Intoduction to Data Analysis by “Nafiseh Sedaghat”

Missing values

There are two types of missing data:

- MCAR: missing completely at random. This is the desirable scenario in case of missing data.
- MNAR: missing not at random. Missing not at random data is a more serious issue and in this case it might be wise to check the data gathering process further and try to understand why the information is missing. For instance, if most of the people in a survey did not answer a certain question, why did they do that? Was the question unclear?

Loading data:

```
data <- airquality
dim(data)
```

```
## [1] 153   6
```

```
head(data)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41     190  7.4   67     5   1
## 2    36     118  8.0   72     5   2
## 3    12     149 12.6   74     5   3
## 4    18     313 11.5   62     5   4
## 5    NA      NA 14.3   56     5   5
## 6    28      NA 14.9   66     5   6
```

Checking for features (columns) and samples (rows) to see how amount of data is missing

```
pMiss <- function(x){sum(is.na(x))/length(x)*100}
print("Amount of missing values in features:")
```

```
## [1] "Amount of missing values in features:"
```

```
apply(data,2,pMiss)
```

```
##     Ozone   Solar.R      Wind      Temp     Month       Day
## 24.183007  4.575163  0.000000  0.000000  0.000000  0.000000
```

```
cat("\n") # blank line
```

```
print("Amount of missing values in samples:")
```

```
## [1] "Amount of missing values in samples:"
```

```
apply(data,1,pMiss)
```

```
##   [1]  0.00000  0.00000  0.00000  0.00000 33.33333 16.66667  0.00000
##   [8]  0.00000  0.00000 16.66667 16.66667  0.00000  0.00000  0.00000
##  [15]  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
##  [22]  0.00000  0.00000  0.00000 16.66667 16.66667 33.33333  0.00000
##  [29]  0.00000  0.00000  0.00000 16.66667 16.66667 16.66667 16.66667
##  [36] 16.66667 16.66667  0.00000 16.66667  0.00000  0.00000 16.66667
##  [43] 16.66667  0.00000 16.66667 16.66667  0.00000  0.00000  0.00000
##  [50]  0.00000  0.00000 16.66667 16.66667 16.66667 16.66667 16.66667
##  [57] 16.66667 16.66667 16.66667 16.66667 16.66667  0.00000  0.00000
##  [64]  0.00000 16.66667  0.00000  0.00000  0.00000  0.00000  0.00000
##  [71]  0.00000 16.66667  0.00000  0.00000 16.66667  0.00000  0.00000
##  [78]  0.00000  0.00000  0.00000  0.00000  0.00000 16.66667 16.66667
##  [85]  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
##  [92]  0.00000  0.00000  0.00000  0.00000 16.66667 16.66667 16.66667
##  [99]  0.00000  0.00000  0.00000 16.66667 16.66667  0.00000  0.00000
## [106]  0.00000 16.66667  0.00000  0.00000  0.00000  0.00000  0.00000
## [113]  0.00000  0.00000 16.66667  0.00000  0.00000  0.00000 16.66667
## [120]  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
## [127]  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
## [134]  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
## [141]  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000  0.00000
## [148]  0.00000  0.00000 16.66667  0.00000  0.00000  0.00000
```

Here, we are going to omit or impute missing values using a the airquality dataset (available in R).

Detecting missing values:

```
complete.cases(data)
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
##  [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE FALSE  TRUE
##  [45] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
##  [67]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
##  [78]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
##  [89]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
## [100]  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
## [111]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [122]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [144]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
```

Omitting missing values

- The resulting logical can be used to remove incomplete records from the matrix/data.frame.
- Alternatively the na.omit function, does the same.

```
ind <- complete.cases(data)
newData <- data[ind,]
dim(data)
```

```
## [1] 153   6
```

```
dim(newData)
```

```
## [1] 111   6
```

Handling missing values usin mice R package

```
library(mice)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'mice'
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
mice.obj <- mice(data, m=3, maxit=10, meth='pmm', seed=500)
```

```
##
##  iter imp variable
##   1   1  Ozone  Solar.R
##   1   2  Ozone  Solar.R
##   1   3  Ozone  Solar.R
##   2   1  Ozone  Solar.R
##   2   2  Ozone  Solar.R
##   2   3  Ozone  Solar.R
##   3   1  Ozone  Solar.R
##   3   2  Ozone  Solar.R
##   3   3  Ozone  Solar.R
##   4   1  Ozone  Solar.R
##   4   2  Ozone  Solar.R
##   4   3  Ozone  Solar.R
##   5   1  Ozone  Solar.R
##   5   2  Ozone  Solar.R
##   5   3  Ozone  Solar.R
##   6   1  Ozone  Solar.R
##   6   2  Ozone  Solar.R
##   6   3  Ozone  Solar.R
##   7   1  Ozone  Solar.R
##   7   2  Ozone  Solar.R
##   7   3  Ozone  Solar.R
##   8   1  Ozone  Solar.R
##   8   2  Ozone  Solar.R
##   8   3  Ozone  Solar.R
##   9   1  Ozone  Solar.R
##   9   2  Ozone  Solar.R
##   9   3  Ozone  Solar.R
##   10   1  Ozone  Solar.R
##   10   2  Ozone  Solar.R
##   10   3  Ozone  Solar.R
```

```
newData <- complete(mice.obj)
dim(data)
```

```
## [1] 153   6
```

```
dim(newData)
```

```
## [1] 153   6
```

```
anyNA(data)
```

```
## [1] TRUE
```

```
anyNA(newData)
```

```
## [1] FALSE
```

For more information about imputation using `mice' package please see: https://datascienceplus.com/imputing-missing-data-with-r-mice-package/ (https://datascienceplus.com/imputing-missing-data-with-r-mice-package/)

Handling missing values using caret R package

The caret package (short for Classification And REgression Training) contains functions to streamline the model training process for complex regression and classification problems.

Create the knn imputation model on the data:

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
preProcess_missingdata_model <- preProcess(data, method='knnImpute')
preProcess_missingdata_model
```

```
## Created from 111 samples and 6 variables
##
## Pre-processing:
##   - centered (6)
##   - ignored (0)
##   - 5 nearest neighbor imputation (6)
##   - scaled (6)
```

```
# Use the imputation model to predict the values of missing data points
library(RANN)  # required for knnInpute
newData <- predict(preProcess_missingdata_model, newdata = data)
anyNA(newData)
```

```
## [1] FALSE
```

```
dim(newData)
```

```
## [1] 153   6
```

Feature Selection

Contains functions for the data analysis with the emphasis on biological data, including several algorithms for feature ranking, feature selection, classification algorithms with the embedded validation procedures.

The functions can deal with numerical as well as with nominal features.

```
set.seed(7)
# load the library
library(mlbench)
library(Biocomb)
```

```
## Loading required package: gtools
```

```
## Loading required package: Rcpp
```

```
# load the data
data(PimaIndiansDiabetes)
dim(PimaIndiansDiabetes)
```

```
## [1] 768   9
```

```
head(PimaIndiansDiabetes)
```

```
##   pregnant glucose pressure triceps insulin mass pedigree age diabetes
## 1        6     148       72      35       0 33.6    0.627  50      pos
## 2        1      85       66      29       0 26.6    0.351  31      neg
## 3        8     183       64       0       0 23.3    0.672  32      pos
## 4        1      89       66      23      94 28.1    0.167  21      neg
## 5        0     137       40      35     168 43.1    2.288  33      pos
## 6        5     116       74       0       0 25.6    0.201  30      neg
```

```
cat("\n")
```

```
print("The selected features are:")
```

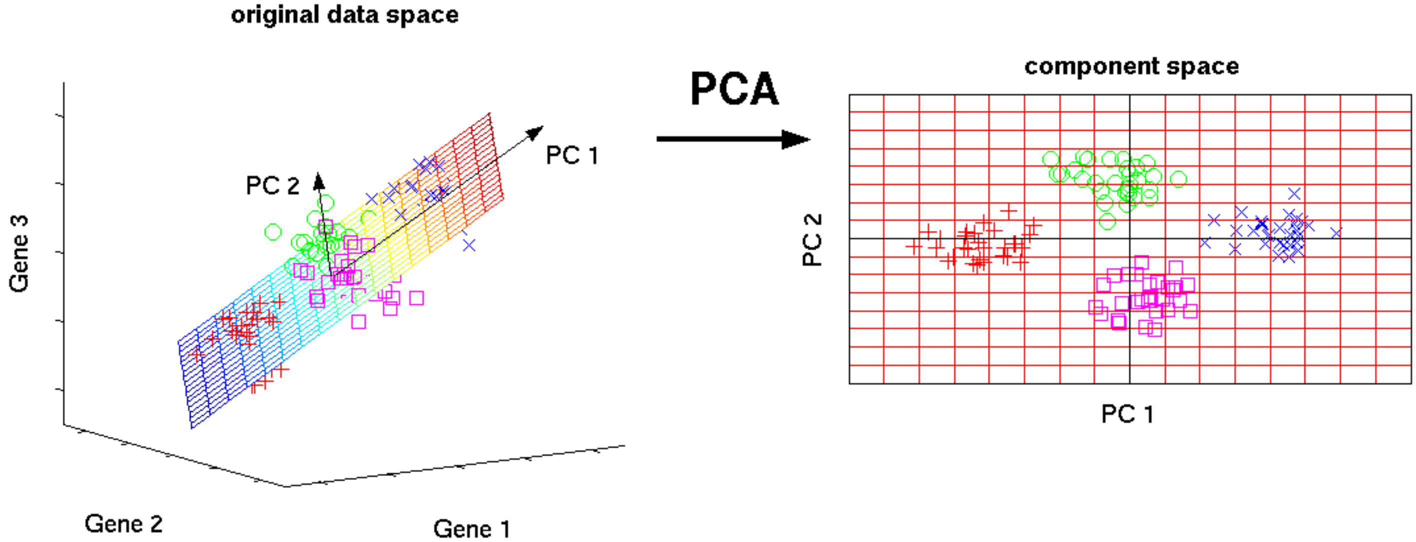
```
## [1] "The selected features are:"
```

```
select.cfs(PimaIndiansDiabetes)
```

```
##         Biomarker Index
## glucose   glucose     2
## mass         mass     6
## age           age     8
```

Feature Extraction

One of the well-known methods to extract features is Principal Component Analysis (PCA). ***

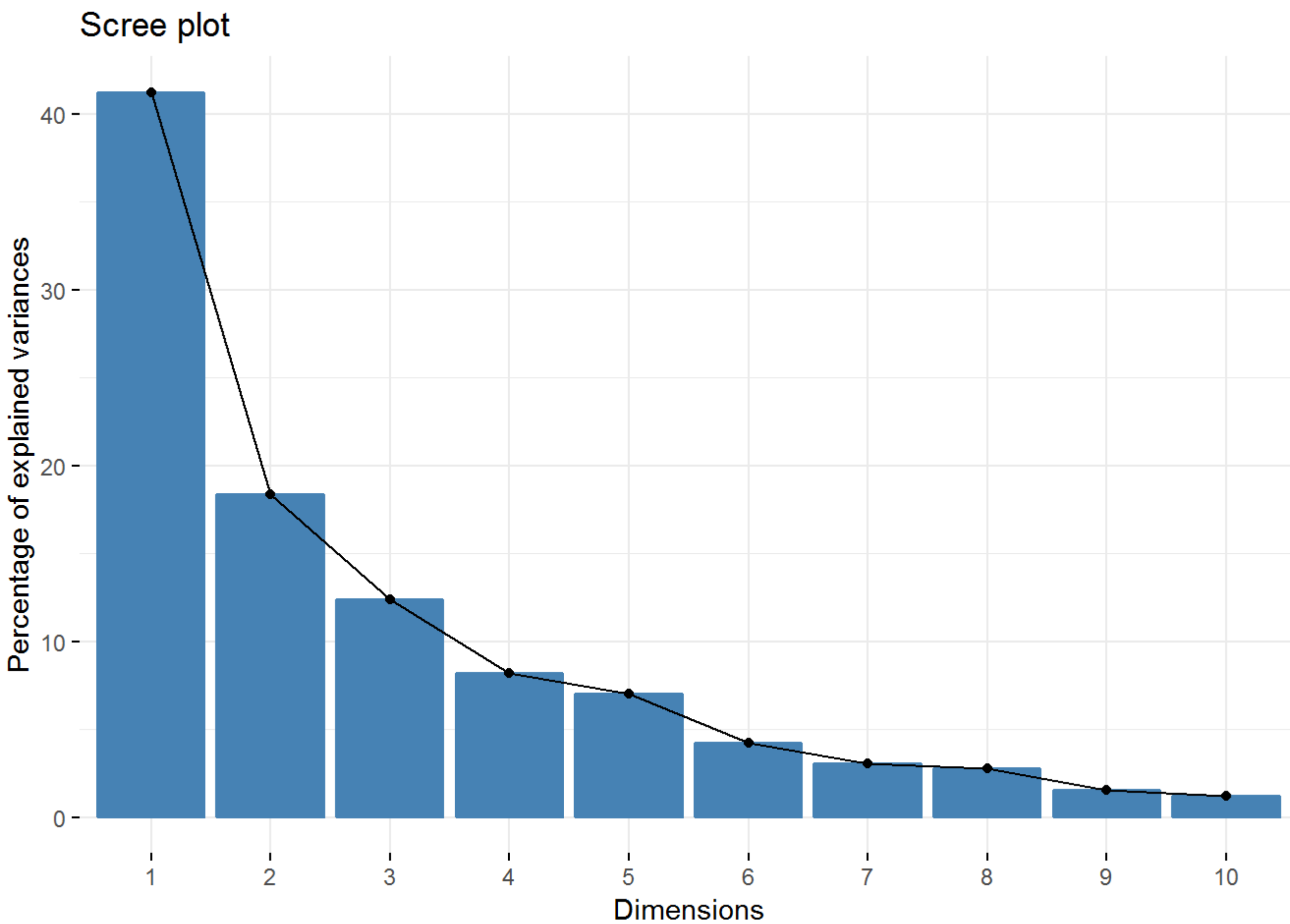


For more information please see: http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-analysis-in-r-prcomp-vs-princomp/ (http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/118-principal-component-analysis-in-r-prcomp-vs-princomp/)

```
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
```

```
res.pca <- prcomp(decathlon2.active, scale = TRUE)
fviz_eig(res.pca)
```




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

```
##                   Dim.1       Dim.2      Dim.3       Dim.4      Dim.5
## X100m        -0.8506257  0.17939806 -0.3015564  0.03357320 -0.1944440
## Long.jump     0.7941806 -0.28085695  0.1905465 -0.11538956  0.2331567
## Shot.put      0.7339127 -0.08540412 -0.5175978  0.12846837 -0.2488129
## High.jump     0.6100840  0.46521415 -0.3300852  0.14455012  0.4027002
## X400m        -0.7016034 -0.29017826 -0.2835329  0.43082552  0.1039085
## X110m.hurdle -0.7641252  0.02474081 -0.4488873 -0.01689589  0.2242200
##                     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
##                   Dim.10
## X100m         0.04441797
## Long.jump     0.02971945
## Shot.put      0.21745195
## High.jump    -0.13356677
## X400m        -0.03417067
## X110m.hurdle -0.01562991
```

```r
head(res.var$contrib)        # Contributions to the PCs
```

```
##                 Dim.1      Dim.2     Dim.3       Dim.4     Dim.5
## X100m        17.544293  1.7505098  7.338659  0.13755240  5.389252
## Long.jump    15.293168  4.2904162  2.930094  1.62485936  7.748815
## Shot.put     13.060137  0.3967224 21.620432  2.01407269  8.824401
## High.jump     9.024811 11.7715838  8.792888  2.54987951 23.115504
## X400m        11.935544  4.5799296  6.487636 22.65090599  1.539012
## X110m.hurdle 14.157544  0.0332933 16.261261  0.03483735  7.166193
##                    Dim.6       Dim.7       Dim.8     Dim.9     Dim.10
## X100m         0.295915322  2.75705260  3.99520353 59.174001  1.6175614
## Long.jump     0.269003613  7.87159392 57.53322220  1.714683  0.7241439
## Shot.put     13.596858744  0.03230371  0.21618512  1.471201 38.7676858
## High.jump    19.159607001  0.26202607  2.59565787  8.101552 14.6264909
## X400m         0.574509906 27.05274658 19.87344405  4.348967  0.9573050
## X110m.hurdle  0.001638634 45.26163460  0.02537025 16.857939  0.2002887
```

Model Training and evaluation

The function createDataPartition can be used to create a stratified random sample of the data into training and test sets:

```r
library(mlbench)
data(Sonar)
library(caret)
set.seed(998)
inTraining <- createDataPartition(Sonar$Class, p = .75, list = FALSE)
training <- Sonar[ inTraining,]
testing  <- Sonar[-inTraining,]
```

Here, we train a random forest model and evaluate its performance using 5-fold cross-validation.

```r
rf_model<-train(Class~., data=training, method="rf", # Model is random forest
                trControl=trainControl(method="cv",number=5)) # 5-fold cross-valuation
print(rf_model)
```

```
## Random Forest
##
## 157 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 126, 125, 125, 127, 125
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8279704  0.6498436
##   31    0.7965188  0.5892971
##   60    0.7771505  0.5498401
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
print(rf_model$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 15.29%
## Confusion matrix:
##    M  R class.error
## M 78  6  0.07142857
## R 18 55  0.24657534
```

Evaluating trained model on test data that is independent from training data.

```
# predict the outcome on a test data
rf_pred <- predict(rf_model, testing)
# compare predicted outcome and true outcome
confusionMatrix(rf_pred, testing[,'Class'])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M  R
##          M 26  5
##          R  1 19
##
##                Accuracy : 0.8824
##                  95% CI : (0.7613, 0.9556)
##     No Information Rate : 0.5294
##     P-Value [Acc > NIR] : 8.488e-08
##
##                   Kappa : 0.7617
##  Mcnemar's Test P-Value : 0.2207
##
##             Sensitivity : 0.9630
##             Specificity : 0.7917
##          Pos Pred Value : 0.8387
##          Neg Pred Value : 0.9500
##              Prevalence : 0.5294
##          Detection Rate : 0.5098
##    Detection Prevalence : 0.6078
##       Balanced Accuracy : 0.8773
##
##        'Positive' Class : M
##
```

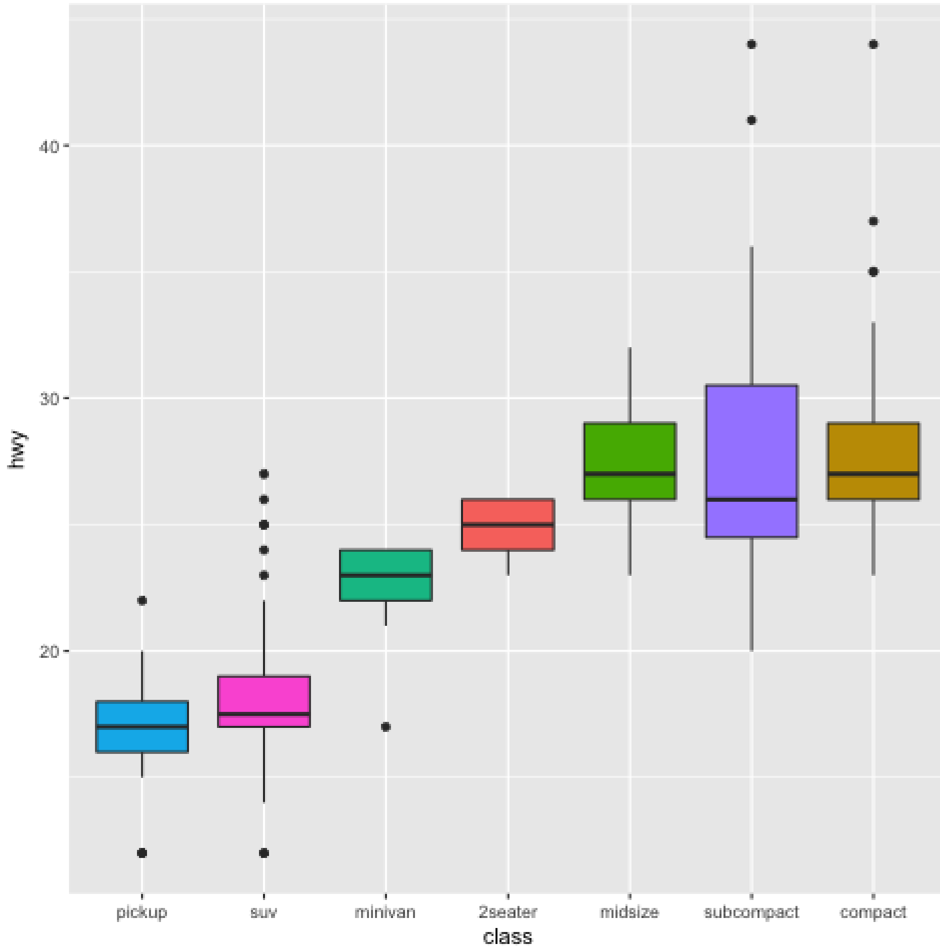
If we still have time…

Let’s talk about outliers.

Outliers

A general definition:

- An outlier in a data set is an observation (or set of observations) which appear to be inconsistent with that set of data.
- Outliers do not equal errors. They should be detected, but not necessarily removed. Their inclusion in the analysis is a statistical decision.



Outliers in a sample box-plot

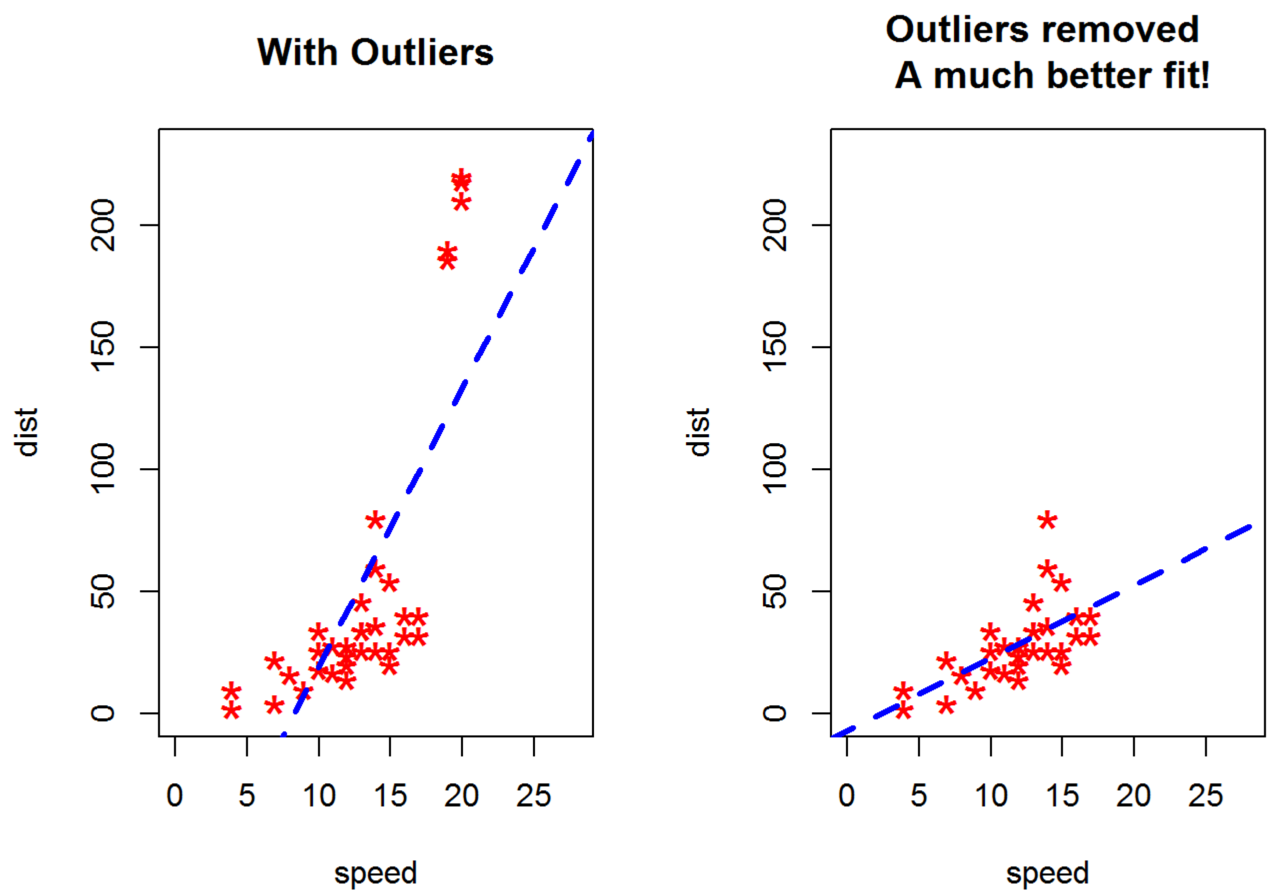
```
x <- c(1:10, 20, 30)
boxplot.stats(x)$out
```

```
## [1] 20 30
```

Inject outliers into data

```
# Inject outliers into data.
cars1 <- cars[1:30, ]  # original data
cars_outliers <- data.frame(speed=c(19,19,20,20,20), dist=c(190, 186, 210, 220, 218))  # introdu
ce outliers.
cars2 <- rbind(cars1, cars_outliers)  # data with outliers.
```

See the ablines in these plots:



Outlier detection

The ``outliers" package provides a number of useful functions to systematically extract outliers. One of these functions is outlier().

```
library(outliers)
set.seed(1234)
y <- rnorm(16)
print("Vector y is:")
```

```
## [1] "Vector y is:"
```

```
cat("\n")
```

```
y
```

```
##  [1] -1.20706575  0.27742924  1.08444118 -2.34569770  0.42912469
##  [6]  0.50605589 -0.57473996 -0.54663186 -0.56445200 -0.89003783
## [11] -0.47719270 -0.99838644 -0.77625389  0.06445882  0.95949406
## [16] -0.11028549
```

```r
cat("\n")
```

```r
print("The outlier in y is:")
```

```
## [1] "The outlier in y is:"
```

```r
outlier(y)
```

```
## [1] -2.345698
```

```r
cat("\n")
```

```r
print("Removing the outlier from y:")
```

```
## [1] "Removing the outlier from y:"
```

```r
rm.outlier(y)
```

```
##  [1] -1.20706575  0.27742924  1.08444118  0.42912469  0.50605589
##  [6] -0.57473996 -0.54663186 -0.56445200 -0.89003783 -0.47719270
## [11] -0.99838644 -0.77625389  0.06445882  0.95949406 -0.11028549
```