Breast cancer prediction

Classification with 'workflow_set' package

Damiano Pincolini

2024-08-31

1. PREFACE

1.1. Project goal

As stated in the on line documentation (https://workflowsets.tidymodels.org/), the goal of workflowsets package is to allow users to create and easily fit a large number of models and preprocessing recipes. In fact, workflowsets can create a workflow set that holds multiple workflow objects. These objects can be created by crossing all combinations of preprocessors (e.g., formula, recipe, etc) and model specifications. This set can be tuned or resampled using a set of specific functions. Aiming to better understand how this package works, this project is conceived as a pretext for experimenting this tool from the R's tidymodels ecosystem.

I have picked Breast Cancer Wisconsin (Diagnostic) dataset from kaggle.com (https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset?select=breast-cancer.csv) which is suitable for the purpose of this work. Specifically, it contains some outcome of some cells analysis made to discover if a cancer exists or not. Breast cancer starts when cells in the breast begin to grow out of control. These cells usually form tumors that can be seen via X-ray or felt as lumps in the breast area. Starting from some measures of these cells, the key challenges is how to classify tumors as malignant (cancerous) or benign(non cancerous).

With this aim in mind, I want to train and test some different models combined with some different preprocess recipes.

1.2. Loading packages

```
pacman::p_load(tidyverse,
               SmartEDA,
               DescTools,
```

```
                factoextra,
                ggcorrplot,
                GGally,
                ggforce,
                corrplot,
                cowplot,
                tidymodels,
                randomForest,
                kknn,
                kernlab,
                broom,
                themis,
                rpart.plot,
                vip,
                shapviz,
                knitr)
```

1.3. Data loading and content analysis

After downloading and saving the file breast-cancer.csv from kaggle (see link above), I save into R environment.

```
DataOrigin <- read_csv("RawData/breastCancer.csv",
                       show_col_types = FALSE)
```

1.4. DATASET ANALYSIS

1.4.1. Dataset structure and datatype analysis

```
DataExp1 <- ExpData(DataOrigin, type=1)

DataExp1|>
  kable()
```

Descriptions	Value
Sample size (nrow)	569
No. of variables (ncol)	32
No. of numeric/interger variables	31

Descriptions	Value
No. of factor variables	0
No. of text variables	1
No. of logical variables	0
No. of identifier variables	1
No. of date variables	0
No. of zero variance variables (uniform)	0
%. of variables having complete cases	100% (32)
%. of variables having >0% and <50% missing cases	0% (0)
%. of variables having >=50% and <90% missing cases	0% (0)
%. of variables having >=90% missing cases	0% (0)

The dataset is pretty small (almost 600 rows and 32 columns). There's only one categorical variable (the target) and there's also an identifier column which is supposed to be useless because it does not provide with any predictive support.

The quality of the dataset is pretty high since there aren't any uncomplete case.

```
DataExp2 <- ExpData(DataOrigin, type=2)

DataExp2|>
  kable()
```

Index	Variable_Name	Variable_Type	Sample_n	Missing_Count	Per_of_Missing	No_of_distinct_values
1	id	numeric	569	0	0	569
2	diagnosis	character	569	0	0	2
3	radius_mean	numeric	569	0	0	456
4	texture_mean	numeric	569	0	0	479
5	perimeter_mean	numeric	569	0	0	522
6	area_mean	numeric	569	0	0	539
7	smoothness_mean	numeric	569	0	0	474
8	compactness_mean	numeric	569	0	0	537
9	concavity_mean	numeric	569	0	0	537
10	concave points_mean	numeric	569	0	0	542
11	symmetry_mean	numeric	569	0	0	432
12	fractal_dimension_mean	numeric	569	0	0	499
13	radius_se	numeric	569	0	0	540
14	texture_se	numeric	569	0	0	519
15	perimeter_se	numeric	569	0	0	533

Index	Variable_Name	Variable_Type	Sample_n	Missing_Count	Per_of_Missing	No_of_distinct_values
16	area_se	numeric	569	0	0	528
17	smoothness_se	numeric	569	0	0	547
18	compactness_se	numeric	569	0	0	541
19	concavity_se	numeric	569	0	0	533
20	concave points_se	numeric	569	0	0	507
21	symmetry_se	numeric	569	0	0	498
22	fractal_dimension_se	numeric	569	0	0	545
23	radius_worst	numeric	569	0	0	457
24	texture_worst	numeric	569	0	0	511
25	perimeter_worst	numeric	569	0	0	514
26	area_worst	numeric	569	0	0	544
27	smoothness_worst	numeric	569	0	0	411
28	compactness_worst	numeric	569	0	0	529
29	concavity_worst	numeric	569	0	0	539
30	concave points_worst	numeric	569	0	0	492
31	symmetry_worst	numeric	569	0	0	500
32	fractal_dimension_worst	numeric	569	0	0	535

Also from this point of view, it is possibile to appreaciate the absence of missing case and the dicotomical nature of the target variable.

Finally, let's have a look to the data type and to main values of features.

```
summary(DataOrigin)
```

```
       id             diagnosis          radius_mean      texture_mean  
 Min.   :     8670   Length:569         Min.   : 6.981   Min.   : 9.71  
 1st Qu.:   869218   Class :character   1st Qu.:11.700   1st Qu.:16.17  
 Median :   906024   Mode  :character   Median :13.370   Median :18.84  
 Mean   : 30371831                      Mean   :14.127   Mean   :19.29  
 3rd Qu.:  8813129                      3rd Qu.:15.780   3rd Qu.:21.80  
 Max.   :911320502                      Max.   :28.110   Max.   :39.28  
 perimeter_mean     area_mean      smoothness_mean   compactness_mean 
 Min.   : 43.79   Min.   : 143.5   Min.   :0.05263   Min.   :0.01938  
 1st Qu.: 75.17   1st Qu.: 420.3   1st Qu.:0.08637   1st Qu.:0.06492  
 Median : 86.24   Median : 551.1   Median :0.09587   Median :0.09263  
 Mean   : 91.97   Mean   : 654.9   Mean   :0.09636   Mean   :0.10434  
 3rd Qu.:104.10   3rd Qu.: 782.7   3rd Qu.:0.10530   3rd Qu.:0.13040  
```

```
Max.   :188.50   Max.   :2501.0   Max.   :0.16340   Max.   :0.34540
concavity_mean    concave points_mean symmetry_mean    fractal_dimension_mean
Min.   :0.00000   Min.   :0.00000     Min.   :0.1060   Min.   :0.04996
1st Qu.:0.02956   1st Qu.:0.02031     1st Qu.:0.1619   1st Qu.:0.05770
Median :0.06154   Median :0.03350     Median :0.1792   Median :0.06154
Mean   :0.08880   Mean   :0.04892     Mean   :0.1812   Mean   :0.06280
3rd Qu.:0.13070   3rd Qu.:0.07400     3rd Qu.:0.1957   3rd Qu.:0.06612
Max.   :0.42680   Max.   :0.20120     Max.   :0.3040   Max.   :0.09744
  radius_se        texture_se      perimeter_se       area_se
Min.   :0.1115   Min.   :0.3602   Min.   : 0.757   Min.   :  6.802
1st Qu.:0.2324   1st Qu.:0.8339   1st Qu.: 1.606   1st Qu.: 17.850
Median :0.3242   Median :1.1080   Median : 2.287   Median : 24.530
Mean   :0.4052   Mean   :1.2169   Mean   : 2.866   Mean   : 40.337
3rd Qu.:0.4789   3rd Qu.:1.4740   3rd Qu.: 3.357   3rd Qu.: 45.190
Max.   :2.8730   Max.   :4.8850   Max.   :21.980   Max.   :542.200
smoothness_se      compactness_se      concavity_se     concave points_se
Min.   :0.001713   Min.   :0.002252   Min.   :0.00000   Min.   :0.000000
1st Qu.:0.005169   1st Qu.:0.013080   1st Qu.:0.01509   1st Qu.:0.007638
Median :0.006380   Median :0.020450   Median :0.02589   Median :0.010930
Mean   :0.007041   Mean   :0.025478   Mean   :0.03189   Mean   :0.011796
3rd Qu.:0.008146   3rd Qu.:0.032450   3rd Qu.:0.04205   3rd Qu.:0.014710
Max.   :0.031130   Max.   :0.135400   Max.   :0.39600   Max.   :0.052790
 symmetry_se       fractal_dimension_se  radius_worst   texture_worst
Min.   :0.007882   Min.   :0.0008948    Min.   : 7.93   Min.   :12.02
1st Qu.:0.015160   1st Qu.:0.0022480    1st Qu.:13.01   1st Qu.:21.08
Median :0.018730   Median :0.0031870    Median :14.97   Median :25.41
Mean   :0.020542   Mean   :0.0037949    Mean   :16.27   Mean   :25.68
3rd Qu.:0.023480   3rd Qu.:0.0045580    3rd Qu.:18.79   3rd Qu.:29.72
Max.   :0.078950   Max.   :0.0298400    Max.   :36.04   Max.   :49.54
perimeter_worst    area_worst     smoothness_worst  compactness_worst
Min.   : 50.41   Min.   : 185.2   Min.   :0.07117   Min.   :0.02729
1st Qu.: 84.11   1st Qu.: 515.3   1st Qu.:0.11660   1st Qu.:0.14720
Median : 97.66   Median : 686.5   Median :0.13130   Median :0.21190
Mean   :107.26   Mean   : 880.6   Mean   :0.13237   Mean   :0.25427
3rd Qu.:125.40   3rd Qu.:1084.0   3rd Qu.:0.14600   3rd Qu.:0.33910
Max.   :251.20   Max.   :4254.0   Max.   :0.22260   Max.   :1.05800
concavity_worst  concave points_worst symmetry_worst   fractal_dimension_worst
Min.   :0.0000   Min.   :0.00000      Min.   :0.1565   Min.   :0.05504
1st Qu.:0.1145   1st Qu.:0.06493      1st Qu.:0.2504   1st Qu.:0.07146
Median :0.2267   Median :0.09993      Median :0.2822   Median :0.08004
Mean   :0.2722   Mean   :0.11461      Mean   :0.2901   Mean   :0.08395
3rd Qu.:0.3829   3rd Qu.:0.16140      3rd Qu.:0.3179   3rd Qu.:0.09208
Max.   :1.2520   Max.   :0.29100      Max.   :0.6638   Max.   :0.20750
```

To sum up what has arisen from this initial evaluation, dataset may need the following changes:

1. De-select "id" column which does not bring any information.
2. Convert "diagnosis" column from character to factor.
3. Rename predictors names by swapping blank spaces with "_" Since I haven't found any detail about the unit of measurement, I take for granted that the scale is the same for all predictors.

1.4.2. Data featuring (conversion to factor, binning, renaming, etc)

```
Data <- DataOrigin|>
  select(-id)|>
  mutate(diagnosis = as_factor(diagnosis))|>
  rename(concave_points_mean = 'concave points_mean',
         concave_points_se = 'concave points_se')
```

1.5. DATA PARTITIONING

```
set.seed(987, sample.kind = "Rounding")

DataSplit <- Data|>
  initial_split(prop = 0.80,
                strata = diagnosis)

DataTrain <- training(DataSplit)

DataTest <- testing(DataSplit)
```

I want to check that target variable's proportion in the train and test dataset is the same as in the original dataset.

```
prop.table(table(Data$diagnosis))|>
  kable()
```

Var1	Freq
M	0.3725835
B	0.6274165

```
prop.table(table(DataTrain$diagnosis))|>
  kable()
```

Var1	Freq
M	0.3722467
B	0.6277533

```
prop.table(table(DataTest$diagnosis))|>
  kable()
```

Var1	Freq
M	0.373913
B	0.626087

02. EDA

2.1 Target analysis

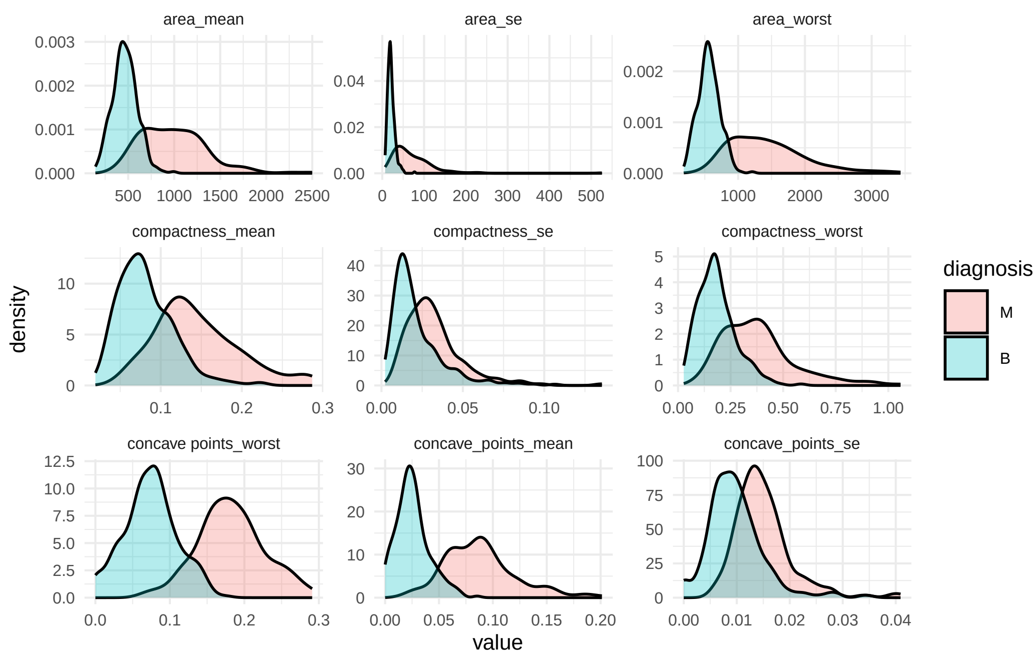
As already seen during previous data quality step target value is quite pretty balance: almost 37% of cases are malignous while the 63% are not.

2.2. Univariate analysis

2.2.1 distribution analysis for each predictor

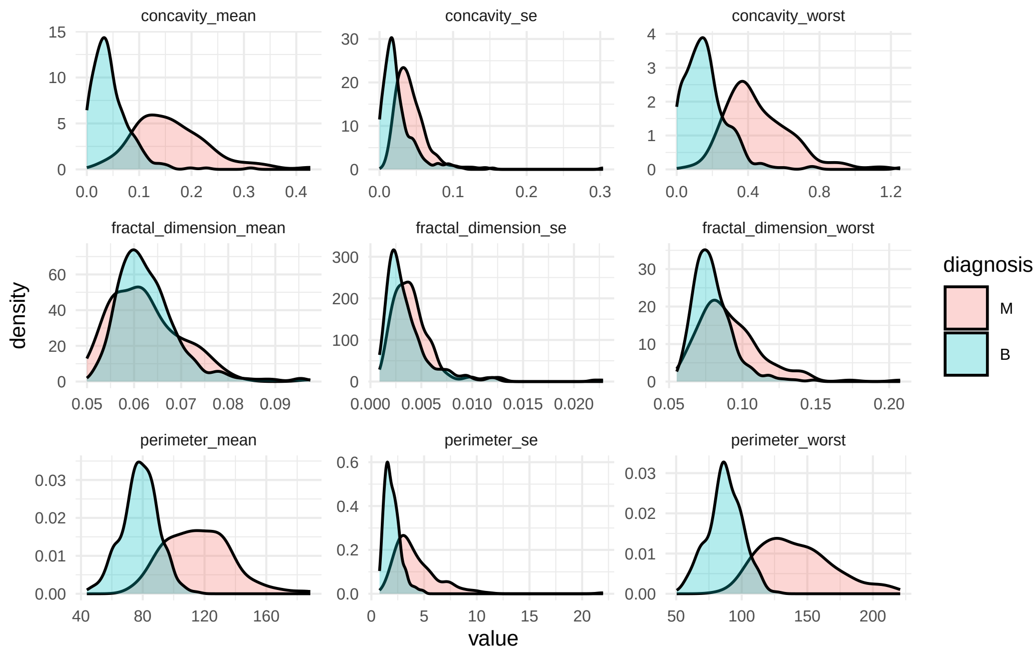
In order to have a bird's eye view of the different behaviour of how target variable "replies" to each single predictor, I prefer not to create a single plot for each feature; instead, I'll plot a wrapped collection of all of the plots I want to visualize in a single "frame". This should help reducing coding time and improving overall readability. To do that, I will reframe with pivot_longer command the DataTrain tibble bringing the 20 feature into one (longer) column that is going to include every single variable.

```r
options(scipen = 999)


DataTrainLong <- DataTrain|>
  pivot_longer(cols = -diagnosis,
               names_to = "variable",
               values_to = "value")

DataTrainLong|>
  ggplot(aes(x = value,
             fill = diagnosis))+
  geom_density(alpha = 0.30)+
  facet_wrap_paginate(~ variable,
                      ncol = 3,
                      nrow = 3,
                      scales = "free",
                      page = 1)+
  theme_minimal(base_size = 8)
```

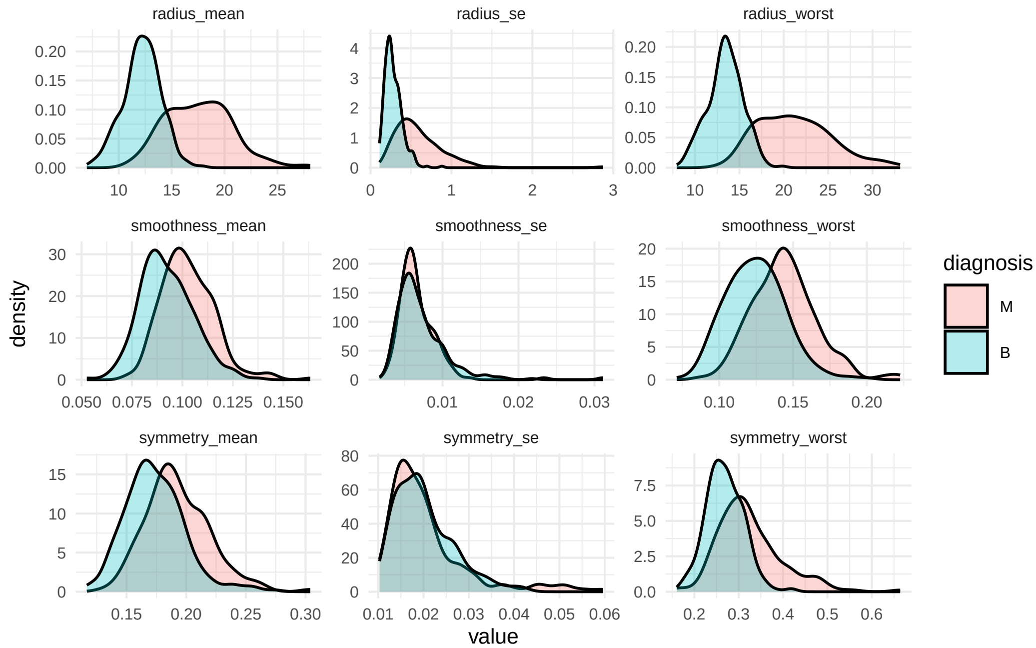



```r
DataTrainLong|>
  ggplot(aes(x = value,
             fill = diagnosis))+
  geom_density(alpha = 0.30)+
```

```
  facet_wrap_paginate(~ variable,
                      ncol = 3,
                      nrow = 3,
                      scales = "free",
                      page = 2)+
  theme_minimal(base_size = 8)
```




```
DataTrainLong|>
  ggplot(aes(x = value,
             fill = diagnosis))+
  geom_density(alpha = 0.30)+
  facet_wrap_paginate(~ variable,
                      ncol = 3,
                      nrow = 3,
                      scales = "free",
                      page = 3)+
  theme_minimal(base_size = 8)
```



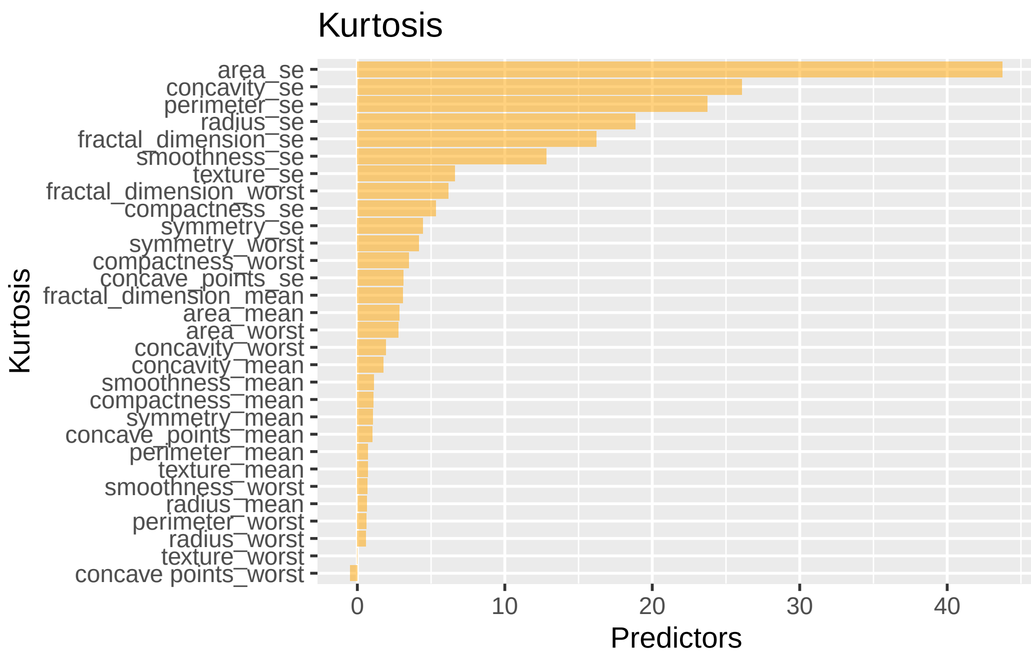
At a first glance, it is possible to see that:

1. Quite a lot of predictors show a different distribution based on target value (malignant/begninat). The trend is less prominent with some features that compute standard error ("concave_point_se", "concavity_se", "fractal_dimension_se").
2. A lot of features have some skewness.
3. Kurtosis is, as well, present in a slightly less significant way.

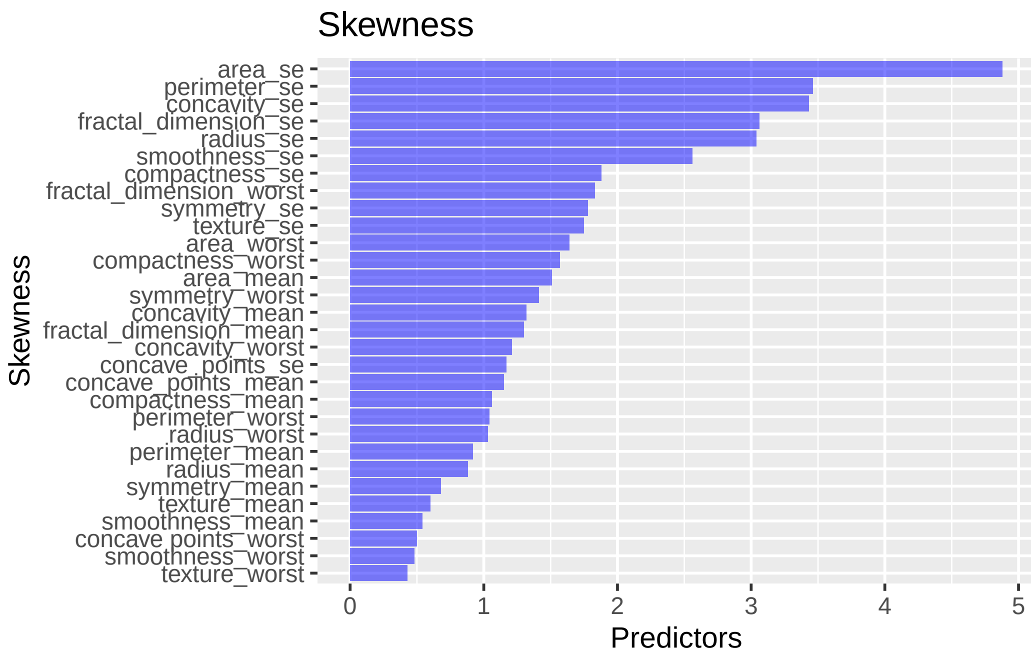
As far as skewness and kurtosis are concerned, SmartEDA package will help in getting another view of these measures.

```
EdaNum <- DataTrain|>
  ExpNumStat(by = "GA",
             gp = "diagnosis",
             Qnt = c(0.25, 0.75),
             Nlim = 2,
             MesofShape = 2,
             Outlier = TRUE,
             round = 2)

EdaDistributionShape <- EdaNum|>
  select(Vname, Group, Skewness, Kurtosis)|>
  filter(Group == 'diagnosis:All')|>
  arrange(desc(Skewness))
```

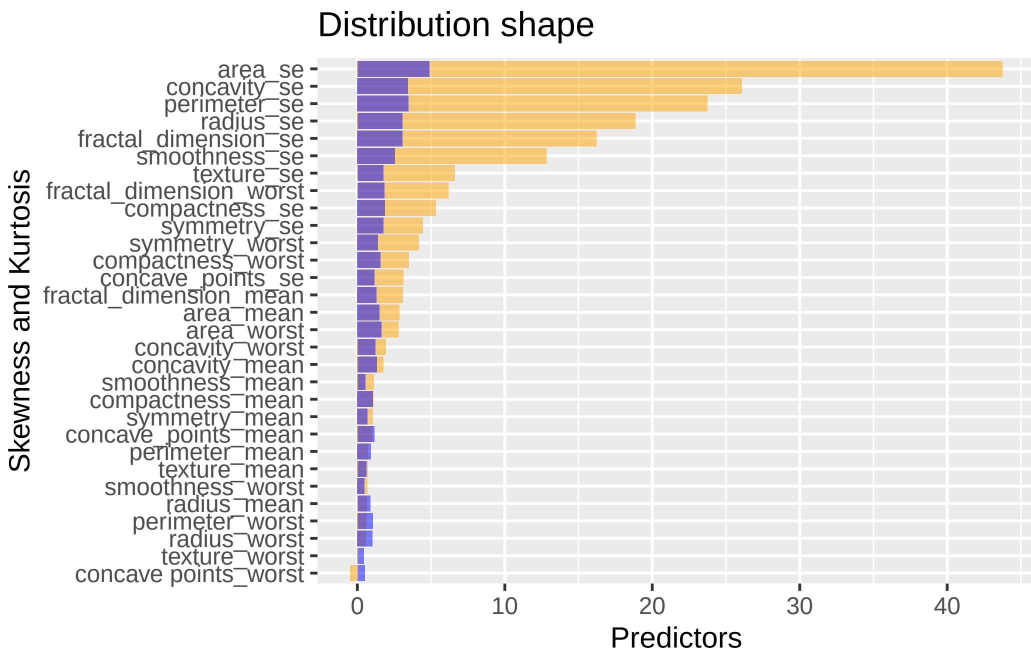
```r
EdaDistributionShape|>
  ggplot()+
  geom_col(aes(reorder(Vname, Kurtosis), Kurtosis), alpha = 0.5, fill = "orange")+
  coord_flip()+
  labs(x = "Kurtosis", y = "Predictors",
       title ="Kurtosis")
```




```r
EdaDistributionShape|>
  ggplot()+
  geom_col(aes(reorder(Vname, Skewness), Skewness), alpha = 0.5, fill = "blue")+
  coord_flip()+
  labs(x = "Skewness", y = "Predictors",
       title ="Skewness")
```



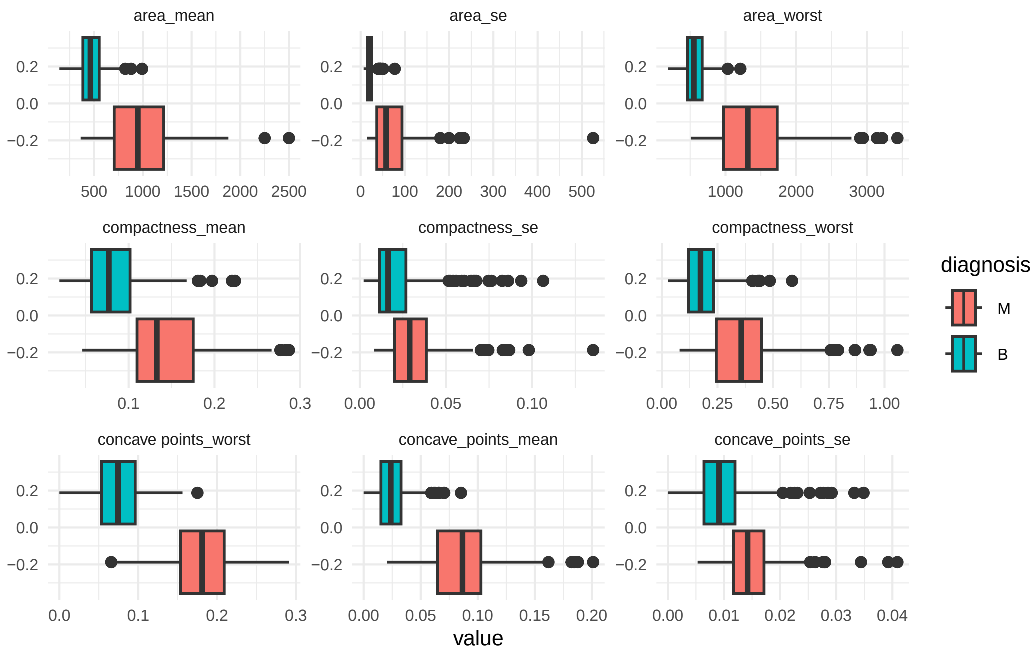

```
EdaDistributionShape|>
  ggplot()+
  geom_col(aes(reorder(Vname, Kurtosis), Kurtosis), alpha = 0.5, fill = "orange")+
  geom_col(aes(Vname, Skewness), alpha = 0.5, fill = "blue")+
  coord_flip()+
  labs(x = "Skewness and Kurtosis", y = "Predictors",
       title ="Distribution shape")+
  guides(fill = "color")
```



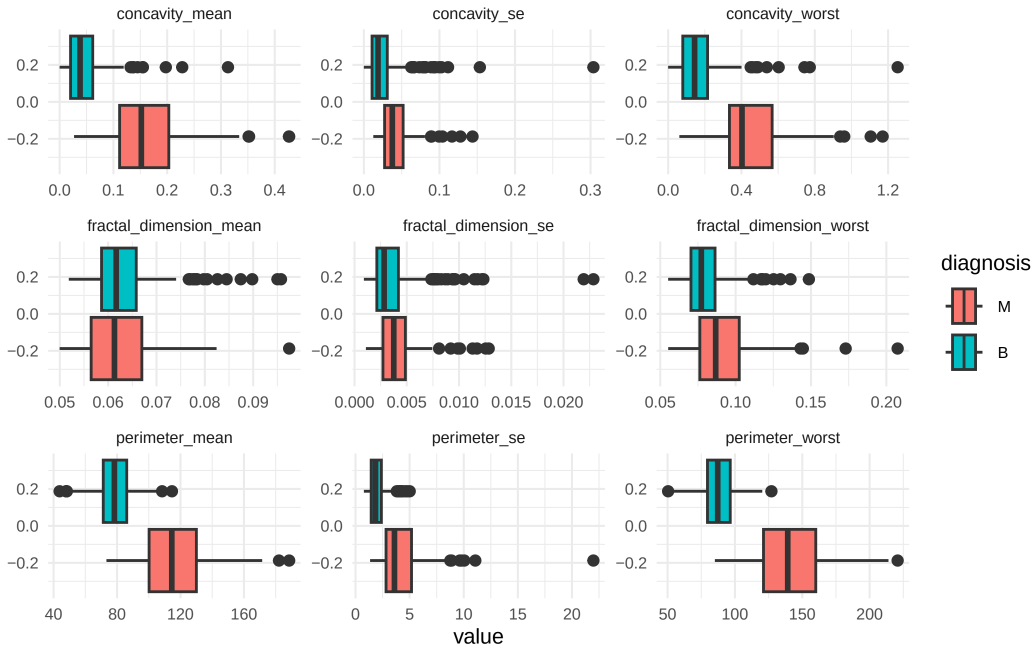
2.2.2. outlier detection

Not, it's time to try and understand something more about potential outliers.

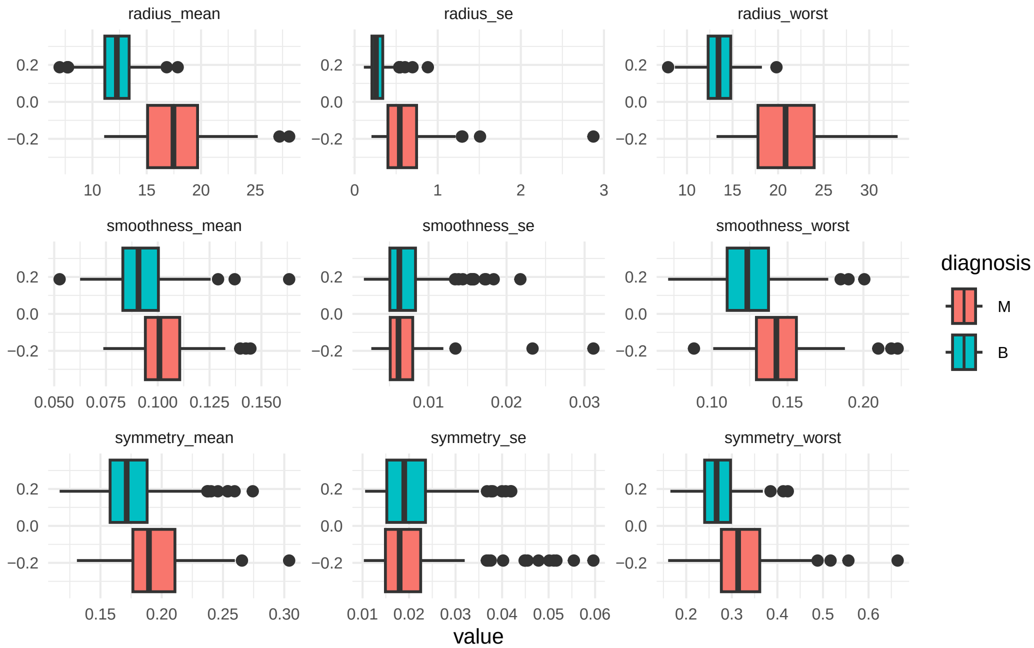
```
DataTrainLong|>
  ggplot(aes(x = value,
             fill = diagnosis))+
  geom_boxplot()+
  facet_wrap_paginate(~ variable,
                      ncol = 3,
                      nrow = 3,
                      scales = "free",
                      page = 1)+
  theme_minimal(base_size = 8)
```




```r
DataTrainLong|>
  ggplot(aes(x = value,
             fill = diagnosis))+
  geom_boxplot()+
  facet_wrap_paginate(~ variable,
                      ncol = 3,
                      nrow = 3,
                      scales = "free",
                      page = 2)+
  theme_minimal(base_size = 8)
```




```
DataTrainLong|>
  ggplot(aes(x = value,
             fill = diagnosis))+
  geom_boxplot()+
  facet_wrap_paginate(~ variable,
                      ncol = 3,
                      nrow = 3,
                      scales = "free",
                      page = 3)+
  theme_minimal(base_size = 8)
```

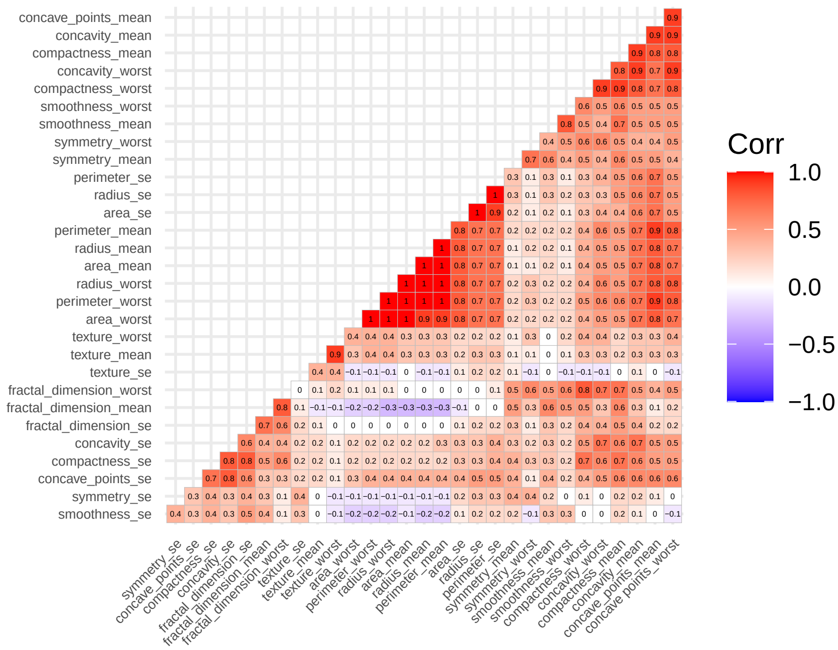


Due to the dataset's content and after observing the differences in the box plots for the two outcomes (malignant/benignant), it seems logical to accept "extreme" values (especially on the right side of the axis) as useful to detect cancer.

2.3. Multivariate analysis

2.3.1 Correlation between numerical predictors

```
EdaCorMatr <- round(cor(DataTrain[,c(2:31)], use="complete.obs"), 1)

ggcorrplot(EdaCorMatr,
           hc.order = TRUE,
           type = "lower",
           lab = TRUE,
           lab_size = 1,
           tl.cex = 5,
           digits = 1)
```



Clearly, some features are highly correlated and this issue (which can negatively affect the performance of machine learning model) has to be taken into account before training models.

2.3.2. Association between categorical predictors

There are no categorical predictor to look for association.

2.3.3. Principal Component Analysis (factoextra approach)

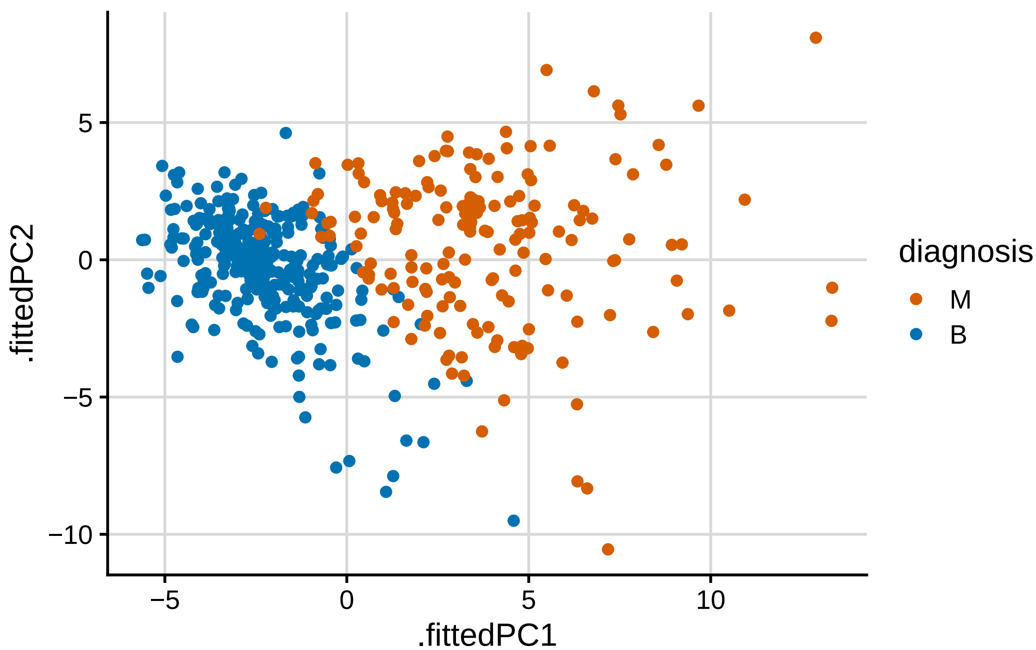
All predictors are numeric. This could be a good use case for PCA analysis in order to try to reduce multidimensionality.

I start by running the PCA and storing the result in a variable pca_fit. There are two issues to consider here. First, the prcomp() function can only deal with numeric columns, so we need to remove all non-numeric columns from the data. This is straightforward using the where(is.numeric) tidyselect construct. Second, we normally want to scale the data values to unit variance before PCA. We do so by using the argument scale = TRUE in prcomp().

```
EdaPca<- DataTrain|>
  select(where(is.numeric))|>
  prcomp(scale=TRUE)
```

Now, we want to plot the data in PC coordinates. In general, this means combining the PC coordinates with the original dataset, so we can color points by categorical variables present in the original data but removed for the PCA. We do this with the augment() function from broom, which takes as arguments the fitted model and the original data. The columns containing the fitted coordinates are called .fittedPC1, .fittedPC2, etc.

```
EdaPca %>%
  augment(DataTrain)|> # add original dataset back in
  ggplot(aes(.fittedPC1, .fittedPC2, color = diagnosis)) +
  geom_point(size = 1.5) +
  scale_color_manual(values = c(M = "#D55E00", B = "#0072B2")) +
  theme_half_open(12) +
  background_grid()
```

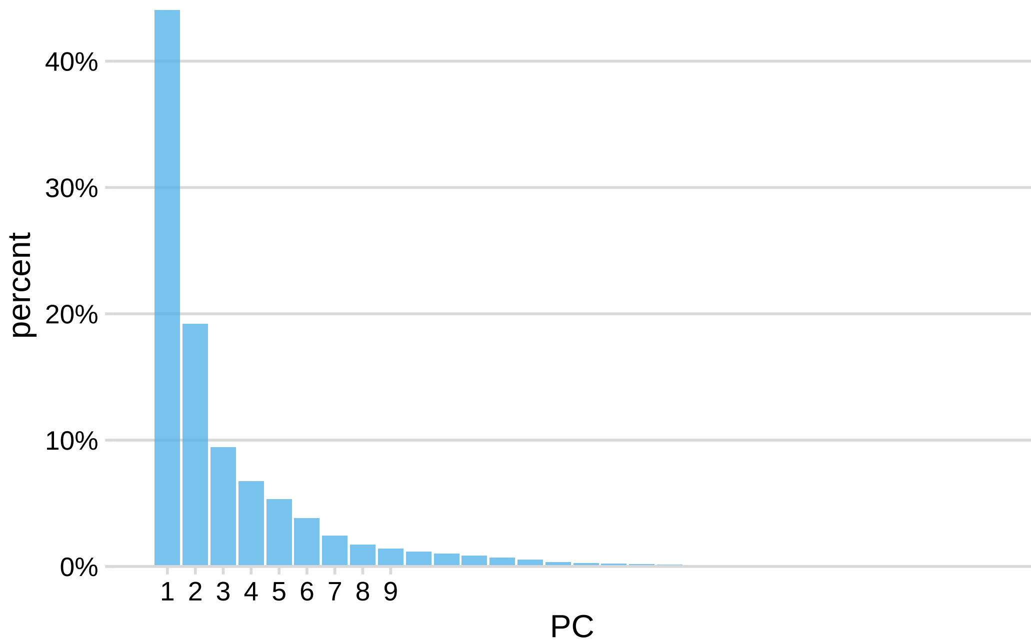


It is useful to look at the variance explained by each principal component. We can extract this information using the tidy() function from broom, now by setting the matrix argument to matrix = "eigenvalues". I'm using both a table and a couple of plots.

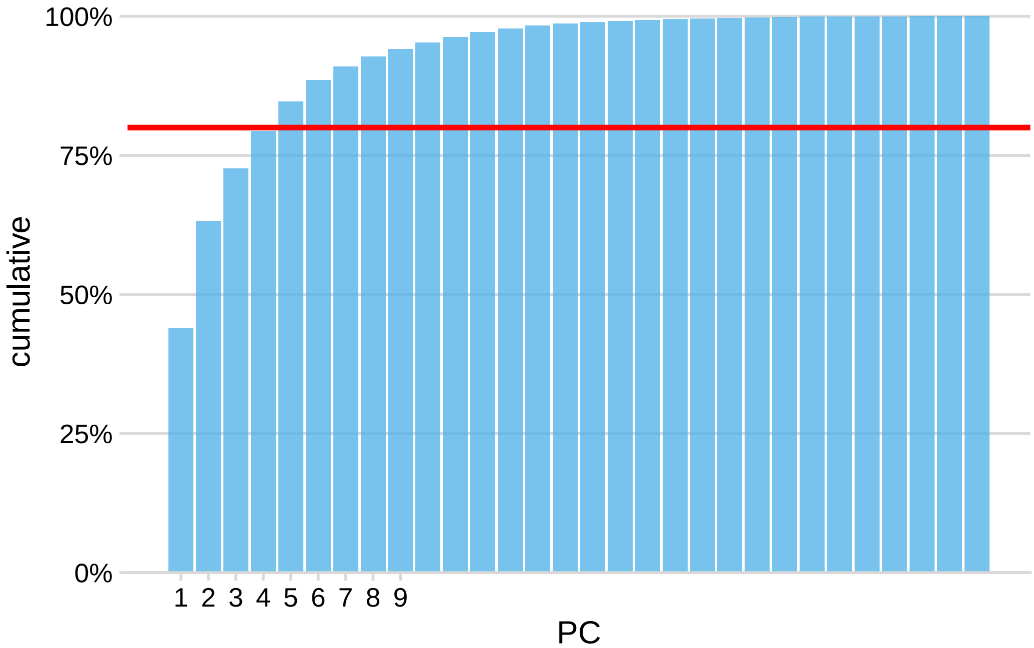
```
EdaPca |>
  tidy(matrix = "eigenvalues")|>
  kable()
```

PC	std.dev	percent	cumulative
1	3.6346082	0.44035	0.44035
2	2.3998270	0.19197	0.63232
3	1.6809809	0.09419	0.72651
4	1.4206459	0.06727	0.79378
5	1.2652049	0.05336	0.84714
6	1.0710080	0.03824	0.88538
7	0.8572031	0.02449	0.90987
8	0.7212044	0.01734	0.92721
9	0.6459981	0.01391	0.94112
10	0.5916971	0.01167	0.95279
11	0.5459631	0.00994	0.96272
12	0.5078211	0.00860	0.97132
13	0.4559524	0.00693	0.97825
14	0.4037734	0.00543	0.98368
15	0.3201487	0.00342	0.98710
16	0.2856169	0.00272	0.98982
17	0.2446197	0.00199	0.99181
18	0.2297396	0.00176	0.99357
19	0.2070808	0.00143	0.99500
20	0.1711129	0.00098	0.99598
21	0.1658533	0.00092	0.99690
22	0.1569287	0.00082	0.99772
23	0.1496584	0.00075	0.99846
24	0.1304760	0.00057	0.99903
25	0.1051977	0.00037	0.99940
26	0.0919169	0.00028	0.99968
27	0.0840083	0.00024	0.99992
28	0.0410185	0.00006	0.99997
29	0.0263576	0.00002	1.00000
30	0.0112426	0.00000	1.00000

```r
EdaPca %>%
  tidy(matrix = "eigenvalues")|>
  ggplot(aes(PC, percent))+
  geom_col(fill = "#56B4E9", alpha = 0.8)+
  scale_x_continuous(breaks = 1:9)+
  scale_y_continuous(labels = scales::percent_format(),
                     expand = expansion(mult = c(0, 0.01)))+
  theme_minimal_hgrid(12)
```




```
EdaPca %>%
  tidy(matrix = "eigenvalues")|>
  ggplot(aes(PC, cumulative))+
  geom_col(fill = "#56B4E9", alpha = 0.8)+
  scale_x_continuous(breaks = 1:9)+
  scale_y_continuous(labels = scales::percent_format(),
                     expand = expansion(mult = c(0, 0.01)))+
  geom_hline(aes(yintercept = 0.80),
             color = "red",
             linetype = 1,
             linewidth = 1)+
  theme_minimal_hgrid(12)
```



PCA gives back the following information:

1. 30 principal component are defined
2. The first two explain 63% of variance.
3. It takes five principal components to get an 85 percent explained variance.

2.4. Conclusions

1. From univariate analysis we have seen that some predictors show high skewness (14 of them reach a value of Fisher's Gamma Index varing from 1.50 to 4.88) and kurtosis (we have 14 variables with a Person's Beta index from 3.10 to 43.75). A transformation to handle skewness is appropriate (Yeo-Johnson).
2. Outliers do exist. without any domain knowledge, I assume every observation is fair and correct.
3. Quite a lot of predictors are correlated between themselves. It seems useful preprocess data in order to reduce multicollinearity.
4. PCA shows the relevance of 4 to 5 PC. Since I'm afraid of losing readibility of final result, I prefer, in this moment, not to consider PCA in preprocessing phase.

3. TRAINING AND TESTING THE CLASSIFICATION MODELS WITH WORKFLOW_SETS PACKAGE

In the context of a classification project, explorative data analysis has highlighted the need for a transformation to reduce skewness and a feature selection aimed to drop some correlated predictors. Since my aim here is to test and experiment the use of workflow_sets package, which can handle multiple recipe and models, I'm going to prepare the following recipes:

1. No preprocessing step a part from target variable definition.
2. Target variable definition + step_corr.
3. Target variable definition + step_corr + step_YeoJohnson.
4. Target variable definition + step_corr + step_YeoJohnson + step_norm.

The idea is to see how these different recipes affect ML models' final results.

As far as models are concerned, I will pick:

1. Decision tree
2. Random Forest
3. XG boost
4. knn
5. SVM

For all of them, I want to optimize one or more hyperparameters.

In order to select properly the recipe-model combination, I want to use the metric of sensitivity defined as True Positive/(True Positive + False Negative). Due to the specific nature of data (diasease prediction), the most important performance seems to be to correctly detect any effective positive case. Sensitivity tells how often the classifier predicts YES (malignant in this case) when it is actually YES.

3.1. Hyperparameter Tuning and metrics customization

```
set.seed(123, sample.kind = "Rounding")

WfSetCvFolds <- vfold_cv(DataTrain, v = 5)

custom_metrics <- metric_set(accuracy,
                             bal_accuracy,
                             specificity,
                             sensitivity,
                             f_meas)
```

3.2. Preprocessing recipes

```
WfSetRecipe1 <-
  recipe(diagnosis ~ ., data = DataTrain)

WfSetRecipe2 <-
  recipe(diagnosis ~ ., data = DataTrain)|>
  step_corr(all_numeric_predictors())

WfSetRecipe3 <-
  recipe(diagnosis ~ ., data = DataTrain)|>
  step_corr(all_numeric_predictors())|>
  step_YeoJohnson(all_numeric_predictors())

WfSetRecipe4 <-
  recipe(diagnosis ~ ., data = DataTrain)|>
  step_corr(all_numeric_predictors())|>
  step_YeoJohnson(all_numeric_predictors())|>
  step_normalize(all_numeric_predictors())
```

3.3. Model Specifications

```
WfSetModDt <-
    decision_tree(tree_depth = tune(),
                  cost_complexity = tune())|>
    set_engine("rpart")|>
    set_mode("classification")

WfSetModRf <-
  rand_forest(mtry = tune(),
              trees = tune(),
              min_n = tune())|>
  set_engine("ranger")|>
  set_mode("classification")

WfSetModXgb <-
  boost_tree(tree_depth = tune(),
             trees = tune())|>
  set_engine("xgboost")|>
  set_mode("classification")
```

```
WfSetModKnn <-
  nearest_neighbor(weight_func = tune(),
                   dist_power = tune(),
                   neighbors = tune())|>
  set_engine("kknn")|>
  set_mode("classification")

WfSetModSvm <-
  svm_linear(cost = tune())|>
  set_engine("kernlab")|>
  set_mode("classification")
```

3.4. Workflow Sets

```
WfSetWorkflows <- workflow_set(
  preproc = list(recPlain = WfSetRecipe1,
                 recCorr = WfSetRecipe2,
                 recYJ = WfSetRecipe3,
                 recNorm = WfSetRecipe4),
  models = list(Dt = WfSetModDt,
                Rf = WfSetModRf,
                Xgb = WfSetModXgb,
                Knn = WfSetModKnn,
                Svm = WfSetModSvm))
```

3.5. Tuning

```
WfSetGridCtrl<- control_grid(
  save_pred = TRUE,
  parallel_over = "resamples",
  save_workflow = TRUE)

WfSetGridResults <-
  WfSetWorkflows %>%
  workflow_map(
    seed = 1503,
    resamples = WfSetCvFolds,
    grid = 5,
```

```r
    metrics = custom_metrics,
    control = WfSetGridCtrl)

WfSetRankResults <- WfSetGridResults|>
  rank_results(rank_metric = "sensitivity",
               select_best = FALSE)|>
  filter(.metric %in% c("accuracy", "bal_accuracy", "specificity", "sensitivity", "f_meas"))
  select(wflow_id, .metric, mean, rank)|>
  group_by(wflow_id, .metric)|>
  summarize(CvAvgScore = mean(mean))|>
  filter(.metric == "sensitivity")|>
  arrange(desc(CvAvgScore))
```

The metric values are calculated from the cross-validation folds created from the training data (WfSetCvFolds).

This means these results represent the model's performance on the training data, using the cross-validation process to estimate how well the model will generalize to new data.

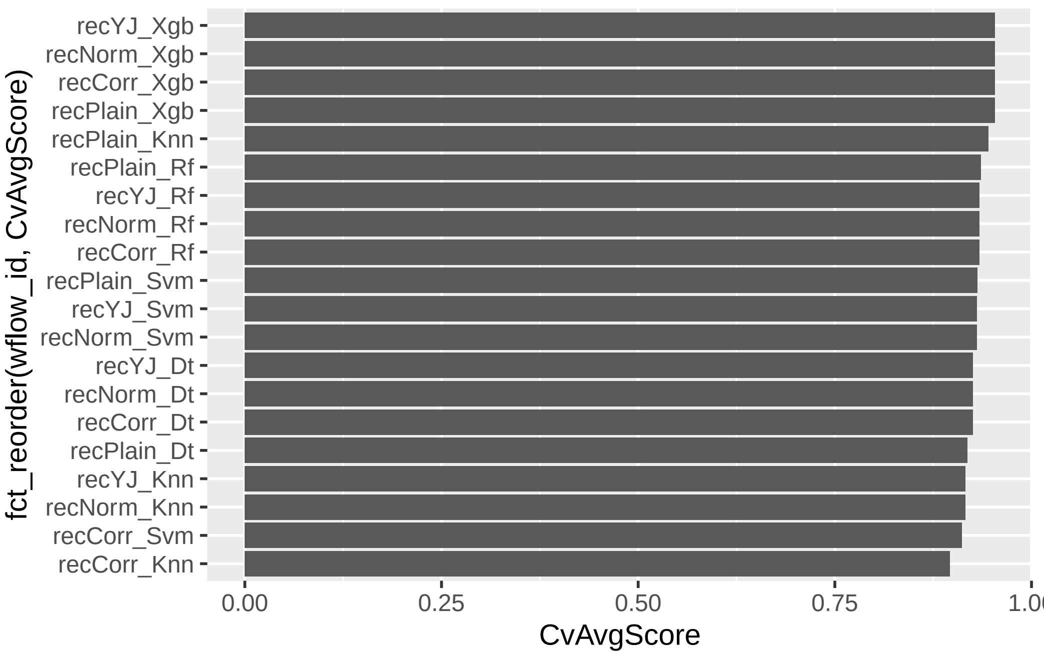
Let's see metric ranking.

```r
kable(WfSetRankResults)
```

wflow_id	.metric	CvAvgScore
recCorr_Xgb	sensitivity	0.9537062
recNorm_Xgb	sensitivity	0.9537062
recYJ_Xgb	sensitivity	0.9537062
recPlain_Xgb	sensitivity	0.9535415
recPlain_Knn	sensitivity	0.9450932
recPlain_Rf	sensitivity	0.9357708
recCorr_Rf	sensitivity	0.9336027
recNorm_Rf	sensitivity	0.9336027
recYJ_Rf	sensitivity	0.9336027
recPlain_Svm	sensitivity	0.9310447
recNorm_Svm	sensitivity	0.9306349
recYJ_Svm	sensitivity	0.9306349
recCorr_Dt	sensitivity	0.9252515
recNorm_Dt	sensitivity	0.9252515
recYJ_Dt	sensitivity	0.9252515
recPlain_Dt	sensitivity	0.9188267
recNorm_Knn	sensitivity	0.9160631

wflow_id	.metric	CvAvgScore
recYJ_Knn	sensitivity	0.9160631
recCorr_Svm	sensitivity	0.9116924
recCorr_Knn	sensitivity	0.8960520

```
WfSetRankResults|>
  ggplot(aes(x = fct_reorder(wflow_id, CvAvgScore), y = CvAvgScore))+
  geom_col()+
  coord_flip()
```



There are four recipe-model combinations that have produced the same highest result in term of sensitivity (the choosen metric): XgBoost model has led to the same sensitivity regardless the used preprocessing recipe. It seems that what has been done to the dataset before the training has not affected the final result.

Generally speaking and with some slight approssimation, we can see that as far as tree models are concerned, there is little if any impact of preprocessing on final results. Things changes for knn and SVM

Once I can choose the best performing model (after training), I want to test it. As said, there are four ex aequo workflows: I'll pick recCorr_Xgb combination (which has the most complete recipe).

4. BEST MODEL FINAL FIT AND TEST

I want to finalize the workflow set called "recCorr_Xgb"

```
WfSetBestResult1 <-
  WfSetGridResults %>%
  extract_workflow_set_result("recCorr_Xgb") %>%
  select_best(metric = "sensitivity")

WfSetBestResult1|>
  kable()
```

trees	tree_depth	.config
1801	2	Preprocessor1_Model1

```
WfSetTestResults1 <-
  WfSetGridResults %>%
  extract_workflow("recCorr_Xgb") %>%
  finalize_workflow(WfSetBestResult1) %>%
  last_fit(split = DataSplit,
           metrics = custom_metrics)  # Specify the custom metrics here


collect_metrics(WfSetTestResults1)|>
  kable()
```

.metric	.estimator	.estimate	.config
accuracy	binary	0.9217391	Preprocessor1_Model1
bal_accuracy	binary	0.9093992	Preprocessor1_Model1
specificity	binary	0.9583333	Preprocessor1_Model1
sensitivity	binary	0.8604651	Preprocessor1_Model1
f_meas	binary	0.8915663	Preprocessor1_Model1

```
WfSetTestPredictions <- WfSetTestResults1 %>%
  collect_predictions()

WfSetConfMatrix <- WfSetTestPredictions|>
  conf_mat(truth = diagnosis,
           estimate = .pred_class,
```

```r
           dnn=c("Prediction","Truth"),
           case_weights=NULL)

print(WfSetConfMatrix)
```

```
          Truth
Prediction  M  B
         M 37  3
         B  6 69
```

5. CONCLUSION AND LESSONS LEARNED

1. Workflow set package sis a very powerful tool that allows to increase speed and accuracy when a certain amount of recipes and models are involved in a ML project.
2. An explorative analysis is (as always) useful to optimize the dataset in order to properly preprocess dataset and thus realize proper training and testing phases.
3. It seems also useful to tune the hyperparameters. To do that, I've followed a cross validation approach and I let workflow_set to define hyperparameters' value, by defining grid lenght, thus avoiding manually defined value ranges which could potentially introduce any kind of bias.
4. In the case of this project (breast cancer), the workflow set approach, has led to choose and XG boost model whose performance has been the same regardless each of the four preprocessing recipes that have been used.
5. In this context, the sensitivity (the metric I've decided to primarily use to rank workflows prediction attitude) performance on the test set has been equal to 0.86 against the 0.97 obtained during training session. It seems quite a bit" large change in performance that could be read as an "overfitting hint".