Research Log Project Machine Learning

Diagnosing malignancy of breast masses using Machine Learning

Student: Vincent Talen

Student number: 389015

Class: BFV3

Study: Bioinformatics

Institute: Institute for Life Science & Technology

Teachers: Dave Langers (LADR) and Bart Barnard (BABA)

Date: 2022-09-28

Contents

1 Preparing R environment

For the data analysis and further processes multiple libraries are needed, they are loaded in here. A few other options/settings are configured and used scripts also loaded.

```r
# Load required packages from vector using invisible and lapply
packages <- c("tidyverse", "pander", "ggplot2", "data.table", "ggpubr")
invisible(lapply(packages, library, character.only = TRUE))
remove(packages) # Drop variable since it will not be used again

# Source functions
source("split_violin_plot.R")

# Disable printing 'table continues' lines between split sections
pander::panderOptions("table.continues", "")
# Change affix after table caption if it's a split table
pander::panderOptions("table.continues.affix", "(table continues below)")
```

2 Exploratory Data Analysis

2.1 About the chosen data set

Origin of the data

The data set that is used is the *Wisconsin Breast Cancer (Diagnostic) Data Set*, which is publicly available from the UCI Machine Learning Repository. There are two published research articles, from the same team of researchers, where the data set was first used, namely [1] and [2]. The samples for the data were collected from 569 patients at the University of Wisconsin Hospital with the goal of creating a machine learning model that was faster, improved the correctness and increased the objectivity of the diagnosis process of breast cancer.

Collection of the data

The data was gathered by first collecting the fine needle aspirates (FNA), which are expressed on a glass slide and stained. The images were generated by a color video camera mounted on top of a microscope, that projected the images with a 63x objective and 2.5x ocular into the camera. The images were then captured by a color frame grabber board as a 512x480, 8-bit-per-pixel Targa file.
The digitized image is then analyzed in the program Xcyt (custom made by Nick Street). First the user marks approximate initial boundaries of the nuclei and then the actual boundaries are further defined with an active contour model known as "Snake". In the end the snake reaches a point where it's curve accurately corresponds to the boundary of a cell nucleus. From the snake-generated cell nuclei boundaries 10 features are extracted, these are numerically modeled such that larger values will typically indicate a higher likelihood of malignancy.

The ten features that are extracted for each cell nucleus are the following:

1. Radius (mean of distances from center to points on the perimeter)
2. Texture (standard deviation of gray-scale values)
3. Perimeter (the total distance between all the points of the snake-generated boundary)
4. Area (the nuclear area is the sum of pixels on the interior, with half of the pixels of the perimeter)
5. Smoothness (local variation in radius lengths)
6. Compactness (perimeter^2 / area - 1.0)
7. Concavity (severity of concave portions of the contour)
8. Concave points (number of concave portions of the contour)
9. Symmetry (difference in length of perpendicular lines to the longest chord through the center, in both directions)
10. Fractal dimension (approximated using Mandelbrot's "coastline approximation" - 1)

For each feature and for every image, three final values were computed and saved to the data set, namely the mean, standard error and the extreme (largest) value.

Data structure and codebook

The data set has 569 instances/rows with 32 columns, an ID column, a classification column with the diagnosis (benign or malignant) and 30 columns describing the nuclei boundaries (10x mean/extreme/se).
Because the data set itself does not come with an annotated header with column names, a codebook has been manually made. This codebook has the abbreviated column name, the full column name, the data type and a description for each feature/column.

Below is an overview of the columns in the data set, shown using the contents of the codebook after it has been loaded in:

```r
codebook <- readr::read_delim("data/codebook.txt", delim = "|", show_col_types = FALSE)

# Pretty print the codebook (without descriptions) using pander
pander::pander(codebook[, 1:3], style = "rmarkdown", caption = "Overview of created codebook excluding 
```

Table 1: Overview of created codebook excluding descriptions

Column Name	Full Name	Type
id	ID	dbl
diagnosis	Diagnosis	fct
radius_mean	Mean Radius	dbl
texture_mean	Mean Texture	dbl
perimeter_mean	Mean Perimeter	dbl
area_mean	Mean Area	dbl
smoothness_mean	Mean Smoothness	dbl
compactness_mean	Mean Compactness	dbl
concavity_mean	Mean Concavity	dbl
concave_pts_mean	Mean Concave Points	dbl
symmetry_mean	Mean Symmetry	dbl
fractal_dim_mean	Mean Fractal Dimension	dbl
radius_se	Radius SE	dbl
texture_se	Texture SE	dbl
perimeter_se	Perimeter SE	dbl
area_se	Area SE	dbl
smoothness_se	Smoothness SE	dbl
compactness_se	Compactness SE	dbl
concavity_se	Concavity SE	dbl
concave_pts_se	Concave Points SE	dbl
symmetry_se	Symmetry SE	dbl
fractal_dim_se	Fractal Dimension SE	dbl
radius_worst	Worst Radius	dbl
texture_worst	Worst Texture	dbl
perimeter_worst	Worst Perimeter	dbl
area_worst	Worst Area	dbl
smoothness_worst	Worst Smoothness	dbl
compactness_worst	Worst Compactness	dbl
concavity_worst	Worst Concavity	dbl
concave_pts_worst	Worst Concave Points	dbl
symmetry_worst	Worst Symmetry	dbl
fractal_dim_worst	Worst Fractal Dimension	dbl

As can be seen, all the features are of the type double except the main diagnosis classification factor.

2.2 Loading in the data set

The data will be loaded in with the `read_csv` function from the `readr` package, this function returns the data as a tibble data frame. This function allows a vector with column names to be given with an argument, the names from the column `Column Name` of the codebook will be used.

```
data <- readr::read_csv("data/wdbc.data", col_names = codebook[[1]], show_col_types = FALSE)

# Print the amount of samples and columns
cat("Amount of samples:", dim(data)[1], "\tColumns in dataframe:", dim(data)[2], "\n")
```

```
## Amount of samples: 569   Columns in dataframe: 32
```

The amount of samples and columns are as expected, so in this aspect the data set has been read correctly. However, what is more important is if the values are read correctly, since the values are that what is actually going to be used.

```
str(data)
```

```
## spec_tbl_df [569 x 32] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ id                : num [1:569] 842302 842517 84300903 84348301 84358402 ...
##  $ diagnosis         : chr [1:569] "M" "M" "M" "M" ...
##  $ radius_mean       : num [1:569] 18 20.6 19.7 11.4 20.3 ...
##  $ texture_mean      : num [1:569] 10.4 17.8 21.2 20.4 14.3 ...
##  $ perimeter_mean    : num [1:569] 122.8 132.9 130 77.6 135.1 ...
##  $ area_mean         : num [1:569] 1001 1326 1203 386 1297 ...
##  $ smoothness_mean   : num [1:569] 0.1184 0.0847 0.1096 0.1425 0.1003 ...
##  $ compactness_mean  : num [1:569] 0.2776 0.0786 0.1599 0.2839 0.1328 ...
##  $ concavity_mean    : num [1:569] 0.3001 0.0869 0.1974 0.2414 0.198 ...
##  $ concave_pts_mean  : num [1:569] 0.1471 0.0702 0.1279 0.1052 0.1043 ...
##  $ symmetry_mean     : num [1:569] 0.242 0.181 0.207 0.26 0.181 ...
##  $ fractal_dim_mean  : num [1:569] 0.0787 0.0567 0.06 0.0974 0.0588 ...
##  $ radius_se         : num [1:569] 1.095 0.543 0.746 0.496 0.757 ...
##  $ texture_se        : num [1:569] 0.905 0.734 0.787 1.156 0.781 ...
##  $ perimeter_se      : num [1:569] 8.59 3.4 4.58 3.44 5.44 ...
##  $ area_se           : num [1:569] 153.4 74.1 94 27.2 94.4 ...
##  $ smoothness_se     : num [1:569] 0.0064 0.00522 0.00615 0.00911 0.01149 ...
##  $ compactness_se    : num [1:569] 0.049 0.0131 0.0401 0.0746 0.0246 ...
##  $ concavity_se      : num [1:569] 0.0537 0.0186 0.0383 0.0566 0.0569 ...
##  $ concave_pts_se    : num [1:569] 0.0159 0.0134 0.0206 0.0187 0.0188 ...
##  $ symmetry_se       : num [1:569] 0.03 0.0139 0.0225 0.0596 0.0176 ...
##  $ fractal_dim_se    : num [1:569] 0.00619 0.00353 0.00457 0.00921 0.00511 ...
##  $ radius_worst      : num [1:569] 25.4 25 23.6 14.9 22.5 ...
##  $ texture_worst     : num [1:569] 17.3 23.4 25.5 26.5 16.7 ...
##  $ perimeter_worst   : num [1:569] 184.6 158.8 152.5 98.9 152.2 ...
##  $ area_worst        : num [1:569] 2019 1956 1709 568 1575 ...
##  $ smoothness_worst  : num [1:569] 0.162 0.124 0.144 0.21 0.137 ...
##  $ compactness_worst: num [1:569] 0.666 0.187 0.424 0.866 0.205 ...
##  $ concavity_worst   : num [1:569] 0.712 0.242 0.45 0.687 0.4 ...
##  $ concave_pts_worst: num [1:569] 0.265 0.186 0.243 0.258 0.163 ...
##  $ symmetry_worst    : num [1:569] 0.46 0.275 0.361 0.664 0.236 ...
##  $ fractal_dim_worst: num [1:569] 0.1189 0.089 0.0876 0.173 0.0768 ...
```

Luckily it looks like the values for every column have correctly been read, but there is one thing that could still be changed; the diagnosis column. It would be more helpful if diagnosis was a factor and not just a character column.

```r
data$diagnosis <- factor(data$diagnosis, labels = c("Benign", "Malignant"))

ggplot(data, aes(x = diagnosis, fill = diagnosis)) + geom_bar() +
  geom_text(stat='count', aes(label = after_stat(count)), nudge_y = -15) +
  scale_fill_hue(direction = 1, h.start = 180) +
  ggtitle("Barplot of diagnosis column")
```

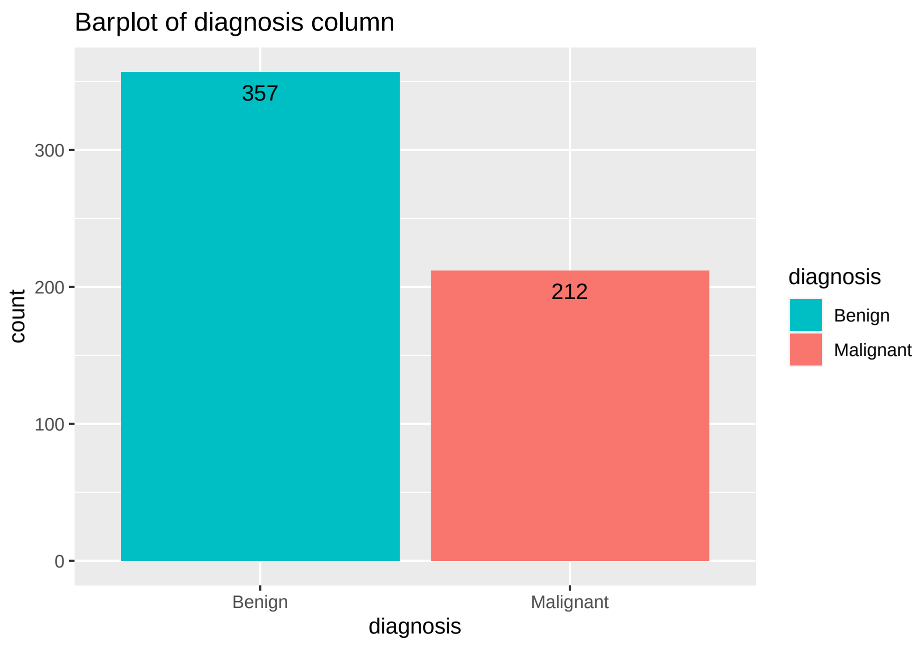

Figure 1: Barplot showing distribution of diagnosis classification labels

There are more benign cases than malignant, this means the data set is not entirely balanced and could cause bias if not handled correctly.

The data set has the `id` column, this column is not needed for the analysis that will be performed so it will therefore be dropped from the data frame. The `id` column could even cause small a hiccup when creating the machine learning model, since all the values are unique the model could use that as the only feature to predict the classification label. This would of course not work since unseen data will not have the same id's as the data used for training the machine learning algorithm.

```r
data <- dplyr::select(data, -id)
colnames(data)
```

```
##  [1] "diagnosis"         "radius_mean"       "texture_mean"
##  [4] "perimeter_mean"    "area_mean"         "smoothness_mean"
##  [7] "compactness_mean"  "concavity_mean"    "concave_pts_mean"
## [10] "symmetry_mean"     "fractal_dim_mean"  "radius_se"
## [13] "texture_se"        "perimeter_se"      "area_se"
## [16] "smoothness_se"     "compactness_se"    "concavity_se"
## [19] "concave_pts_se"    "symmetry_se"       "fractal_dim_se"
## [22] "radius_worst"      "texture_worst"     "perimeter_worst"
## [25] "area_worst"        "smoothness_worst"  "compactness_worst"
## [28] "concavity_worst"   "concave_pts_worst" "symmetry_worst"
## [31] "fractal_dim_worst"
```

2.3 Data inspection

Checking for missing values

It is important to have a good understanding of what the data is like, for example how the data is distributed and if there is data corruption. A few things that should be checked are if there are any outliers present, any data is skewed (an asymmetry in data distribution) or if there is any missing data.

```
cat("Missing values:", any(is.na(data)))
```

```
## Missing values: FALSE
```

Luckily there are no missing values in the data set. But now it is good to get an idea of what the values of the columns look like, what ranges do their values fall in? This can be done with the function `summary()` for all columns at the same time, it will create a basic statistics overview about the columns.

Overall data summary

```
pander::pander(summary(data), caption = "Summary with basic statistics about the data colums")
```

Table 2: Summary with basic statistics about the data colums (table continues below)

diagnosis	radius_mean	texture_mean	perimeter_mean
Benign :357	Min. : 6.981	Min. : 9.71	Min. : 43.79
Malignant:212	1st Qu.:11.700	1st Qu.:16.17	1st Qu.: 75.17
NA	Median :13.370	Median :18.84	Median : 86.24
NA	Mean :14.127	Mean :19.29	Mean : 91.97
NA	3rd Qu.:15.780	3rd Qu.:21.80	3rd Qu.:104.10
NA	Max. :28.110	Max. :39.28	Max. :188.50

area_mean	smoothness_mean	compactness_mean	concavity_mean
Min. : 143.5	Min. :0.05263	Min. :0.01938	Min. :0.00000
1st Qu.: 420.3	1st Qu.:0.08637	1st Qu.:0.06492	1st Qu.:0.02956
Median : 551.1	Median :0.09587	Median :0.09263	Median :0.06154
Mean : 654.9	Mean :0.09636	Mean :0.10434	Mean :0.08880
3rd Qu.: 782.7	3rd Qu.:0.10530	3rd Qu.:0.13040	3rd Qu.:0.13070
Max. :2501.0	Max. :0.16340	Max. :0.34540	Max. :0.42680

concave_pts_mean	symmetry_mean	fractal_dim_mean	radius_se
Min. :0.00000	Min. :0.1060	Min. :0.04996	Min. :0.1115
1st Qu.:0.02031	1st Qu.:0.1619	1st Qu.:0.05770	1st Qu.:0.2324
Median :0.03350	Median :0.1792	Median :0.06154	Median :0.3242
Mean :0.04892	Mean :0.1812	Mean :0.06280	Mean :0.4052
3rd Qu.:0.07400	3rd Qu.:0.1957	3rd Qu.:0.06612	3rd Qu.:0.4789

concave_pts_mean	symmetry_mean	fractal_dim_mean	radius_se
Max. :0.20120	Max. :0.3040	Max. :0.09744	Max. :2.8730

texture_se	perimeter_se	area_se	smoothness_se
Min. :0.3602	Min. : 0.757	Min. : 6.802	Min. :0.001713
1st Qu.:0.8339	1st Qu.: 1.606	1st Qu.: 17.850	1st Qu.:0.005169
Median :1.1080	Median : 2.287	Median : 24.530	Median :0.006380
Mean :1.2169	Mean : 2.866	Mean : 40.337	Mean :0.007041
3rd Qu.:1.4740	3rd Qu.: 3.357	3rd Qu.: 45.190	3rd Qu.:0.008146
Max. :4.8850	Max. :21.980	Max. :542.200	Max. :0.031130

compactness_se	concavity_se	concave_pts_se	symmetry_se
Min. :0.002252	Min. :0.00000	Min. :0.000000	Min. :0.007882
1st Qu.:0.013080	1st Qu.:0.01509	1st Qu.:0.007638	1st Qu.:0.015160
Median :0.020450	Median :0.02589	Median :0.010930	Median :0.018730
Mean :0.025478	Mean :0.03189	Mean :0.011796	Mean :0.020542
3rd Qu.:0.032450	3rd Qu.:0.04205	3rd Qu.:0.014710	3rd Qu.:0.023480
Max. :0.135400	Max. :0.39600	Max. :0.052790	Max. :0.078950

fractal_dim_se	radius_worst	texture_worst	perimeter_worst
Min. :0.0008948	Min. : 7.93	Min. :12.02	Min. : 50.41
1st Qu.:0.0022480	1st Qu.:13.01	1st Qu.:21.08	1st Qu.: 84.11
Median :0.0031870	Median :14.97	Median :25.41	Median : 97.66
Mean :0.0037949	Mean :16.27	Mean :25.68	Mean :107.26
3rd Qu.:0.0045580	3rd Qu.:18.79	3rd Qu.:29.72	3rd Qu.:125.40
Max. :0.0298400	Max. :36.04	Max. :49.54	Max. :251.20

area_worst	smoothness_worst	compactness_worst	concavity_worst
Min. : 185.2	Min. :0.07117	Min. :0.02729	Min. :0.0000
1st Qu.: 515.3	1st Qu.:0.11660	1st Qu.:0.14720	1st Qu.:0.1145
Median : 686.5	Median :0.13130	Median :0.21190	Median :0.2267
Mean : 880.6	Mean :0.13237	Mean :0.25427	Mean :0.2722
3rd Qu.:1084.0	3rd Qu.:0.14600	3rd Qu.:0.33910	3rd Qu.:0.3829
Max. :4254.0	Max. :0.22260	Max. :1.05800	Max. :1.2520

concave_pts_worst	symmetry_worst	fractal_dim_worst
Min. :0.00000	Min. :0.1565	Min. :0.05504
1st Qu.:0.06493	1st Qu.:0.2504	1st Qu.:0.07146
Median :0.09993	Median :0.2822	Median :0.08004
Mean :0.11461	Mean :0.2901	Mean :0.08395
3rd Qu.:0.16140	3rd Qu.:0.3179	3rd Qu.:0.09208
Max. :0.29100	Max. :0.6638	Max. :0.20750

By glancing over the summary created above a few things can be noticed and questions can arise, for example the `area_mean`, `concave_pts_mean`, `radius_se`, `perimeter_se` and `area_worst` columns. These,

among other columns, have a wide range of values with their minimum or maximum values far from the quantiles or median.
This could mean that there are outliers in the data, the questions that arise because of this is if these points are actually outliers and if they should be excluded from the data. It can however not be determined with just the summary above if these are actually outliers and if they need to be removed.

Univariate analysis

To check if points are outliers the data needs to be visualized, this can be done by creating a box plot and/or density plot for each of the columns. Creating them separately for all 30 feature columns would result in 60 plots, which is a lot and perhaps too many. To reduce the amount of total plots, the density plots will instead be visualized using violin plots with the box plots inside of them. Because printing all the data of a feature column in a single box or violin does not show what is desired, the plots will be split on the diagnosis classification factor.

There is not a function in `ggplot` to create these split violin plots, the source code file `split_violin_plot.R` contains a function that uses ggplot as a basis to create split violin plots. Then, using this function from the source file, another function is created that assembles the full plot including the box plot, plot title and themes. After this a list of column names can be given and a plot will be generated for each of them and they will the be arranged with a common legend to the output file. So there is a clearer overview where the data can be compared the plots for the feature columns will be split on their specification, resulting in three arranged layouts with the mean-, standard error- and worst plots.

```r
createFeaturePlot <- function(col_name, figure_tab) {
  plot <- ggplot(data, aes(x = "", y = !!sym(col_name), fill = diagnosis)) +
    geom_split_violin(alpha = 0.6, trim = FALSE) +
    geom_boxplot(width = 0.2, alpha = 0.6, fatten = NULL, show.legend = F) +
    stat_summary(fun.data = "mean_se", geom = "pointrange", show.legend = F,
                 position = position_dodge(0.2), size = 0.3) +
    scale_fill_brewer(palette = "Dark2", name = "Diagnosis:") +
    ggtitle(dplyr::filter(codebook, `Column Name` == col_name)$`Full Name`) +
    theme_minimal() + labs(y = NULL, x = NULL, tag = figure_tab) +
    theme(plot.title = element_text(size = 9, face = "bold"))#, hjust = 0.5))
  return(list(plot))
}

createPlotGrid <- function(extension) {
  desired_col_names <- colnames(data)[colnames(data) %like% extension]
  figure_tabs <- letters[1:length(desired_col_names)]
  # Create plots and put them in a list
  plot_list <- mapply(createFeaturePlot, desired_col_names, figure_tabs)
  # Print the plots in an arranged grid with the legend at the bottom
  ggpubr::ggarrange(plotlist = plot_list, ncol = 4, nrow = 3,
                    common.legend = TRUE, legend = "bottom")
}
```

```r
# Create split violin plots for all mean feature columns
ggpubr::annotate_figure(
  createPlotGrid("_mean"),
  top = text_grob("Grid of violin plots with boxplots for each 'mean' feature column",
                  face = "bold", size = 14))
```

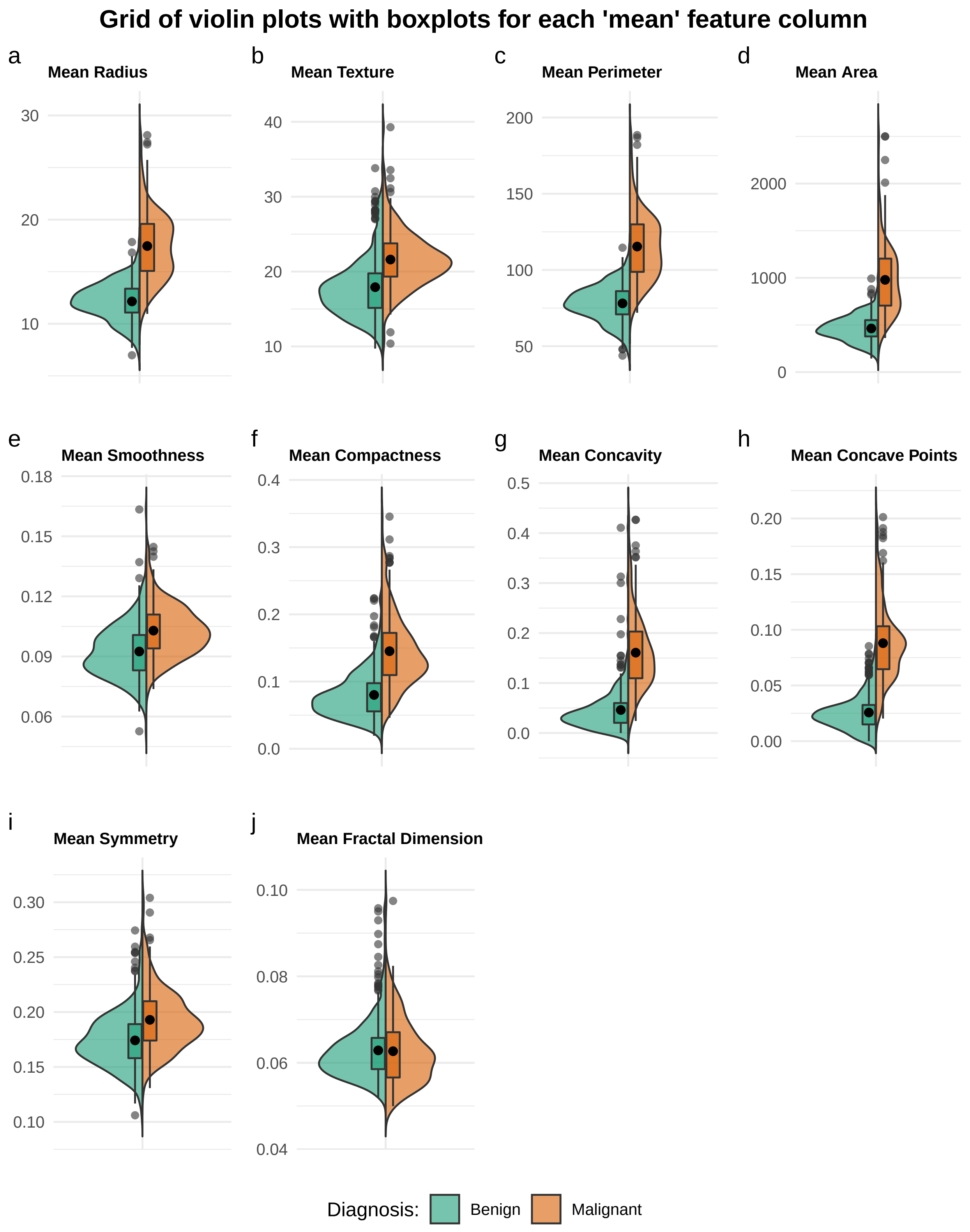

Figure 2: Split violin plots showing density joined with box plots showing quartiles and outliers, for all 'mean' feature columns

```
# Create split violin plots for all standard error feature columns
ggpubr::annotate_figure(
  createPlotGrid("_se"),
  top = text_grob("Grid of violin plots with boxplots for each 'standard error' feature column",
                  face = "bold", size = 14))
```

```
# Create split violin plots for all worst/extreme feature columns
ggpubr::annotate_figure(
  createPlotGrid("_worst"),
  top = text_grob("Grid of violin plots with boxplots for each 'worst' feature column",
                  face = "bold", size = 14))
```

Looking at the resulting plots it can easily be seen if there is a distinctive correlation between the classification factor and the values of feature columns. The couple of columns that stand out the most are the `Radius`, `Perimeter`, `Area`, `Concavity` and `Concave Points` feature columns, these all seem to have a clear distinction between the diagnosis classification.
Another thing that should be noted is that in most of the standard error columns a lot of outliers can be seen, for now nothing will be done with these but it is a good thing to know might problems arise later on.

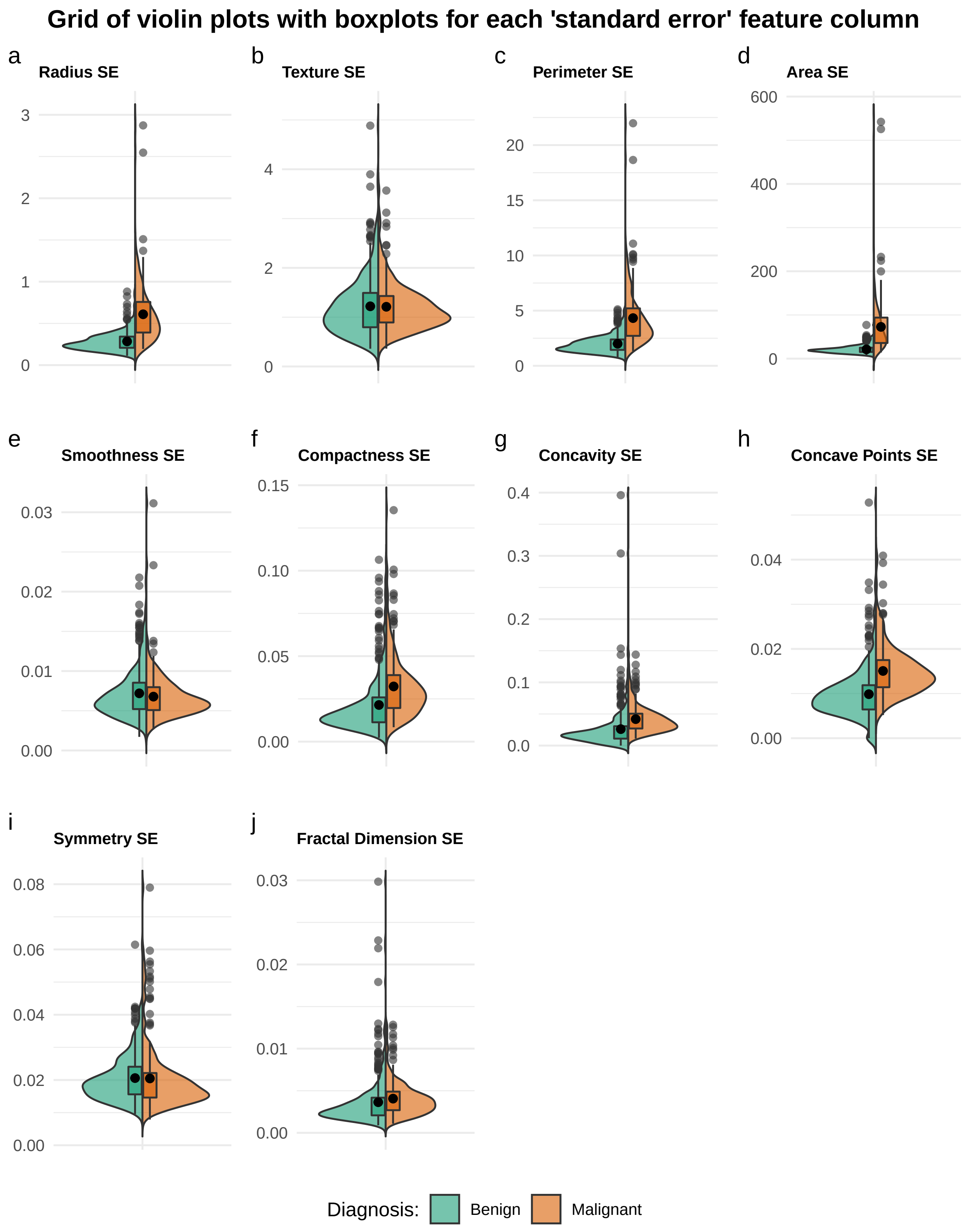

Figure 3: Split violin plots showing density joined with box plots showing quartiles and outliers, for all 'standard error' feature columns

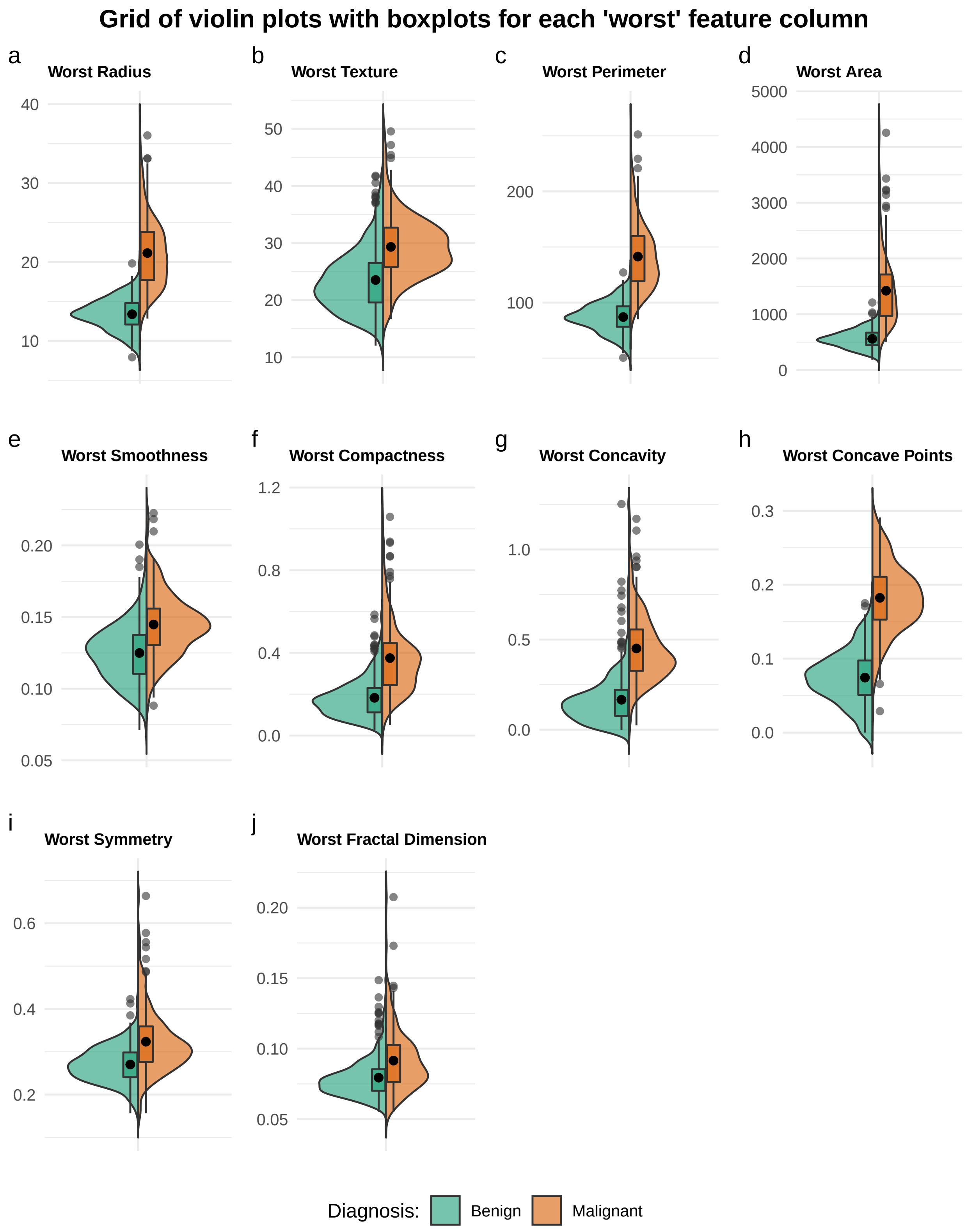

Figure 4: Split violin plots showing density joined with box plots showing quartiles and outliers, for all 'worst/extreme' feature columns

Bivariate analysis

Trying to visually identify correlations between features using the above plots is not very exact, a better way is to create a correlation matrix. In the correlation matrix all the possible pairs of features have their correlation calculated.
The correlation matrix can then be visualized as a heatmap, where the strength of correlations is represented with colors so they can easily be visually seen.

```r
# Create correlation matrix with only the numerical feature columns
cor_matrix <- stats::cor(select(data, -diagnosis))
cor_matrix <- tibble::as_tibble(cor_matrix)

# Insert column with feature names and set it as the left most column
column_names <- colnames(cor_matrix)
cor_matrix <- cor_matrix %>%
    dplyr::mutate(col_names = all_of(column_names)) %>%
    dplyr::select(31, 1:30)

# Show first five columns of correlation matrix
pander::pander(head(cor_matrix[1:5], n = 5))
```

col_names	radius_mean	texture_mean	perimeter_mean	area_mean
radius_mean	1	0.3238	0.9979	0.9874
texture_mean	0.3238	1	0.3295	0.3211
perimeter_mean	0.9979	0.3295	1	0.9865
area_mean	0.9874	0.3211	0.9865	1
smoothness_mean	0.1706	-0.02339	0.2073	0.177

Before the correlation matrix can be used to create a heatmap it needs to be converted to long format, this is because of the way R and ggplot read and use data.

```r
# Convert data to long format so a heatmap can be made from it
cor_matrix_long <- tidyr::pivot_longer(data = cor_matrix, cols = all_of(column_names),
                                       names_to = "variable", values_to = "cor")
# Show what the data looks like now
pander::pander(head(cor_matrix_long, n = 5))
```

col_names	variable	cor
radius_mean	radius_mean	1
radius_mean	texture_mean	0.3238
radius_mean	perimeter_mean	0.9979
radius_mean	area_mean	0.9874
radius_mean	smoothness_mean	0.1706

In the matrix the features are not physically paired as actual instances, in the long format the features are now saved as pairs in rows with the correlation score. This way the two columns, each with a feature of a pair, can be used as the axes of the heatmap plot.

```r
ggplot(data = cor_matrix_long, aes(x = col_names, y = variable, fill = cor)) +
  geom_tile() + labs(x = NULL, y = NULL) +
  scale_fill_gradient(high = "purple", low = "white" ) +
  theme(axis.text.x = element_text(angle = 45, hjust=1)) +
  ggtitle("Heatmap of column correlations")
```

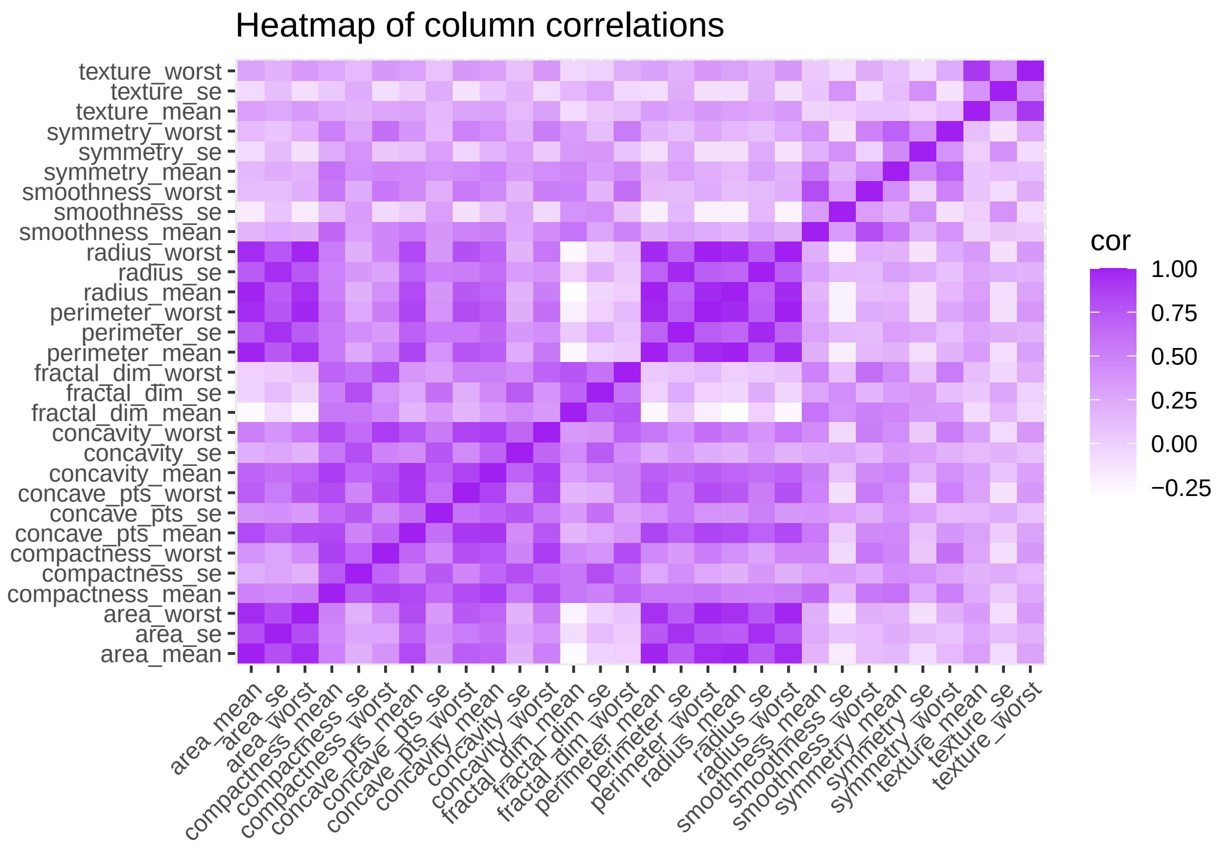

Figure 5: Heatmap visualizing pairwise correlation matrix of all feature colums

One thing that can immediately be noticed and should not be surprising is that the mean-, standard error- and worst feature columns of each nuclei boundary feature, i.e. of the radius, have a high correlation to each other.

When looking at the heatmap it can be seen very clearly that multiple features are heavily correlated together. Next, it could be a good idea to create scatterplots with trend lines of these heavily correlated feature pairs.

Data clustering

```r
# Tree, Scatter plot, PCA plot
```

References

[1] W.N. Street, W.H. Wolberg and O.L. Mangasarian. (1993), *Nuclear feature extraction for breast tumor diagnosis.*, 1993 International Symposium on Electronic Imaging: Science and Technology, volume 1905, pages 861-870, https://doi.org/10.1117/12.148698 (accessed Sep 16, 2022).

[2] O.L. Mangasarian, W.N. Street and W.H. Wolberg. (1995), *Breast cancer diagnosis and prognosis via linear programming*, Operations Research, volume 43, issue 4, pages 570-577, https://doi.org/10.1287/opre.43.4.570 (accessed Sep 17, 2022).