



SAPIENZA
UNIVERSITÀ DI ROMA

Advanced statistics for finance report: *Florida_survey*

Corso di laurea in Financial risk and data analysis

Agata Caso
Matricola 1963134

A.A. 2023-2024

The aim of this project is to perform all the models and methodologies we have acquired during the course on a dataset.

The starting point of the work was the preparation of the dataset. Then, I went through the analysis by following these steps:

1. **Linear regression;**
2. **Lasso regression with cross-validation;**
3. **Forward stepwise regression using cross-validation;**
4. **GAM model and best spline;**
5. **Logistic model, linear discriminant analysis and KNN algorithm;**
6. **Predictions.**

For the preparation of the data, I uploaded the dataset “Florida_survey” which contains 60 rows and 18 columns. The observations are related to the profile of students linked to some specific patterns: *subject, gender, age, high_school_GPA, college_GPA, distance_home, distance_residence, TV, sports, newspapers, AIDS, vegetarian, political_affiliation, political_ideology, religiosity, abortion_legalize, affirmative_action_support, life_after_death*. Once the data was uploaded, I started looking for anomalies or Not Available values in the dataset. There weren’t any NA values but there were some values identified as “u” in the observations which I supposed was for “uncertain” or “unknown”. I solved the problem by looking at the mean of how many yes or no there were in each column and by changing the “u” values proportionally in yes or no.

```
1 getwd()
2 setwd("C:/Users/Agata/Desktop/Statistica")
3 data_DEF = read.csv("Florida_survey.csv")
4 View(data_DEF)
5 data_DEF$life_after_death
6 print(data_DEF$life_after_death)
7 table(data_DEF$life_after_death)
8 table(data_DEF$affirmative_action_support)
9 data_DEF$affirmative_action_support[17]
10 data_DEF$affirmative_action_support[17] <- "y"
11 data_DEF$life_after_death[2:3]
12 data_DEF$life_after_death[2:3] <- "y"
13 data_DEF$life_after_death[6] <- "y"
14 data_DEF$life_after_death[c(9,12,27,29,30)] <- "y"
15 data_DEF$life_after_death[c(34,35,37,40,42,54,55,57)] <- "n"
16 View(data_DEF)
17 table(data_DEF$life_after_death)
18 table(data_DEF$affirmative_action_support)
19 data_DEF = na.omit(data_DEF)
20 data_DEF
21 sum(is.na(data_DEF))
```

```

> table(data_DEF$life_after_death)

  n  y
21 39
> table(data_DEF$affirmative_action_support)

  n  y
16 44
> |
> sum(is.na(data_DEF))
[1] 0
> |

```

1. Take the number of hours devoted to sport as your response variable and run a linear regression.

To start the analysis I performed a linear regression taking as my response variable “*sports*” using the command `lm()` in R studio.

```

> ### exercise 1
> modello_linear_regression = lm(data_DEF$sports ~ ., data = data_DEF)
> summary(modello_linear_regression)

Call:
lm(formula = data_DEF$sports ~ ., data = data_DEF)

Residuals:
    Min       1Q   Median       3Q      Max
-5.0941 -1.6772 -0.0192  1.4364 10.7564

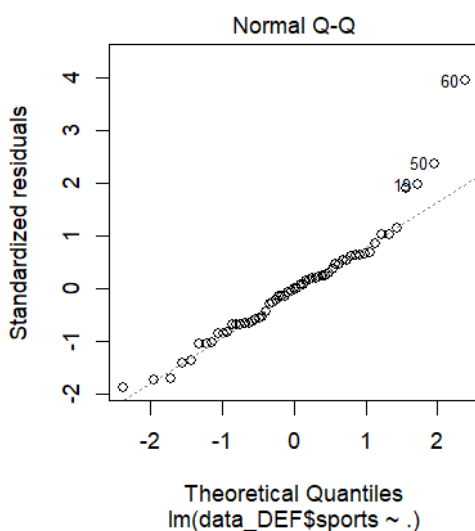
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   5.4915629  10.5162870   0.522  0.60513
subject       -0.0134382   0.0376374  -0.357  0.72340
genderm        1.2457516   1.2448527   1.001  0.32447
age            0.1130627   0.0833129   1.357  0.18425
high_sch_GPA  -1.2564641   1.2992244  -0.967  0.34075
college_GPA   -0.5085964   2.2729314  -0.224  0.82436
distance_home -0.0003568   0.0003736  -0.955  0.34675
distance_residence -0.1487070   0.1560381  -0.953  0.34773
TV             0.1323786   0.0994081   1.332  0.19238
newspapers0_5 -1.3607121   3.9836590  -0.342  0.73490
newspapers1    8.2683368   2.7525824   3.004  0.00514 **
newspapers12   12.4929742   4.6141842   2.708  0.01079 *
newspapers14    8.3551011   3.8456208   2.173  0.03732 *
newspapers2     2.2979138   2.8447074   0.808  0.42518
newspapers3     3.5750096   2.6305999   1.359  0.18365
newspapers4     5.8494296   3.5190491   1.662  0.10624
newspapers5     2.5294599   2.8472639   0.888  0.38096
newspapers6     2.9753158   3.7133660   0.801  0.42890
newspapers7     4.7613324   2.7256970   1.747  0.09026 .
AIDS            0.1884876   0.2611294   0.722  0.47565
vegetariany    -1.2180183   1.6769736  -0.726  0.47292
political_affiliationi 1.0699117   1.5636154   0.684  0.49874
political_affiliationr -1.7962826   2.1747527  -0.826  0.41494
political_ideology  0.4141312   0.5899027   0.702  0.48773
religiosity    -0.1688391   0.8185531  -0.206  0.83789
abortion_legalizey -2.8658540   1.9077809  -1.502  0.14285
affirmative_action_supporty -0.0276969   1.8560430  -0.015  0.98819
life_after_death -0.9774018   1.5008956  -0.651  0.51956

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.518 on 32 degrees of freedom
Multiple R-squared:  0.5495,    Adjusted R-squared:  0.1695
F-statistic: 1.446 on 27 and 32 DF,  p-value: 0.1582
> |

```

After running the code, I could check for the statistical summary of the model so obtained. From this latter we can get the p-values, standard errors for the coefficients, R^2 and the F_statistic. We can check which variables are the best ones to describe how many hours a student dedicates to do sports. To do that we search for the variables having the smallest p-value. The p-value here is associated to the t value that the model performs for each variable. The t-value is a way to quantify the difference between the population means and the p-value is the probability of obtaining a t-value with an absolute value at least as large as the one we actually observed in the sample data if the null hypothesis is actually true. If the p-value is less than a certain value (e.g. 0.05) then we reject the null hypothesis of the test. In our study, we can see that the only variables having a statistically significant p-value are: *Newspapers1*, *Newspapers12* and *Newspapers14*. The overall model has an adjusted $R^2 = 0.1695$ meaning that the model so built doesn't explain well the relationship between sports and all the other variables. Additionally, the p-value associated to the F-statistics is high meaning that the samples are not from a normal distribution with the same variance. When running a linear regression, it's important to analyse the residuals of the model, in particular their normality and homoscedasticity. To analyse the normality of the residuals (meaning that they distribute as a normal random variable) I looked at the Q-Q plot:

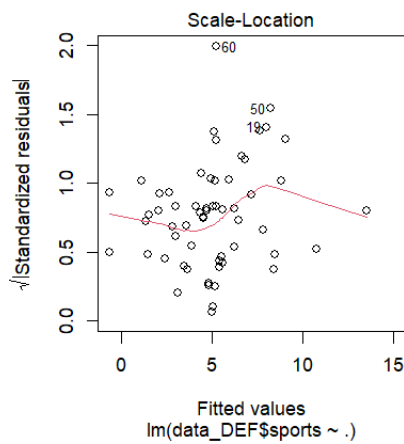


As shown in figure 1, if the points lie along the line, it indicates that the residuals follow a normal distribution. If the points deviate significantly from the line, it suggests deviations from normality. The points in the lower left tail and the upper right tail show some deviation from the line. This suggests that the residuals have heavier tails, meaning extreme values, than a normal distribution.

Figure 1 : Q-Q plot

To analyse the homoscedasticity of the residuals (meaning constant variance) I've used a Scale-Location plot as it's displayed in figure 2.

The red line is used to visualize the trend in the spread of residuals. The Scale-Location plot suggests that there may be some heteroscedasticity in the model, as indicated by the curvature in the



red line. Additionally, there are a few potential outliers. These observations might be affecting the homoscedasticity assumption of the linear model.

Figure 2: residuals vs fitted

From the summary I could also see that the Adjusted R-squared (0.1695) is significantly lower than the Multiple R-squared (0.5495), indicating that adding more predictors doesn't improve the model as much as expected. This can happen when multicollinearity is present. Another symptom of it could be that there are some differences between estimates of the standard errors and the actual standard errors and in some cases the coefficients don't have the expected sign. Multicollinearity occurs when predictor variables in a model are highly correlated with each other, making it difficult to assess the individual contribution of each predictor to the model. To assess multicollinearity I've used the VIF (*variance inflation factor*). A VIF value greater than 5 (or sometimes 10) indicates a problematic level of multicollinearity. The objective of the VIF is to quantify how much the variance (or standard error) of a regression coefficient is inflated due to multicollinearity:

```
> vif(modello_linear_regression)
      GVIF Df GVIF^(1/(2*Df))
subject    2.060162  1    1.435326
gender     1.876522  1    1.369862
age        2.377548  1    1.541930
high_sch_GPA 1.700587  1    1.304066
college_GPA  3.043475  1    1.744556
distance_home 1.882611  1    1.372083
distance_residence 1.968679  1    1.403096
TV          2.126173  1    1.458140
newspapers  88.212741 10    1.251055
AIDS        1.921827  1    1.386300
vegetarian   1.738694  1    1.318595
political_affiliation 7.926218 2    1.677902
political_ideology  4.441645  1    2.107521
religiosity   3.050482  1    1.746563
abortion_legalize 2.995410  1    1.730725
affirmative_action_support 3.266681  1    1.807396
life_after_death 2.485094  1    1.576418
> Vif(modello_linear_regression)
```

To calculate the VIF in this model I've used the Generalized VIF (GVIF) which is used when dealing with categorical variables that have more than two levels. Generally, GVIF values close to 1 indicate that there is very little multicollinearity for that predictor. $GVIF^{1/(2 \cdot Df)}$ is a transformation of the GVIF that allows easier comparison across predictors with different degrees of freedom. A common rule of thumb is that values of $GVIF^{1/(2 \cdot Df)}$ greater than 2 or 3 suggest problematic multicollinearity. From the summary I could state that the variable with the highest

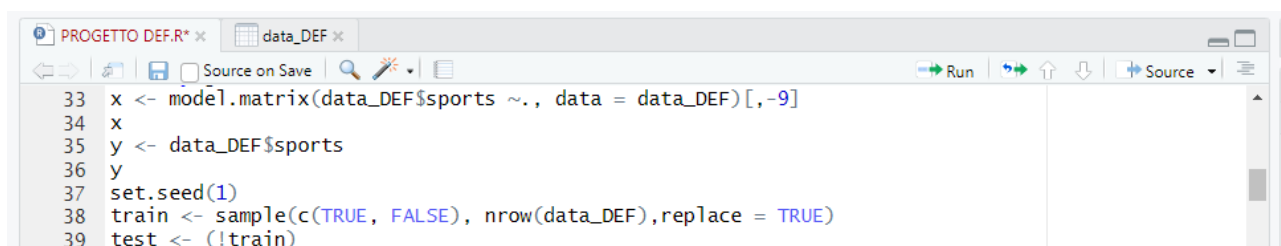
collinearity is newspapers. Also for college_GPA, subject, age, TV, political_affiliation, religiosity, abortion_legalize, affirmative_action_support and life_after_death it was encountered a high GVIF, even though the transformation is still under 2. Only for political_ideology it was stated a high GVIF and also a high transformed GVIF.

We can conclude that there could be a problem of multicollinearity linked with the overall construction of the model.

2. Choose the best subset of variables using lasso with Cross-validation.

The second step of the analysis was to run a Lasso regression through cross-validation. The aim is to find whether the Lasso model can get better results than the linear regression. Lasso regression is a regularization technique that applies a penalty to prevent overfitting and enhance the accuracy of statistical models. Lasso stands for Least Absolute Shrinkage and Selection Operator. It is frequently used in machine learning to handle high dimensional data as it facilitates automatic feature selection with its application. It does this by adding a penalty term to the residual sum of squares (RSS), which is then multiplied by the regularization parameter (lambda or λ). This regularization parameter controls the amount of regularization applied. Larger values of lambda increase the penalty, shrinking more of the coefficients towards zero; this subsequently reduces the importance of (or altogether eliminates) some of the features from the model, resulting in automatic feature selection. Conversely, smaller values of lambda reduce the effect of the penalty, retaining more features within the model. Instead of arbitrarily choosing λ , it would be better to use cross-validation to choose the tuning parameter λ .

To do that, first I have created a matrix containing all the predictors changed in dummy variables by using the `model.matrix()` function and I have isolated our response variable. Then, I have constructed a function to split the data in training and testing ones. This is a crucial step because in this way we can test in the end the models we have built on data that we didn't use to train the models themselves.

A screenshot of the RStudio code editor. The window title is 'PROGETTO DEF.R *'. The code is as follows:

```
33 x <- model.matrix(data_DEF$sports ~., data = data_DEF)[,-9]
34 x
35 y <- data_DEF$sports
36 y
37 set.seed(1)
38 train <- sample(c(TRUE, FALSE), nrow(data_DEF), replace = TRUE)
39 test <- (!train)
```

Once the data's splitting is correctly completed, we can use the `cv.glmnet()` function by setting `alpha=1` to state that we want to run a Lasso regression and use cross-validation to choose lambda.

```
> cv.out <- cv.glmnet(x[train, ], y[train], alpha = 1)
> cv.out

Call:  cv.glmnet(x = x[train, ], y = y[train], alpha = 1)

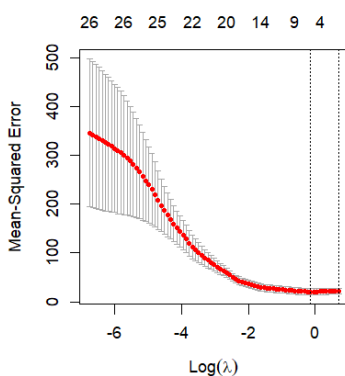
Measure: Mean-Squared Error

      Lambda Index Measure      SE Nonzero
min 0.8693    10   20.06 6.551         4
1se 2.0082     1   21.58 6.266         0
> bestlam <- cv.out$lambda.min
> bestlam
[1] 0.8692854
>
```

By default, the built-in cross-validation function `cv.glmnet()` performs a ten-fold cross-validation. The value of lambda that results in the smallest cross-validation error is 0.8392854.

In particular, the output of the function gives:

- **Measure:** This indicates the performance metric used for cross-validation, which is the mean-squared error (MSE) in this case. The MSE is a measure of the quality of the estimator; it is always non-negative, and values closer to zero are better. This column indicates the mean-squared error for the model corresponding to each lambda value.
- **Lambda:** This is the regularization parameter that controls the amount of shrinkage applied to the coefficients. Smaller values of lambda result in less regularization.
- **Index:** This is the index of the lambda values in the sequence tested during cross-validation.
- **SE:** This is the standard error of the mean-squared error.
- **Nonzero:** This column indicates the number of non-zero coefficients (features) in the model for each lambda value.



We can see from the figure 3 how the logarithm of lambda changes with respect to the mean-squared error associated to it. Each red dot represents the cross-validated MSE for a specific value of λ . The vertical lines (error bars) around each red dot represent the standard error of the MSE at each value of λ . These error bars give a sense of the variability of the MSE estimates. The numbers at the top of the plot indicate the number of non-zero coefficients in the

Figure 3: log-lambda vs mean_squared error

model for the corresponding value of λ . As λ increases, the number of non-zero coefficients generally decreases, leading to a simpler model. The λ corresponding to the minimum MSE provides the best trade-off between bias and variance, leading to the best predictive performance. We can extract the coefficients (the variables) associated with the minimum lambda by using the `coef()` function. From the output we can see that the 4 non-zero predictors are : *genderm*, *newspapers1*, *newspapers12* and *AIDS*.

```
> best_coefs <- coef(cv.out, s = bestlam)
> print(best_coefs)
28 x 1 sparse Matrix of class "dgCMatrix"
               s1
(Intercept)    4.4010047
(Intercept)    .
subject        .
genderm        1.8499047
age            .
high_sch_GPA   .
college_GPA    .
distance_home  .
distance_residence .
newspapers0_5  .
newspapers1    2.9387492
newspapers12   3.0807284
newspapers14   .
newspapers2    .
newspapers3    .
newspapers4    .
newspapers5    .
newspapers6    .
newspapers7    .
AIDS           0.1815075
vegetariany    .
political_affiliationi .
political_affiliationr .
political_ideology .
religiosity    .
abortion_legalizex .
```

Finally, we can state that with respect to the linear regression here we have different significant variables since *genderm*, *AIDS* are added and *newspaper14* is deleted.

3. Run a forward stepwise regression and choose the number of variables through cross-validation.

Stepwise regression is the step-by-step iterative construction of a regression model that involves the selection of independent variables to be used in a final model. In particular, forward stepwise regression begins with no variables in the model, tests each variable as it is added to the model, then keeps those that are deemed most statistically significant, repeating the process until the results are optimal. After we run the model, we have to choose a criteria to select the best model among all of them. To do that, it would be too simplistic to rely on a low RSS or a high R^2 which are related to a

low training error. To check whether the testing error is high or low, one approach is to use k-folds cross-validation to estimate the testing error.

First, I have uploaded all the needed packages: the 'leaps' package for model selection, and the 'margins' and 'dplyr' packages for data manipulation. After that, I modified the dataset converting all the character columns into factors. Then, I went on with the cross-validation set up. I used cross-validation to evaluate model's performance. In particular, 10-fold cross-validation was implemented to do that, meaning that the dataset is divided into 10 subsets (folds). Each fold is used as a validation set once, while the remaining folds are used for training. This process is repeated 10 times. Additionally, I've then created a matrix to store the cross-validation errors for models with up to 26 predictors:

```
> ### exercise 3
> library("leaps")
> library("margins")
> library(dplyr)
> data_DEF <- data_DEF %>% mutate_if(is.character, as.factor)
> k=10
> set.seed(11)
> folds=sample(rep(1:k,length=nrow(data_DEF)))
> folds
[1] 8 4 6 5 6 9 7 2 1 2 10 9 10 2 5 9 3 5 9 1 10 4 6 3 6 8 3 5 2 4 10
[32] 10 6 1 2 9 8 7 4 3 10 3 7 5 2 8 6 4 7 1 8 5 7 3 1 9 1 7 4 8
> table(folds)
folds
 1  2  3  4  5  6  7  8  9 10
 6  6  6  6  6  6  6  6  6  6
> folds <- as.factor(folds)
> cv.errors=matrix(NA,k,26)
> |
```

I defined a custom prediction function for the `regsubsets` object:

```
> predict.regsubsets=function(object,newdata,id,...){
+   form=as.formula(object$call[[2]])
+   mat=model.matrix(form,newdata)
+   coefi=coef(object,id=id)
+   mat[,names(coefi)]%*%coefi
+ }
> |
```

For each fold, I've fitted a forward stepwise regression model and computed the mean squared error (MSE) for each model size:

```

> for(h in 1:k){
+   for(i in 1:26){
+     best.fit=regsubsets(sports ~.,data= data_DEF[folds!=h,],nvmax=26,method="forward")
+     pred=predict.regsubsets(best.fit,data_DEF[folds==h,],id=i)
+     cv.errors[h,i]=mean( (data_DEF$sports[folds==h]-pred)^2)
+   }
+ }

```

The output from the `summary(best.fit)` function provides details about the best models selected during the forward stepwise regression process. From the first part (forced in/forced out) I could see that all variables are neither forced in nor forced out of the model, meaning the selection process could freely include or exclude them. The second part shows the best subset of variables for each model size (from 1 to 26 variables). For each model size, the subset of variables selected by the forward stepwise algorithm is indicated. A "*" symbol denotes that the variable is included in the model for that particular subset size. The results were:

- **Size 1:** No variables are selected.
- **Size 2:** `genderm` is selected.
- **Size 3:** `genderm`, `newspapers1` and `AIDS` are selected.

Each row shows the selected variables for models of increasing size.

As an evaluation metric I've computed the root mean squared error (RMSE) for each model size and identified the model with the minimum RMSE. This choice was led due to the fact that the RMSE is smaller than the MSE and it can be of easier interpretation.

```

> rmse.cv=sqrt(apply(cv.errors,2,mean))
> rmse.cv
[1] 3.995843 3.904575 3.621046 3.894693 3.989726 4.164901 4.250208 4.311725 4.365614 4.395421
[11] 4.622899 4.773000 4.956257 4.981574 5.047731 5.093445 5.184191 5.218442 5.258995 5.279393
[21] 5.265769 5.262817 5.518618 5.502319 5.545914 5.568109
> summary(rmse.cv)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   3.621   4.266   4.969   4.768   5.262   5.568
> which.min(rmse.cv)
[1] 3
>

```

Graphically from figure 4, I could see that while the variables are added in the model there isn't a major improvement in terms of RMSE. In fact, after the 3-variables model the RMSE tends to increase significantly.

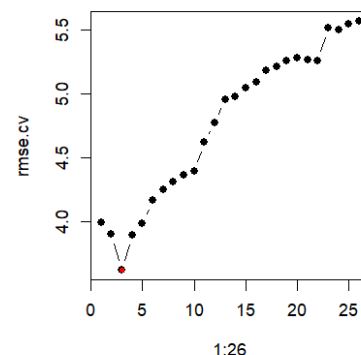


Figure 4: relationship between coefficient and RMSE

The final step is to refit the best model on the entire dataset using the optimal number of predictors identified by cross-validation, therefore the number of coefficients corresponding to the smallest RMSE identified by the `which.min()` function. We see that cross-validation selects a 3-variable model:

```
> reg.best <- regsubsets(sports ~ ., data = data_DEF, nvmax = 26)
> coef(reg.best, 3)
(Intercept)      genderm  newspapers1 newspapers12
      3.513374      2.145815      4.771355      9.340811
> |
```

We can conclude that the model chosen by the forward stepwise selection through cross-validation is a 3-variables model: *genderm*, *newspapers1* and *newspapers12*. There are some differences with respect to the linear regression which had *Newspapers1*, *Newspapers12* and *Newspapers14* as significant variables and with respect to the Lasso regression which had *genderm*, *newspapers1*, *newspapers12* and *AIDS* as non-zero coefficients.

Task 4: Consider the regression of “sports” on “gender” and “high_sch_GPA”. Use a GAM model and choose the best spline for “high_sch_GPA”.

A Generalized Additive Model is a statistical model that extends generalized linear models (GLMs) by allowing for non-linear relationships between the predictor variables and the response variable. GAMs achieve this by combining linear and non-linear functions of the predictors in an additive manner. The model assumes that the effects of the predictor variables can be added together to determine the response variable. All these patterns make the GAM model very suitable for modelling complex, non-linear relationships without specifying the exact form of the non-linearity. Generalized Additive Models (GAMs) and splines are closely related because splines are often used as the smooth functions within GAMs. Specifically, in a GAM, the goal is to model the relationship between predictors and the response variable using smooth functions. Splines are a common choice for these smooth functions due to their flexibility and ability to model non-linear relationships without specifying a particular functional form.

Going to the coding part, I started by installing the packages ‘gam’, ‘splines’ and ‘foreach’.

The first step was to run a GAM regression model by using *sports* as my response variable and *gender* and *high_sch_GPA* as my predictors. To fit more general sorts of GAMs, using smoothing splines, we will need to use the *gam* library in R. The `s()` function, which is part of the *gam* library, is used to indicate that we would like to use a smoothing spline. By using `gam()`, I got:

```
> modello_GAM = gam(data_DEF$sports~s(high_sch_GPA)+ gender,data=data_DEF)
> summary(modello_GAM)

Call: gam(formula = data_DEF$sports ~ s(high_sch_GPA) + gender, data = data_DEF)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-5.6459 -2.2376 -0.6459  1.0271 11.6315

(Dispersion Parameter for gaussian family taken to be 14.1161)

Null Deviance: 878.9833 on 59 degrees of freedom
Residual Deviance: 762.2692 on 54.0001 degrees of freedom
AIC: 336.7898

Number of Local Scoring Iterations: NA

Anova for Parametric Effects
      Df Sum Sq Mean Sq F value Pr(>F)
s(high_sch_GPA) 1  13.96   13.960   0.9889 0.32444
gender          1  54.37   54.368   3.8515 0.05486 .
Residuals      54 762.27   14.116
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova for Nonparametric Effects
      Npar Df  Npar F  Pr(F)
(Intercept)
s(high_sch_GPA)      3 0.90273 0.4459
gender
```

From the summary of the GAM model so built, I could check for the statistically significant variables. GAM models use the ANOVA (Analysis of Variance) table for parametric and nonparametric effects which provides a way to assess the significance of each type of effect included in the model. From the ANOVA for parametric effects, I could see that the variable *gender* has a p-value of 0.05486, which is close to the conventional significance threshold of 0.05, indicating it might have some effect on the response variable *sports*. From the ANOVA for nonparametric effects, I could check that the smooth term '*s(high_sch_GPA)*' is not significant since its p-value is 0.4459.

From figure 5 this plot we can also assess that when the gender is male, the hours dedicated to sports tend to increase. Vice versa for the female gender.

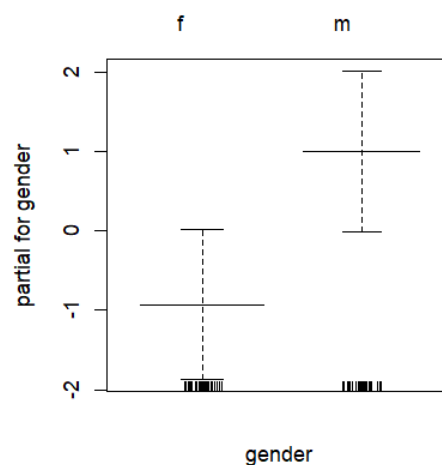


Figure 5: how sports change w.r.t gender

To find the best spline for “*high_sch_GPA*”, I’ve used the `smooth.spline()` function in order to fit it. The output displays the results of fitting a smoothing spline to the data using cross-validation to determine the optimal smoothing parameters:

```
> best_spline = smooth.spline(data_DEF$high_sch_GPA, data_DEF$sports, cv=TRUE)
> best_spline
Call:
smooth.spline(x = data_DEF$high_sch_GPA, y = data_DEF$sports,
              cv = TRUE)

Smoothing Parameter spar= 0.7393504 lambda= 0.1006639 (11 iterations)
Equivalent Degrees of Freedom (Df): 2.718612
Penalized Criterion (RSS): 177.2454
PRESS(l.o.o. CV): 15.43021
> |
```

The elements of the output are:

- **Smoothing Parameter (*spar*):**

- *spar* = 0.7393504: This is the smoothing parameter chosen by the cross-validation process. It controls the trade-off between goodness of fit and smoothness of the fitted spline curve. A larger *spar* leads to a smoother curve but may sacrifice goodness of fit.

- **Lambda (*lambda*):**

- *lambda* = 0.1006639: In the context of smoothing splines, *lambda* represents the penalty parameter applied to the roughness penalty. It helps in controlling the complexity of the spline fit. A larger *lambda* imposes a stronger penalty on deviations from smoothness, which encourages a simpler model.

- **Equivalent Degrees of Freedom (*Df*):**

- *Df* = 2.718612: This is the effective degrees of freedom associated with the smoothing spline. It indicates the effective complexity of the model, considering both the goodness of fit and the penalty applied.

- **Penalized Criterion (RSS):**

- $RSS = 177.2454$: This is the penalized residual sum of squares, which measures the overall fit of the smoothing spline after penalizing for complexity. It accounts for both bias and variance in the model.

- **PRESS (Prediction Residual Sum of Squares):**

- $PRESS(1.0.0. CV) = 15.43021$: This is the prediction residual sum of squares computed during leave-one-out cross-validation (LOO CV). It measures the predictive performance of the model when each observation is left out one at a time.

Once we have the best spline, we can use it to rerun the GAM model and see if there are some changes.

```
> bestspline_model = gam(data_DEF$sports ~ s(high_sch_GPA, sp = best_spline$sp) + gender, data = data_DEF)
> summary(bestspline_model)
```

Call: gam(formula = data_DEF\$sports ~ s(high_sch_GPA, sp = best_spline\$sp) + gender, data = data_DEF)

Deviance Residuals:

Min	1Q	Median	3Q	Max
-6.2235	-2.0858	-0.6019	0.9905	11.9813

(Dispersion Parameter for gaussian family taken to be 13.9714)

Null Deviance: 878.9833 on 59 degrees of freedom
Residual Deviance: 786.3332 on 56.2816 degrees of freedom
AIC: 334.0916

Number of Local Scoring Iterations: NA

Anova for Parametric Effects

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
s(high_sch_GPA, sp = best_spline\$sp)	1.000	13.96	13.960	0.9992	0.32179
gender	1.000	62.01	62.011	4.4384	0.03961 *
Residuals	56.282	786.33	13.971		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Anova for Nonparametric Effects

	Npar	Df	Npar F	Pr(F)
(Intercept)				
s(high_sch_GPA, sp = best_spline\$sp)	0.7	1.4112	0.2264	
gender				

```
> |
```

From the summary I can state that *gender*'s p-value had decreased proving that it's more significant, *high_school_GPA*'s p-value had slightly decreased but it's still not significant.

Task 5: Consider, as your variable of interest, "abortion_legalize" and use the first 40 observations for training:

- A logistic model
- A discriminant analysis
- Knn algorithm with different values of k

- **Logistic model**

Logistic regression is a statistical method used for binary classification problems. It predicts the probability that Y belongs to a particular category of X. Unlike linear regression, which predicts continuous values, logistic regression is used for predicting categorical outcomes typically coded as 0 or 1. The output of the logistic function is a probability between 0 and 1. This probability is used to classify the input into one of the two classes. For example, if the probability is greater than 0.5, the input is classified as class 1; otherwise, it is classified as class 0. The coefficients in logistic regression are typically estimated using Maximum Likelihood Estimation (MLE). MLE finds the parameter values that maximize the likelihood of the observed data. Each coefficient represents the change in the log-odds (a linear combination of the input features) of the outcome for a one-unit change in the corresponding feature. To evaluate the model, we can use different tools:

- **Confusion Matrix:**

- A table showing the true positive, true negative, false positive, and false negative predictions.

- **Accuracy:**

- The proportion of correct predictions (both true positives and true negatives) out of the total predictions.

- **Precision and Recall:**

- Precision: The proportion of true positives out of all positive predictions.
- Recall: The proportion of true positives out of all actual positives.

- **ROC Curve and AUC:**

- ROC (Receiver Operating Characteristic) Curve: A plot of the true positive rate (recall) against the false positive rate.
- AUC (Area Under the Curve): A measure of the model's ability to distinguish between the classes.

Going on with the code, I started by taking as the training dataset the first 40 observations and by taking as my response variable *abortion_legalize* which was transformed in a dummy variable:

```
> ### exercise 5
> ### Logistic model
> x <- model.matrix(data_DEF$abortion_legalize ~., data = data_DEF)[-16]
> x = x[,-1]
> y <- data_DEF$abortion_legalize
> y_train <- y[c(1:40)]
> X_train = x[c(1:40),]
> y_train <- ifelse(y_train == "y",1,0)
> DATA_training <- cbind(X_train, y_train)
> DATA_training <- as.data.frame(DATA_training)
> |
```

Once the training data is correctly divided, I use the `lm()` function setting `family = binomial` to specify that I want to run a logistic regression:

```
> summary(logistic_model)

Call:
glm(formula = DATA_training$y_train ~ ., family = binomial, data = DATA_training)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-8.734e-06  2.110e-08  1.310e-06  3.902e-06  8.691e-06

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  6.635e+01  1.181e+06     0      1
subject      -2.442e-01  1.195e+04     0      1
genderm      -3.668e+01  1.801e+05     0      1
age          -3.570e-01  2.305e+04     0      1
high_sch_GPA -6.445e+00  3.173e+05     0      1
college_GPA   2.215e+01  3.996e+05     0      1
distance_home  1.107e-03  9.720e+01     0      1
distance_residence -7.191e+00  2.888e+04     0      1
TV            -8.278e-01  2.319e+04     0      1
sports        -6.738e-01  3.968e+04     0      1
`newspapers0 5` 6.436e+01  8.452e+05     0      1
newspapers1    2.672e+01  5.249e+05     0      1
newspapers12    2.551e+01  5.386e+05     0      1
newspapers14    7.562e+00  4.651e+05     0      1
newspapers2   -1.563e+01  6.301e+05     0      1
newspapers4    7.239e+00  7.216e+05     0      1
newspapers5    6.015e+01  3.891e+05     0      1
newspapers6   -1.171e+01  6.726e+05     0      1
newspapers7    5.536e+01  2.507e+05     0      1
distance_home  1.107e-03  9.720e+01     0      1
distance_residence -7.191e+00  2.888e+04     0      1
TV            -8.278e-01  2.319e+04     0      1
sports        -6.738e-01  3.968e+04     0      1
`newspapers0 5` 6.436e+01  8.452e+05     0      1
newspapers1    2.672e+01  5.249e+05     0      1
newspapers12    2.551e+01  5.386e+05     0      1
newspapers14    7.562e+00  4.651e+05     0      1
newspapers2   -1.563e+01  6.301e+05     0      1
newspapers4    7.239e+00  7.216e+05     0      1
newspapers5    6.015e+01  3.891e+05     0      1
newspapers6   -1.171e+01  6.726e+05     0      1
newspapers7    5.536e+01  2.507e+05     0      1
AIDS           1.511e+00  5.414e+04     0      1
vegetariany    -1.442e+01  3.555e+05     0      1
political_affiliationi 3.255e+01  3.561e+05     0      1
political_affiliationr -7.704e+00  2.430e+05     0      1
political_ideology  -6.716e+00  1.224e+05     0      1
religiosity     9.104e+00  1.687e+05     0      1
affirmative_action_supporty -1.834e+01  2.465e+05     0      1
life_after_deathy  -3.485e+01  4.816e+05     0      1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3.7098e+01  on 39  degrees of freedom
Residual deviance: 7.4923e-10  on 13  degrees of freedom
AIC: 54

Number of Fisher Scoring iterations: 25

> |
```

From the summary of the model, I could see that all predictors have p-values equal to 1, indicating no significant relationship between any of the predictors and the response variable. This is highly unusual and suggests a problem with the model or the data, such as multicollinearity. The residual deviance is extremely low, indicating a nearly perfect fit to the training data, which again suggests overfitting.

In conclusion, the output suggests severe overfitting and potential multicollinearity among the predictors. The near-zero residuals, extremely high standard errors, and non-significant p-values for all predictors point to issues with the model.

- Linear discriminant analysis

Linear Discriminant Analysis (LDA) is a classification method that finds a linear combination of features that best separates two or more classes of objects or events. It is particularly useful when dealing with normally distributed data and can also be applied as a dimensionality reduction technique. LDA aims to project the data onto a lower-dimensional space with good class-separability. The objective is to maximize the distance between the means of different classes while minimizing the variation within each class. The resulting combination is a linear discriminant function. Essentially, LDA is a powerful method for both classification and dimensionality reduction, particularly when the assumptions of normality and equal covariance matrices hold. It transforms the features in a way that maximizes class separability, making it easier to classify new observations or visualize high-dimensional data.

To run a linear discriminant analysis for *abortion_legalize* the 'MASS' library was needed. Then, I've used the `lda()` function to effectively run it:

```
> LDA_model <- lda(DATA_training$y_train ~., data = DATA_training)
> LDA_model
Call:
lda(DATA_training$y_train ~., data = DATA_training)

Prior probabilities of groups:
 0      1 
0.175 0.825 

Group means:
  subject      genderm      age high_sch_GPA college_GPA distance_home distance_residence      TV
0 18.57143 0.7142857 30.14286 3.185714 3.271429 607.5714 6.071429 5.857143
1 20.90909 0.5151515 29.90909 3.378788 3.427273 1517.7273 3.053030 7.500000
 sports newspapers0 5 newspapers1 newspapers12 newspapers14 newspapers2 newspapers4
0 6.142857 0.00000000 0.14285714 0.00000000 0.00000000 0.28571429 0.00000000
1 4.939394 0.03030303 0.09090909 0.03030303 0.03030303 0.09090909 0.06060606
 newspapers5 newspapers6 newspapers7 AIDS vegetarian political_affiliationi
0 0.0000000 0.00000000 0.0000000 1.857143 0.1428571 0.2857143
1 0.1212121 0.09090909 0.2121212 1.515152 0.2121212 0.4848485
 political_affiliationr political_ideology religiosity affirmative_action_supporty
0 0.5714286 4.428571 1.8571429 0.5714286
1 0.1212121 2.424242 0.9393939 0.7575758
 life_after_death
0 1.0000000
1 0.6666667
```

From the first part of the summary we can list various elements:

- **Prior probabilities:** These values represent the proportion of observations in each class (0 and 1) in the training dataset. Class 0 has 17.5% of the observations, and class 1 has 82.5%.
- **Group means:** It provides the average value of each predictor variable for each class (0 and 1). These means are helpful to see the central tendency of each predictor within each class.

Analysing the second part of the code:

```
Coefficients of linear discriminants:
                                LD1
subject                        -0.0059881739
genderm                       -1.2676075490
age                           0.0267779174
high_sch_GPA                  0.0725233109
college_GPA                   0.7705319378
distance_home                 0.0001347557
distance_residence            -0.2323704268
TV                            -0.0306899974
sports                        -0.0009857512
`newspapers0 5`               4.6469878885
newspapers1                   1.1203666462
newspapers12                  2.3999609879
newspapers14                  1.4347613273
newspapers2                   -0.7627014107
newspapers4                   1.4150802174
newspapers5                   2.0791699115
newspapers6                   1.4859137703
newspapers7                   1.7454975630
AIDS                          -0.1647605301
vegetariany                   0.4703943178
political_affiliationi         0.6722011714
political_affiliationr        -0.8433762343
political_ideology             -0.3657551425
religiosity                   0.2571592358
affirmative_action_supporty   -0.7084065177
life_after_death              -0.8545125697
> |
```

We can check the LDA coefficients. The coefficients of linear discriminants are the weights assigned to each predictor variable to form the linear discriminant function (LD1). This function is used to classify new observations. Negative LDA coefficients suggest that as the predictor increases, it influences the classification towards the other class. Positive LDA coefficients, on the opposite, heavily influence the classification towards a particular class. The variables related to newspaper readership (*newspapers0 5*, *newspapers1*, *newspapers12*, *newspapers14*, *newspapers4*, *newspapers5*, *newspapers6*, and *newspapers7*) have some of the highest positive coefficients, indicating a strong influence on the discriminant score. *College_GPA* also has a relatively high positive coefficient, suggesting that *college GPA* is a significant factor in discriminating between the groups. The coefficient for *genderm* is quite negative, indicating that being male negatively impacts the discriminant score. Both *political_affiliationr* (Republican) and *life_after_death* have negative coefficients, while *political_affiliationi* (Independent) has a positive coefficient, suggesting political leanings and beliefs play a role in group separation.

From figure 6 we can see how the LDA coefficients distribute between the two classes 0 and 1. The histogram for group 0 shows that the data points are concentrated around -2, with a few data points around -3 and 0. There are very few or no data points for values greater than 0.

The histogram for group 1 shows a more spread-out distribution, with data points ranging from approximately -3 to 4. The distribution is more uniform compared to group 0, indicating a more varied set of values for the

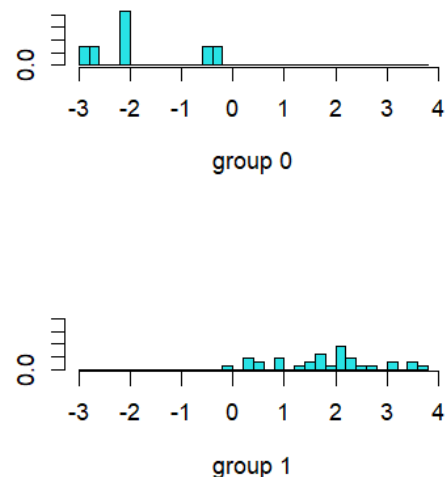


Figure 6: LDA coefficients distribution.

variable within this group. There seems to be some overlap between the two groups around the value of 0, which might indicate that some data points from both groups have similar scores.

- KNN algorithm

K-nearest neighbors (KNN) is a simple, non-parametric, and lazy learning algorithm used for classification and regression. KNN classifies a data point based on how its neighbors are classified. The basic idea is that similar data points tend to be near each other. K is the number of nearest neighbors to consider when making a decision about the classification of a new data point. Common choices for K are small positive integers. For a new data point, we compute the distance between this point and all the points in the training set. Common distance metrics include Euclidean distance (most common), Manhattan distance, or Minkowski distance. The aim is to find the K points in the training data that are closest to the new data point. In this case we want to use the Knn algorithm to pursue classification objectives and predictions. We start from the classification part by running the KNN in R by using the `knn()` function, which is part of `class` library. This function works rather differently from the other model fitting functions because it forms predictions using a single command.

We start by setting `X_test` and `Y_test` equal to the last 20 observations since we took as training data the first 40 ones.

```
> ## Knn algorithm
> X_test = x[c(41:60),]
> y <- ifelse(y == "y",1,0)
> Y_test = y[c(41:60)]
> library(class)
```

Then, I run the code of the KNN:

```
> set.seed(1)
> knn_1 <- knn(X_train, X_test, y_train, k = 1)
> summary(knn_1)
 0  1
 1 19
> knn_3 <- knn(X_train, X_test, y_train, k=3)
> summary(knn_3)
 0  1
 2 18
> knn_5 <- knn(X_train, X_test, y_train, k=5)
> summary(knn_5)
 0  1
 0 20
> knn_7 <- knn(X_train, X_test, y_train, k=7)
> summary(knn_7)
 0  1
 0 20
> |
```

From the summary of the code we can see how the KNN makes the classification by using different values of k. In the next task I will assess its prediction's accuracy.

Task 6: predict the results of the last 20 observations using the above models.

- Logistic model's predictions

In order to make predictions using the logistic model I have built, I've created the testing data by using the `cbind()` function applied to `X_test` and `Y_test` which I've previously created. The function `predict()` uses a trained logistic regression model to predict probabilities on the test data and finally grouping the results in a confusion matrix:

```
> ### Exercise 6
> ### Logistic predictions
> DATA_testing = cbind(X_test, Y_test)
> DATA_testing = as.data.frame(DATA_testing)
> logistic_probs <- predict(logistic_model, DATA_testing, type = "response")
> logistic_pred <- rep("No", 20)
> logistic_pred[logistic_probs > .5] <- "Yes"
> table(logistic_pred, Y_test)
      Y_test
logistic_pred 0  1
No           2  2
Yes          4 12
> |
```

The `logistic_pred` will contain "Yes" for predicted probabilities above 0.5 and "No" otherwise. To compare the results, a confusion matrix was plotted in which each row represents the predicted class (`logistic_pred`) and each column represent the actual class (`Y_test`). The true negative and the false negative both are equal to 2. The true positive are equal to 12 and the false positive are equal to 4.

Other evaluation metrics can be the accuracy, the precision, the sensitivity and the specificity:

```

> ### Accuracy
> 14/20
[1] 0.7
> ### Precision
> 12/16
[1] 0.75
> ### sensitivity
> 12/14
[1] 0.8571429
> ### specificity
> 2/6
[1] 0.3333333
> |

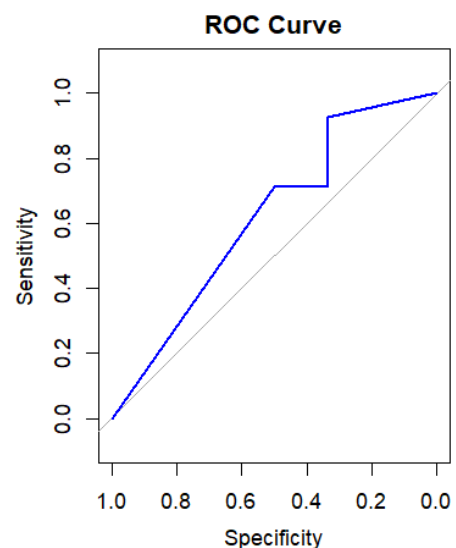
```

From these metrics we can see that specificity (which is the true negative rate) is low, we can check its relationship with sensitivity (which is how many true positives there are out of all actual positives) using the ROC curve and the AUC value. These are powerful tools for evaluating the performance of a binary classification model. The ROC curve is a graphical representation of a classifier's performance across all possible classification thresholds. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The ROC curve helps visualize the trade-off between sensitivity (recall) and specificity (1 - FPR). AUC measures the entire two-dimensional area underneath the ROC curve. It provides a single scalar value to summarize the classifier's performance:

- **AUC = 1:** Perfect classifier. It ranks all positives higher than all negatives.
- **AUC = 0.5:** No discriminative power. Equivalent to random guessing.
- **$0.5 < \text{AUC} < 1$:** The classifier has some discriminative power. The closer to 1, the better the classifier.

In figure 7, at some points, the curve becomes vertical, indicating an increase in sensitivity without an increase in the false positive rate, which is a desirable feature. The curve reaches a sensitivity of 1.0 at a specificity of approximately 0.2, meaning that the model correctly identifies all true positives at this point, but with a false positive rate of 0.8.

Figure 7 ROC curve



The ROC curve shows a reasonable performance of the classifier, as it lies above the diagonal line. But the false positive rate is still very high so we can conclude that the overall predictions performance is not that good.

```
> roc_obj <- roc(Y_test, logistic_probs)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> plot(roc_obj, main = "ROC Curve", col = "blue")
> auc_value <- auc(roc_obj)
> print(paste("AUC:", auc_value))
[1] "AUC: 0.619047619047619"
```

The AUC is 0.6190 which indicates a model that has some ability to distinguish between classes but is only slightly better than random guessing.

- Linear discriminant analysis's predictions

The process of predicting using the LDA model I've plotted before is similar to the one I've used for the logistic model's one.

```
> ### LDA predictions
> LDA_predictions <- predict(LDA_model, DATA_testing)
> names(LDA_predictions)
[1] "class" "posterior" "x"
> lda_class <- LDA_predictions$class
> table(lda_class, Y_test)
      Y_test
lda_class 0  1
      0  2  2
      1  4 12
> |
```

As we can see, the confusion matrix is the same as the one of the logistic and consequently the evaluation metrics, ROC curve and AUC. It is possible for two different models to have confusion matrices that are equal or very similar, especially when applied to the same dataset. The confusion matrix depends on the specific predictions made by each model on the test dataset, and these predictions are influenced by how well each model has learned from the training data and how it generalizes to new, unseen data.

- KNN's predictions

To make predictions using the KNN algorithm already plotted, I just had to extract the confusion matrices since KNN makes predictions in a single command.

```

> ## Knn predictions
> table(knn_1, Y_test)
  Y_test
knn_1  0  1
      0  1  0
      1  5 14
> ### Accuracy
> 15/20
[1] 0.75
> ### Precision
> 14/14+0
[1] 1
> ### Sensitivity
> 14/19
[1] 0.7368421
> ### Specificity
> 1/1+0
[1] 1

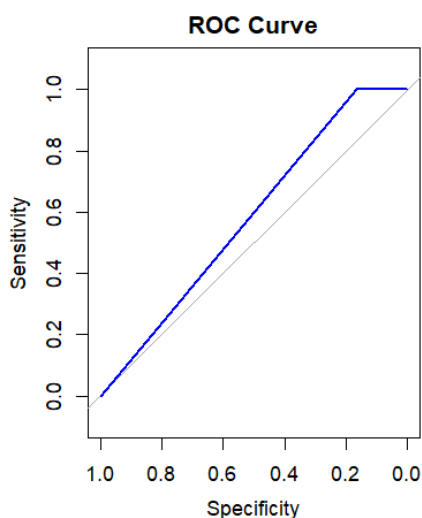
> table(knn_3, Y_test)
  Y_test
knn_3  0  1
      0  0  2
      1  6 12
> ### Accuracy
> 12/20
[1] 0.6
> ### Precision
> 12/14
[1] 0.8571429
> ### Sensitivity
> 12/18
[1] 0.6666667
> ### Specificity
> 0/0+2
[1] NaN

> table(knn_5, Y_test)
  Y_test
knn_5  0  1
      0  0  0
      1  6 14
> ### Accuracy
> 14/20
[1] 0.7
> ### Precision
> 14/14
[1] 1
> ### Sensitivity
> 14/20
[1] 0.7
> ### Specificity
> 0/0
[1] NaN

> table(knn_7, Y_test)
  Y_test
knn_7  0  1
      0  0  0
      1  6 14
> ### Accuracy
> 14/20
[1] 0.7
> ### Precision
> 14/14
[1] 1
> ### Sensitivity
> 14/20
[1] 0.7
> ### Specificity
> 0/0
[1] NaN

```

It's interesting to analyse the differences in the evaluation metrics for each k. For KNN models with k=5 and k=7, we notice that specificity cannot be calculated because there are no true negatives (TN = 0). Specificity requires TN to be non-zero for calculation. In such cases, sensitivity (recall) remains the same as accuracy and precision because there are no true negatives to influence the calculation. So we can conclude that the best k for predictions is 1 since also a KNN model with k=3 has undefined specificity. As a final proof of the performance of the KNN model with k=1, we can plot a ROC curve and the AUC value.



From figure 8 we can see that at the beginning the curve almost lies on the diagonal meaning that we're close to random guessing. At one point as the sensitivity is 1 at a specificity of exactly 0.2 meaning that there is a false positive rate of 0.8.

Figure 8 ROC curve for the KNN with k=1

This trend is confirmed by the AUC, which is:

```

> auc_value <- auc(roc_obj)
> print(paste("AUC:", auc_value))
[1] "AUC: 0.583333333333333"
>

```

This value suggests that the model performs almost as random chance and it's not very well in distinguishing between classes. This means that the model is close to have not a discriminatory

power, it means that there is a 58.33% chance that the model will rank a randomly chosen positive instance higher than a randomly chosen negative instance.