Breast Cancer Dataset

Emma Pedersen, Heidi Andersen & Melani Pedersen

27 September, 2021

```
sapply(c("dplyr", "FactoMineR", "factoextra", "ggplot2", "resample",
         "skimr", "tidyverse", "umap", "uwot", "readr",
         "tidymodels", "lubridate", "magrittr", "uwot", "GGally", "kableExtra", "knitr", "rsample", "VIM",
         "vip", "xgboost"
),
       require, character.only = TRUE)
```

```
     dplyr FactoMineR factoextra    ggplot2   resample      skimr  tidyverse
      TRUE       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
      umap       uwot      readr tidymodels  lubridate   magrittr       uwot
      TRUE       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
    GGally kableExtra      knitr    rsample        VIM        vip    xgboost
      TRUE       TRUE       TRUE       TRUE       TRUE       TRUE       TRUE
```

```
data <- read_csv("data.csv")
```

In this M1 examination, we have chosen to work with structured data [1]. The dataset we have chosen is: *Breast Cancer Wisconsin (Diagnostic)* which can be found at Kaggle: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data)
The dataset contains a listing of features that are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass and describes characteristics of the cell nuclei present in the image according to the description on Kaggle. The features can be seen in 32 columns, divided into 1 ID column, 1 string column, and 30 decimal column. The mean, standard error, and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in the 30 features. For instance, column 3 is the mean radius, column 13 is the standard error of the radius, column 23 is the worst radius. All feature values are recoded with four significant digits, which is summarized to:

- id: ID number
- diagnosis: Diagnosis of breast tissues (M = malignant, B = benign)
- …33: *unintended column*

And the decimal columns (all divided between mean, standard error and "worst"): * radius: Distance from center to points on the perimeter * texture: Standard deviation of gray-scale values * perimeter: Size of the core tumor * area * smoothness: Local variation in radius lengths * compactness: perimeter^2 / area - 1.0 * concavity: Severity of concave portions of the contour * concave points: Number of concave portions of the contour * symmetry * fractal_dimension: "coastline approximation" - 1

Exploratory Data Analysis and Manipulation

To get a quick overview of the data frame, several functions in R can be used. Our personal favorite are `head()` and `glimpse()` , but others like `dim()` , or `colnames()` could also be used. We use `head()` and `glimpse()` because it gives a quick overview of the data frame, and from here it can be seen if there are some troubles that need to be addressed before we take a closer look at it [2]. `tail()` could also have been used here because they are generic functions (`head`) [3].

```
head(data)
```

id <dbl>	diagnosis <chr>	radius_mean <dbl>	texture_mean <dbl>	perimeter_mean <dbl>	area_mean <dbl>	smoothness_mean <dbl>
842302	M	17.99	10.38	122.80	1001.0	0.11840
842517	M	20.57	17.77	132.90	1326.0	0.08474
84300903	M	19.69	21.25	130.00	1203.0	0.10960
84348301	M	11.42	20.38	77.58	386.1	0.14250
84358402	M	20.29	14.34	135.10	1297.0	0.10030
843786	M	12.45	15.70	82.57	477.1	0.12780

6 rows | 1-7 of 33 columns

```
glimpse(data)
```

```
Rows: 568
Columns: 33
$ id                      <dbl> 842302, 842517, 84300903, 84348301, 84358402, …
$ diagnosis               <chr> "M", "M", "M", "M", "M", "M", "M", "M", "M", "…
$ radius_mean             <dbl> 17.990, 20.570, 19.690, 11.420, 20.290, 12.450…
$ texture_mean            <dbl> 10.38, 17.77, 21.25, 20.38, 14.34, 15.70, 19.9…
$ perimeter_mean          <dbl> 122.80, 132.90, 130.00, 77.58, 135.10, 82.57, …
$ area_mean               <dbl> 1001.0, 1326.0, 1203.0, 386.1, 1297.0, 477.1, …
$ smoothness_mean         <dbl> 0.11840, 0.08474, 0.10960, 0.14250, 0.10030, 0…
$ compactness_mean        <dbl> 0.27760, 0.07864, 0.15990, 0.28390, 0.13280, 0…
$ concavity_mean          <dbl> 0.30010, 0.08690, 0.19740, 0.24140, 0.19800, 0…
$ `concave points_mean`   <dbl> 0.14710, 0.07017, 0.12790, 0.10520, 0.10430, 0…
$ symmetry_mean           <dbl> 0.2419, 0.1812, 0.2069, 0.2597, 0.1809, 0.2087…
$ fractal_dimension_mean  <dbl> 0.07871, 0.05667, 0.05999, 0.09744, 0.05883, 0…
$ radius_se               <dbl> 1.0950, 0.5435, 0.7456, 0.4956, 0.7572, 0.3345…
$ texture_se              <dbl> 0.9053, 0.7339, 0.7869, 1.1560, 0.7813, 0.8902…
$ perimeter_se            <dbl> 8.589, 3.398, 4.585, 3.445, 5.438, 2.217, 3.18…
$ area_se                 <dbl> 153.40, 74.08, 94.03, 27.23, 94.44, 27.19, 53.…
$ smoothness_se           <dbl> 0.006399, 0.005225, 0.006150, 0.009110, 0.0114…
$ compactness_se          <dbl> 0.049040, 0.013080, 0.040060, 0.074580, 0.0246…
$ concavity_se            <dbl> 0.05373, 0.01860, 0.03832, 0.05661, 0.05688, 0…
$ `concave points_se`     <dbl> 0.015870, 0.013400, 0.020580, 0.018670, 0.0188…
$ symmetry_se             <dbl> 0.03003, 0.01389, 0.02250, 0.05963, 0.01756, 0…
$ fractal_dimension_se    <dbl> 0.006193, 0.003532, 0.004571, 0.009208, 0.0051…
$ radius_worst            <dbl> 25.38, 24.99, 23.57, 14.91, 22.54, 15.47, 22.8…
$ texture_worst           <dbl> 17.33, 23.41, 25.53, 26.50, 16.67, 23.75, 27.6…
$ perimeter_worst         <dbl> 184.60, 158.80, 152.50, 98.87, 152.20, 103.40,…
$ area_worst              <dbl> 2019.0, 1956.0, 1709.0, 567.7, 1575.0, 741.6, …
$ smoothness_worst        <dbl> 0.1622, 0.1238, 0.1444, 0.2098, 0.1374, 0.1791…
$ compactness_worst       <dbl> 0.6656, 0.1866, 0.4245, 0.8663, 0.2050, 0.5249…
$ concavity_worst         <dbl> 0.71190, 0.24160, 0.45040, 0.68690, 0.40000, 0…
$ `concave points_worst`  <dbl> 0.26540, 0.18600, 0.24300, 0.25750, 0.16250, 0…
$ symmetry_worst          <dbl> 0.4601, 0.2750, 0.3613, 0.6638, 0.2364, 0.3985…
$ fractal_dimension_worst <dbl> 0.11890, 0.08902, 0.08758, 0.17300, 0.07678, 0…
$ ...33                   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA…
```

From the above, it can be seen, that this data is not in accordance with Kaggle's description. According to Kaggle, there should be 32 columns while the dataset in R contains 33 columns. From a quick view, it seems like it is column *… .33* that is the problem and we will therefore need to take a closer looked at that.

In addition, we look for whether the data appears to be *tidy*, ie. each variable has its own column, each observation has its own row and each

value has its own cell. This is important as tidy data is a consistent way of storing data, and a lot of the tools in R require that. But from a quick view, it seems like this requirement is fulfilled, and we will therefore continue with this data.
To get a more detailed look at the data, several functions in R can be used. Our personal favorite is the `skim()` function [4], but others like `str()` [5] can also be used. We use the `skim()` function because it providing a broad overview of the data frame than `str()`, like the column type, mean and standard deviation.

```
skim(data)
```

Data summary

Name	data
Number of rows	568
Number of columns	33
_______________________	
Column type frequency:	
character	1
logical	1
numeric	31
________________________	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
diagnosis	0	1	1	1	0	2	0

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
…33	568	0	NaN	:

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
id	0	1	30425139.67	125124311.81	8670.00	869222.50	906157.00	8825022.25	911320502.00	▇▁▁▁▁
radius_mean	0	1	14.14	3.52	6.98	11.71	13.38	15.80	28.11	▂▇▃▁▁
texture_mean	0	1	19.28	4.30	9.71	16.17	18.84	21.79	39.28	▂▇▃▁▁
perimeter_mean	0	1	92.05	24.25	43.79	75.20	86.29	104.15	188.50	▂▇▃▁▁
area_mean	0	1	655.72	351.66	143.50	420.30	551.40	784.15	2501.00	▇▃▂▁▁
smoothness_mean	0	1	0.10	0.01	0.06	0.09	0.10	0.11	0.16	▁▇▆▁▁
compactness_mean	0	1	0.10	0.05	0.02	0.07	0.09	0.13	0.35	▇▇▂▁▁
concavity_mean	0	1	0.09	0.08	0.00	0.03	0.06	0.13	0.43	▇▃▂▁▁
concave points_mean	0	1	0.05	0.04	0.00	0.02	0.03	0.07	0.20	▇▃▂▁▁
symmetry_mean	0	1	0.18	0.03	0.11	0.16	0.18	0.20	0.30	▁▇▅▁▁
fractal_dimension_mean	0	1	0.06	0.01	0.05	0.06	0.06	0.07	0.10	▆▇▂▁▁
radius_se	0	1	0.41	0.28	0.11	0.23	0.32	0.48	2.87	▇▁▁▁▁
texture_se	0	1	1.22	0.55	0.36	0.83	1.11	1.47	4.88	▇▅▁▁▁
perimeter_se	0	1	2.87	2.02	0.76	1.61	2.29	3.36	21.98	▇▁▁▁▁
area_se	0	1	40.37	45.52	6.80	17.85	24.57	45.24	542.20	▇▁▁▁▁
smoothness_se	0	1	0.01	0.00	0.00	0.01	0.01	0.01	0.03	▇▃▁▁▁
compactness_se	0	1	0.03	0.02	0.00	0.01	0.02	0.03	0.14	▇▃▁▁▁
concavity_se	0	1	0.03	0.03	0.00	0.02	0.03	0.04	0.40	▇▁▁▁▁
concave points_se	0	1	0.01	0.01	0.00	0.01	0.01	0.01	0.05	▇▇▁▁▁
symmetry_se	0	1	0.02	0.01	0.01	0.02	0.02	0.02	0.08	▇▃▁▁▁
fractal_dimension_se	0	1	0.00	0.00	0.00	0.00	0.00	0.00	0.03	▇▁▁▁▁
radius_worst	0	1	16.28	4.83	7.93	13.02	14.97	18.79	36.04	▆▇▃▁▁
texture_worst	0	1	25.67	6.15	12.02	21.07	25.41	29.68	49.54	▂▇▆▁▁
perimeter_worst	0	1	107.35	33.57	50.41	84.15	97.66	125.53	251.20	▇▇▃▁▁
area_worst	0	1	881.66	569.28	185.20	515.67	686.55	1085.00	4254.00	▇▂▁▁▁
smoothness_worst	0	1	0.13	0.02	0.07	0.12	0.13	0.15	0.22	▁▇▇▂▁
compactness_worst	0	1	0.25	0.16	0.03	0.15	0.21	0.34	1.06	▇▅▁▁▁

concavity_worst	0	1	0.27	0.21	0.00	0.12	0.23	0.38	1.25	▇▅▂▁▁
concave points_worst	0	1	0.11	0.07	0.00	0.06	0.10	0.16	0.29	▅▇▅▃▁
symmetry_worst	0	1	0.29	0.06	0.16	0.25	0.28	0.32	0.66	▅▇▁▁▁
fractal_dimension_worst	0	1	0.08	0.02	0.06	0.07	0.08	0.09	0.21	▇▃▁▁▁

Looking at the *skim()* table above, firstly we notice that the feature “diagnosis” is a character, with 2 unique elements: M = malignant, B = benign. A character in R is just a piece of text, and can not be used when creating a regression. Therefore we want to change “diagnosis” to a dummy variable. This will be done in the chapter *Supervised machine learning*.

Secondly we notice that the column *… 33* is of the logical type, meaning it only contains TRUE and FALSE values. However, this column contains 568 NA variables and therefore does not provide useful information. Checking for columns with NA values can also be done as follows:

```
data %>%
  select_if(function(x) any(is.na(x))) %>%
  summarise_each(funs(sum(is.na(.))))
```

...33 <int>
568

1 row

The remaining features are of a numerical nature and it is seen that none of these contain missing values. It can, in addition, be seen that these values range widely, and it should therefore be considered whether the variables should be scaled so that the independent features are standardized. If this is not done, then a machine learning algorithm will often tend to weigh greater values higher and consider smaller values as the lower values, regardless of the unit of the values. However, this is not necessary to do independently, as most machine learning functions eventually have this built-in.

Based on the above, it can therefore be concluded that the following cleaning of the data should be performed: Remove colunm *..33* In addition, it can be argued that the serial numbers should be the patients’ ID number and not just the observation number, and this is therefore also changed:

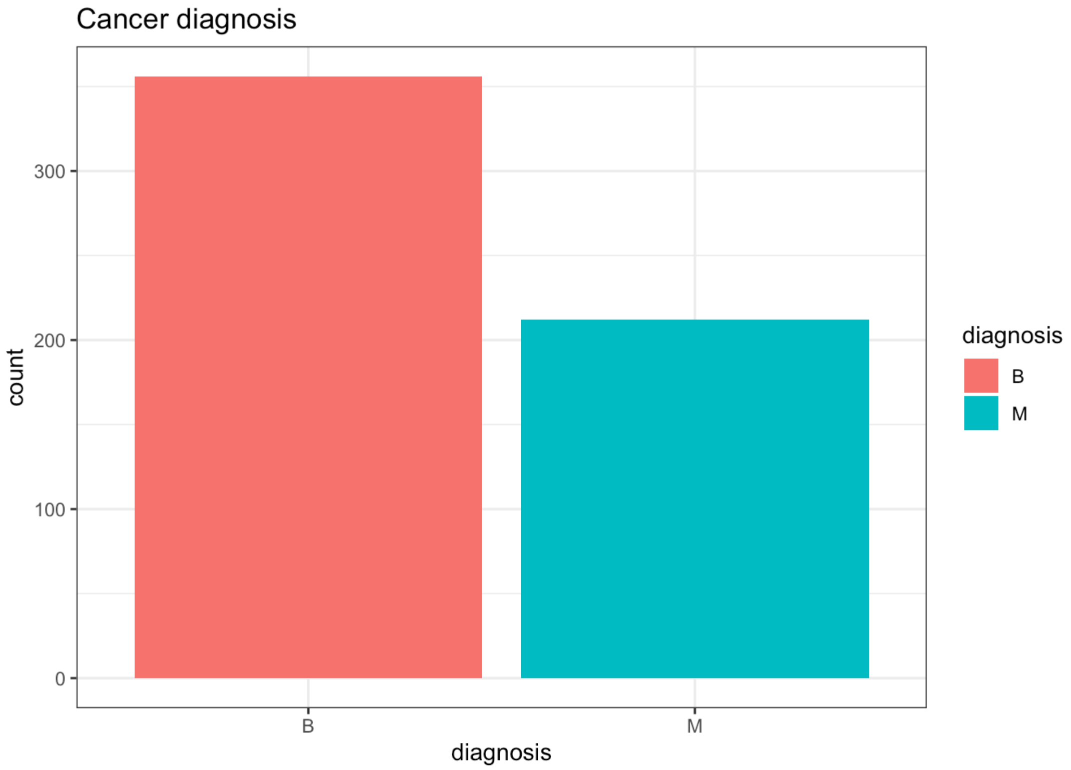
```
# Removing columns containing NA's
data %<>% select_if(~all(!is.na(.)))
```

```
# Change rox names
data %<>%
  column_to_rownames("id")
```

Now as the data is cleaned, it is now possible to explore the data via EDA analysis.

To get an overview of the number of possible cancer diagnoses, and how many actually get that diagnoses, a bar plot is made. We have chosen this type of plot because it creates a quick visual overview of the distribution of cancer diagnosis, at the same time as it can be seen which diagnoses there are. An alternative to this could be a pie chart, especially if one wants to work with percentages.

```
ggplot(data = data) +
  geom_bar(aes(x = diagnosis, fill=diagnosis)) +
  ggtitle(label = "Cancer diagnosis") +
  theme_bw()
```



From the above, it seems that a bit over 1/3 of all diagnoses are malignant. Printing a table with the specific numbers of the allocation of malignant and benign:

```
data %>%
  count(diagnosis)
```

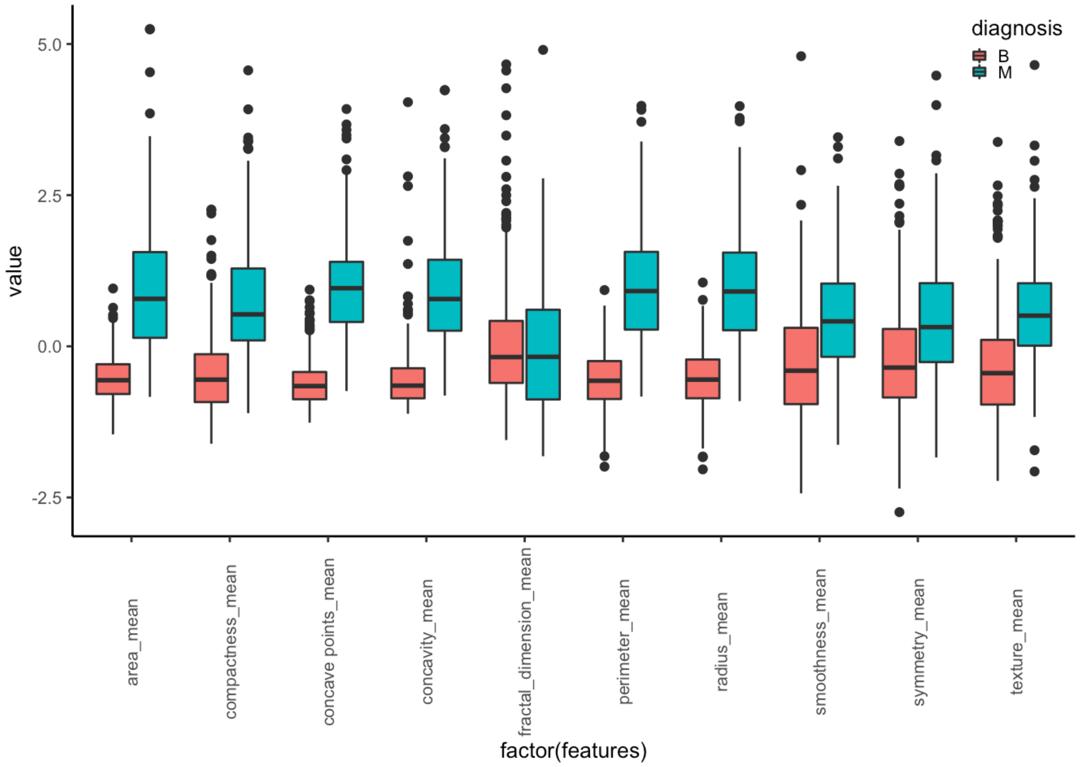
diagnosis <chr>	**n** <int>
B	356
M	212

2 rows

212 patients are told that they have malignant cancer when they have had an FNA performed, while 356 parties are told that it is benign cancer. This larger number of malignant patients makes it possible to make a prediction in supervised machine learning.
To get a more clear overview of which characteristics a FNA malignant and benign diagnoses have, we have chosen to plot a boxplot, but a density plot could also have been used. To improve the visualization of the boxplot, we have divided the data into its natural 3 categories; “mean”, “standard error” and “worst”, which where explained further above. In addition the data is then scaled, to get a more normalized data set.

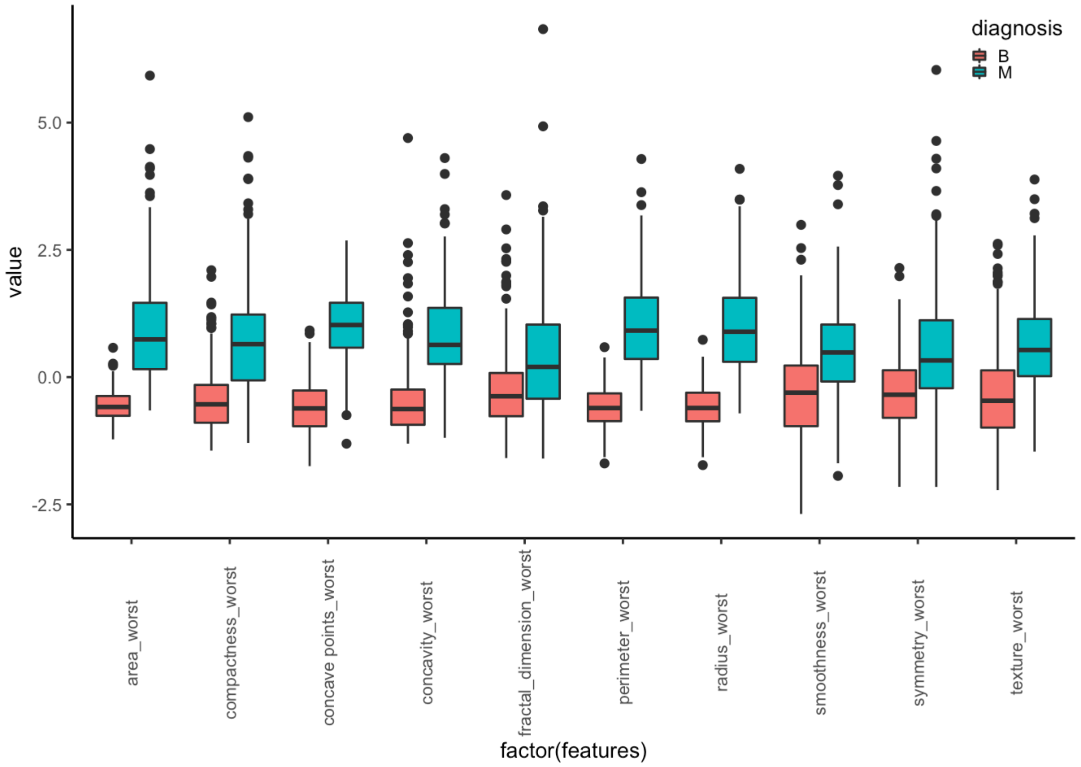
```
x_mean <- data %>% select(ends_with("mean")) %>% scale() %>% as.tibble() %>%   mutate(data["diagnosis"])
x_se <- data %>% as.tibble() %>% select(ends_with("se")) %>% scale() %>% as.tibble() %>% mutate(data["diagnosis
"])
x_worst <- data %>% as.tibble() %>% select(ends_with("worst")) %>% scale() %>% as.tibble() %>%   mutate(data["d
iagnosis"])
```

```
x_mean %>%
  pivot_longer(col = -diagnosis, names_to = "features", values_to = "value") %>%
  ggplot(aes(factor(features),
         value, fill = diagnosis)) +
  geom_boxplot() +
  theme_extra +
  theme(axis.text.x = element_text(angle=90, vjust=0.5)) +
  legend_top_right_inside
```



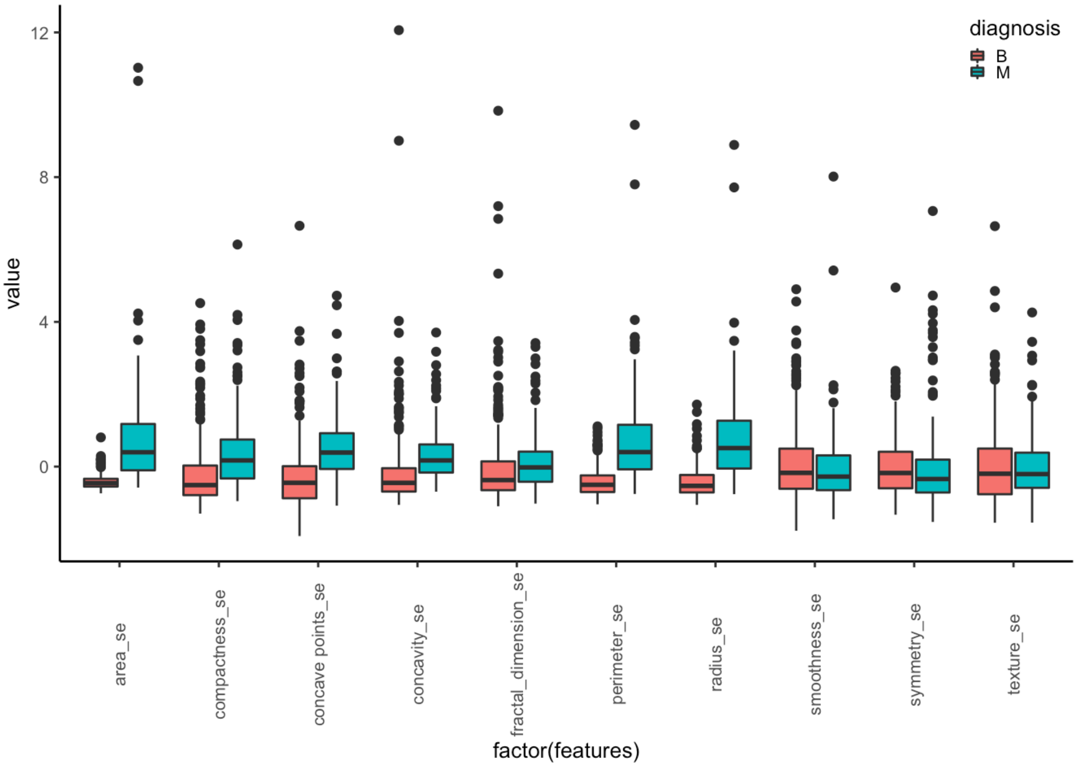
The plot above shows the boxplots of the “mean” features. Notice that the boxplots of malignant diagnosis (blue) are generally more spread out/has a wider range than the boxplots of benign diagnosis (red). This implies that, eg, the size of concave points could be used as a proxy for the diagnostics. However, this is not bulletproof, as can be seen by the outliers, and should therefore not hold the entire prediction by itself. In addition, it should also be pointed out from the boxplots that the fractal_dimension_mean generally has the same median, and can therefore not be used to see the difference between benign and malignant cancer.

```
x_worst %>%
  pivot_longer(col = -diagnosis, names_to = "features", values_to = "value") %>%
  ggplot(aes(factor(features),
           value, fill = diagnosis)) +
  geom_boxplot() +
  theme_extra +
  theme(axis.text.x = element_text(angle=90, vjust=0.5)) +
  legend_top_right_inside
```



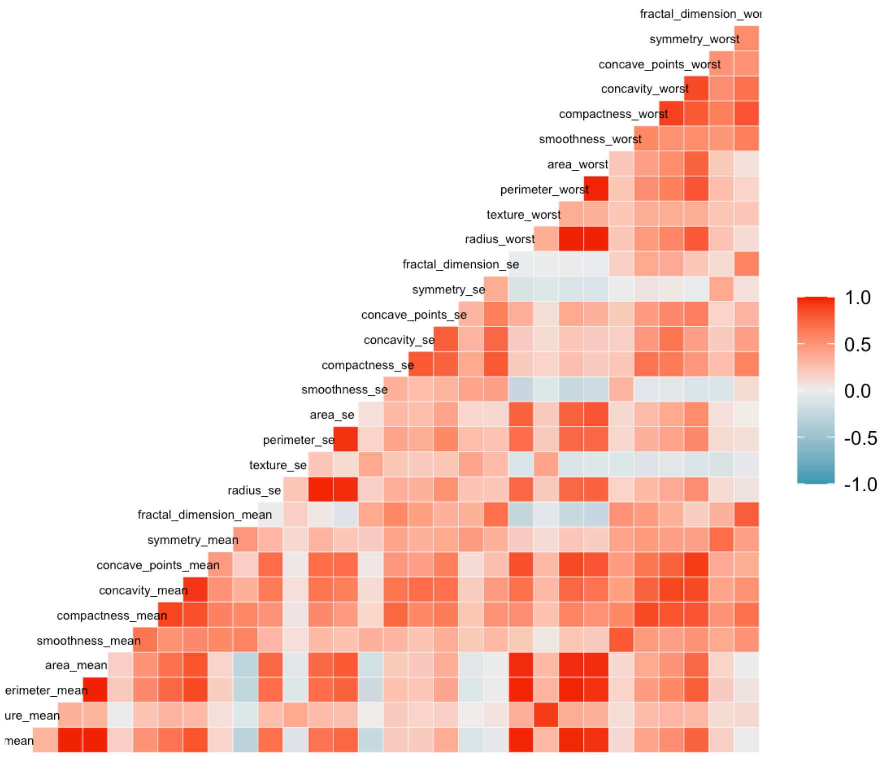
The plot above shows the boxplots of the “worst” features. Here we recognize that the malignant boxplots (blue) are generally more spread out/has a wider range than the benign boxplots (red) as was the case in the “means” boxplot. This again implies that eg. the size of concave points can be used as a proxy for the diagnostics, but should not hold the intier prediction

```
x_se %>%
  pivot_longer(col = -diagnosis, names_to = "features", values_to = "value") %>%
  ggplot(aes(factor(features),
           value, fill = diagnosis)) +
  geom_boxplot() +
  theme_extra +
  theme(axis.text.x = element_text(angle=90, vjust=0.5)) +
  legend_top_right_inside
```



The plot above shows the boxplots of the “standard errors” (se) features. It can be seen that the boxplots of both malignant and benign has a lot of outliers, which should be kept in mind when used for prediction in supervised machine learning. Furthermore the medians of the boxplots of both malignant and benign cancer lies relatively close to each other. Therefore, they are seemingly not a good indicator for predicting what underlies malignant cancer.
Because some of the features seem visually close to each other, we would like to check for the correlation between them. If the correlation is very high between two features, some features might be dispensed in the unsupervised machine learning part, since they contain the same information. We show the correlation between the features in a heat map which is a graphical representation of data where the individual values contained in a matrix are represented as colors.

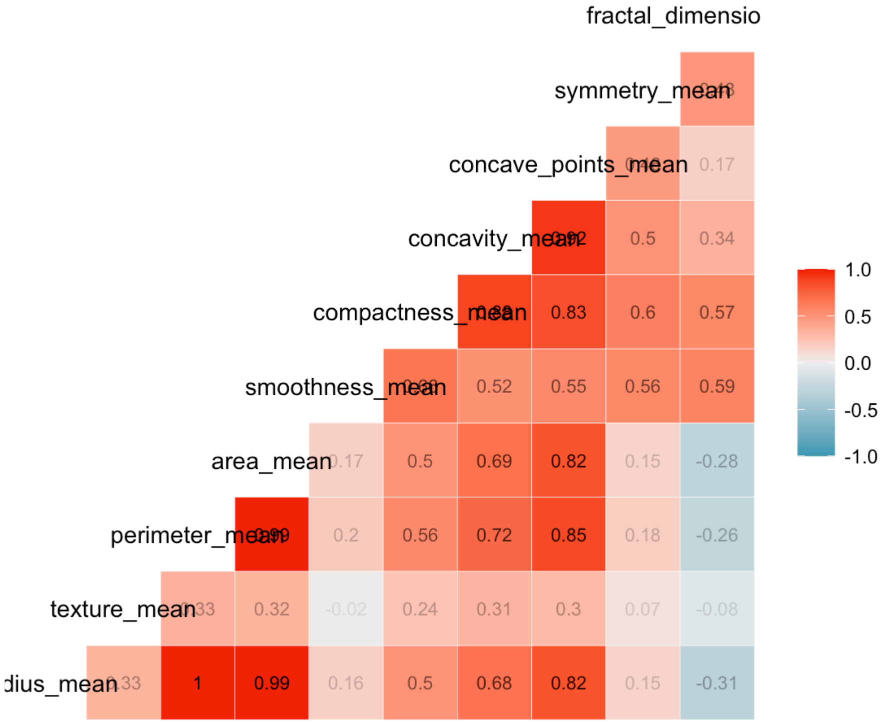
```
data %>%
  ggcorr(hjust = 0.8, size = 2)
```



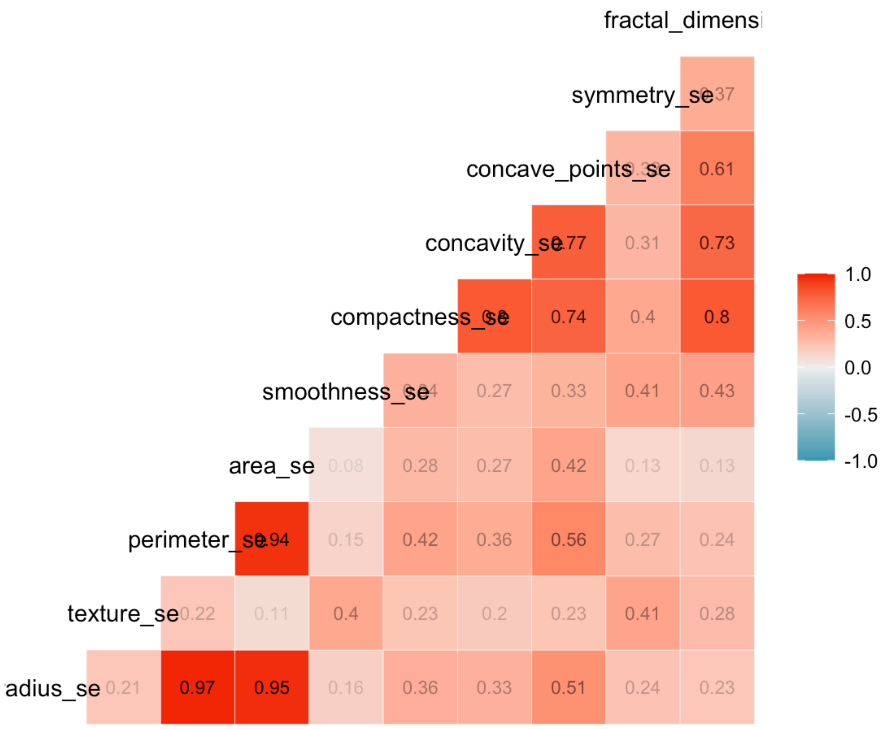
From the correllogram above, it can be seen that a number of the features are strongly correlated, which may indicate that they measure the same. However, with our limited knowledge of FNA, this is not known with certainty.

Due to the large number of features it can be difficult to see which features are actually correlated. We therefore choose to divide the data set naturally into 3 subsets from which we make 3 new heatmaps. In this regard, it is important to mention that this can lead to some missing elements since we are going to miss seeing correlations between features in the different subsets, e.g. radius_mean and radius_se.

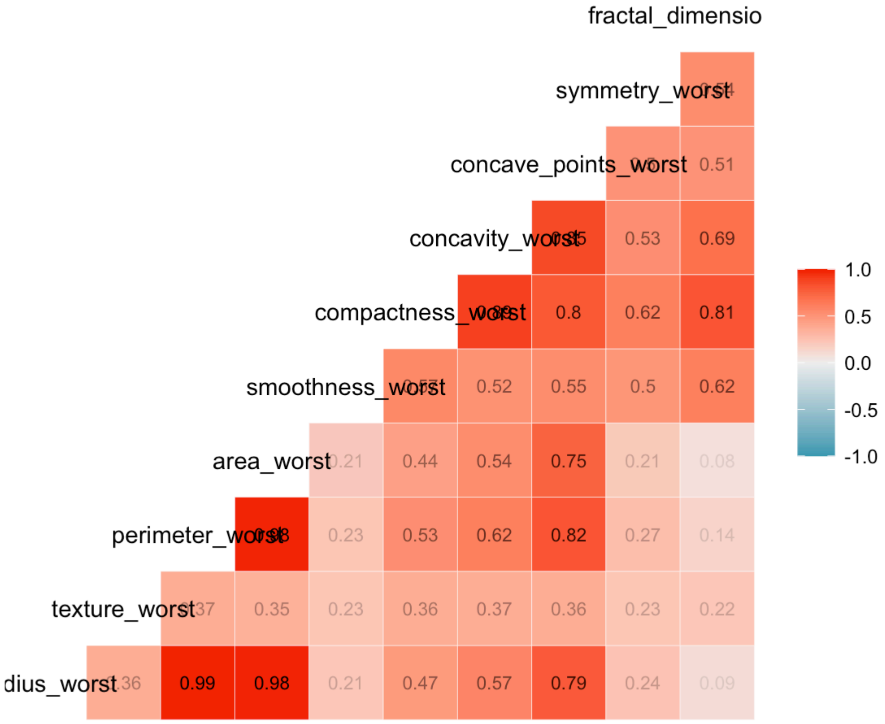
```
# heatmap for mean features
data %>% select(1:11) %>%
  ggcorr(label = TRUE,
               label_size = 3,
               label_round = 2,
               label_alpha = TRUE)
```



```
# heatmap for SE features
data %>% select(c(1, 12:21)) %>%
  ggcorr(label = TRUE,
               label_size = 3,
               label_round = 2,
               label_alpha = TRUE)
```



```
# heatmap for worst features
data %>% select(c(1, 22:31)) %>%
  ggcorr(label = TRUE,
               label_size = 3,
               label_round = 2,
               label_alpha = TRUE)
```



From the subsetting, it is more clear that the data set contains some highly correlated values in all of the subsets, especially positive ones. Some examples of high positive correlations are radius_mean and the area_mean (0.99), radius_se and perimer_ se (0.97), as well as radius_worst and area_worst (0.98). However, the subset mean contains far more negative correlations, e.g. between radius_mean and fractal_dimension.
As mentioned, it must be taken in to consideration that although a division into three heatmaps creates a more clear overview, a problem also arises because of missing correlations between features in the different subsets.

Unsupervised machine learning

Because several of the features are strongly correlated, they may be measuring the same phenomenon. We don’t want our models double counting the same thing, and will therefore use dimensionality reduction to help us reduce the data set. The dimensionality reduction method we have chosen is PCA, but other types of dimensionality reduction such as *SVD* or *umap* could also have been used.

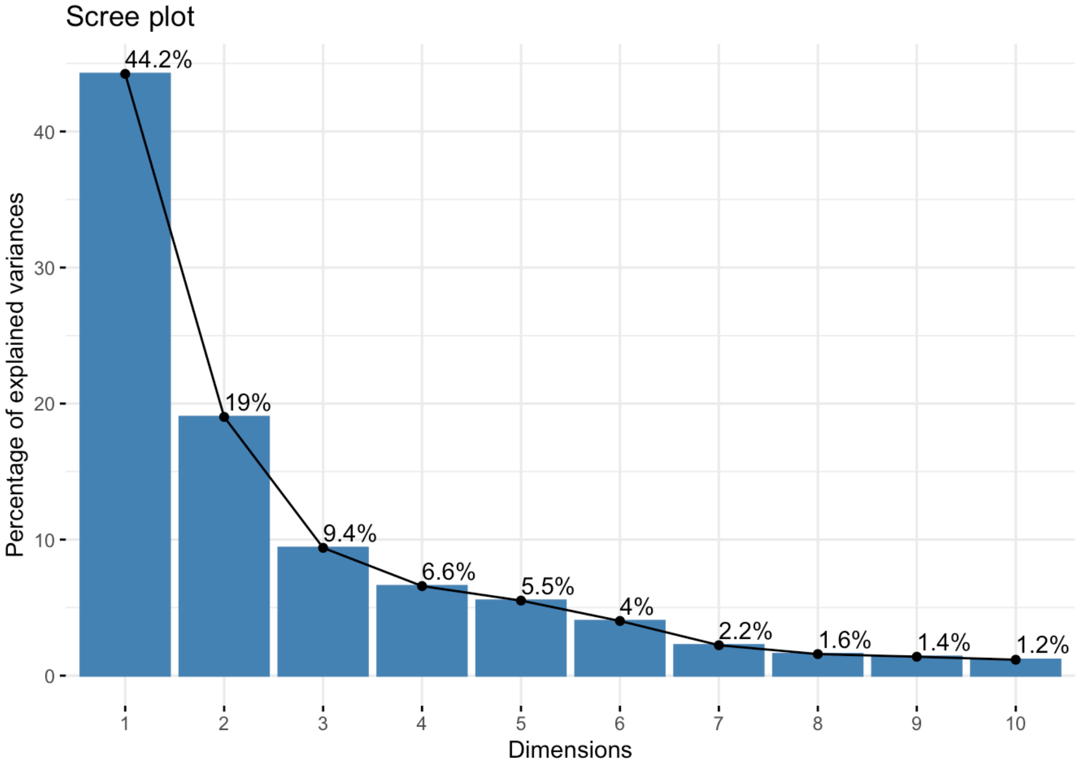
Principal Components Analysis

A Principal Components Analysis (PCA) makes it possible to construct new variables. These build on the weighted sums of existing variables, with the purpose of creating fewer, less correlated variables while stille keeping the majority of the variation from the original dataset. To perform a PCA analysis, the `PCA()` function is used. The PCA is computed over the exogenous variables. Furthermore, the data is scaled to normalize it.

```
# PCA test
data_pca <- data %>%
  select_if(is.numeric) %>%
  PCA(scale.unit = TRUE, graph = FALSE)
```

The `fviz_screeplot` is used for plotting the dimensions in order to visualize how much each dimension captures and how many dimensions should be used for further examination of the data. Furthermore, the eigenvalues are computed. An eigenvalue corresponds to the amount of the variation is explained by each principal component in the PCA and is therefore a method for determining the number of dimensions to retain in an PCA: a rule of thumb states that an eigenvalue must be above 1 in order for the dimension to be used.

```
data_pca %>%
  fviz_screeplot(addlabels = TRUE)
```



The screeplot indicates that the dataset can be minimized to around 2-3 dimensions since the “elbow” of the plot is around there. However, it is not a clear elbow and we therefore check the eigenvalues as well to examine how many dimensions should be used according to eigenvalues.

```
# Eigenvalue
data_pca$eig %>%
  as_tibble() %>%
  round(2)
```

eigenvalue	percentage of variance	cumulative percentage of variance
<dbl>	<dbl>	<dbl>
13.27	44.24	44.24

5.71	19.02	63.26
2.82	9.40	72.65
1.98	6.58	79.24
1.66	5.52	84.75
1.21	4.02	88.77
0.67	2.24	91.01
0.48	1.59	92.60
0.42	1.39	93.99
0.35	1.17	95.16

1-10 of 30 rows Previous **1** 2 3 Next

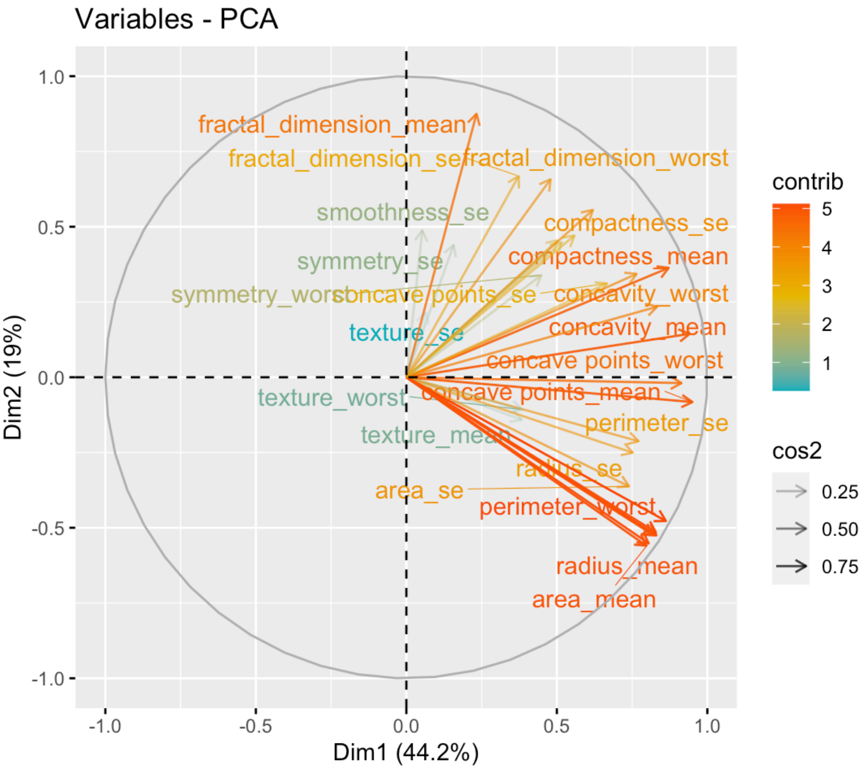
The table containing eigenvalues suggests that the dataset should be cut down to 6 dimensions in stead of the 30 dimensions from the original dataset; 6 dimensions will explain 88.88% of the variance from the original dataset.
To obtain a better outline of these dimensions we compute a table of the features' coordinates in each dimension.

```
data_pca$var$coord %>%
  head()
```

```
                   Dim.1    Dim.2     Dim.3     Dim.4     Dim.5
radius_mean      0.79714 -0.55952 -0.012813 -0.056345 -0.048654
texture_mean     0.38313 -0.14142  0.100971  0.846932  0.070212
perimeter_mean   0.82874 -0.51481 -0.014228 -0.057366 -0.048187
area_mean        0.80499 -0.55189  0.049018 -0.074456 -0.013257
smoothness_mean  0.51655  0.44770 -0.173145 -0.223341  0.474122
compactness_mean 0.87150  0.36339 -0.124886 -0.046979 -0.016126
```

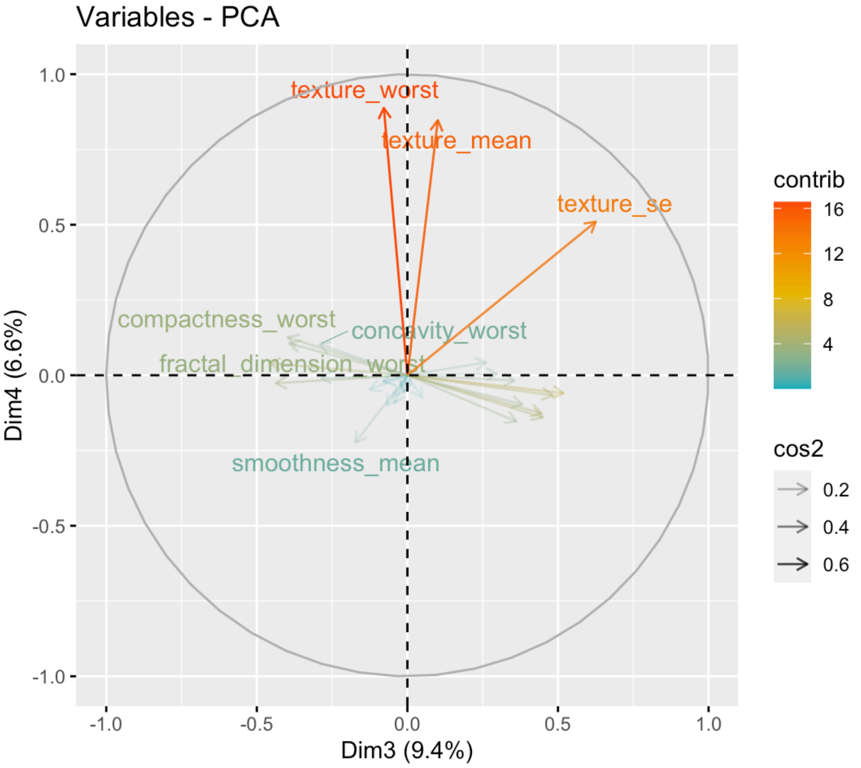
Since the table only shows 5 dimensions, it is assumed that the data can be cut to this rather than 6. These dimensions are further explored using …

```
data_pca %>%
  fviz_pca_var(alpha.var = "cos2",
               col.var = "contrib",
               gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
               repel = TRUE,
               ggtheme = theme_gray())
```



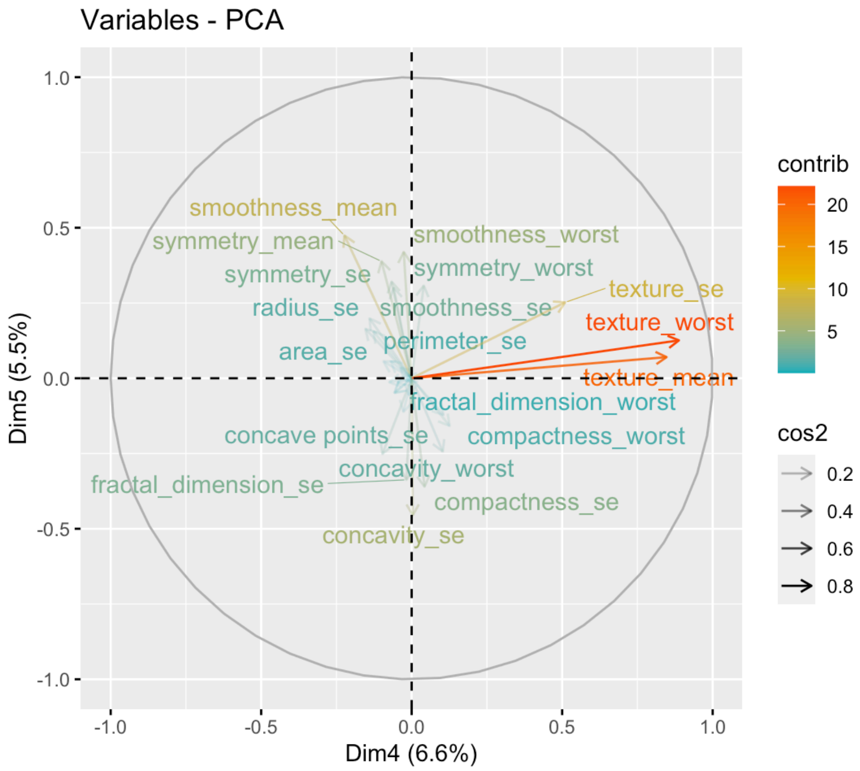
The figure above shows the features according to the first 2 dimensions. It can be seen that the features move jointly according to the first dimension shown on the horizontal axis, meaning they move in the same direction. However, they split in the second dimension/vertical axis. It can furthermore be seen that radius, perimeter and area are correlated, while fractal dimension, smoothness and compactness are correlated. This could reflect that the second dimension is a measure of the complexity of the observations.

```
data_pca %>%
  fviz_pca_var(axes = c(3,4),
               alpha.var = "cos2",
               col.var = "contrib",
               gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
               repel = TRUE,
               ggtheme = theme_gray())
```



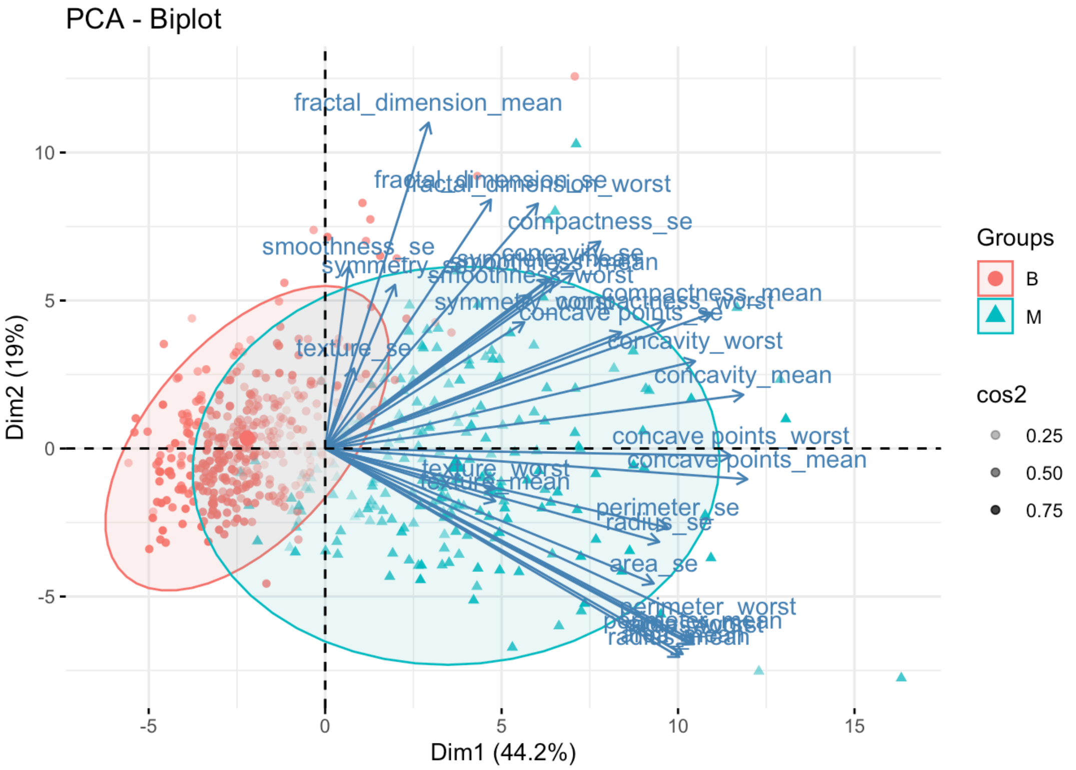
The figure above shows the features according to the 3 and 4 dimensions. It can be seen that the features move jointly according to the 4 dimension, meaning they move in the same direction. However, they split in the 3 dimension.

```
data_pca %>%
  fviz_pca_var(axes = c(4,5),
               alpha.var = "cos2",
               col.var = "contrib",
               gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
               repel = TRUE,
               ggtheme = theme_gray())
```



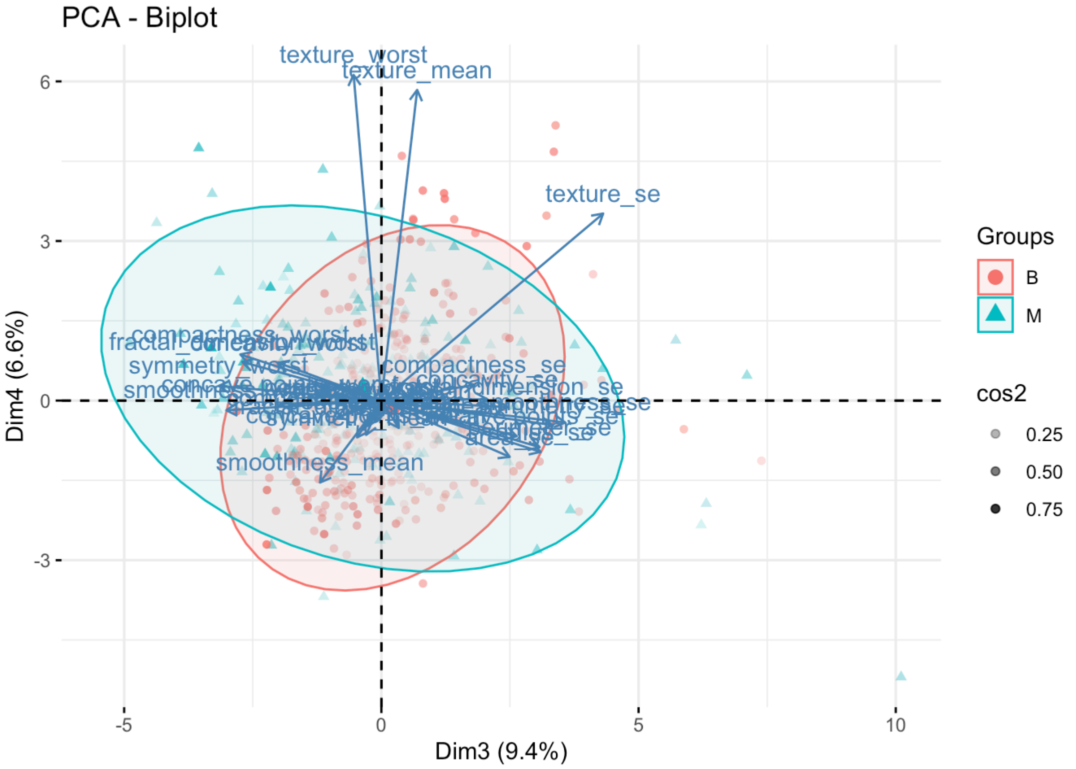
The figure above shows the 4 and 5 dimension. Here we notice that the three dominating feautures (the red arrows) move jointly in the same direction. Since they lie so close to each other means they are very positively correlated, meaning that they almost bring the same information to the analysis. This makes it easier to argue whether whe should remove the 4 and 5 dimensions from the analysis.

```
data_pca %>%
  fviz_pca_biplot( axes = c(1,2),
    alpha.ind = "cos2",
                  geom = "point",
                  habillage = data %>% pull(diagnosis) %>% factor(),
                  addEllipses = TRUE)
```



The figure above illustrates the biplot grouped by the diagnosis for the first 2 dimensions. Here, the benign cases are scattered across the red ellipsis while the malignant cases are scattered across the blue ellipsis. However, some of the observations lie outside their corresponding ellipsis. From the biplot it can be seen that the malignant cases are furthest to the right on horizontal axis, hence the features are correlated with a malignant diagnosis. In terms of the vertical allocation, there is no clear distinction between benign or malignant diagnoses.

```
data_pca %>%
  fviz_pca_biplot( axes = c(3,4),
    alpha.ind = "cos2",
                  geom = "point",
                  habillage = data %>% pull(diagnosis) %>% factor(),
                  addEllipses = TRUE)
```

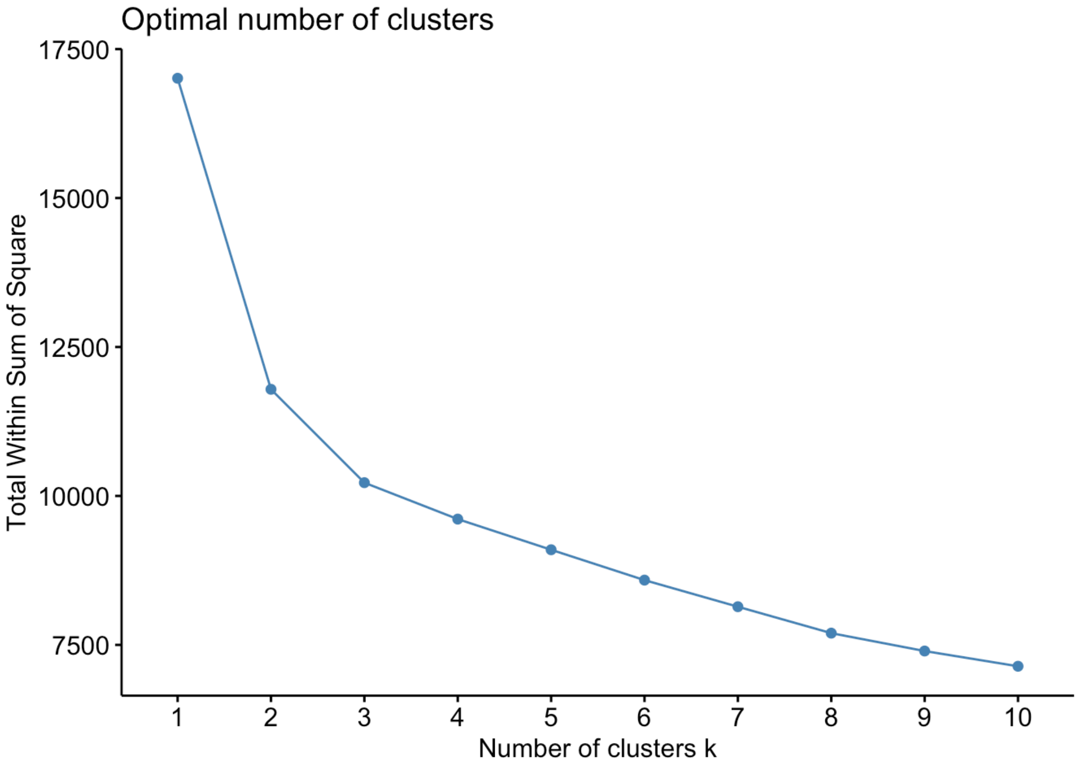


The figure above illustrates the biplot grouped by the diagnosis for the 3 and 4 dimensions. Here, again the benign cases are scattered across the red ellipsis while the malignant cases are scattered across the blue ellipsis. However, some of the observations lie outside their corresponding ellipsis. From the biplot it can however be seen that the there is no clear distinction between benign or malignant diagnoses, as it looks like they are on top of each other.

Clustering

Since the data set is now minimized to the most essential, a clustering analysis can be performed. This is done to classify the data into structures that can be more easily understood and manipulated. In this analysis we have chosen to use *kmeans*, but *hierarchical clustering* or *HDBSCAN* could also have been used. *Kmeans* is characterized by finding homogeneous subgroups within a population. The `kmeans()` function has a random component, which means that it runs multiple times, and then the best solution is selected. The best solution is the one where the *total within cluster sum of squares* is as small as possible. This is being calculated by taking the squared distance from each observation to the cluster center, and is called the squared Euclidean distance. Before we can run a kemans, we need to know which number of clusters that is appropriate in the data. It could be argued that the data contains 2 clusters since there are 2 possible outcomes: benign and malignant cancer. However, we still check for the optimal number of clusters using a screeplot. The x-axis represents the number of clusters while the y-axis represents the *total within cluster sum of squares* for each number of clusters. We are here looking for an *elbow* in the plot, i.e. where the total within cluster sum of squares decreases at a slower rate.

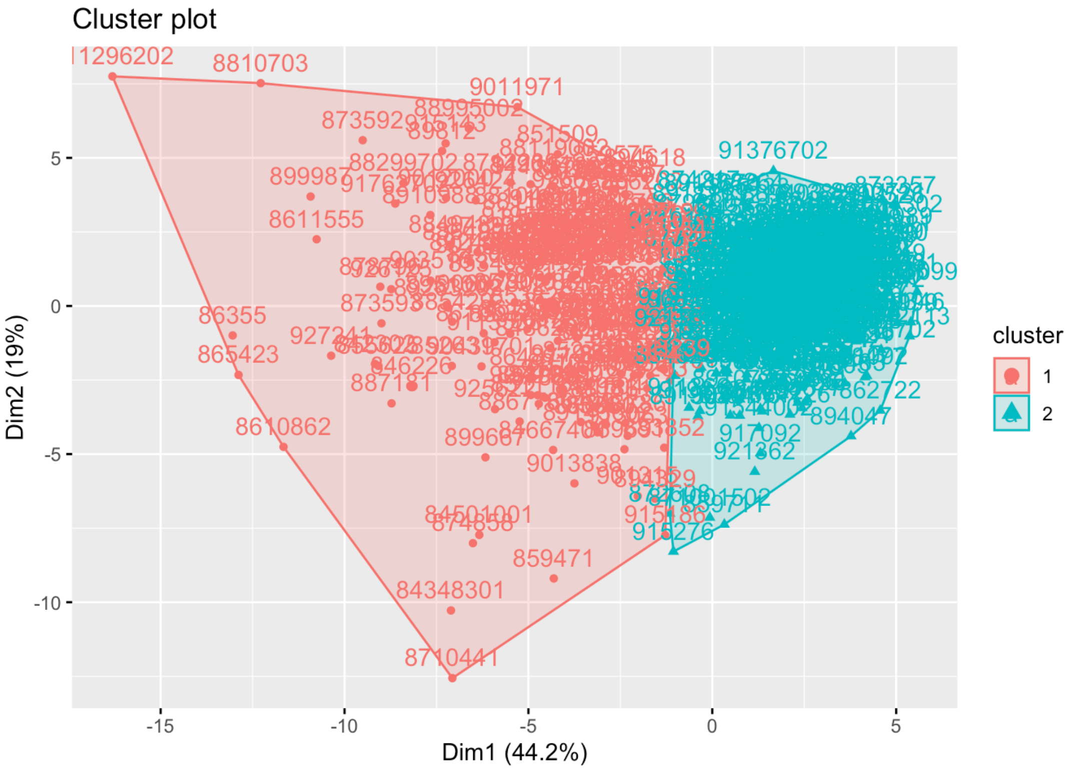
```
data %>%
  select_if(is_numeric) %>%
  scale() %>%
  fviz_nbclust(hcut, method = "wss")
```



It is not clear from the plot where the elbow is, but it seems to be around 2 clusters. This value has then been used to make the kmeans.

```
data1 <- data[,2:31]


data_km <- data1 %>%
  scale() %>%
  kmeans(centers = 2, nstart = 20)
```

```
data_km %>%
  fviz_cluster(data = data1 %>% select_if(is.numeric))
```



It is seen that the observations are split with a clean cut between the two clusters. Therefore, it seems that this method is working well on dividing the data between groups. However, the clustering does not show how the diagnoses are separated across the clusters; does the clustering distinguish between malignant and benign cases, or are they spread ambiguously? The following table shows the distribution of diagnoses across clusters.

```
table(data$diagnosis, data_km$cluster)
```

```
      1   2
  B  14 342
  M 175  37
```

From the table above it can be seen that cluster 1 contains 175 FNA malignant cancer patients, and 14 benign, while cluster 2 contains 342 benign cancer patients, and 37 malignant. Therefore, it seems that the clustering does a good job in separating the diagnoses across clusters. Hence, 96.06742% of benign cases and 82.54717% of malignant cases seem to be clustered correctly given a clustering dividing the diagnoses completely. This indicates that the clustering is separating the diagnoses well, and therefore it is assumed to be possible to investigate the features of the model according to the clusters.

```
data1 %>%
  bind_cols(cluster = data_km$cluster) %>%
  select_if(is_numeric) %>%
  group_by(cluster) %>%
  mutate(n = n()) %>%
  summarise_all(funs(mean))
```

cluster	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean ▸
<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	17.557	21.359	116.408	993.67	0.104921	0.158199
2	12.434	18.244	79.898	487.20	0.092206	0.077643

2 rows | 1-7 of 32 columns

It can be seen that the cluster containing the majority of malignant diagnoses takes on a higher value for the mean of all features. This could be indicating that on average, a higher value of all features respectively indicates a higher possibility of the diagnosis being malignant.

Supervised Learning

After the data set has been minimized with the maximal information, it is possible to perform supervised machine learning in order to predict whether an FNA analysis gives a diagnosis of malignant or benign cancer based on the features of the model.

Preprocessing

Split data

First, the data is split into two distinct sets. The majority of the observations in the dataset are placed in the training dataset, and are used to develop and optimize the model.

The other portion of the observations are placed into the test set which is withheld until one or two models are chosen as the methods that are most likely to succeed. The test set is then used as the final arbiter to determine the efficacy of the model(s). The test set should only be looked at once, when testing the model; otherwise, it becomes part of the modeling process.

Here, 75% of the data are allocated into the training dataset, while 25% of the data are allocated into the test dataset. The `initial_split()` function is used for this, taking the dataset and the proportion to be placed into the training set as arguments. Furthermore, the dependent variable *diagnosis* is set in the `strata` argument in order to use stratified sampling; this is done to keep the distributions of the dependent variable similar between the training and test datasets.

```
data_split <- data %>%
  initial_split(prop = 0.75, strata = diagnosis)
```

The `data_split` only contains the partitioning information about the split; to get the resulting datasets, the `training()` and `testing()` functions are applied.

```
data_train <- data_split %>%
  training()

data_test <- data_split %>%
  testing()
```

Preprocessing recipe

The `recipe` function is used for determining he data types of each column. This is done by transforming the data so that it can be encoded in a way suited for the model. Furthermore, the recipe is an object that defines a series of steps for data processing. It defines the steps without immediately executing them; it is only a specification of what should be done.

First, diagnosis is defined as the outcome while all other variables are defined as predictors through the `recipe()` . Thereafter, the numeric variables are centered to $mean = 0$ and scaled to $sd = 1$ using the `step_center()` and `step_scale()` , respectively. Fianlly, nominal values (here, only the dependent variable *diagnosis* is nominal) are turned into dummies using `step_dummy()` .

The recipe will apply these data manipulations to any data, including new data, when the recipe is used.

```
(data_recipe <- data_train %>%
  recipe(diagnosis ~ .) %>%
  step_center(all_numeric(), -all_outcomes()) %>%
  step_scale(all_numeric(), -all_outcomes()) %>%
  step_dummy(all_nominal(), -all_outcomes())
   )
```

```
Data Recipe

Inputs:

      role #variables
   outcome          1
 predictor         30

Operations:

Centering for all_numeric(), -all_outcomes()
Scaling for all_numeric(), -all_outcomes()
Dummy variables from all_nominal(), -all_outcomes()
```

Models

The models that are going to be used in fitting data and should be compared are chosen to be a logistic model, a gradient boosting (`xgboost`) framework and a random forest model. All the chosen models are applicable to a data set with a binomial outcome.

The logistic model is a highly used model for prediction of classification problems and is therefore chosen here. The XGBoost is designed to be highly flexible, efficient and portable and provides a parallel tree boosting that solves problems in a fast and accurate way. of The random forest model is chosen instead of a decision tree model since the random forest is a collection of randomly created decision trees and hence is assumed to give a better model; decision trees in general differ from logistic regression by the ability to segment the predictor space into rectangular regions.

The models are made by setting up a model structure with parsnip.

```
glm_model <- logistic_reg(mode = "classification") %>%
  set_engine("glm", family = binomial)
```

```
xg_model <- boost_tree(mode = "classification",
                       trees = tune(),
                       mtry = tune(),
                       min_n = tune(),
                       tree_depth = tune(),
                       learn_rate = tune()
                       ) %>%
  set_engine("xgboost")
```

```
rf_model <- rand_forest(mode = "classification",
                        mtry = tune(), min_n = tune()) %>%
    set_engine('randomForest')
```

Defining workflow

The workflow is used to bind models and preprocessing objects (recipes) together. To create a workflow, an empty workflow is initialized with the `workflow()` function. The `data_recipe` is used to preprocess the data for modeling and is added to the general workflow with the `add_recipe()` function. Hereafter, the workflow for each model is created by combining the general workflow `wf_general` with the models using the `add_model()` function. This computes a workflow for each model that bundles the respective model with the steps from the preprocessing recipe.

```
wf_general <- workflow() %>%
  add_recipe(data_recipe)

glm_wf <- wf_general %>%
  add_model(glm_model)

xg_wf <- wf_general %>%
  add_model(xg_model)

rf_wf <- wf_general %>%
  add_model(rf_model)
```

Resampling using K-fold cross validation

By using a training dataset, we only get one estimate of the model performance. To get more estimates herefore, K-fold cross validation is used. This is a technique that provides K estimates of model performance and is used to compare the different model types, here logistic regression, xgboost and random forest.

For the cross validation, the training data is randomly partitioned into K sets of roughly equal size (known as folds), which are used to perform K iterations of model fitting and evaluation. In each iteration, the related fold is reserved for model evaluation while the others are used for model training; eg. in the first iteration, fold 1 is reserved for model evaluation.

The stratification variable is set as *diagnosis* and a 5 x 5-folds cross validation is performed.

```
data_resample <- data_train %>%
  vfold_cv(strata = diagnosis,
           v = 5,#number of folds
           repeats = 5)
```

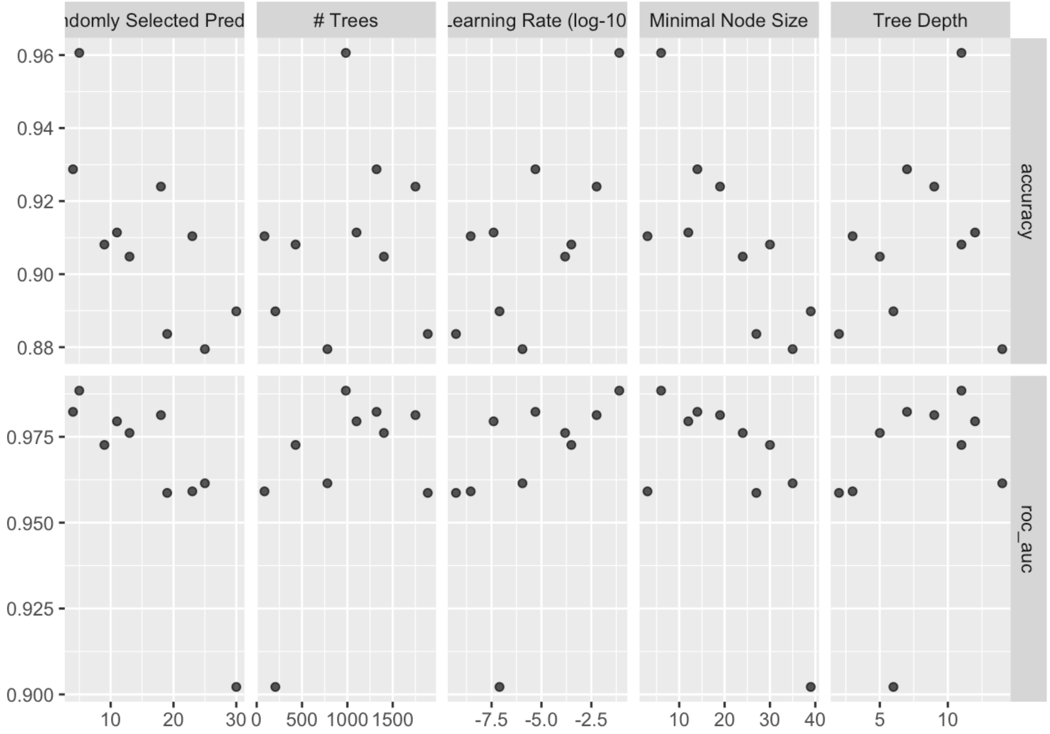
Tune hyperparameters

Hyperparameters are model parameters whose values are being set before training the model. They specify how the training is supposed to happen and control the model complexity. Hyperparameter tuning is accomplished using grid search, where a grid of hyperparameter values is generated. For each combination, cross validation is used to estimate model performance and the best performing combination is chosen.

The `tune()` function is used to label model hyperparameters. The values of each hyperparameter is set to `tune` within the model. Hereafter, the tuned workflows are passed into the `tune_grid()` function. Furthermore, the `data_resample` and a tuning grid are passed into the function.

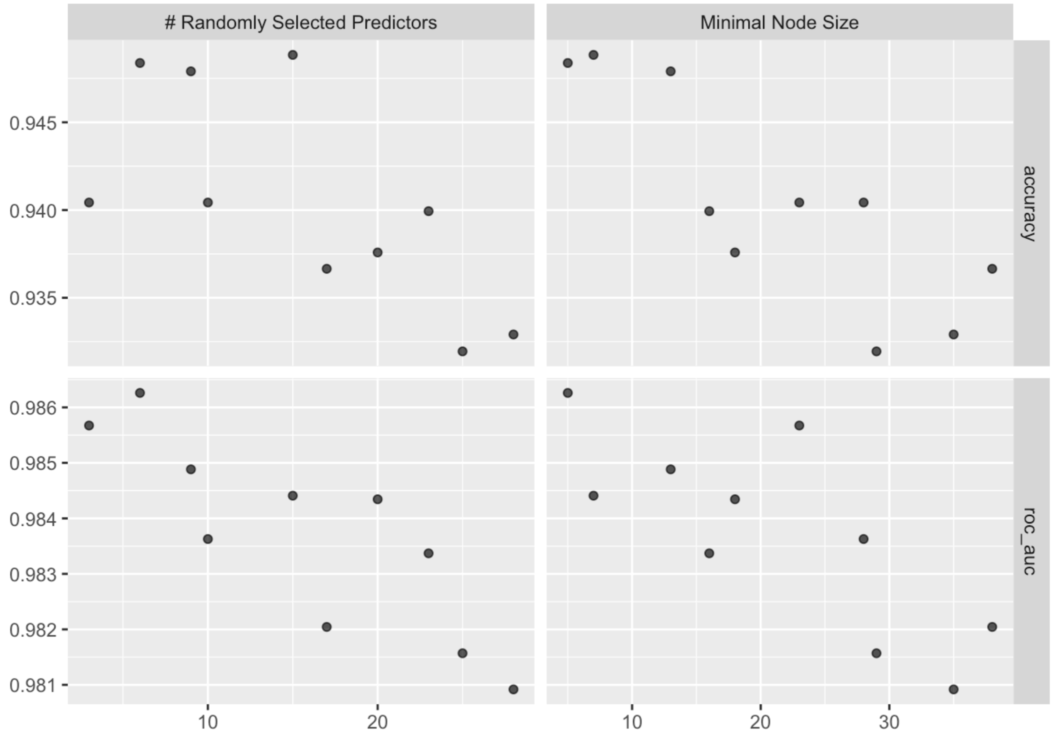
```
xg_tune <- tune_grid(
    xg_wf,
    resamples = data_resample,
    grid = 10
  )
```

```
xg_tune %>% autoplot()
```



```
rf_tune <- tune_grid(
    rf_wf,
    resamples = data_resample,
    grid = 10
  )
```

```
rf_tune %>% autoplot()
```



Finalizing workflows

The `select_best()` function is used to select the best performing hyperparameters for the models. It is applied to the tuned objects and returns a tibble with the hyperparameter values that produced the largest average performance metric value.

```
(best_param_xg <- xg_tune %>% select_best())
```

mtry	trees	min_n	tree_depth	learn_rate	.config
<int>	<int>	<int>	<int>	<dbl>	<fct>
5	983	6	11	0.079139	Preprocessor1_Model02

1 row

For the xgboost model, the best performing hyperparameters include a tree depth of 10 and a min_n of 8.

```
(best_param_rf <- rf_tune %>% select_best())
```

mtry	min_n	.config
<int>	<int>	<fct>
6	5	Preprocessor1_Model10

1 row

For the random forest model, the best performing parameters include a min_n of 17 and an mtry of 3.

The workflows for the tuned models are finalized using the best performing hyperparameters.

```
xg_final_wf <- xg_wf %>%
  finalize_workflow(parameters = best_param_xg)

rf_final_wf <- rf_wf %>%
  finalize_workflow(parameters = best_param_rf)
```

Fit models

The models are fitted using the workflows and the trained data.

```
fit_glm <- glm_wf %>%
  fit(data_train)

fit_xg <- xg_final_wf %>%
  fit(data_train)
```

```
[08:51:38] WARNING: amalgamation/../src/learner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metr
ic used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric
if you'd like to restore the old behavior.
```

```
fit_rf <- rf_final_wf %>%
  fit(data_train)
```

Compare model performance

The model performance is compared across the three models. This is done in order to choose the model best suited for prediction which will be used for the final fitting and prediction of the data.

```
(pred_collected <- tibble(
  truth = data_train %>% pull(diagnosis) %>% as.factor(),
  glm = fit_glm %>% predict(new_data = data_train) %>% pull(.pred_class),
  xg = fit_xg %>% predict(new_data = data_train) %>% pull(.pred_class),
  rf = fit_rf %>% predict(new_data = data_train) %>% pull(.pred_class)
  ) %>%
  pivot_longer(cols = -truth,
               names_to = 'model',
               values_to = '.pred')) %>% head()
```

truth <fct>	model <chr>	.pred <fct>
B	glm	B
B	xg	B
B	rf	B
B	glm	B
B	xg	B
B	rf	B

6 rows

From the table above it is seen that for the first few observations, all models predict correctly. However, we are also interested in how accurately they are in predicting overall. This is shown in the following table.

```
(pred_acc <- pred_collected %>%
  group_by(model) %>%
  accuracy(truth = truth, estimate = .pred) %>%
  select(model, .estimate) %>%
  arrange(desc(.estimate)))
```

model <chr>	.estimate <dbl>
glm	1.00000
rf	0.99296
xg	0.98826

3 rows

It is here seen that the logarithmic model is the most accurate with an accuracy of 100%. However, a 100% accurate model does not seem plausible. Furthermore, the random forest gives an accuracy of 99.3% while the xgboost gives an accuracy of 98.83%.

Final prediction

After comparing the model performances, it is now time to compute the final fit for the model. The best performing model is used for the fitting, and therefore, the logistic model is used. The predictions are then collected which makes it possible to compare the predictions to the true model.

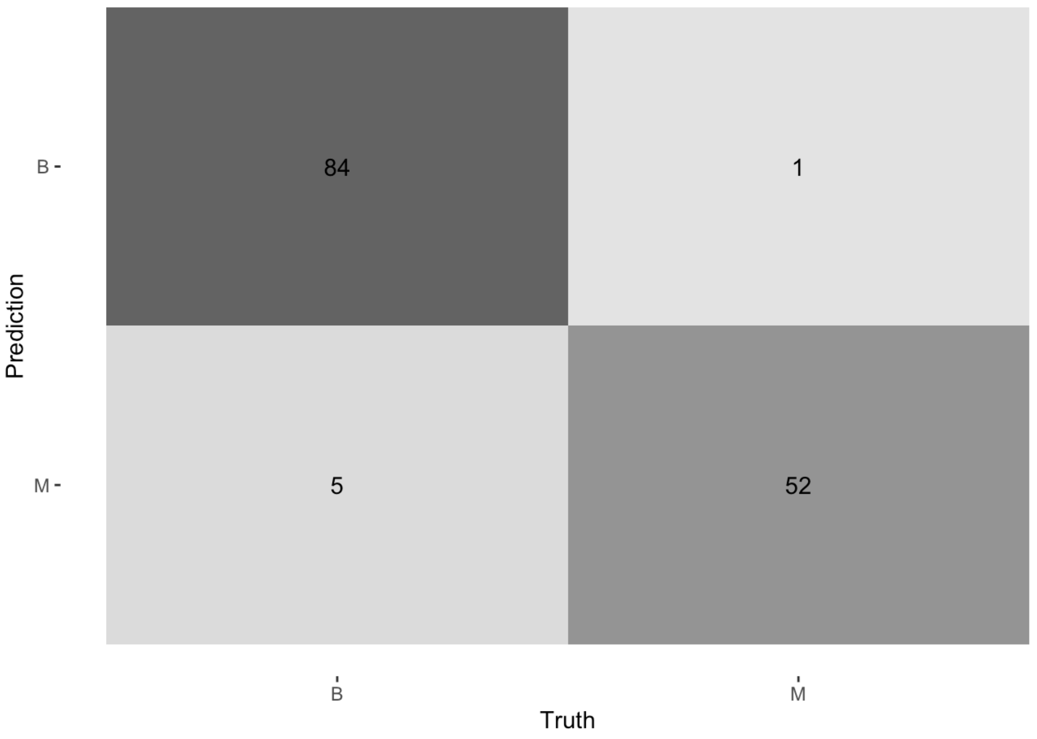
```
final_fit_glm <- glm_wf %>%
  last_fit(split = data_split)

final_fit_glm %>%
  collect_predictions() %>%
  select(c(2:3, 5:6)) %>%
  head()
```

.pred_B <dbl>	.pred_M <dbl>	.pred_class <fct>	diagnosis <fct>
0.00000000000000022204	0.99999999999999977796	M	M
0.00000000000000022204	0.99999999999999977796	M	M
0.99999999999999977796	0.00000000000000022204	B	M
0.00000000000000022204	0.99999999999999977796	M	M
0.00000000000000022204	0.99999999999999977796	M	M
0.99999999999999977796	0.00000000000000022204	B	B

6 rows

It is seen from the first few observations of the prediction that the model does a fairly good job in predicting the model outcome. To see the full distribution of prediction vs. true outcome. This is done using a confusion matrix.

```
final_fit_glm %>% collect_predictions() %>%
  conf_mat(truth = diagnosis, estimate = .pred_class) %>%
  autoplot(type = "heatmap")
```



From the confusion matrix it can be seen that the model is predicting the outcome well compared to the true outcome.

This can also be tested through the use of metrics. The metrics chosen for this are accuracy, sensibility and specificity. Accuracy is calculated as true values divided by all values:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Sensitivity is calculated as true positive divided by true positive and false negative:

$$sens = \frac{TP}{TP + FN}$$

Specificity is calculated as true negative divided by true negative and false positive:

$$spec = \frac{TN}{TN + FP}$$

where TP = true positive, TN = true negative, FP = false positive, FN = false negative. Here positive = benign and negative = malignant.

```
custom_metrics <- metric_set(accuracy, sens, specificity)
```

```
(preds <- final_fit_glm %>% collect_predictions() %>% custom_metrics(truth = diagnosis, estimate = .pred_class)
)
```

.metric	.estimator	.estimate
<chr>	<chr>	<dbl>
accuracy	binary	0.95775
sens	binary	0.94382
spec	binary	0.98113

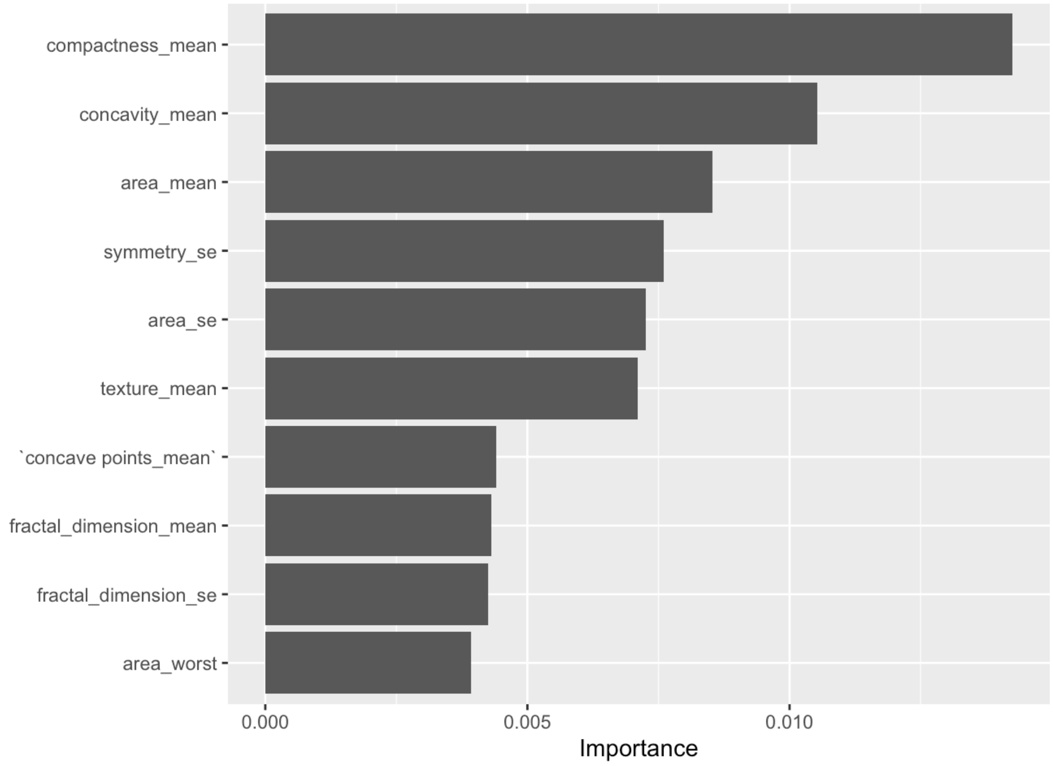
3 rows

From the table above, it can be seen that the model has managed to accurately predict 95.77% of the test observations from the FNA correctly. This is seen by series *accuracy*, where the number of true predictions is divided by all observations. In addition, it can be seen from the table that the model sensitivity is 94.38%, i.e. out of all predictions of benign cancer, only 5.62% are predicted incorrectly. On the other hand, the specificity computes the share of truly predicted outcomes of malignant with a specificity of 98.11%, i.e. out of all predictions of malignant cancer, only 1.89% are predicted incorrectly.

It is seen that the metrics take on high values indicating a well predicted model.

Finally, the variable importance will be plotted. Variable importance refers to how much a given model uses that variable to make accurate predictions.

```
final_fit_glm %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip(num_features = 10)
```



It is seen that the two most important variables to make accurate predictions in the logarithmic model are symmetry_se and texture_mean.

Final prediction, alternative model

A final prediction for the second best model, i.e. the random forest model, is also computed.

```
final_fit_rf <- rf_final_wf %>%
  last_fit(split = data_split)

final_fit_rf %>%
  collect_predictions() %>%
  select(c(2:3, 5:6)) %>%
  head()
```

.pred_B	.pred_M	.pred_class	diagnosis
<dbl>	<dbl>	<fct>	<fct>
0.182	0.818	M	M
0.064	0.936	M	M
0.334	0.666	M	M
0.160	0.840	M	M
0.076	0.924	M	M
0.966	0.034	B	B

6 rows

It is seen for the first few observation that the random forest predicts all outcomes correctly. This suggests that the random forest model actually does a better job predicting the true outcome than the logistic model which was deemed the best model for prediction in the comparison of model performance.

```
final_fit_rf %>% collect_predictions() %>%
  conf_mat(truth = diagnosis, estimate = .pred_class)
```

```
          Truth
Prediction  B  M
         B 89  4
         M  0 49
```

It is seen from the confusion matrix that the random forest model predicts almost all the outcomes accurately. This is further explored in the metrics set.

```
final_fit_rf %>% collect_predictions() %>%
  custom_metrics(truth = diagnosis, estimate = .pred_class)
```

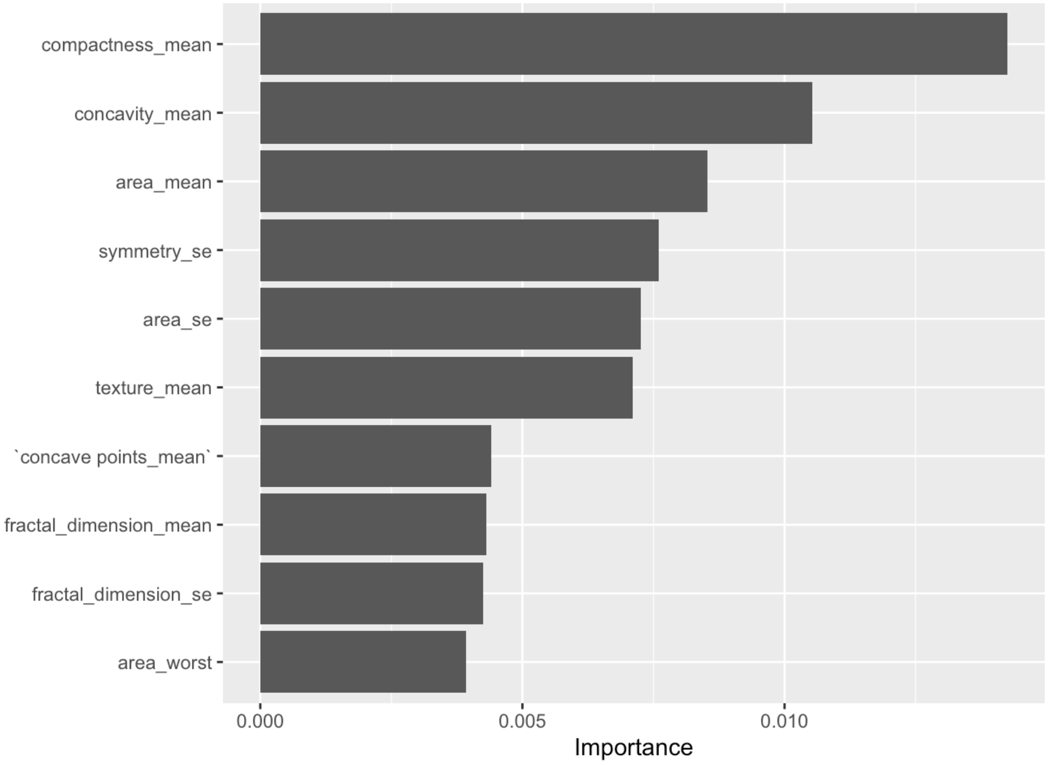
.metric <chr>	.estimator <chr>	.estimate <dbl>
accuracy	binary	0.97183
sens	binary	1.00000
spec	binary	0.92453

3 rows

From the metrics set it is seen that the random forest model predicts the model very well. As the confusion matrix and shown predictions, this also indicates that the random forest model is predicting the model better than the logistic model.

Finally, the variable importance will be plotted for the random forest model.

```
final_fit_glm %>%
  pluck(".workflow", 1) %>%
  pull_workflow_fit() %>%
  vip(num_features = 10)
```



It is seen that the two most important variables to make accurate predictions in the logarithmic model are symmetry_se and texture_mean as well as it is for the logarithmic model.

Github repository

Since it was not possible to make a functional Colab, we have chosen to link to our github repository:

https://github.com/emma-pedersen/SDS-M1 (https://github.com/emma-pedersen/SDS-M1)

---

1. Structured data is characterized by the fact that it can be meaningfully expressed in a tabular format, ie. in the form of rows and columns. In addition, this type of data also often contains quantitative and qualitative data.↩
2. https://www.rdocumentation.org/packages/dplyr/versions/0.3/topics/glimpse (https://www.rdocumentation.org/packages/dplyr/versions/0.3/topics/glimpse)↩
3. https://www.rdocumentation.org/packages/utils/versions/3.6.2/topics/head (https://www.rdocumentation.org/packages/utils/versions/3.6.2/topics/head)↩
4. https://www.rdocumentation.org/packages/skimr/versions/2.1.3/topics/skim (https://www.rdocumentation.org/packages/skimr/versions/2.1.3/topics/skim)↩
5. https://www.rdocumentation.org/packages/utils/versions/3.6.2/topics/str (https://www.rdocumentation.org/packages/utils/versions/3.6.2/topics/str)↩