Homework 3

Huanzhang Xia

Table of contents

Link to the Github repository

❗ Due: Thu, Mar 2, 2023 @ 11:59pm

Please read the instructions carefully before submitting your assignment.

1. This assignment requires you to only upload a `PDF` file on Canvas
2. Don't collapse any code cells before submitting.
3. Remember to make sure all your code output is rendered properly before uploading your submission.

Please add your name to the author information in the frontmatter before submitting your assignment

For this assignment, we will be using the Wine Quality dataset from the UCI Machine Learning Repository. The dataset consists of red and white *vinho verde* wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests

We will be using the following libraries:

```r
rm(list=ls())
library(readr)
library(tidyr)
library(dplyr)
```

```

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(purrr)
library(car)
```

```
Loading required package: carData

Attaching package: 'car'

The following object is masked from 'package:purrr':

    some

The following object is masked from 'package:dplyr':

    recode
```

```r
library(glmnet)
```

```
Loading required package: Matrix

Attaching package: 'Matrix'
```

```
The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-6
```

```r
library(corrplot)
```

```
corrplot 0.92 loaded
```

```r
library(janitor)
```

```
Attaching package: 'janitor'

The following objects are masked from 'package:stats':

    chisq.test, fisher.test
```

Appendix

Convenience function for creating a formula object

The following function which takes as input a vector of column names `x` and outputs a `formula` object with `quality` as the response variable and the columns of `x` as the covariates.

```r
make_formula <- function(x){
  as.formula(
    paste("quality ~ ", paste(x, collapse = " + "))
  )
}

# For example the following code will
# result in a formula object
# "quality ~ a + b + c"
make_formula(c("a", "b", "c"))
```

```
quality ~ a + b + c
<environment: 0x7f88f8a58958>
```

Convenience function for `glmnet`

The `make_model_matrix` function below takes a `formula` as input and outputs a **rescaled** model matrix `X` in a format amenable for `glmnet()`

```
make_model_matrix <- function(formula){
  X <- model.matrix(formula, df)[, -1]
  cnames <- colnames(X)
  for(i in 1:ncol(X)){
    if(!cnames[i] == "typewhite"){
      X[, i] <- scale(X[, i])
    } else {
      colnames(X)[i] <- "type"
    }
  }
  return(X)
}
```

Question 1

50 points

Regression with categorical covariate and *t*-Test

1.1 (5 points)

Read the wine quality datasets from the specified URLs and store them in data frames `df1` and `df2`.

```
url1 <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequalit
url2 <- "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequalit
df1 <- read.table(url1,sep=";") # Insert your code here
df1 <- df1 %>%
  row_to_names(row_number = 1)
df2 <- read.table(url2,sep=";") # Insert your code here
df2 <- df2 %>%
  row_to_names(row_number = 1)
```

1.2 (5 points)

Perform the following tasks to prepare the data frame `df` for analysis:

1. Combine the two data frames into a single data frame `df`, adding a new column called `type` to indicate whether each row corresponds to white or red wine.
2. Rename the columns of `df` to replace spaces with underscores
3. Remove the columns `fixed_acidity` and `free_sulfur_dioxide`
4. Convert the `type` column to a factor
5. Remove rows (if any) with missing values.

```
red = "red"
white = "white"
df1$type <- white
df2$type <- red
df <- rbind(df1,df2) # Insert your code here
names(df) <- gsub(" ", "_", names(df))
df <- df %>%
  select(!c(fixed_acidity,free_sulfur_dioxide))
invisible(as.factor(df$type))
df <- df%>%
  drop_na()
dim(df)
```

```
[1] 6497   11
```

Your output to `R dim(df)` should be

```
[1] 6497   11
```

1.3 (20 points)

Recall from STAT 200, the method to compute the t statistic for the the difference in means (with the equal variance assumption)

1. Using `df` compute the mean of `quality` for red and white wine separately, and then store the difference in means as a variable called `diff_mean`.

2. Compute the pooled sample variance and store the value as a variable called `sp_squared`.

3. Using `sp_squared` and `diff_mean`, compute the t Statistic, and store its value in a variable called `t1`.

```r
#1
df$quality <- as.numeric(df$quality)
df13 <- aggregate(x= df$quality,by = list(df$type),FUN = mean)
diff_mean <- df13[2,2]-df13[1,2]
#2
n1 <- nrow(df[df$type=="white",])
n2 <- nrow(df[df$type=="red",])
var1 <- var(df$quality[df$type=="white"])
var2 <- var(df$quality[df$type=="red"])
sp_squared <- ((n1-1)*var1 + (n2-1)*var2) / (n1+n2-2)
#3
t1 <- (diff_mean-0)/(sqrt(sp_squared)*sqrt((1/n1)+(1/n2)))
t1
```

```
[1] 9.68565
```

1.4 (10 points)

Equivalently, R has a function called `t.test()` which enables you to perform a two-sample *t*-Test without having to compute the pooled variance and difference in means.

Perform a two-sample t-test to compare the quality of white and red wines using the `t.test()` function with the setting `var.equal=TRUE`. Store the t-statistic in `t2`.

```r
df14white <- df %>%
  filter(type == "white") %>%
  pull(quality)
df14red <- df %>%
  filter(type == "red") %>%
  pull(quality)
t_test <- t.test(df14white, df14red,var.equal=TRUE)
t2 <- t_test$statistic
```

1.5 (5 points)

Fit a linear regression model to predict `quality` from `type` using the `lm()` function, and extract the *t*-statistic for the `type` coefficient from the model summary. Store this *t*-statistic in `t3`.

```r
fit <- lm(df$quality~df$type)
t3 <- tail(coef(summary(fit))[, "t value"],1)
```

1.6 (5 points)

Print a vector containing the values of `t1`, `t2`, and `t3`. What can you conclude from this? Why?

Calculating t statistic by all mesures should yield same result. Because all three methods are using the same function in the background.

```r
vector<-c(t1, t2, t3) # Insert your code here
print(vector)
```

```
                          t df$typewhite 
    9.68565      9.68565      9.68565 
```

Question 2

25 points

Collinearity

2.1 (5 points)

Fit a linear regression model with all predictors against the response variable `quality`. Use the `broom::tidy()` function to print a summary of the fitted model. What can we conclude from the model summary?

We can conclude that variables volatile_acidity,residual_sugar,chlorides,density,ph,sulphates,alcohol,quality are significant, while the others are not. Total_sulfur_dioxide and citric_acid are not good predictors of the quality, given all other variables are accounted for.

```r
df$volatile_acidity <- as.numeric(df$volatile_acidity)
df$citric_acid <- as.numeric(df$citric_acid)
df$residual_sugar <- as.numeric(df$residual_sugar)
df$chlorides <- as.numeric(df$chlorides)
df$total_sulfur_dioxide <- as.numeric(df$total_sulfur_dioxide)
df$density <- as.numeric(df$density)
df$pH <- as.numeric(df$pH)
df$sulphates <- as.numeric(df$sulphates)
df$alcohol <- as.numeric(df$alcohol)
df$quality <- as.numeric(df$quality)
df$type <- as.factor(df$type)
sapply(df, class)
```

```
    volatile_acidity          citric_acid       residual_sugar
           "numeric"            "numeric"            "numeric"
           chlorides total_sulfur_dioxide              density
           "numeric"            "numeric"            "numeric"
                  pH            sulphates              alcohol
           "numeric"            "numeric"            "numeric"
             quality                 type
           "numeric"             "factor"
```

```r
full_model <- lm(quality~.,df)
broom::tidy(full_model)
```

```
# A tibble: 11 x 5
   term                  estimate std.error statistic  p.value
   <chr>                    <dbl>     <dbl>     <dbl>    <dbl>
 1 (Intercept)          57.5       9.33         6.17  7.44e-10
 2 volatile_acidity     -1.61      0.0806     -20.0   4.07e-86
 3 citric_acid           0.0272    0.0783       0.347 7.28e- 1
 4 residual_sugar        0.0451    0.00416     10.8   3.64e-27
 5 chlorides            -0.964     0.333       -2.90  3.78e- 3
 6 total_sulfur_dioxide -0.000329  0.000262    -1.25  2.10e- 1
 7 density             -55.2       9.32        -5.92  3.34e- 9
 8 pH                    0.188     0.0661       2.85  4.38e- 3
 9 sulphates             0.662     0.0758       8.73  3.21e-18
10 alcohol               0.277     0.0142      19.5   1.87e-82
11 typewhite            -0.386     0.0549      -7.02  2.39e-12
```

2.2 (10 points)

Fit two **simple** linear regression models using `lm()`: one with only `citric_acid` as the predictor, and another with only `total_sulfur_dioxide` as the predictor. In both models, use `quality` as the response variable. How does your model summary compare to the summary from the previous question?

The two variables alone are significant predictors of quality. The previous question concludes that they are not.

```r
model_citric <- lm(quality~citric_acid,df)
summary(model_citric)
```

```

Call:
lm(formula = quality ~ citric_acid, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.9938 -0.7831  0.1552  0.2426  3.1963 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  5.65461    0.02602 217.343   <2e-16 ***
citric_acid  0.51398    0.07429   6.918    5e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8701 on 6495 degrees of freedom
Multiple R-squared:  0.007316,  Adjusted R-squared:  0.007163 
F-statistic: 47.87 on 1 and 6495 DF,  p-value: 5.002e-12
```

```r
model_sulfur <- lm(quality~total_sulfur_dioxide,df)
summary(model_sulfur)
```

```

Call:
lm(formula = quality ~ total_sulfur_dioxide, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-2.8866 -0.7971  0.1658  0.2227  3.1965 
```

```
Coefficients:
                       Estimate Std. Error t value Pr(>|t|)    
(Intercept)           5.8923848  0.0246717 238.831  < 2e-16 ***
total_sulfur_dioxide -0.0006394  0.0001915  -3.338 0.000848 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.8726 on 6495 degrees of freedom
Multiple R-squared:  0.001713,	Adjusted R-squared:  0.001559 
F-statistic: 11.14 on 1 and 6495 DF,  p-value: 0.000848
```

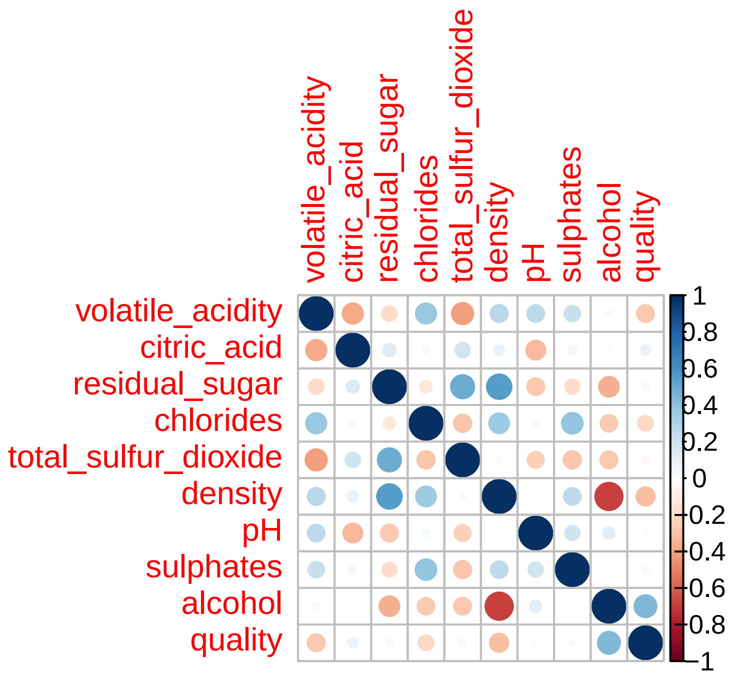
2.3 (5 points)

Visualize the correlation matrix of all numeric columns in `df` using `corrplot()`

```
sapply(df, class)
```

```
    volatile_acidity          citric_acid       residual_sugar 
           "numeric"            "numeric"            "numeric" 
           chlorides total_sulfur_dioxide              density 
           "numeric"            "numeric"            "numeric" 
                  pH            sulphates              alcohol 
           "numeric"            "numeric"            "numeric" 
             quality                 type 
           "numeric"             "factor" 
```

```
df23 <- df%>%
  select(!c(type))
dfcor = cor(df23)
corrplot(dfcor)
```



2.4 (5 points)

Compute the variance inflation factor (VIF) for each predictor in the full model using `vif()` function. What can we conclude from this?

Most predictors have a low vif value, where as type and density has a vif value of more than 5, meaning tha they are highly correlated and multicollinearity should be considered and the variables should be dropped.

```
vif(full_model)
```

```
volatile_acidity          citric_acid       residual_sugar
        2.103853             1.549248             4.680035
       chlorides total_sulfur_dioxide              density
        1.625065             2.628534             9.339357
              pH            sulphates              alcohol
        1.352005             1.522809             3.419849
            type
        6.694679
```

Question 3

40 points

Variable selection

3.1 (5 points)

Run a backward stepwise regression using a `full_model` object as the starting model. Store the final formula in an object called `backward_formula` using the built-in `formula()` function in R

```
backward <- step(full_model,direction='backward', scope=formula(full_model), trace=0)
backward_formula <- formula(backward)
backward_formula
```

```
quality ~ volatile_acidity + residual_sugar + chlorides + density +
    pH + sulphates + alcohol + type
```

3.2 (5 points)

Run a forward stepwise regression using a `null_model` object as the starting model. Store the final formula in an object called `forward_formula` using the built-in `formula()` function in R

```
null_model <- lm(quality ~ 1, data=df)
forward <- step(null_model,direction='forward', scope=formula(full_model), trace=0)
forward_formula <- formula(forward)
forward_formula
```

```
quality ~ alcohol + volatile_acidity + sulphates + residual_sugar +
    type + density + chlorides + pH
```

3.3 (10 points)

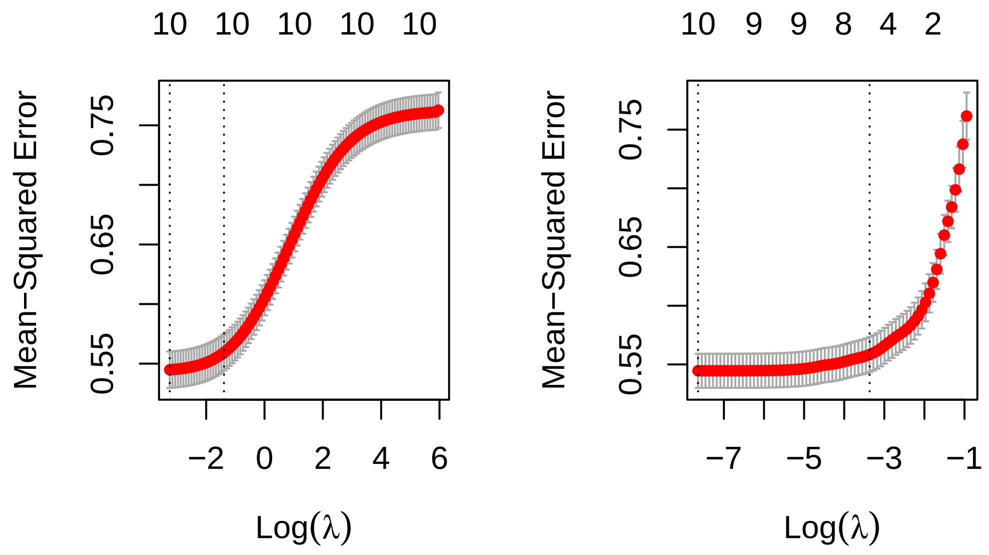
1. Create a `y` vector that contains the response variable (`quality`) from the `df` dataframe.
2. Create a design matrix `X` for the `full_model` object using the `make_model_matrix()` function provided in the Appendix.
3. Then, use the `cv.glmnet()` function to perform LASSO and Ridge regression with `X` and `y`.

```
y <- df$quality
X <- make_model_matrix(full_model)
lasso <- cv.glmnet(X,y,alpha=1)
ridge <- cv.glmnet(X,y,alpha=0)
```

Create side-by-side plots of the ridge and LASSO regression results. Interpret your main findings.

I have found that The minimum MSE is achieved when λ=0.03879736 for ridge regression, and the minimum MSE is achieved when λ=0.0006323003 for LASSO regression.

```
par(mfrow=c(1, 2))
plot(ridge)
plot(lasso)
```



```r
ridge$lambda.min
```

```
[1] 0.03879736
```

```r
lasso$lambda.min
```

```
[1] 0.000478312
```

3.4 (5 points)

Print the coefficient values for LASSO regression at the `lambda.1se` value? What are the variables selected by LASSO?

Store the variable names with non-zero coefficients in `lasso_vars`, and create a formula object called `lasso_formula` using the `make_formula()` function provided in the Appendix.

```r
lasso$lambda.1se
```

```
[1] 0.034538
```

```r
lasso_vars <- coef(lasso, s = "lambda.1se")
lasso_vars
```

```
11 x 1 sparse Matrix of class "dgCMatrix"
                              s1
(Intercept)           5.81837771
volatile_acidity     -0.18833502
citric_acid           .
residual_sugar        0.03434200
chlorides             .
total_sulfur_dioxide  .
density               .
pH                    .
sulphates             0.04909968
alcohol               0.35886559
type                  .
```

```r
lasso_formula <- make_formula(lasso_vars)
```

3.5 (5 points)

Print the coefficient values for ridge regression at the `lambda.1se` value? What are the variables selected here?

Store the variable names with non-zero coefficients in `ridge_vars`, and create a formula object called `ridge_formula` using the `make_formula()` function provided in the Appendix.

```r
ridge$lambda.1se
```

```
[1] 0.2493923
```

```r
ridge_vars <- coef(ridge, s = "lambda.1se")
ridge_vars
```

```
11 x 1 sparse Matrix of class "dgCMatrix"
                              s1
(Intercept)           5.85958721
volatile_acidity     -0.16471804
citric_acid           0.02178118
residual_sugar        0.08858405
chlorides            -0.04745954
total_sulfur_dioxide -0.03952828
density              -0.08354208
pH                    0.02378773
sulphates             0.07807954
alcohol               0.25818761
type                 -0.05466274
```

```r
ridge_formula <- make_formula(ridge_vars)
```

3.6 (10 points)

What is the difference between stepwise selection, LASSO and ridge based on you analyses above?

The stepwise regressions generated the same 8-predictor model, Lasso formula included the least terms, Ridge formula included all 8 predictors, but many of them has very small slope.

```
summary(backward)
```

```
Call:
lm(formula = quality ~ volatile_acidity + residual_sugar + chlorides + 
    density + pH + sulphates + alcohol + type, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.3317 -0.4695 -0.0422  0.4563  3.1248 

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)    
(Intercept)       57.225176   8.751451   6.539 6.67e-11 ***
volatile_acidity  -1.626323   0.074814 -21.738  < 2e-16 ***
residual_sugar     0.044254   0.003998  11.070  < 2e-16 ***
chlorides         -0.950667   0.329461  -2.886  0.00392 ** 
density          -54.876246   8.714620  -6.297 3.23e-10 ***
pH                 0.175885   0.062554   2.812  0.00494 ** 
sulphates          0.652336   0.075353   8.657  < 2e-16 ***
alcohol            0.280731   0.013231  21.217  < 2e-16 ***
typewhite         -0.417595   0.046578  -8.966  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.737 on 6488 degrees of freedom
Multiple R-squared:  0.2885,    Adjusted R-squared:  0.2877 
F-statistic: 328.9 on 8 and 6488 DF,  p-value: < 2.2e-16
```

```
summary(forward)
```

```
Call:
lm(formula = quality ~ alcohol + volatile_acidity + sulphates + 
    residual_sugar + type + density + chlorides + pH, data = df)
```

```
Residuals:
    Min      1Q  Median      3Q     Max 
-3.3317 -0.4695 -0.0422  0.4563  3.1248 

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)    
(Intercept)       57.225176   8.751451   6.539 6.67e-11 ***
alcohol            0.280731   0.013231  21.217  < 2e-16 ***
volatile_acidity  -1.626323   0.074814 -21.738  < 2e-16 ***
sulphates          0.652336   0.075353   8.657  < 2e-16 ***
residual_sugar     0.044254   0.003998  11.070  < 2e-16 ***
typewhite         -0.417595   0.046578  -8.966  < 2e-16 ***
density          -54.876246   8.714620  -6.297 3.23e-10 ***
chlorides         -0.950667   0.329461  -2.886  0.00392 ** 
pH                 0.175885   0.062554   2.812  0.00494 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.737 on 6488 degrees of freedom
Multiple R-squared:  0.2885,    Adjusted R-squared:  0.2877 
F-statistic: 328.9 on 8 and 6488 DF,  p-value: < 2.2e-16
```

```
lasso_formula
```

```
quality ~ 5.81837771279066 + -0.188335020240248 + 0 + 0.0343420015225683 + 
    0 + 0 + 0 + 0 + 0.0490996756976685 + 0.358865586117007 + 
    0
<environment: 0x7f88e4a4c8d0>
```

```
ridge_formula
```

```
quality ~ 5.8595872105273 + -0.164718041565637 + 0.0217811819274036 + 
    0.0885840504798778 + -0.0474595383202974 + -0.0395282781377516 + 
    -0.0835420820358288 + 0.0237877298703364 + 0.0780795437330479 + 
    0.258187612588152 + -0.0546627412811334
<environment: 0x7f88ea4ddca0>
```

Question 4

70 points

Variable selection

4.1 (5 points)

Excluding `quality` from `df` we have 10 possible predictors as the covariates. How many different models can we create using any subset of these 10 coavriates as possible predictors? Justify your answer.

There could be 3628801 models. (Factorial of 10 gives the permutation of the variables, and +1 for the null model)

```
factorial(10)
```

```
[1] 3628800
```

4.2 (20 points)

Store the names of the predictor variables (all columns except `quality`) in an object called `x_vars`.

```
x_vars <- colnames(df %>% select(-quality))
```

Use:

- the `combn()` function (built-in R function) and
- the `make_formula()` (provided in the Appendix)

to **generate all possible linear regression formulas** using the variables in `x_vars`. This is most optimally achieved using the `map()` function from the `purrr` package.

```
length42 <-  1:length(x_vars)
formulas <- map(
  length42,
  \(x){
    vars <- combn(x_vars,x) # Insert code here
    map(split(vars,rep(1:ncol(vars),each=nrow(vars))), make_formula) # Insert code here
  }
) %>% unlist()
```

If your code is right the following command should return something along the lines of:

```
sample(formulas, 4) %>% as.character()
```

```
[1] "quality ~ chlorides + total_sulfur_dioxide + density"
[2] "quality ~ residual_sugar + chlorides + total_sulfur_dioxide + density + pH + type"
[3] "quality ~ citric_acid"
[4] "quality ~ citric_acid + chlorides + pH + type"
```

```
# Output:
# [1] "quality ~ volatile_acidity + residual_sugar + density + pH + alcohol"
# [2] "quality ~ citric_acid"
# [3] "quality ~ volatile_acidity + citric_acid + residual_sugar + total_sulfur_dioxide +
# [4] "quality ~ citric_acid + chlorides + total_sulfur_dioxide + pH + alcohol + type"
```

4.3 (10 points)

Use `map()` and `lm()` to fit a linear regression model to each formula in `formulas`, using `df` as the data source. Use `broom::glance()` to extract the model summary statistics, and bind them together into a single tibble of summaries using the `bind_rows()` function from `dplyr`.

```
models <- map(formulas,~lm(quality~.,data=df)) # Insert your code here
#summaries <- map(data.frame(models), bind_rows(broom::glance)) # Insert your code here
```

4.4 (5 points)

Extract the `adj.r.squared` values from `summaries` and use them to identify the formula with the ***highest*** adjusted R-squared value.

```
#summaries%>%
  #select(adj.r.squared)
```

Store resulting formula as a variable called `rsq_formula`.

```
rsq_formula <- ... # Insert your code
```

4.5 (5 points)

Extract the `AIC` values from `summaries` and use them to identify the formula with the ***lowest*** AIC value.

```
... # Insert your code here
```

Store resulting formula as a variable called `aic_formula`.

```
aic_formula <- ... # Insert your code
```

4.6 (15 points)

Combine all formulas shortlisted into a single vector called `final_formulas`.

```
null_formula <- formula(null_model)
full_formula <- formula(full_model)

final_formulas <- c(
  null_formula,
  full_formula,
  backward_formula,
  forward_formula,
  lasso_formula,
  ridge_formula,
  #rsq_formula,
  #aic_formula
)
```

- Are `aic_formula` and `rsq_formula` the same? How do they differ from the formulas shortlisted in question 3?
- Which of these is more reliable? Why?
- If we had a dataset with $10,000$ columns, which of these methods would you consider for your analyses? Why?

4.7 (10 points)

Use `map()` and `glance()` to extract the `sigma`, `adj.r.squared`, `AIC`, `df`, and `p.value` statistics for each model obtained from `final_formulas`. Bind them together into a single data frame `summary_table`. Summarize your main findings.

```r
summary_table <- map(
  final_formulas,
  \(x) ... # Insert your code here
) %>% bind_rows()

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

Appendix

Convenience function for creating a formula object

The following function which takes as input a vector of column names `x` and outputs a `formula` object with `quality` as the response variable and the columns of `x` as the covariates.

```r
make_formula <- function(x){
  as.formula(
    paste("quality ~ ", paste(x, collapse = " + "))
  )
}

# For example the following code will
# result in a formula object
# "quality ~ a + b + c"
make_formula(c("a", "b", "c"))
```

```
quality ~ a + b + c
<environment: 0x7f88e5f42680>
```

Convenience function for `glmnet`

The `make_model_matrix` function below takes a `formula` as input and outputs a **rescaled** model matrix `X` in a format amenable for `glmnet()`

```r
make_model_matrix <- function(formula){
  X <- model.matrix(formula, df)[, -1]
  cnames <- colnames(X)
  for(i in 1:ncol(X)){
    if(!cnames[i] == "typewhite"){
      X[, i] <- scale(X[, i])
    } else {
      colnames(X)[i] <- "type"
    }
  }
  return(X)
}
```

Session Information

Print your `R` session information using the following command

```r
sessionInfo()
```

```
R version 4.2.1 (2022-06-23)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Big Sur ... 10.16

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2/Resources/lib/libRlapack.dylib

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices datasets  utils     methods   base

other attached packages:
 [1] janitor_2.2.0 corrplot_0.92 glmnet_4.1-6  Matrix_1.4-1  car_3.1-1
 [6] carData_3.0-5 purrr_1.0.1   dplyr_1.1.0   tidyr_1.3.0   readr_2.1.4

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.10      pillar_1.8.1     compiler_4.2.1   iterators_1.0.14
 [5] tools_4.2.1      digest_0.6.31    timechange_0.2.0 lubridate_1.9.2
 [9] jsonlite_1.8.4   evaluate_0.20    lifecycle_1.0.3  tibble_3.1.8
[13] lattice_0.20-45  pkgconfig_2.0.3  rlang_1.0.6      foreach_1.5.2
[17] cli_3.6.0        yaml_2.3.7       xfun_0.37        fastmap_1.1.1
[21] withr_2.5.0      stringr_1.5.0    knitr_1.42       generics_0.1.3
[25] vctrs_0.5.2      hms_1.1.2        grid_4.2.1       tidyselect_1.2.0
[29] snakecase_0.11.0 glue_1.6.2       R6_2.5.1         fansi_1.0.4
[33] survival_3.3-1   rmarkdown_2.20   tzdb_0.3.0       magrittr_2.0.3
[37] backports_1.4.1  splines_4.2.1    codetools_0.2-18 ellipsis_0.3.2
[41] htmltools_0.5.4  abind_1.4-5      shape_1.4.6      renv_0.16.0-53
[45] utf8_1.2.3       stringi_1.7.12   broom_1.0.3
```