Dim.10
#> SEBRLE       2.21256620  0.621426384 2.992045e-02 12.177477305  0.03819185
#> CLAY        17.48801877  0.651413899 6.035125e+00  0.101262442  3.58568943
#> BERNARD      1.65019840  2.373652810 7.847747e+00  0.810319793  0.34994507
#> YURKOV       0.10878629  1.516564073 2.094806e-01  0.336009790  0.51072064
#> ZSIVOCZKY    8.24561722  0.111923276 6.485544e-01  7.664919832  4.32741147
#> McMULLEN     0.35788945  3.287016354 1.360753e+00  0.312501167  5.51053518
#> MARTINEAU    2.14409841  1.291109482 1.347216e+00  1.398195851 13.54402896
#> HERNU        1.95655942  4.604850849 3.705288e+01  0.001071345 10.92781554
#> BARRAS       0.08376135  0.670038259 7.380544e+00 12.547331617 10.04537028
#> NOOL         2.54127369  4.067669683 2.975270e+00  0.375477289  0.33003418
#> BOURGUIGNON  0.06913582  5.144247534 5.032108e+00  7.744571086  0.12223626
#> Sebrle       0.07047579  0.001483775 1.481898e-02 20.105546253  1.72063803
#> Clay         6.20972751  0.952824148 5.340583e+00  4.703566841 13.52708188
#> Karpov       1.11680500 26.720158115 1.888802e+00  1.043988269  1.68193477
#> Macey        0.40014909  1.041910483 2.089853e+00  3.798767930  1.60957713
#> Warners      3.18673563  1.027384225 1.898339e-01  0.910422384  2.08904756
#> Zsivoczky    0.13322327  0.022889042 4.585705e+00  9.617852173  0.72605208
#> Hernu        1.13110069  2.821027418 9.687304e-01  0.125399768  1.55234328
#> Bernard      1.10655655  1.899449022 6.021268e-04  2.188071254  3.66566729
#> Schwarzl     4.87961053  4.598122119 7.477531e+00  0.001957159  3.25357879
#> Pogorelov   26.11665608  8.825322559 2.242329e-02  0.477268755  0.04873597
#> Schoenbeck   0.26890572 10.566272800 1.036933e+00  8.917302863  3.14020004
#> Barras      14.17432302 12.835417603 2.116763e+00  0.292892746 13.34533825
```

```
res.ind$cos2           # Quality of representation
```

```
#>                   Dim.1      Dim.2        Dim.3       Dim.4        Dim.5
#> SEBRLE      0.007530179 0.49747323 8.132523e-02 0.001386688 2.689027e-01
#> CLAY        0.048701249 0.45701660 1.436281e-01 0.125791741 5.078506e-02
#> BERNARD     0.197199804 0.28996555 4.294015e-03 0.411819183 7.567259e-04
#> YURKOV      0.096109800 0.02382571 7.782303e-01 0.061812637 2.022798e-02
#> ZSIVOCZKY   0.001574385 0.57641944 2.397542e-01 0.001388216 5.465497e-03
#> McMULLEN    0.002175437 0.15219499 1.101379e-01 0.266486530 3.892621e-01
#> MARTINEAU   0.404013915 0.11654676 5.694575e-02 0.009826320 3.552552e-01
#> HERNU       0.399282749 0.15506199 2.731529e-03 0.088699901 1.793538e-03
#> BARRAS      0.616241975 0.03330700 8.478249e-02 0.010239088 1.070152e-02
#> NOOL        0.489872515 0.15711146 2.342405e-02 0.153694675 1.288433e-01
#> BOURGUIGNON 0.859698130 0.07045912 4.446015e-06 0.019618511 7.753120e-03
#> Sebrle      0.675380606 0.08395940 1.596674e-01 0.037079012 3.558507e-05
#> Clay        0.687592867 0.16475409 2.391051e-02 0.006274965 2.025440e-02
#> Karpov      0.783666922 0.03014772 4.873187e-02 0.004644764 3.007790e-02
#> Macey       0.363436037 0.45308203 1.020057e-01 0.036362957 4.952707e-09
#> Warners     0.255651956 0.01055582 5.311341e-01 0.138081100 2.182965e-04
#> Zsivoczky   0.045053176 0.17401353 6.051030e-01 0.006921739 3.361236e-02
#> Hernu       0.024824321 0.44184663 9.459148e-03 0.234196727 1.545686e-01
#> Bernard     0.289347476 0.33813318 1.069834e-01 0.085980212 1.144234e-01
#> Schwarzl    0.116721435 0.10030142 2.506043e-01 0.198892209 1.929118e-02
#> Pogorelov   0.007803472 0.12591966 1.504272e-02 0.312101619 3.848427e-05
```

```
#> Schoenbeck  0.067070098 0.46204603 2.900467e-03 0.016987442 1.138116e-01
#> Barras      0.018972684 0.12765099 1.411800e-01 0.066043061 1.156018e-01
#>                   Dim.6        Dim.7        Dim.8        Dim.9
#> SEBRLE      0.0443241299 8.907507e-03 3.890334e-04 8.954067e-02
#> CLAY        0.1326907339 3.536548e-03 2.972084e-02 2.820119e-04
#> BERNARD     0.0179131165 1.843634e-02 5.529104e-02 3.228572e-03
#> YURKOV      0.0013453555 1.341980e-02 1.681440e-03 1.525225e-03
#> ZSIVOCZKY   0.1129176906 1.096685e-03 5.764478e-03 3.852703e-02
#> McMULLEN    0.0053876990 3.540616e-02 1.329562e-02 1.726733e-03
#> MARTINEAU   0.0147210347 6.342774e-03 6.003515e-03 3.523552e-03
#> HERNU       0.0212478795 3.578167e-02 2.611676e-01 4.270425e-06
#> BARRAS      0.0015944528 9.126203e-03 9.118662e-02 8.766746e-02
#> NOOL        0.0157010551 1.798232e-02 1.193105e-02 8.514912e-04
#> BOURGUIGNON 0.0002967459 1.579887e-02 1.401866e-02 1.220108e-02
#> Sebrle      0.0003886276 5.854423e-06 5.303795e-05 4.069384e-02
#> Clay        0.0377627839 4.145976e-03 2.107924e-02 1.049876e-02
#> Karpov      0.0051379747 8.795817e-02 5.639959e-03 1.762907e-03
#> Macey       0.0041397727 7.712721e-03 1.403282e-02 1.442502e-02
#> Warners     0.0411689767 9.496848e-03 1.591742e-03 4.317040e-03
#> Zsivoczky   0.0026253777 3.227467e-04 5.865332e-02 6.956790e-02
#> Hernu       0.0357707217 6.383462e-02 1.988402e-02 1.455601e-03
#> Bernard     0.0166586433 2.046050e-02 5.883405e-06 1.209056e-02
#> Schwarzl    0.1098063093 7.403638e-02 1.092132e-01 1.616543e-05
#> Pogorelov   0.4314162233 1.043115e-01 2.404103e-04 2.893750e-03
#> Schoenbeck  0.0071500829 2.010275e-01 1.789520e-02 8.702893e-02
#> Barras      0.2621297474 1.698426e-01 2.540745e-02 1.988116e-03
#>                   Dim.10
#> SEBRLE      0.0002206741
#> CLAY        0.0078471026
#> BERNARD     0.0010956493
#> YURKOV      0.0018217256
#> ZSIVOCZKY   0.0170924251
#> McMULLEN    0.0239268142
#> MARTINEAU   0.0268211980
#> HERNU       0.0342288717
#> BARRAS      0.0551531863
#> NOOL        0.0005881295
#> BOURGUIGNON 0.0001513277
#> Sebrle      0.0027366539
#> Clay        0.0237264222
#> Karpov      0.0022318265
#> Macey       0.0048028954
#> Warners     0.0077841113
#> Zsivoczky   0.0041268259
#> Hernu       0.0141595965
#> Bernard     0.0159167991
#> Schwarzl    0.0211173850
#> Pogorelov   0.0002322016
#> Schoenbeck  0.0240826922
#> Barras      0.0711836486
```

Chapter 3

Predict using PCA

In this section, we'll show how to predict the coordinates of supplementary individuals and variables using only the information provided by the previously performed PCA.

1. Data: rows 24 to 27 and columns 1 to to 10 [in decathlon2 data sets]. The new data must contain columns (variables) with the same names and in the same order as the active data used to compute PCA.

```
# Data for the supplementary individuals
ind.sup <- decathlon2[24:27, 1:10]
ind.sup[, 1:6]
```

```
#>         X100m Long.jump Shot.put High.jump X400m X110m.hurdle
#> KARPOV  11.02      7.30    14.77      2.04 48.37        14.09
#> WARNERS 11.11      7.60    14.31      1.98 48.68        14.23
#> Nool    10.80      7.53    14.26      1.88 48.81        14.80
#> Drews   10.87      7.38    13.07      1.88 48.51        14.01
```

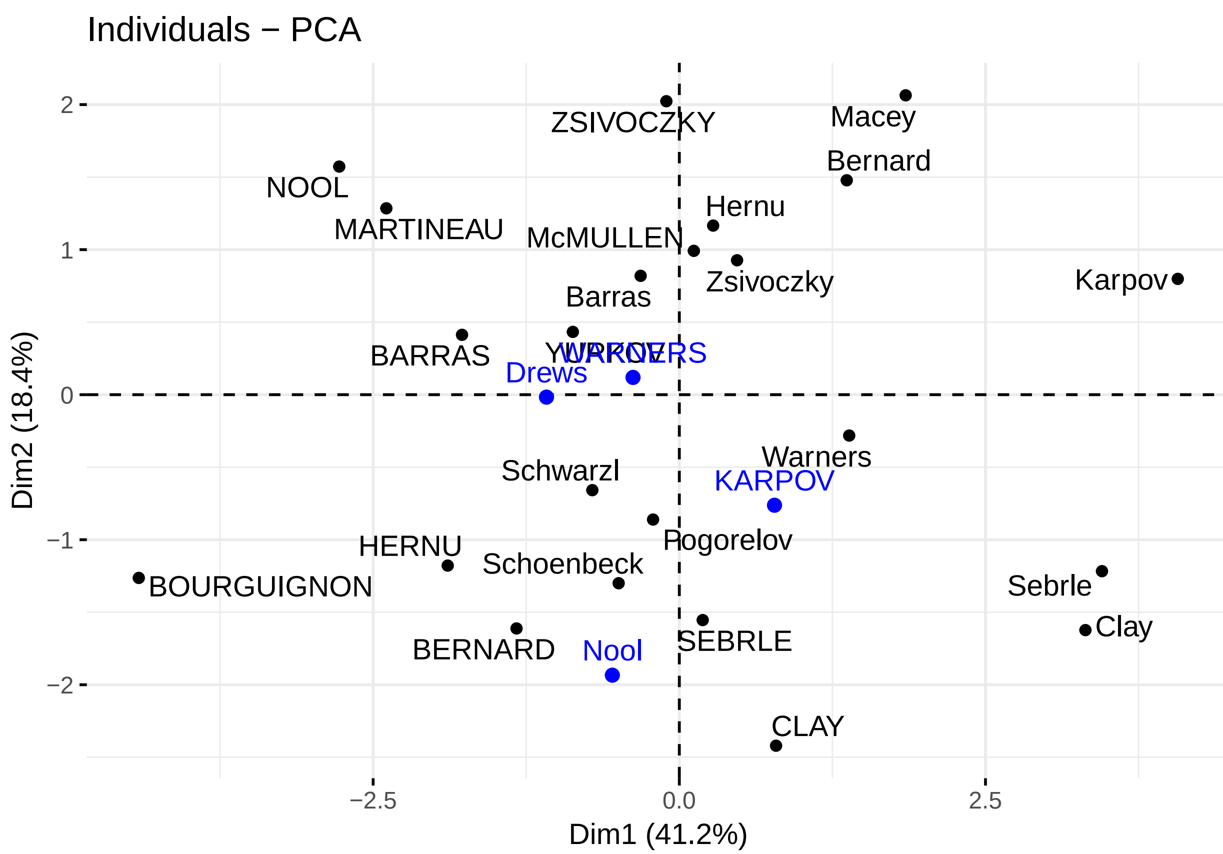
2. Predict the coordinates of new individuals data. Use the R base function predict():

```
ind.sup.coord <- predict(res.pca, newdata = ind.sup)
ind.sup.coord[, 1:4]
```

```
#>                PC1         PC2       PC3        PC4
#> KARPOV   0.7772521 -0.76237804 1.5971253  1.6863286
#> WARNERS -0.3779697  0.11891968 1.7005146 -0.6908084
#> Nool    -0.5468405 -1.93402211 0.4724184 -2.2283706
#> Drews   -1.0848227 -0.01703198 2.9818031 -1.5006207
```

3. Graph of individuals including the supplementary individuals:

```
# Plot of active individuals
p <- fviz_pca_ind(res.pca, repel = TRUE)
# Add supplementary individuals
fviz_add(p, ind.sup.coord, color ="blue")
```



The predicted coordinates of individuals can be manually calculated as follow:

1. Center and scale the new individuals data using the center and the scale of the PCA
2. Calculate the predicted coordinates by multiplying the scaled values with the eigenvectors (loadings) of the principal components. The R code below can be used :

```
# Centering and scaling the supplementary individuals
ind.scaled <- scale(ind.sup,
                    center = res.pca$center,
                    scale = res.pca$scale)
# Coordinates of the individividuals
coord_func <- function(ind, loadings){
  r <- loadings*ind
  apply(r, 2, sum)
}
pca.loadings <- res.pca$rotation
ind.sup.coord <- t(apply(ind.scaled, 1, coord_func, pca.loadings ))
ind.sup.coord[, 1:4]
```

```
#>                PC1         PC2       PC3        PC4
#> KARPOV   0.7772521 -0.76237804 1.5971253  1.6863286
#> WARNERS -0.3779697  0.11891968 1.7005146 -0.6908084
#> Nool    -0.5468405 -1.93402211 0.4724184 -2.2283706
#> Drews   -1.0848227 -0.01703198 2.9818031 -1.5006207
```

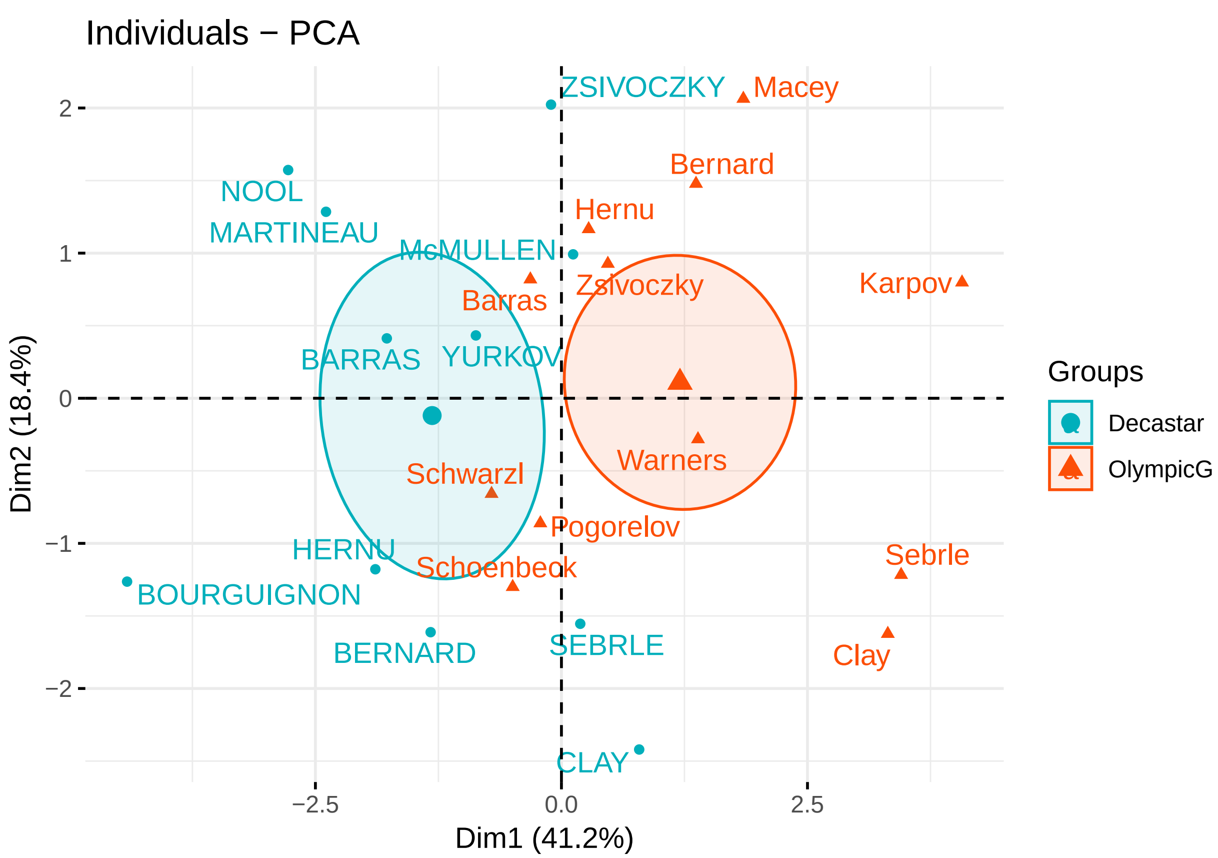
3.1 Supplementary variables

3.1.1 Qualitative / categorical variables

The data sets decathlon2 contain a supplementary qualitative variable at columns 13 corresponding to the type of competitions.

Qualitative / categorical variables can be used to color individuals by groups. The grouping variable should be of same length as the number of active individuals (here 23).

```
groups <- as.factor(decathlon2$Competition[1:23])
fviz_pca_ind(res.pca,
             col.ind = groups, # color by groups
             palette = c("#00AFBB",  "#FC4E07"),
             addEllipses = TRUE, # Concentration ellipses
             ellipse.type = "confidence",
             legend.title = "Groups",
             repel = TRUE
             )
```



Calculate the coordinates for the levels of grouping variables. The coordinates for a given group is calculated as the mean coordinates of the individuals in the group.

```
library(magrittr) # for pipe %>%
library(dplyr)   # everything else

# 1. Individual coordinates
res.ind <- get_pca_ind(res.pca)
```

```r
# 2. Coordinate of groups
coord.groups <- res.ind$coord %>%
  as_data_frame() %>%
  select(Dim.1, Dim.2) %>%
  mutate(competition = groups) %>%
  group_by(competition) %>%
  summarise(
    Dim.1 = mean(Dim.1),
    Dim.2 = mean(Dim.2)
    )
coord.groups
```

```
#> # A tibble: 2 x 3
#>   competition Dim.1  Dim.2
#>   <fct>       <dbl>  <dbl>
#> 1 Decastar    -1.31 -0.119
#> 2 OlympicG     1.20  0.109
```

3.1.2 Quantitative variables

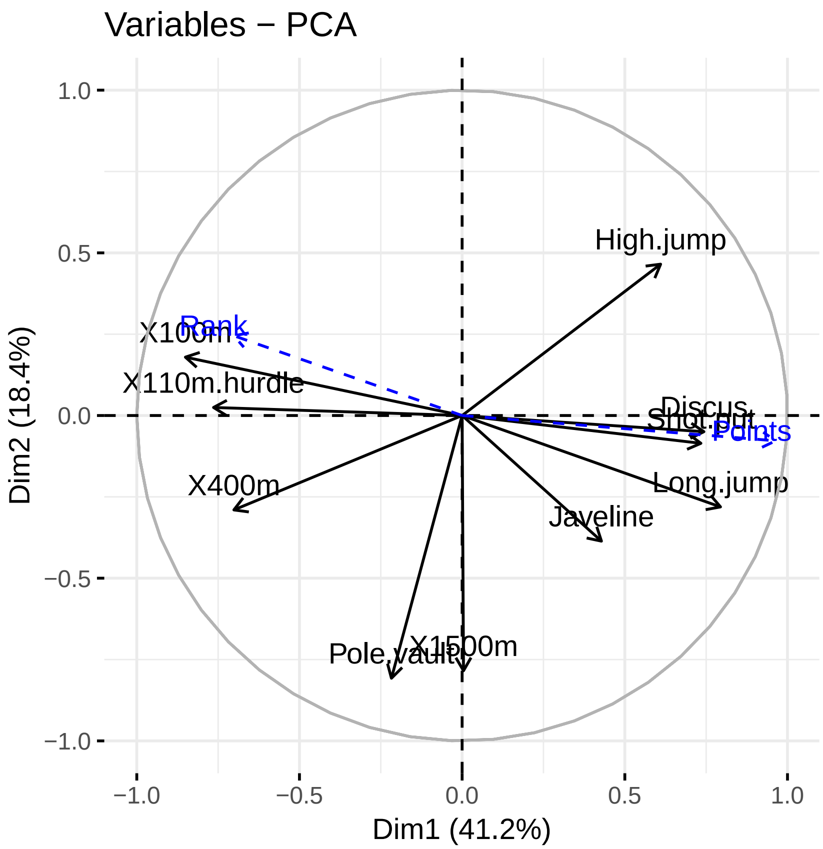
Data: columns 11:12. Should be of same length as the number of active individuals (here 23)

```r
quanti.sup <- decathlon2[1:23, 11:12, drop = FALSE]
head(quanti.sup)
```

```
#>             Rank Points
#> SEBRLE         1   8217
#> CLAY           2   8122
#> BERNARD        4   8067
#> YURKOV         5   8036
#> ZSIVOCZKY      7   8004
#> McMULLEN       8   7995
```

The coordinates of a given quantitative variable are calculated as the correlation between the quantitative variables and the principal components.

```r
# Predict coordinates and compute cos2
quanti.coord <- cor(quanti.sup, res.pca$x)
quanti.cos2 <- quanti.coord^2
# Graph of variables including supplementary variables
p <- fviz_pca_var(res.pca)
fviz_add(p, quanti.coord, color ="blue", geom="arrow")
```



3.2 Theory behind PCA results

3.2.1 PCA results for variables

Here we'll show how to calculate the PCA results for variables: coordinates, cos2 and contributions:

`var.coord` = loadings * the component standard deviations `var.cos2` = var.coord^2 `var.contrib`. The contribution of a variable to a given principal component is (in percentage) : (var.cos2 * 100) / (total cos2 of the component)

```
# Helper function
#::::::::::::::::::::::::::::::::::::::::
var_coord_func <- function(loadings, comp.sdev){
  loadings*comp.sdev
}
```

```
# Compute Coordinates
#::::::::::::::::::::::::::::::::::::::::
loadings <- res.pca$rotation
sdev <- res.pca$sdev
var.coord <- t(apply(loadings, 1, var_coord_func, sdev))
head(var.coord[, 1:4])
```

```
#>                   PC1         PC2        PC3         PC4
#> X100m      -0.8506257  0.17939806 -0.3015564  0.03357320
#> Long.jump   0.7941806 -0.28085695  0.1905465 -0.11538956
```

```
#> Shot.put      0.7339127 -0.08540412 -0.5175978  0.12846837
#> High.jump     0.6100840  0.46521415 -0.3300852  0.14455012
#> X400m        -0.7016034 -0.29017826 -0.2835329  0.43082552
#> X110m.hurdle -0.7641252  0.02474081 -0.4488873 -0.01689589
```

```
# Compute Cos2
#::::::::::::::::::::::::::::::::::::::::
var.cos2 <- var.coord^2
head(var.cos2[, 1:4])
```

```
#>                    PC1          PC2        PC3          PC4
#> X100m        0.7235641 0.0321836641 0.09093628 0.0011271597
#> Long.jump    0.6307229 0.0788806285 0.03630798 0.0133147506
#> Shot.put     0.5386279 0.0072938636 0.26790749 0.0165041211
#> High.jump    0.3722025 0.2164242070 0.10895622 0.0208947375
#> X400m        0.4922473 0.0842034209 0.08039091 0.1856106269
#> X110m.hurdle 0.5838873 0.0006121077 0.20149984 0.0002854712
```

```
# Compute contributions
#::::::::::::::::::::::::::::::::::::::::
comp.cos2 <- apply(var.cos2, 2, sum)
contrib <- function(var.cos2, comp.cos2){var.cos2*100/comp.cos2}
var.contrib <- t(apply(var.cos2,1, contrib, comp.cos2))
head(var.contrib[, 1:4])
```

```
#>                    PC1        PC2       PC3         PC4
#> X100m        17.544293  1.7505098  7.338659  0.13755240
#> Long.jump    15.293168  4.2904162  2.930094  1.62485936
#> Shot.put     13.060137  0.3967224 21.620432  2.01407269
#> High.jump     9.024811 11.7715838  8.792888  2.54987951
#> X400m        11.935544  4.5799296  6.487636 22.65090599
#> X110m.hurdle 14.157544  0.0332933 16.261261  0.03483735
```

3.2.2 PCA results for individuals

- `ind.coord` = res.pca$x
- Cos2 of individuals. Two steps:
  - Calculate the square distance between each individual and the PCA center of gravity: d2 = [(var1_ind_i - mean_var1)/sd_var1]^2 + …+ [(var10_ind_i - mean_var10)/sd_var10]^2 + …+..
  - Calculate the cos2 as ind.coord^2/d2
- Contributions of individuals to the principal components: 100 * (1 / number_of_individuals)*(ind.coord^2 / comp_sdev^2). Note that the sum of all the contributions per column is 100

```
# Coordinates of individuals
#::::::::::::::::::::::::::::::::::
ind.coord <- res.pca$x
head(ind.coord[, 1:4])
```

```
#>                  PC1        PC2        PC3         PC4
#> SEBRLE     0.1912074 -1.5541282 -0.6283688  0.08205241
#> CLAY       0.7901217 -2.4204156  1.3568870  1.26984296
#> BERNARD   -1.3292592 -1.6118687 -0.1961500 -1.92092203
#> YURKOV    -0.8694134  0.4328779 -2.4739822  0.69723814
#> ZSIVOCZKY -0.1057450  2.0233632  1.3049312 -0.09929630
#> McMULLEN   0.1185550  0.9916237  0.8435582  1.31215266
```

```r
# Cos2 of individuals
#:::::::::::::::::::::::::::::::::
# 1. square of the distance between an individual and the
# PCA center of gravity
center <- res.pca$center
scale<- res.pca$scale

getdistance <- function(ind_row, center, scale){
  return(sum(((ind_row-center)/scale)^2))
}

d2 <- apply(decathlon2.active,1, getdistance, center, scale)
# 2. Compute the cos2. The sum of each row is 1
cos2 <- function(ind.coord, d2){return(ind.coord^2/d2)}
ind.cos2 <- apply(ind.coord, 2, cos2, d2)
head(ind.cos2[, 1:4])
```

```
#>                   PC1        PC2         PC3         PC4
#> SEBRLE    0.007530179 0.49747323 0.081325232 0.001386688
#> CLAY      0.048701249 0.45701660 0.143628117 0.125791741
#> BERNARD   0.197199804 0.28996555 0.004294015 0.411819183
#> YURKOV    0.096109800 0.02382571 0.778230322 0.061812637
#> ZSIVOCZKY 0.001574385 0.57641944 0.239754152 0.001388216
#> McMULLEN  0.002175437 0.15219499 0.110137872 0.266486530
```

```r
# Contributions of individuals
#:::::::::::::::::::::::::::::::
contrib <- function(ind.coord, comp.sdev, n.ind){
  100*(1/n.ind)*ind.coord^2/comp.sdev^2
}
ind.contrib <- t(apply(ind.coord, 1, contrib,
                       res.pca$sdev, nrow(ind.coord)))
head(ind.contrib[, 1:4])
```

```
#>                  PC1        PC2        PC3         PC4
#> SEBRLE    0.03854254  5.7118249  1.3854184  0.03572215
#> CLAY      0.65814114 13.8541889  6.4600973  8.55568792
#> BERNARD   1.86273218  6.1441319  0.1349983 19.57827284
#> YURKOV    0.79686310  0.4431309 21.4755770  2.57939100
#> ZSIVOCZKY 0.01178829  9.6816398  5.9748485  0.05231437
#> McMULLEN  0.01481737  2.3253860  2.4967890  9.13531719
```

Chapter 4

Diagnostic Plots

I have tried to use fortify function in ggplot2 which can access different statistics related to linear model. The basic diagnostic plot which we often get using plot function in the fitted model using lm command.

The function `diagPlots` gives an list of six different plots which can be arranged in a grid using `grid` and `gridExtra` packages.

```
library(ggplot2)

diagPlot<-function(model){
    p1<-ggplot(model, aes(.fitted, .resid))+geom_point()
    p1<-p1+stat_smooth(method="loess")+geom_hline(yintercept=0, col="red", linetype="dashed")
    p1<-p1+xlab("Fitted values")+ylab("Residuals")
    p1<-p1+ggtitle("Residual vs Fitted Plot")+theme_bw()

    p2<-ggplot(model, aes(qqnorm(.stdresid)[[1]], .stdresid))+geom_point(na.rm = TRUE)
    p2<-p2+geom_abline(aes(qqline(.stdresid)))+xlab("Theoretical Quantiles")+ylab("Standardized Residua
    p2<-p2+ggtitle("Normal Q-Q")+theme_bw()

    p3<-ggplot(model, aes(.fitted, sqrt(abs(.stdresid))))+geom_point(na.rm=TRUE)
    p3<-p3+stat_smooth(method="loess", na.rm = TRUE)+xlab("Fitted Value")
    p3<-p3+ylab(expression(sqrt("|Standardized residuals|")))
    p3<-p3+ggtitle("Scale-Location")+theme_bw()

    p4<-ggplot(model, aes(seq_along(.cooksd), .cooksd))+geom_bar(stat="identity", position="identity")
    p4<-p4+xlab("Obs. Number")+ylab("Cook's distance")
    p4<-p4+ggtitle("Cook's distance")+theme_bw()

    p5<-ggplot(model, aes(.hat, .stdresid))+geom_point(aes(size=.cooksd), na.rm=TRUE)
    p5<-p5+stat_smooth(method="loess", na.rm=TRUE)
    p5<-p5+xlab("Leverage")+ylab("Standardized Residuals")
    p5<-p5+ggtitle("Residual vs Leverage Plot")
    p5<-p5+scale_size_continuous("Cook's Distance", range=c(1,5))
    p5<-p5+theme_bw()+theme(legend.position="bottom")

    p6<-ggplot(model, aes(.hat, .cooksd))+geom_point(na.rm=TRUE)+stat_smooth(method="loess", na.rm=TRUE
    p6<-p6+xlab("Leverage hii")+ylab("Cook's Distance")
    p6<-p6+ggtitle("Cook's dist vs Leverage hii/(1-hii)")
    p6<-p6+geom_abline(slope=seq(0,3,0.5), color="gray", linetype="dashed")
```

```
    p6<-p6+theme_bw()

    return(list(rvfPlot=p1, qqPlot=p2, sclLocPlot=p3, cdPlot=p4, rvlevPlot=p5, cvlPlot=p6))
}
```

Using the mtcars datasets, a linear model is fitted with mpg as response and cyl, disp, hp, drat and wt has predictor variable

```
lm.model <- lm(mpg ~ cyl+disp+hp+drat+wt, data=mtcars)
diagPlts <- diagPlot(lm.model)
```

To display the plots in a grid, some packages mentioned above should be installed.

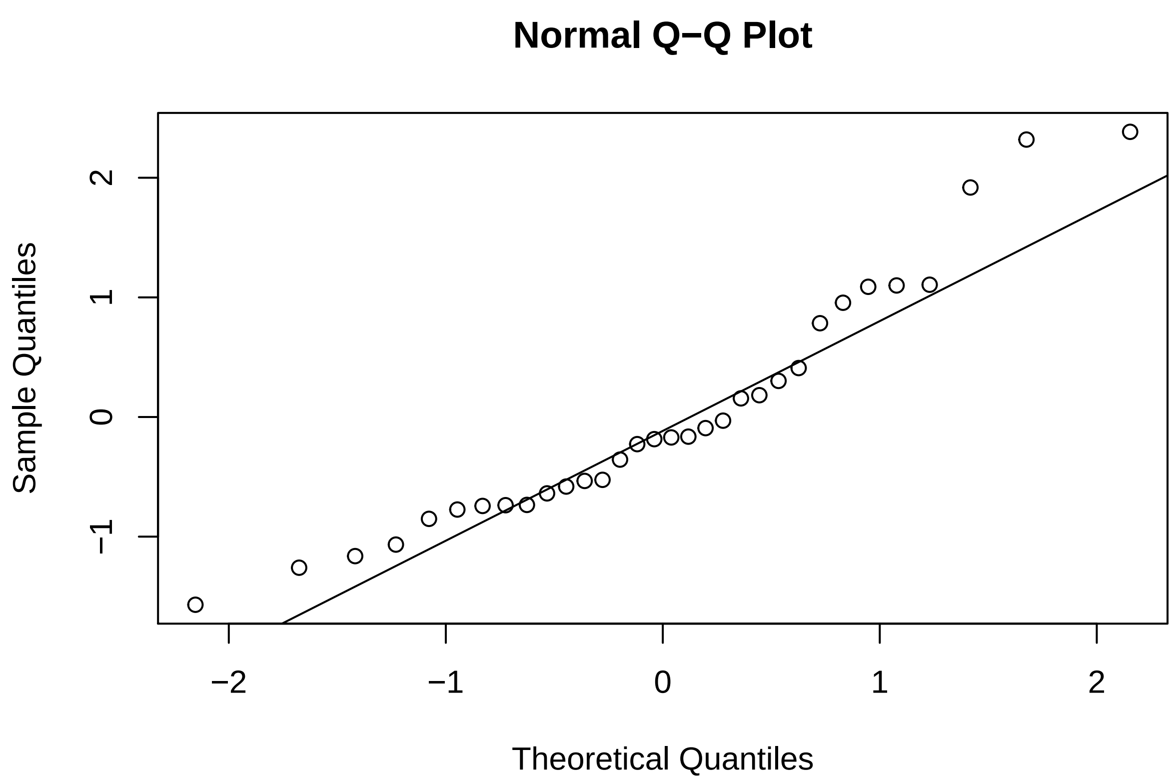
```
lbry <- c("grid", "gridExtra")
lapply(lbry, require, character.only=TRUE, warn.conflicts = FALSE, quietly = TRUE)
```

```
:> [[1]]
:> [1] TRUE
:>
:> [[2]]
:> [1] TRUE
```

Thus the plot obtained is,

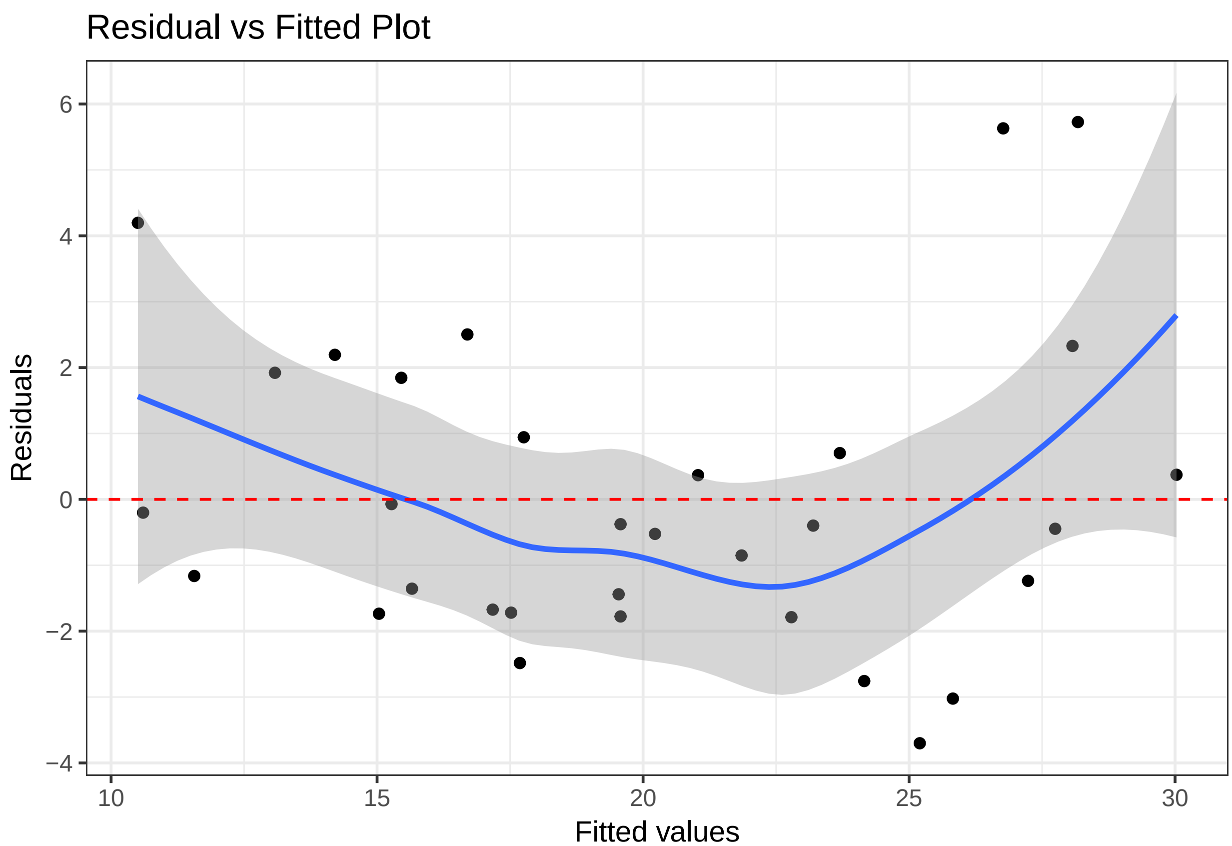
```
do.call(grid.arrange, c(diagPlts, main="Diagnostic Plots", ncol=3))
```

```
:> Error: Aesthetics must be either length 1 or the same as the data (32): x
```



4.1 Residual vs Fitted plot

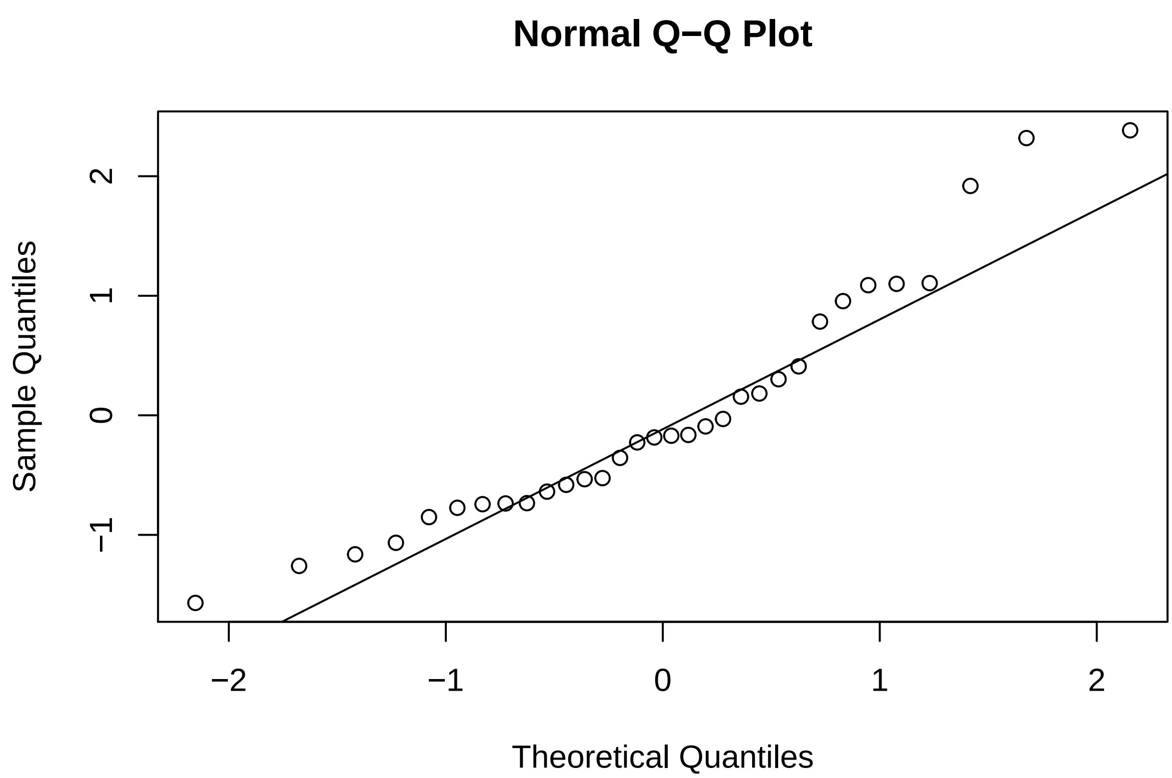
```
model <- lm.model
    p1<-ggplot(model, aes(.fitted, .resid)) +
        geom_point() +
        stat_smooth(method="loess") +
        geom_hline(yintercept=0, col="red", linetype="dashed") +
        xlab("Fitted values") +
        ylab("Residuals") +
        ggtitle("Residual vs Fitted Plot") +
        theme_bw()
    p1
```



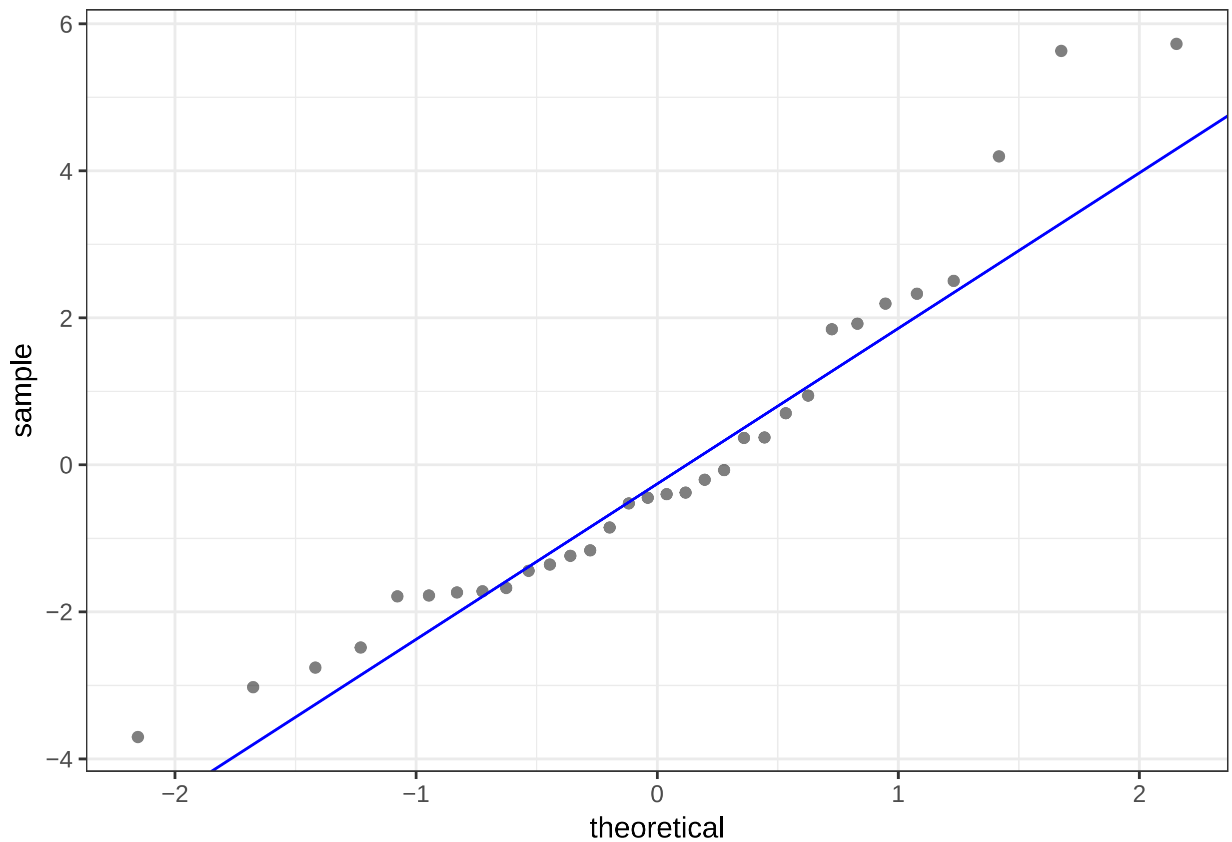
4.2 Normal QQ

```
    p2 <- ggplot(model, aes(qqnorm(.stdresid)[[1]], .stdresid)) +
    geom_point(na.rm = TRUE) +
    geom_abline(aes(qqline(.stdresid)))
    # xlab("Theoretical Quantiles") +
    # ylab("Standardized Residuals") +
    # ggtitle("Normal Q-Q") +
    # theme_bw()

p2
```

```
:> Error: Aesthetics must be either length 1 or the same as the data (32): x
```



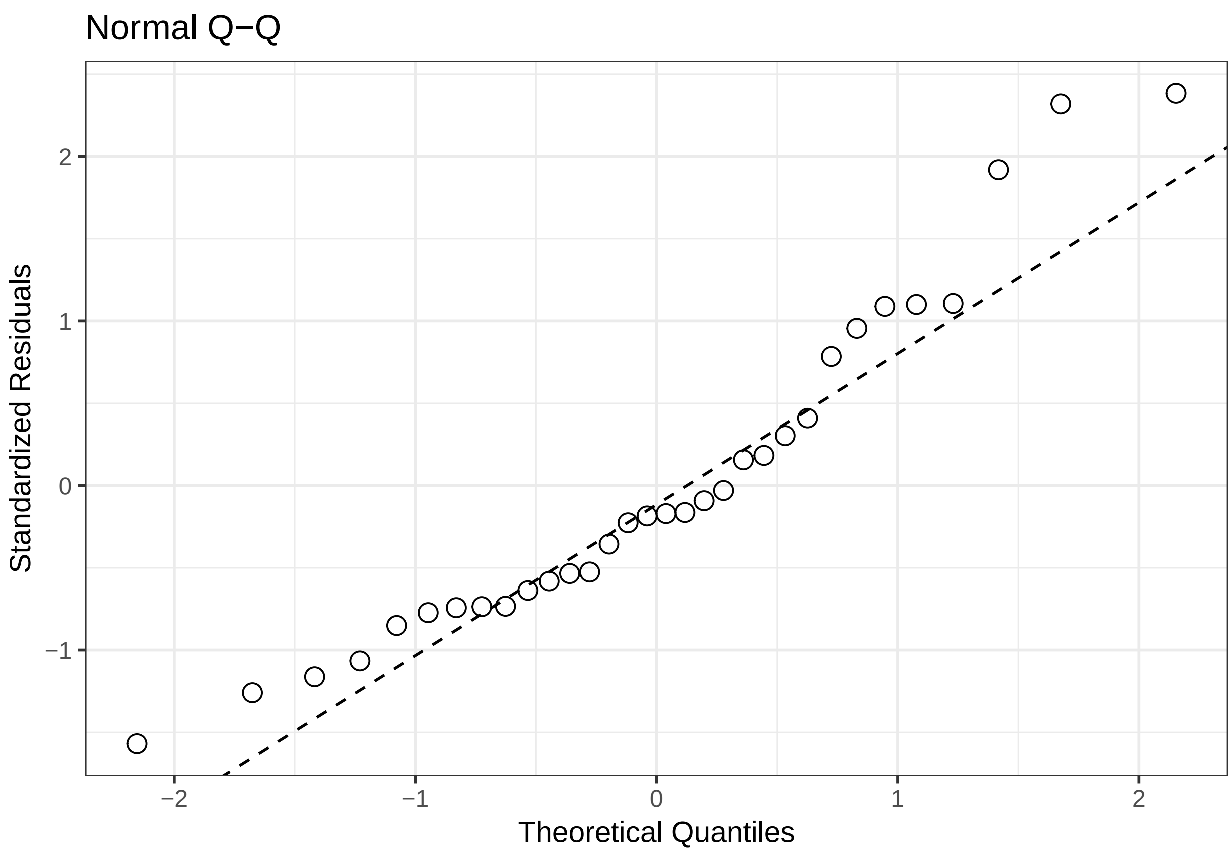
```
y <- quantile(model$resid[!is.na(model$resid)], c(0.25, 0.75))
x <- qnorm(c(0.25, 0.75))
slope <- diff(y)/diff(x)
int <- y[1L] - slope * x[1L]
p <- ggplot(model, aes(sample = .resid)) +
    stat_qq(alpha = 0.5) +
    geom_abline(slope = slope, intercept = int, color="blue") +
    theme_bw()

p
```



The standard Q-Q diagnostic for linear models plots quantiles of the standardized residuals vs. theoretical quantiles of N(0,1). Peter's `ggQQ` function plots the residuals. The snippet below amends that and adds a few cosmetic changes to make the plot more like what one would get from plot(lm(…)).

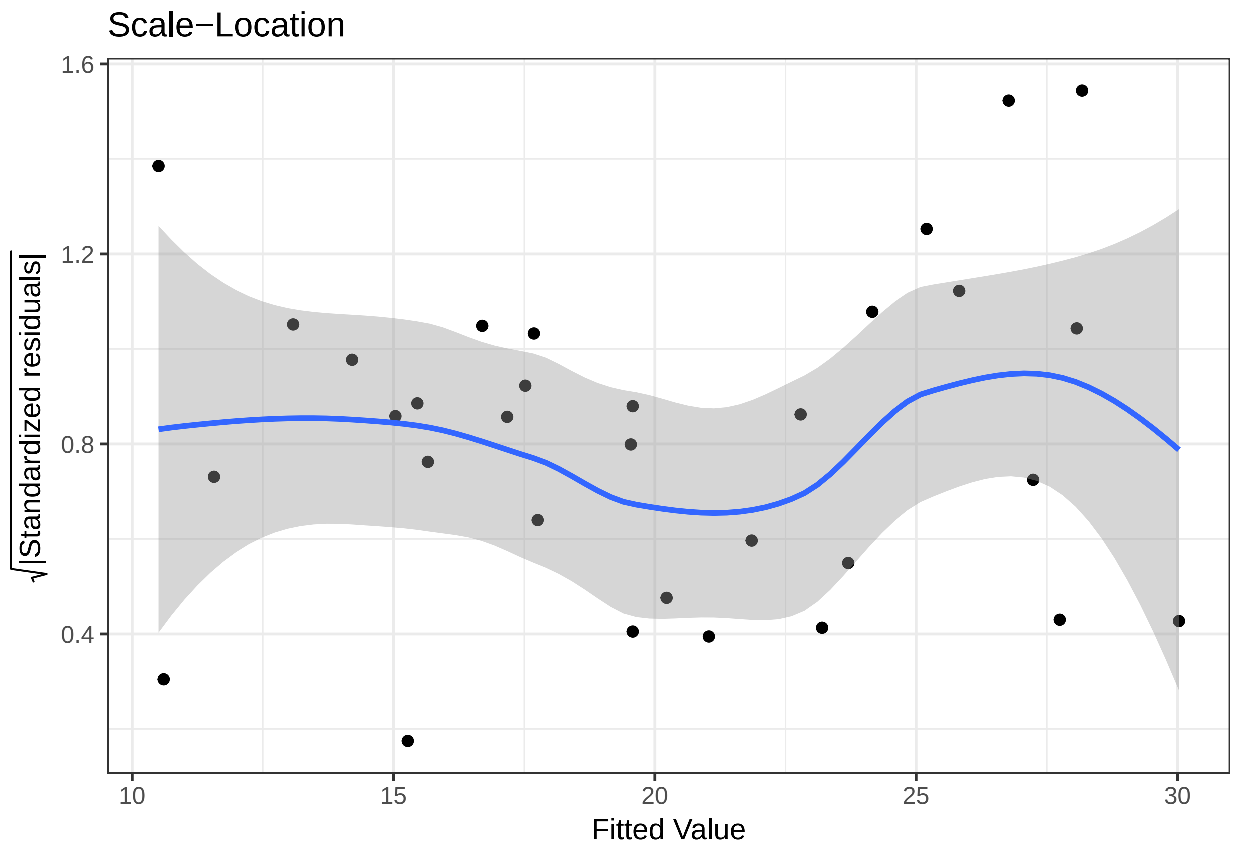
```
# https://stackoverflow.com/a/19990107/5270873
ggQQ = function(lm) {
  # extract standardized residuals from the fit
  d <- data.frame(std.resid = rstandard(lm))
  # calculate 1Q/4Q line
  y <- quantile(d$std.resid[!is.na(d$std.resid)], c(0.25, 0.75))
  x <- qnorm(c(0.25, 0.75))
  slope <- diff(y)/diff(x)
  int <- y[1L] - slope * x[1L]

  p <- ggplot(data=d, aes(sample=std.resid)) +
    stat_qq(shape=1, size=3) +           # open circles
```

```
    labs(title="Normal Q-Q",             # plot title
         x="Theoretical Quantiles",      # x-axis label
         y="Standardized Residuals") +   # y-axis label
    geom_abline(slope = slope, intercept = int, linetype="dashed")  + # dashed reference line
      theme_bw()
  return(p)
}

ggQQ(model)
```



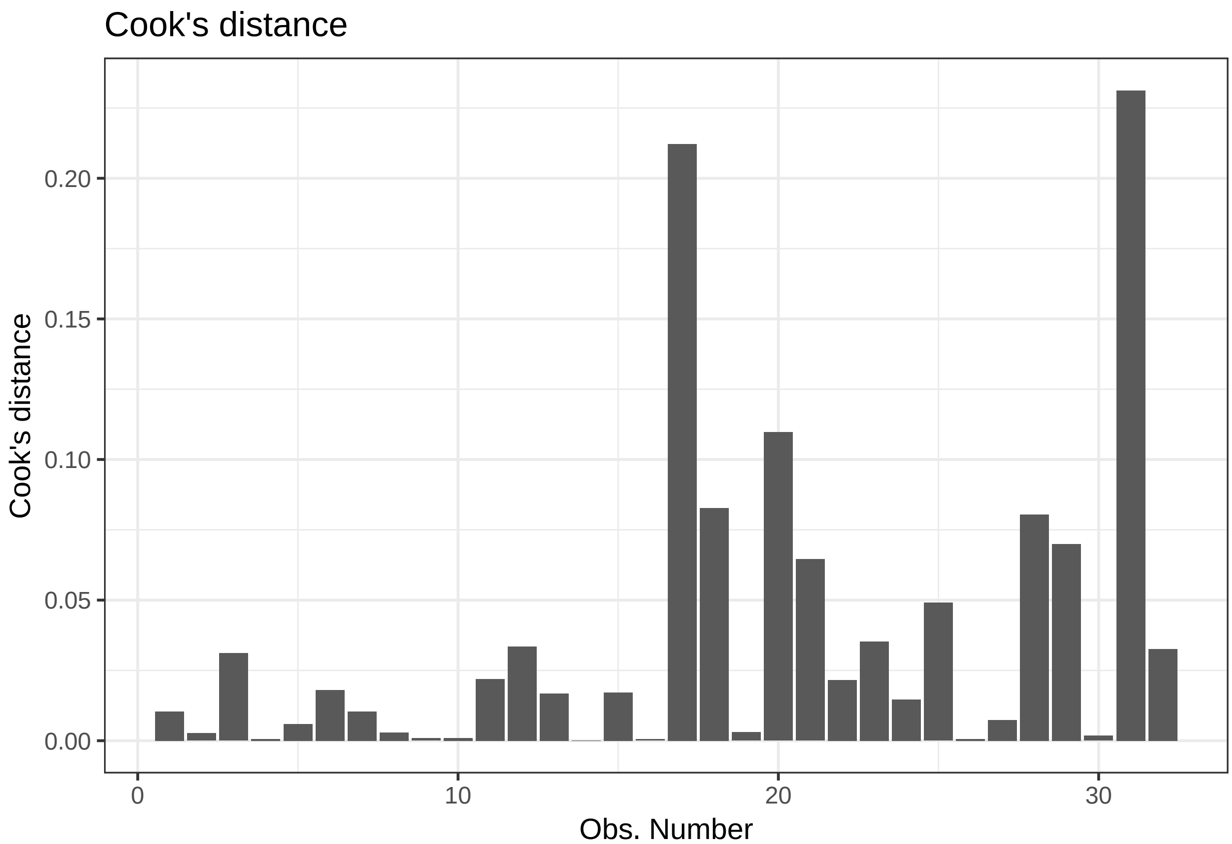
4.3 Scale-location

```
    p3 <- ggplot(model, aes(.fitted, sqrt(abs(.stdresid)))) +
    geom_point(na.rm=TRUE) +
    stat_smooth(method="loess", na.rm = TRUE) +
    xlab("Fitted Value") +
    ylab(expression(sqrt("|Standardized residuals|"))) +
    ggtitle("Scale-Location") +
    theme_bw()

    p3
```



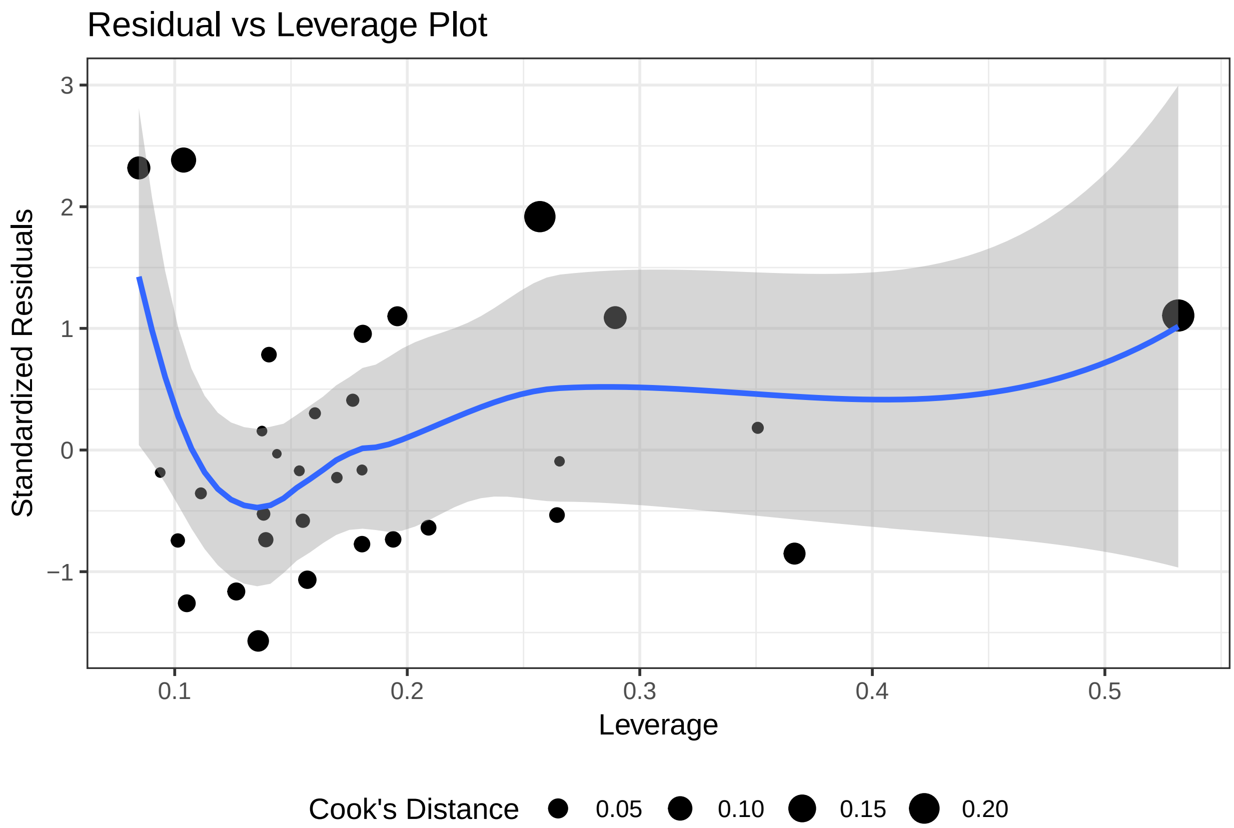
4.4 Cook's Distance

```
p4 <- ggplot(model, aes(seq_along(.cooksd), .cooksd)) +
geom_bar(stat="identity", position="identity") +
xlab("Obs. Number") +
ylab("Cook's distance") +
ggtitle("Cook's distance") +
theme_bw()

p4
```



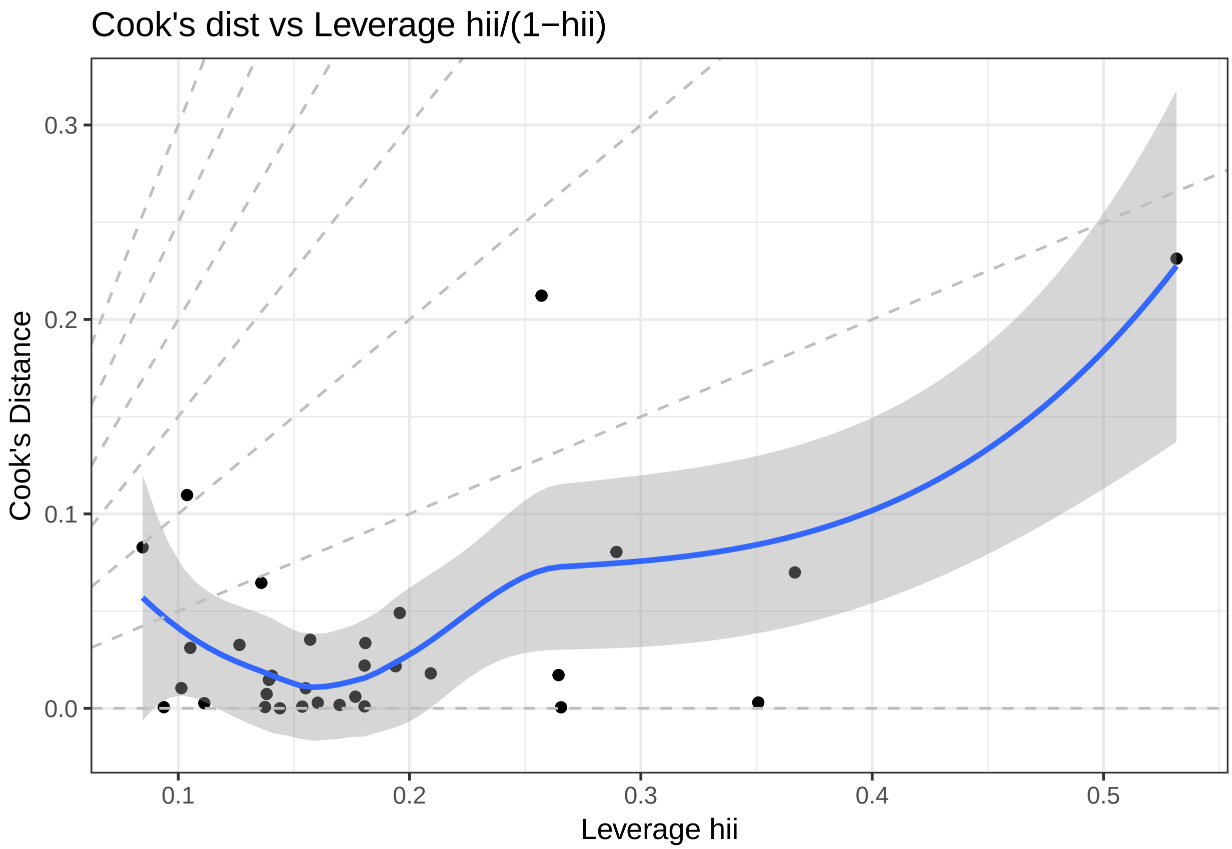
4.5 Residual vs Leverage Plot

```
p5 <- ggplot(model, aes(.hat, .stdresid)) +
geom_point(aes(size=.cooksd), na.rm=TRUE) +
stat_smooth(method="loess", na.rm=TRUE) +
xlab("Leverage") +
ylab("Standardized Residuals") +
ggtitle("Residual vs Leverage Plot") +
scale_size_continuous("Cook's Distance", range=c(1,5)) +
theme_bw() +
theme(legend.position="bottom")

p5
```



4.6 Cook's dist vs Leverage hii/(1-hii)

```
    p6 <- ggplot(model, aes(.hat, .cooksd)) +
    geom_point(na.rm=TRUE) +
    stat_smooth(method="loess", na.rm=TRUE) +
    xlab("Leverage hii") +
    ylab("Cook's Distance") +
    ggtitle("Cook's dist vs Leverage hii/(1-hii)") +
    geom_abline(slope=seq(0,3,0.5), color="gray", linetype="dashed") +
    theme_bw()

p6
```


Cook's dist vs Leverage hii/(1−hii)
Cook's Distance
0.3
0.2
0.1
0.0
0.1
0.2
0.3
0.4
0.5
Leverage hii

Chapter 5

Temperature modeling using nested dataframes

5.1 Prepare the data

http://ijlyttle.github.io/isugg_purrr/presentation.html#(1)

5.1.1 Packages to run this presentation

```
library("readr")
library("tibble")
library("dplyr")
library("tidyr")
library("stringr")
library("ggplot2")
library("purrr")
library("broom")
```

5.1.2 Motivation

As you know, purrr is a recent package from Hadley Wickham, focused on lists and functional programming, like dplyr is focused on data-frames.

I figure a good way to learn a new package is to try to solve a problem, so we have a dataset:

- you can view or download
- you can download the source of this presentation
- these are three temperatures recorded simultaneously in a piece of electronics
- it will be very valuable to be able to characterize the transient temperature for each sensor
- we want to apply the same set of models across all three sensors
- it will be easier to show using pictures

5.1.3 Let's get the data into shape

Using the readr package

```
temperature_wide <-
  read_csv(file.path(data_raw_dir, "temperature.csv")) %>%
  print()
```

```
# A tibble: 327 x 4
   instant             temperature_a temperature_b temperature_c
   <dttm>                      <dbl>         <dbl>         <dbl>
 1 2015-11-13 06:10:19          116.          91.7          84.2
 2 2015-11-13 06:10:23          116.          91.7          84.2
 3 2015-11-13 06:10:27          116.          91.6          84.2
 4 2015-11-13 06:10:31          116.          91.7          84.2
 5 2015-11-13 06:10:36          116.          91.7          84.2
 6 2015-11-13 06:10:41          116.          91.6          84.2
 7 2015-11-13 06:10:46          116.          91.5          84.2
 8 2015-11-13 06:10:51          116.          91.5          84.2
 9 2015-11-13 06:10:56          116.          91.5          84.2
10 2015-11-13 06:11:01          115.          91.5          84.2
# ... with 317 more rows
```

5.1.4 Is `temperature_wide` "tidy"?

```
# A tibble: 327 x 4
   instant             temperature_a temperature_b temperature_c
   <dttm>                      <dbl>         <dbl>         <dbl>
 1 2015-11-13 06:10:19          116.          91.7          84.2
 2 2015-11-13 06:10:23          116.          91.7          84.2
 3 2015-11-13 06:10:27          116.          91.6          84.2
 4 2015-11-13 06:10:31          116.          91.7          84.2
 5 2015-11-13 06:10:36          116.          91.7          84.2
 6 2015-11-13 06:10:41          116.          91.6          84.2
 7 2015-11-13 06:10:46          116.          91.5          84.2
 8 2015-11-13 06:10:51          116.          91.5          84.2
 9 2015-11-13 06:10:56          116.          91.5          84.2
10 2015-11-13 06:11:01          115.          91.5          84.2
# ... with 317 more rows
```

Why or why not?

5.1.5 Tidy data

1. Each column is a variable
2. Each row is an observation
3. Each cell is a value

(http://www.jstatsoft.org/v59/i10/paper)

My personal observation is that "tidy" can depend on the context, on what you want to do with the data.

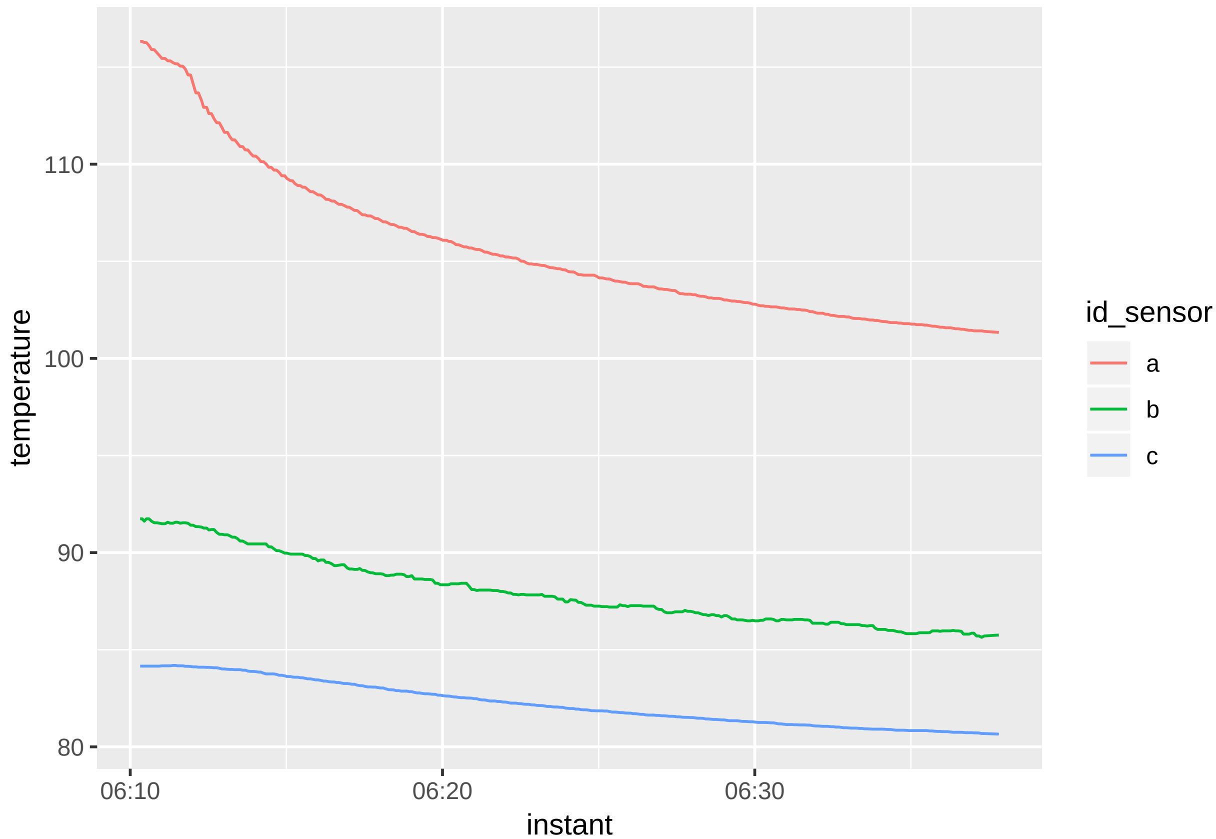
5.1.6 Let's get this into a tidy form

```
temperature_tall <-
  temperature_wide %>%
  gather(key = "id_sensor", value = "temperature", starts_with("temp")) %>%
  mutate(id_sensor = str_replace(id_sensor, "temperature_", "")) %>%
  print()
```

```
# A tibble: 981 x 3
   instant             id_sensor temperature
   <dttm>              <chr>           <dbl>
 1 2015-11-13 06:10:19 a                116.
 2 2015-11-13 06:10:23 a                116.
 3 2015-11-13 06:10:27 a                116.
 4 2015-11-13 06:10:31 a                116.
 5 2015-11-13 06:10:36 a                116.
 6 2015-11-13 06:10:41 a                116.
 7 2015-11-13 06:10:46 a                116.
 8 2015-11-13 06:10:51 a                116.
 9 2015-11-13 06:10:56 a                116.
10 2015-11-13 06:11:01 a                115.
# ... with 971 more rows
```

5.1.7 Now, it's easier to visualize

```
temperature_tall %>%
  ggplot(aes(x = instant, y = temperature, color = id_sensor)) +
  geom_line()
```



5.1.8 Calculate delta time (Δt) and delta temperature (ΔT)

`delta_time` Δt

change in time since event started, s

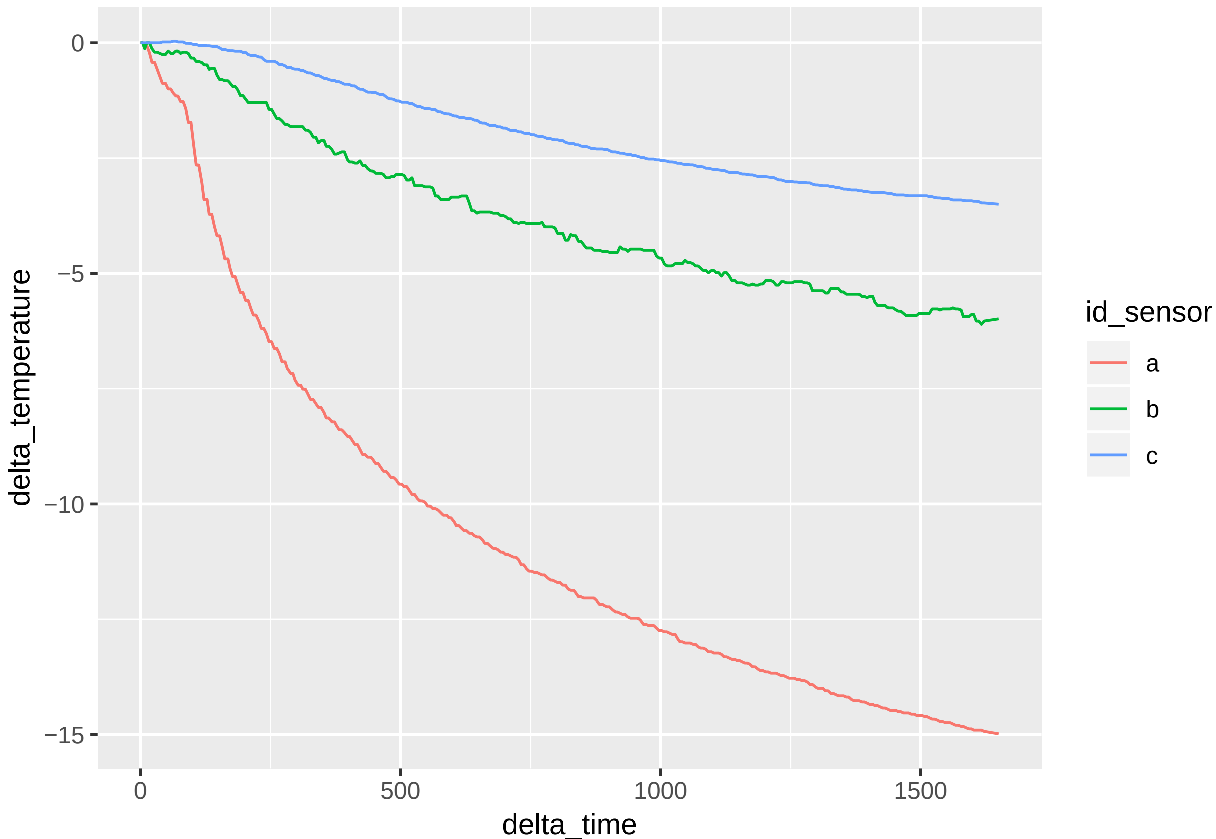
`delta_temperature`: ΔT

change in temperature since event started, °C

```
delta <-
  temperature_tall %>%
  arrange(id_sensor, instant) %>%
  group_by(id_sensor) %>%
  mutate(
    delta_time = as.numeric(instant) - as.numeric(instant[[1]]),
    delta_temperature = temperature - temperature[[1]]
  ) %>%
  select(id_sensor, delta_time, delta_temperature)
```

5.1.9 Let's have a look

```
# plot delta time vs delta temperature, by sensor
delta %>%
  ggplot(aes(x = delta_time, y = delta_temperature, color = id_sensor)) +
  geom_line()
```



5.2 Define the models

We want to see how three different curve-fits might perform on these three data-sets:

5.2.0.1 Newtonian cooling

$$\Delta T = \Delta T_0 * (1 - e^{-\frac{\delta t}{\tau_0}})$$

5.2.1 Semi-infinite solid

$$\Delta T = \Delta T_0 * erfc(\sqrt{\frac{\tau_0}{\delta t}}))$$

5.2.2 Semi-infinite solid with convection

$$\Delta T = \Delta T_0 * \left[\mathrm{e}rfc(\sqrt{\frac{\tau_0}{\delta t}}) - e^{Bi_0 + (\frac{Bi_0}{2})^2 \frac{\delta t}{\tau_0}} * \mathrm{e}rfc(\sqrt{\frac{\tau_0}{\delta t}} + \frac{Bi_0}{2} * \sqrt{\frac{\delta t}{\tau_0}}\right]$$

5.2.3 `erf` and `erfc` functions

```
# reference: http://stackoverflow.com/questions/29067916/r-error-function-erfz
# (see Abramowitz and Stegun 29.2.29)
erf <- function(x) 2 * pnorm(x * sqrt(2)) - 1
erfc <- function(x) 2 * pnorm(x * sqrt(2), lower = FALSE)
```

5.2.4 Newton cooling equation

```
newton_cooling <- function(x) {
  nls(
    delta_temperature ~ delta_temperature_0 * (1 - exp(-delta_time/tau_0)),
    start = list(delta_temperature_0 = -10, tau_0 = 50),
    data = x
  )
}
```

5.2.5 Temperature models: simple and convection

```
semi_infinite_simple <- function(x) {
  nls(
    delta_temperature ~ delta_temperature_0 * erfc(sqrt(tau_0 / delta_time)),
    start = list(delta_temperature_0 = -10, tau_0 = 50),
    data = x
  )
}

semi_infinite_convection <- function(x){
  nls(
    delta_temperature ~
      delta_temperature_0 * (
        erfc(sqrt(tau_0 / delta_time)) -
        exp(Bi_0 + (Bi_0/2)^2 * delta_time / tau_0) *
          erfc(sqrt(tau_0 / delta_time) +
        (Bi_0/2) * sqrt(delta_time / tau_0))
      ),
    start = list(delta_temperature_0 = -5, tau_0 = 50, Bi_0 = 1.e6),
    data = x
  )
}
```

5.3 Test modeling on one dataset

5.3.1 Before going into purrr

Before doing anything, we want to show that we can do something with one dataset and one model-function:

```
# only one sensor; it is a test
tmp_data <- delta %>% filter(id_sensor == "a")

tmp_model <- newton_cooling(tmp_data)
```

```r
summary(tmp_model)
```

```
Formula: delta_temperature ~ delta_temperature_0 * (1 - exp(-delta_time/tau_0))

Parameters:
                     Estimate Std. Error t value Pr(>|t|)
delta_temperature_0 -15.06085    0.05262  -286.2   <2e-16 ***
tau_0               500.01382    4.83673   103.4   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3267 on 325 degrees of freedom

Number of iterations to convergence: 7
Achieved convergence tolerance: 4.136e-06
```

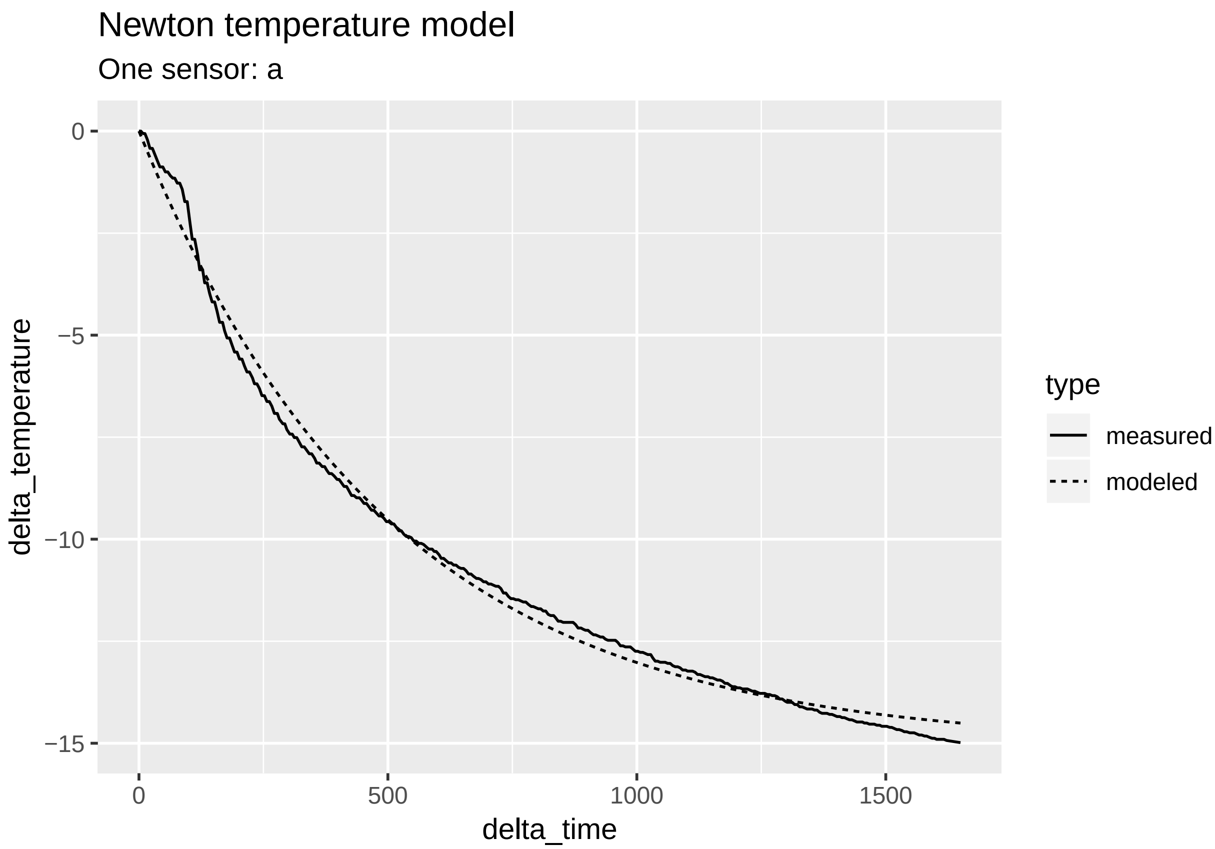
5.3.2 Look at predictions

```r
# apply prediction and make it tidy
tmp_pred <-
  tmp_data %>%
  mutate(modeled = predict(tmp_model, data = .)) %>%
  select(id_sensor, delta_time, measured = delta_temperature, modeled) %>%
  gather("type", "delta_temperature", measured:modeled) %>%
  print()
```

```
# A tibble: 654 x 4
# Groups:   id_sensor [1]
   id_sensor delta_time type     delta_temperature
   <chr>          <dbl> <chr>                <dbl>
 1 a                  0 measured             0
 2 a                  4 measured             0
 3 a                  8 measured            -0.06
 4 a                 12 measured            -0.06
 5 a                 17 measured            -0.211
 6 a                 22 measured            -0.423
 7 a                 27 measured            -0.423
 8 a                 32 measured            -0.574
 9 a                 37 measured            -0.726
10 a                 42 measured            -0.878
# ... with 644 more rows
```

5.3.3 Plot Newton model

```r
tmp_pred %>%
  ggplot(aes(x = delta_time, y = delta_temperature, linetype = type)) +
  geom_line() +
  labs(title = "Newton temperature model", subtitle = "One sensor: a")
```



5.3.4 "Regular" data-frame (deltas)

```
print(delta)
```

```
# A tibble: 981 x 3
# Groups:   id_sensor [3]
   id_sensor delta_time delta_temperature
   <chr>          <dbl>             <dbl>
 1 a                  0             0
 2 a                  4             0
 3 a                  8            -0.06
 4 a                 12            -0.06
 5 a                 17            -0.211
 6 a                 22            -0.423
 7 a                 27            -0.423
 8 a                 32            -0.574
 9 a                 37            -0.726
10 a                 42            -0.878
# ... with 971 more rows
```

Each column of the dataframe is a vector - in this case, a character vector and two doubles

5.4 Making a nested dataframe

5.4.1 How to make a weird data-frame

Here's where the fun starts - a column of a data-frame can be a list.

- use `tidyr::nest()` to makes a column `data`, which is a list of data-frames
- this seems like a stronger expression of the `dplyr::group_by()` idea

```
# nest delta_time and delta_temperature variables
delta_nested <-
  delta %>%
  nest(-id_sensor) %>%
  print()
```

```
# A tibble: 3 x 2
  id_sensor data
  <chr>     <list>
1 a         <tibble [327 x 2]>
2 b         <tibble [327 x 2]>
3 c         <tibble [327 x 2]>
```

5.4.2 Map dataframes to a modeling function (Newton)

- `map()` is like `lapply()`
- `map()` returns a list-column (it keeps the weirdness)

```
model_nested <-
  delta_nested %>%
  mutate(model = map(data, newton_cooling)) %>%
  print()
```

```
# A tibble: 3 x 3
  id_sensor data               model
  <chr>     <list>             <list>
1 a         <tibble [327 x 2]> <nls>
2 b         <tibble [327 x 2]> <nls>
3 c         <tibble [327 x 2]> <nls>
```

We get an additional list-column `model`.

5.4.3 We can use `map2()` to make the predictions

- `map2()` is like `mapply()`
- designed to map two columns (`model`, `data`) to a function `predict()`

```
predict_nested <-
  model_nested %>%
  mutate(pred = map2(model, data, predict)) %>%
  print()
```

```
# A tibble: 3 x 4
  id_sensor data               model  pred
  <chr>     <list>             <list> <list>
```

```
1 a         <tibble [327 x 2]> <nls>  <dbl [327]>
2 b         <tibble [327 x 2]> <nls>  <dbl [327]>
3 c         <tibble [327 x 2]> <nls>  <dbl [327]>
```

Another list-column `pred` for the prediction results.

5.4.4 We need to get out of the weirdness

- use `unnest()` to get back to a regular data-frame

```
predict_unnested <-
  predict_nested %>%
  unnest(data, pred) %>%
  print()
```

```
# A tibble: 981 x 4
   id_sensor   pred delta_time delta_temperature
   <chr>      <dbl>      <dbl>             <dbl>
 1 a          0              0             0
 2 a         -0.120          4             0
 3 a         -0.239          8            -0.06
 4 a         -0.357         12            -0.06
 5 a         -0.503         17            -0.211
 6 a         -0.648         22            -0.423
 7 a         -0.792         27            -0.423
 8 a         -0.934         32            -0.574
 9 a         -1.07          37            -0.726
10 a         -1.21          42            -0.878
# ... with 971 more rows
```

5.4.5 We can wrangle the predictions

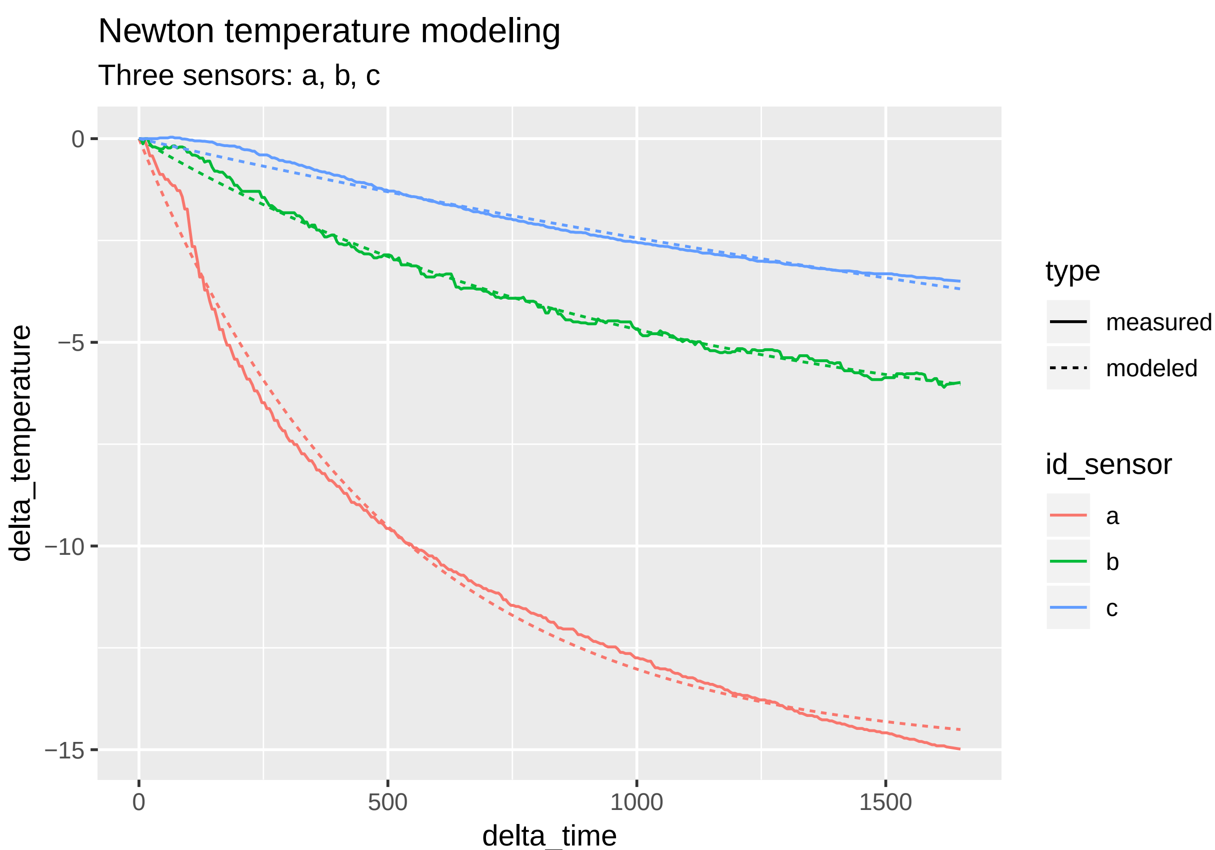
- get into a form that makes it easier to plot

```
predict_tall <-
  predict_unnested %>%
  rename(modeled = pred, measured = delta_temperature) %>%
  gather("type", "delta_temperature", modeled, measured) %>%
  print()
```

```
# A tibble: 1,962 x 4
   id_sensor delta_time type    delta_temperature
   <chr>          <dbl> <chr>               <dbl>
 1 a                  0 modeled             0
 2 a                  4 modeled            -0.120
 3 a                  8 modeled            -0.239
 4 a                 12 modeled            -0.357
 5 a                 17 modeled            -0.503
 6 a                 22 modeled            -0.648
 7 a                 27 modeled            -0.792
 8 a                 32 modeled            -0.934
 9 a                 37 modeled            -1.07
10 a                 42 modeled            -1.21
# ... with 1,952 more rows
```

5.4.6 We can visualize the predictions

```
predict_tall %>%
  ggplot(aes(x = delta_time, y = delta_temperature)) +
  geom_line(aes(color = id_sensor, linetype = type)) +
  labs(title = "Newton temperature modeling",
       subtitle = "Three sensors: a, b, c")
```



5.5 Apply multiple models on a nested structure

5.5.1 Step 1: Selection of models

Make a list of functions to model:

```
list_model <-
  list(
    newton_cooling = newton_cooling,
    semi_infinite_simple = semi_infinite_simple,
    semi_infinite_convection = semi_infinite_convection
  )
```

5.5.2 Step 2: write a function to define the "inner" loop

```
# add additional variable with the model name

fn_model <- function(.model, df) {
  # one parameter for the model in the list, the second for the data
  # safer to avoid non-standard evaluation
  # df %>% mutate(model = map(data, .model))

  df$model <- map(df$data, possibly(.model, NULL))
  df
}
```

- for a given model-function and a given (weird) data-frame, return a modified version of that data-frame with a column `model`, which is the model-function applied to each element of the data-frame's `data` column (which is itself a list of data-frames)
- the purrr functions `safely()` and `possibly()` are **very** interesting. I think they could be useful outside of purrr as a friendlier way to do error-handling.

5.5.3 Step 3: Use `map_df()` to define the "outer" loop

```
# this dataframe will be the second input of fn_model
delta_nested %>%
  print()
```

```
# A tibble: 3 x 2
  id_sensor data
  <chr>     <list>
1 a         <tibble [327 x 2]>
2 b         <tibble [327 x 2]>
3 c         <tibble [327 x 2]>
```

```
# fn_model is receiving two inputs: one from list_model and from delta_nested
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  print()
```

```
# A tibble: 9 x 4
  id_model                 id_sensor data               model
  <chr>                    <chr>     <list>             <list>
1 newton_cooling           a         <tibble [327 x 2]> <nls>
2 newton_cooling           b         <tibble [327 x 2]> <nls>
3 newton_cooling           c         <tibble [327 x 2]> <nls>
4 semi_infinite_simple     a         <tibble [327 x 2]> <nls>
5 semi_infinite_simple     b         <tibble [327 x 2]> <nls>
6 semi_infinite_simple     c         <tibble [327 x 2]> <nls>
7 semi_infinite_convection a         <tibble [327 x 2]> <NULL>
8 semi_infinite_convection b         <tibble [327 x 2]> <NULL>
9 semi_infinite_convection c         <tibble [327 x 2]> <NULL>
```

- for each element of a list of model-functions, run the inner-loop function, and row-bind the results into a data-frame

- we want to discard the rows where the model failed
- we also want to investigate why they failed, but that's a different talk

5.5.4 Step 4: Use `map()` to identify the null models

```
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  mutate(is_null = map(model, is.null)) %>%
  print()
```

```
# A tibble: 9 x 5
  id_model                 id_sensor data               model  is_null
  <chr>                    <chr>     <list>             <list> <list>
1 newton_cooling           a         <tibble [327 x 2]> <nls>  <lgl [1]>
2 newton_cooling           b         <tibble [327 x 2]> <nls>  <lgl [1]>
3 newton_cooling           c         <tibble [327 x 2]> <nls>  <lgl [1]>
4 semi_infinite_simple     a         <tibble [327 x 2]> <nls>  <lgl [1]>
5 semi_infinite_simple     b         <tibble [327 x 2]> <nls>  <lgl [1]>
6 semi_infinite_simple     c         <tibble [327 x 2]> <nls>  <lgl [1]>
7 semi_infinite_convection a         <tibble [327 x 2]> <NULL> <lgl [1]>
8 semi_infinite_convection b         <tibble [327 x 2]> <NULL> <lgl [1]>
9 semi_infinite_convection c         <tibble [327 x 2]> <NULL> <lgl [1]>
```

- using `map(model, is.null)` returns a list column
- to use `filter()`, we have to escape the weirdness

5.5.5 Step 5: `map_lgl()` to identify nulls and get out of the weirdness

```
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  mutate(is_null = map_lgl(model, is.null)) %>%
  print()
```

```
# A tibble: 9 x 5
  id_model                 id_sensor data               model  is_null
  <chr>                    <chr>     <list>             <list> <lgl>
1 newton_cooling           a         <tibble [327 x 2]> <nls>  FALSE
2 newton_cooling           b         <tibble [327 x 2]> <nls>  FALSE
3 newton_cooling           c         <tibble [327 x 2]> <nls>  FALSE
4 semi_infinite_simple     a         <tibble [327 x 2]> <nls>  FALSE
5 semi_infinite_simple     b         <tibble [327 x 2]> <nls>  FALSE
6 semi_infinite_simple     c         <tibble [327 x 2]> <nls>  FALSE
7 semi_infinite_convection a         <tibble [327 x 2]> <NULL> TRUE
8 semi_infinite_convection b         <tibble [327 x 2]> <NULL> TRUE
9 semi_infinite_convection c         <tibble [327 x 2]> <NULL> TRUE
```

- using `map_lgl(model, is.null)` returns a vector column

5.5.6 Step 6: `filter()` nulls and `select()` variables to clean up

```
model_nested_new <-
  list_model %>%
  map_df(fn_model, delta_nested, .id = "id_model") %>%
  mutate(is_null = map_lgl(model, is.null)) %>%
  filter(!is_null) %>%
  select(-is_null) %>%
  print()
```

```
# A tibble: 6 x 4
  id_model             id_sensor data               model
  <chr>                <chr>     <list>             <list>
1 newton_cooling       a         <tibble [327 x 2]> <nls>
2 newton_cooling       b         <tibble [327 x 2]> <nls>
3 newton_cooling       c         <tibble [327 x 2]> <nls>
4 semi_infinite_simple a         <tibble [327 x 2]> <nls>
5 semi_infinite_simple b         <tibble [327 x 2]> <nls>
6 semi_infinite_simple c         <tibble [327 x 2]> <nls>
```

5.5.7 Step 7: Calculate predictions on nested dataframe

```
predict_nested <-
  model_nested_new %>%
  mutate(pred = map2(model, data, predict)) %>%
  print()
```

```
# A tibble: 6 x 5
  id_model             id_sensor data               model  pred
  <chr>                <chr>     <list>             <list> <list>
1 newton_cooling       a         <tibble [327 x 2]> <nls>  <dbl [327]>
2 newton_cooling       b         <tibble [327 x 2]> <nls>  <dbl [327]>
3 newton_cooling       c         <tibble [327 x 2]> <nls>  <dbl [327]>
4 semi_infinite_simple a         <tibble [327 x 2]> <nls>  <dbl [327]>
5 semi_infinite_simple b         <tibble [327 x 2]> <nls>  <dbl [327]>
6 semi_infinite_simple c         <tibble [327 x 2]> <nls>  <dbl [327]>
```

5.5.8 `unnest()`, make it tall and tidy

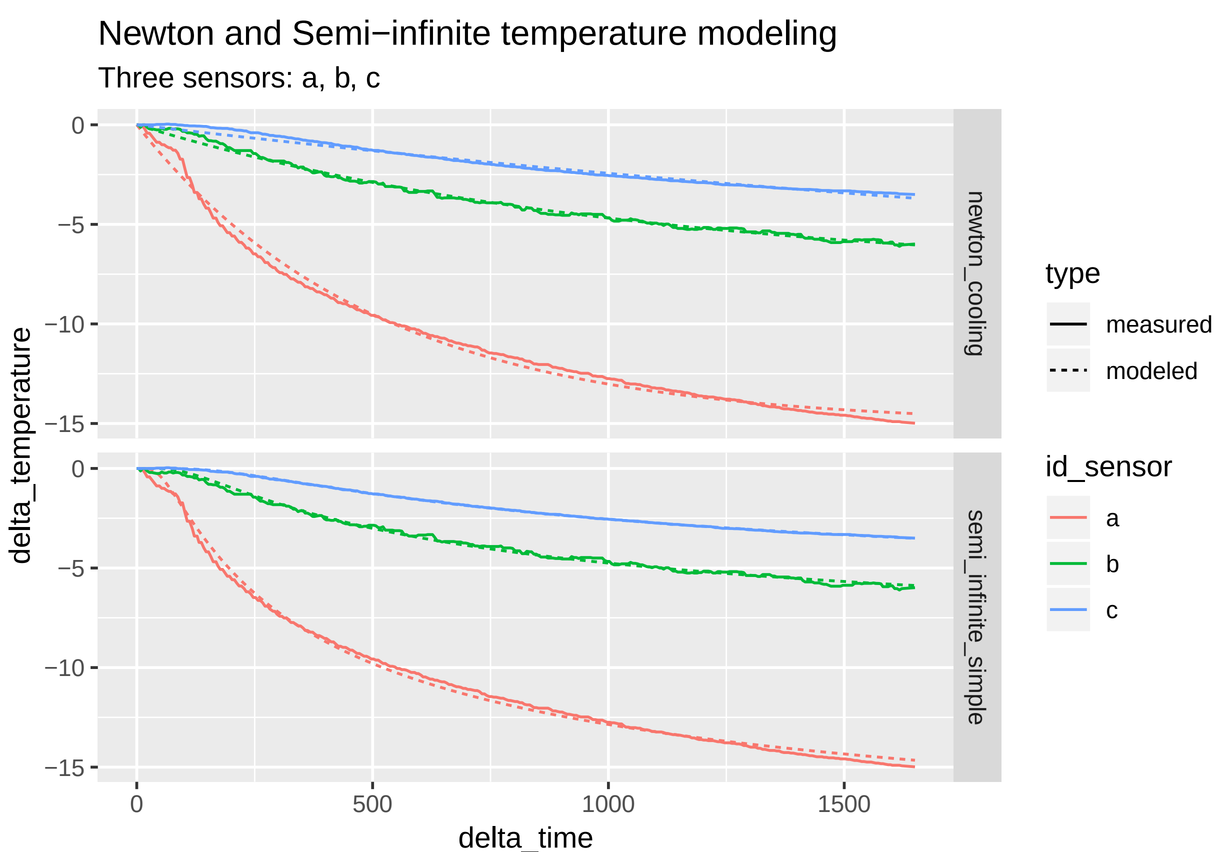
```
predict_tall <-
  predict_nested %>%
  unnest(data, pred) %>%
  rename(modeled = pred, measured = delta_temperature) %>%
  gather("type", "delta_temperature", modeled, measured) %>%
  print()
```

```
# A tibble: 3,924 x 5
   id_model       id_sensor delta_time type    delta_temperature
   <chr>          <chr>          <dbl> <chr>               <dbl>
 1 newton_cooling a                  0 modeled             0
 2 newton_cooling a                  4 modeled            -0.120
 3 newton_cooling a                  8 modeled            -0.239
```

```
 4 newton_cooling a                 12 modeled            -0.357
 5 newton_cooling a                 17 modeled            -0.503
 6 newton_cooling a                 22 modeled            -0.648
 7 newton_cooling a                 27 modeled            -0.792
 8 newton_cooling a                 32 modeled            -0.934
 9 newton_cooling a                 37 modeled            -1.07
10 newton_cooling a                 42 modeled            -1.21
# ... with 3,914 more rows
```

5.5.9 Visualize the predictions

```
predict_tall %>%
  ggplot(aes(x = delta_time, y = delta_temperature)) +
  geom_line(aes(color = id_sensor, linetype = type)) +
  facet_grid(id_model ~ .) +
  labs(title = "Newton and Semi-infinite temperature modeling",
       subtitle = "Three sensors: a, b, c")
```



5.5.10 Let's get the residuals

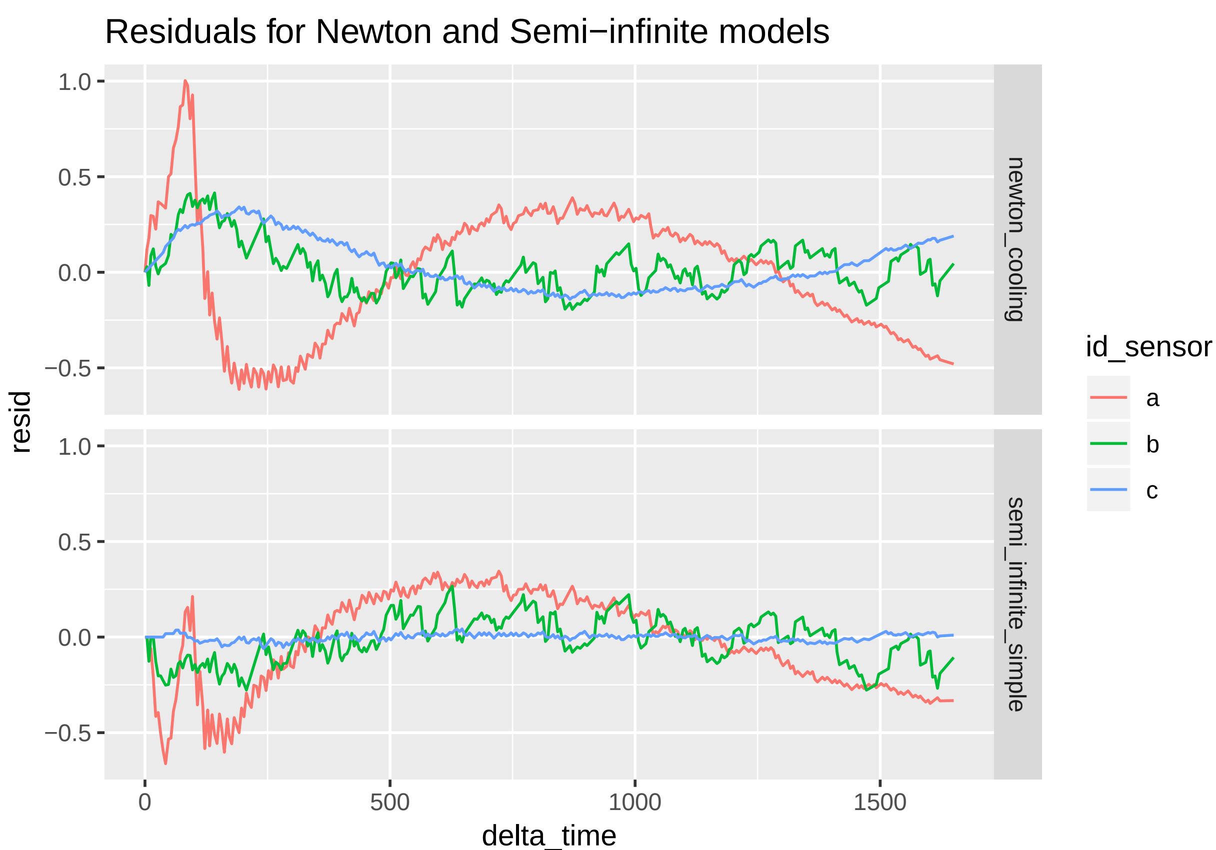
```
resid <-
  model_nested_new %>%
  mutate(resid = map(model, resid)) %>%
```

```
  unnest(data, resid) %>%
  print()
```

```
# A tibble: 1,962 x 5
   id_model       id_sensor resid delta_time delta_temperature
   <chr>          <chr>     <dbl>      <dbl>             <dbl>
 1 newton_cooling a         0              0             0
 2 newton_cooling a         0.120          4             0
 3 newton_cooling a         0.179          8            -0.06
 4 newton_cooling a         0.297         12            -0.06
 5 newton_cooling a         0.292         17            -0.211
 6 newton_cooling a         0.225         22            -0.423
 7 newton_cooling a         0.369         27            -0.423
 8 newton_cooling a         0.360         32            -0.574
 9 newton_cooling a         0.348         37            -0.726
10 newton_cooling a         0.335         42            -0.878
# ... with 1,952 more rows
```

5.5.11 And visualize them

```
resid %>%
  ggplot(aes(x = delta_time, y = resid)) +
  geom_line(aes(color = id_sensor)) +
  facet_grid(id_model ~ .) +
  labs(title = "Residuals for Newton and Semi-infinite models")
```



5.6 Using broom package to look at model-statistics

We will use a previous defined dataframe with the model and data:

```
model_nested_new %>%
  print()
```

```
# A tibble: 6 x 4
  id_model            id_sensor data               model
  <chr>               <chr>     <list>             <list>
1 newton_cooling      a         <tibble [327 x 2]> <nls>
2 newton_cooling      b         <tibble [327 x 2]> <nls>
3 newton_cooling      c         <tibble [327 x 2]> <nls>
4 semi_infinite_simple a         <tibble [327 x 2]> <nls>
5 semi_infinite_simple b         <tibble [327 x 2]> <nls>
6 semi_infinite_simple c         <tibble [327 x 2]> <nls>
```

The `tidy()` function extracts statistics from a model.

```
# apply over model_nested_new but only three variables
model_parameters <-
  model_nested_new %>%
  select(id_model, id_sensor, model) %>%
  mutate(tidy = map(model, tidy)) %>%
  select(-model) %>%
  unnest() %>%
  print()
```

```
# A tibble: 12 x 7
   id_model      id_sensor term      estimate std.error statistic   p.value
   <chr>         <chr>     <chr>        <dbl>     <dbl>     <dbl>     <dbl>
 1 newton_cooli~ a         delta_te~   -15.1     0.0526    -286.  0.
 2 newton_cooli~ a         tau_0       500.      4.84       103.  1.07e-250
 3 newton_cooli~ b         delta_te~    -7.59    0.0676    -112.  6.38e-262
 4 newton_cooli~ b         tau_0      1041.     16.2         64.2 9.05e-187
 5 newton_cooli~ c         delta_te~    -9.87    0.704      -14.0 3.16e- 35
 6 newton_cooli~ c         tau_0      3525.    299.          11.8 5.61e- 27
 7 semi_infinit~ a         delta_te~   -21.5     0.0649    -332.  0.
 8 semi_infinit~ a         tau_0       139.      1.15       121.  2.14e-272
 9 semi_infinit~ b         delta_te~   -10.6     0.0515    -206.  0.
10 semi_infinit~ b         tau_0       287.      2.58       111.  1.46e-260
11 semi_infinit~ c         delta_te~    -8.04    0.0129    -626.  0.
12 semi_infinit~ c         tau_0       500.      1.07       468.  0.
```

5.6.1 Get a sense of the coefficients

```
model_summary <-
  model_parameters %>%
  select(id_model, id_sensor, term, estimate) %>%
  spread(key = "term", value = "estimate") %>%
  print()
```

```
# A tibble: 6 x 4
  id_model            id_sensor delta_temperature_0 tau_0
  <chr>               <chr>                   <dbl> <dbl>
```

```
1 newton_cooling       a                        -15.1   500.
2 newton_cooling       b                         -7.59 1041.
3 newton_cooling       c                         -9.87 3525.
4 semi_infinite_simple a                        -21.5   139.
5 semi_infinite_simple b                        -10.6   287.
6 semi_infinite_simple c                         -8.04  500.
```

5.6.2 Summary

- this is just a smalll part of purrr
- there seem to be parallels between `tidyr::nest()/purrr::map()` and `dplyr::group_by()/dplyr::do()`
  - to my mind, the purrr framework is more understandable
  - update tweet from Hadley

References from Hadley:

- purrr 0.1.0 announcement
- purrr 0.2.0 announcement
- chapter from Garrett Grolemund and Hadley's forthcoming book

Chapter 6

Logistic Regression. Diabetes

6.1 Introduction

Source: https://github.com/AntoineGuillot2/Logistic-Regression-R/blob/master/script.R Source: http://enhancedatascience.com/2017/04/26/r-basics-logistic-regression-with-r/ Data: https://www.kaggle.com/uciml/pima-indians-diabetes-database

The goal of logistic regression is to predict whether an outcome will be positive (aka 1) or negative (i.e: 0). Some real life example could be:

- Will Emmanuel Macron win the French Presidential election or will he lose?
- Does Mr.X has this illness or not?
- Will this visitor click on my link or not?

So, logistic regression can be used in a lot of binary classification cases and will often be run before more advanced methods. For this tutorial, we will use the diabetes detection dataset from Kaggle.

This dataset contains data from Pima Indians Women such as the number of pregnancies, the blood pressure, the skin thickness, … the goal of the tutorial is to be able to detect diabetes using only these measures.

6.2 Exploring the data

As usual, first, let's take a look at our data. You can download the data here then please put the file diabetes.csv in your working directory. With the summary function, we can easily summarise the different variables:

```
library(ggplot2)
library(data.table)

DiabetesData <- data.table(read.csv(file.path(data_raw_dir, 'diabetes.csv')))

# Quick data summary
summary(DiabetesData)
```

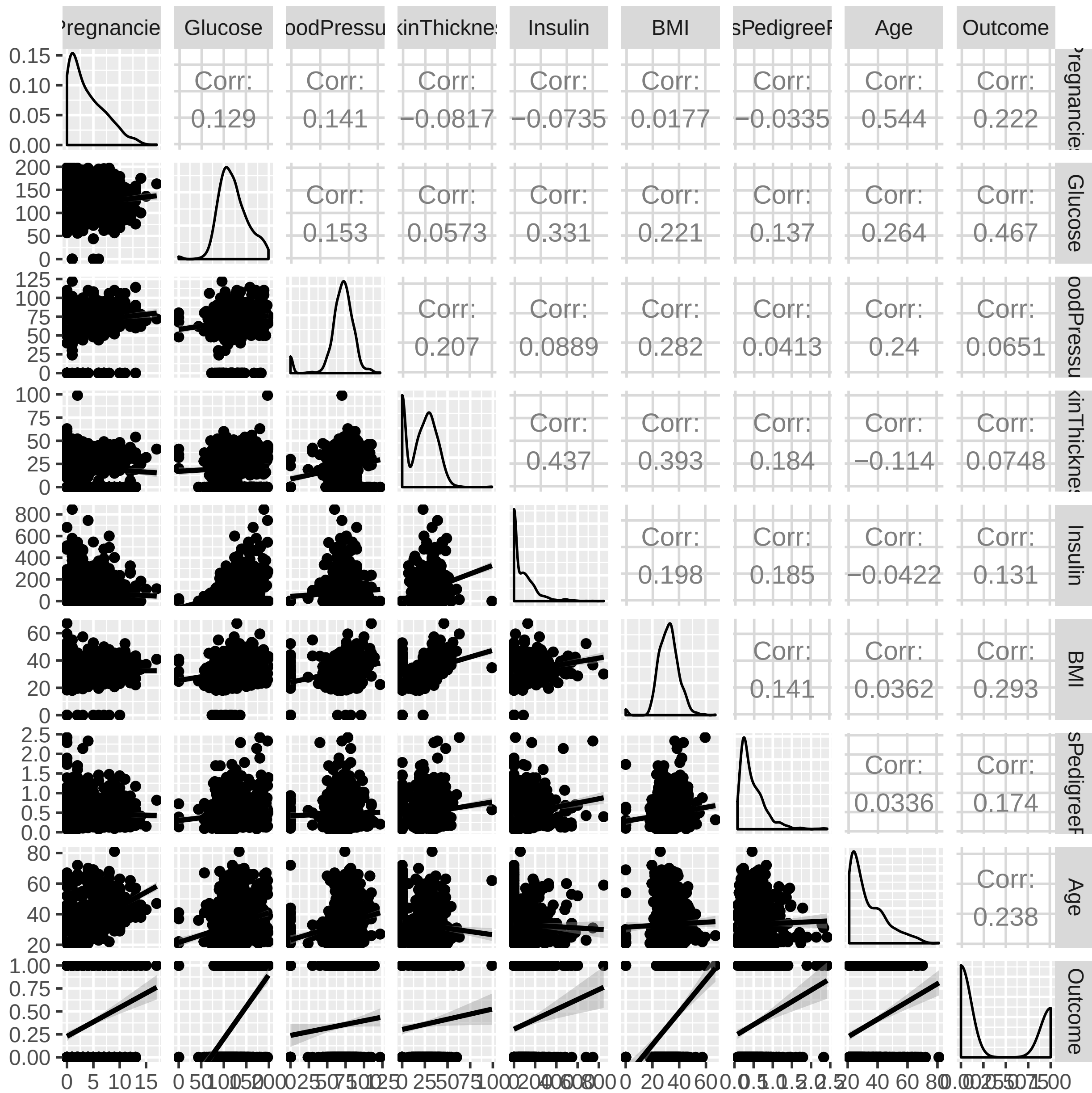
```
  Pregnancies        Glucose      BloodPressure    SkinThickness
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
```

```
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00  
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00  
    Insulin           BMI        DiabetesPedigreeFunction      Age       
 Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00  
 1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00  
 Median : 30.5   Median :32.00   Median :0.3725           Median :29.00  
 Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24  
 3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00  
 Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00  
    Outcome     
 Min.   :0.000  
 1st Qu.:0.000  
 Median :0.000  
 Mean   :0.349  
 3rd Qu.:1.000  
 Max.   :1.000  
```

The mean of the outcome is 0.35 which shows an imbalance between the classes. However, the imbalance should not be too strong to be a problem.

To understand the relationship between variables, a Scatter Plot Matrix will be used. To plot it, the GGally package was used.

```r
# Scatter plot matrix
library(GGally)
ggpairs(DiabetesData, lower = list(continuous='smooth'))
```



The correlations between explanatory variables do not seem too strong. Hence the model is not likely to suffer from multicollinearity. All explanatory variable are correlated with the Outcome. At first sight, glucose rate is the most important factor to detect the outcome.

6.3 Logistic regression with R

After variable exploration, a first model can be fitted using the glm function. With stargazer, it is easy to get nice output in ASCII or even Latex.

```
# first model: all features
glm1 = glm(Outcome~.,
           DiabetesData,
           family = binomial(link="logit"))
```

```r
summary(glm1)
```

```

Call:
glm(formula = Outcome ~ ., family = binomial(link = "logit"),
    data = DiabetesData)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5566  -0.7274  -0.4159   0.7267   2.9297

Coefficients:
                           Estimate Std. Error z value Pr(>|z|)
(Intercept)              -8.4046964  0.7166359 -11.728  < 2e-16 ***
Pregnancies               0.1231823  0.0320776   3.840 0.000123 ***
Glucose                   0.0351637  0.0037087   9.481  < 2e-16 ***
BloodPressure            -0.0132955  0.0052336  -2.540 0.011072 *
SkinThickness             0.0006190  0.0068994   0.090 0.928515
Insulin                  -0.0011917  0.0009012  -1.322 0.186065
BMI                       0.0897010  0.0150876   5.945 2.76e-09 ***
DiabetesPedigreeFunction  0.9451797  0.2991475   3.160 0.001580 **
Age                       0.0148690  0.0093348   1.593 0.111192
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 993.48  on 767  degrees of freedom
Residual deviance: 723.45  on 759  degrees of freedom
AIC: 741.45

Number of Fisher Scoring iterations: 5
```

```r
require(stargazer)
stargazer(glm1,type='text')
```

```

====================================================
                             Dependent variable:
                         ---------------------------
                                   Outcome
----------------------------------------------------
Pregnancies                       0.123***
                                   (0.032)

Glucose                           0.035***
                                   (0.004)

BloodPressure                     -0.013**
                                   (0.005)

SkinThickness                       0.001
                                   (0.007)
```

```
Insulin                              -0.001
                                     (0.001)

BMI                                 0.090***
                                     (0.015)

DiabetesPedigreeFunction            0.945***
                                     (0.299)

Age                                   0.015
                                     (0.009)

Constant                           -8.405***
                                     (0.717)

-----------------------------------------------------
Observations                           768
Log Likelihood                      -361.723
Akaike Inf. Crit.                    741.445
=====================================================
Note:                     *p<0.1; **p<0.05; ***p<0.01
```

The overall model is significant. As expected the glucose rate has the lowest p-value of all the variables. However, Age, Insulin and Skin Thickness are not good predictors of Diabetes.

6.4 A second model

Since some variables are not significant, removing them is a good way to improve model robustness. In the second model, SkinThickness, Insulin, and Age are removed.

```r
# second model: selected features
glm2 = glm(Outcome~.,
         data = DiabetesData[,c(1:3,6:7,9), with=F],
         family = binomial(link="logit"))

summary(glm2)
```

```
Call:
glm(formula = Outcome ~ ., family = binomial(link = "logit"),
    data = DiabetesData[, c(1:3, 6:7, 9), with = F])

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.7931  -0.7362  -0.4188   0.7251   2.9555

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)
(Intercept)              -7.954952   0.675823 -11.771  < 2e-16 ***
Pregnancies               0.153492   0.027835   5.514  3.5e-08 ***
Glucose                   0.034658   0.003394  10.213  < 2e-16 ***
BloodPressure            -0.012007   0.005031  -2.387  0.01700 *
BMI                       0.084832   0.014125   6.006  1.9e-09 ***
DiabetesPedigreeFunction  0.910628   0.294027   3.097  0.00195 **
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 993.48  on 767  degrees of freedom
Residual deviance: 728.56  on 762  degrees of freedom
AIC: 740.56

Number of Fisher Scoring iterations: 5
```

6.5 Classification rate and confusion matrix

Now that we have our model, let's access its performance.

```r
# Correctly classified observations
mean((glm2$fitted.values>0.5)==DiabetesData$Outcome)
```

```
[1] 0.7747396
```

Around 77.4% of all observations are correctly classified. Due to class imbalance, we need to go further with a confusion matrix.

```r
###Confusion matrix count
RP=sum((glm2$fitted.values>=0.5)==DiabetesData$Outcome & DiabetesData$Outcome==1)
FP=sum((glm2$fitted.values>=0.5)!=DiabetesData$Outcome & DiabetesData$Outcome==0)
RN=sum((glm2$fitted.values>=0.5)==DiabetesData$Outcome & DiabetesData$Outcome==0)
FN=sum((glm2$fitted.values>=0.5)!=DiabetesData$Outcome & DiabetesData$Outcome==1)
confMat<-matrix(c(RP,FP,FN,RN),ncol = 2)
colnames(confMat)<-c("Pred Diabetes",'Pred no diabetes')
rownames(confMat)<-c("Real Diabetes",'Real no diabetes')
confMat
```

```
                 Pred Diabetes Pred no diabetes
Real Diabetes              154              114
Real no diabetes            59              441
```

The model is good to detect people who do not have diabetes. However, its performance on ill people is not great (only 154 out of 268 have been correctly classified).

You can also get the percentage of Real/False Positive/Negative:

```r
# Confusion matrix proportion
RPR=RP/sum(DiabetesData$Outcome==1)*100
FNR=FN/sum(DiabetesData$Outcome==1)*100
FPR=FP/sum(DiabetesData$Outcome==0)*100
RNR=RN/sum(DiabetesData$Outcome==0)*100
confMat<-matrix(c(RPR,FPR,FNR,RNR),ncol = 2)
colnames(confMat)<-c("Pred Diabetes",'Pred no diabetes')
rownames(confMat)<-c("Real Diabetes",'Real no diabetes')
confMat
```

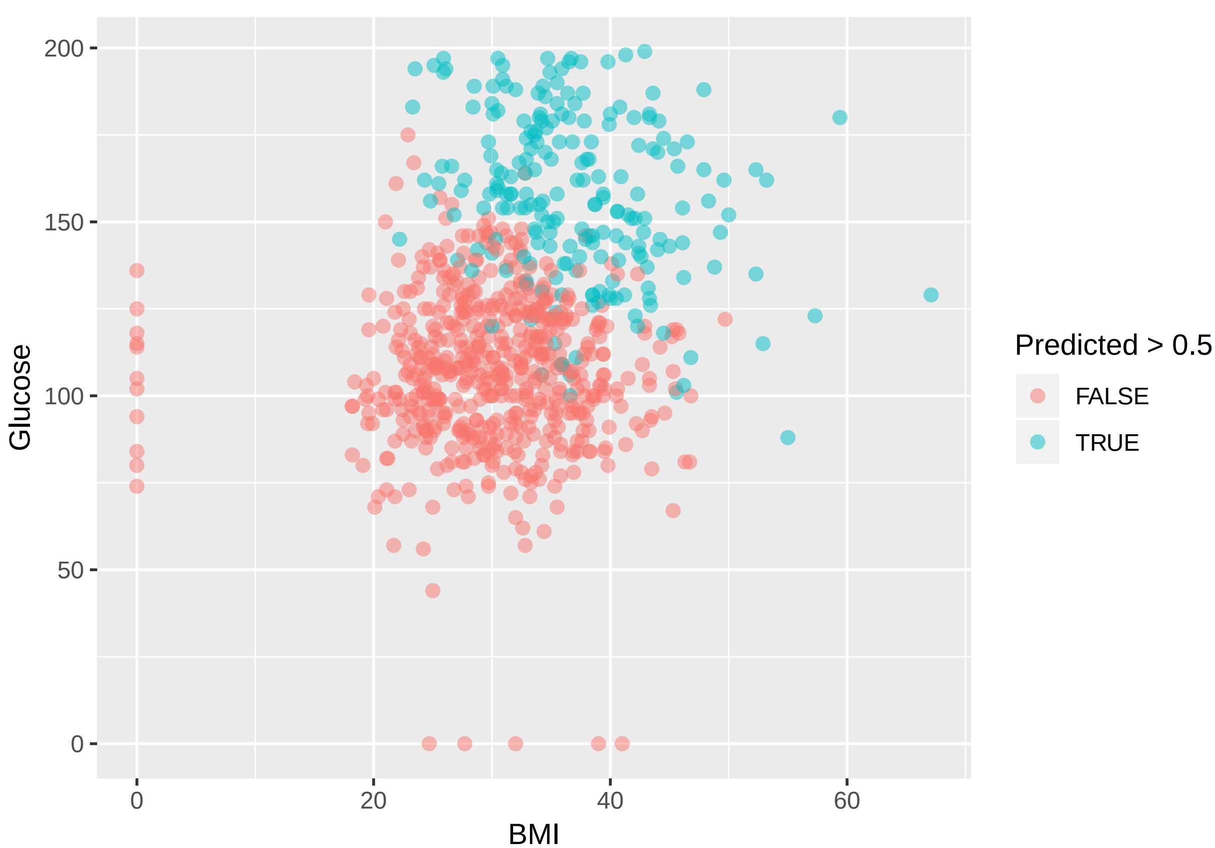
```
                 Pred Diabetes Pred no diabetes
Real Diabetes         57.46269         42.53731
Real no diabetes      11.80000         88.20000
```

And here is the matrix, 57.5% of people with diabetes are correctly classified. A way to improve the false negative rate would lower the detection threshold. However, as a consequence, the false positive rate would increase.

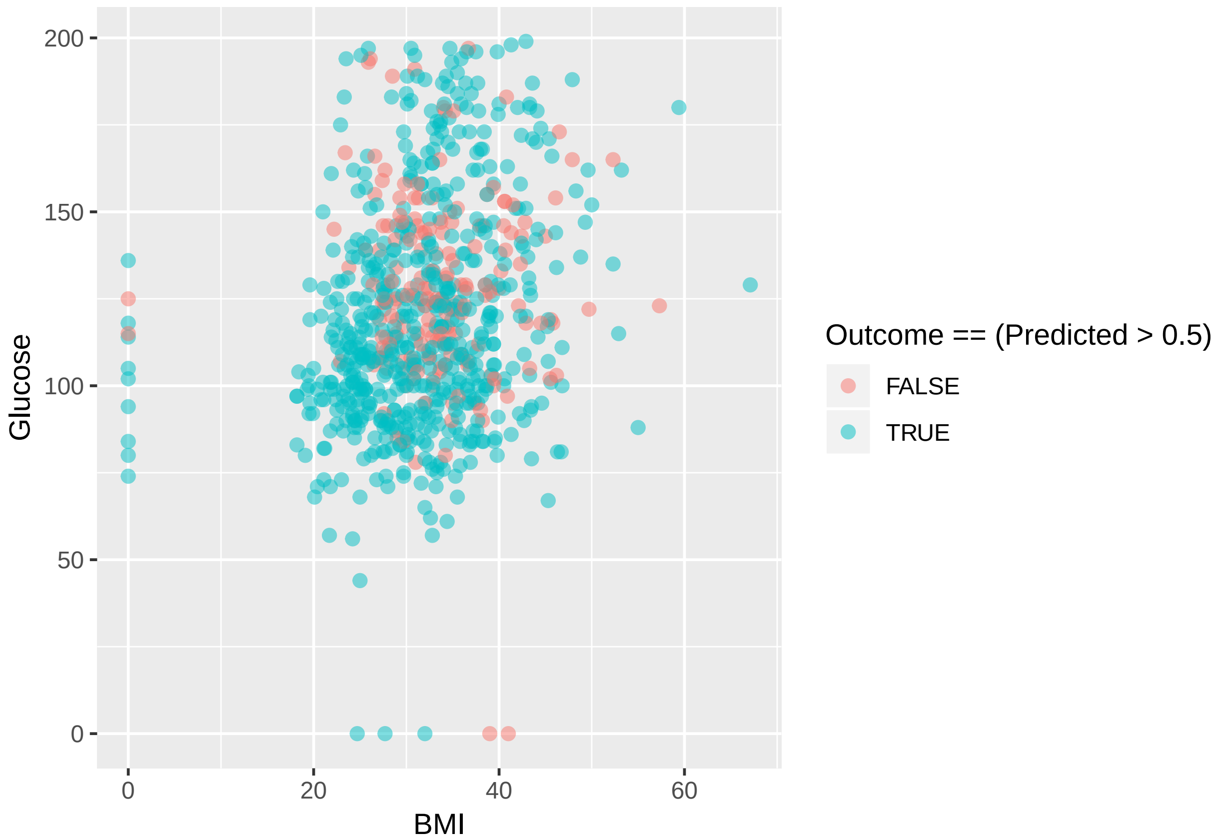
6.6 Plots and decision boundaries

The two strongest predictors of the outcome are Glucose rate and BMI. High glucose rate and high BMI are strong indicators of Diabetes.

```
# Plot and decision boundaries
DiabetesData$Predicted <- glm2$fitted.values

ggplot(DiabetesData, aes(x = BMI, y = Glucose, color = Predicted > 0.5)) +
    geom_point(size=2, alpha=0.5)
```




```
ggplot(DiabetesData, aes(x=BMI, y = Glucose, color=Outcome == (Predicted > 0.5))) +
    geom_point(size=2, alpha=0.5)
```

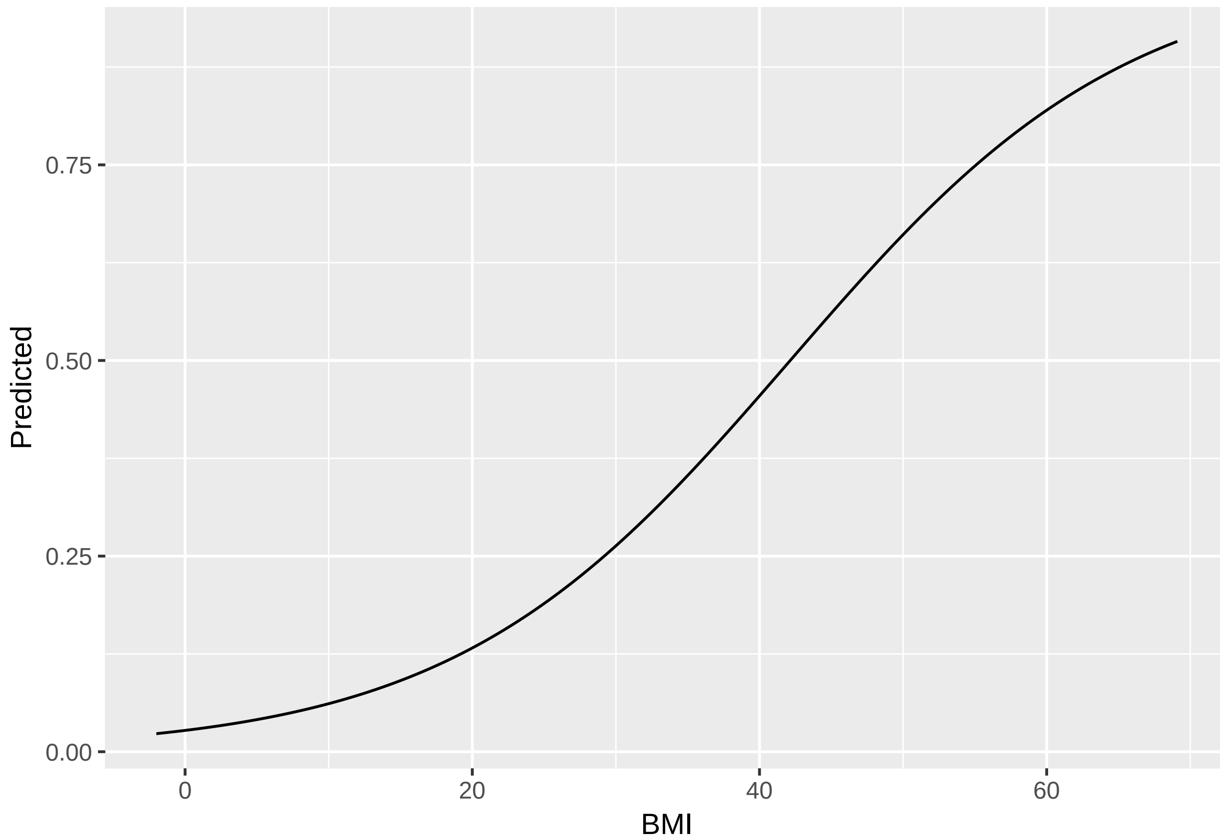


We can also plot both BMI and glucose against the outcomes, the other variables are taken at their mean level.

```
range(DiabetesData$BMI)
```

```
[1]  0.0 67.1
```

```
# BMI vs predicted
BMI_plot = data.frame(BMI = ((min(DiabetesData$BMI-2)*100):
                             (max(DiabetesData$BMI+2)*100))/100,
                      Glucose = mean(DiabetesData$Glucose),
                      Pregnancies = mean(DiabetesData$Pregnancies),
                      BloodPressure = mean(DiabetesData$BloodPressure),
                      DiabetesPedigreeFunction = mean(DiabetesData$DiabetesPedigreeFunction))

BMI_plot$Predicted = predict(glm2, newdata = BMI_plot, type = 'response')
ggplot(BMI_plot, aes(x = BMI, y = Predicted)) +
    geom_line()
```



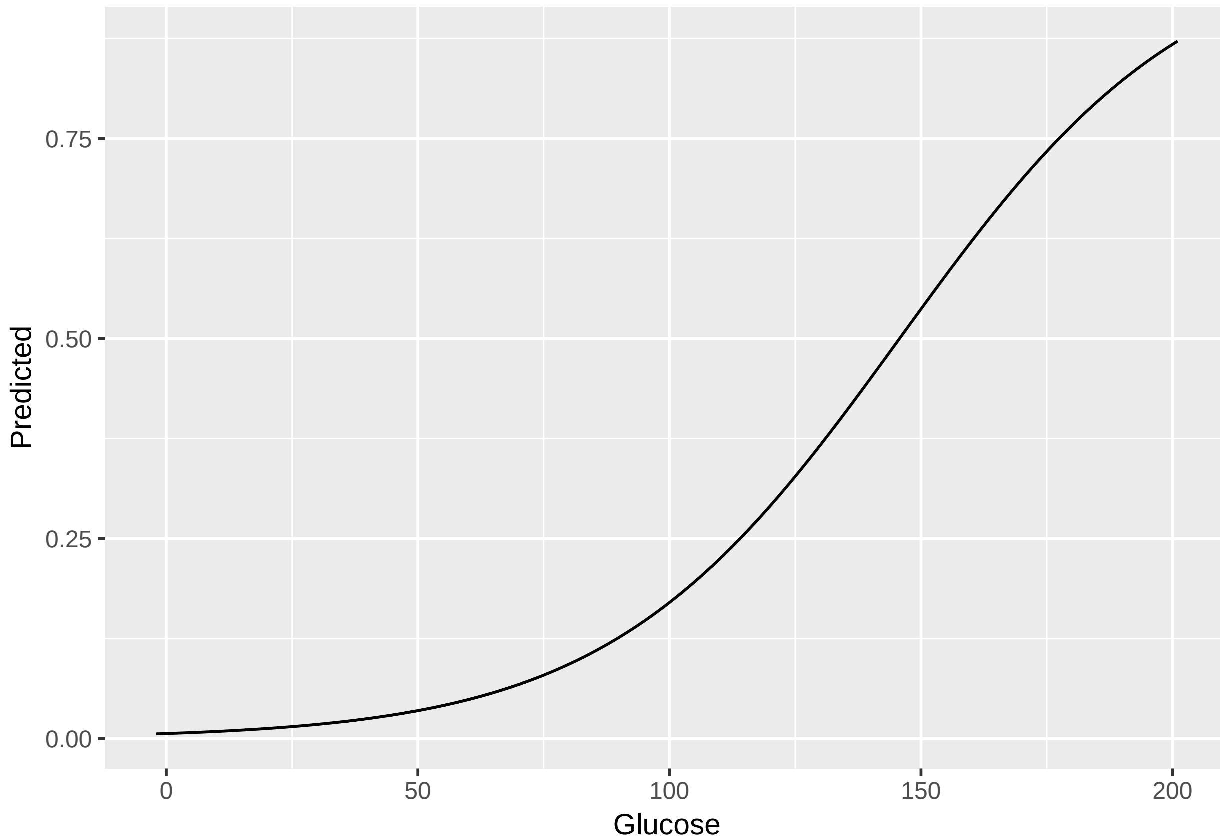

```
range(BMI_plot$BMI)
```

```
[1] -2.0 69.1
```

```
range(DiabetesData$Glucose)
```

```
[1]   0 199
```

```
# Glucose vs predicted
Glucose_plot=data.frame(Glucose =
                        ((min(DiabetesData$Glucose-2)*100):
                             (max(DiabetesData$Glucose+2)*100))/100,
                    BMI=mean(DiabetesData$BMI),
                    Pregnancies=mean(DiabetesData$Pregnancies),
                    BloodPressure=mean(DiabetesData$BloodPressure),
                    DiabetesPedigreeFunction=mean(DiabetesData$DiabetesPedigreeFunction))

Glucose_plot$Predicted = predict(glm2, newdata = Glucose_plot, type = 'response')
ggplot(Glucose_plot, aes(x = Glucose, y = Predicted)) +
    geom_line()
```




```r
range(Glucose_plot$Glucose)
```

```
[1]  -2 201
```