Biomechanical features of orthopedic patients

HarvardX - PH125.9x Data Science - Capstone - Project 2

Emil Efendiev

03-March-2020

Contents

1 Introduction

1.1 Project Overview

In this project, we are going to build a machine learning algorithm for predicting abnormalities based on biomechanical attributes derived from the shape and orientation of the pelvis and lumbar spine. A public dataset called "Biomechanical features of orthopedic patients"(see the Sources section - (1)) will be used for this project. The dataset is part of the "ML-friendly Public Datasets" collection at Kaggle (link provided by the course organizers (2).

We are going to download the dataset, review and analyze it. We shall split the dataset into two partitions (train and test) and build a machine learning algorithm using different methods. After that, we are going

to review the performance of various methods. For better transparency, all code is explicitly shown in the report.

1.2 Project initialization

The dataset is located at the Kaggle website (1). However, because Kaggle requires login and password to download the dataset and to avoid creating a separate account (or revealing my password), I have downloaded the dataset from Kaggle and then uploaded it to GitHub.

Please note that the initial dataset consists of two different tables, which are identical except for the class column: one table consists of two classes (Abnormal and Normal), while the other one consists of three classes (Hernia, Spondylolisthesis and Normal). In the table with two classes, Hernia and Spondylolisthesis are merged and called Abnormal. We are going to use the table with two classes since our dataset is already relatively small, and having an additional class would decrease overall prediction accuracy. Also, some methods work only with binary outcomes.

This dataset consists of six predictors and an outcome (called class, in italic below):

1. pelvic incidence
2. pelvic tilt
3. lumbar lordosis angle
4. sacral slope
5. pelvic radius
6. grade (/degree) of spondylolisthesis
7. *class* (binary: Normal and Abnormal)

Our goal: Build an ML algorithm which would determine class based on predictors (as described above).

Below is the code which installs all required libraries, downloads the dataset and splits it to train and test subsets.

```
# Installing libraries
if(!require(tidyverse))
  install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret))
  install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table))
  install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(ranger))
  install.packages("ranger", repos = "http://cran.us.r-project.org")
if(!require(naivebayes))
  install.packages("naivebayes", repos = "http://cran.us.r-project.org")
if(!require(kernlab))
  install.packages("kernlab", repos = "http://cran.us.r-project.org")
if(!require(ggthemes))
  install.packages("ggthemes", repos = "http://cran.us.r-project.org")
if(!require(knitr))
  install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(e1071))
  install.packages("e1071", repos = "http://cran.us.r-project.org")

# Dataset URL
urlKG1 <- 'https://raw.githubusercontent.com/emil-efendiev/'
urlKG2 <- 'HarvardX---PH125.9x-Data-Science---Capstone/master/BFOP_2R.csv'
```

```
urlKG <- (paste(urlKG1,urlKG2,sep=""))
rm(urlKG1,urlKG2)

# importing dataset
df <- read.csv(urlKG)

# Split data into train (75%) and test sets (25%)
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

test_index <- createDataPartition(y = df$class, times = 1, p = 0.25, list = FALSE)

train <- df[-test_index,]
test <- df[test_index,]
```

In the next section, we are going to explore the dataset, build ML models and test them.

2 Methods and data analysis

To build and test an ML algorithm, the caret package is going to be used. This package has a simple syntax and allows us to build an ML algorithm using straightforward functions: train and predict. For the train function to work properly, we need to pass three elements (plus additional arguments if we wish to change default behaviour): predictors (x), outcome (y) and method to be used. Only the train dataset is going to be used for training purposes.

Then, once we train the model and save it in the relevant object, we are going to pass this model to the predict function. We also need to pass a set of predictors from the test set. The result of the prediction is then going to be cross-checked against the outcomes in the test set.

In terms of methods, the train function of the caret package supports 238 models/methods, of which 190 in one or another shape work with classification exercises. We are going to review five popular methods and compare results (based on the accuracy of predictions):

1. Generalized Linear Model (method = "glm")
2. k-Nearest Neighbors (method = "knn")
3. Random Forest (method = "ranger", additional library "ranger")
4. Naive Bayes (method = "naive_bayes", additional library "naivebayes")
5. Support Vector Machines with Polynomial Kernel (method = "svmPoly", additional library "kernlab")

One of the advantages of the train function is that in many cases, it tries several default tuning parameters to determine the best model.

Now let's explore the data.

2.1 Overview of the dataset

To avoid using the test dataset and gaining insights from it, in this section, we are going to review the train dataset. Let's start with simple code to see first rows.

```
head(train)
```

```
##   pelvic_incidence pelvic_tilt.numeric lumbar_lordosis_angle sacral_slope
## 1         63.02782           22.552586              39.60912     40.47523
## 2         39.05695           10.060991              25.01538     28.99596
## 5         49.71286            9.652075              28.31741     40.06078
## 6         40.25020           13.921907              25.12495     26.32829
## 7         53.43293           15.864336              37.16593     37.56859
## 8         45.36675           10.755611              29.03835     34.61114
##   pelvic_radius degree_spondylolisthesis    class
## 1      98.67292                -0.254400 Abnormal
## 2     114.40543                 4.564259 Abnormal
## 5     108.16872                 7.918501 Abnormal
## 6     130.32787                 2.230652 Abnormal
## 7     120.56752                 5.988551 Abnormal
## 8     117.27007               -10.675871 Abnormal
```

Let's also have a look at the summary of our dataset.

```
summary(train)
```

```
##  pelvic_incidence pelvic_tilt.numeric lumbar_lordosis_angle  sacral_slope
##  Min.   : 26.15   Min.   :-6.555      Min.   : 14.00        Min.   : 13.37
##  1st Qu.: 45.67   1st Qu.:10.758      1st Qu.: 36.67        1st Qu.: 33.22
##  Median : 56.83   Median :16.253      Median : 47.94        Median : 42.06
##  Mean   : 59.75   Mean   :17.296      Mean   : 51.00        Mean   : 42.46
##  3rd Qu.: 71.45   3rd Qu.:21.823      3rd Qu.: 62.65        3rd Qu.: 51.38
##  Max.   :129.83   Max.   :49.432      Max.   :100.74        Max.   :121.43
##  pelvic_radius    degree_spondylolisthesis      class
##  Min.   : 70.08   Min.   :-11.058          Abnormal:157
##  1st Qu.:110.94   1st Qu.:  1.358          Normal  : 75
##  Median :117.48   Median :  9.210
##  Mean   :117.35   Mean   : 24.341
##  3rd Qu.:125.05   3rd Qu.: 38.330
##  Max.   :163.07   Max.   :418.543
```

This dataset is taken from the ML optimized library; thus, theoretically, no data cleaning is required (we can also indirectly see this by looking at the summary above although some questions arise as explained later in this section).

Before we visualize our data, we need to understand what each of these categories means. The class column is what we try to predict. It can take two values: Normal and Abnormal (meaning that a person has either Hernia or Spondylolisthesis). All other columns in our dataset are predictors:

1. Pelvic incidence - pelvisacral angle, defined as the angle between a line perpendicular to the sacral plate at its midpoint and a line connecting the same point to the center of the bicoxofemoral axis (3). This might be challenging to understand for a non-specialist. There are additional information and visualization available on the OrthoBullets website (Imaging section) (4), which might be helpful. Further exploration of the topic shows that pelvic incidence can be calculated as pelvic tilt plus sacral slope. Given that these two items are also in our dataset as predictors, the inclusion of pelvic incidence into our dataset for predictions might distort the results. Thus we are going to exclude it from our set of predictors to keep it straightforward (in fact, such action doesn't materially change results as it was separately tested as part of the report preparation).

2. Pelvic tilt - an angle measured by a vertical reference line from the center of the femoral head and a line from the center of the femoral head to the midpoint of the sacral end plate (3). Again, this

description might be not very informative for non-specialists. There is a short but informative explanation on Wikipedia (as of the moment of this report writing) (5). There are additional information and visualization available on the OrthoBullets website (Imaging section) (4).

3. Lumbar lordosis angle - the angle between the upper plate of the first lumbar and first sacral vertebral bodies (6). Discussion of the topic on the verywellhealth website (7) might be useful for non-specialists.

4. Sacral slope - an angle subtended by a line parallel to the sacral end plate and a horizontal reference line (3). There are additional information and visualization available on the OrthoBullets website (Imaging section) (4).

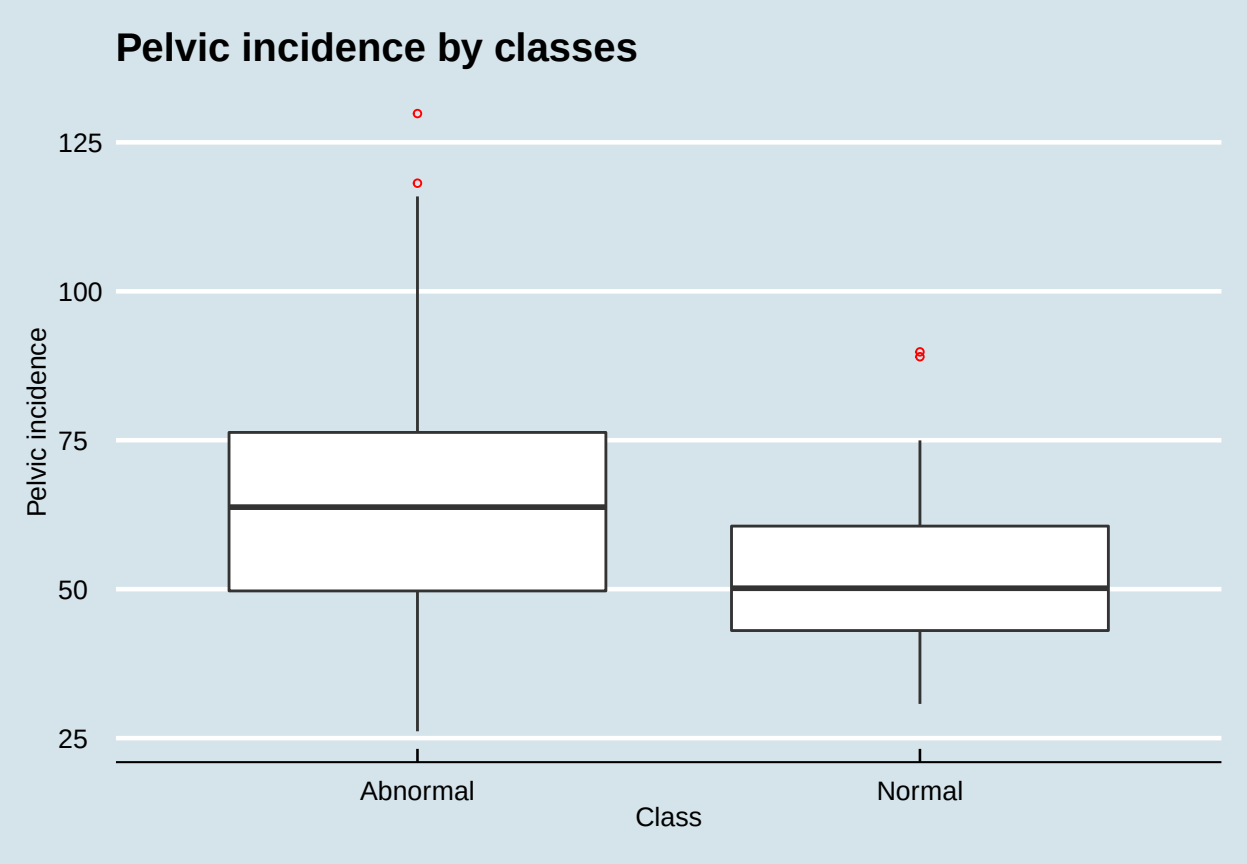
5. Pelvic radius - the distance of the line joining the hip axis and the posterior corner of the S1 endplate. Some clarifications and visualizations are available in "Spino-pelvic radiological parameters in normal Indian population" report (8).

6. Grade of spondylolisthesis. Spondylolisthesis is the displacement of one spinal vertebra compared to another (9). It is not clear what do Grade of spondylolisthesis refers to in this dataset as numbers in the dataset don't match with most of the descriptions on other resources. Moreover, if we have a look at the summary of our dataset (see above) and explore the numbers in this column, we can figure out that there is one obvious outlier. Some people have asked questions with regards to this column on Kaggle; however, no answers were received. We are not going to take out the outlier since we assume that the dataset is correct (especially given that it is part of the ML optimized collection). Also, I have tested the performance of the models without this outlier, and the accuracy of the used models hasn't changed materially (except for the KNN model).

Now, let's visualize our dataset see how predictors are distributed within classes.

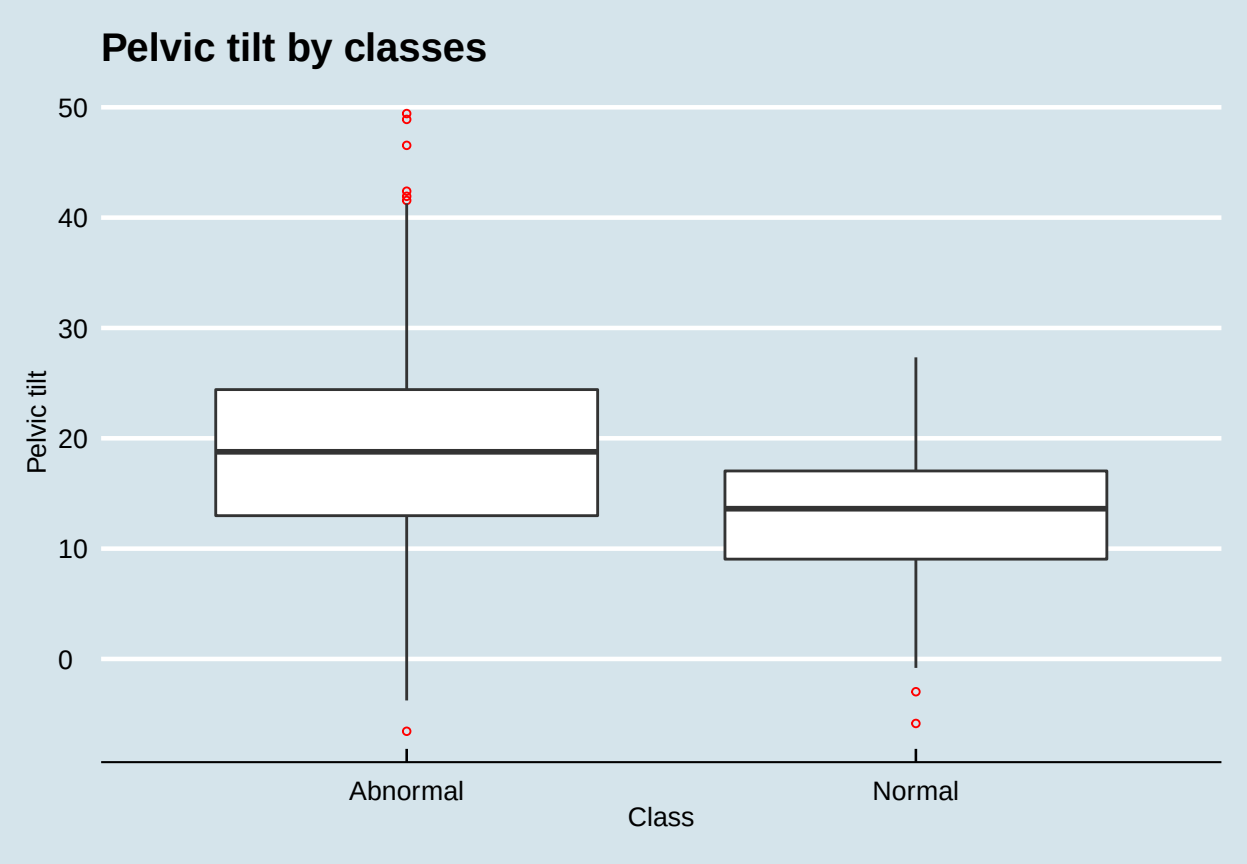
We'll start with Pelvic incidence.

```
train %>%
  ggplot(aes(x=class, y=train[,1])) +
  geom_boxplot(outlier.colour="red",
               outlier.shape=1,
               outlier.size=1) +
  xlab('Class') +
  ylab('Pelvic incidence') +
  ggtitle('Pelvic incidence by classes') +
  theme_economist()
```



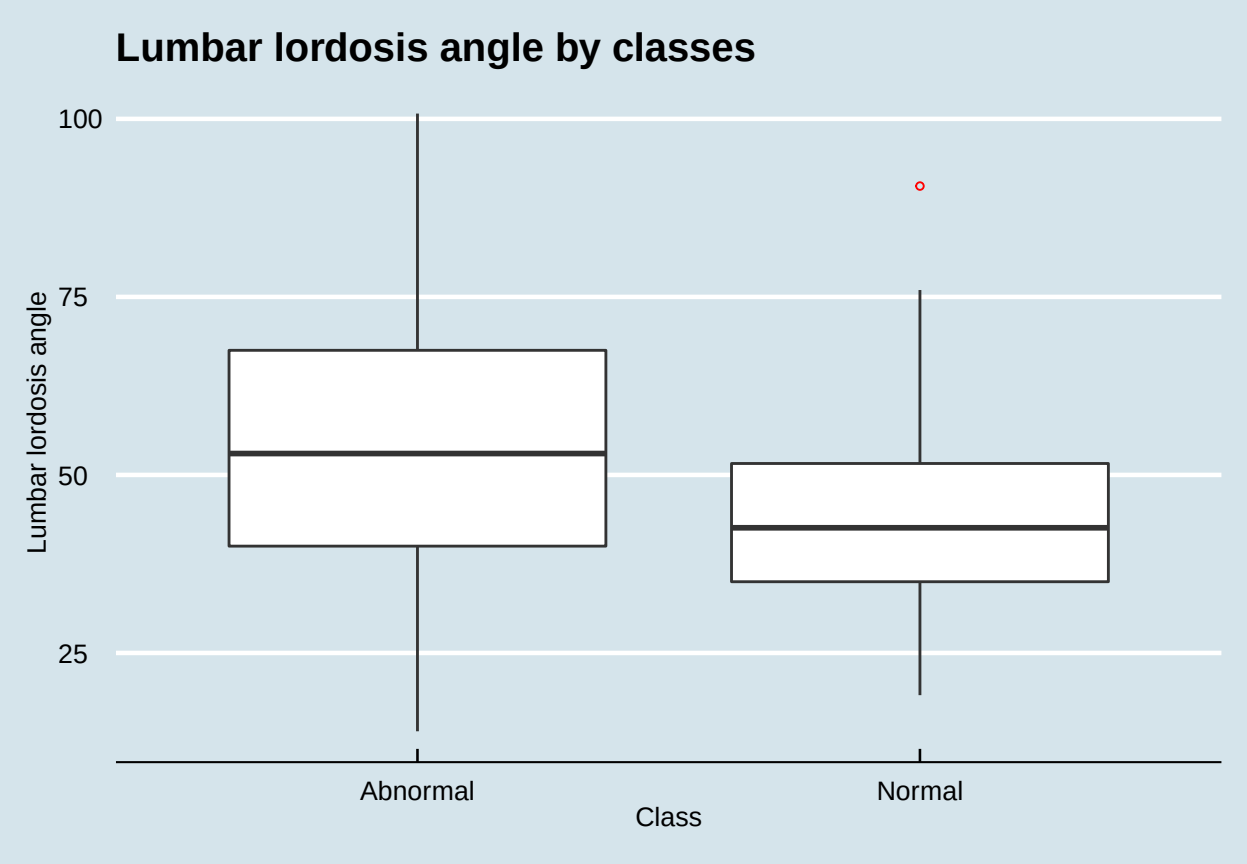
The next metric to explore is Pelvic tilt.

```
train %>%
  ggplot(aes(x=class, y=train[,2])) +
  geom_boxplot(outlier.colour="red",
               outlier.shape=1,
               outlier.size=1) +
  xlab('Class') +
  ylab('Pelvic tilt') +
  ggtitle('Pelvic tilt by classes') +
  theme_economist()
```



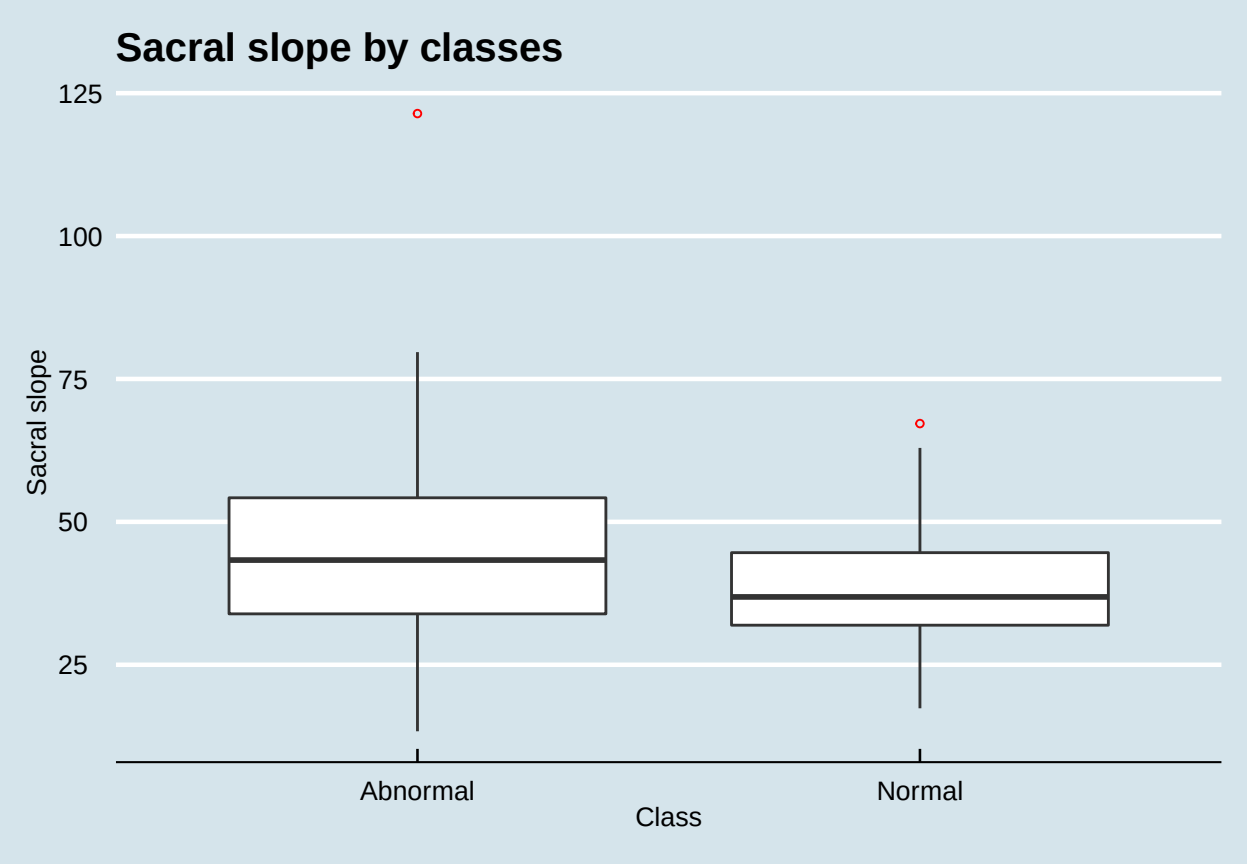
The next metric to explore is Lumbar lordosis angle.

```
train %>%
  ggplot(aes(x=class, y=train[,3])) +
  geom_boxplot(outlier.colour="red",
               outlier.shape=1,
               outlier.size=1) +
  xlab('Class') +
  ylab('Lumbar lordosis angle') +
  ggtitle('Lumbar lordosis angle by classes') +
  theme_economist()
```



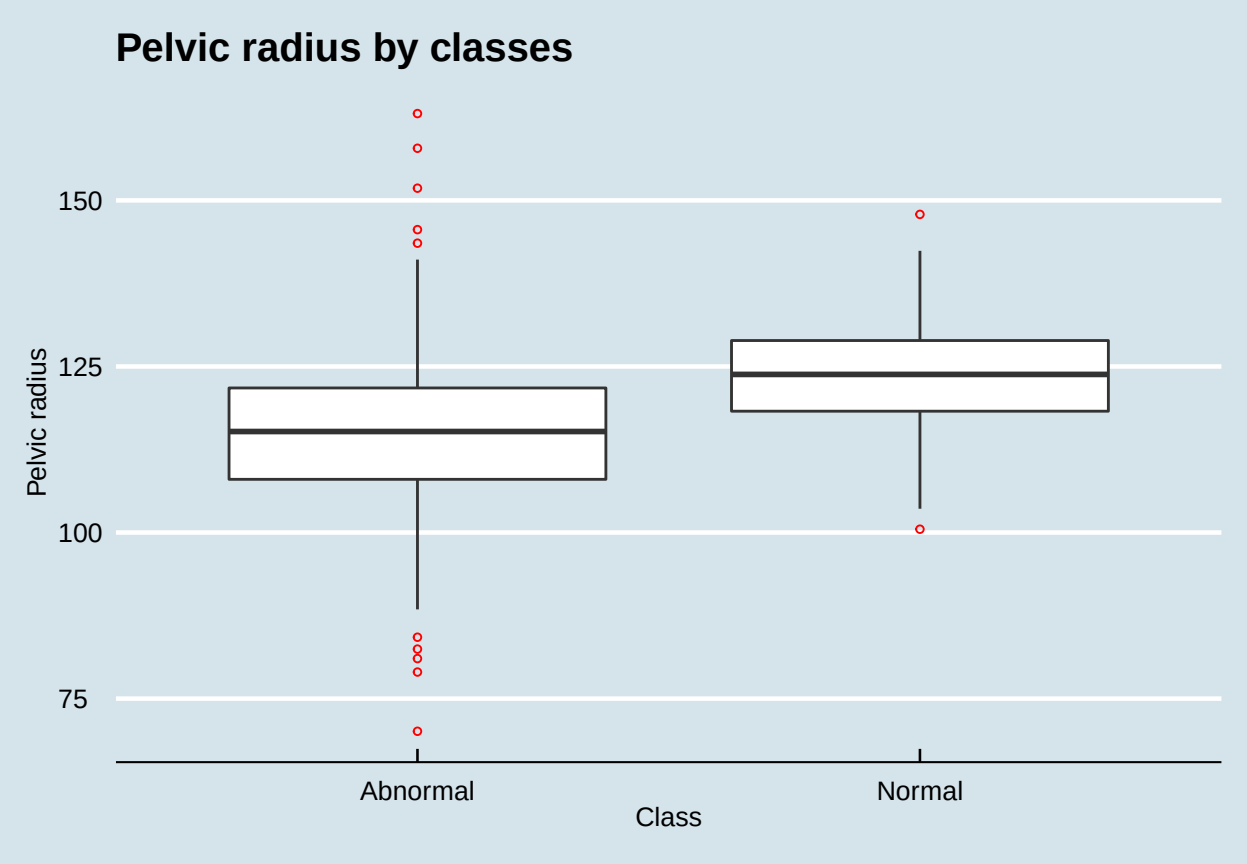
The next metric to explore is Sacral slope.

```
train %>%
  ggplot(aes(x=class, y=train[,4])) +
  geom_boxplot(outlier.colour="red",
               outlier.shape=1,
               outlier.size=1) +
  xlab('Class') +
  ylab('Sacral slope') +
  ggtitle('Sacral slope by classes') +
  theme_economist()
```



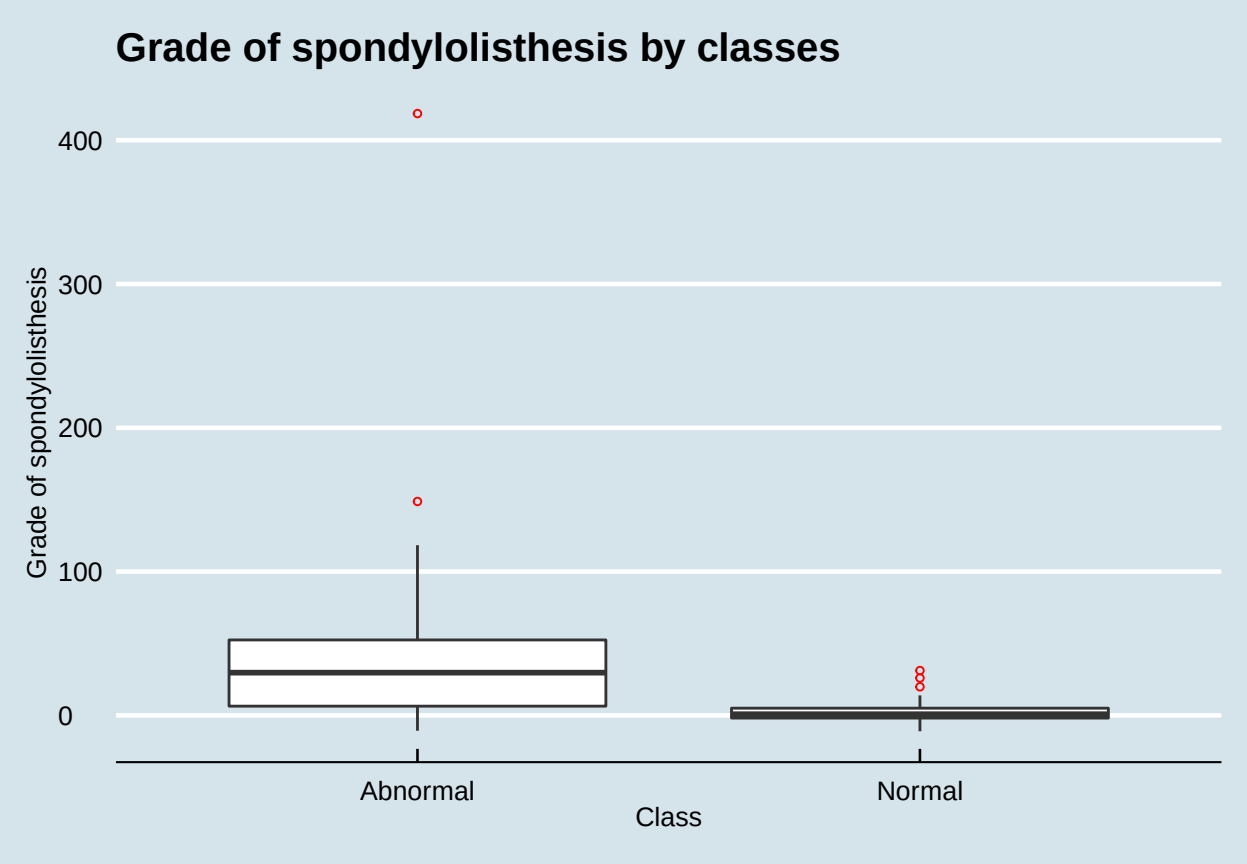
The next metric to explore is Pelvic radius.

```
train %>%
  ggplot(aes(x=class, y=train[,5])) +
  geom_boxplot(outlier.colour="red",
               outlier.shape=1,
               outlier.size=1) +
  xlab('Class') +
  ylab('Pelvic radius') +
  ggtitle('Pelvic radius by classes') +
  theme_economist()
```



And finally, the last metric to explore - Grade of spondylolisthesis.

```
train %>%
  ggplot(aes(x=class, y=train[,6])) +
  geom_boxplot(outlier.colour="red",
               outlier.shape=1,
               outlier.size=1) +
  xlab('Class') +
  ylab('Grade of spondylolisthesis') +
  ggtitle('Grade of spondylolisthesis by classes') +
  theme_economist()
```



Based on these charts, it is clear that the last two predictors (Pelvic radius and Grade of spondylolisthesis) should have the most impact on classes. Also, it is visible that there is an outlier in Grade of spondylolisthesis, however, as mentioned above, we are not going to make adjustments (and impact of this outlier on final models is not very significant).

Let's also build a correlation table for our predictors:

```
# Creating table
Cor_Tab <- cor(train[,1:6])

# Changing format of the table to make it look better and display the table
colnames(Cor_Tab) = c ("PI", "PT", "LLA", "SS", "PR", "DS")
rownames(Cor_Tab) = c ("PI", "PT", "LLA", "SS", "PR", "DS")
formatC(Cor_Tab, digits = 2, format = 'f', big.mark = ',') %>% knitr::kable()
```

	PI	PT	LLA	SS	PR	DS
PI	1.00	0.60	0.72	0.83	-0.30	0.63
PT	0.60	1.00	0.37	0.06	-0.00	0.30
LLA	0.72	0.37	1.00	0.64	-0.16	0.48
SS	0.83	0.06	0.64	1.00	-0.37	0.57
PR	-0.30	-0.00	-0.16	-0.37	1.00	-0.11
DS	0.63	0.30	0.48	0.57	-0.11	1.00

From the table, we can see that some numbers seem to be highly correlated. We already know that Pelvic tilt and Sacral Slope are related to Pelvic incidence. The lumbar lordosis angle is also highly correlated with

the metrics mentioned above.

By now, we have a good understanding of the dataset. Let's start building ML algorithms.

2.2 Defining arguments

Before we start, we have to define the parameters that are going to be passed into our functions (x and y for both train and test subsets).

```
# define key parameters for train and test sets

Tr_Pre <- as.matrix(train[,2:6]) #Train_Predictors
Tr_Cl <- train$class #Train_Class
Te_Pre <- as.matrix(test[,2:6]) #Test_Predictors
Te_Cl <- test$class #Test_Class
```

2.3 Generalized Linear Model (GLM)

Linear regression is one of the most commonly used ML algorithms. However, this approach cannot be applied to classification problems. Instead, an approach called logistic regression is used. This approach uses the logistic function to calculate probability as determined by the following formula for independent variables X.

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}$$

The formula results in values between 0 and 1, which represent two outcomes (logistic regression works with a binary dependent variable (Y) if used without extensions). Depending on the value of $p(X)$, the function assigns inputs to one of the classes (in our case: Normal and Abnormal).

Below is the code which trains the model (using the train subset) and makes relevant predictions for the test subset. Please note that since glm doesn't have any tuning parameters in caret, no additional arguments are passed into the train function.

```
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

train_glm <- train(x = Tr_Pre, y = Tr_Cl, method = "glm")

y_hat_glm <- predict(train_glm, Te_Pre)

m1acc <- confusionMatrix(y_hat_glm, Te_Cl)$overall['Accuracy']

cat('Model accuracy is equal to ',
    formatC(m1acc, digits = 5, format = 'f', big.mark = ','), '. ',
    'Below is the confusion matrix:', sep = "")
```

```
## Model accuracy is equal to 0.91026. Below is the confusion matrix:
```

```
confusionMatrix(y_hat_glm, Te_Cl)$table
```

```
##           Reference
## Prediction Abnormal Normal
##   Abnormal       49      3
##   Normal          4     22
```

Let's move to other algorithms to see if we can improve accuracy.

2.4 k-Nearest Neighbors (KNN)

KNN is a simple and widely used ML algorithm. The idea behind the algorithm is similar to the famous proverb: "Tell me who your friends are, and I'll tell you who you are." The algorithm calculates distances between various points, then selects k (which is a tuning parameter) "nearest neighbours" based on the distance. After this, in the case of classification, the algorithm detects the most common label.

KNN is a popular algorithm; however, it should be used with caution for large datasets with many predictors since it might be challenging to run it from a computational standpoint. In our case, we have a relatively small dataset so that we can proceed with calculations.

Below is the relevant code which trains the model using the train subset and makes predictions for the test subset, as well as shows optimal k used in the model. By default, the train function checks k = 5, 7 and 9; however, we are going to tweak tuneGrid argument of the train function to check other k values and determine the most relevant one. For other options of the train function (e.g. trainControl or preProcess), we are going to rely on default values since changes in those values do not materially improve the algorithm in our case (as part of the report preparation, various options were checked).

```
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

train_knn <- train(x = Tr_Pre, y = Tr_Cl, method = "knn",
                   tuneGrid = data.frame(k = seq(5, 60, 1)))

y_hat_knn <- predict(train_knn, Te_Pre)

m2acc <- confusionMatrix(y_hat_knn, Te_Cl)$overall['Accuracy']

cat('Model accuracy is equal to ',
    formatC(m2acc, digits = 5, format = 'f', big.mark = ','), '. ',
    'Below is the confusion matrix:', sep = "")
```

```
## Model accuracy is equal to 0.93590. Below is the confusion matrix:
```

```
confusionMatrix(y_hat_knn, Te_Cl)$table
```

```
##           Reference
## Prediction Abnormal Normal
##   Abnormal       49      1
##   Normal          4     24
```

```
# Shows optimal k from finalModel.
cat('Optimal k is ',
    formatC(train_knn$finalModel$k, digits = 0,
            format = 'f', big.mark = ','),
    sep = "")
```

```
## Optimal k is 21
```

2.5 Random Forest (RF)

In its core, RF is based on the decision tree method which most people encounter regularly. However, instead of building one decision tree, RF method assumes creation of multiple trees (thus "Forest" in its names), which individually are not very relevant but altogether (the algorithm aggregates results of individual trees) perform very well. One of the reasons why the algorithm shows good performance is because it introduces an element of randomness using two methods: 1) bootstrap aggregation (with replacement) and 2) randomly selecting features for each tree (thus reducing correlation between trees). As a result, in a certain way, RF mitigates the issue of overfitting to the training dataset, which is common for decision trees.

Below is the relevant code for RF. We use ranger library which has more tuning parameters than randomForest library. We are going to check a range of values to determine most relevant tuning parameters. Other arguments of the train function are going to be set at default level (since they do not have material impact on the results, checked as part of the course preparation).

```
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

# Setting tuning parameters for the train function
grid <- expand.grid(mtry=seq(1, 5, 1),
                    splitrule = c("extratrees", "gini"),
                    min.node.size = seq(1, 50, 10))

train_rf <- train(x = Tr_Pre, y = Tr_Cl, method = "ranger", tuneGrid = grid)

y_hat_rf <- predict(train_rf, Te_Pre)

m3acc <- confusionMatrix(y_hat_rf, Te_Cl)$overall['Accuracy']

cat('Model accuracy is equal to ',
    formatC(m3acc, digits = 5, format = 'f', big.mark = ','), '. ',
    'Below is the confusion matrix:', sep = "")
```

```
## Model accuracy is equal to 0.93590. Below is the confusion matrix:
```

```
confusionMatrix(y_hat_rf, Te_Cl)$table
```

```
##           Reference
## Prediction Abnormal Normal
##   Abnormal       51      3
##   Normal          2     22
```

2.6 Naive Bayes (NB)

Naive Bayes approach can be explained by breaking it into two parts: "Bayes" and "Naive". It is called Bayes because it represents a probabilistic classifier based on Bayes theorem. The other part of the method is called Naive because it assumes that input variables/predictors are independent. This is not always a very accurate assumption, and in many cases, it results in worse performance; however, this assumption makes calculations much easier to run, especially for large datasets. This makes the approach a pretty popular choice among data scientists.

Below is the code for our dataset. Please note that the only manual adjustment used in this code is the inclusion of the train control function. Since standard manual tuning doesn't improve performance (compared

to automatic selection of parameters), lines of code related to manual tuning are added as comments (to save computational time significantly).

```
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

# Lines of codes for manual tuning:
# grid <- expand.grid(laplace=seq(0, 5, 0.5),
#                     usekernel = c(TRUE,FALSE), adjust=seq(0, 5, 0.5))
# tuneGrid = grid to be added to train function

# Setting train control argument for the train function
control <- trainControl(method = "cv", number = 10, p = .9)

train_nb <- train(x = Tr_Pre, y = Tr_Cl, method = "naive_bayes", trControl = control)

y_hat_nb <- predict(train_nb, Te_Pre)

m4acc <- confusionMatrix(y_hat_nb, Te_Cl)$overall['Accuracy']

cat('Model accuracy is equal to ',
    formatC(m4acc, digits = 5, format = 'f', big.mark = ','), '. ',
    'Below is the confusion matrix:', sep = "")
```

```
## Model accuracy is equal to 0.85897. Below is the confusion matrix:
```

```
confusionMatrix(y_hat_nb, Te_Cl)$table
```

```
##           Reference
## Prediction Abnormal Normal
##   Abnormal       44      2
##   Normal          9     23
```

2.7 Support Vector Machines with Polynomial Kernel (SVM)

The idea behind SVM is to find a hyperlane that separates data points into classes (hyperlane is n-1 dimensional). By its essence, the algorithm performs very well when classes are easily separable, with outliers having less impact than in the case of some other algorithms. Also, SVM is known to perform well in high dimensional spaces. However, the algorithm is very "costly" from a computational standpoint; thus, it requires a lot of time for large datasets.

One of the key characteristics of SVM is that it uses "Kernel trick", which effectively converts/transports linearly inseparable data to higher dimensional spaces where such data can be separated. This allows us to save on computational resources required to build a hyperlane significantly. There are multiple types of kernels. For this exercise, we are going to use one of the most common and widely used kernels - Polynomial Kernel.

Below is the relevant code. Please note that in this code, we do not manually tune parameters (C, degree and scale) or, in any other way, change the default behaviour of the train function. The reason is that such adjustments make the code run much longer without materially improving the accuracy of the results (such adjustments were performed during this report preparation). The code below doesn't include text related to manual adjustments (this code is logically similar to the code added as comments in Naive Bayes section).

```r
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

train_svm <- train(x = Tr_Pre, y = Tr_Cl, method = "svmPoly")

y_hat_svm <- predict(train_svm, Te_Pre)

m5acc <- confusionMatrix(y_hat_svm, Te_Cl)$overall['Accuracy']

cat('Model accuracy is equal to ',
    formatC(m5acc, digits = 5, format = 'f', big.mark = ','), '. ',
    'Below is the confusion matrix:', sep = "")
```

```
## Model accuracy is equal to 0.92308. Below is the confusion matrix:
```

```r
confusionMatrix(y_hat_svm, Te_Cl)$table
```

```
##           Reference
## Prediction Abnormal Normal
##   Abnormal       49      2
##   Normal          4     23
```

3 Results

Let's summarize our results in the following table using simple R code.

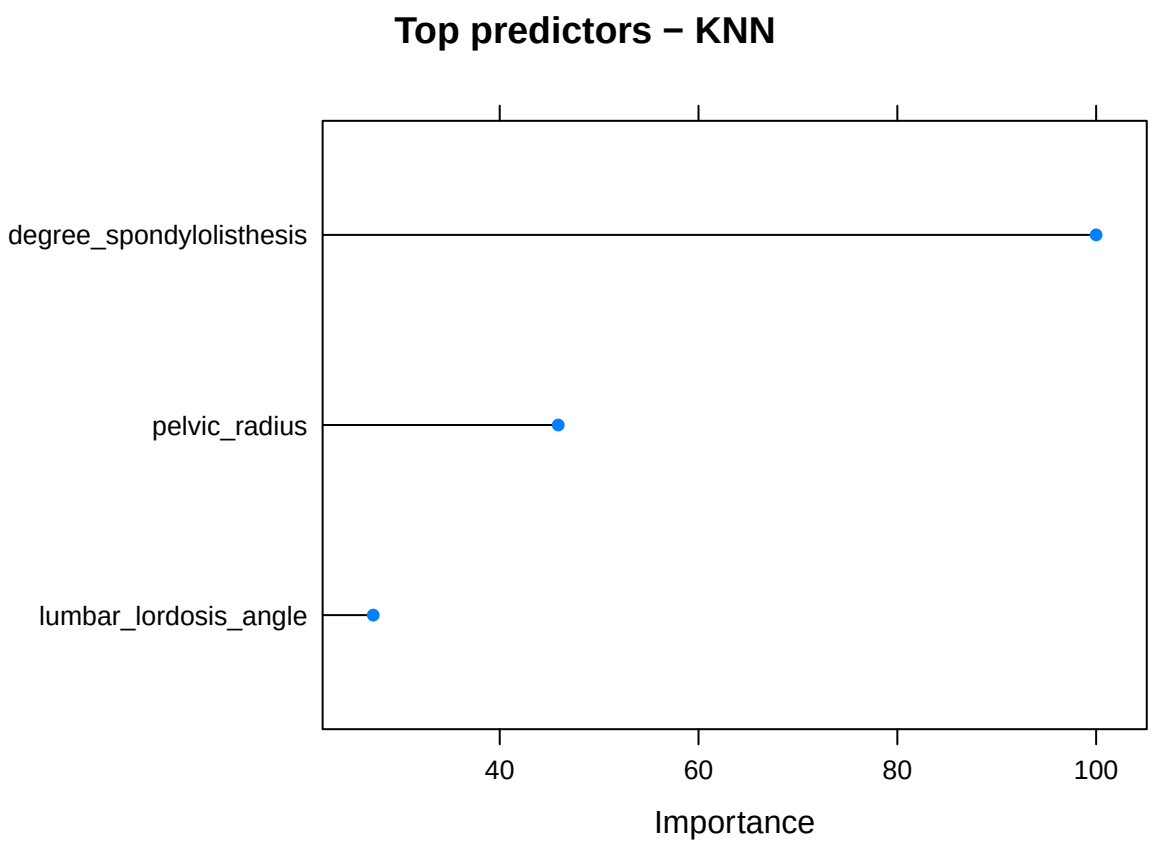
```r
models <- c('1. Generalized Linear Model',
            '2. k-Nearest Neighbors',
            '3. Random Forest',
            '4. Naive Bayes',
            '5. Support Vector Machines with Polynomial Kernel')

acc_values <- formatC(c(m1acc, m2acc, m3acc, m4acc, m5acc), digits = 5, big.mark=",")

acc_table  <- data.frame(Accuracy = acc_values, row.names = models)

acc_table %>% knitr::kable()
```

	Accuracy
1. Generalized Linear Model	0.91026
2. k-Nearest Neighbors	0.9359
3. Random Forest	0.9359
4. Naive Bayes	0.85897
5. Support Vector Machines with Polynomial Kernel	0.92308

As it is clear from the table above, most algorithms have relatively high accuracy. Naive Bayes shows the least impressive performance (partially explained by the assumptions used to run this algorithm, as described in the relevant section above). KNN and RF show the best performance.

Let's also have a look at the top three predictors for KNN (as one of the best performing method) to see if they match our initial assumption that Pelvic radius and Grade of spondylolisthesis would have the highest impact.

```
plot(varImp(train_knn), top=3, main="Top predictors - KNN")
```



Indeed, Pelvic radius and Grade of spondylolisthesis have the highest importance on determining the class.

4 Conclusion

In this report, we have analyzed the Biomechanical features of orthopedic patients dataset available at Kaggle. We reviewed key predictors and then started building ML models using the caret library as a basis.

We used five popular ML methods to train models (using only the train set):

1. Generalized Linear Model
2. k-Nearest Neighbors
3. Random Forest
4. Naive Bayes
5. Support Vector Machines with Polynomial Kernel

Then we tested these models on the test set. All methods show relatively high accuracy, especially considering the small size of the dataset. The best performance was demonstrated by KNN and RF. The worst by Naive Bayes.

The dataset is very "ML friendly"; however, some numbers still need clarifications.

As a way to improve our ML results, we can consider increasing the size of the dataset. In turn, it will allow using other methods, which might further increase accuracy.

5 Sources

Referenced websites:

1. https://www.kaggle.com/uciml/biomechanical-features-of-orthopedic-patients
2. https://www.kaggle.com/annavictoria/ml-friendly-public-datasets?utm_medium=email&utm_source=intercom&utm_campaign=data+projects+onboarding
3. https://www.sciencedirect.com/topics/nursing-and-health-professions/pelvic-incidence
4. https://www.orthobullets.com/spine/2038/adult-isthmic-spondylolisthesis
5. https://en.wikipedia.org/wiki/Pelvic_tilt
6. https://www.researchgate.net/figure/Lumbar-lordosis-and-segmental-angle-measurements-radiographies-on-end-plate-angle_fig2_260397886
7. https://www.verywellhealth.com/lumbar-lordosis-angle-what-is-normal-296978
8. https://pdfs.semanticscholar.org/306d/2c889f7783e1b2944c9c684fc7342c77d206.pdf
9. https://en.wikipedia.org/wiki/Spondylolisthesis

Other sources used:

1. Introduction to Data Science - Data Analysis and Prediction Algorithms with R by Rafael A. Irizarry (https://rafalab.github.io/dsbook/)
2. Caret library (https://topepo.github.io/caret/)
3. towards data science (https://towardsdatascience.com)