Higher Dimensional Regression Methods

STA378: Research Project

Mohammed Hamlili

1 Regression: An overview.

Statistical learning is a field that focuses on developing methods to analyze and extract knowledge from data. It involves building models that can generalize well to new data and predict the outcome variable accurately. Regression analysis is a fundamental statistical leaning tool for modeling the relationship between a response variable and one or more predictor variables. The traditional approach to regression analysis is to fit a linear model to the data, which assumes that the relationship between the variables is linear. Regression analysis is widely used in many fields, including economics, finance, psychology, and engineering. For example, in economics, regression analysis can be used to model the relationship between a company's revenue and its advertising budget. In finance, regression analysis can be used to model the relationship between a stock's price and various economic indicators. In psychology, regression analysis can be used to model the relationship between a person's happiness and their social support network. In engineering, regression analysis can be used to model the relationship between a product's performance and its various design parameters.

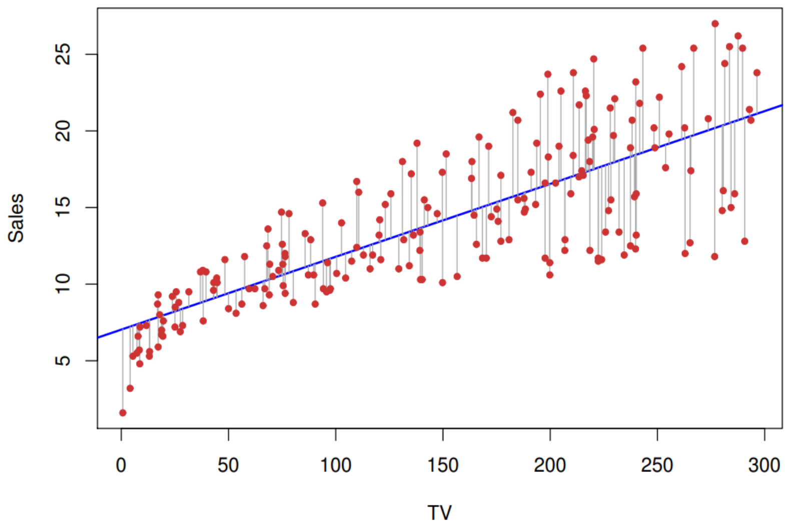
The simple linear regression model is widely used when we have only one predictor variable. The multiple linear regression model extends the simple linear regression model and allows us to examine the effect of multiple predictor variables on the response variable. In this paper, we will provide an overview of simple and multiple linear regression and introduce higher dimensional regression methods.

1.1 Simple Linear Regression

Simple linear regression is a statistical technique that investigates the linear relationship between a dependent variable and a single independent variable. In other words, simple linear regression aims to fit a line that best describes the relationship between two variables, and provides the most accurate predictions. The equation for a simple linear regression model is as follows:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where Y is a dependent variable, X is an independent variable, β_0 is the intercept of the relationship between the 2 variables, and β_1 is the slope. Translating the goal of simple linear regression amounts to estimating the values of β_0 and β_1 that minimize the sum of residuals, where the residual of a point is the difference $(y_i - \hat{y}_i)$, $\hat{y}_i$ being the predicted value of y_i. The line created with the values $\hat{\beta}_0$ and $\hat{\beta}_1$ is called the ordinary least squares regression line.



The above picture shows actual data with the least squares fit in blue. The vertical lines that extend from the blue line to the point that projects onto the line are the residuals of each point in the data.

1.2 Multiple Linear Regression

Multiple linear regression is the extension of the single linear regression model to the case where we want to examine the linear relationship between a dependent variable and multiple independent variables. In other words, multiple linear regression aims to fit a hyperplane that best describes the relationship between the response variable and the predictor variables.

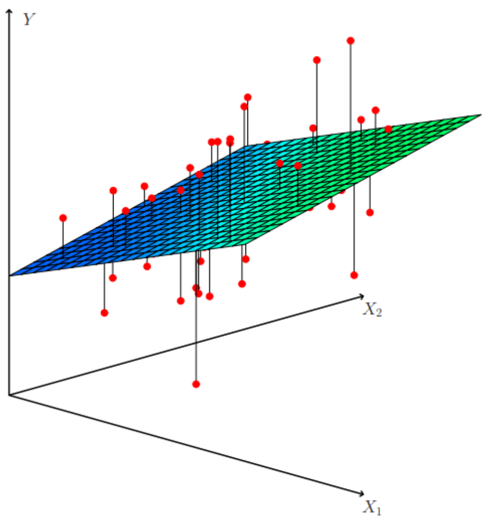


FIGURE 3.4. *In a three-dimensional setting, with two predictors and one response, the least squares regression line becomes a plane. The plane is chosen to minimize the sum of the squared vertical distances between each observation (shown in red) and the plane.*

The above picture gives the equivalent analysis method for two predictors instead of one. The equation for a linear regression model with p predictors becomes :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon$$

and the least squares fit now tries to estimate all the β_i parameters that minimize the sum of squared errors between the observed and predicted values of Y.

1.3 Rise of a problem: Collinearity

Collinearity is a common problem that arises in multiple linear regression when two or more predictor variables are closely related independent variables. It can lead to inaccurate and unstable coefficient estimates, making it difficult to interpret the results of the regression analysis. It is difficult to separate out the individual effects of the collinear variables. This implies that small changes to the data may lead to very different estimates for the parameters in the collinear case. Mathematically, this translates into a large standard error for the coefficient estimates, and low t-statistics. This can lead to the incorrect conclusion that a variable is not significant when it is.

A simple way to detect collinearity is to take a look at the correlation matrix of the predictors and identify which variables are highly correlated. High absolute values of correlation between 2 variables indicate a collinearity problem in the data, which has to be taken into account when fitting the model. However, is is possible for collinearity to exist even if no pair of variables are highly correlated: That is when 3 or more variables are correlated, which brings us to what we call multicollinearity.

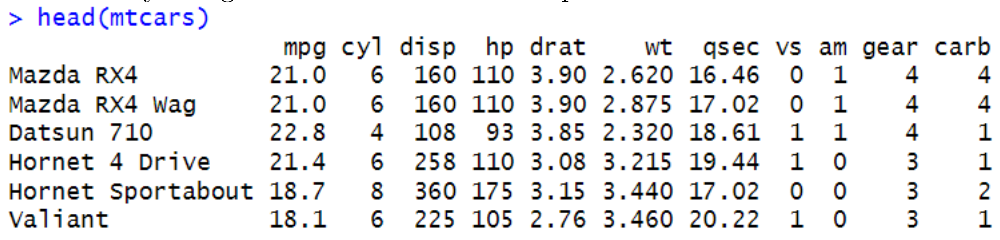
In addition to collinearity, high-dimensional regression analysis can also face other challenges such as overfitting and model complexity. Overfitting occurs when the model fits the training data too closely, leading to poor performance on new data. Model complexity refers to the degree of flexibility of the model, which can lead to overfitting if the model is too complex.

High-dimensional regression analysis is an important area of statistical learning that has gained tremendous attention in recent years due to the emergence of high-dimensional data. Traditional regression analysis faces various challenges when the number of predictors increases, including multicollinearity, overfitting, and model complexity. Various high-dimensional regression methods have been developed to address these challenges, including regularization and variable selection methods. These methods provide robust and accurate predictions and are widely used in various fields, including genetics, finance, and social sciences. The objective of this study is to go over the "traditional" way of handling high-dimensional data and then present other methods and go over their benefits and why they outperform the standard analysis method.

2 Case Study: Simple Analysis of high dimensional data using Multiple Linear Regression

For this analysis, we will be using the**mtcars** dataset, which is built-in R. This analysis is done with a dataset that allows for a simple analysis in order to motivate the further study of high dimensional regressional methods.

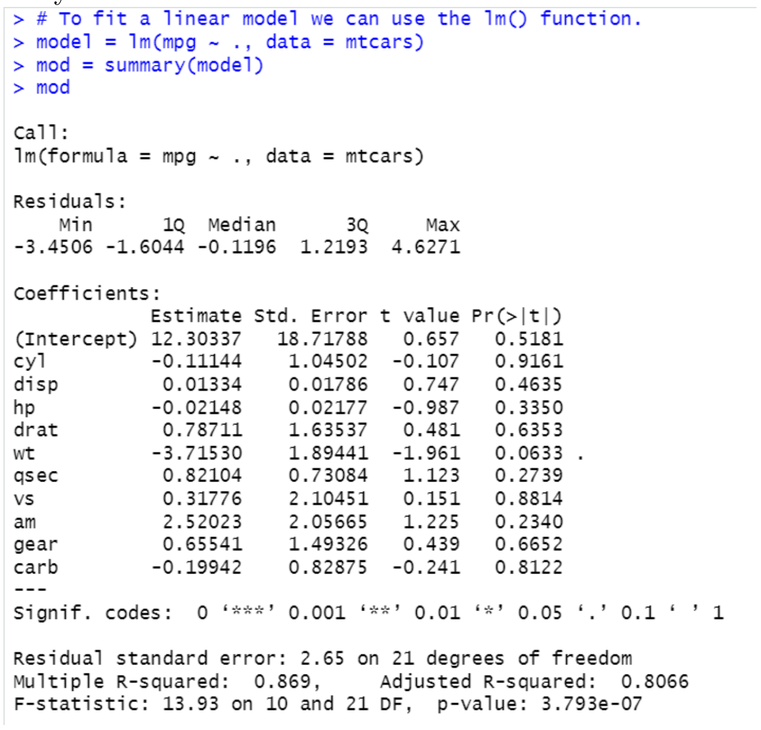
As explained by the R Documentation, this dataset " was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles ". We can obtain a look at what the data looks like by using the head function which provides the first 6 records of the dataset.



```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

A useful study would be to analyze how various features of the car impact the miles that can be covered per gallon of gas. The use of this would be to improve the gas usage which has an obviously important environmental effect.

We can start by fitting a linear model using all predictors, and then displaying the summary of the model as follows:



```
> # To fit a linear model we can use the lm() function.
> model = lm(mpg ~ ., data = mtcars)
> mod = summary(model)
> mod

Call:
lm(formula = mpg ~ ., data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.4506 -1.6044 -0.1196  1.2193  4.6271 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)  
(Intercept) 12.30337   18.71788   0.657   0.5181  
cyl         -0.11144    1.04502  -0.107   0.9161  
disp         0.01334    0.01786   0.747   0.4635  
hp          -0.02148    0.02177  -0.987   0.3350  
drat         0.78711    1.63537   0.481   0.6353  
wt          -3.71530    1.89441  -1.961   0.0633 .
qsec         0.82104    0.73084   1.123   0.2739  
vs           0.31776    2.10451   0.151   0.8814  
am           2.52023    2.05665   1.225   0.2340  
gear         0.65541    1.49326   0.439   0.6652  
carb        -0.19942    0.82875  -0.241   0.8122  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.65 on 21 degrees of freedom
Multiple R-squared:  0.869,	Adjusted R-squared:  0.8066 
F-statistic: 13.93 on 10 and 21 DF,  p-value: 3.793e-07
```

Notice that the "~" is used to denote all the variables. The multiple R-squared is obviously high, since it always increases when fitting additional variables. That doesn't mean that the model is adequate however since once we add insignificant predictors, the R^2 just keeps on increasing slightly or stays constant. As is reflected by the t-statistics above, we can not find confident proof that any of the variables is significant within this model, except for *wt* whose t-value is relatively smaller than others. This is a typical problem due to collinearity.

The first we can do is take a look at the correlation matrix of all variables. We can generate that matrix as follows:

```
> cor(mtcars)
            mpg        cyl       disp         hp        drat         wt        qsec         vs          am       gear        carb
mpg   1.0000000 -0.8521620 -0.8475514 -0.7761684  0.68117191 -0.8676594  0.41868403  0.6640389  0.59983243  0.4802848 -0.55092507
cyl  -0.8521620  1.0000000  0.9020329  0.8324475 -0.69993811  0.7824958 -0.59124207 -0.8108118 -0.52260705 -0.4926866  0.52698829
disp -0.8475514  0.9020329  1.0000000  0.7909486 -0.71021393  0.8879799 -0.43369788 -0.7104159 -0.59122704 -0.5555692  0.39497686
hp   -0.7761684  0.8324475  0.7909486  1.0000000 -0.44875912  0.6587479 -0.70822339 -0.7230967 -0.24320426 -0.1257043  0.74981247
drat  0.6811719 -0.6999381 -0.7102139 -0.4487591  1.00000000 -0.7124406  0.09120476  0.4402785  0.71271113  0.6996101 -0.09078980
wt   -0.8676594  0.7824958  0.8879799  0.6587479 -0.71244065  1.0000000 -0.17471588 -0.5549157 -0.69249526 -0.5832870  0.42760594
qsec  0.4186840 -0.5912421 -0.4336979 -0.7082234  0.09120476 -0.1747159  1.00000000  0.7445354 -0.22986086 -0.2126822 -0.65624923
vs    0.6640389 -0.8108118 -0.7104159 -0.7230967  0.44027846 -0.5549157  0.74453544  1.0000000  0.16834512  0.2060233 -0.56960714
am    0.5998324 -0.5226070 -0.5912270 -0.2432043  0.71271113 -0.6924953 -0.22986086  0.1683451  1.00000000  0.7940588  0.05753435
gear  0.4802848 -0.4926866 -0.5555692 -0.1257043  0.69961013 -0.5832870 -0.21268223  0.2060233  0.79405876  1.0000000  0.27407284
carb -0.5509251  0.5269883  0.3949769  0.7498125 -0.09078980  0.4276059 -0.65624923 -0.5696071  0.05753435  0.2740728  1.00000000
```

We analyze this matrix by looking at each entry not on the diagonal. The entry at index i, j will give us the correlation coefficient between the predictors X_i and X_j. An absolute value that is high enough would indicate that the 2 variables are highly correlated, and therefore implies that there is a collinearity problem when fitting all the variables against *mpg*. In fact, we can easily find pairs of variables that have a correlation coefficient by looking at this matrix. For example, *cyl* and *vs* have a coefficient of -0.8108.

Another way to identify multicollinearity (when more than 2 predictors are highly correlated) is by analyzing the VIF values of each predictor. Variance Inflation Factor (VIF) is the ratio of the variance of $\hat{\beta}_j$ when fitting the full model divided by the variance if fit on its own. It is as such a measure of how other predictors in the model "inflate" its variance. Theoretically, it is computed easily as the following ratio:

$$VIF(\hat{\beta}_j) = \frac{1}{R_{X_j|X_{-j}}}$$

where $R_{X_j|X_{-j}}$ is used to indicate the R^2 error from a regression of the predictor X_j on all other predictors.

Higher values of VIF indicate that a predictor has a collinearity problem. There are many different cutoffs according to different fields and their different requirements, so we can choose a cutoff of 10 where that indicates a collinearity problem. Obtaining the VIF values for each predictors in the full model (i.e. the model with all predictors) this is what we obtain:

```
> library(car)
> vif(model)
      cyl      disp        hp      drat        wt      qsec        vs        am      gear 
15.373833 21.620241  9.832037  3.374620 15.164887  7.527958  4.965873  4.648487  5.357452 
     carb 
 7.908747 
```

We can identify *cyl* and *disp* as having the higher VIF values.

As demonstrated, the full model is evidently not the ideal model and possesses collinearity problems. Thus, we have to eliminate redundant and insignificant predictors. We can apply a method that was traditionally developed to deal with the high dimensional challenges of regression analysis, which is Variable Selection.

2.1 Variable Stepwise Selection

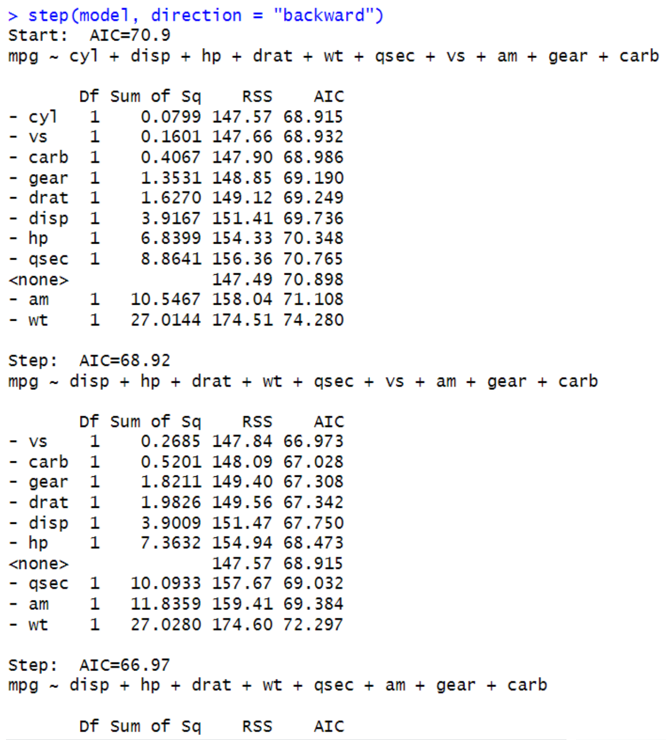
Variable selection methods involve selecting a subset of the independent variables that are most relevant to the dependent variable. The most commonly used variable selection methods are Forward Stepwise Selection, Backward Stepwise Selection, and Best Subset Selection. Forward Stepwise Selection starts with no variables and adds variables one at a time, based on their contribution to the model. Backward Stepwise Selection starts with all the variables and removes them one at a time, based on their contribution to the model. Best Subset Selection considers all possible subsets of variables and selects the best subset based on some criterion.

Best subset selection, forward selection and backward selection all end up with the same final step, where we have a candidate list of models and we need a way to determine which is best. Standard universal statistics such as RSS (residual error square) or R^2 are adapted to the training error and are thus not suitable to be the criterion for selecting a model. In order to find a criterion that helps select the best model, one can either indirectly estimate the test error by adjusting the training error to account for overfitting, or directly estimate it using a cross validation approach. The following criteria adjust the train error for the model size in order to estimate the test error: Mallows' C_p, Akaike information criterion (AIC), Bayesian information criterion (BIC), the adjusted R^2. The AIC criterion is defined for a large class of models fit by maximum likelihood. In our specific case study of *mtcars*, the maximum likelihood refers to least squares. AIC is thus given by :

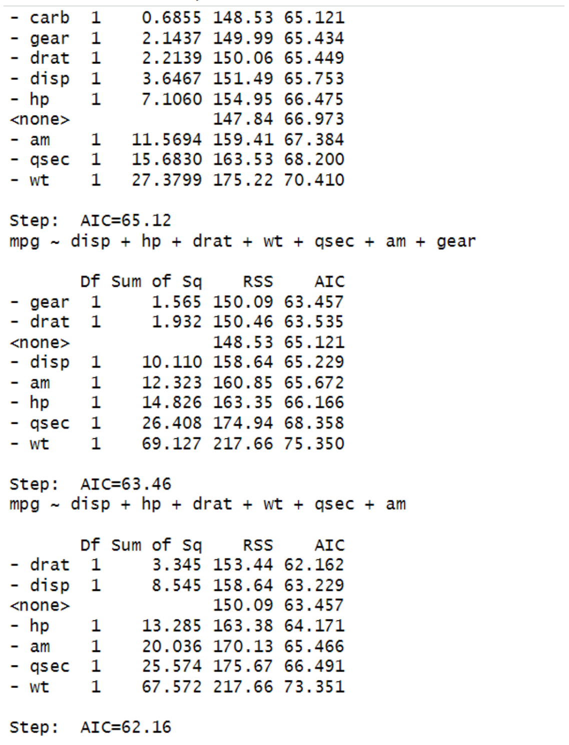
$$AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2)$$

AIC has a rigorous justification beyond the scope of this paper, but what is of interest here is how it can be used in order to express backward stepwise regression in order to find the best possible model according to this criterion.

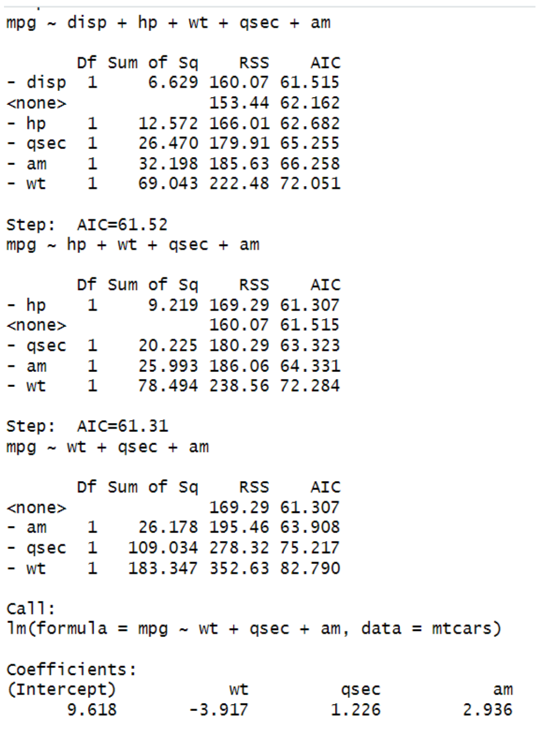
The Backward Stepwise Selection will have to compare p models that are fitted with $p-1$ predictors in order to determine the best model. Then, it chooses the model and repeats the same step until it becomes worst to take off a variable. This represents a computational problem, as the algorithm has to compare what can become a very large number of models once p becomes large enough. In our case study, we can make use of the function *step* provided by R, which generates the best model using Backward Stepwise Subset selection. The following results give us a model with 3 predictors instead of the initial 10.



```
> step(model, direction = "backward")
Start:  AIC=70.9
mpg ~ cyl + disp + hp + drat + wt + qsec + vs + am + gear + carb

       Df Sum of Sq    RSS    AIC
- cyl   1    0.0799 147.57 68.915
- vs    1    0.1601 147.66 68.932
- carb  1    0.4067 147.90 68.986
- gear  1    1.3531 148.85 69.190
- drat  1    1.6270 149.12 69.249
- disp  1    3.9167 151.41 69.736
- hp    1    6.8399 154.33 70.348
- qsec  1    8.8641 156.36 70.765
<none>              147.49 70.898
- am    1   10.5467 158.04 71.108
- wt    1   27.0144 174.51 74.280

Step:  AIC=68.92
mpg ~ disp + hp + drat + wt + qsec + vs + am + gear + carb

       Df Sum of Sq    RSS    AIC
- vs    1    0.2685 147.84 66.973
- carb  1    0.5201 148.09 67.028
- gear  1    1.8211 149.40 67.308
- drat  1    1.9826 149.56 67.342
- disp  1    3.9009 151.47 67.750
- hp    1    7.3632 154.94 68.473
<none>              147.57 68.915
- qsec  1   10.0933 157.67 69.032
- am    1   11.8359 159.41 69.384
- wt    1   27.0280 174.60 72.297

Step:  AIC=66.97
mpg ~ disp + hp + drat + wt + qsec + am + gear + carb

       Df Sum of Sq    RSS    AIC
```

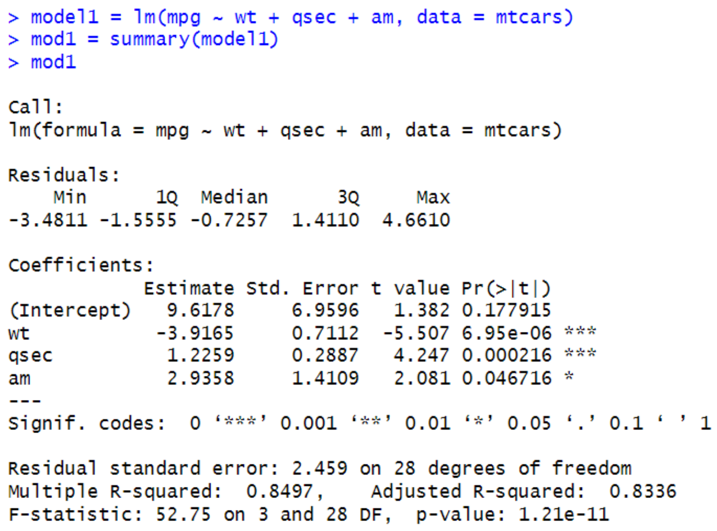


```
- carb  1    0.6855 148.53 65.121
- gear  1    2.1437 149.99 65.434
- drat  1    2.2139 150.06 65.449
- disp  1    3.6467 151.49 65.753
- hp    1    7.1060 154.95 66.475
<none>              147.84 66.973
- am    1   11.5694 159.41 67.384
- qsec  1   15.6830 163.53 68.200
- wt    1   27.3799 175.22 70.410

Step:  AIC=65.12
mpg ~ disp + hp + drat + wt + qsec + am + gear

       Df Sum of Sq    RSS    AIC
- gear  1     1.565 150.09 63.457
- drat  1     1.932 150.46 63.535
<none>              148.53 65.121
- disp  1    10.110 158.64 65.229
- am    1    12.323 160.85 65.672
- hp    1    14.826 163.35 66.166
- qsec  1    26.408 174.94 68.358
- wt    1    69.127 217.66 75.350

Step:  AIC=63.46
mpg ~ disp + hp + drat + wt + qsec + am

       Df Sum of Sq    RSS    AIC
- drat  1     3.345 153.44 62.162
- disp  1     8.545 158.64 63.229
<none>              150.09 63.457
- hp    1    13.285 163.38 64.171
- am    1    20.036 170.13 65.466
- qsec  1    25.574 175.67 66.491
- wt    1    67.572 217.66 73.351

Step:  AIC=62.16
```



```
mpg ~ disp + hp + wt + qsec + am

       Df Sum of Sq    RSS    AIC
- disp  1     6.629 160.07 61.515
<none>              153.44 62.162
- hp    1    12.572 166.01 62.682
- qsec  1    26.470 179.91 65.255
- am    1    32.198 185.63 66.258
- wt    1    69.043 222.48 72.051

Step:  AIC=61.52
mpg ~ hp + wt + qsec + am

       Df Sum of Sq    RSS    AIC
- hp    1     9.219 169.29 61.307
<none>              160.07 61.515
- qsec  1    20.225 180.29 63.323
- am    1    25.993 186.06 64.331
- wt    1    78.494 238.56 72.284

Step:  AIC=61.31
mpg ~ wt + qsec + am

       Df Sum of Sq    RSS    AIC
<none>              169.29 61.307
- am    1    26.178 195.46 63.908
- qsec  1   109.034 278.32 75.217
- wt    1   183.347 352.63 82.790

Call:
lm(formula = mpg ~ wt + qsec + am, data = mtcars)

Coefficients:
(Intercept)           wt         qsec           am  
      9.618       -3.917        1.226        2.936  
```

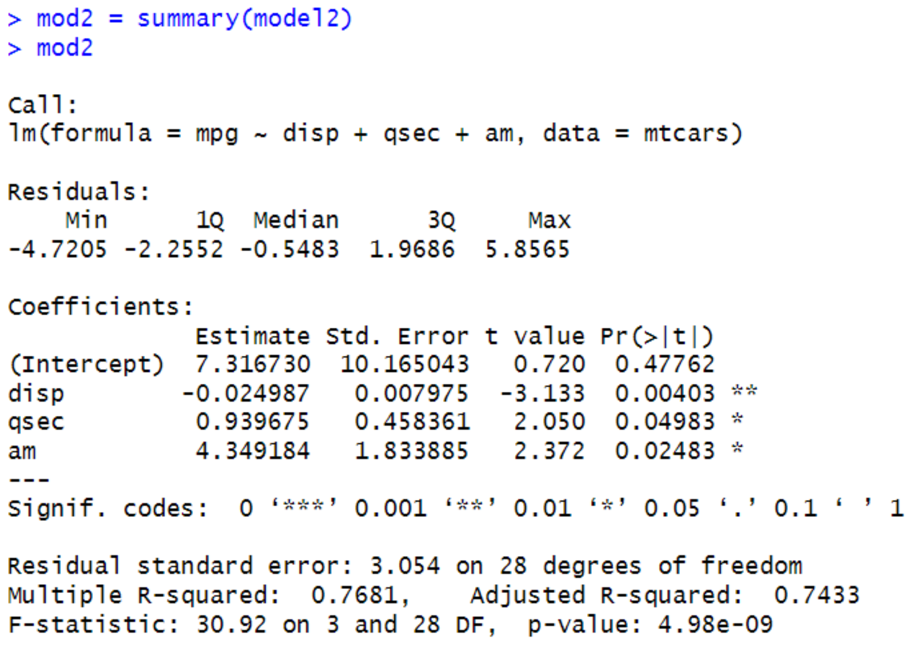
The predictors retained at the end of the process (i.e. the predictors that yield the best model using Backwards Selection Regression) are *wt*, *qsec* and *am*. This is the summary of the model with these predictors.



```
> model1 = lm(mpg ~ wt + qsec + am, data = mtcars)
> mod1 = summary(model1)
> mod1

Call:
lm(formula = mpg ~ wt + qsec + am, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.4811 -1.5555 -0.7257  1.4110  4.6610 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   9.6178     6.9596   1.382 0.177915    
wt           -3.9165     0.7112  -5.507 6.95e-06 ***
qsec          1.2259     0.2887   4.247 0.000216 ***
am            2.9358     1.4109   2.081 0.046716 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.459 on 28 degrees of freedom
Multiple R-squared:  0.8497,	Adjusted R-squared:  0.8336 
F-statistic: 52.75 on 3 and 28 DF,  p-value: 1.21e-11
```

We can notice that unlike the initial model, the p values of each predictors is below 5%, so we can be fairly confident about the significance of each variable in the model. It doesn't hurt to compare other models in order to see how well this one performs.

If we look at the correlation matrix, we can see wt is highly correlated with disp and am is highly correlated with gear. We can try fitting another model with these 2 predictors respectively replaced. This is the following model output :

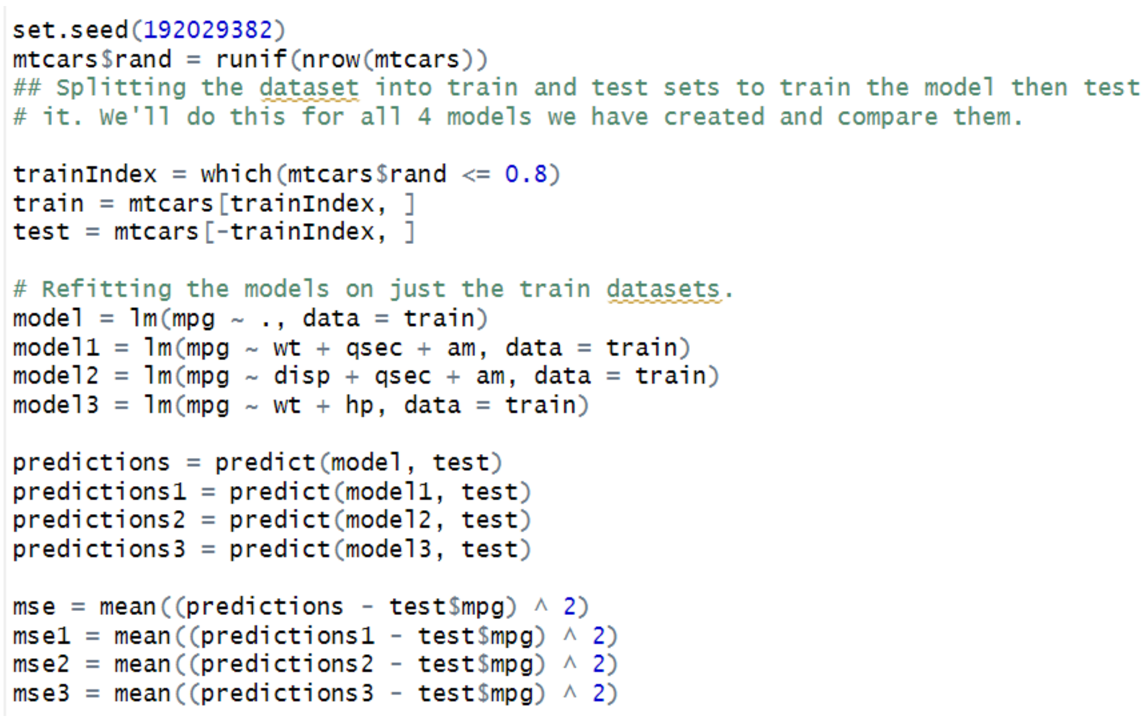


```
> mod2 = summary(model2)
> mod2

Call:
lm(formula = mpg ~ disp + qsec + am, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.7205 -2.2552 -0.5483  1.9686  5.8565 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)   
(Intercept)  7.316730  10.165043   0.720  0.47762   
disp        -0.024987   0.007975  -3.133  0.00403 **
qsec         0.939675   0.458361   2.050  0.04983 * 
am           4.349184   1.833885   2.372  0.02483 * 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.054 on 28 degrees of freedom
Multiple R-squared:  0.7681,	Adjusted R-squared:  0.7433 
F-statistic: 30.92 on 3 and 28 DF,  p-value: 4.98e-09
```

Here again, we see that the p-values lead to the conclusion that the predictors are significant. The R^2 statistic however is lower than for the previous model. It might also be useful to compare these models with a model containing only 2 predictors :

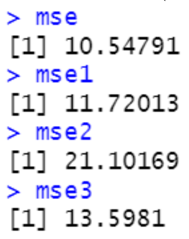
```
> model3 = lm(mpg ~ wt + hp, data = mtcars)
> mod3 = summary(model3)
> mod3

Call:
lm(formula = mpg ~ wt + hp, data = mtcars)

Residuals:
   Min     1Q Median     3Q    Max 
-3.941 -1.600 -0.182  1.050  5.854 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 37.22727    1.59879  23.285  < 2e-16 ***
wt          -3.87783    0.63273  -6.129 1.12e-06 ***
hp          -0.03177    0.00903  -3.519  0.00145 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.593 on 29 degrees of freedom
Multiple R-squared:  0.8268,	Adjusted R-squared:  0.8148 
F-statistic: 69.21 on 2 and 29 DF,  p-value: 9.109e-12
```

We can compare all 4 models of this case study by splitting the dataset into a training set and testing set and running comparing the mean squared error of the predictions of each model. The splitting is done randomly by generating a new predictor which is just a random number associated to each data point. 80% of the full data set is used as the training data set, and the rest will be the testing data set. The following shows how to reproduce this:



```r
set.seed(192029382)
mtcars$rand = runif(nrow(mtcars))
## Splitting the dataset into train and test sets to train the model then test
# it. We'll do this for all 4 models we have created and compare them.

trainIndex = which(mtcars$rand <= 0.8)
train = mtcars[trainIndex, ]
test = mtcars[-trainIndex, ]

# Refitting the models on just the train datasets.
model = lm(mpg ~ ., data = train)
model1 = lm(mpg ~ wt + qsec + am, data = train)
model2 = lm(mpg ~ disp + qsec + am, data = train)
model3 = lm(mpg ~ wt + hp, data = train)

predictions = predict(model, test)
predictions1 = predict(model1, test)
predictions2 = predict(model2, test)
predictions3 = predict(model3, test)

mse = mean((predictions - test$mpg) ^ 2)
mse1 = mean((predictions1 - test$mpg) ^ 2)
mse2 = mean((predictions2 - test$mpg) ^ 2)
mse3 = mean((predictions3 - test$mpg) ^ 2)
```

The results (i.e. the mean squared errors of the 4 models) are as follows:



```
> mse
[1] 10.54791
> mse1
[1] 11.72013
> mse2
[1] 21.10169
> mse3
[1] 13.5981
```

3 Shrinkage Methods: Lasso & Ridge

The computational complexity of stepwise variable selection makes it an undesirable option once we start considering models with high complexity. New methods have recently emerged that tackle the multicollinearity problem in a completely different manner, which is the goal of this paper.

Although stepwise variable selection can help mitigate the multicollinearity problem by selecting a subset of predictors, it has some disadvantages. For example, it relies on p-values to add or remove predictors, which can be unstable and sensitive to small changes in the data. Additionally, stepwise selection ignores the joint effect of predictors, which can lead to biased estimates and poor predictive performance.

To address these issues, a popular alternative is to use regularization methods such as Lasso and Ridge Regression. These methods add a penalty term to the linear regression objective function, which shrinks the coefficients towards zero and reduces overfitting.
In particular, Lasso regression uses the L1-norm penalty, which encourages sparsity and can perform variable selection by setting some coefficients to zero. On the other hand, Ridge regression uses the L2-norm penalty, which smooths the coefficients and can improve the conditioning of the design matrix.
To demonstrate how to use Lasso and Ridge Regression, we can fit these models to the same dataset and compare their performance to the stepwise selection method. We can also discuss how to tune the regularization hyperparameters using cross-validation, and interpret the resulting models in terms of feature importance and coefficient magnitude.

Overall, regularization methods like Lasso and Ridge Regression can provide a more principled and flexible approach to handling multicollinearity and improving the predictive power of linear regression models.

3.1 L1-norm & L2-norm penalty

When we fit a linear regression model, we usually seek to minimize the sum of squared errors between the predicted values and the actual values of the response variable. This objective function is often written as:

$$RSS = ||Y - X\beta||^2 = \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2$$

where Y is the vector of observed response values, X is the matrix of predictor variables, β is the vector of regression coefficients, and $||.||^2$ denotes the squared Euclidean norm.

Ridge does something very similar, minimizing the following mathematical expression:

$$\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

, where λ is commonly referred to as the tuning parameter. When combined with the sum of β_js, the term is called the shrinkage penalty or penalty term.

To avoid overfitting, we can add this penalty term to the objective function that encourages smaller values of the regression coefficients. As we can see from the equation, the penalty term is essentially a function of the magnitudes of the coefficients, and it can be based on either the L1-norm or the L2-norm of the coefficients.

The L2-norm penalty is given by:

$$\lambda||\beta||_2^2$$

where $||.||_2$ denotes the L2-norm, which is the square root of the sum of the squared coefficients:

$$||\beta||_2 = \sqrt{\beta_1^2 + \beta_2^2 + ... + \beta_p^2}$$

The L2-norm penalty tends to produce smooth solutions, where all coefficients are non-zero but small. This is because the penalty has a "rounded" shape that smooths the coefficients towards zero. Ridge regression uses the L2-norm penalty to fit a model that achieves both prediction accuracy and stabilization of the regression coefficients.

On the other hand, the L1-norm penalty is given by:

$$\lambda||\beta||_1$$

where λ is a hyperparameter that controls the strength of the penalty and $||.||_1$ denotes the L1-norm, which is the sum of the absolute values of the coefficients:

$$||\beta||_1 = |\beta_1| + |\beta_2| + ... + |\beta_p|$$

The L1-norm penalty tends to produce sparse solutions, where some coefficients are exactly zero and some are non-zero. This is because the penalty has "sharp corners" at the coordinate axes, so it tends to push the coefficients to lie on the axes. Lasso regression uses the L1-norm penalty to fit a model that achieves both prediction accuracy and variable selection.
In summary, Lasso and Ridge regression use different penalty functions based on the L1-norm and L2-norm of the regression coefficients, respectively. These penalties encourage sparsity and smoothness in the coefficients and provide a way to balance prediction accuracy and model complexity.

3.2 The Tuning Parameter

The tuning parameter is a key component in both Lasso and Ridge regression, as it determines the strength of the penalty term in the objective function. Depending on the value chosen for the tuning parameter, the coefficients will be regularized accordingly to minimize the penalty term. A higher tuning parameter thus means that the coefficients will be shrunk to a higher degree.

It is therefore important to select an "optimal" value of the tuning parameter. What this really means is that we should select a value of the tuning parameter such that the coefficients are shrunk in an optimal way. Cross-validation offers a simple way to tackle this problem. We choose a grid of possible values taken by λ and fit the model with that value and then compare the models using the cross-validation error. The λ used in the model with the smallest said error would be the optimal tuning parameter. There are many ways to carry out this cross-validation method of selecting the tuning parameter but this is not the main goal of the paper and R already does this for us.

3.3 Application of Shrinkage Methods

The R function that allows to fit the Lasso and Ridge methods is *glmnet*. This comes from the *glmnet* package that fits generalized linear and similar models via penalized maximum likelihood. To do so, it uses a special form of Elastic Net regression.

To understand where this comes from, we need to understand what Elastic-Net regression means. Elastic-Net combines the sum of squared residuals, the L1-norm penalty term and the L2-norm penalty term.

$$RSS + \lambda_1 \sum_{j=1}^{p} \beta_j + \lambda_2 \sum_{j=1}^{p} \beta_j^2$$

This allows to overcome the limitations of Ridge and Lasso individually.

In R this is implemented slightly differently. The *glmnet* function simulates Elastic-Net with only one λ parameter but multiplies it with a constant that involves a fixed α parameter. The L1-norm term is multiplied by α, whereas the L2-norm is multiplied by $(1 - \alpha)$. The final version of the term that is minimized is thus :

$$RSS + \lambda(\alpha \times ||\beta||_1 + (1 - \alpha) \times ||\beta||_2)$$

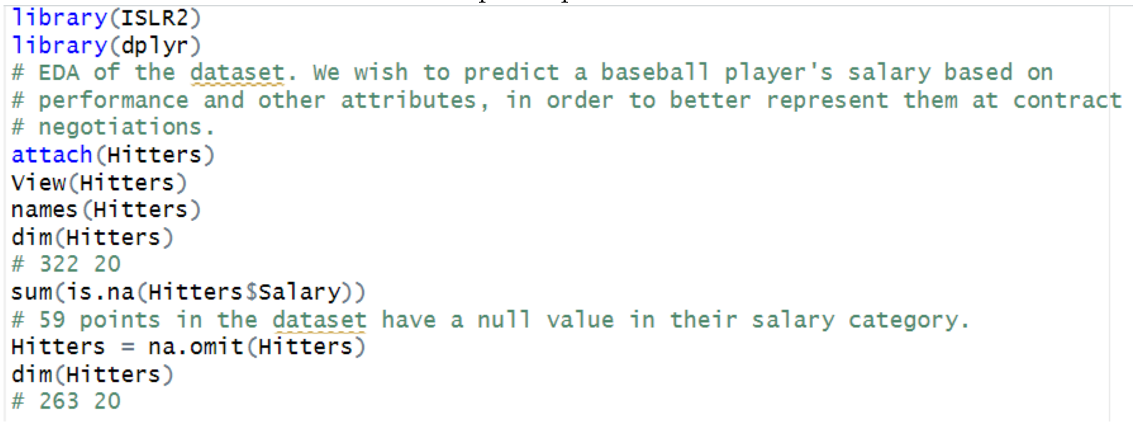
Therefore, simulating the Elastic-Net regression using $\alpha = 0$ amounts to minimizing $RSS + \lambda \times 1||\beta||_2$. In other words, by fixing $\alpha = 0$, we can simulate Ridge regression. Inversely, setting $\alpha = 1$ allows to simulate Lasso regression.

To select optimal values for $\lambda > 0$, we use a special version of *glmnet* which is *cv.glmnet*. This version uses cross-validation to select test different values for λ and select the value for which coefficients are regularized efficiently and optimally.

Our application will use the **Hitters dataset** which is part of ISLR2 package. This dataset reports useful data from the Major League Baseball from the 1986 and 1987 seasons. On the basis of various statistics associated with a player's performance in the previous year, we wish to be able to predict a player's Salary.

3.3.1 Data Cleaning and Splitting

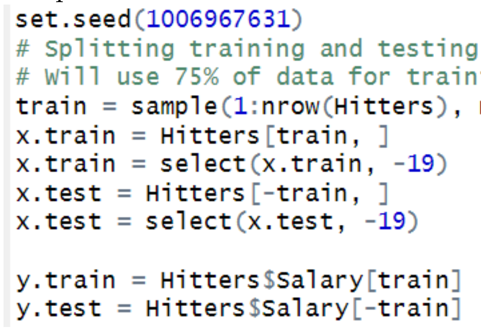
We note that the Salary variable has missing values for certain players. It is important that the Salary variable has no missing values as that is the dependent variable we are trying to predict. The *is.na*() function is useful to identify these rows with missing value. It returns a vector with a Boolean for each observation, where $FALSE$ means the value is not missing, and $TRUE$ means it is missing. Additionally, the *na.omit* function allows to eliminate all the rows from the dataset that have a missing value. Here is a demonstration on how that these simple steps can be achieved:



```r
library(ISLR2)
library(dplyr)
# EDA of the dataset. We wish to predict a baseball player's salary based on
# performance and other attributes, in order to better represent them at contract
# negotiations.
attach(Hitters)
View(Hitters)
names(Hitters)
dim(Hitters)
# 322 20
sum(is.na(Hitters$Salary))
# 59 points in the dataset have a null value in their salary category.
Hitters = na.omit(Hitters)
dim(Hitters)
# 263 20
```

We can see that the final dimensions of the dataset are 20 rows for 263 observations. Note that these 20 predictors will induce a study of multicollinearity between all these predictors. This is where we will use our shrinkage methods in order to study the use of these methods and their effectiveness compared to traditional stepwise subset selection methods.

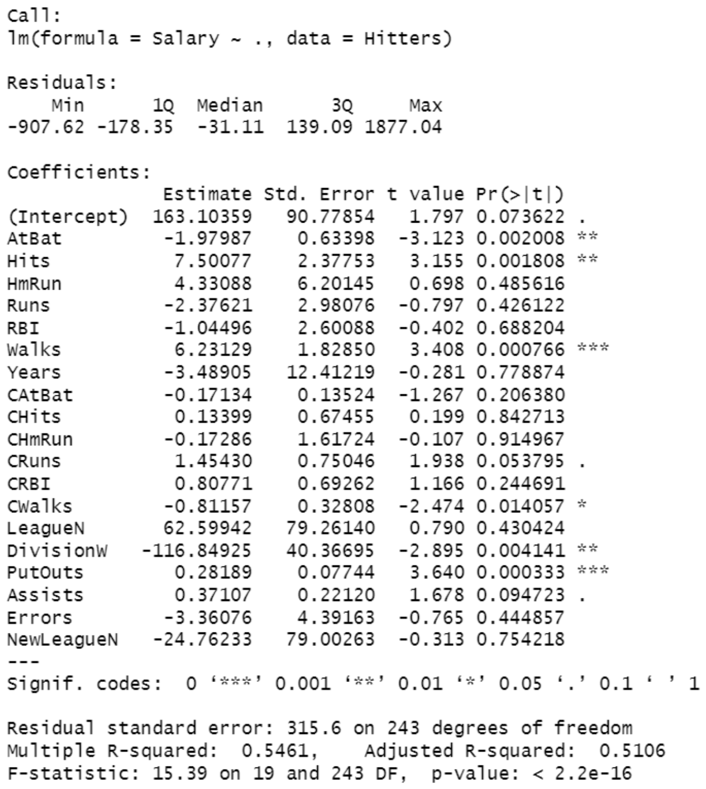
In order to fit our models and test them out, we will be randomly splitting our dataset in a 75:25 ratio. This means that 75% of the dataset will be used to train our model, while the remaining 25% will be used to test it out and obtain a test MSE so we can compare the different models we will be fitting. Here is how to carry this out:



```r
set.seed(1006967631)
# Splitting training and testing datasets.
# Will use 75% of data for training and 25% for testing.
train = sample(1:nrow(Hitters), nrow(Hitters) * 0.75)
x.train = Hitters[train, ]
x.train = select(x.train, -19)
x.test = Hitters[-train, ]
x.test = select(x.test, -19)

y.train = Hitters$Salary[train]
y.test = Hitters$Salary[-train]
```

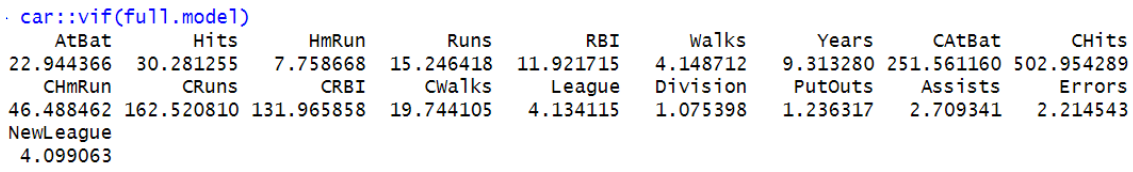
3.3.2 Initial Model

We first fit the full model and obtain the following results.



```
Call:
lm(formula = Salary ~ ., data = Hitters)

Residuals:
    Min      1Q  Median      3Q     Max 
-907.62 -178.35  -31.11  139.09 1877.04 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  163.10359   90.77854   1.797 0.073622 .  
AtBat         -1.97987    0.63398  -3.123 0.002008 ** 
Hits           7.50077    2.37753   3.155 0.001808 ** 
HmRun          4.33088    6.20145   0.698 0.485616    
Runs          -2.37621    2.98076  -0.797 0.426122    
RBI           -1.04496    2.60088  -0.402 0.688204    
Walks          6.23129    1.82850   3.408 0.000766 ***
Years         -3.48905   12.41219  -0.281 0.778874    
CAtBat        -0.17134    0.13524  -1.267 0.206380    
CHits          0.13399    0.67455   0.199 0.842713    
CHmRun        -0.17286    1.61724  -0.107 0.914967    
CRuns          1.45430    0.75046   1.938 0.053795 .  
CRBI           0.80771    0.69262   1.166 0.244691    
CWalks        -0.81157    0.32808  -2.474 0.014057 *  
LeagueN       62.59942   79.26140   0.790 0.430424    
DivisionW   -116.84925   40.36695  -2.895 0.004141 ** 
PutOuts        0.28189    0.07744   3.640 0.000333 ***
Assists        0.37107    0.22120   1.678 0.094723 .  
Errors        -3.36076    4.39163  -0.765 0.444857    
NewLeagueN   -24.76233   79.00263  -0.313 0.754218    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 315.6 on 243 degrees of freedom
Multiple R-squared:  0.5461,	Adjusted R-squared:  0.5106 
F-statistic: 15.39 on 19 and 243 DF,  p-value: < 2.2e-16
```

The high number of predictors unsurprisingly leads to a situation where we have to find a better model: Although the p-value of the F statistic is $< 2.2e-16$, more than two thirds of all 19 predictors have a p-value over the common 0.05 threshold. That means that more than two thirds of the available predictors are not significant. Moreover, the adjusted R-squared is evaluated at 0.5106.



```
car::vif(full.model)
     AtBat       Hits      HmRun       Runs        RBI      Walks      Years     CAtBat      CHits 
 22.944366  30.281255   7.758668  15.246418  11.921715   4.148712   9.313280 251.561160 502.954289 
    CHmRun      CRuns       CRBI     CWalks     League   Division    PutOuts    Assists     Errors 
 46.488462 162.520810 131.965858  19.744105   4.134115   1.075398   1.236317   2.709341   2.214543 
 NewLeague 
  4.099063 
```

Another strong indicator of the multicollinearity in this dataset is shown by the VIF values.

The performance of this model when fitting it on 75% of the data and testing the model on the remaining 25% yields an MSE of 122079.2.

3.3.3 Ridge Model

Let's now finally use these shrinkage methods. After loading the *glmnet* package, we fit the ridge model as follows.

```
ridge.fit = cv.glmnet(data.matrix(x.train), y.train, type.measure = "mse", alpha = 0,
                      family = "gaussian")
```

A few notes here: the *data.matrix*(*x.train*) is used to convert the dataset that contains the predictors data into a matrix as that is the format the *glmnet* function expects. We use a measure of "MSE" and the family of "Gaussian" is a parameter that indicates to the *glmnet* function that we are fitting a linear regression model. If we were fitting a logistic regression for instance, we would have change the family to "deviance".

We can use the predict() function for a number of purposes. One of them is to predict the value of Salary with new data. We can input the testing subset of the dataset. The optimal tuning parameter can be obtained by looking at the *lambda.min* attribute of the object representing the model in R. After obtaining the predictions we can compute the difference from the actual true value that we have in the test dataset. We thus compute use this idea to compute the MSE for the model. This can be done as follows:

```
ridge.predicted = predict(ridge.fit, s = ridge.fit$lambda.min, newx = data.matrix(x.test))
mean((y.test - ridge.predicted) ^ 2)
```

We get the following results: $MSE = 98506.94$

3.3.4 Lasso Model

Similarly, we fit the Lasso model as follows:

```
lasso.fit = cv.glmnet(data.matrix(x.train), y.train, type.measure = "mse", alpha = 1,
                      family = "gaussian")
lasso.predicted = predict(lasso.fit, s=lasso.fit$lambda.min, newx = data.matrix(x.test))
mean((y.test - lasso.predicted) ^ 2)
```

Then we test out the model and obtain an $MSE = 99850.11$. It is worth noting that the Lasso has the advantage of yielding simpler and thus more interpretable models since minimizing the coefficients with respect to the L1-norm can lead to some coefficients being set to 0. Thus, the Lasso usually offers models with less predictors than the Ridge.

In conclusion however, we can see that in terms of performance for our Hitters dataset, the Ridge fit performs much better than the full model and slightly better than the Lasso.

3.3.5 Elastic Net

The *cv.glmnet*() function can set an $0 < \alpha < 1$ which combines the L1-norm and L2-norm penalty in an optimal way than Ridge and Lasso do on their own individually.

To do so, we can use a for loop that will iteratively fit the model using a value of α of the form $\frac{i}{10}$ where i goes from 1 to 10. Each model will then be tested on the test dataset and we will record its MSE so we can compare them.

My implementation makes the use of arrays to store each model and a data frame that we re-bind at each iteration in order to store the resulting mean square error of that model.

```r
list.of.models = list()
for (i in 1:10) {
  model.name = paste0("alpha", i/10)
  list.of.models[[model.name]] = cv.glmnet(data.matrix(x.train), y.train,
                                           type.measure = "mse", alpha = i/10,
                                           family = "gaussian")

}

result = data.frame()
for (i in 1:10) {
  model.name = paste0("alpha", i/10)
  model.predicted = predict(list.of.models[[model.name]],
                            s = list.of.models[[model.name]]$lambda.min,
                            newx = data.matrix(x.test))

  mse = mean((y.test - model.predicted) ^ 2)
  temp = data.frame(alpha = i/10, mse = mse, model.name = model.name)
  result = rbind(result, temp)
}
```

The results for the 10 generated models are the following:

```
> result
   alpha       mse model.name
1    0.1 105989.60   alpha0.1
2    0.2 107055.47   alpha0.2
3    0.3 102159.09   alpha0.3
4    0.4 105838.21   alpha0.4
5    0.5 105526.76   alpha0.5
6    0.6 105819.58   alpha0.6
7    0.7 112559.72   alpha0.7
8    0.8 115009.77   alpha0.8
9    0.9  97394.68   alpha0.9
10   1.0 109219.31     alpha1
```

The yielded best result thus uses $\alpha = 0.9$ and it performs better than both Ridge and Lasso.

4 Discussion

Since they can improve the performance of linear regression models by resolving the issues of multicollinearity and overfitting, shrinking approaches like Ridge and Lasso regression have gained popularity in data science. These techniques work by including a penalty term in the regression model's objective function that encourages smaller regression coefficient values. The magnitude of the coefficients is effectively reduced by this penalty term, therefore the name "shrinking methods."

The advantages of shrinking methods include their computational feasibility and ability to improve model accuracy by reducing overfitting. However, these methods can lead to higher complexity models and may be less interpretable than linear regression models that are obtained via using more straightforward methods such as stepwise model selection.

Lasso is seen to be able to provide the best of both words as it is a shrinking method that can set coefficients to 0 and can as such be viewed as a variable selection method too. It has the nice property of just getting rid of predictors that are totally useless where as Ridge does not eliminate any predictor. However, although it does often perform better than Ridge and offers more interpretable models, that is not always the case as we have seen with our study where the Ridge model outperformed the Lasso one.

Elastic Net is an even newer method which combines both L1 and L2 penalties than either of them does individually. However, it involves testing out a certain number of values for its tuning parameter α which can increase the computational complexity of this method. It does however offer a secondary alternative that these recent methods have enabled.

As a final note, careful interpretation of the results is necessary to assess the trade-offs between model complexity and interpretability. It is important to use these methods with caution and carefully interpret the results to make sound data-driven decisions.

5 References

- **An Introduction to Statistical Learning with Applications in R** : Second Edition by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani
- **StatQuest with Josh Starmer** : Youtube Channel.
- **The Elements of Statistical Learning**: Second Edition by Trevor Hastie, Robert Tibshirani, Jerome Friedman