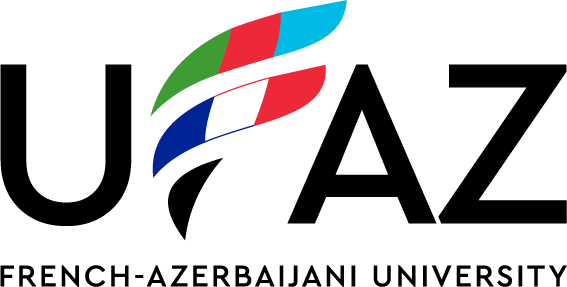
**Student:** Ibrahim Aliyev

**Group:** DSAI-22

**Topic:** R project – Data3

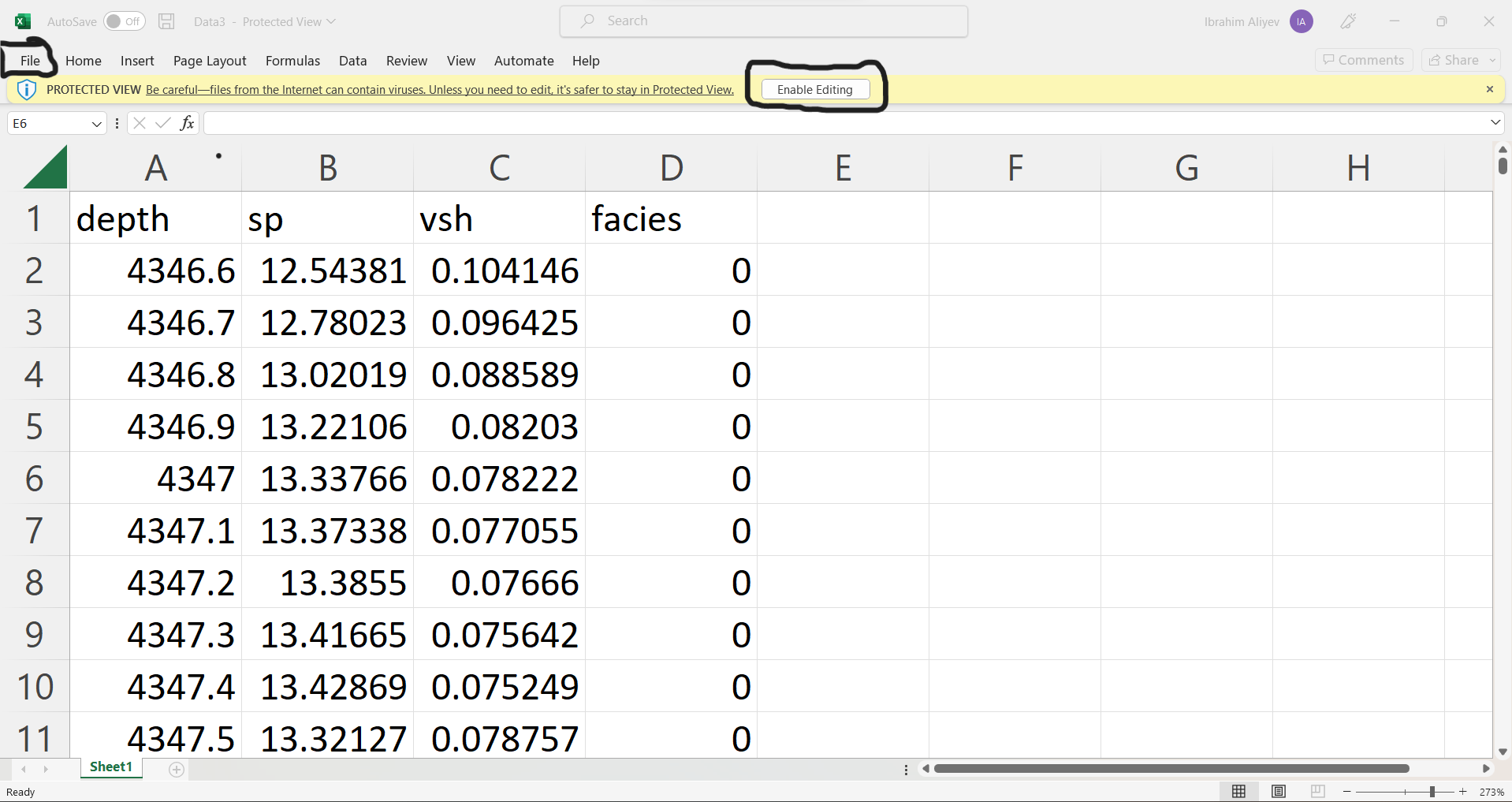
**Description:** This report contains code, results of code, comparison between results and information about details of function. I also added information about some key concepts of data science and other ways to do some important stuffs contained in this project.I did all things alone, myself. But, in the report while referring to me used pronoun ‘we’ instead of ‘I’. Again, it is my own individual work. I used R notebook from R studio to write code and extract it as word document, added some more information and tried to make it more and more readable for readers. Enjoy reading!



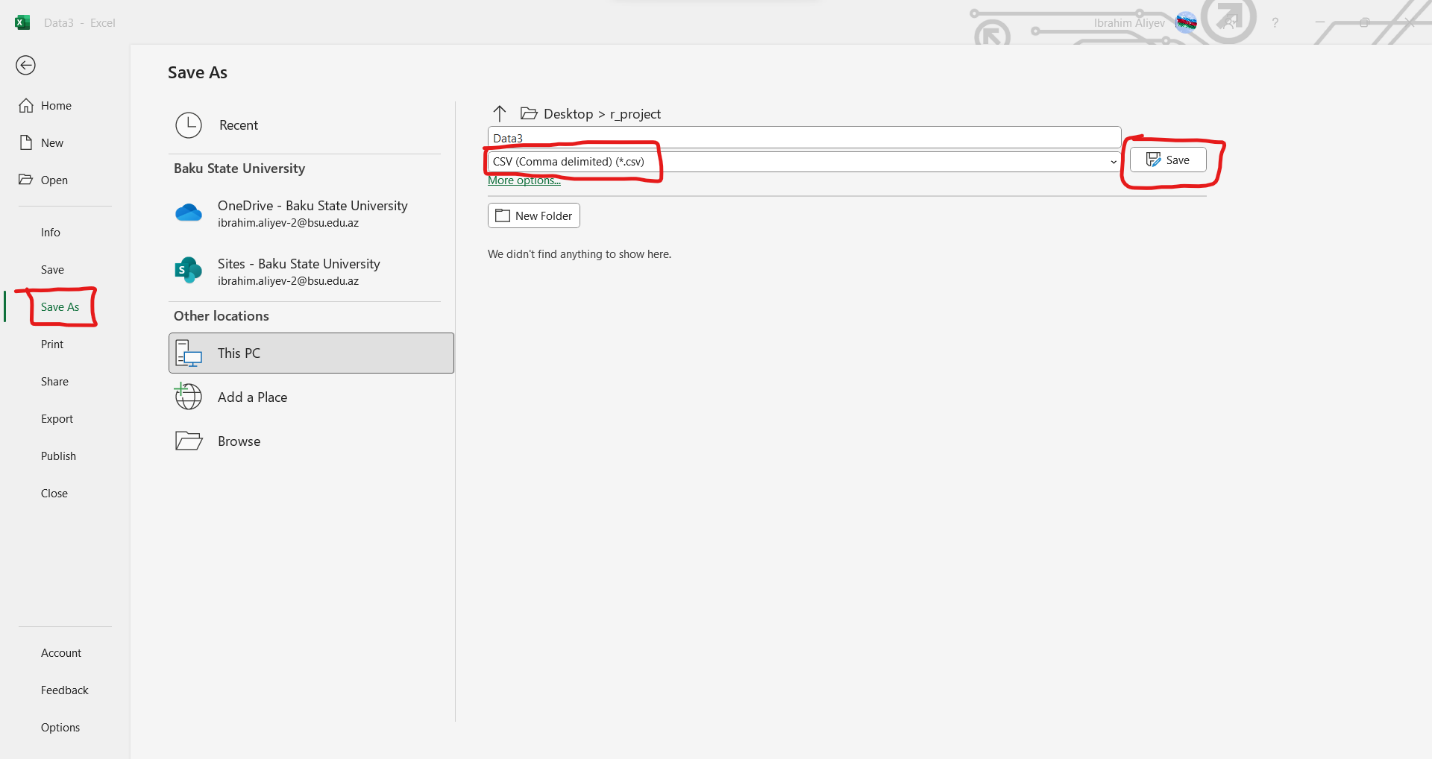
**Introduction**

In this report, we used read\_csv() function to read data and changed data from xlsx to csv using Microsoft Excel like below:

After opening xlsx file in Excel we accept enable editing, and then in the ribbon we choose “File”:

****

And then from the menu, click save as: csv format and clicking yes. But we should specify folder that we want to save into.

****

By using library, we can read xlsx file directly too.

****

I searched about converting directly into csv from xlsx in R itself. Here the results:

****

**Solutions to the tasks:**

***Reading CSV file***

Our first task is to read CSV file. As we changed file format of give xlsx file into csv, and changed working directory into our specified directory, now we can call our file directly in built-in read.csv function. head() function is used to see first 6 row of data.

df<-read.csv("Data3.csv")  
head(df)

## depth sp vsh facies  
## 1 4346.6 12.54381 0.10414585 0  
## 2 4346.7 12.78023 0.09642523 0  
## 3 4346.8 13.02019 0.08858898 0  
## 4 4346.9 13.22106 0.08202959 0  
## 5 4347.0 13.33766 0.07822198 0  
## 6 4347.1 13.37338 0.07705545 0

Task 1 has been completed successfully.

***Column Lithotype***

Let us see how to do task 2. Firstly we need to create new column and bind it to our dataframe. There is easy way like below. We create column with values NA and directly bind it to df.

df['lithotype'] <- NA  
head(df)

## depth sp vsh facies lithotype  
## 1 4346.6 12.54381 0.10414585 0 NA  
## 2 4346.7 12.78023 0.09642523 0 NA  
## 3 4346.8 13.02019 0.08858898 0 NA  
## 4 4346.9 13.22106 0.08202959 0 NA  
## 5 4347.0 13.33766 0.07822198 0 NA  
## 6 4347.1 13.37338 0.07705545 0 NA

Now, as mentioned in the task, we need to specify values for lithotype depending on values of vsh column. My suggestion like below: Firstly, I make all values as ‘S5’, because it is mentioned in section else and we don’t know exact condition for these part. So, let’s do it:

df['lithotype'] <- 'S5'  
head(df)

## depth sp vsh facies lithotype  
## 1 4346.6 12.54381 0.10414585 0 S5  
## 2 4346.7 12.78023 0.09642523 0 S5  
## 3 4346.8 13.02019 0.08858898 0 S5  
## 4 4346.9 13.22106 0.08202959 0 S5  
## 5 4347.0 13.33766 0.07822198 0 S5  
## 6 4347.1 13.37338 0.07705545 0 S5

Then we assign other values of lithotype for conditions given from values of vsh column. All else values that don’t satisfy any of the given conditions still will be equal to S5.

df[df$vsh<0.2, "lithotype"] <- "S1"  
df[df$vsh>0.2 & df$vsh<0.4, "lithotype"] <- "S2"  
df[df$vsh>0.4 & df$vsh<0.6, "lithotype"] <- "S3"  
df[df$vsh>0.6 & df$vsh<0.8, "lithotype"] <- "S4"  
head(df)

## depth sp vsh facies lithotype  
## 1 4346.6 12.54381 0.10414585 0 S1  
## 2 4346.7 12.78023 0.09642523 0 S1  
## 3 4346.8 13.02019 0.08858898 0 S1  
## 4 4346.9 13.22106 0.08202959 0 S1  
## 5 4347.0 13.33766 0.07822198 0 S1  
## 6 4347.1 13.37338 0.07705545 0 S1

Task 2 has been completed successfully.

***Statistics about dataset***

Let us tend to task 3. Dimension of dataframe:

dim(df)

## [1] 3454 5

Numebr of rows and columns:

nrow(df)

## [1] 3454

ncol(df)

## [1] 5

As you can see, dim() function do the job of 2 functions directly, gives number of rows and number of columns.

We can get types of attributes using attributes(df) and as we will do it for each column:

for (column in names(df))  
{  
 print(typeof(df[column]))  
}

## [1] "list"  
## [1] "list"  
## [1] "list"  
## [1] "list"  
## [1] "list"

This operation can be done like below too:

for (column in attributes(df)$names)  
{  
print(typeof(df[column]))  
}

## [1] "list"  
## [1] "list"  
## [1] "list"  
## [1] "list"  
## [1] "list"

Let us print some rows of our dataframe. To do that we specify range of rows in first part of square brackets. Second part is used for colums. Below code will show first 10 rows:

print(df[1:10, ])

## depth sp vsh facies lithotype  
## 1 4346.6 12.54381 0.10414585 0 S1  
## 2 4346.7 12.78023 0.09642523 0 S1  
## 3 4346.8 13.02019 0.08858898 0 S1  
## 4 4346.9 13.22106 0.08202959 0 S1  
## 5 4347.0 13.33766 0.07822198 0 S1  
## 6 4347.1 13.37338 0.07705545 0 S1  
## 7 4347.2 13.38550 0.07665975 0 S1  
## 8 4347.3 13.41665 0.07564233 0 S1  
## 9 4347.4 13.42869 0.07524924 0 S1  
## 10 4347.5 13.32127 0.07875695 0 S1

But there are other built-in functions like head() and tail() to see first 6 and last 6 rows of our data:

head(df)

## depth sp vsh facies lithotype  
## 1 4346.6 12.54381 0.10414585 0 S1  
## 2 4346.7 12.78023 0.09642523 0 S1  
## 3 4346.8 13.02019 0.08858898 0 S1  
## 4 4346.9 13.22106 0.08202959 0 S1  
## 5 4347.0 13.33766 0.07822198 0 S1  
## 6 4347.1 13.37338 0.07705545 0 S1

tail(df)

## depth sp vsh facies lithotype  
## 3449 4691.4 6.981652 0.2857830 2 S2  
## 3450 4691.5 5.973287 0.3187121 2 S2  
## 3451 4691.6 5.073740 0.3480875 2 S2  
## 3452 4691.7 4.374261 0.3709297 2 S2  
## 3453 4691.8 3.846707 0.3881574 2 S2  
## 3454 4691.9 3.409267 0.4024424 2 S3

Let us see levels of each column:

levels(factor(df$lithotype))

## [1] "S1" "S2" "S3" "S4" "S5"

levels(factor(df$facies))

## [1] "0" "1" "2"

Now let us discovery statistics of our dataframe. str stands for structure of dataset:

str(df)

## 'data.frame': 3454 obs. of 5 variables:  
## $ depth : num 4347 4347 4347 4347 4347 ...  
## $ sp : num 12.5 12.8 13 13.2 13.3 ...  
## $ vsh : num 0.1041 0.0964 0.0886 0.082 0.0782 ...  
## $ facies : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ lithotype: chr "S1" "S1" "S1" "S1" ...

Minimum and maximum values can be found using built-in min() and max() functions:

min(df$depth)

## [1] 4346.6

max(df$depth)

## [1] 4691.9

min(df$sp)

## [1] -14.53372

max(df$sp)

## [1] 14.10383

min(df$vsh)

## [1] 0.05320196

max(df$vsh)

## [1] 0.9883867

Let us see mean, standard deviation and variance of column depth:

mean(df$depth)

## [1] 4519.25

sd(df$depth)

## [1] 99.72282

var(df$depth)

## [1] 9944.642

We can easily get first and third quartile, also other quantiles by specifying percent as float number. quantile() function will be used in each case. For example, as first quartile is 25% we will take argument as 0.25, and for third quartile (75%) it will be 0.75:

quantile(df$depth, 0.25) # first quartile for depth column

## 25%   
## 4432.925

quantile(df$depth, 0.75) # third quartile for depth column

## 75%   
## 4605.575

Summary function gives us minimum, maximum values, 1st and 3rd quartile, mean and median directly for each column that contains numerical data:

summary(df)

## depth sp vsh facies lithotype   
## Min. :4347 Min. :-14.534 Min. :0.0532 Min. :0.000 Length:3454   
## 1st Qu.:4433 1st Qu.: -5.153 1st Qu.:0.1797 1st Qu.:0.000 Class :character   
## Median :4519 Median : 6.178 Median :0.3120 Median :2.000 Mode :character   
## Mean :4519 Mean : 3.162 Mean :0.4105 Mean :1.204   
## 3rd Qu.:4606 3rd Qu.: 10.231 3rd Qu.:0.6820 3rd Qu.:2.000   
## Max. :4692 Max. : 14.104 Max. :0.9884 Max. :2.000

Let us find coefficient of variation for depth column. It is found by dividing standard deviation to mean:

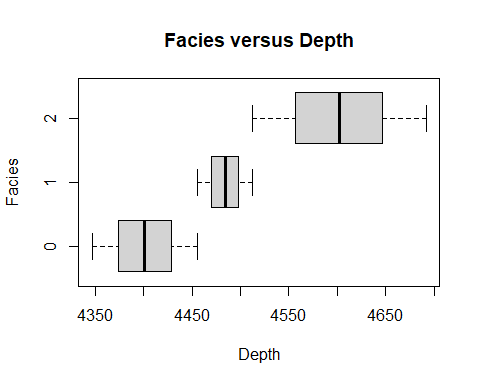
sd(df$depth) / mean(df$depth)

## [1] 0.02206623

**Boxplot operations**

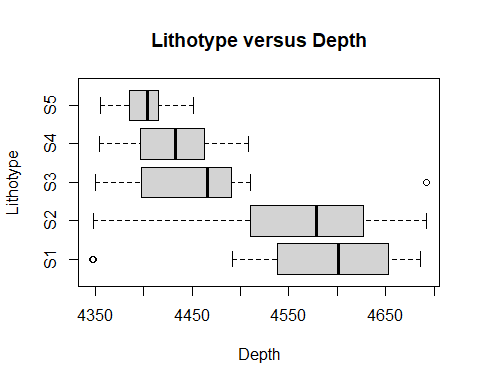
Here we comparing columns Depth and Facies:

boxplot(df$depth ~ df$facies, horizontal = TRUE, xlab = "Depth", ylab = "Facies", main="Facies versus Depth")



Now let us do second boxplot operation (Lithotype versus Depth):

boxplot(df$depth ~ df$lithotype, horizontal = TRUE, xlab = "Depth", ylab = "Lithotype", main="Lithotype versus Depth")



**Support Vector Machine**

Now, about SVM. Abbreviation SVM stands for Support Vector Machines that is supervised learning model and used to analyze data for classification and regression analysis. For the operations in SVM, we will use library caTools for splitting. Let’s install it and check if it is work or not.

install.packages('caTools')

## Installing package into 'C:/Users/Ibrahim Aliyev/AppData/Local/R/win-library/4.2'  
## (as 'lib' is unspecified)

## package 'caTools' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Ibrahim Aliyev\AppData\Local\Temp\RtmpiygpzJ\downloaded\_packages

library(caTools)

## Warning: package 'caTools' was built under R version 4.2.2

head(df)

## depth sp vsh facies lithotype  
## 1 4346.6 12.54381 0.10414585 0 S1  
## 2 4346.7 12.78023 0.09642523 0 S1  
## 3 4346.8 13.02019 0.08858898 0 S1  
## 4 4346.9 13.22106 0.08202959 0 S1  
## 5 4347.0 13.33766 0.07822198 0 S1  
## 6 4347.1 13.37338 0.07705545 0 S1

***SVM for facies column***

Now we should specify our work area. After ignoring sp column and useless other columns we have dataset like below:

svm\_df<-subset(df, select = c(depth, vsh, facies))  
head(svm\_df)

## depth vsh facies  
## 1 4346.6 0.10414585 0  
## 2 4346.7 0.09642523 0  
## 3 4346.8 0.08858898 0  
## 4 4346.9 0.08202959 0  
## 5 4347.0 0.07822198 0  
## 6 4347.1 0.07705545 0

Here, our target feature will be facies column. Let us take it into account like below:

svm\_df$facies<-factor(svm\_df$facies, levels = c(0,1,2))

Now, we are splitting our data into parts like training set and test set:

set.seed(123)  
split <- sample.split(svm\_df$facies, SplitRatio = 0.75) # 75% and 25%  
  
training\_df <- subset(svm\_df, split == TRUE)  
test\_df <- subset(svm\_df, split == FALSE)

Now let us see how our training set and test dataset look like:

head(training\_df) # training dataset

## depth vsh facies  
## 1 4346.6 0.10414585 0  
## 3 4346.8 0.08858898 0  
## 6 4347.1 0.07705545 0  
## 7 4347.2 0.07665975 0  
## 9 4347.4 0.07524924 0  
## 10 4347.5 0.07875695 0

head(test\_df) # test dataset

## depth vsh facies  
## 2 4346.7 0.09642523 0  
## 4 4346.9 0.08202959 0  
## 5 4347.0 0.07822198 0  
## 8 4347.3 0.07564233 0  
## 11 4347.6 0.08861729 0  
## 16 4348.1 0.18627112 0

Let us scale our features to the size that computer can easily understand:

training\_df[-3] <- scale(training\_df[-3])  
test\_df[-3] <- scale(test\_df[-3])

After making this operation, our datasets will be like below:

head(training\_df)

## depth vsh facies  
## 1 -1.731168 -1.141100 0  
## 3 -1.729163 -1.198700 0  
## 6 -1.726156 -1.241404 0  
## 7 -1.725154 -1.242869 0  
## 9 -1.723149 -1.248091 0  
## 10 -1.722147 -1.235104 0

head(test\_df)

## depth vsh facies  
## 2 -1.729691 -1.1428683 0  
## 4 -1.727684 -1.1961669 0  
## 5 -1.726681 -1.2102642 0  
## 8 -1.723670 -1.2198151 0  
## 11 -1.720660 -1.1717765 0  
## 16 -1.715642 -0.8102219 0

Now, we will try to fit SVM to our trained set. But for this operation we need to install package named “e1071”. Let us install it and then continue to fitting.

install.packages('e1071')

## Installing package into 'C:/Users/Ibrahim Aliyev/AppData/Local/R/win-library/4.2'  
## (as 'lib' is unspecified)

## package 'e1071' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Ibrahim Aliyev\AppData\Local\Temp\RtmpiygpzJ\downloaded\_packages

library(e1071)

## Warning: package 'e1071' was built under R version 4.2.2

Tend to fitting:

svm\_classifier <- svm(formula = facies ~ ., data=training\_df, type = 'C-classification',kernel="linear")  
svm\_classifier

##   
## Call:  
## svm(formula = facies ~ ., data = training\_df, type = "C-classification", kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
##   
## Number of Support Vectors: 238

Predicting the test set result:

y\_pred = predict(svm\_classifier, newdata = test\_df[-4])

Let us see confusion matrix like below command:

confusion\_matrix <- table(test\_df[, 3], y\_pred)  
confusion\_matrix

## y\_pred  
## 0 1 2  
## 0 273 0 0  
## 1 1 140 1  
## 2 0 0 449

Okay, so, we showed confusion matrix and last step is to see accuracy. We look for where Facies column is not equal to our prediction and calculating mean after then. Subtracting it from 1 will give us accuracy level.

svm\_accuracy = 1-mean(test\_df$facies != y\_pred)  
svm\_accuracy

## [1] 0.9976852

Done, great! It is closer to 1. Therefore we can say that it is working well.

***SVM for lithotype column***

Now let us do these steps for lithotype column. Oops, we must ignore sp column. Let us see our dataframe again.

head(df)

## depth sp vsh facies lithotype  
## 1 4346.6 12.54381 0.10414585 0 S1  
## 2 4346.7 12.78023 0.09642523 0 S1  
## 3 4346.8 13.02019 0.08858898 0 S1  
## 4 4346.9 13.22106 0.08202959 0 S1  
## 5 4347.0 13.33766 0.07822198 0 S1  
## 6 4347.1 13.37338 0.07705545 0 S1

We have chosen specified columns to continue using subset function:

svm\_df<-subset(df, select = c(depth, vsh, lithotype))  
head(svm\_df)

## depth vsh lithotype  
## 1 4346.6 0.10414585 S1  
## 2 4346.7 0.09642523 S1  
## 3 4346.8 0.08858898 S1  
## 4 4346.9 0.08202959 S1  
## 5 4347.0 0.07822198 S1  
## 6 4347.1 0.07705545 S1

Now, we change our target to lithotype column.

svm\_df$lithotype <-factor(svm\_df$lithotype, levels = c('S1', 'S2', 'S3', 'S4', 'S5'))

Again, we are splitting our data into training set and test set:

set.seed(123)  
split <- sample.split(svm\_df$lithotype, SplitRatio = 0.75) # 75% and 25%  
  
training\_df <- subset(svm\_df, split == TRUE)  
test\_df <- subset(svm\_df, split == FALSE)

Now let us see how our training set and test dataset look like:

head(training\_df) # training dataset

## depth vsh lithotype  
## 1 4346.6 0.10414585 S1  
## 3 4346.8 0.08858898 S1  
## 6 4347.1 0.07705545 S1  
## 7 4347.2 0.07665975 S1  
## 9 4347.4 0.07524924 S1  
## 10 4347.5 0.07875695 S1

head(test\_df) # test dataset

## depth vsh lithotype  
## 2 4346.7 0.09642523 S1  
## 4 4346.9 0.08202959 S1  
## 5 4347.0 0.07822198 S1  
## 8 4347.3 0.07564233 S1  
## 11 4347.6 0.08861729 S1  
## 16 4348.1 0.18627112 S1

Let us scale our features to the size that computer can easily understand:

training\_df[-3] <- scale(training\_df[-3])  
test\_df[-3] <- scale(test\_df[-3])

After making this operation, our datasets will be like below:

head(training\_df)

## depth vsh lithotype  
## 1 -1.724967 -1.135238 S1  
## 3 -1.722964 -1.192818 S1  
## 6 -1.719959 -1.235507 S1  
## 7 -1.718957 -1.236971 S1  
## 9 -1.716953 -1.242192 S1  
## 10 -1.715952 -1.229209 S1

head(test\_df)

## depth vsh lithotype  
## 2 -1.748567 -1.1601657 S1  
## 4 -1.746556 -1.2135070 S1  
## 5 -1.745551 -1.2276157 S1  
## 8 -1.742535 -1.2371743 S1  
## 11 -1.739519 -1.1890971 S1  
## 16 -1.734492 -0.8272523 S1

Now, we will try to fit SVM to our trained set for column lithotype. As we already installed needed package, let us fit directly.

svm\_classifier <- svm(formula = lithotype ~ ., data=training\_df, type = 'C-classification',kernel="linear")  
svm\_classifier

##   
## Call:  
## svm(formula = lithotype ~ ., data = training\_df, type = "C-classification", kernel = "linear")  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: linear   
## cost: 1   
##   
## Number of Support Vectors: 583

Predicting the test set result:

y\_pred = predict(svm\_classifier, newdata = test\_df[-3])

Let us see confusion matrix like below command:

confusion\_matrix <- table(test\_df[, 3], y\_pred)  
confusion\_matrix

## y\_pred  
## S1 S2 S3 S4 S5  
## S1 248 3 0 0 0  
## S2 0 281 1 0 0  
## S3 0 3 74 0 0  
## S4 0 0 0 158 0  
## S5 0 0 0 3 93

And exciting moment! Last one, accuracy! We will see.

svm\_accuracy = 1-mean(test\_df$lithotype != y\_pred)  
svm\_accuracy

## [1] 0.9884259

Great! Again we got value that is closer to 1. So, this means we did awesome job. Let’s tend to KNN!

**K-nearest Neighbors**

KNN stands for K-nearest neighbors and it is one of the supervised learning methods that is used for classification and regression. We need to install class package.

install.packages("class")

## Error in install.packages : Updating loaded packages

library(class)

## Warning: package 'class' was built under R version 4.2.2

***KNN for facies column***

We will use below dataset now:

knn\_df <- subset(df, select = c(depth, vsh, facies))  
head(knn\_df)

## depth vsh facies  
## 1 4346.6 0.10414585 0  
## 2 4346.7 0.09642523 0  
## 3 4346.8 0.08858898 0  
## 4 4346.9 0.08202959 0  
## 5 4347.0 0.07822198 0  
## 6 4347.1 0.07705545 0

Here, our target feature is facies column. Let us take it into account like below:

knn\_df$facies<-factor(knn\_df$facies, levels = c(0,1,2))

Now, we are splitting our data into parts like training set and test set:

set.seed(123)  
split <- sample.split(knn\_df$facies, SplitRatio = 0.75) # 75% and 25%  
  
training\_df <- subset(knn\_df, split == TRUE)  
test\_df <- subset(knn\_df, split == FALSE)

Let us scale our features to the size that computer can easily understand:

training\_df[-3] <- scale(training\_df[-3])  
test\_df[-3] <- scale(test\_df[-3])

After making this operation, our datasets will be like below:

head(training\_df)

## depth vsh facies  
## 1 -1.731168 -1.141100 0  
## 3 -1.729163 -1.198700 0  
## 6 -1.726156 -1.241404 0  
## 7 -1.725154 -1.242869 0  
## 9 -1.723149 -1.248091 0  
## 10 -1.722147 -1.235104 0

head(test\_df)

## depth vsh facies  
## 2 -1.729691 -1.1428683 0  
## 4 -1.727684 -1.1961669 0  
## 5 -1.726681 -1.2102642 0  
## 8 -1.723670 -1.2198151 0  
## 11 -1.720660 -1.1717765 0  
## 16 -1.715642 -0.8102219 0

Let us fit KNN model to fit train dataset, I am going to take k as 3:

knn\_classifier <- knn(train = training\_df[,-3], test = test\_df[,-3], cl = training\_df$facies, k=3)

Let us see confusion matrix like below command:

confusion\_matrix <- table(test\_df[, 3], knn\_classifier)  
confusion\_matrix

## knn\_classifier  
## 0 1 2  
## 0 270 3 0  
## 1 2 139 1  
## 2 0 0 449

Okay, so, we showed confusion matrix and last step is to see accuracy.

knn\_accuracy = 1-mean(test\_df$facies != knn\_classifier)  
knn\_accuracy

## [1] 0.9930556

Again, close to 1. So it is great!

***KNN for lithotype***

Let us specify our dataset:

knn\_df <- subset(df, select = c(depth, vsh, lithotype))  
head(knn\_df)

## depth vsh lithotype  
## 1 4346.6 0.10414585 S1  
## 2 4346.7 0.09642523 S1  
## 3 4346.8 0.08858898 S1  
## 4 4346.9 0.08202959 S1  
## 5 4347.0 0.07822198 S1  
## 6 4347.1 0.07705545 S1

Here, our target feature is lithotype column. Let us take it into account like below:

knn\_df$lithotype<-factor(knn\_df$lithotype, levels = c('S1', 'S2', 'S3', 'S4', 'S5'))

Now, we are splitting our data into parts like training set and test set:

set.seed(123)  
split <- sample.split(knn\_df$lithotype, SplitRatio = 0.75) # 75% and 25%  
  
training\_df <- subset(knn\_df, split == TRUE)  
test\_df <- subset(knn\_df, split == FALSE)

Let us scale our features to the size that computer can easily understand:

training\_df[-3] <- scale(training\_df[-3])  
test\_df[-3] <- scale(test\_df[-3])

After making this operation, our datasets will be like below:

head(training\_df)

## depth vsh lithotype  
## 1 -1.724967 -1.135238 S1  
## 3 -1.722964 -1.192818 S1  
## 6 -1.719959 -1.235507 S1  
## 7 -1.718957 -1.236971 S1  
## 9 -1.716953 -1.242192 S1  
## 10 -1.715952 -1.229209 S1

head(test\_df)

## depth vsh lithotype  
## 2 -1.748567 -1.1601657 S1  
## 4 -1.746556 -1.2135070 S1  
## 5 -1.745551 -1.2276157 S1  
## 8 -1.742535 -1.2371743 S1  
## 11 -1.739519 -1.1890971 S1  
## 16 -1.734492 -0.8272523 S1

Let us fit KNN model to fit train dataset, I am going to take k as 3:

knn\_classifier <- knn(train = training\_df[,-3], test = test\_df[,-3], cl = training\_df$lithotype, k=3)

Let us see confusion matrix like below command:

confusion\_matrix <- table(test\_df[, 3], knn\_classifier)  
confusion\_matrix

## knn\_classifier  
## S1 S2 S3 S4 S5  
## S1 245 6 0 0 0  
## S2 3 276 3 0 0  
## S3 0 0 77 0 0  
## S4 0 0 2 153 3  
## S5 0 0 0 0 96

So, as we showed confusion matrix and last step is to see accuracy.

knn\_accuracy = 1-mean(test\_df$lithotype != knn\_classifier)  
knn\_accuracy

## [1] 0.9803241

Works fine!

**Random Forest**

RF stands for Random Forest and is an ensemble learning method that is used for classification and regression. To work with the random forest we are installing the library named “randomForest”.

install.packages("randomForest")

## Installing package into 'C:/Users/Ibrahim Aliyev/AppData/Local/R/win-library/4.2'  
## (as 'lib' is unspecified)

## package 'randomForest' successfully unpacked and MD5 sums checked  
##   
## The downloaded binary packages are in  
## C:\Users\Ibrahim Aliyev\AppData\Local\Temp\RtmpiygpzJ\downloaded\_packages

library(randomForest)

## Warning: package 'randomForest' was built under R version 4.2.2

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

***RF for facies***

We will use below dataset now:

rf\_df <- subset(df, select = c(depth, vsh, facies))  
head(rf\_df)

## depth vsh facies  
## 1 4346.6 0.10414585 0  
## 2 4346.7 0.09642523 0  
## 3 4346.8 0.08858898 0  
## 4 4346.9 0.08202959 0  
## 5 4347.0 0.07822198 0  
## 6 4347.1 0.07705545 0

Here, our target feature is facies column. Let us take it into account like below:

rf\_df$facies<-factor(rf\_df$facies, levels = c(0,1,2))

Now, we are splitting our data into parts like training set and test set:

set.seed(123)  
split <- sample.split(rf\_df$facies, SplitRatio = 0.75) # 75% and 25%  
  
training\_df <- subset(rf\_df, split == TRUE)  
test\_df <- subset(rf\_df, split == FALSE)

Let us scale our features to the size that computer can easily understand:

training\_df[-3] <- scale(training\_df[-3])  
test\_df[-3] <- scale(test\_df[-3])

After making this operation, our datasets will be like below:

head(training\_df)

## depth vsh facies  
## 1 -1.731168 -1.141100 0  
## 3 -1.729163 -1.198700 0  
## 6 -1.726156 -1.241404 0  
## 7 -1.725154 -1.242869 0  
## 9 -1.723149 -1.248091 0  
## 10 -1.722147 -1.235104 0

head(test\_df)

## depth vsh facies  
## 2 -1.729691 -1.1428683 0  
## 4 -1.727684 -1.1961669 0  
## 5 -1.726681 -1.2102642 0  
## 8 -1.723670 -1.2198151 0  
## 11 -1.720660 -1.1717765 0  
## 16 -1.715642 -0.8102219 0

Let us fit RF model to fit train dataset:

rf\_classifier = randomForest(x = training\_df[,-3], y = training\_df$facies, ntree = 500)  
rf\_classifier

##   
## Call:  
## randomForest(x = training\_df[, -3], y = training\_df$facies, ntree = 500)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 1  
##   
## OOB estimate of error rate: 0.08%  
## Confusion matrix:  
## 0 1 2 class.error  
## 0 818 1 0 0.0012210012  
## 1 0 424 0 0.0000000000  
## 2 0 1 1346 0.0007423905

Let us go to prediction step for test set results:

y\_pred = predict(rf\_classifier, newdata = test\_df[-3])

Let us see confusion matrix like below command:

confusion\_matrix <- table(test\_df[, 3], y\_pred)  
confusion\_matrix

## y\_pred  
## 0 1 2  
## 0 272 1 0  
## 1 0 141 1  
## 2 0 0 449

Okay, so, we showed confusion matrix and last step is to see accuracy.

rf\_accuracy = 1-mean(test\_df$facies != y\_pred)  
rf\_accuracy

## [1] 0.9976852

Great result! Now, let us do same steps for lithotype column

***RF for lithotype***

We will use below dataset now:

rf\_df <- subset(df, select = c(depth, vsh, lithotype))  
head(rf\_df)

## depth vsh lithotype  
## 1 4346.6 0.10414585 S1  
## 2 4346.7 0.09642523 S1  
## 3 4346.8 0.08858898 S1  
## 4 4346.9 0.08202959 S1  
## 5 4347.0 0.07822198 S1  
## 6 4347.1 0.07705545 S1

Here, our target feature is lithotype column. Let us take it into account like below:

rf\_df$lithotype<-factor(rf\_df$lithotype, levels = c('S1', 'S2', 'S3', 'S4', 'S5'))

Now, we are splitting our data into parts like training set and test set:

set.seed(123)  
split <- sample.split(rf\_df$lithotype, SplitRatio = 0.75) # 75% and 25%  
  
training\_df <- subset(rf\_df, split == TRUE)  
test\_df <- subset(rf\_df, split == FALSE)

Let us scale our features to the size that computer can easily understand:

training\_df[-3] <- scale(training\_df[-3])  
test\_df[-3] <- scale(test\_df[-3])

After making this operation, our datasets will be like below:

head(training\_df)

## depth vsh lithotype  
## 1 -1.724967 -1.135238 S1  
## 3 -1.722964 -1.192818 S1  
## 6 -1.719959 -1.235507 S1  
## 7 -1.718957 -1.236971 S1  
## 9 -1.716953 -1.242192 S1  
## 10 -1.715952 -1.229209 S1

head(test\_df)

## depth vsh lithotype  
## 2 -1.748567 -1.1601657 S1  
## 4 -1.746556 -1.2135070 S1  
## 5 -1.745551 -1.2276157 S1  
## 8 -1.742535 -1.2371743 S1  
## 11 -1.739519 -1.1890971 S1  
## 16 -1.734492 -0.8272523 S1

Let us fit RF model to fit train dataset:

rf\_classifier = randomForest(x = training\_df[,-3], y = training\_df$lithotype, ntree = 500)  
rf\_classifier

##   
## Call:  
## randomForest(x = training\_df[, -3], y = training\_df$lithotype, ntree = 500)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 1  
##   
## OOB estimate of error rate: 0.15%  
## Confusion matrix:  
## S1 S2 S3 S4 S5 class.error  
## S1 753 0 0 0 0 0.000000000  
## S2 0 845 0 0 0 0.000000000  
## S3 0 1 230 0 0 0.004329004  
## S4 0 0 1 470 1 0.004237288  
## S5 0 0 0 1 288 0.003460208

Let us go to prediction step for test set results:

y\_pred = predict(rf\_classifier, newdata = test\_df[-3])

Let us see confusion matrix like below command:

confusion\_matrix <- table(test\_df[, 3], y\_pred)  
confusion\_matrix

## y\_pred  
## S1 S2 S3 S4 S5  
## S1 250 1 0 0 0  
## S2 0 282 0 0 0  
## S3 0 0 77 0 0  
## S4 0 0 0 158 0  
## S5 0 0 0 0 96

Okay, so, we showed confusion matrix and last step is to see accuracy.

rf\_accuracy = 1-mean(test\_df$lithotype != y\_pred)  
rf\_accuracy

## [1] 0.9988426

And that’s it! **We got great results. Our all models work very well.**

**Comparing accuracy**

In the table below, I inserted results of accuracy for SVM, KNN, RF while classifying columns facies and lithotype:

|  |  |  |
| --- | --- | --- |
| **Accuracy Levels** | Facies column | Lithotype column |
| Support Vector Machines | 0.9976852 | 0.9884259 |
| K-nearest neighbors | 0.9930556 | 0.9803241 |
| Random Forest | 0.9976852 | 0.9988426 |

As you can see, we got the nearest results that all are closer to 1 and some are the same or almost the same. For facies column SVM and RF showed best and same result. We specified K as 3 in KNN. For lithotype column, RF showed the best result.