~~~
---
title: "STA5077Z Assignment"
author: "Blake Cuningham CNNBLA001"
output:
  pdf_document:
    toc: true
    toc_depth: 3
    number_sections: true
  html_notebook: default
  html_document: default

bibliography: library.bib
---

```{r load_libraries, eval=T, echo=F, message=FALSE}
##################################################
# load required libraries and saved data objects
##################################################
rm(list = ls())

list.of.packages <- c(
                      "MASS",
                      "caret",
                      "ggplot2",
                      "reshape2",
                      "cluster",
                      # "xlsx",
                      "data.table",
                      # "XLConnect",
                      "openxlsx",
                      "tidyverse",
                      # "tsne",
                      "Rtsne",
                      "smacof",
                      "kohonen",
                      "captioner"
                      )

new.packages <- list.of.packages[!(list.of.packages %in%
installed.packages()[,"Package"])]

if (length(new.packages)>=1){
  for (i in 1:length(new.packages)){
    install.packages(new.packages[i])
  }
}


for (i in 1:length(list.of.packages)){
  suppressWarnings(library(list.of.packages[i], character.only = T))
}

rm(list = c("i","list.of.packages", "new.packages"))

# setwd("E:\\DataScienceData\\SL_Assignment")
setwd("/Users/blakecuningham/Dropbox
(Personal)/MScDataScience/Unsupervised Learning/Assignment")

leuk <- readRDS("leuk.rds")

tu_tsne_frame <- readRDS("tu_tsne_frame.rds")
tu_model_matrix_scaled_dist_out_classic <-
readRDS("tu_model_matrix_scaled_dist_out_classic.rds")
tu_model_matrix_scaled_dist_out_sammon <-
readRDS("tu_model_matrix_scaled_dist_out_sammon.rds")
tu_model_matrix_scaled_dist_out_smacofM <-
readRDS("tu_model_matrix_scaled_dist_out_smacofM.rds")
tu_model_matrix_scaled_dist_out_smacofNM <-
readRDS("tu_model_matrix_scaled_dist_out_smacofNM.rds")
tu_model_matrix_scaled_dist_out_Kruskal <-
readRDS("tu_model_matrix_scaled_dist_out_Kruskal.rds")

leuk_transformed_cols_range01 <-
readRDS("leuk_transformed_cols_range01.rds")
leuk_transformed_full_range01 <-
readRDS("leuk_transformed_full_range01.rds")
leuk_transformed_cols_centeredscaled <-
readRDS("leuk_transformed_cols_centeredscaled.rds")
leuk_transformed_log <- readRDS("leuk_transformed_log.rds")
leuk_transformed_log_cols_range01 <-
readRDS("leuk_transformed_log_cols_range01.rds")
leuk_transformed_log_cols_centeredscaled <-
readRDS("leuk_transformed_log_cols_centeredscaled.rds")
list_scaled <- readRDS("list_scaled.rds")
list_dist <- readRDS("list_dist.rds")
list_top100_scaled <- readRDS("list_top100_scaled.rds")
list_top100_dist <- readRDS("list_top100_dist.rds")

###Q2

tu_cars2 <- readRDS("tu_cars2.rds")
tu_model_matrix <- readRDS("tu_model_matrix.rds")

~~~

```{r initfigs, echo=F, eval=T}
##################################################
# initialise captions
##################################################
figs <- captioner(prefix="Figure")
tbls <- captioner(prefix="Table")
```

# Project 1: Leukemia dataset
## Introduction

The Leukemia data-set used in this project provides an interesting
challenge - the number of variables far exceeds the number of
```
```

observations. For this reason, there is a need to employ dimensionality reduction. Specifically, principal component analysis is investigated in order to test the impact of using less noisy data, but also to identify a limited set of "top" variables.

The efficacy of using the top variables only in the principal component analysis is tested by visually comparing labeled bi-plots of the 16 observations in order to observe the separability of the data. It is shown that using the top 100 variables (gene expression levels in this case) does not improve this separability, but does not significantly worsen it.

Finally, we move beyond visual inspection to test how well our data clusters into two groups - hopefully matching what we know about the "good" and "poor" labels of our observations. Both agglomerative and K-means clustering approaches are conducted on the full data, and the top 100 gene data. For each set of data, six different scaling methods are used as input. Again, we see similar performance for the top 100 gene data (as was the case with PCA and visual inspection), but we also observe performance differences of the clustering approaches and the types of scaling. The performance is measured by observing the best possible accuracy from a confusion matrix of the assigned clusters and the known labels (e.g. the higher of 40% and 60% would be 60%). The key observations are:

* Clustering techniques: K-means generally performs better, and never worse, than agglomerative clustering.
* Scaling techniques: No particular technique performed best overall, but versions of log transforms of the data were consistently good performers.
* Data completeness: Using the full data as input we were able to find relatively accurate clusters regardless of clustering approach. Using the top 100 genes performed similarly, but consistently well for all scaling techniques under K-means clustering. The PCA data performed well with K-means clustering.

## PCA analysis

~~~
```{r, eval=F, echo=F}

##################################################
# prepare leukemia dataframe
##################################################

leuk <- read.table("leukemia_array.txt")
leuk <- t(leuk)
leuk <- data.frame(leuk)

leuk$class <- row.names(leuk)
leuk$class <- sapply(strsplit(as.character(leuk$class),'_'), "[", 2)
leuk <- leuk[,c(ncol(leuk),1:(ncol(leuk)-1))]

saveRDS(leuk, "leuk.rds")
```

```{r, eval=F, echo=F}
##################################################
# transform full leukemia data and save objects
##################################################

#Tranformation: range between 0 and 1
leuk_transformed_cols_range01 <- cbind("class" = leuk$class,
data.frame(apply(leuk[,-1], MARGIN = 2, FUN = function(X) (X -
min(X))/diff(range(X)))))

#Tranformation: range between 0 and 1 for range of full matrix
leuk_transformed_full_range01 <- cbind("class" = leuk$class,
data.frame((leuk[,-1] - min(leuk[,-1])) / diff(range(leuk[,-1]))))

#Tranformation: center and divide by sd
leuk_transformed_cols_centeredscaled <- cbind("class" = leuk$class,
data.frame(apply(leuk[,-1], MARGIN = 2, FUN = function(X) (X -
mean(X)) / sd(X))))

#Tranformation: log transform
leuk_transformed_log <- cbind("class" = leuk$class,
data.frame(apply(leuk[,-1], MARGIN = 2, FUN = function(X) log(X))))

#Tranformation: range between 0 and 1 of log tranform
leuk_transformed_log_cols_range01 <- cbind("class" = leuk$class,
data.frame(apply(leuk[,-1], MARGIN = 2, FUN = function(X) (log(X) -
min(log(X)))/diff(range(log(X))))))

#Tranformation: center and divide by sd of log transform
leuk_transformed_log_cols_centeredscaled <- cbind("class" =
leuk$class, data.frame(apply(leuk[,-1], MARGIN = 2, FUN = function(X)
(log(X) - mean(log(X))) / sd(log(X)))))

saveRDS(leuk_transformed_cols_range01,
"leuk_transformed_cols_range01.rds")
saveRDS(leuk_transformed_full_range01,
"leuk_transformed_full_range01.rds")
saveRDS(leuk_transformed_cols_centeredscaled,
"leuk_transformed_cols_centeredscaled.rds")
saveRDS(leuk_transformed_log, "leuk_transformed_log.rds")
saveRDS(leuk_transformed_log_cols_range01,
"leuk_transformed_log_cols_range01.rds")
saveRDS(leuk_transformed_log_cols_centeredscaled,
"leuk_transformed_log_cols_centeredscaled.rds")

list_scaled <- list("Range 0-1 per column" =
leuk_transformed_cols_range01,
                    "Range 0-1 for all data" =
leuk_transformed_full_range01,
                    "Centred and scaled per column" =
leuk_transformed_cols_centeredscaled,
                    "Log transform" = leuk_transformed_log,
                    "Log transform and Range 0-1 per column" =
~~~

~~~~
leuk_transformed_log_cols_range01,
                     "Log transform and centred and scaled per columns"
= leuk_transformed_log_cols_centeredscaled)

saveRDS(list_scaled, "list_scaled.rds")

```


## Principal component analysis
```{r, eval=T, echo=F}
##################################################
# Get PC's of full leukemia data
##################################################


#PCA
set.seed(7)
leuk_pca_model <- prcomp(leuk_transformed_cols_centeredscaled[,-1])
leuk_pca <- cbind("class" = leuk$class, data.frame(leuk_pca_model$x))

```

Principal component analysis is able to systematically find vectors
within the data that have the highest variance (and thus explain the
most variation), and are orthogonal to other principal component
vectors. The number of principal components is the lower of the the
number of observations, or the number of variables. For this exercise
the following method was used to find principal components:

* "prcomp" package used from "stats" library in R
* Only the centered and scaled data was used (none of the other
scaling methods), because it's critical that each variable have a mean
of 0 [@James2013]. The centered and scaled log transform data may be
interesting to observe in future investigations.

There does appear to be some grouping between the two kinds of
observations ("good" and "poor") when reviewing the first two
components:

```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}

ggplot()+
  geom_point(data = leuk_pca, aes(x = PC1, y = PC2, color = class)) +
  scale_color_discrete(name = "Class")+
  labs(title = "First two principal components of data") +
  theme_light()

```


*`r figs(name="full_pca_2pc","First two principal components trained
from full dataset")`*

Approximately 50% of the variance is captured in these first two
components:
~~~~

~~~~
```{r, eval=T, echo=F}
plot(leuk_pca_model$sdev^2 / ncol(leuk), type = "b",
     main = "Proportion of variance explained by principal
components",
     xlab = "Principal components",
     ylab = "Proportion of variance explained")

```


*`r figs(name="full_pca_var","Proportion of variance explained by
principal components")`*

##Using PCA results to identify the top 100 genes
```{r, eval=T, echo=F}

##################################################
# Identify top 100 genes, and create new dataframe
##################################################

leuk_squared_loadings <- data.frame(leuk_pca_model$rotation ^ 2)


leuk_sl_PC1 <- data.frame("PC1" = leuk_squared_loadings$PC1)
row.names(leuk_sl_PC1) <- row.names(leuk_squared_loadings)
leuk_sl_PC1 <- leuk_sl_PC1[order(leuk_sl_PC1$PC1, decreasing = T), ,
drop = F]
top100_genes <- row.names(leuk_sl_PC1[1:100, , drop = F])

leuk_top100only <- leuk[,c("class", top100_genes)]

```


The method used to find the top 100 genes from the PCA output is as
follows:

1. Retrieve the loadings (rotation matrix) and square all values
2. Discard all PC's and keep the first only
3. Sort these squared values in descending order and identify the top
100

Because the most influential variables will have the highest absolute
loadings on the principal components, the squared loadings will
represent the most influential variables. Because the first principal
component contains the most information, the variables that have the
highest squared loadings therefore represent the variables that
contribute the most to principal component with the most information
and are subsequently considered the most important.

The top five genes are: X204365_s_at, X206766_at, X221870_at,
X204542_at, X215356_at

```{r, eval=F, echo=F}
##################################################
# transformations on top 100 gene data, and saving objects
~~~~

~~~
# + create list object of all transformations
##################################################

#Tranformation: range between 0 and 1
leuk_top100_transformed_cols_range01 <- cbind("class" =
leuk_top100only$class, data.frame(apply(leuk_top100only[,-1], MARGIN =
2, FUN = function(X) (X - min(X))/diff(range(X)))))

#Tranformation: range between 0 and 1 for range of full matrix
leuk_top100_transformed_full_range01 <- cbind("class" =
leuk_top100only$class, data.frame((leuk_top100only[,-1] -
min(leuk_top100only[,-1])) / diff(range(leuk_top100only[,-1]))))

#Tranformation: center and divide by sd
leuk_top100_transformed_cols_centeredscaled <- cbind("class" =
leuk_top100only$class, data.frame(apply(leuk_top100only[,-1], MARGIN =
2, FUN = function(X) (X - mean(X)) / sd(X))))

#Tranformation: log transform
leuk_top100_transformed_log <- cbind("class" = leuk_top100only$class,
data.frame(apply(leuk_top100only[,-1], MARGIN = 2, FUN = function(X)
log(X))))

#Tranformation: range between 0 and 1 of log tranform
leuk_top100_transformed_log_cols_range01 <- cbind("class" =
leuk_top100only$class, data.frame(apply(leuk_top100only[,-1], MARGIN =
2, FUN = function(X) (log(X) - min(log(X)))/diff(range(log(X))))))

#Tranformation: center and divide by sd of log transform
leuk_top100_transformed_log_cols_centeredscaled <- cbind("class" =
leuk_top100only$class, data.frame(apply(leuk_top100only[,-1], MARGIN =
2, FUN = function(X) (log(X) - mean(log(X))) / sd(log(X)))))


saveRDS(leuk_top100_transformed_cols_range01,
"leuk_top100_transformed_cols_range01.rds")
saveRDS(leuk_top100_transformed_full_range01,
"leuk_top100_transformed_full_range01.rds")
saveRDS(leuk_top100_transformed_cols_centeredscaled,
"leuk_top100_transformed_cols_centeredscaled.rds")
saveRDS(leuk_top100_transformed_log,
"leuk_top100_transformed_log.rds")
saveRDS(leuk_top100_transformed_log_cols_range01,
"leuk_top100_transformed_log_cols_range01.rds")
saveRDS(leuk_top100_transformed_log_cols_centeredscaled,
"leuk_top100_transformed_log_cols_centeredscaled.rds")

list_top100_scaled <- list("Range 0-1 per column" =
leuk_top100_transformed_cols_range01,
                    "Range 0-1 for all data" =
leuk_top100_transformed_full_range01,
                    "Centred and scaled per column" =
leuk_top100_transformed_cols_centeredscaled,
                    "Log transform" = leuk_top100_transformed_log,
                    "Log transform and Range 0-1 per column" =
leuk_top100_transformed_log_cols_range01,
                    "Log transform and centred and scaled per columns"
= leuk_top100_transformed_log_cols_centeredscaled)

saveRDS(list_top100_scaled, "list_top100_scaled.rds")
```

```{r, eval=T, echo=F}
##################################################
# PCA on top 100 gene data
##################################################

#PCA

leuk_pca_model_top100 <- prcomp(list_top100_scaled$`Centred and scaled
per column`[,-1])
leuk_pca_top100 <- cbind("class" = leuk_top100only$class,
data.frame(leuk_pca_model_top100$x))
```
~~~

These top 100 genes result in two distinct groups of observations (plus one far outlier), one of which is entirely made up of "poor" observations:

~~~
```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}

ggplot()+
  geom_point(data = leuk_pca_top100, aes(x = PC1, y = PC2, color =
class)) +
  scale_color_discrete(name = "Class")+
  labs(title = "First two principal components of top 100 genes") +
  theme_light()

```
~~~

`r figs(name="t100_pca_2pc","First two principal components trained from top 100 genes")`

When considering the proportion of variance explained by each principal component, it is not surprising that almost 90% of the variance is explained by PC1 - this is because we chose the variables that had the highest loadings for PC1. Very little information is contained in the other components.

~~~
```{r, eval=T, echo=F}
plot(leuk_pca_model_top100$sdev^2 / ncol(leuk_top100only), type = "b",
     main = "Proportion of variance explained by principal
components",
     xlab = "Principal components",
     ylab = "Proportion of variance explained")

```
~~~

`r figs(name="t100_pca_var","Proportion of variance explained by principal components")`

##Clustering analysis

Next, clustering analysis is performed on the data. Two different kinds of clustering are used:

1. Agglomerative: This is a bottom-up approach that builds up clusters from 16 observations to a single cluster contained all observations. The "tree" is then cut to ensure two clusters. Only the "complete" method was used for this project after some initial testing indicated that it produced better results with more coherent trees.
2. K-means: Two clusters are specified and an iterative approach begins to adjust cluster centers and cluster allocations until the algorithm stabilizes.

Three different inputs were fed to the clustering algorithms:

1. Each of the six scaled full data-sets
2. Each of the six scaled top 100 gene data-sets
3. Principal components from full data-set

### A note on scaling approaches

As mentioned, there were six scaling techniques used. They are as follows:

1. "Range 0-1 per column": The minimum and maximum for each column (variable) is used to transform each column's data to a number between 0 and 1.
2. "Range 0-1 for all data": The minimum and maximum for the whole data-set is used to transform all the data to a number between 0 and 1.
3. "Centered and scaled per column": This is the typical approach where for each column the mean is subtracted in order to center at 0, and then each column's data is divided by the column's standard deviation.
4. "Log transform": The natural log of each data point is used.
5. "Log transform and Range 0-1 per column": The "Range 0-1 per column" method is applied to the "Log transform" data.
6. "Log transform and centered and scaled per columns" The "Centered and scaled per column" method is applied to the "Log transform" data.

### Hierarchical / agglomerative clustering full dataset

~~~~
```{r, eval=F, echo=F}

###################################################
# create list of distance objects
###################################################

k <- 1
list_dist <- list()
for (i in list_scaled){
  dist.leuk <- daisy(i[,-1])
  # assign(paste0(names(list_scaled)[k]), dist.leuk)
  list_dist[[paste0(names(list_scaled)[k])]] <- dist.leuk
  k = k + 1
}

saveRDS(list_dist, "list_dist.rds")

```
~~~~

Most scaled data-sets result in quite a clear smaller branch of "poor" observations, with a larger mixed branch:

~~~~
```{r, eval=T, echo=F, fig.height=10, fig.width=10,
fig.align="center"}

###################################################
# plot 6 clusters per type of scaling
###################################################

k = 1
par(mfrow=c(3,2))
for (i in list_dist){
  # dist.leuk <- daisy(i[,-1])
  hc.leuk <- hclust(i, method = "complete")
  plot(hc.leuk, main = names(list_dist)[k], xlab = "", ylab = "", sub
= "")
  k = k + 1
}

```
~~~~

`r figs(name="full_hier_plots","Dendrograms of six scaling methods")`

These dendrograms can be summarized neatly into confusion matrices. Because this was an unsupervised task, there is no definitive true positive or true negative region - hence we consider the diagonal with the most observations. Only the "log transform" data achieved the maximum classification accuracy of 13/16 (~81%):

`r tbls(name="full_hier_tables","Confusion matrices per scaling method")`

~~~~
```{r, eval=T, echo=F}
###################################################
# print confusion matrices of clusters
###################################################

k = 1
for (i in list_dist){
  # dist.leuk <- daisy(i[,-1])
  hc.leuk <- hclust(i, method = "complete")
  cut.hc.leuk <- cutree(hc.leuk, 2)
  cat(paste0(names(list_dist)[k], ": "))
  print(table("Assigned cluster" = cut.hc.leuk, leuk$class))
  cat("\n")
  k = k + 1
~~~~

~~~~
}

```

### K-means clustering full dataset

The K-means clustering approach was more consistent, achieving 13/16
classification accuracy for all scaling methods except for the "Range
0-1 for all data":

*`r tbls(name="full_k_tables","Confusion matrices per scaling
method")`*

```{r, eval=T, echo=F}
##################################################
# print confusion matrices of K-means clsuters
##################################################

k = 1
for (i in list_scaled){
  # dist.leuk <- daisy(i[,-1])
  km.leuk <- kmeans(i[,-1], 2)
  cat(paste0(names(list_scaled)[k], ": "))
  print(table("Assigned cluster" = km.leuk$cluster, leuk$class))
  cat("\n")
  k = k + 1
}

```

### Hierarchical / agglomerative clustering top 100 genes
```{r, eval=F, echo=F}

k <- 1
list_top100_dist <- list()
for (i in list_top100_scaled){
  dist.leuk <- daisy(i[,-1])
  list_top100_dist[[paste0(names(list_top100_scaled)[k])]] <-
dist.leuk
  k = k + 1
}

saveRDS(list_top100_dist, "list_top100_dist.rds")

```

Using the top 100 genes only, we see similar looking results to that
of using the full data-set - a small branch of "poor" observations,
and a large branch of mixed observations:

```{r, eval=T, echo=F, fig.height=10, fig.width=10,
fig.align="center"}
k = 1
par(mfrow=c(3,2))
for (i in list_top100_dist){
  # dist.leuk <- daisy(i[,-1])
  hc.leuk <- hclust(i, method = "complete")
  plot(hc.leuk, main = names(list_top100_dist)[k], xlab = "", ylab =
"", sub = "")
  k = k + 1
}

```

*`r figs(name="t100_hier_plots","Dendrograms of six scaling
methods")`*

Reviewing the classification accuracy, it appears that a log transform
of the data was very helpful as all scaling methods employing it
performed at a 13/16 accuracy:

*`r tbls(name="t100_hier_tables","Confusion matrices per scaling
method")`*

```{r, eval=T, echo=F}
k = 1
for (i in list_top100_dist){
  # dist.leuk <- daisy(i[,-1])
  hc.leuk <- hclust(i, method = "complete")
  cut.hc.leuk <- cutree(hc.leuk, 2)
  cat(paste0(names(list_top100_dist)[k], ": "))
  print(table("Assigned cluster" = cut.hc.leuk, leuk$class))
  cat("\n")
  k = k + 1
}

```

### K-means clustering top 100 genses

The K-means clustering on the top 100 data achieved 13/16
classification accuracy regardless of which scaling method was used:

*`r tbls(name="t100_k_tables","Confusion matrices per scaling
method")`*

```{r, eval=T, echo=F}

k = 1
for (i in list_top100_scaled){
  # dist.leuk <- daisy(i[,-1])
  km.leuk <- kmeans(i[,-1], 2)
  cat(paste0(names(list_top100_scaled)[k], ": "))
  print(table("Assigned cluster" = km.leuk$cluster, leuk$class))
  cat("\n")
  k = k + 1
}

```
~~~~

~~~~
### Hierarchical / agglomerative clustering with principal component
data

Using the principal component data results in a dendrogram with the
"poor" outlier branch, and then a smaller three observation "poor"
branch within the second branch. This is not as accurate as many
instances of the original data without PCA:

```{r, eval=T, echo=F}

dist.leuk_pca <- daisy(leuk_pca[,-1])
hc.leuk_pca <- hclust(dist.leuk_pca, method = "complete")
plot(hc.leuk_pca, main = "Agglomerative clustering of principal
component data", sub = "", xlab = "")

```

*`r figs(name="pca_hier_plot","Dendrogram of principal component
data")`*

The result is an accuracy of 9/13, which is very likely to occur by
random chance:

*`r tbls(name="pca_hier_table","Confusion matrix of principal
component data")`*

```{r, eval=T, echo=F}

cut.hc.leuk_pca <- cutree(hc.leuk_pca, 2)
table("Assigned cluster" = cut.hc.leuk_pca, leuk$class)
```

### K-means clustering with principal component data

Using the K-means approach the PCA data is able to produce an accuracy
of 13/16. In general, K-means has more consistently been accurate at
the 13/16 level, so it does not seem that PCA was that helpful.

*`r tbls(name="pca_k_table","Confusion matrix of principal component
data")`*

```{r, eval=T, echo=F}

km.leuk_pca <- kmeans(leuk_pca[,-1], 2)
table("Assigned cluster" = km.leuk_pca$cluster, leuk$class)

```

## Conclusion

While no combination of selected input, scaling, or clustering was
able to perform better than 13/16 accuracy, the results are
interesting in that there are many combinations able to classify 5/8
"poor" observations into their own category. This could mean that gene
expression levels may be analysed to help identify "poor" Leukemia
~~~~

~~~~
prognosis with little chance of a false positive, but a fairly high
chance of false negative - i.e. high precision, but a high false
negative rate.


# Project 2: New vehicle dataset

```{r, eval=F, echo=F}
##################################################
# extract and clean transunion data
# + convert to matrix format for modeling
##################################################


# tu_cars <- read.xlsx2("TRANSUNION_LIST_V5.xlsx", sheetIndex = 1)
# tu_cars <- fread("TRANSUNION_LIST_V5.xlsx")
# tu_cars <- readWorksheetFromFile("TRANSUNION_LIST_V5.xlsx", sheet =
1)
tu_cars <- read.xlsx("TRANSUNION_LIST_V5.xlsx", sheet = 1)

tu_cars2 <- tu_cars %>%
  filter(VehicleType == "A" | VehicleType == "B", RegYear == 2016) %>%
  select(Make,
         Model,
         Variant,
         # RegYear,
         AxleConfiguration,
         BodyType,
         NoOfDoors,
         Drive,
         Seats,
         Wheelbase,
         ManualAuto,
         NoGears,
         # Cooling,
         CubicCapacity,
         # EngineCycle,
         FuelTankSize,
         FuelType,
         Kilowatts,
         NoCylinders,
         # TurboOrSuperCharged,
         # GCM,
         # GVM,
         Tare,
         # Origin,
         # FrontNoTyres,
         # RearNoTyres,
         CO2,
         Length,
         Height,
         Width,
         NewListPrice) %>%
  drop_na()
~~~~

```
tu_cars2 <- tu_cars2[!duplicated(tu_cars2[c("Make", "Model",
"Variant")]),]

# #check na per column
# na_count <-sapply(tu_cars2, function(y)
sum(length(which(is.na(y)))))
# na_count <- data.frame(na_count)
# na_count

#change row.name to concatenation


#Rename columsn and remove redundent columns
row.names(tu_cars2) <- paste(tu_cars2$Make, tu_cars2$Model,
tu_cars2$Variant, sep = "-")
tu_cars2 <- tu_cars2[, !(colnames(tu_cars2) %in%
c("Make","Model","Variant"))]

#convert characters to factors
tu_cars2$AxleConfiguration <- as.factor(tu_cars2$AxleConfiguration)
tu_cars2$BodyType <- as.factor(tu_cars2$BodyType)
tu_cars2$Drive <- ifelse(tu_cars2$Drive == "F/R", "FR",
tu_cars2$Drive)
tu_cars2$Drive <- as.factor(tu_cars2$Drive)
tu_cars2$ManualAuto <- as.factor(tu_cars2$ManualAuto)
tu_cars2$FuelType <- as.factor(tu_cars2$FuelType)
# tu_cars2$Origin <- as.factor(tu_cars2$Origin)

tu_cars2 <- unique(tu_cars2)

#get into workable form
tu_model_matrix <- data.frame(model.matrix( ~ . -1, tu_cars2))

saveRDS(tu_cars2, "tu_cars2.rds")
saveRDS(tu_model_matrix, "tu_model_matrix.rds")

```

## Introduction

This section of the assignment makes use of a data-set of motor
vehicles - specifically 2016 models available in South Africa. The
goal of the analysis is to try and identify groups of vehicles which
exhibit similar characteristics, which may be a useful way to
potentially direct someone who was trying to navigate which set of
vehicles to choose from.

The data is originally from Transunion, and has been cleaned to ensure
that:

* Only 2016 vehicles are considered
* There are no duplicates within the data
* Only interesting variables are considered (e.g. Cooling system is
not included, but price is). 34 variables are included in total.
```

```
It is expected that certain groups of vehicles will cluster: e.g.
expensive sports cars in one group; "bakkies" in one group; small
"cheap" cars in another.

## Principal component analysis of data

In order to better understand the data, a principal component analysis
on scaled data is performed. From the bi-plot it can be seen that the
observations are fairly evenly distributed along the first two
components indicating that there could be some distinct clusters
within the data. The proportion of variance explained shows that most
of the variance is explained by the first five components, although
the remaining components are not negligible.

```{r, eval=T, echo=F}

## PCA exploration
tu_model_matrix_scaled <- data.frame(scale(tu_model_matrix))

tu_pca <- prcomp(tu_model_matrix_scaled)

par(mfrow=c(1,2))
plot(tu_pca$x[,1], tu_pca$x[,2],
     main = "Biplot of all vehicles in dataset",
     xlab = "Principal component 1",
     ylab = "Principal component 2",
     cex.main = 0.8)
plot(tu_pca$sdev^2 / ncol(tu_pca$x), type = "b",
     main = "Proportion of variance explained per component",
     xlab = "Principal components",
     ylab = "Proportion of variance explained",
     cex.main = 0.8)
```

`r figs(name="pca_tu_varAndbi","Biplot and variance explained")`

## Clustering using K-means

In order to better observe the representation of the data in a 2-
dimensional space, the data is clustered into 4 groups using K-means
performed on scaled data (centered and adjusted by standard
deviation). Four groups was decided on after some initial
experimentation, and standard scaling was chosen for simplicity.

## Dimensionality reduction
### t-SNE dimensionality reduction

The first dimensionality reduction approach was the fairly modern t-
SNE (t-distributed stochastic neighbor embedding), which is an
approach developed in 2008 by Geoffrey Hinton and Laurens van der
Maaten [@VanDerMaaten2008]. It is similar to other multi-dimensional
scaling approaches, except that it makes use of a probability
distribution in order to select points. The major downside of t-SNE is
that it can sometimes display patterns resulting from random noise -
often multiple plots with different parameters should be analysed
```
```

~~~
before making any expensive decisions based on it.

```{r, eval=F, echo=F}
##################################################
# create clusters and t-SNE dataframe
##################################################

km.tu_1 <- kmeans(tu_model_matrix_scaled, 4)

tu_tsne <- Rtsne(tu_model_matrix_scaled, dims = 2)

tu_tsne_frame <- data.frame(cbind(tu_tsne$Y, km.tu_1$cluster))
colnames(tu_tsne_frame) <- c("X", "Y", "class")
tu_tsne_frame$class <- as.factor(tu_tsne_frame$class)

km.tu_2 <- kmeans(tu_tsne_frame[,c(1,2)], 15)

tu_tsne_frame <- data.frame(cbind(tu_tsne_frame, km.tu_2$cluster))
colnames(tu_tsne_frame) <- c("X", "Y", "class", "class_tsne")
tu_tsne_frame$class_tsne <- as.factor(tu_tsne_frame$class_tsne)

saveRDS(tu_tsne_frame, "tu_tsne_frame.rds")
```

The four clusters are fairly distinct in this representation, but the
t-SNE clumps indicate that further analysis could yield other clusters
based off the t-SNE data itself:

```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}
ggplot()+
  geom_point(data = tu_tsne_frame, aes(x = X, y = Y, color = class)) +
  scale_color_discrete(name = "class")+
  labs(title = "TSNE") +
  theme_light()
```

*`r figs(name="tu_tsne","2-dimensional t-SNE of data")`*

### MDS: CMDSCALE (classical scaling)
```{r, eval=F, echo=F}
##################################################
# 1. create distance matrix
# 2. scale
# 3. add cluster labels
# 4. save object
##################################################

# tu_model_matrix_scaled_dist <- as.dist(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist <- daisy(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist_out_classic <-
data.frame(cmdscale(tu_model_matrix_scaled_dist))
# tu_model_matrix_scaled_dist.out <-
data.frame(sammon(tu_model_matrix_scaled_dist)$points)

tu_model_matrix_scaled_dist_out_classic <-
~~~

~~~
cbind(tu_model_matrix_scaled_dist_out_classic, km.tu_1$cluster)
colnames(tu_model_matrix_scaled_dist_out_classic) <- c("X1", "X2",
"km_class")
tu_model_matrix_scaled_dist_out_classic$km_class <-
as.factor(tu_model_matrix_scaled_dist_out_classic$km_class)

saveRDS(tu_model_matrix_scaled_dist_out_classic,
"tu_model_matrix_scaled_dist_out_classic.rds")
```

Classical scaling is a metric MDS approach (approximate inter-sample
dissimilarities as closely as possible). The shape of the data here is
quite similar to the PCA bi-plot. The clusters are quite clearly
separated, with no strong outliers:

```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}

ggplot()+
  geom_point(data = tu_model_matrix_scaled_dist_out_classic, aes(x =
X1, y = X2, color = km_class)) +
  scale_color_discrete(name = "class")+
  labs(title = "MDS: Classic") +
  theme_light()
```

*`r figs(name="tu_mds_classic","Classical metric MDS")`*

### MDS SMACOF Metric
```{r, eval=F, echo=F}
# tu_model_matrix_scaled_dist <- as.dist(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist <- daisy(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist_out_smacofM <-
data.frame(smacofSym(tu_model_matrix_scaled_dist, type =
"ratio")$conf)
# tu_model_matrix_scaled_dist.out <-
data.frame(sammon(tu_model_matrix_scaled_dist)$points)

tu_model_matrix_scaled_dist_out_smacofM <-
cbind(tu_model_matrix_scaled_dist_out_smacofM, km.tu_1$cluster)
colnames(tu_model_matrix_scaled_dist_out_smacofM) <- c("X1", "X2",
"km_class")
tu_model_matrix_scaled_dist_out_smacofM$km_class <-
as.factor(tu_model_matrix_scaled_dist_out_smacofM$km_class)

saveRDS(tu_model_matrix_scaled_dist_out_smacofM,
"tu_model_matrix_scaled_dist_out_smacofM.rds")

```

The metric SMACOF approach is similar to the classic approach, but
uses majorization to minimize the cost function. It is known to be an
inefficient algorithm, and can the results show this: many outliers
and a big clump in the center:

```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}
~~~

~~~
ggplot()+
  geom_point(data = tu_model_matrix_scaled_dist_out_smacofM, aes(x =
X1, y = X2, color = km_class)) +
  scale_color_discrete(name = "class")+
  labs(title = "MDS: SMACOF Metric") +
  theme_light()

```

*`r figs(name="tu_mds_smacofM","SMACOF Metric MDS")`*

### MDS SMACOF Non-metric
```{r, eval=F, echo=F}
# tu_model_matrix_scaled_dist <- as.dist(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist <- daisy(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist_out_smacofNM <-
data.frame(smacofSym(tu_model_matrix_scaled_dist, type =
"ordinal")$conf)
# tu_model_matrix_scaled_dist.out <-
data.frame(sammon(tu_model_matrix_scaled_dist)$points)

tu_model_matrix_scaled_dist_out_smacofNM <-
cbind(tu_model_matrix_scaled_dist_out_smacofNM, km.tu_1$cluster)
colnames(tu_model_matrix_scaled_dist_out_smacofNM) <- c("X1", "X2",
"km_class")
tu_model_matrix_scaled_dist_out_smacofNM$km_class <-
as.factor(tu_model_matrix_scaled_dist_out_smacofNM$km_class)

saveRDS(tu_model_matrix_scaled_dist_out_smacofNM,
"tu_model_matrix_scaled_dist_out_smacofNM.rds")

```

The non-metric (ordinal approach) version of SMACOF performs even
worse on the data - the majority of observations are squeezed into the
top right corner with a few outliers in the bottom left:

```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}
ggplot()+
  geom_point(data = tu_model_matrix_scaled_dist_out_smacofNM, aes(x =
X1, y = X2, color = km_class)) +
  scale_color_discrete(name = "class")+
  labs(title = "MDS: SMACOF Non-metric") +
  theme_light()

```

*`r figs(name="tu_mds_smacofNM","SMACOF Non-metric MDS")`*

### MDS: Kruskals non-metric
```{r, eval=F, echo=F}
# tu_model_matrix_scaled_dist <- as.dist(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist <- daisy(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist_out_Kruskal <-
data.frame(isoMDS(tu_model_matrix_scaled_dist)$points)
~~~

~~~
# tu_model_matrix_scaled_dist.out <-
data.frame(sammon(tu_model_matrix_scaled_dist)$points)

tu_model_matrix_scaled_dist_out_Kruskal <-
cbind(tu_model_matrix_scaled_dist_out_Kruskal, km.tu_1$cluster)
colnames(tu_model_matrix_scaled_dist_out_Kruskal) <- c("X1", "X2",
"km_class")
tu_model_matrix_scaled_dist_out_Kruskal$km_class <-
as.factor(tu_model_matrix_scaled_dist_out_Kruskal$km_class)

saveRDS(tu_model_matrix_scaled_dist_out_Kruskal,
"tu_model_matrix_scaled_dist_out_Kruskal.rds")

```

Kruskals is a non-metric approach that tries to minimize the
discrepancy between the rank order of the full dimension distance, and
the 2-dimension distance. While the clusters look somewhat separable,
it is not as clean as the classical metric approach:

```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}
ggplot()+
  geom_point(data = tu_model_matrix_scaled_dist_out_Kruskal, aes(x =
X1, y = X2, color = km_class)) +
  scale_color_discrete(name = "class")+
  labs(title = "MDS: Kruskal") +
  theme_light()

```

*`r figs(name="tu_mds_kruskal","Kruskals Non-metric MDS")`*

### MDS:  Sammon non-metric
```{r, echo=F, eval=F}
# tu_model_matrix_scaled_dist <- as.dist(tu_model_matrix_scaled)
tu_model_matrix_scaled_dist <-
suppressWarnings(daisy(tu_model_matrix_scaled))
tu_model_matrix_scaled_dist_out_sammon <-
data.frame(sammon(tu_model_matrix_scaled_dist)$points)
# tu_model_matrix_scaled_dist.out <-
data.frame(sammon(tu_model_matrix_scaled_dist)$points)

tu_model_matrix_scaled_dist_out_sammon <-
cbind(tu_model_matrix_scaled_dist_out_sammon, km.tu_1$cluster)
colnames(tu_model_matrix_scaled_dist_out_sammon) <- c("X1", "X2",
"km_class")
tu_model_matrix_scaled_dist_out_sammon$km_class <-
as.factor(tu_model_matrix_scaled_dist_out_sammon$km_class)

saveRDS(tu_model_matrix_scaled_dist_out_sammon,
"tu_model_matrix_scaled_dist_out_sammon.rds")

```

The Sammon approach is very similar to Kruskals except that the error
~~~

~~~
is adjusted further by the object distance in the original space. This
seems to help the algorithm, as cleaner cluster separation is observed
than Kruskals (although many outliers remain):

```{r, eval=T, echo=F, fig.height=3, fig.width=5, fig.align="center"}

ggplot()+
  geom_point(data = tu_model_matrix_scaled_dist_out_sammon, aes(x =
X1, y = X2, color = km_class)) +
  scale_color_discrete(name = "class")+
  labs(title = "MDS: Sammon") +
  theme_light()


```

*`r figs(name="tu_mds_sammon","Sammon non-metric")`*

## Self-organising maps

Self organizing maps are an unsupervised neural network that allocates
observations to neurons in a grid, while also adjusting grid
parameters. It is a helpful technique that allows us to capture the
characteristics of each neuron in full dimensionality while still
being able to represent data in a 2-dimensional map. Neurons from a
self-organizing map (or "Kohonen map") can be clustered themselves,
and so the K-means clusters are not considered in this section.

### 9 nodes

The first map considered is a 3x3 hexagonal grid. The data is
presented 1000 times, and the learning rate moves from 0.05 to 0.01.
The following is observed:

1. The training process stabilized after about 400 iterations
2. There are two nodes with most of the observations (makes sense
seeing that most vehicles are very similar, with a few specialty
vehicles)
3. Looking at just two attributes (Kilowatts and price), these seem to
correspond to the same vehicles and they are in the top right node

```{r, eval=T, echo=F}
##################################################
# create SOM object, and plot key charts
##################################################

set.seed(7)

som_grid <- somgrid(xdim = 3, ydim=3, topo="hexagonal")

som_model <- supersom(as.matrix(tu_model_matrix_scaled),
                  grid = som_grid,
                 rlen=1000,
                 alpha=c(0.05,0.01),
                 keep.data = TRUE
~~~

~~~
                 # n.hood='circular'
                 )
# names(som_model)


par(mfrow=c(3,2))
plot(som_model, type="changes")
plot(som_model, type="counts")
plot(som_model, type="dist.neighbours")
plot(som_model, type="codes")
plot(som_model, type = "property", property = som_model$codes[[1]]
[,27], main=names(tu_model_matrix)[27])
plot(som_model, type = "property", property = som_model$codes[[1]]
[,34], main=names(tu_model_matrix)[34])

# names(tu_model_matrix)
```

*`r figs(name="tu_som9_all","Key charts for 9 SOMS")`*

Looking at the observations within each of the nodes (only a limited
number of the 2000+ total observations are plotted), it does look like
similar types of vehicles have been grouped:

* In the grey node in the top left there are a lot of delivery van
type vehicles are grouped
* In the yellow node there are mostly "bakkies"
* The purples nodes seem to contain the most expensive premium
vehicles, while the top purple node has the most powerful of these
(5.0l vehicles)

More clusters could have been chosen, but four was selected in order
to match the MDS section with K-means.

```{r, eval=T, echo=F}

##################################################
# use uniform distribution to limit the amount of observations to
display in each node (i.e. ensure smaller nodes show up, and larger
nodes are not too crowded)
##################################################

node_count <- data.frame(table(som_model$unit.classif))
colnames(node_count) <- c("cat", "freq")

temp <- data.frame(cbind("name" = row.names(tu_model_matrix), "node" =
som_model$unit.classif))
temp$name <- as.character(temp$name)

set.seed(7)
for (i in 1:length(temp$name)){
  node_points <- ceiling((node_count$freq[temp$node[i]] /
sum(node_count$freq)) * 10)
  temp$name[i] <- ifelse(runif(1) <= (node_points + 3) /
node_count$freq[temp$node[i]], temp$name[i], "")
~~~

~~~
}

som_cluster <- cutree(hclust(dist(som_model$codes[[1]])), 4)
plot(som_model, type="mapping", labels = temp$name,
     main = "Clusters", cex = 0.5,
     bgcol = som_cluster + 12)
add.cluster.boundaries(som_model, som_cluster)
```

*`r figs(name="tu_som9_mapping","Mapping and clusters for 9 SOMS")`*

### 4 nodes

A 2x2 grid is used with the same parameters as the 3x3 SOM. The
following is observed:

1. The training process does not seem to stabilize as well as 3x3 grid
2. Expensive powerful cars still move to the same nodes, but are split
across two nodes instead of just one as in the 3x3 grid
3. There is a more even distribution of observations between the nodes

```{r, eval=T, echo=F}

set.seed(7)

som_grid <- somgrid(xdim = 2, ydim=2, topo="hexagonal")

som_model <- supersom(as.matrix(tu_model_matrix_scaled),
                  grid = som_grid,
                 rlen=1000,
                 alpha=c(0.05,0.01),
                 keep.data = TRUE
                 # n.hood='circular'
                 )
# names(som_model)

par(mfrow=c(3,2))
plot(som_model, type="changes")
plot(som_model, type="counts")
plot(som_model, type="dist.neighbours")
plot(som_model, type="codes")
plot(som_model, type = "property", property = som_model$codes[[1]]
[,27], main=names(tu_model_matrix)[27])
plot(som_model, type = "property", property = som_model$codes[[1]]
[,34], main=names(tu_model_matrix)[34])
```

*`r figs(name="tu_som4_all","Key charts for 4 SOMS")`*

Compared to the 3x3 map, there is a far less discernible pattern
within the nodes. It is difficult to say in general what each node
contains.

```{r, eval=T, echo=F}
~~~

~~~
node_count <- data.frame(table(som_model$unit.classif))
colnames(node_count) <- c("cat", "freq")

temp <- data.frame(cbind("name" = row.names(tu_model_matrix), "node" =
som_model$unit.classif))
temp$name <- as.character(temp$name)

set.seed(7)
for (i in 1:length(temp$name)){
  node_points <- ceiling((node_count$freq[temp$node[i]] /
sum(node_count$freq)) * 5)
  temp$name[i] <- ifelse(runif(1) <= (node_points + 7) /
node_count$freq[temp$node[i]], temp$name[i], "")
}

som_cluster <- cutree(hclust(dist(som_model$codes[[1]])), 4)
plot(som_model, type="mapping", labels = temp$name,
     main = "Clusters", cex = 0.5,
     bgcol = som_cluster + 12)
add.cluster.boundaries(som_model, som_cluster)
```

*`r figs(name="tu_som4_mapping","Mapping and clusters for 4 SOMS")`*

### Conclusion

According to the literature, the ideal map should contain about 5-10
observations per node [@Lynn2017]. This would require an approximately
20x20 map which would require a more delicate plotting mechanism -
potentially an interactive map - that would not work well in this
format. However, even the 3x3 map was able to show some sensible
groupings of vehicles indicating that it is a good technique for
finding groups within complex product data.

## Overall conclusion

The goal of project 2 was to be able to identify and visualize the
vehicle data in such a way as to be able to sensible groups within the
data. While it was not investigated further, the t-SNE approach showed
potential for finding more than 4 clusters of data. However, the other
MDS approaches were only able to show the 4 clusters and did not look
as though they would be useful for identifying more.

The most successful approach tested was the larger SOM (3x3 grid)
which was able to show sensible groups of vehicles. Further work
should look at identifying clusters from the t-SNE output, and finding
an optimal SOM grid size to further refine groups.

# References

R Core Team (2017). R: A language and environment for statistical
computing. R Foundation for Statistical Computing, Vienna, Austria.
URL https://www.R-project.org/.
~~~

R. Wehrens and L.M.C. Buydens, Self- and Super-organising Maps in R: the kohonen package J. Stat. Softw., 21(5), 2007

Jesse H. Krijthe (2015). Rtsne: T-Distributed Stochastic Neighbor Embedding using a Barnes-Hut Implementation, URL: https://github.com/jkrijthe/Rtsne

Jan de Leeuw, Patrick Mair (2009). Multidimensional Scaling Using Majorization: SMACOF in R. Journal of Statistical Software, 31(3), 1-30. URL http://www.jstatsoft.org/v31/i03/.

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.(2016). cluster: Cluster Analysis Basics and Extensions. R package version 2.0.5.

Hadley Wickham (2017). tidyverse: Easily Install and Load 'Tidyverse' Packages. R package version 1.1.1. https://CRAN.R-project.org/package=tidyverse