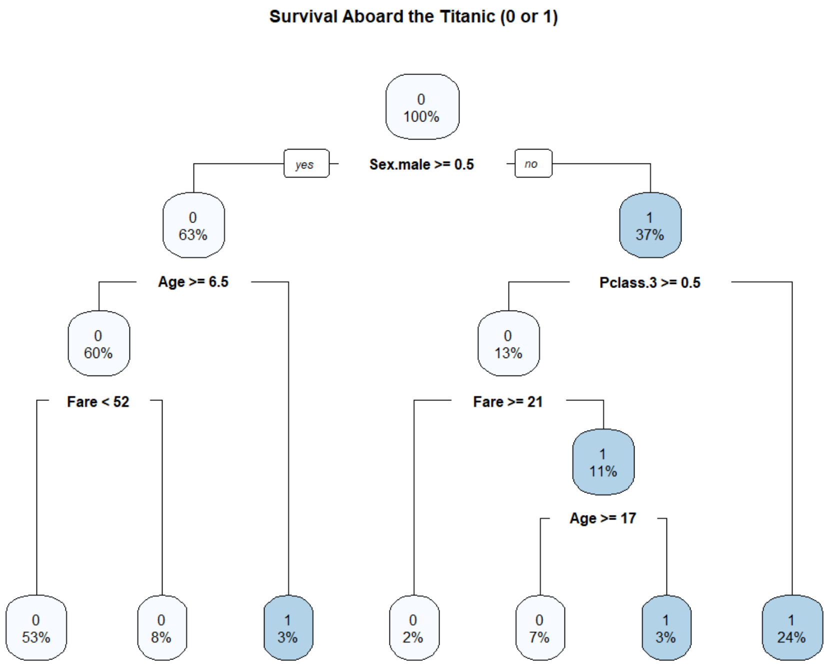
University of Essex
Department of Mathematical Sciences

MA831: Capstone Project Dissertation

Automating the Analysis Process: automodel

Robbie Mowforth, rm18379



Supervisor: **Dr. Yanchun Bao**

April 15, 2022
Colchester

Background

Whilst working on this dissertation, the goal has changed quite a few times. Firstly, this was a project on investigating methods that deal with data-sets that include missing values. After some discussion with my supervisor she mentioned the opportunity to model a larger data-set that was on the proteins found within blood samples. I decided to take on this opportunity, however, this data-set didn't have a public release when I first stated. To prepare for it's release, I did analysis on a similar data-set which I was planning to replicate.

As this analysis went on, I found myself enjoying the challenge of trying to generalize the methods I was using. I also came to learn that due to some unfortunate circumstances, the larger data-set on proteins within blood samples wouldn't be available for public release until after my dissertation. Due to these circumstances I decided to change the angle of my project and turned it into an attempt to automate the analysis process, hence the title: **Automating the Analysis Process: automodel**.

I have had an amazing time making, working and writing on this topic which naturally evolved from my given circumstances. I am hoping to continue working with the results from this dissertation beyond it's finish date and hope to have a better version in the years to come.

I would like to thank the following for their support during this dissertation:

- My supervisor Dr. Yanchun Bao for the mass amount of support and mathematical prowess
- Prof. Meena Kumari for her wealth of knowledge in the Bio-Social field
- Anna Dearman for her technical skills, know-how of Bio-Social data and proof reading
- My mother for helping me improve the words I've put on the page
- The reader for attempting to understand the ramblings I've written on the pages below

Contents

1 Introduction

This dissertation covers the learning of new analysis techniques at a technical/-mathematical level and the application of them within the statistical coding language R.

1.1 Description

The result of this dissertation is a package called **automodel**. **automodel** aims to provide an introduction to the predictive/modelling power within a given data-set. **automodel** provides a range of different models, results and statistics that can be used to inform any further in-depth analysis rather than being considered a final result.

The recommended workflow when using **automodel** is to run the main function, **autoModel**, using the desired dependant variable within a data-set. The results of this run should then be used to inform any further in-depth analysis instead of a final result. Lets say we have a data-set which has a variable we are interested in modelling/predicting. We can run **autoModel** without needing to have any context on the data-set and generate some initial models we can analyze. From here we can use the results of these models to inform the next best steps for modelling/predicting our variable of interest.

autoModel take two inputs:

- The name of the variable which the user wishes to predict (the dependant variable)
- The data-set which includes all the variables to be used when modelling, including the dependant variable

An example call to this function would be

```
autoModel("dependant variable", data)
```

The result of calling this function is a:

- cleaned, reduced & transformed version of the data-set inputted
- K-Means cluster model with predictions
- kNN cluster model with training data
- CART model with predictions
- Refined Ordinary Linear Regression model with predictions
- Best Alpha Elastic Net Regression model with predictions

1.2 Example

The data-set **titanicData** includes data on the passengers of the Titanic. Each observation of the data-set includes information on the individual passengers (**Age**, **Name**) and how they interacted with the Titanic (**Ticket**, **Fare**). In this data-set, we have the variable **Survived**, which is a binary variable where 1 means the passenger survived and 0 means the passenger didn't survive. We are going to use **automodel** to enable us to predict **Survived** using all other variables in **titanicData**.

To use **automodel** in R, we run the following command:

```
titanicResults = autoModel("Survived", titanicData)
```

This command creates all of our results and stores them in a variable called **titanicResults**. Let's evaluate some of the results within this variable.

Below are the predictions made by the kNN model. To get our results table, we execute the following in R

```
titanicResults$kNNResults$predictions
```

Table 1 is the results table from running the above shows what the model has predicted for **Survived** (if a passenger aboard the Titanic survived or not) per passenger.

	predicted	real	freq
1	0	0	86
2	1	0	27
3	0	1	23
4	1	1	41

Table 1: kNN Example Predictions

- The model predicted a passenger didn't survive (**Survived** = 0) correctly 86 times (see row 1)
- The model predicted a passenger didn't survive (**Survived** = 0) incorrectly 23 times (see row 3)
- The model predicted a passenger did survive (**Survived** = 1) correctly 41 times (see row 4)
- The model predicted a passenger did survive (**Survived** = 1) incorrectly 27 times (see row 2)
- In total: The model predicted correctly 127 times meaning the model was correct for 72% of the predictions
- In total: the model predicted in-correctly 50 times meaning the model was in-correct for 28% of the predictions

Next, let's evaluate the predictions made by the Linear Regression model. To get our results, we execute the following in R

```
summary(titanicResults$linearRegression$model)
```

The results from the model show what variables were considered significant in predicting **Survived**.

```
Residuals:
     Min       1Q   Median       3Q      Max
-1.14503 -0.26114 -0.06787  0.23964  0.99263

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.97842    0.03987  24.539  < 2e-16 ***
Pclass.2    -0.21527    0.04990  -4.314 1.91e-05 ***
Pclass.3    -0.40917    0.04592  -8.910  < 2e-16 ***
Sex.male    -0.46835    0.03616 -12.953  < 2e-16 ***
Age         -0.08749    0.01987  -4.402 1.30e-05 ***
SibSp.4     -0.22467    0.10165  -2.210   0.0275 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3929 on 524 degrees of freedom
Multiple R-squared:  0.3745,    Adjusted R-squared:  0.3685
F-statistic: 62.73 on 5 and 524 DF,  p-value: < 2.2e-16
```

We can see that out of all the variables given to **autoModel**, it has deemed the following variables significant in predicting/modelling **Survived**:

- the binary variable of **Pclass.2**, describes if the passenger had a ticket of class 2 or not

- the binary variable of **Pclass.3**, describes if the passenger had a ticket of class 3 or not

- the binary variable **Sex.male**, describes if the passenger was male or female

- **Age**, the age of the passenger

- the binary variable **SibSp.4**, describes if the passenger had a 4 siblings/ spouses aboard the Titanic or not

Why these variables might be predictive is beyond the scope of the **automodel** package, but they certainly throw up some interesting questions for those using the **titanicData** to investigate further. This is the overall goal of the **automodel** package, to hopefully provide the user with interesting insights on their data-set which they can go on to investigate further.

1.3 Video Tutorial

A video tutorial of the **automodel** package is available at the following link: https://www.youtube.com/watch?v=uL2BBJ9gBGM

2 Data

To demonstrate the results generated from using the **automodel** package, we are going to parse multiple different data-set though the **autoModel** function. In this section we will cover the context behind each data-set used and give a brief preliminary analysis on each them.

2.1 Understanding Society & GHQ

Understanding Society (also known as The UK Household Longitudinal Study or UKHLS) is a study that measures a large amount of variables based on a participants well-being, lifestyle & life choices. It aims to have a varied pool of participants from all different backgrounds so that there's a fair representation of the population. "The sample is large enough to have around 10,000 people for each birth cohort per decade from the 1940s on-wards. They also have approximately 17,000 children who have been born into the Study since the year 2000" [1].

We are interested in using the Main Survey and Nurse Visit survey from Understanding Society. The surveys measurements are taken in "waves", where each UKHLS wave takes around two years to be fully processed across all participants. Even though the waves cross-over, each participant is interviewed roughly 1 year apart. Wave 2 was started in the year 2010 and wave 3 was started in the year 2011 with the study of both ranging over the time period 2010-2013 [2]. randomSeed To demonstrate the usage of **automodel** package, we will attempt to predict the variable **scghq1_dv**, which is a total score of a participant gathered from a survey called GHQ (more information can be found on the GHQ in Section 2.1.3).

2.1.1 Data Breakdown

These data-sets are quite large and therefore take a long time to be fully processed when using **automodel**. In Table 2 is a breakdown of each of the data-sets and their exact dimensions. All data-sets used were in *.tab* format. Table 2 describes the number of variables and observations in each data-set, if they include data on the GHQ and what wave the data is measured from.

Data-set	No. Obs	No. Vars	Contains GHQ?	Wave
b_indresp	54569	1652	TRUE	2
b_income	82111	23	FALSE	2
c_indresp	49692	3058	TRUE	3
c_income	76099	23	FALSE	3
xindresp_ns	20699	355	TRUE	2 & 3
xlabblood_ns	13258	35	FALSE	2 & 3

Table 2: Initial Understanding Society data-sets

- **b_indresp**: Main survey data from wave 2 of the UKHLS

- **b_income**: Survey data based around a participants income in wave 2. Participants could choose not to answer

- **c_indresp**: Main survey data from wave 3 of the UKHLS

- **c_income**: Survey data based around a participants income in wave 3. Participants could choose not to answer these questions

- **xindresp_ns**: Data from a nurse visiting the participants in waves 2 & 3. The nurses were to visit each participant once in both waves 2 & 3. Not all participants in the Nurse Visit were included in the Main Survey.

- **xlabblood_ns**: This is the results from the blood samples taken from the nurses during the nurse visits. Participants could choose to not have their blood sampled meaning there's significantly less observations compared to **xindresp_ns**

2.1.2 Initial Data Investigation

Understanding Society data encodes missing data into negative integers based on the reason why it's missing:

-9: missing, data is purely missing

-8: inapplicable, answering a previous question meant they didn't need to answer some following questions

-7: proxy respondent, someone else answered the question on behalf of the participant

-2: refused, refused to answer the question

-1: don't know, no clear reason on why data is missing

When we use this data with **automodel**, we are going to treat all missing values as the same regardless of context. This means we encode -9, -8, -7, -2, -1 as NA within R (See Section 3.7.1)

To preform merges/joins between these 6 data-sets, we use two variables:

- **pidp**: The unique identifier used to recognize each individual participant. Takes the form of a random integer value.

- **wave**: a column initially found in **xindresp_ns** & **xlabblood_ns**, it states the wave which the data is from, example: 2. For a join between **xindresp_ns** & **xlabblood_ns** to be possible with the other data-sets, we need to state the wave the data is from, therefore we manually add the variable wave to **b_indresp**, **b_income**, **c_indresp**, **c_income**. This process is explained further in Section 4.2.

As expected, the survey per wave have different sets of variables/measurements. In total there are:

- 1186 variables both measured in **b_indresp** & **c_indresp**
- 466 variables that are in **b_indresp** and not in **c_indresp**
- 1872 variables that are in **c_indresp** and not in **b_indresp**

Due to these differences, when analyzing **b_indresp** & **c_indresp** in the same data-set, we will only use the 1186 shared variables. More about this shared data-set will be explained in Section 4.1.

2.1.3 GHQ Preliminary Analysis

The General Health Questionnaire (GHQ) was authored by David Goldberg in 1978. "The GHQ focuses on the client's ability to carry out 'normal' functions and the appearance of any new disturbing phenomena." [3]. In more lay-mans terms, the GHQ is designed to measure the psychological distress of a participant. The version used within the Understanding Society survey is the GHQ-12, which consists of 12 questions. The GHQ-12 is descried as a "A quick screener for survey use" [3] and therefore the results should not be used to diagnose mental health. For each question, 1 of 4 responses can be chosen (with each response garnering a score between 1 to 4). The scores from each question are them summed up to create the GHQ score (**scghq1_dv**) of a participant (in the GHQ-12, the minimum GHQ score is 0 and the maximum is 36) The name of each of the questions in the GHQ-12 have the following format:

- sc: This stand for *Self Completion* and means that the questions were filled out my the participant themselves
- ghq: Represents that these questions are from the GHQ-12 used in the Understanding Society questionnaire
- a-l: The letter at the end of the question denotes each individual question used within the GHQ-12.

Knowing the above, the questions used in the GHQ-12 and their responses are:

scghqa: *Concentration*, Have you recently been able to concentrate on whatever you're doing?

- better than usual, score: 0
- same as usual, score: 1
- less than usual, score: 2
- much less than usual, score: 3

scghqb: *Loss of sleep*, Have you recently lost much sleep over worry?

- not at all, score: 0
- no more than usual, score: 1
- rather more than usual, score: 2
- much more than usual, score: 3

scghqc: *Playing a useful role*, Have you recently felt that you were playing a useful part in things?

- more than usual, score: 0
- same as usual, score: 1
- less so than usual, score: 2
- much less than usual, score: 3

scghqd: *Capable of making decisions*, Have you recently felt capable of making decisions about things?

- more so than usual, score: 0
- same as usual, score: 1
- less so than usual, score: 2
- much less capable, score: 3

scghqe: *Constantly under strain*, Have you recently felt constantly under strain?

- not at all, score: 0
- no more than usual, score: 1
- rather more than usual, score: 2
- much more than usual, score: 3

scghqf: *Problem overcoming difficulties*, Have you recently felt you couldn't overcome your difficulties?

- not at all, score: 0
- no more than usual, score: 1
- rather more than usual, score: 2
- much more than usual, score: 3

scghqg: *Enjoy day-to-day activities*, Have you recently been able to enjoy your normal day-to-day activities?

- more than usual, score: 0
- same as usual, score: 1
- less so than usual, score: 2
- much less than usual, score: 3

scghqh: *Ability to face problems*, Have you recently been able to face up to problems?

- more so than usual, score: 0
- same as usual, score: 1
- less so than usual, score: 2
- much less able, score: 3

scghqi: *Unhappy or depressed*, Have you recently been feeling unhappy or depressed?

- not at all, score: 0
- no more than usual, score: 1
- rather more than usual, score: 2
- much more than usual, score: 3

scghqj: *Losing confidence*, Have you recently been losing confidence in yourself?

- not at all, score: 0
- no more than usual, score: 1
- rather more than usual, score: 2
- much more than usual, score: 3

scghqk: *Believe in self-worth*, Have you recently been thinking of yourself as a worthless person?

- not at all, score: 0
- no more than usual, score: 1
- rather more than usual, score: 2
- much more than usual, score: 3

scghql: *General happiness*, Have you recently been feeling reasonably happy, all things considered?

- more so than usual, score: 0
- same as usual, score: 1
- less so than usual, score: 2
- much less than usual, score: 3

Since GHQ is present in multiple data-sets, we should preform some checks on the differences in distribution across the data-sets we are loading in. Please see Figure 1 for the density plots of each data-set that includes GHQ data. We can see that even though there is noticeable differences between the distributions, the GHQ score (*scghq1_dv*) falls within a similar distribution for all data-sets.

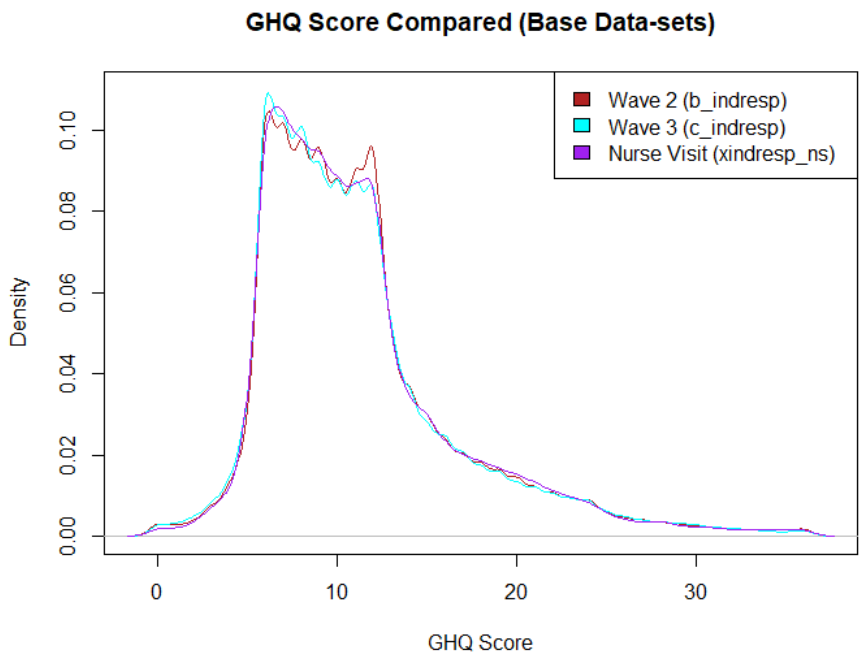


Figure 1: Density of base data-sets

Now lets do some two sided t.tests and two sided f.tests on the GHQ score from each data-set. The below test were conducted in the coding language R. The variable names used are described in Section 4.1, for now, use the titles given before each test result as a guide.

Firstly, tests for Wave 2 (**b_indresp**) and Wave 3 (**c_indresp**)

```
        Welch Two Sample t-test

data:  w2indresp$scghq1_dv and w3indresp$scghq1_dv
t = 3.3084, df = 83601, p-value = 0.0009388
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.05135789 0.20066465
sample estimates:
mean of x mean of y
 11.20125  11.07524

        F test to compare two variances

data:  w2indresp$scghq1_dv and w3indresp$scghq1_dv
F = 0.99797, num df = 43422, denom df = 40575, p-value = 0.8349
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.979047 1.017249
sample estimates:
ratio of variances
         0.9979685
```

Secondly, tests for Wave 2 (**b_indresp**) and Nurse Visit (**xindresp_ns**)

```
        Welch Two Sample t-test

data:  w2indresp$scghq1_dv and mixNurse$scghq1_dv
t = 0.30387, df = 36019, p-value = 0.7612
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.07931319  0.10841751
sample estimates:
mean of x mean of y
 11.20125  11.18670

        F test to compare two variances

data:  w2indresp$scghq1_dv and mixNurse$scghq1_dv
F = 1.0125, num df = 43422, denom df = 18842, p-value = 0.3166
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.9882163 1.0371829
sample estimates:
ratio of variances
          1.012463
```

Lastly, tests for Wave 3 (**c_indresp**) and Nurse Visit (**xindresp_ns**)

```
        Welch Two Sample t-test

data:  w3indresp$scghq1_dv and mixNurse$scghq1_dv
t = -2.3021, df = 36968, p-value = 0.02133
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.20635509 -0.01656312
sample estimates:
mean of x mean of y
 11.07524  11.18670

        F test to compare two variances

data:  w3indresp$scghq1_dv and mixNurse$scghq1_dv
F = 1.0145, num df = 40575, denom df = 18842, p-value = 0.2485
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.9899783 1.0395630
sample estimates:
ratio of variances
          1.014524
```

Due to the large number of observations within each data-set (degrees of freedom), the tests are rather sensitive to differences between values. This can be seen above as the t.test between Wave 2 and Wave 3 (**b_indresp** & **c_indresp**) rejects the null hypothesis at the 95% confidence interval (p-value of 0.0009388) when the mean of Wave 2 is 11.20125 and the mean of Wave 3 is 11.07524 (a difference of 0.126). The t.test between Wave 3 and Nurse Visit (**c_indresp** & **xindresp_ns**) also rejected the null hypothesis at the 95% confidence interval (p-value 0.02133) when the mean of Wave 3 is 11.07524 and the mean of Nurse Visit is 11.18670 (a difference of 0.111). Even though the true differences between these means are statistically significant, the magnitude of the difference is rather small given the context of the variable.

All of tests preformed failed the reject the null hypothesis. Considering the conditions of the rejections found in the previous tests mentioned, it's safe to assume that the differences between the means and variances of the GHQ score per data-set is negligible.

Now let's have a look at the summaries of the GHQ score (*scghq1_dv*) and the individual questions (**scghqa - scghql**). These results can be seen in Table 3, Table 4 and Table 5. Given the above investigation into the distributions, we are only going to look at the Wave 2 (**b_indresp**) summaries of the variables. The same summaries on all data-sets can be found Section 10.

scghq1_dv	
Min	0
1st Q	7
Median	10
Mode	6
Mean	11.2
3rd Q	13
Max	36
NA's	11146
Non-NA's	43423

Table 3: GHQ score Summary

Names	scghqa	scghqb	scghqc	scghqd	scghqe	scghqf
Min.	1	1	1	1	1	1
1st Q	2	1	2	2	1	1
Median	2	2	2	2	2	2
Mean	2.147	1.876	2.082	2.03	1.997	1.789
3rd Q	2	2	2	2	2	2
Max.	4	4	4	4	4	4

Table 4: GHQ Questions a - f Summary

Names	scghqg	scghqh	scghqi	scghqj	scghqk	scghql
Min.	1	1	1	1	1	1
1st Q	2	2	1	1	1	2
Median	2	2	2	2	1	2
Mean	2.159	2.063	1.844	1.728	1.445	2.056
3rd Q	2	2	2	2	2	2
Max.	4	4	4	4	4	4

Table 5: GHQ Questions g - l Summary

Secondly, we can have a look at Figure 2 for the histogram of the scghq1_dv variable to understand the distribution. We can see that we have a rather positive skew distribution in the GHQ score. The most frequent GHQ scores are within the 6 - 12 range.

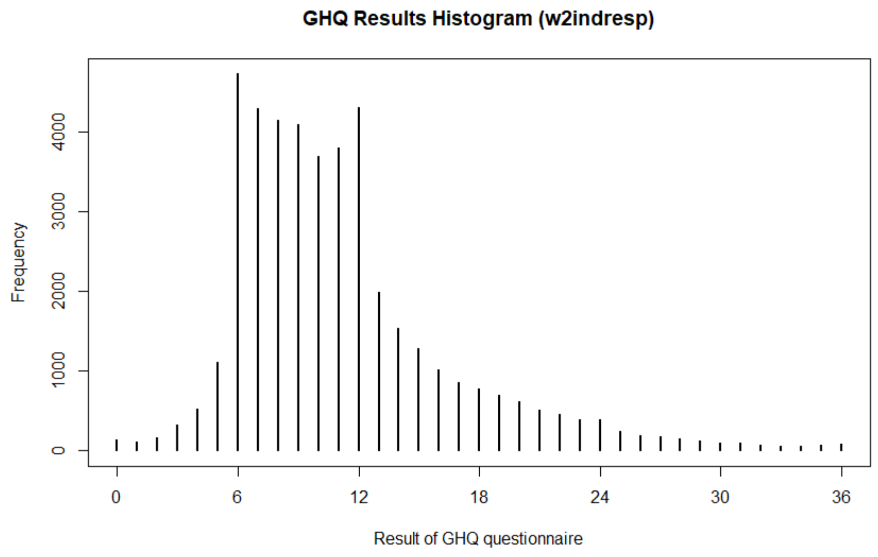


Figure 2: Distribution Plot of Wave 2

Thirdly, we will have a look at Figure 3 for the qqPlot of the GHQ score. We can see that the GHQ score doesn't fit the normal distribution and has a rather large right tail [4].

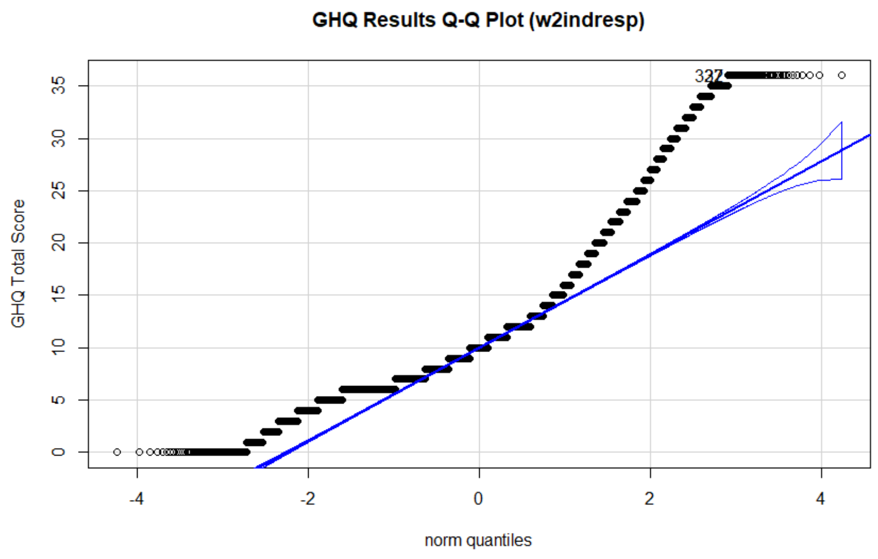


Figure 3: qqPlot of Wave 2 (Normality)

Lastly, we will have a look at Figure 4 for the correlation matrix between all of individual GHQ-12 questions. From the results we can see all that all the correlation coefficients calculated are less than 0.75 meaning there's good reason to assume the questions we have in the GHQ garner varied responses from different participants.

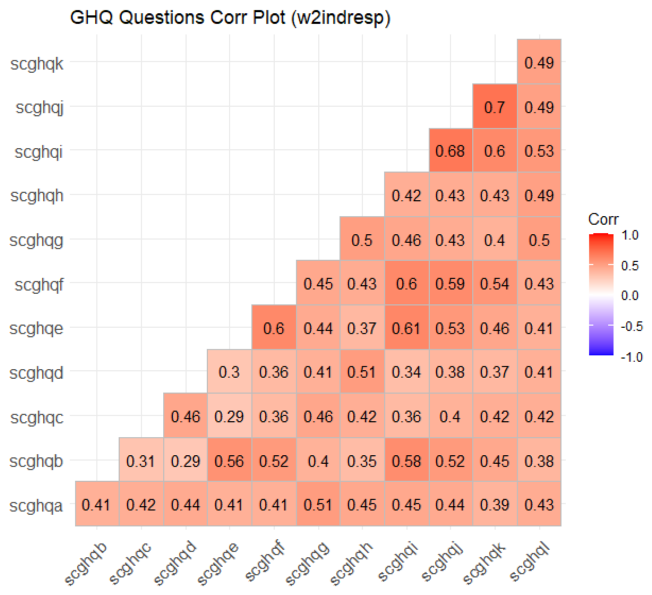


Figure 4: correlations between questions in Wave 2

Let's have a look at some of the most correlated questions:

- ghqe (Constantly under strain) & ghqf (Problem overcoming difficulties): coef of 0.6
- ghqe (Constantly under strain) & ghqi (Unhappy or depressed): coef of 0.61
- ghqf (Problem overcoming difficulties) & ghqi (Unhappy or depressed): coef of 0.6
- ghqi (Unhappy or depressed) & ghqj (Losing confidence): coef of 0.68
- ghqj (Losing confidence)& ghqk (Believe in self-worth): coef of 0.7

This finishes the GHQ preliminary analysis, we will continue the analysis of the Understanding Society Data in Section 4.

2.2 Kaggle Titanic

The Titanic data comes from a Kaggle Machine Learning Competition where participants should create a model which predicts weather a passenger survived or not [5]. This competition is considered as an entry point for those new to machine learning and Kaggle. The data-set provided by Kaggle has already been split into a train and test set. In this dissertation we will be running this data though our function and submitting our results onto the Kaggle leader-board to compare the results.

The Titanic **train.csv** data-set contains 891 observations of 12 variables. Here's a breakdown of what's included:

- **PassengerId**: A unique identifier for each passenger
- **Survived**: Boolean variable, 1 = Survived, 0 = Not Survived. This is our dependant variable, the variable we are looking to predict
- **Pclass**: A passengers ticket class on the titanic, is either 1,2 or 3 (1st, 2nd or 3rd class respectively).
- **Name**: A passengers full name
- **Sex**: Boolean variable, represents the sex of a passenger (either 'male' or 'female').
- **Age**: represents a passengers age in years
- **SibSp**: number of siblings / spouses aboard the titanic
- **Parch**: number of parents / children aboard the titanic
- **Ticket**: The unique Ticket number
- **Fare**: How much was paid by the passenger to board
- **Cabin**: the cabin number the passenger stayed in
- **Embarked**: The port of embarkation (where the passenger boarded the Titanic). C = Cherbourg, Q = Queenstown, S = Southampton.

In the data-set, we are going to be attempting to model the variable **Survived**. In Table 6, we can see a breakdown of the variable **Survived**. We can see that the majority of people didn't survive the titanic, 61% didn't survive and 39% didn't.

Survived	
0 (Didn't Survived)	1 (Survived)
549	342

Table 6: Breakdown Of **Survived** Variable

Overall this data-set provides a good amount of observations and variables to generate initial models with **automodel**. The run of this data-set using **automodel** will be documented in Section 5.

2.3 Edgar Anderson's Iris Data

The Edgar Anderson's Iris data-set is in-built into R as a data-set to be used for initial learning/testing. The data-set "gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris"[6]. The data-set is commonly used to demonstrate clustering and classification algorithms due to the nature of the 3 iris species. In this dissertation, we will be modelling the variable **Species**/ treating **Species** as our dependant variable.

The Iris data-set contains 150 observations of 5 variables. Here's a breakdown of whats included:

- **Sepal.Length**: The length of the flowers sepal (the green leaf that covers the flower before bloom) in centimeters
- **Sepal.Width**: The width of the flowers sepal in centimeters
- **Petal.Length**: The length of the petals from the flower
- **Petal.Width**: The width of the petals from the flower
- **Species**: The species of Iris flower

Next, the summary statistics for the Iris data variables are in Table 7 & Table 8. The below shows that there's a decent distribution across the continuous variables and the categorical variable, **Species**, is equally distributed.

Name	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Min	4.3	2	1	0.1
1st Q	5.1	2.8	1.6	0.3
Median	5.8	3	4.35	1.3
Mean	5.843	3.057	3.758	1.199
3rd Q	6.4	3.3	5.1	1.8
Max	7.9	4.4	6.9	2.5

Table 7: Continuous Variable Summaries

Species	Count
Setosa	50
Versicolor	50
Virginica	50

Table 8: Categorical Variable Summaries

A very neat graphical summary of the data can be seen in Figure 5 [7]. In Figure 5 we can see the scatter-plots of all the continuous variables included in Iris coloured by **Species** (black is *setosa*, red is *versicolor*, green is *virginica*). The plot shows that the Iris data-set seems to have some nice clustering within it's data and therefore would benefit from a clustering model.

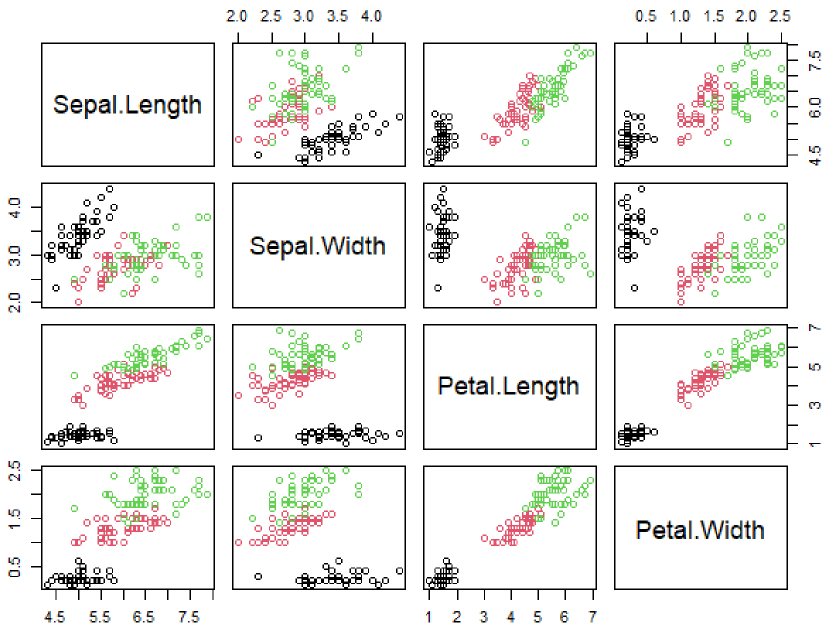


Figure 5: Iris Data-set Plotted

Since we have a small amount of variables, it's worth us doing a brief correlation check on the variables. Figure 6 shows the correlation matrix between all the continuous variables within the Iris data-set. We can see that there is a high amount of correlation between the variables, suggesting a regression model will be a poor fit due to high multicollinearity.

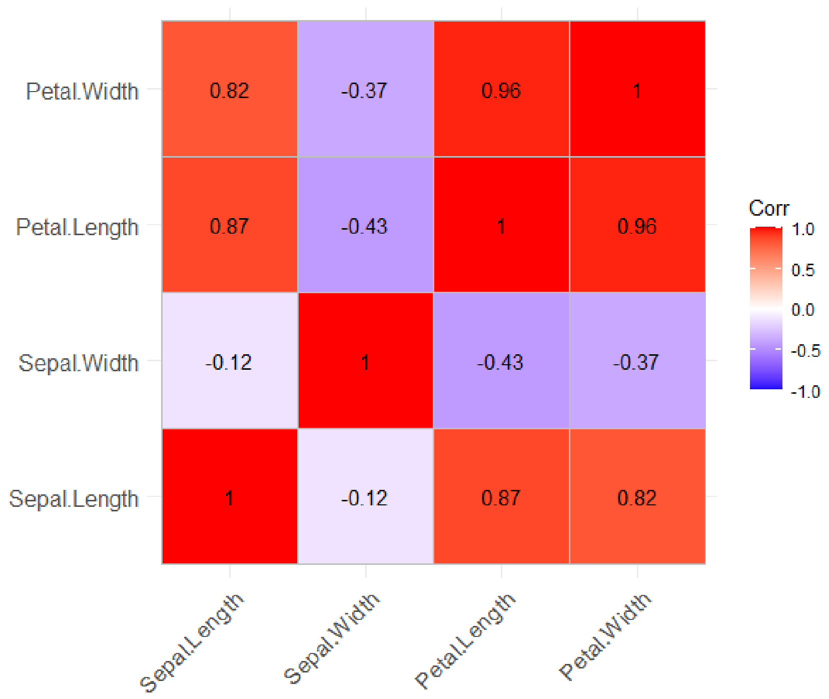


Figure 6: Iris Correlation Matrix

Overall, though our initial analysis, the Iris data looks to be a data-set where some clusters can be easily identifiable, therefore it's a good test for **automodel** to see if it picks up these clusters efficiently. We will continue the usage of the Iris data in Section 6.

3 Methodology

In the **automodel** package, we have various different methods & functions. In this Section we will be covering the details behind each R Package, method and function used within the **automodel** package.

3.1 R Packages

To produce our results, a fair amount of packages/libraries have been used. In this Section we will credit the creators of the packages/libraries and explain how they have been used in this dissertation.

- **broom**

Made/Maintained by:	Simon Couch & Team
Reference:	https://cran.r-project.org/web/packages/broom/broom.pdf
Usage:	Data cleaning tools. The function *tidy()* is used within **automodel**.

- **car**

Made/Maintained by:	John Fox & Team
Reference:	https://cran.r-project.org/web/packages/car/car.pdf
Usage:	Variance Inflation Factor (VIF) and qqPlots. The function *vif()* is used within **automodel** and the function *qqPlot()* is used within the custom assisting function *ghq_analyze()*

- **caret**

Made/Maintained by:	Max Kuhn & Team
Reference:	https://cran.r-project.org/web/packages/caret/caret.pdf
Usage:	Dummy-tization of variables (creating dummy variables). The function *dummyVars()* is used within **automodel**

- **class**

Made/Maintained by:	Brian Ripley, William Venables
Reference:	https://cran.r-project.org/web/packages/class/class.pdf
Usage:	k Nearest Neighbours (kNN). The function *knn()* is used within *automodel*

- **devtools**

Made/Maintained by:	Jennifer Bryan & Team
Reference:	https://cran.r-project.org/web/packages/devtools/devtools.pdf
Usage:	Local R package management. Used to document, format and install **automodel** as a R package.

- **dplyr**

Made/Maintained by: Hadley Wickham & Team

Reference: https://cran.r-project.org/web/packages/dplyr/dplyr.pdf

Usage: tools for working with data. The function *select()* and *all_of()* is used within **automodel**

- **ggcorrplot**

Made/Maintained by: Alboukadel Kassambara

Reference: https://cran.r-project.org/web/packages/ggcorrplot/ggcorrplot.pdf

Usage: Correlation Matrix plots. The function *ggcorrplot()* is used within the custom assisting function *ghq_analyze()*

- **glmnet**

Made/Maintained by: Trevor Hastie & Team

Reference: https://cran.r-project.org/web/packages/glmnet/glmnet.pdf

Usage: Elastic Net Regression Model. The function *cv.glmnet()* is used within **automodel**

- **Matrix**

Made/Maintained by: Martin Maechler & Team

Reference: https://cran.r-project.org/web/packages/Matrix/Matrix.pdf

Usage: Matrix manipulation. Needed so that we can work with sparse matrices within **automodel**

- **mice**

Made/Maintained by: Stef van Buuren & Team

Reference: https://cran.r-project.org/web/packages/mice/mice.pdf

Usage: Missing value imputation. The main function for imputing is *mice()*, used within **automodel**.

- **readr**

Made/Maintained by: Jennifer Bryan & Team

Reference: https://cran.r-project.org/web/packages/readr/readr.pdf

Usage: Reading .tab files. The function *read_table()* is used to load in the Understanding Society data.

- **rlang**

Made/Maintained by: Lionel Henry & Team

Reference: https://cran.r-project.org/web/packages/rlang/rlang.pdf

Usage: tools for working with data. The function *is_empty()* is used within **automodel**.

- rpart

Made/Maintained by: Terry Therneau, Beth Atkinson, Brian Ripley

Reference: https://cran.r-project.org/web/packages/rpart/rpart.pdf

Usage: Classification and Regression Tree Models (CART). The function *rpart()* is used within *automodel*.

- sqldf

Made/Maintained by: G. Grothendieck

Reference: https://cran.r-project.org/web/packages/sqldf/sqldf.pdf

Usage: Running SQL queries in R. the function *sqldf()* is used to transform the income data from Understanding Society (See Section 4.3 Data Processing)

- stats

Made/Maintained by: R Core Team

Reference: https://stat.ethz.ch/R-manual/R-devel/library/stats/html/00Index.html

Usage: General statistical tools. The following functions are from stats: *AIC*, *coef*, *cor*, *kmeans*, *lm*, *na.omit*, *predict*, *princomp*, *runif*, *sd*.

- stringr

Made/Maintained by: Hadley Wickham & Team

Reference: https://cran.r-project.org/web/packages/stringr/stringr.pdf

Usage: Simple string manipulations. The functions *str_remove()* and *str_replace()* are used within **automodel**.

3.2 Cleaning Functions

The first process we need to preform to our data-sets is cleaning. This process aims to put the data-set into a readable format for the transformation process (See Section 3.3 Transforming).

3.2.1 Data-frame Checks

Sometimes the data parsed to the function might not be in the right format for analysis. In this method we make sure that the data is interpreted correctly by casting the entire data set to a data.frame as setting blank data cells as 'NA'. This is achieved though the following two lines.

```
data = as.data.frame(data)
data[data == ""] = NA
```

3.2.2 Data Type Checks

When a variable contains any other data type that numeric, we can't analyze the variable therefore we have this check to remove/change non-numeric data. This method works by detecting which variables are non-numeric and re-codes them into factor variables. Values are re-coded in numerical order then in alphabetical order, please see Table 9 for an example run. (if 1 = "male", 2 = "female", make sure variables is all numeric or text)

Example Transform	
Original	1345, 32233, "sheep", "kangaroo", "kangaroo", NA
After	1, 2, 4, 3, 3, NA

Table 9: Data Type Check Example

3.2.3 High Missing Data Variables

If a variable has a large proportion of their data missing, this may lead to all of the data being removed if **automodel** preforms List-wise deletion. To avoid this, we remove variables which have a certain proportion or more of their data as NA/missing. This proportion is called **naPercent** and can be custom set on each run (default is 20% or 0.2). This means, by default, this function removes variables which have 20% or more of their data as NA/missing.

Table 10 is an example on a arbitrary data-set with 10 observations, we will be assuming that **naPercent** is it's default value (20%) meaning we will remove variables with 2 or more observations as NA/missing

3.2.4 Missing Value Imputation

In our data-sets, it's most likely that there will be values that are missing. These missing values can make it quite hard to generate model and therefore introduce a wealth of issues as data-sets gets large (more variables and observations).

Original Data					Transformed Data		
VarA	VarB	VarC	VarD	VarE	VarA	VarC	VarE
0	NA	100	NA	”yes”	0	100	”yes”
1	28	101	NA	”yes”	1	101	”yes”
1	NA	102	NA	”yes”	1	102	”yes”
0	27	103	NA	”no”	0	103	”no”
0	NA	104	NA	”no”	0	104	”no”
1	103.2	105	NA	NA	1	105	NA
0	NA	106	NA	”no”	0	106	”no”
1	NA	107	NA	”yes”	1	107	”yes”
1	107.4	108	NA	”yes”	1	108	”yes”
1	NA	109	NA	”yes”	1	109	”yes”

Table 10: High Missing Data Variables

In this paper, we will use Single Imputation to handle missing values within a data-set. In this Dissertation, we use Classification and Regression Trees (CART) as our imputation method. This is because at the stage where we want to impute our missing values, it’s possible for a data-set to have a highly correlated variables which would cause issues with most regression imputation methods. With CART, correlated variables don’t effect the model as much and therefore can be used. We achieve this using the mice package (See Section 3.1) in R. Imputation works by initially assigning a temporary value for each missing observation then refines these values by generating CART models to predict these values [9].

To run an imputation, we use the following code:

```
imp = mice(data,
           seed = randomSeed,
          m=1,
           maxit = 5,
           method = "cart",
           threshold = 2)
```

Here’s an in-depth explanation of the above code:

-
  ```
  imp
  ```

  This is the object that holds the imputed values, the object type is a MICE defined *mids* object.

-
  ```
  mice()
  ```

  The function which runs our missing value imputation on our data

-
  ```
  data
  ```

  The data-set which we wish to impute missing data for

-
  ```
  seed = \textbf{randomSeed}
  ```

This sets the randomness in the function to a fixed random space, makes the results reproducible if wanted. **randomSeed** is a function variable (See Section 3.6).

-
  ```
  m = 1
  ```

  This is what tells us that we are doing single imputation. We only create one iteration of imputed values for the missing values.

-
  ```
  maxit = 5
  ```

  This is the amount of iterations the single imputation goes though. This basically means how many times we create a CART model for each variable. This allows for the missing values to converge on a best fit from the initially assigned temporary values.

-
  ```
  method = "cart"
  ```

  This specifies that we are imputing the missing values using classification and regression trees. See Section 3.4.3 for a further explanation of this imputation method.

-
  ```
  threshold = 2
  ```

  MICE will automatically remove predictor variables based on co-linearity. Since we have checks for co-linearity later on, we set the variable 'threshold' to 2 so that mice doesn't automatically ignore/remove highly correlated variables (since we are using CART as our imputation method, we don't have to worry about co-linearity when imputing)

This function can ran by the user by setting the function variable **impFlag** to TRUE (**impFlag** is FALSE by default since imputation can be a timely process). The process can also be speed up by increasing the **automodel** variable **cartSplit** however this can results in the CART models made being a worse fit for the data-set.

There are some assumptions associated with missing data that are important to know for missing value imputation

- **MCAR**: Missing Completely at Random (MCAR) is when the reason for the missing data is unrelated to the data itself. Example: A sever glitch causes random data entries to go missing or corrupt. As Stef Van Buuren puts it [10]

  > *If the probability of being missing is the same for all cases, then the data are said to be missing completely at random (MCAR). This effectively implies that causes of the missing data are unrelated to the data*

- **MAR**: Missing at Random (MAR) is a more realistic assumption of randomness compared to MCAR and takes on a broad range of possible cases. MAR is when randomness is caused by a feature of the data. Example: when using a pedometer in summer, we may produce more missing values than if used in winter. If we can assume that for both in summer and winter, the data is MCAR, then we can assume the overall data is MAR. As Stef Van Buuren puts it [10]

  > *If the probability of being missing is the same only within groups defined by the observed data, then the data are missing at random (MAR). MAR is a much broader class than MCAR. For example, when placed on a soft surface, a weighing scale may produce more missing values than when placed on a hard surface. Such data are thus not MCAR.*

- **MNAR**: Missing Not at Random (MNAR) is applied if MCAR and MAR don't hold. It means that our reason for why there are missing values is unknown to us. Example: Without our knowledge, the longer the walk with a pedometer, the higher the likelihood of their being missing values. As Stef Van Buuren puts it [10]

  > *If neither MCAR nor MAR holds, then we speak of missing not at random (MNAR). In the literature one can also find the term NMAR (not missing at random) for the same concept. MNAR means that the probability of being missing varies for reasons that are unknown to us. For example, the weighing scale mechanism may wear out over time, producing more missing data as time progresses, but we may fail to note this*

If missing values are handled appropriately, unwanted bias can be introduced in the models created. Example: a group for people refused to answer a survey for a collective/similar reason. If the missing values caused by this aren't handled appropriately, the models created with this data-set could lead to Information Bias, Selection Bias and incorrect interpretations of the models generated [11] [12].

A criticism of using Single Imputation is that it doesn't account for the variability that comes with missing value imputation. The more accepted method for missing values is Multiple Imputation. In this method you impute the missing values multiple times (in terms of the *mice()* function, this means imputing the data m times), build models on each of the imputed data-sets then average/pool the results across all models made. Using Multiple Imputation accounts for the known variability that comes with imputing missing values (results can be quite different per imputation) and therefore allows for better modelling of data. We use Single Imputation in this package instead of Multiple Imputation for 2 reasons:

1. Having to generate & model m imputations can drastically increase the overall run-time in our function (which is already quite high for large data sets).
2. Having Single Imputation is better than not having any Imputation option at all (just List-wise deletion as the only option may leave some analysis a little short of options).

Multiple Imputation is included in Section 7.3 of this dissertation. Even though single imputation is a valid method for missing value imputation, there is good reason to account for the variability in imputations correctly by using multiple imputations.

3.2.5 List-wise Deletion

To be able to model our data, we must make sure that there is no NA values within the analyzed data-set. The simplest way to deal with this is by preforming List-wise Deletion. In simple terms, List-wise Deletion is if a observation has a NA value within any variable, the observation is removed from the data-set. This can be achieved with this very simple function in R:

```
na.omit()
```

One criticism of this method is that it can introduce selection bias. Selection Bias is when the participants in a study differ from the population of interest [12]. In List-wise deletion, we remove a large amount of data based on missing values, if these missing values are assumed to be MAR or MNAR (see Section 3.2.4) we may cause selection bias in our results. Using the Understanding Society data as an example, preforming List-wise deletion on the data will mean we don't consider the participants which refuse to answer certain questions (such as household income). This would cause a bias against a group of participants that don't answer survey questions about their household income.

When used in **automodel**, if variables with a large amount of missing values aren't removed (the method to handle this is detailed in Section 3.2.3), List-wise deletion could remove almost all observations in a data-set.

To show how List-wise deletion works, Table 11 is an example arbitrary data set with 10 observations which we will preform List-wise Deletion on

Original Data		
VarA	VarB	VarC
0	100	"yes"
NA	101	"yes"
1	102	"yes"
0	103	"no"
0	104	"no"
1	NA	NA
0	106	"no"
1	107	"yes"
1	108	"yes"
1	109	"yes"

Transformed Data		
VarA	VarB	VarC
0	100	"yes"
1	102	"yes"
0	103	"no"
0	104	"no"
0	106	"no"
1	107	"yes"
1	108	"yes"
1	109	"yes"

Table 11: List-wise Deletion example

3.2.6 Pair-wise Deletion

When accounting for missing values from data-set, we may have conditions meaning we would only want to remove observations with missing values in a select group of variables. Pair-wise deletion works by deleting observations based on the missing values within a select group of variables within a data-set. The resulting method will result in no missing values in the variables that were within the group and potentially some missing values in the variables not in the group (depends on the observations that are removed). It's worth noting that if the group of variables selected is the entire data-set, the deletion preformed will be the same as the List-wise deletion (See Section 3.2.5).

Table 12 is an example in a arbitrary data set with 10 observations. For our example Pair-Wise Deletion, we only need to remove observations which has NA values for **VarA**.

Original Data		
VarA	VarB	VarC
0	100	”yes”
NA	101	”yes”
NA	NA	”yes”
0	103	”no”
0	NA	”no”
1	NA	NA
0	106	”no”
1	107	”yes”
1	108	NA
1	109	”yes”

Transformed Data		
VarA	VarB	VarC
0	100	”yes”
0	103	”no”
0	NA	”no”
0	NA	NA
0	106	”no”
1	107	”yes”
1	108	NA
1	109	”yes”

Table 12: Pair-wise Deletion Example 1

in **automodel**, we use Pair-wise Deletion on the dependant variable within the data-set. We run this before removing variables which have a large proportion of their data as missing (see Section 3.2.3). Doing so means we can potentially end up with more variables in the overall analysis.

An example of this effect can be seen in Table 13. We do is a Pair-Wise deletion on **VarA**, then we remove variables from the analysis if they have 2 or more missing values. As seen below, if we were to remove variables based on their missing values before preforming Pair-Wise Deletion (removing based on the Original Data), we would of removed the entire data-set.

Original Data		
VarA	VarB	VarC
0	100	”yes”
NA	NA	NA
NA	NA	”yes”
0	103	”no”
0	104	”no”
NA	105	NA
0	106	”no”
1	NA	”yes”
1	108	NA
1	109	NA

Pair-Wise Data		
VarA	VarB	VarC
0	100	”yes”
0	103	”no”
0	104	”no”
0	106	”no”
1	NA	”yes”
1	108	NA
1	109	NA

Variable Selected Data	
VarA	VarB
0	100
0	103
0	104
0	106
1	NA
1	108
1	109

Table 13: Pair-wise Deletion Example 2

3.2.7 Low Level Removal

Some factor variables may have levels (unique values within the variable) which have a low amount of observations. This data wouldn’t be reliable to model with so we remove those levels from the data-set. The amount of observations needed per level is determined by the function variable **obsPerLevel**, default value is 10. Like List-wise Deletion (See Section 3.2.5), if the missing data is assumed MAR or MNAR, the deletion process can cause Selection Bias. For example, say you have a dummy variable (variable that only takes the value 1 or 0) that asks if a participant has a rare condition. Since the condition is rare, you may expect there to only be 4-5 observations of it out of 10,000, if **obsPerLevel** is set to 10, this will remove all observations of the rare condition, meaning we will have bias against those who have this condition.

As an example, Table 14 is an arbitrary data-set of 10 observations. We are removing levels where there is only 1 observation. In our example:

- **VarA**: Factor variable with the levels 0,1 & 2
- **VarB**: Continuous variable, no defined levels
- **VarC**: Factor variable with the levels "yes", "no" & "maybe"

Original Data		
VarA	VarB	VarC
0	100	"yes"
1	101	"yes"
1	102	"yes"
0	103	"no"
0	104	"no"
1	105	"maybe"
0	106	"no"
1	107	"yes"
1	108	"yes"
2	109	"yes"

Transformed Data		
VarA	VarB	VarC
0	100	"yes"
1	102	"yes"
0	103	"no"
0	104	"no"
0	106	"no"
1	107	"yes"
1	108	"yes"

Table 14: Low Level Removal Example

3.2.8 Observation to Variable Ratio

During the analysis process, the data-set used may have a lack of observations to create meaningful models. Throughout the *automodel* process, there are checks on the observation to variable ratio where if the ratio is less than 5, *automodel* throws an error message [13].

Here are the following checks done and the error messages they throw:

- After List-wise Deletion (see Section 3.2.5) or Missing Value Imputation (see Section 3.2.4) is done the following error message is thrown if the observation to variable ratio is less than 5 (in the error message below, *¡data-set ratio¿* is the observation to variable ratio of the current data-set as calculated in R):

Observation to Variable ratio of ¡data-set ratio¿ is less than 5 after Listwise Deletion (or has always been if using Multiple Imputation). Consider the following:

  – *Lower the value of function variable: 'naPercent'*
  – *Run missing value imputation by setting impFlag = TRUE*
  – *Use less variables*
  – *Gather more observations*

- After Low Level Removal (see Section 3.2.7) is done the following error message is thrown if the observation to variable ratio is less than 5 (in the error message below, *¡data-set ratio¿* is the observation to variable ratio of the current data-set as calculated in R):

  Observation to Variable ratio of ¡data-set ratio¿ is less than 5 after Low Level Removal. Consider the following:

  – *Lower the value of function variable: 'obsPerLevel'*
  – *Use less variables*
  – *Gather more observations*

3.2.9 Variance Checks

If a variable has 0 variance, the variable provides no predictive power since the variables value is the same for every observation. Since these variables can cause issues in other methods, we remove them during this check. This is achieved simply by calculating the variance of all variables, then removing those which have 0 variance.

Table 15 is an example using an arbitrary data-set with 10 observations.

Original Data			Transformed Data	
VarA	VarB	VarC	VarB	VarC
0	100	"yes"	100	"yes"
0	101	"yes"	101	"yes"
0	102	"no"	102	"no"
0	103	"yes"	103	"yes"
0	104	"yes"	104	"yes"
0	105	"no"	105	"no"
0	106	"no"	106	"no"
0	107	"yes"	107	"yes"
0	108	"yes"	108	"yes"
0	109	"yes"	109	"yes"

Table 15: Variance Check Example

3.2.10 Correlation Checks

If two or more variables are correlated, this would introduce multicollinearity to our regression models, specifically our Ordinary Linear Regression Model. If a model has multicollinearity [14]:

- It becomes hard to choose a list of significant variables since the model will give heavily varied results
- Coefficient estimates can vary by a large degree
- The model could be over-fit to the data meaning when applying the model to another sample of the same data, the accuracy of the model will be poor.

For those reasons, we preform checks to remove variables which are highly correlated. In these checks, we generate a correlation matrix, then remove variables according to a limit that is set by the function variable **corrConfLevel**, default is 0.8. In the automatic approach/check, the method of removal is finding the variable which has the most correlation coefficients above the **corrConfLevel** level. The count of the correlation coefficients are done on the full correlation matrix and the lower triangle of the correlation matrix (both with their diagonals set to 0). These counts are done independent of each other and then summed to generate our criterion on which we remove our variables. This means, by default, if VarA correlated with 4 variables with a coefficient of 0.8 and VarB correlated with 3 variables with a coefficient of 0.8, VarA will be removed first.

This design was chosen because correlation is non-transitive and removing the variable with the most correlations can lead to removing less variables overall. Also, If there is a tie between variables, the leftmost variable in the data-set is removed in the correlation pair. This means the user can have some manual input on the correlation checks by moving variables they want to keep in these checks to the end of the data-set.

the correlation matrix calculates the Pearson's correlation coefficient. The equation is the following:

$$\rho_{XY} = \frac{Cov(X,Y)}{\sqrt{Var(X)Var(Y)}} \tag{1}$$

- ρ_{XY}: The Pearson's correlation coefficient at the population level
- $Cov(X,Y)$: The Co-variance between X and Y
- $Var(X)$: The Variance of X
- $Var(Y)$: The Variance of Y
- X: The X variable
- Y: The Y variable

$$\rho_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

- ρ_{xy}: The Pearson's correlation coefficient at the sample level

- x_i: The ith observations of the X variable

- $\bar{x}$: The mean of the observations of X

- y_i: the ith observations of the Y variable

- $\bar{y}$: the mean of the observations of Y

To demonstrate the selection process, below is are 3 correlated variables. These values correlate in such a way so that X_1 correlates with X_2, X_2 correlates with X_3 but X_1 doesn't correlate with X_3 (demonstrating the non-transitive nature of correlations [15]).

$$X_1 = \{2, 0, 0, 1\}, X_2 = \{1, 0, 0, 1\}, X_3 = \{1, 0, 0, 2\}$$

Here's the full correlation matrix for the variables X_1, X_2, X_3

$$\begin{array}{c c} & \begin{array}{ccc} X_1 & X_2 & X_3 \end{array} \\ \begin{array}{c} X_1 \\ X_2 \\ X_3 \end{array} & \begin{pmatrix} 0 & 0.9 & 0.64 \\ 0.9 & 0 & 0.9 \\ 0.64 & 0.9 & 0 \end{pmatrix} \end{array}$$

and here's the lower triangle of the correlation matrix for the variables X_1, X_2, X_3

$$\begin{array}{c c} & \begin{array}{ccc} X_1 & X_2 & X_3 \end{array} \\ \begin{array}{c} X_1 \\ X_2 \\ X_3 \end{array} & \begin{pmatrix} 0 & 0 & 0 \\ 0.9 & 0 & 0 \\ 0.64 & 0.9 & 0 \end{pmatrix} \end{array}$$

We are looking to remove variables based on if their correlation coefficient is above 0.8 or not. For our first check, here are the counts of the full correlation matrix and lower correlation matrix summed.

- X_1: 2 coefficients

- X_2: 3 coefficients

- X_3: 1 coefficients

The above shows that this iteration of the check identified X_2 as the most correlated variable and has discriminated against X_1 over X_3 since X_1 is the leftmost variable (showing the effect a user can have on the selection process, if wanted). Given this, X_2 is removed from the data-set, the remaining correlation matrices are:

$$\begin{array}{c|cc} & X_1 & X_3 \\ X_1 & 0 & 0.64 \\ X_3 & 0.64 & 0 \end{array} \quad \begin{array}{c|cc} & X_1 & X_3 \\ X_1 & 0 & 0 \\ X_3 & 0.64 & 0 \end{array}$$

No more correlation coefficients are above out limit of 0.8 therefore we end our correlation checking process.

If we were to simply remove the variables in order of their position in the data-set (left-most variables to be removed first), we would of removed X_1 then X_2, resulting in two variables being removed. This isn't ideal and by preforming our methods, we will end up keeping more variables overall whilst giving the user the functionality to control the results.

3.3 Transforming

After cleaning our data-set, we preform transformation methods that aim to better represent our data. This data will then be used to fit our models (See Section 3.4).

3.3.1 Factorization

For our predictions to be accurate, we need to model our variables appropriately. We do this by detecting if a variable should be considered as a factor based on the number of unique values/levels the variable has. This number of unique levels is controlled by the **automodel** variable **catLevels**, which by default is 15. If a variable has **catLevels** or less unique values/levels, the variable is encoded as a factor (categorical), otherwise is considered numeric (continuous). Factor variables will be later encoded as dummy variables in Section 3.3.3. Numerical/Continuous variables will be later scaled in Section 3.3.2.

If there's a categorical variable with a more unique levels than **catLevels** (example: a variable holding the name of a person observed), the variable will be treated as continuous. This may not be the right way to model the variable however since our goal is prediction there are some things to consider:

- If this variable was made a factor, we could end up removing a lot of levels which have less than **obsPerLevel** observations.

- Using name as a scalar may find a strange hidden relationship so therefore, considering the above, should still be considered for modelling (VIF, backwards selection etc can still remove this variable)

- If the variable was a abstract string column that did measured in a scalar fashion, we would capture this by converting the column to a factor (see Section 3.2.2) then into a continuous variable. We would loose the string values associated with this column (however they can be inferred since they are en-coded in a systematic manor)

3.3.2 Standardization

When working with multiple continuous variables, they will most likely all be measured on different scales. If we model with these variables without standardization, we will introduce bias into our predictions due to the differences in scale. Example: **VarA** is measured in the range 0-10, **VarB** is measured in the range 0-10000. A 1 unit increase in **VarA** would be considered more impact than a 1 unit increase in **VarB** however our model won't be able to correctly model these impacts due to their difference in scale.

Another issue is that if our continuous variable is measured on a large scale (0-1000000), calculations on the variable may results in incredibly large values. This isn't desired since large numbers are computationally hard to solve and the memory within the coding language R can only handle integer values up to 2147483647, formally known as the *integer.max* within R.

To solve this issue, we standardize (create the Z-scores) our continuous variables so that it they have a mean of 0 and standard deviation of 1. This puts all of our continuous variables on the same scale ready for modelling.

$$X = \frac{X - E(X)}{\sqrt{Var(X)}} \tag{2}$$

This is Standardization equation at the population level

- X: The X variable being standardized
- $E(X)$: The mean of the X variable (The expected value)
- $Var(X)$: The variance of the X variable

$$x = \frac{x - \bar{x}}{\sqrt{\frac{1}{n-1}\sum_{j=1}^{n}(x_j - \bar{x})^2}}$$

This is Standardization equation at the sample level

- x: The x observation being standardized
- $\bar{x}$: The mean of the X variable
- n: The number of observations
- x_j: The jth observation of x (used to calculate the standard deviation per ith observation)

3.3.3 Dummy Variables

To correctly model our factor/categorical variables, we need to create dummy variables from them. This is achieved by taking a factor of n levels and producing n-1 dummy variables to represent it. Given this transformation, our resulting data-set for analysis will have more variables than the data-set inputted.

Table 16 is an arbitrary data set of 10 observations which we will turn into dummy variables. These variables are:

- **VarA**: a factor variable, its levels are yes & no
- **VarB**: a continuous variable
- **VarC**: a factor variable, its levels are 1,2 & 3

Original Data			Transformed Data			
VarA	VarB	VarC	VarA.yes	VarB	VarC.2	VarC.3
yes	100	1	1	100	0	0
no	101	2	0	101	1	0
no	102	1	0	102	0	0
yes	103	3	1	103	0	1
no	104	3	0	104	0	1
yes	105	2	1	105	1	0
no	106	2	0	106	1	0
no	107	1	0	107	0	0
no	108	3	0	108	0	1
yes	109	2	1	109	1	0

Table 16: Dummy Variables Example

3.3.4 Principal Component Analysis (PCA)

Optional to the user, a PCA transformation on the data-set can be preformed. This method takes our data-set and rotates it into new dimensions/axis which describes the variance the data-set in parts. This is done by calculating the co-variance matrix of the data-set, finding the eigenvalues & eigenvectors of said matrix and then rotating/transforming our data using the eigenvectors as the directions for our new axes. This then creates a data-set of Principal Components which are ordered left to right by the amount of variance explained [16]. Below we will go over the mathematics behind this transformation [17].

$$\begin{pmatrix} Var(X) & Cov(Y,X) & Cov(Z,X) \\ Cov(X,Y) & Var(Y) & Cov(Z,Y) \\ Cov(X,Z) & Cov(Y,Z) & Var(Z) \end{pmatrix} = Q\Lambda Q^{-1} \tag{3}$$

- X, Y, Z: Arbitrary predictor variables
- Q: Matrix of eigenvectors, follows the design:

$$\begin{pmatrix} q_{11} & q_{21} & \cdots & q_{n1} \\ q_{12} & q_{22} & \cdots & q_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ q_{1n} & q_{2n} & \cdots & q_{nn} \end{pmatrix} \tag{4}$$

The matrix is square with dimensions $n \times n$ since our original matrix, the co-variance matrix, is also square with dimensions $n \times n$. Every column of Q represents one eigenvector which describes the direction of the corresponding principal axis. These eigenvectors are ordered left to right in order of amount of variance explained

- Λ: Matrix of eigenvalues, follows the design:

$$\begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix} \tag{5}$$

  The matrix is square with dimensions $n \times n$ since our original matrix, the co-variance matrix, is also square with dimensions $n \times n$. Every diagonal value is the amount of variance explained by each principal component (PC). These eigenvalues are ordered left to right in order of amount of variance explained.

Dimension reduction is done using the eigenvalues/cumulative variance to remove the PC's. we keep all PC's before & including the PC where the cumulative sum of eigenvalues (cumulative variance) goes over **pcPercent** (default is 95% or 0.95) for the first time.

As an example, below is an arbitrary data set where we want to reduce our data set to only describe 95% of the variance in the data. Below is the matrix of eigenvalues for this example

$$\begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$$

We know that the sum of the variance described by the first 2 PCs (the first two eigenvalues λ_1, λ_2) is ¿= 95%, this can be represented like so

$$let : \frac{\lambda_1 + \lambda_2}{\lambda_1 + \lambda_2 + \lambda_3} >= 95\%$$

Therefore, to reduce the amount of dimensions we have in our data whilst still describing 95% of the variance in the data, we remove the third PC (the third eigenvalue λ_3). This creates the following matrix as our new reduced matrix of eigenvalues

$$\begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

3.3.5 Train Test Split

To allow for predictions, we need to split our data into training and testing sets. The size of the testing set is defined by the function variable **testPercent**, default value of 25% or 0.25. Since it's possible for a Train Test Split to make a variable have 0 variance in the training set, this processes is preformed in a loop until it's confirmed that the training set has non 0 variance for all variables. If a split cannot be found, the following error message is thrown:

> *No good Train Test split was found, please try increasing 'obsPerLevel' or changing 'randomSeed' if it was set and try again.*

If the user so desires, they can set **testPercent** to -1 so that the split only chooses one observation for the testing set (useful if you already have a testing set, seen used for the Titanic data in Section 5.4).

Table 17 is an example using an arbitrary data set of 12 observations using the default value of **testPercent** (which is 0.25). For the example given, notice that all the variables in the Training Set (**VarA**, **VarB** & **VarC**) maintain a non-0 variance, this feature is required for a Train Test Split to be considered.

Original Data		
VarA	VarB	VarC
40	100	1
50	101	2
50	102	1
40	103	3
50	104	3
40	105	2
50	106	2
50	107	1
50	108	3
40	109	2
40	110	1
50	111	3

Training Set		
VarA	VarB	VarC
40	100	1
50	102	1
50	104	3
40	105	2
50	106	2
50	108	3
40	109	2
40	110	1
50	111	3

Testing Set		
VarA	VarB	VarC
50	101	2
40	103	3
50	107	1

Table 17: Train Test Split Example

3.4 Modelling

As our final output of the **automodel**, we generate various different models of our data-set. These models are a mixture of classification and regression techniques to provide a varied look at what models best describe the data-set.

3.4.1 K-Means

This is a clustering algorithm which aims to cluster similar observations into the same clusters [7]. In K-Means, we are aiming to minimize the sum of the sum of the squared difference between observations and the centroid of a cluster. To break this down, the sum of the squared difference between observations and the centroid of a cluster is mathematically similar to the sum of the squared residuals (SSR) from Ordinary Linear Regression (see Section 3.4.4 equation 10). We are aiming to minimise the sum of the SSR per cluster to produce our K clusters. To evaluate this model, we use within and between MSE and exact prediction accuracy (more on this in Section 3.4.6). Given this is a classification model, the MSE produced should not be compared to other models within **automodel** expect other K-Means models.

$$\underset{s}{argmin} \sum_{i=1}^{k} \sum_{x \in S_i} |x - \mu_i|^2 \tag{6}$$

- $\underset{s}{argmin}$: This is telling us that we are aiming to minimize $||x - \mu_i||^2$ for each cluster, S
- k: The amount of clusters
- x: an observation in the current cluster, S_i
- S_i: The respective cluster (the ith cluster)
- μ_i: the mean of the points in the ith cluster (the centroid of cluster S_i)

In this dissertation, we use this classification method in an unorthodox way were we use the clusters to predict a variable. This is generally not the desired use for K-Means and the model should be instead evaluated using inside and between MSE. We would expect this model to produce a lower predictive power than the regression models generated using the **automodel** function.

3.4.2 kNN

kNN, or K Nearest neighbors, is a clustering algorithm which attempts to predict a data-point based on the K closest observations. To evaluate this model, we use MSE and exact prediction accuracy (more on this in Section 3.4.6). Given this is a classification model, the MSE produced should not be compared to other models within **automodel** expect other kNN models. The algorithm step by step is as follows:

1. a Train Test split is preformed on a data-set so that we have two sets of observations of the same data-set
2. We then decide how many neighbors should be considered (the value of k) in our kNN model
3. We then take the first observation from the testing data and calculate the Euclidean distance between this observation and all observation in the training data
4. We then order the resulting Euclidean distances from smallest to largest and take the top k training observation as the neighbors to be considered.
5. Among these k nearest neighbors, we then take the most popular/frequent value and predict our test observation to be the same
6. we repeat the steps 3 to 5 for each observation in the testing data-set.

The general equation for Euclidean Distance in the context in kNN is as follows [18]:

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{7}$$

- n: The number of dimensions each observation has
- q_i: The ith dimension within the training observation, q
- p_i: The ith dimension within the testing observation, p

The help explain further, please see Figure 7 [19]. In Figure 7 it shows how a test observation (see the yellow square) would be classified given different values for k using the kNN algorithm. When $k = 3$, our test observation would be classified as class B (a green triangle). When $k = 7$, our test observation would be classified as class A (a red star).

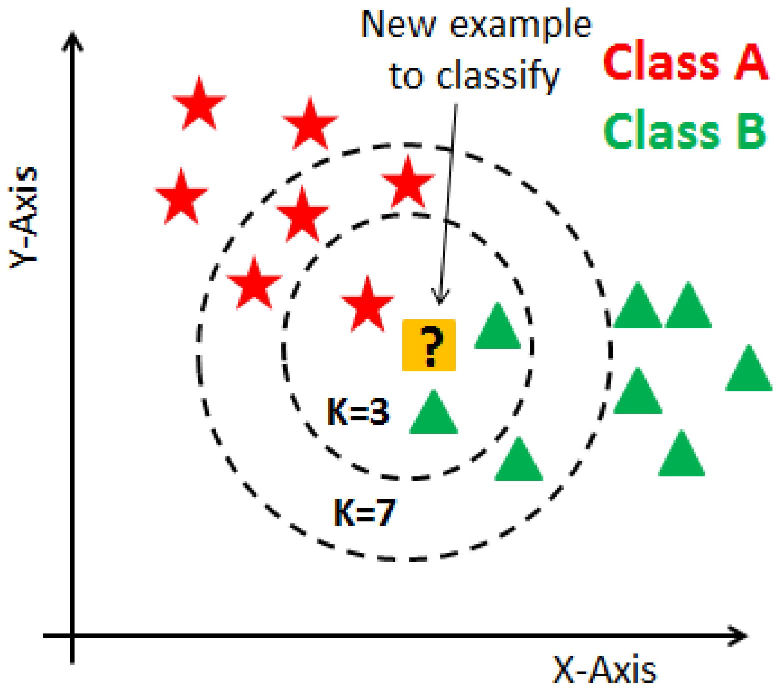


Figure 7: kNN at different values of k

3.4.3 Classification and Regression Trees (CART)

CART models work by creating thresholds that splits the data into groups per observations per variable. A certain number of observations are needed before we can decide to make a threshold at any stage in the tree which can be controlled by the function variable **cartSplit**, the default is 20. In **automodel**, CART is used to preform our missing value imputation (See Section 3.2.4) and to create a prediction model. It generates a tree which can predict the dependant variable and gives a list of variables which were the most important in predicting. We evaluate our CART model using MSE and exact prediction accuracy. The MSE created can be compared to the other regression models within **automodel** but not to the classification models.

Here's a explanation of what happens at each stage during the CART process [20] [21]:

1. a dependant variable, Y, and the predictor variables, X, are decided
2. starting from the first predictor variable, X_1, we create a threshold using the first observation, x_1. An example threshold at this stage would be if the ith observation x_i is greater than x_1.
3. We calculate the mean value of our dependant variable, Y, in the groups that are made due to the threshold.
4. Using our calculated mean values for Y in each group, we calculate the Sum of the Squared Residuals (SSR) per group using the mean value as the predicted value.
5. We then take the SSR of each group and sum them together to create a final score for the current threshold.
6. We repeat steps 2 to 5 going though each observation of each predictor variable until we have calculated a score for each possible threshold split of the data across all predictor variables.
7. We then pick the smallest score and set the associated threshold as a node on the tree. This node will then predict the mean values found in step 3.
8. Then, if the group created by the threshold has **cartSplit** or more observations, we repeat steps 2 to 7 again, treating all the data-points in one group independently from all other groups.
9. Once no more groups can be split, we have made our CART model which can be used to predict the dependant variable, Y

The general equation to calculate a groups mean in the context of CART is as follows:

$$\bar{y_i} = \frac{1}{n_i} \sum_{j=1}^{n_1} y_j \tag{8}$$

- $\bar{y_i}$: The mean value of Y for group i (one of the group created by the current threshold)
- n_i: The number of observations that have fallen into group i
- y_j: the jth observation of the dependant variable in group i.

The general equation to calculate a groups SSR in the context of CART is as follows:

$$\sum_{j=1}^{n_i}(y_j - \bar{y_i})^2 \tag{9}$$

- n_i: The number of observations that have fallen into group i due to the threshold used.
- y_j: the jth observation of the dependant variable in group i.
- $\bar{y_i}$: The mean value of Y for group i (one of the group created by the current threshold)

The help explain further, please see Figure 8 [20] .In Figure 8, we only have 1 predictor variable, *Drug Dosage*, and our dependant variable is *Drug Effectiveness*. We only split a node only if they have 7 or more observations (equivalent to setting **cartSplit** to 7). The result is a CART model with 3 nodes that predict 1 of 4 different values for **Drug Effectiveness** based on the value of **Drug Dosage**.

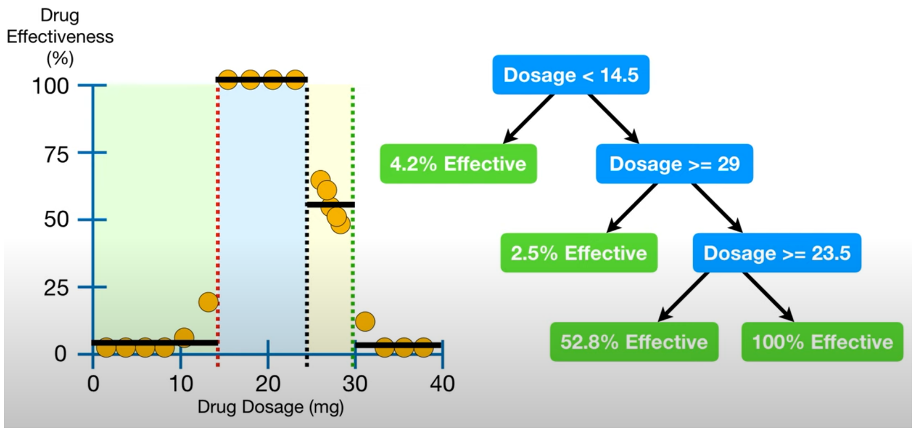


Figure 8: Simple CART model from Josh Starmer's *Regression Trees Clearly Explained*

3.4.4 Ordinary Linear Regression (OLR)

We use the base standard Ordinary Linear Regression (OLR) model as provided by R using the function **lm()**. Our model is fitted by minimizing the Sum of the Squared Residuals (SSR), which in layman's terms, is reducing the total distance between the observations and the fitted model. We evaluate our Ordinary Linear Regression model using MSE and exact prediction accuracy. The MSE created can be compared to the other regression models within **automodel** but not to the classification models.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 ... + \beta_n x_n + \epsilon \tag{10}$$

- $\hat{y}$: The predicted value of the dependant variable
- β_i: The coefficient of the ith variable
- n: the amount of independent variables in the final model

$$SSR = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{11}$$

- SSR: This is the short hand for the Sum of Squared Residuals, what we minimize to fit the linear model
- y_i: The ith value of the dependant variable (y-variable)
- $\hat{y}$: The ith predicted value of the dependant variable (y-variable)

Starting from the full model, we use Variance Inflation Factor (VIF) to remove any leftover multicollinearity from our model. VIF is calculated per predictor variable and gives a measurement of how likely it is for the current predictor variable to be predicted by all other predictor variables. Variables are removed from a model in order largest to smallest VIF score. We stop removing predictor variables once all VIF scores are below the function variable **vifSelectionLevel**, which by default is 10.

$$VIF = \frac{1}{1 - R_n^2} \tag{12}$$

- R_n^2: This is the R^2 score from the regression of X_n on all other predictor variables. For example, given we have a Ordinary Linear Regression model fit (See Section 3.4.4 Ordinary Linear Regression, Equation 10), R_1^2 is calculated from the following equation:

  $$\hat{\beta_1} = \beta_0 + \beta_2 x_2 + \beta_3 x_3 ... + \beta_n x_n + \epsilon \tag{13}$$

  The equation used to calculated R_1^2 would then be as follows:

  $$R_1^2 = \frac{\sum(\beta_{1i} - \hat{\beta}_{1i})^2}{\beta_{1i} - \bar{\beta}_{1i})^2} \tag{14}$$

After preforming our VIF checks, we do a backwards selection on the remaining model variables based on their individual significance. The level of significance required is determined by the **automodel** variable **modelSigLevel**, which by default is set to 0.95 (meaning a 95% confidence interval). A predictor variable is kept in the final Ordinary Linear Regression model if they are considered significant at the **modelSigLevel** confidence level. To calculate a variables significance, we first divide the variables estimate coefficient by it's standard error to give us a t-value. Using this t-value, when then preform a test to determine if the absolute value of the t-distribution is greater than the absolute value of our t-value. This test then generates a p-value where the smaller the value, the more significant the predictor variable (the smaller the p-value, the more likely it is that the absolute value of our calculated t-value is greater than the absolute value of the t-distribution). We are selecting of variables based on if their p-value is less than or equal to 1 - **modelSigLevel**. [22]

3.4.5 Elastic Net Regression (ENR)

The final model within **automodel**, this creates a regression model based on minimizing the SSR of the model plus a penalty function. The penalty function used is a mixture of the penalty functions from Lasso and Ridge Regression. We evaluate our Elastic Net Regression (ENR) model using MSE and exact prediction accuracy. The MSE created can be compared to the other regression models within **automodel** but not to the classification models.

To get a proper understanding of the differences between Ordinary Linear, Lasso, Ridge and Elastic Net Regression, we will go over what each model is attempting to minimize/ The cost function of each model.

1. **Ordinary Linear Regression**

$$SSR = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{15}$$

- SSR: This is the short hand for the Sum of Squared Residuals, what we minimize in Ordinary Linear Regression (see Section 3.4.4 equation 10)
- n: The number of observations
- y_i: The ith value of the dependant variable (y-variable)
- $\hat{y}$: The ith predicted value of the dependant variable (y-variable)

2. **Lasso Regression**

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{k}|\beta_j| \tag{16}$$

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)$: Notice this is SSR, as seen in Ordinary Linear Regression
- n: The number of observations
- y_i: The ith value of the dependant variable (y-variable)
- $\hat{y}$: The ith predicted value of the dependant variable (y-variable)
- λ: A tuning parameter that determines the severity of the effect the slope has on fitting the model. It can take a value from 0 to $+\infty$. The value for this parameter is chosen though cross-validation in this paper.
- k: The number of predictor variables
- β_j: the coefficient of the jth predictor variable within the regression model. [23] [24]

3. **Ridge Regression**

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) + \lambda\sum_{j=1}^{k}\beta_j^2 \tag{17}$$

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)$: Notice this is SSR, as seen in Ordinary Linear Regression
- n: The number of observations
- y_i: The ith value of the dependant variable (y-variable)
- $\hat{y}$: The ith predicted value of the dependant variable (y-variable)
- λ: A tuning parameter that determines the severity of the effect the slope has on fitting the model. It can take a value from 0 to $+\infty$. The value for this parameter is chosen though cross-validation in this dissertation.
- k: The number of predictor variables
- β_j: the coefficient of the jth predictor variable within the regression model. [23] [25]

4. **Elastic Net Regression**

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i) + \lambda[\alpha\sum_{j=1}^{k}|\beta_j| + (1-\alpha)\sum_{j=1}^{k}\beta_j^2] \tag{18}$$

- $\sum_{i=1}^{n}(y_i - \hat{y}_i)$: Notice this is SSR, as seen in Ordinary Linear Regression
- n: The number of observations
- y_i: The ith value of the dependant variable (y-variable)
- $\hat{y}$: The ith predicted value of the dependant variable (y-variable)
- λ: A tuning parameter that determines the severity of the effect the slope has on fitting the model. It can take a value from 0 to $+\infty$. The value for this parameter is chosen though Cross Validation in this dissertation (explained further in depth within this section). We choose our λ value by picking the lowest value found during Cross Validation.
- α: a tuning parameter that determines the weight of the lasso and ridge penalties. It can take a value between 0 and 1, when $\alpha = 0$, it's identical to Ridge Regression, when $\alpha = 1$, it's identical to Lasso Regression.
- k: The number of predictor variables
- β_j: the coefficient of the jth predictor variable within the regression model. [23] [26]

When running **automodel**, we fit **elasticCount** plus 1 models each with a different value for α. The values for *alpha* are decided by incrementing from 0 in steps of size $\frac{1}{\textbf{elasticCount}}$. This would means that when **elasticCount** is 10, we fit models with the alpha value 0, 0.1, 0.2, 0.3 ... 1. The user will be able to see the MSE and R^2 value for each model but not the exact prediction accuracy. We then pick to model with the best MSE as our model of best fit and calculate an exact prediction accuracy for this model.

We use cross validation (10 fold cross validation to be precise) is used in Elastic Net Regression to choose the best λ per fit [27].

Below is an example 10 fold cross validation being preformed to choose the best value for the arbitrary parameter λ. Our error which we are looking to minimize is taken from Ridge Regression (a shortened version is used to simply the example)

$$SSR + \lambda\hat{\beta}^2 \tag{19}$$

- SSR: Represents the Sum of the Squared Residuals
- λ: Value to be determined in cross validation

- $\hat{\beta}^2$: The matrix of coefficients describing the fit of the model

In this example, as lambda increases, a larger emphasis is put on reducing the model's *slope* squared (represented as $\hat{\beta}^2$) over reducing the Sum of the Squared Residuals (SSR). This means that as λ varies, SSR and $\hat{\beta}$ changes. Since SSR and $\hat{\beta}$ change, the cost function (the equation used in this example) and λ aren't directly proportional. Then we pick our λ by a criterion of the users choosing (like picking the λ with the smallest value)

3.4.6 Model Evaluation Metrics

To evaluate our models, we calculate various statistics on the results of our predictions. Below we will list though these stats that are used.

Firstly, we use R^2 to evaluate our Ordinary Linear and Elastic Net Regression models.R^2 is a measurement of the of how much of the variance in y is explained by the model/predictor variables. In **automodel**, it's used within Ordinary Linear Regression, VIF and Elastic Net Regression. Below is the equation used to calculate R^2 [28]

$$R^2 = \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{20}$$

- y_i: The ith value of the y variable
- $\hat{y}_i$): the predicted value for y_i
- $\bar{y}$: The mean value of the y variable

Secondly, we use Mean Squared Error (MSE) to evaluate all of our prediction models. MSE is the Mean Squared Error and depending on the model, will be calculated differently. Example: the MSE used in K-Means is a calculation on the Euclidean Distance between a observation and the centroid of the cluster it's apart of. The MSE calculated in Ordinary Linear Regression is the mean of the squared error between the real and predicted values of the dependant variable. Since the method to calculate MSE is different in these two models, they shouldn't be used to compare the goodness of fit for each model. Below is the general equation used to calculate MSE for a regression model.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{21}$$

This is the general equation for the MSE in a regression model.

- n: The number of observations
- y_i: The ith value of the y variable
- $\hat{y}_i$): the predicted value for y_i

Thirdly, we use Akaike information criterion (AIC) to evaluate our Ordinary Linear Regression (OLR) model. AIC is a measurement used to compare different models of the data to determine which model was the best fit. The magnitude of AIC alone doesn't provide the user with much information, its the differences in AIC between two models trained on the same data that is useful [29].

$$AIC = 2k - 2ln(\hat{L}) \tag{22}$$

- k: The number of predictor variables in the model
- $ln()$: This is the Natural logarithm, log_{exp}
- $\hat{L}$: The maximum value of the likelihood function for the model

Lastly, we calculate a prediction accuracy to evaluate all of our prediction models. We calculate the accuracy within a few different contexts/ranges:

- **Exact Accuracy**: An exact prediction accuracy is calculated by counting the number of times the model correctly predicted the dependant variables and diving it by the total number of observations. This method of evaluation is only useful when the dependant variable is categorical/ has a relatively small measurement scale.
- **Confidence Interval Accuracy**: For only the Ordinary Linear Regression Model, we predict within a 95% confidence interval. This gives us a little more information for the Ordinary Linear Regression model for us to make evaluations with. It's especially useful if our dependant variable is continuous and therefore accounts for predictions where the error in prediction may be small but not 0. The equation for this interval is:

$$\hat{y} \pm t_{\alpha/2,n-2}\sqrt{MSE(\frac{1}{n} + \frac{(x_k - \bar{x})^2}{\Sigma(x_i - \bar{x}^2)})} \tag{23}$$

  – $\hat{y}$: The given predicted value
  – α: is the confidence level, which in our case is 0.05
  – $t_{\alpha/2,n-2}$: Is the respective value from the T table distribution
  – MSE: This is the Mean Squared Error of the model
  – n: The number of observations
  – x_k: The given predictor variables
  – $\bar{x}$: The mean of the predictor variables
  – x_i: Represents the i-th predictor variable

- **+-1 Range Accuracy**: Only used for the Understanding Society data, for some variables we are predicting (like the GHQ score/scghq1_dv) we want to see if our prediction of within a 1 unit difference of the real value. This is done by if a prediction is 1 one unit change up or down (+-) from the real value, we mark the prediction as correct.

- **GHQ Range Accuracy**: Only used for the Understanding Society data, this is a calculated interval based on the results found for the GHQ score (scghq1_dv) and the grouped version of the GHQ score (scghq2_dv). The calculation for the unit change for this interval is as follows:

$$\hat{y} \pm ((\frac{unique_{ghq1}}{unique_{ghq2}}) \div 2) \tag{24}$$

  - $\hat{y}$: The given predicted value
  - $unique_{ghq1}$: The amount of unique scores in **scghq1_dv**, found in R using

```
length(unique(scghq1\_dv))
```

  - $unique_{ghq2}$: The amount of unique scores in **scghq2_dv**, found in R using

```
length(unique(scghq2\_dv))
```

3.5 Run Order

When running **automodel**, we process all of our methods in a select order. This ordered list will describes to order of which this methods are executed. As seen below, Some function may be ran multiple times, this is done to ensure a smooth run.

3.2.1 Data-frame Checks

3.2.2 Data Type Checks

3.2.6 Pair-wise Deletion

3.2.3 High Missing Data Variables

3.2.4 Missing Value Imputation, if **impFlag** = TRUE (see Section 3.6.6)

3.2.5 List-wise Deletion

3.2.8 Observation to Variable Ratio

3.3.1 Factorization

3.2.7 Low Level Removal

3.2.8 Observation to Variable Ratio

3.2.9 Variance Checks

3.3.3 Dummy Variables

3.2.9 Variance Checks

3.4.1 K-Means

3.2.10 Correlation Checks

3.3.2 Standardization

3.3.4 Principal Component Analysis (PCA), if **PCAFlag** = TRUE (see Section 3.6.11)

3.3.5 Train Test Split

3.4.2 kNN

3.4.4 Ordinary Linear Regression (OLR)

3.4.5 Elastic Net Regression (ENR)

3.6 Function Variables

To allow for user customization, we have a list of different variables which can be set by the user to change the process of the analysis. For each variable we will explain it's initial state (default value), description, provide an example case of it being used and any associated issues. The variables appear in order of usage within the **automodel** function.

3.6.1 predictVar

- initial state: Needed from user
- description: The name of the variable that is to be predicted/the name of the dependant variable
- example case: a user wants to predict the variable **age** from a data-set, **predictVar** should be set to: ”age”
- associated issues: If not specified, function will not run. Value inputted needs to be a string value or else the function won't run

3.6.2 data

- initial state: Needed from user
- description: The data-set of all the predictor variables and the dependant variable
- example case: a user from to predict a variable from the loaded in data-set **footballdata**, **data** should be set to: footballdata
- associated issues: If not specified, function will not run. The data parsed needs to be able to cast to a data.frame object in R, if not, the function will not run.

3.6.3 naPercent

- initial state: 0.2 (represents 20%)
- description: This is the percentage amount of NA values allowed in a predictor variable. If the predictor variable have **naPercent** percent or more of their values as NA/missing, the predictor variable is removed
- example case: Say we are happy to keep columns with a larger amount of missing data, we can set **naPercent** to 0.4 to keep variables that have less than 40% of their data as missing/NA
- associated issues: If this value is set to high, List-wise deletion may delete almost all observations and missing value imputation may impute misleading results

3.6.4 cartSplit

- initial state: 20
- description: The amount of observations needed for a new threshold/node to be generated in a CART model. CART is used to impute missing values and to generate a prediction model of the data within the code.
- example case: When using a data-set with a low amount of observations and predictor variables (say 100 observations of 3 variables), we may want to split the nodes when there are less observations. Therefore we would set **cartSplit** to a lower value like 5
- associated issues: If this value is greater than or equal to half the amount of observations in the data-set, CART will always be creating one node which can't be split, potentially leading to very poor predictions. If **cartSplit** is too low (like 2), we risk over-fitting the model to the training data

3.6.5 impFlag

- initial state: FALSE
- description: Flag to tell the function if we are wanting to impute the missing values instead of preforming a List-wise deletion. If TRUE, a single CART imputation is done to impute the missing values.
- example case: If we have a data-set with a relatively low amount of missing data where the missing data isn't heavily bias towards any outcome (MACR or MAR assumption), we can impute the missing values to use more observations in out modelling. By changing **impFlag** to TRUE, we are telling the automodel to preform a missing value imputation on the data-set (this happens after the removal of variables with high missing values, see **naPercent**)
- associated issues: Imputation can take a long time on a large data-set and therefore should be taken into consideration. If the missing values in the data doesn't uphold the MACR or MAR assumptions, missing value imputation would badly represent the missing values in the data.

3.6.6 randomSeed

- initial state: NULL
- description: To make results reproducible, we must set a random seed within the function. **randomSeed** can be set to any integer in the range -2147483647 to 2147483647 (2147483647 is the maximum integer allowed in R).
- example case: To produce results for a report on a data-set, we would want to make sure that a observer can reproduce the results found. To ensure this, we would set **randomSeed** to any integer value, such as 1, each time we want to run the automodel function.
- associated issues: Due to the amount of models that need to have their random effect controlled, it's not possible to use **set.seed()** before calling **automodel**. Sometimes it can be quite hard to find a seed that has a Train Test split that processes correctly, to confirm a working seed exists, try running the function without setting **randomSeed** first.

3.6.7 catLevels

- initial state: 15
- description: Decides on how many unique values/levels a variable needs to be considered as a factor/categorical variable when modelling. If a variable has less than or equal to **catLevels** levels, the variable gets encoded as a factor.
- example case: Say we know that all of our categorical variables have less than 5 unique levels, we could change **catLevels** to 5 so that we ensure that no variables that were intended to be continuous get encoded as a factor (since if a continuous variable only has less than or equal to **catLevels** unique values/levels, it will be encoded as a factor).
- associated issues: If **catLevels** is set to low (for example, 2), incorrect modelling of variables may occur resulting in poor predictions. If set to high (the number of observations in the data-set), we risk over-fitting our model to the exact observations seen in continuous variables, causing poor predictions

3.6.8 obsPerLevel

- initial state: 5
- description: How many observations are needed of a level in a categorical/factor variable. If the level in the variable has less than **obsPerLevel** observations, all observations of this level are removed from the data-set
- example case: If our data-set is known to have 2 observations of a case in a categorical variable and all other levels have 3+ observations which we are happy to keep. We can set **obsPerLevel** to 3 so that this case with 2 observations is removed.
- associated issues: There are quite a few issues involved with the tweaking of this variable. Setting **obsPerLevel** too low may lead to the function not finding a valuable test, train split which maintains variance in all variables in the training data. If we have a known low amount of observations in categorical variables but wish to keep them, we can set the variable **testPercent** to -1 so that we only take one observation as our test set. This risks over-fitting and should be done at the digression of the user. Setting **testPercent** to high risks deleting large amounts of observations within the data-set, therefore making the resulting data-set unusable for modelling. Sometimes an error message from the automodel function will tell the user to modify this variable, it should be known that if this is the case, this variable should be incremented in small steps to find a workable solution.

3.6.9 clusterAmount

- initial state: determined by a logical statement

  ```
  ifelse(length(unique(data[[predictVar]])) <= catLevels,
  length(unique(data[[predictVar]])),
  catLevels)
  ```

  -
    ```
    predictVar
    ```

    The dependant variable which the user wishes to predict (function variable, **predictVar**)

  -
    ```
    length(unique(data[[predictVar]]))
    ```

    The number of unique values within the dependant variable

  -
    ```
    catLevels
    ```

    The number of unique values within the dependant variable (function variable, **catLevels**)

  The logical statement sets the number of clusters to the amount of unique values/levels in the dependant variable unless this amount is larger than **catLevels**, which if true, then sets the number of clusters to **catLevels**.

- description: Sets the amount of clusters/centroids to be used in K-Means modelling. Our K-Means model has **clusterAmount** clusters.

- example case: If we know that our dependant variable is binary (dependant variables can only take two values), we can set **clusterAmount** to two.

- associated issues: Setting **clusterAmount** to high will risk over-fitting our clustering model to the current observations (risk of making uninformative clusters). Predicting a continuous dependant variable using clustering doesn't garner good results.

3.6.10 corrConfLevel

- initial state: 0.8 (represents 80%)
- description: The cutoff point of when a correlation between two predictor variables is deemed to large/ will cause multicollinearity in a model. If the absolute value of the correlation coefficient between two predictor variables is greater than **corrConfLevel**, these variables will be considered for deletion.
- example case: If we are ok to have correlated variables in our data-set (we are more interested in the CART and clustering models than the regression models), we can set **corrConfLevel** to 0.99 so that all expect extremely similar predictor variables (basically predictor variables that are the same) are considered for deletion
- associated issues: If **corrConfLevel** is set to high (like 0.99), the results from our regression models will be bad and shouldn't be considered accurate (too much multicollinearity in the predictor variables, unstable model).

3.6.11 PCAFlag

- initial state: FALSE
- description: Flag that tells the **automodel** function to preform PCA on the data-set. If TRUE, PCA using the co-variance matrix and eigenvalue decomposition is preformed.
- example case: If we aren't worried about seeing what variables are significant towards predicting our predictor variable and instead we want to fit the most accurate model with the smallest amount of variables, we would want to preform PCA. To do this, we set **PCAFlag** to TRUE.
- associated issues: PCA will make the regression models prediction using principal components instead of predictor variables therefore making these models harder to interpret (performance/goodness statistics are still relevant though, like R^2.)

3.6.12 pcPercent

- initial state: 0.95 (represents 95%)

- description: This sets the needed amount of variance explained by the reduced principal component (PC) data-set. When preforming dimension reduction, we keep the smallest amount of PC's that describe**pcPercent** of the variance in the data-set.

- example case: We want to reduce the amount of variables in our data-set whilst still explaining 90% of it's variance. We initially set **PCAFlag** to TRUE to tell automodel that we want to preform PCA, then we set **pcPercent** to 0.9.

- associated issues: it's worth considering that setting **pcPercent** low may result in using one PC, this is fine based on the data used and the value **pcPercent** was set to.

3.6.13 testPercent

- initial state: 0.25 (represents 25%)

- description: The proportion of observations after transformation within the testing data. When preforming our Train Test split on our data, we set **testPercent** percent of our observations as our testing data and $1 -$ **testPercent** as our training data. Our training data must maintain variance in all predictor variables and therefore the splitting process iterates until a correct split can be found. If **randomSeed** is set, it only evaluates the Train Test split associated with the seed once. A user can set **testPercent** to -1 if only one observation should be tested.

- example case: We want to have a Train Test split where 10% of our observations after transformation are in the training data. To achieve this we set **testPercent** to 0.1.

- associated issues: Since we are looking for a split where the Training data maintains variance for all variables, data-sets with low observations per categorical variable levels will struggle to maintain this variance in the training data. This is unavoidable since if we don't maintain variance in the training data, those predictor variables with no variance are practically useless when modelling (since they have the same value for all observations). If a Train Test split can't be found because of this reason, it's recommended that the automodel variable **obsPerLevel** should be increased or missing value imputation is preformed on the data-set (done by setting **impFlag** to TRUE). For these reasons, it would also be hard to find a working random seed value (setting **randomSeed**) since the amount of splits a data-set of this nature could do would be limited.

3.6.14 kNNCount

- initial state: determined by the calculation

  ```
  round(sqrt(nrow(data)),0)
  ```

  This is the same as doing $\sqrt{n}$. n is the number of observations in the data-set and the answer is rounded to 0 decimal places.

- description: The number of neighbors that are considered for each test observation in the kNN model. Each test observation considers **kNNCount** neighbors to predict the dependant variable for the observation.

- example case: We want to consider the 10 closest points (the neighbors) to predict each observation in the kNN model

- associated issues: the **kNNCount** has to be an integer value between 1 and the number of observations in the transformed data-set. Setting this value to large (20% or more of the number of observations) will cause all predictions to converge to the same prediction, making the model inaccurate (will always predict the same values).

3.6.15 vifSelectionLevel

- initial state: 10

- description: We will be removing variables from the Ordinary Linear Regression model in order of their VIF score until all variables have a VIF score equal to or below **vifSelectionLevel**.

- example case: Given we have a good understanding of the co-linearity between the predictor variables and our data has many predictor variables (which naturally increases R^2), we want to increase the VIF threshold to 20. To do this, we change the value of **vifSelectionLevel** to 20

- associated issues: Setting **vifSelectionLevel** to a large value (such as 100) will create a model with high multicollinearity which generate poor/unstable predictions of our dependant variable.

3.6.16 modelSigLevel

- initial state: 0.95 (stands for 95%)
- description: To refine our Ordinary Linear Regression Model we remove variables based on their statistical significance, we use **modelSigLevel** to decide the minimum confidence level that each variable in the model needs to satisfy. Each model needs to be considered significant at the **modelSigLevel** confidence level.
- example case: The data-set we are analyzing contains a large amount of predictor variables (100+) and therefore we want to the model to only contain model that are highly significant/significant at the 99% confidence level. To do this, we set **modelSigLevel** to 0.99.
- associated issues: If this value is is set to 1, we would end up with a null model, if set to 0, we would end up with the full model after VIF checks.

3.6.17 elasticCount

- initial state: 10
- description: When fitting our elastic net models, we need to fit models for different values of α (which takes values between 0 and 1). **elasticCount** decides how many different values of alpha we try and the different values are incremented equally based on the value of **elasticCount**
- example case: Say we want to fit an Elastic Net Regression model for each 0.05 increment in α of value. To do this we would set the value of **elasticCount** to 20 (since $\frac{1}{20}$ is 0.05).
- associated issues: **elasticCount** needs to be an integer value large than 1. Setting **elasticCount** to a large integer could significantly increase the run-time of **automodel** based on the size of the data-set being processed

3.7 Assisting Custom Functions

To represent all of our data correctly, we need some custom functions to merge & analyze the data-sets to be used with **automodel**. All the functions mentioned here are for the Understanding Society (see Section 2.1) only due to the mass amount of data we are going to process from it. It's worth keeping in mind though that the idea of this dissertation was to keep most of the process in the **automodel**

3.7.1 recodeNA

This function is needed for the Understanding Society data. It takes all the different categories of missing values in the Understanding Society data and re-codes them to **NA**

```
recodeNA = function(data){
data[data == -9] = NA
data[data == -8] = NA
data[data == -7] = NA
data[data == -2] = NA
data[data == -1] = NA
return(data)
}
```

The above is finding all instances of -9, -8, -7, -2, -1 within **data** and changing them to **NA**. The returned Understanding Society data-set will have all missing entries in the data-set encoded as **NA**.

3.7.2 sqlTransform

To be able to properly analyze the income data of the participants alongside their survey data, we need to transform the income data into a form where the *pidp* is unique per row. The final SQL query is executed though the **sqldf** library.

```
sqlTransform = function(data){

#start of the SQL query
querySQL = "SELECT pidp,SUM(frmnthimp_dv) as frmnthimp_dv_total"

#for loop which turns all ficodes into binary columns
for (i in 1:max(data$ficode)){
    colName = paste("ficode", i,sep="")
    data[[colName]] = ifelse(data$ficode == i, 1, 0)
    querySQL = paste(querySQL,",SUM(",colName,") as ",colName,sep = "")
    }

#finish of the SQL query
querySQL = paste(querySQL,"FROM data GROUP BY pidp")

#SQL to turn the table into version where pidp is unique per row
return(sqldf(querySQL))
}.
```

In the above, **data** represents the respective income data passed though (**w2income** or **w3income**), **ficode** tells us what category of income the row represents & **frmnthimp_dv** is the amount of income for the respective **ficode**. In summary, what we are doing is creating a binary variable (1 or 0) for each **ficode** present therefore making our **pidp** unique per row.

To assist with explanation, please Table 18 and Table 19 below for a representation of the transformation:

Non-Transformed Income Data		
pidp	frmnthimp_dv	ficode
0001	800	1
0001	1200	2
0002	500	3
0002	700	4

Table 18: sqlTransform Example Original Data

Transformed Income Data					
pidp	frmnthimp_dv_total	ficode1	ficode2	ficode3	ficode4
0001	2000	1	1	0	0
0002	1200	0	0	1	1

Table 19: sqlTransform Example Transformed Data

The returned data-set is a transformed version of the respective income data which now has a unique *pidp* per row.

3.7.3 ghq_analyze

Before running our GHQ data though the main function, we do preliminary analysis on the GHQ data within each join of the Understanding Society data. This function includes different numerical and graphical summaries. The code for this function can be found in Section 10

To summarize what the function is doing in order of execution:

1. Numerical summaries of the *scghq1_dv* variable
2. Plots a histogram of *scghq1_dv*
3. Plots a QQ-plot of *scghq1_dv*
4. Plots the correlation matrix of all the GHQ questions

3.7.4 ghq_clean_move

Since we have some preliminary understanding of the Understanding Society and the GHQ data, we want to remove the individual GHQ questions/variables from the data. We also want to move all variables that end in *val* or *_dv* to the end of the data-set since we know these variables are derived/averages of other variables they would highly correlate with (and we prefer to keep these variables).

```
ghq_clean_move = function(data) {

#changing the name of the ghq total score and
#removing the questions from the data-set
names(data)[names(data) == names(select(data,contains("ghq1")))]
```

```
    = "total_score"
data = select(data,-contains("ghq"))

#re-ordering the columns for the correlation check so
#that we automatically keep all 'val' variables
for (i in 1:length(names(data))) {
    if (grepl("val",colnames(data)[i], fixed = T) |
        grepl("_dv",colnames(data)[i], fixed = T)) {
        data = data %>% relocate(colnames(data)[i], .after = last_col())
    }
}

#return the cleaned data-set
return(data)

}
```

To summarize what the function is doing in order of execution:

1. Changes the name of the variable *scghq1_dv* to *total_score*
2. re-orders the data parsed so that variables ending in *val* or *_dv* are at the end/rightmost within the data

3.7.5 predGHQadd

For our GHQ data, we have would like to see how the predictions for each model do within certain ranges. To achieve this we take the prediction results from all the models and calculate the accuracy of the prediction under a +-1 range and a calculated range based on the groupings found in the variable *scghq2_dv*. The code for this function can be found in Section 10.

To summarize what the function is doing in order of execution:

1. gathers the predictions for all of the models calculated by the **autoModel** function.
2. creates predictions for the K-Means and kNN model within a +-1 range
3. add +-1 and GHQ ranges to the predictions **autoModel** function calculated for CART, OLR & ENR.
4. Calculates the prediction accuracy within the given ranges for all of the models and adds returns the new prediction results.

4 Understanding Society Analysis

Here we will go though a run of the function where we create models to predict the GHQ score of a participant observed in the Understanding Society data. For this run we will be looking at different Understanding Society data-sets in different ways/joins. This will be the largest test of the automodel function in this dissertation since the Understanding Society data is rather large and we use multiple different joins between data-sets. To make this entire run reproducible, all models will have the function variable **randomSeed** set as an integer (see Section 3.6.6)

4.1 Data Breakdown

Firstly, we have to load in all the different data-sets as descried in Section 2 Understanding Society.

- **b_income**: Loaded in as **w2income**
- **b_indresp**: Loaded in as **w2indresp**
- **c_income**: Loaded in as **w3income**
- **c_indresp**: Loaded in as **w3indresp**
- **xindresp_ns**: Loaded in as **mixNurse**
- **xlabblood_ns**: Loaded in as **mixBloodData**

Then after some processing (this will be covered in Section 4.2) Table 20 contains all the different data-sets which we wish to analyze.

Data-set	No. Obs	No. Vars	GHQ Included?	Wave
w2indresp	54569	1653	TRUE	2
w2Merge	33364	1693	TRUE	2
w2MergeNurse	10921	2032	TRUE	2
w2MergeNurseBlood	6926	2065	TRUE	2
w3indresp	49692	3059	TRUE	3
w3Merge	30487	3099	TRUE	3
w3MergeNurse	3472	3438	TRUE	3
w3MergeNurseBlood	2336	3471	TRUE	3
wShared	104261	1187	TRUE	2, 3
wSMerge	63851	1227	TRUE	2, 3
wSMergeNurse	14393	1553	TRUE	2, 3
wSMergeNurseBlood	9262	1586	TRUE	2, 3
mixNurse	20699	355	TRUE	2, 3
mixNurseBlood	13247	388	TRUE	2, 3

Table 20: Analyzed Understanding Society data-sets

- **w2indresp**: The same as **b_indresp**, described in Section 2.1.1.
- **w2Merge**: This is a join between the transformed **w2income** (see Section 4.1) and **w2indresp** on the variable **pidp**.
- **w2MergeNurse**: This is a join between **w2Merge** (see previous) and **joinSurveyData** (see Section 4.1) on the variables **pidp** & **wave**.
- **w2MergeNurseBlood**: This is a join between **w2MergeNurse** (see previous) and **mixBloodData** on the variables **pidp** & **wave**.
- **w3indresp**: The same as **c_indresp**, described in Section 2.1.1.
- **w3Merge**: This is a join between the transformed **w3income** (see Section 4.1) and **w3indresp** on the variable **pidp**.
- **w3MergeNurse**: This is a join between **w3Merge** (see previous) and **joinSurveyData** on the variables **pidp** & **wave**.
- **w3MergeNurseBlood**: This is a join between **w3MergeNurse** (see previous) and **mixBloodData** on the variables **pidp** & **wave**.
- **wShared**: This is a union of the shared variables across **w2indresp** & **w3indresp**.
- **wSMerge**: This is a union of only the shared variables across **w2Merge** & **w3Merge**.
- **wSharedNurse**: This is a join between **wSMerge** (see previous) and **joinSurveyData** on the variables **pidp** & **wave**.
- **wSharedNurseBlood**: This is a join between **wSMergeNurse** (see previous) and **mixBloodData** on the variables **pidp** & **wave**.
- **mixNurse**: The same as **xindresp_ns**, described in Section 2.1.1.
- **mixNurseBlood**: This is a join between **mixNurse** and **mixNurseBlood** on the variables **pidp** & **wave**.

4.2 Data Processing

Before we can run the data though our function, there are quite a few steps we must take that are unique to the GHQ data.

Firstly, we need to deal with the encoding of missing data within the Understanding Society data (see Section 2.1.2). For this run, we are taking a blanket approach by encoding all values to NA. This is done using the function specified in Section 3.7.1.

Next, we have some small adjustments:

1. We create a version of the **mixNurse** which doesn't include any GHQ data. This is so when it comes to joining **mixNurse** to waves 2 & 3 data respectively we don't duplicate the GHQ data. This new data-set is called **joinSurveyData**

2. waves 2 & 3 data (**w2indresp** & **w3indresp**) start their variable names with **b_** and **c_** respectively. To ensure joins can be preformed correctly we must strip **b_** and **c_** from the start of all variable names in **w2indresp** & **w3indresp**.

3. To provide a base for the joins to occur, we must add a new **wave** variable to both **w2indresp** and **w3indresp** to highlight which wave the data is from. This is achieved though the following two lines of R:

   ```
   w2indresp\$wave = 2
   w3indresp\$wave = 3
   ```

Next, we need to transform the **w2income** & **w3income** data into a form which makes the **pidp** unique per row (**pidp** as a primary key). Using the R package **sqldf** we are able to run a SQL query which achieves this result. With a for loop, we are able to automate the process of generating the SQL query, the function used for this transformation was mentioned in Section 3.7.2.

Lastly, we need to gather all the shared variables across **w2indresp** & **w3indresp** and union them into one data-set which we will call **wShared**. This is done by running the following code in R:

```
w2Shared =
      w2indresp[,names(w2indresp)[names(w2indresp) %in% names(w3indresp)]]
w3Shared =
      w3indresp[,names(w2indresp)[names(w2indresp) %in% names(w3indresp)]]
wShared =
      rbind(w2Shared,w3Shared)
```

The same procedure above is then carried out on **w2Merge** & **w3Merge**, which replace **w2indresp** and **w3indresp** respectively, to create the data-set **wSMerge**

Our data is now ready to merge into all the different data-sets as specified in Section 4.1.

Before running all the joins though the function, we are do some initial analysis on the variables that make up the GHQ data we are interested in. To achieve this analysis per join efficiently, we will use the function ghq_analyze (see Section 3.7.3) to automate though all the joins. All of the results from this procedure can be found in Section 10

Lastly we will run the function ghq_clean_move on all the data-sets (see Section 3.7.4).

4.3 Evaluation Method

Thought-out the results gathered from Understand Society's data, we will be using some terms to explain different prediction intervals:

- **Exact**: This means the accuracy of the model when predicting the GHQ score exactly. This means that if the model predicted a score of 6, the model is correct only when the real GHQ score is 6.

- **+-1 Range**: Since a GHQ score of 6 is rather similar to a score of 7 or 8, it's worth us seeing how well our models predict within the +-1 Range. This means if the model predicts a GHQ score of 6, the model is correct when the real GHQ score is either 6,7 or 8

- **Conf. Interval**: This is a calculated 95% confidence (Mean) interval only available with the Ordinary Linear Regression model, please see Section 3.3.3.3. Model Evaluation Metrics. When the model predicts a GHQ score of 6, the model is correct when the prediction falls within a 95% confidence interval range of the value 6.

- **GHQ Range**: This is a calculated interval using the **scghq1_dv** and **scghq2_dv** variables, please see Section 3.3.3.3. Model Evaluation Metrics for more. When the model predicts a GHQ score of 6, the model is correct when the prediction values within +-1.42 of 6.

For each model in each data join, we will compare the accuracy of predictions with the following initial models all models have been calculated using the GHQ score from **w2indresp**. We only use **w2indresp** since tests preformed in Section 2 Understanding Society showed the data to be very similar for GHQ score and we are only looking for a rough benchmark to judge our models predictions accuracy by. Please see Table 21 for the benchmarks we will be using.

Model	Exact	+-1 Range
Random Choice (RC)	0.027	0.0796
Mode Choice (MC)	0.1089	0.3033

Table 21: Benchmark Predictions Accuracies

- **Random Choice**: This is the chance of getting the correct GHQ score by complete random.
  - **Exact Random Choice** is simply 1 in 37 (where 37 is the number of possible GHQ scores)
  - **+-1 Range Random Choice** is very close to 3 in 37 however we have to consider what happens when we randomly pick a score such as 0 or 36. It doesn't make sense to say *accept 0 if GHQ score is -1* since a GHQ score of -1 doesn't exist, same goes for *accept 36 if GHQ score is 37* since a GHQ score of 37 doesn't exist. Therefore when

considering all correct cases, we have to remove the bottom and top tail of the acceptable range. To explain further please see the below calculation.

$$\frac{3}{37} * \frac{37}{37} = \frac{111}{1369}$$

This calculation is taking the chance of scoring within a +-1 range and multiplying it by the number of levels there are in the GHQ score. This gives us the fraction $\frac{111}{1369}$ which tells us that there are 111 correct cases for the GHQ score as a whole. To remove our bottom and top tail from this consideration, we do the following calculation.

$$\frac{111}{1369} - \frac{2}{1369} = \frac{109}{1369}$$

This removes the bottom and top tail cases (which were *accept 0 if GHQ score is -1* and *accept 36 if GHQ score is 37*). The leftover fraction, $\frac{109}{1369}$ is the correct amount of cases we should consider for the GHQ score and therefore is used as out benchmark.

- **Mode Choice**: This is the chance of getting the correct GHQ score if you always selected the mode.
  - **Exact Mode Choice**: The mode of our GHQ score is 6 with 4728 observations. Our prediction accuracy is then calculated by taking the number of observations of our mode and dividing by the total number of observations: $\frac{4728}{43423}$
  - **+-1 Range Mode Choice**: The mode +-1 Range is (6,7,8) with total observations of 13169. Our prediction accuracy is then calculated like so $\frac{13169}{43423}$

Applying the GHQ range to the Random Choice (RC) and Mode Choice (MC) models makes little sense since the GHQ score only takes integer values however the predictions from regression models predict in a continuous space (which gives us reason to check the GHQ range).

To then gather our prediction accuracies for the *+-1 range* for all models and the *GHQ range* for regression models we run each join though the function predGHQadd (see Section 3.7.5). This then gives us all of the accuracies needed to compare our models.

4.4 Run Results

4.4.1 Introduction

The following is what we expect to see in the console for each join when running though the **autoModel** function:

- Console output telling us the changes that are made to the data for it to be cleaned
- K-Means model with prediction results
- kNN model with prediction results and training data used
- CART model with prediction results
- Ordinary Linear Regression model with prediction results
- Elastic Net Regression model with a list of fits attempted & prediction results (based on the best model)
- Console output of the model evaluation metrics calculated for each of the models above

Since we are working with lots of Understand Society data which are derived from the same data sources (See Section 4.1) we will only go over the results gathered for each prediction model together per data-set. Full runs of each data-set can be found in Section 10.

4.4.2 w2indresp

Our run of the data-set w2Merge run successfully in 2432 seconds (41 minutes) and produced all of the models correctly.

In Table 22 we have all of the prediction accuracies calculated per model. We can see that all the models had a better Exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 16% accuracy. When predicting in a +-1 range, the ENR model is best with a 44% accuracy and when predicting in a GHQ range the CART model is best with a 57% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1624	NA	0.4438	0.5541
OLR	0.1607	0.3452	0.4251	0.5326
kNN	0.1527	NA	0.3681	NA
CART	0.1464	NA	0.4136	0.5746
K-Means	0.1227	NA	0.3349	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 22: w2indresp Predictions

4.4.3 w2Merge

Our run of the data-set w2Merge run successfully 2385 seconds (40 minutes) in and produced all of the models correctly.

In Table 23 we have all of the prediction accuracies calculated per model. We can see that all the models had a better Exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 18% accuracy. When predicting in a +-1 range, the ENR model is best with a 47% accuracy and when predicting in a GHQ range the CART model is best with a 58% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1764	NA	0.4691	0.5706
kNN	0.1714	NA	0.3655	NA
OLR	0.1659	0.3809	0.4515	0.5441
K-Means	0.1298	NA	0.3422	NA
CART	0.1279	NA	0.4168	0.5843
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 23: w2Merge Predictions

4.4.4 w2MergeNurse

Our first run of **automodel** using **w2MergeNurse** threw the us a Observation to Variable ratio error (See Section 3.2.8) since the calculated ratio was 2.31. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.1. This run then gave use a different error related stating that there was too much multicollinearity in the model (see Section 3.4.4). To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.1 and a **corrConfLevel** = 0.5. This run managed to compile successfully in 544 seconds (9 minutes) and outputted all the desired models.

In Table 24 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 18% accuracy. When predicting in a +-1 range, the ENR model is best with a 48% accuracy and when predicting in a GHQ range the ENR model is best with a 59% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1808	NA	0.4777	0.5857
OLR	0.1714	0.4049	0.4601	0.5575
CART	0.1667	NA	0.4695	0.5305
kNN	0.1514	NA	0.3345	NA
K-Means	0.1385	NA	0.3563	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 24: w2MergeNurse Predictions

4.4.5 w2MergeNurseBlood

Our first run of **automodel** using **w2MergeNurseBlood** threw the us a Observation to Variable ratio error (See Section 3.2.8) since the calculated ratio was 1.36. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.05. This run managed to compile successfully in 3217 seconds (54 minutes) and outputted all the desired models.

In Table 25 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 17% accuracy. When predicting in a +-1 range, the ENR model is best with a 46% accuracy and when predicting in a GHQ range the ENR model is best with a 56% accuracy (CART was very close to ENR).

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1697	NA	0.4627	0.5577
OLR	0.1652	0.4197	0.431	0.5396
kNN	0.1538	NA	0.3507	NA
K-Means	0.1349	NA	0.3551	NA
CART	0.1278	NA	0.3812	0.5554
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 25: w2MergeNurseBlood Predictions

4.4.6 w3indresp

Our first run of **automodel** using **w3indresp** threw the us a VIF error (See Section 3.4.4) since their was to much multicollinearity in the data. To resolve this issue, we re-ran **automodel** with a **corrConfLevel** value of 0.5. This run managed to compile successfully in 1542 seconds (26 minutes) and outputted all the desired models.

In Table 26 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 18% accuracy. When predicting in a +-1 range, the ENR model is best with a 45% accuracy and when predicting in a GHQ range the CART model is best with a 55% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.178	NA	0.4482	0.5397
OLR	0.1716	0.361	0.4317	0.5261
kNN	0.1344	NA	0.3117	NA
CART	0.1265	NA	0.3796	0.5482
K-Means	0.1249	NA	0.3315	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 26: w3indresp Predictions

4.4.7 w3Merge

Our first run of **automodel** using **w3Merge** threw the us a VIF error (See Section 3.4.4) since their was to much multicollinearity in the data. To resolve this issue, we re-ran **automodel** with a **corrConfLevel** value of 0.5. This run managed to compile successfully in 1381 seconds (23 minutes) and outputted all the desired models.

In Table 27 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 16% accuracy. When predicting in a +-1 range, the ENR model is best with a 46% accuracy and when predicting in a GHQ range the ENR model is best with a 57% accuracy (The CART model was close to ENR).

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1587	NA	0.46	0.5747
CART	0.1564	NA	0.4079	0.5736
OLR	0.1518	0.4508	0.4206	0.5226
K-Means	0.133	NA	0.3413	NA
kNN	0.1228	NA	0.314	NA
Mode Choice (MC)	0.1089	NA		NA
Random Choice (RC)	0.027	NA		NA

Table 27: w3Merge Predictions

4.4.8 w3MergeNurse

Our first run of **automodel** using **w3MergeNurse** threw the us a Observation to Variable ratio error (See Section 3.2.8) since the calculated ratio was 0.54. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.009. This run then threw us a Observation to Variable ratio error again (this one however, occurred after the Lower Level Removal step) since the calculated ratio was 4.97. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.009 and a **obsPerLevel** value of 4. This run threw us a error stating that VIF (see Section 3.4.4) couldn't run since there was too much multicollinearity in our data. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.009, a **obsPerLevel** value of 4 and a **corrConfLevel** of 0.5. This run managed to compile successfully in 919 seconds (15 minutes) and generated all of the models.

In Table 28 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 16% accuracy. When predicting in a +-1 range, the ENR model is best with a 45% accuracy and when predicting in a GHQ range the ENR model is best with a 52% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1705	NA	0.4474	0.5156
CART	0.1605	NA	0.4148	0.5114
K-Means	0.1398	NA	0.3339	NA
OLR	0.1335	0.4347	0.3636	0.4574
kNN	0.1307	NA	0.3026	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 28: w3MergeNurse Predictions

4.4.9 w3MergeNurseBlood

Our first run of **automodel** using **w3MergeNurse** threw the us a Observation to Variable ratio error (See Section 3.2.8) since the calculated ratio was 0.4. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.0001. This run then threw us a Observation to Variable ratio error again (this one however, occurred after the Lower Level Removal step) since the calculated ratio was 4.75. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.0001 and a **obsPerLevel** value of 1. This run threw us a error that a good Train Test Split couldn't be found (most likely due to the fact **obsPerLevel** was set to 1). Given that the way to resolve a Train Test Split (see Section 3.3.5) not being found is to increase **obsPerLevel** we had come to a standstill. Since we can't increase **obsPerLevel** anymore (1 was the only value that resolved our issue seen in the 2nd run) and decreasing **naPercent** any lower doesn't make any changes, we must conclude that **automodel** can't run this data-set. The conclusion of these turn of events is that the user should go into the data-set and begin a manual variable selection process to lower the amount of variables included in their data-set.

4.4.10 wShared

Our first run compiled in a time of 2180 seconds (36 minutes) and generated all of the models desired.

In Table 29 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 16% accuracy. When predicting in a +-1 range, the ENR model is best with a 45% accuracy and when predicting in a GHQ range the CART model is best with a 54% accuracy (The ENR model is very close to the CART).

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1607	NA	0.4457	0.5338
OLR	0.1546	0.2773	0.4372	0.5301
kNN	0.1491	NA	0.3521	NA
CART	0.1403	NA	0.3849	0.537
K-Means	0.1112	NA	0.3137	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 29: wShared Predictions

4.4.11 wSMerge

Our first run threw use a VIF error (See Section 3.4.4) since there was too much multicollinearity in the OLR model. To resolve this issue, we re-ran **automodel** with a **corrConfLevel** value of 0.5. This run compile successfully in a time of 581 seconds (10 minutes) and generated all of the desired models.

In Table 30 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 16% accuracy. When predicting in a +-1 range, the ENR model is best with a 43% accuracy and when predicting in a GHQ range the OLR model is best with a 52% accuracy (The ENR model was close to the OLR model).

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1599	NA	0.4272	0.5203
OLR	0.1561	0.2701	0.43	0.5229
CART	0.1379	NA	0.3751	0.5022
kNN	0.1363	NA	0.3213	NA
K-Means	0.1114	NA	0.3108	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 30: wSMerge Predictions

4.4.12 wSMergeNurse

Our first run compiled successfully in a time of 525 seconds (9 minutes) and generated all of the desired models.

In Table 31 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 17% accuracy. When predicting in a +-1 range, the ENR model is best with a 45% accuracy and when predicting in a GHQ range the CART model is best with a 54% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.1658	NA	0.446	0.5383
OLR	0.1532	0.3935	0.4113	0.5194
kNN	0.1522	NA	0.3326	NA
CART	0.1427	NA	0.3799	0.5435
K-Means	0.1333	NA	0.3495	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 31: wSMergeNurse Predictions

4.4.13 wSMergeNurseBlood

Our first run threw a Observation to Variable ratio error (See Section 3.2.8) with a calculated ratio of 4.5. To resolve this issue, we re-ran **automodel** with a **naPercent** value of 0.15. This run compile successfully in a time of 724 seconds (12 minutes) and generated all of the desired models.

In Table 32 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that all the models had a better exact prediction accuracy when compared to the MC and RC models, with Elastic Net Regression being the best with a 17% accuracy. When predicting in a +-1 range, the ENR model is best with a 45% accuracy and when predicting in a GHQ range the CART model is best with a 57% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
ENR	0.168	NA	0.4487	0.5523
OLR	0.1549	0.3783	0.4054	0.5091
K-Means	0.1336	NA	0.3464	NA
kNN	0.1308	NA	0.331	NA
CART	0.1217	NA	0.3994	0.5744
Mode Choice (MC)	0.1089	NA	0.3033	NA
Random Choice (RC)	0.027	NA	0.0796	NA

Table 32: wSMergeNurseBlood Predictions

4.4.14 mixNurse

Our first run compiled successfully in a time of 29 seconds and generated all of the desired models.

In Table 33 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that only our classification models (K-Means and kNN) did better than MC with K-Means having the highest accuracy of 12%. All models still preformed better than RC. When predicting in the +-1 range, Only the kNN model preforms better than the MC model with a 33%, all preformed better than the RC model still. When predicting in the GHQ range, our regression models more preform better than the MC model with ENR being the best with a 37% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
K-Means	0.1236	NA	0.3327	NA
kNN	0.1148	NA	0.2967	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
ENR	0.0949	NA	0.2839	0.3724
OLR	0.0921	0.195	0.2855	0.3652
CART	0.0853	NA	0.2883	0.3612
Random Choice (RC)	0.027	NA	0.796	NA

Table 33: mixNurse Predictions

4.4.15 mixNurseBlood

Our first run compiled successfully in a time of 31 seconds and generated all of the desired models.

In Table 34 we have all of the prediction accuracies calculated per model for the run which managed to compile all of our models. We can see that only our classification models (K-Means and kNN) did better than MC with K-Means having the highest accuracy of 13%. All models still preformed better than RC. When predicting in the +-1 range, Only the K-Means model preforms better than the MC model, getting prediction accuracy of 34% (kNN was very close to MC), all preformed better than the RC model still. When predicting in the GHQ range, our regression models more preform better than the MC model with the CART model being the best with a 36% accuracy.

Model	Exact	Conf. Interval	+-1 range	GHQ range
K-Means	0.1288	NA	0.3424	NA
kNN	0.1107	NA	0.3031	NA
Mode Choice (MC)	0.1089	NA	0.3033	NA
ENR	0.0968	NA	0.2763	0.3546
OLR	0.0951	0.226	0.2802	0.3574
CART	0.09	NA	0.269	0.363
Random Choice (RC)	0.027	NA	0.796	NA

Table 34: mixNurseBlood Predictions

4.5 Conclusions

Overall, we managed to run automodel on all joins of the Understanding Society data expect for **w3MergeNurseBlood**. Some of the joins (for example **w2MergeNurse** and **w3MergeNurse**) required a little tweaking of the **automodel** parameters for a full run to be achieved. The results generally showed that the regression models fitted the data better for predicting **Survived**. The best model for predicting the GHQ score exactly was the ENR model. When predicting within a +-1 range, the ENR model preformed the best. When predicting within the GHQ range, both CART and ENR models seemed to be the best.

When looking at the results and applying our known context of the Understanding Society data (see Section 2.1), we can see some variables that were considered the most useful for predicting the GHQ score. Below we will go over some variables in a general case and not on a variable by variable case (due to the vast amount present).

- **SF-12 Physical and Mental Component Summary**: These are similar questionnaires to the GHQ that asks questions based on the participants physical and mental health. Many different answers and the total result of all of them were deemed significant.

- **Subjective financial situation - Current**: Known as **finnow**, this is a measurement of the participants current financial situation. The levels associated with the answers *Finding it quite difficult* or *Finding it very difficult* were deemed significant.

- **Current economic activity**: Known as **jbstat**, this is a question asked on the participants current financial situation. The level associated with the answer *LT sick of disabled* was deemed significant.

- **usual type of dairy consumption**: Known as **usdairy**, this is a question asked on the participants usual type of dairy consumption. The level associated with the answer *skimmed milk* was deemed significant.

- **Health Condition**: These are question asked about any certain health conditions the participant had. A participant having mentioning health condition not asked by the other health condition questions was deemed significant **hcondn96**

- **1st mentioned important event of year**: This is an event that the participant deemed most significant in the current year. The variable is categorical (97 levels) however was modelled as continuous variable (since **catLevels** was 15) therefore further investigation into the individual levels should be conducted (since when modelled as continuous, was still deemed significant).

- **1st mentioned important event of year**: This is an event that the participant deemed most significant in the current year. The variable is categorical (97 levels) however was modelled as continuous variable (since **catLevels** was 15) therefore further investigation into the individual levels should be conducted (since when modelled as continuous, was still deemed significant).

- **Satisfaction with health, Income and Life**: These are questions asked to a participants about their current satisfaction with health, income and life. Different **sclfsat1**, **sclfsat2** & **sclfsato** levels were deemed significant.
- **Control Over Things at Home**: This is a question completed by the participant that asks how their control over things at home is (known as **schmcont**). The responses *slightly agree* and *moderately disagree.*
- **What Happens in Life is Beyond my Control**: This is a question completed by the participant that asks how if what happens in life is beyond their control. The response *strongly disagree* was considered significant.
- **Different Demands on me Hard to Combine**: This is a question completed by the participant that asks if different demands on them are hard to combine. The responses *moderately agree*, *slightly disagree* and *strongly disagree* were considered significant.
- **In General, I Have Enough Time to do Everything**: This is a question completed by the participant that asks if, in general, they have enough time to do everything. The responses *moderately agree* and *strongly disagree* were considered significant.
- **Sex**: A participants sex, was considered significant.

Overall our **automodel** package has proved to be able to work with large quantities of data. The results can provide a user who's interested in the Understanding Society's data some interesting points to continue further analysis with.

5 Titanic Analysis

This is the results seen when running to following instance of **automodel**

```
titanicData.r = autoModel("Survived", titanicData, randomSeed = 3)
```

We will be covering the initial processing, evaluation methods and the results seen from this run.

5.1 Data Processing

The Titanic data-set didn't require any pre-processing to run. All we have done is encode the **Survived** variable as Didn't Survive if **Survived** was 0 and **Survived** if **Survived** was 1. This was done so that the feature of the dependant variable legend can be seen.

5.2 Evaluation Method

Our dependant variable, **Survived**, is a categorical variable that only takes 2 unique values. For this reason we will evaluate our results by only using the exact prediction accuracy of each model (MSE can be misleading given the regression models predict on a continuous scale). MSE and more is covered in Section 3.4.6.

We will compare the accuracy of predictions with the initial models/statistics in Table 35

Model	Exact
Random Choice (RC)	0.5
Mode Choice (MC)	0.6162

Table 35: Benchmark Accuracies for Titanic

- **Random Choice**: This is the accuracy of predicting if a passenger survived the Titanic (**Survived**) by complete random. This can be viewed as predicting a 1 with a 50% chance and a 0 with a 50% chance.

- **Mode Choice**: This is the accuracy of predicting if a passenger survived the Titanic (**Survived**) by always choosing the mode. This can be viewed as always predicting 0 since the mode passenger didn't survive the Titanic (The mode of **Survived** is 0).

5.3 Run Results

The following is what we expect to see in the console when running the Titanic data-set though the **autoModel** function:

- Console outputted legend telling us how the dependant variable (**Survived** in this case) has been encoded
- Console output telling us the changes that are made to the data for it to be cleaned
- K-Means model with prediction results
- kNN model with prediction results and training data used
- CART model with prediction results
- Ordinary Linear Regression model with prediction results
- Elastic Net Regression model with a list of fits attempted & prediction results (based on the best model)
- Console output of the prediction tables, MSE and prediction accuracies of each model

The data-set was processed and we received a full output from **automodel**. It took a total of 0.69 seconds to process and the results were output into the variable **titanicData.r**. We will go over each model made by **automodel** model and their respective evaluation metrics. The full processing of the data can be found in Section 10.

5.3.1 K-Means

Our K-Means model was created successfully with a total amount of clusters 2. The Within MSE of each cluster is quite similar with Cluster 1 having a smaller MSE than Cluster 2. From having a look at the results table, we can see that both clusters mainly contained observations of passengers that didn't survive (0). This means that our K-Means model will always predict that a passenger didn't survive (**Survived** = 0).

From Table 36, we can see that our K-Means model preforms 9% better than random choice (RC) and 2% worse than mode choice (MC). Overall, not a good model to use for predictions.

Model	Exact
K-Means	0.5926
Compared to RC	0.0926
Compared to MC	-0.236

Table 36: K-Mean Predictions for Titanic

5.3.2 kNN

Our kNN model successfully ran and considered 30 neighbors per testing observation. The MSE of of the model was 0.0113. From having a look at results table/confusion matrix, we can see that:

- The model predicted a passenger didn't survive correctly 104 times
- The model predicted a passenger didn't survive incorrectly 36 times
- The model predicted a passenger did survive correctly 31 times
- The model predicted a passenger did survive incorrectly 6 times

From Table 37, we can see that out kNN model preforms 26% better than random choice (RC) and 15% better than mode choice (MC). Overall a decent prediction accuracy that could be improved by tweaked the number of neighbours appropriately.

Model	Exact
kNN	0.7627
Compared to RC	0.2627
Compared to MC	0.1465

Table 37: kNN Predictions for Titanic

5.3.3 CART prediction model

Our CART model successfully ran and produced the following tree

```
n= 530

node), split, n, deviance, yval
      * denotes terminal node

 1) root 530 128.8472000 1.416981
   2) Sex.male>=0.5 332  56.3855400 1.216867
     4) Age>=-1.143733 305  45.0819700 1.180328
       8) Fare< 0.3436248 268  31.8917900 1.138060
        16) Ticket< 1.555194 255  27.2313700 1.121569
          32) Fare< -0.1525384 210  18.0952400 1.095238 *
          33) Fare>=-0.1525384 45   8.3111110 1.244444
            66) Fare>=-0.06442615 23   1.8260870 1.086957 *
            67) Fare< -0.06442615 22   5.3181820 1.409091
             134) PassengerId< 0.4470703 15   2.4000000 1.200000 *
             135) PassengerId>=0.4470703 7   0.8571429 1.857143 *
        17) Ticket>=1.555194 13   3.2307690 1.461538 *
       9) Fare>=0.3436248 37   9.2432430 1.486486
        18) Name>=0.2926589 11   0.0000000 1.000000 *
        19) Name< 0.2926589 26   5.5384620 1.692308
          38) PassengerId< -1.026823 7   0.8571429 1.142857 *
          39) PassengerId>=-1.026823 19   1.7894740 1.894737 *
     5) Age< -1.143733 27   6.2962960 1.629630
      10) SibSp.1< 0.5 15   3.3333330 1.333333 *
      11) SibSp.1>=0.5 12   0.0000000 2.000000 *
   3) Sex.male< 0.5 198  36.8737400 1.752525
     6) Pclass.3>=0.5 78  19.4871800 1.487179
      12) Fare>=-0.2132273 13   0.9230769 1.076923 *
      13) Fare< -0.2132273 65  15.9384600 1.569231 *
     7) Pclass.3< 0.5 120   8.3250000 1.925000 *
```

The CART model deemed the the top 3 most important variables for modelling/predicting **Survived** are:

- *Sex.male*, measures if the passenger was male or not, importance of 35.5879.
- *Fare*, measures the amount the passenger paid for their ticket, importance of 19.545
- *Age*, measures the age of the passenger, importance of 11.9058

The MSE of the model when predicting the testing data is 0.1396. Looking at Table 38, we can see that the CART model had an exact prediction accuracy of 0.8136 (81%). We can see that out CART model preforms 31% better than random choice (RC) and 20% better than mode choice (MC). Overall a better prediction accuracy that K-Means and kNN and could be improved by further tweaking.

Model	Exact
CART	0.8136
Compared to RC	0.3136
Compared to MC	0.1974

Table 38: CART Predictions for Titanic

5.3.4 Ordinary Linear Regression

The process of generating an Ordinary Linear Regression from the full model to the improved model ran successfully. Our initial full model had an AIC of 531.4729. In Variance Inflation Factor (VIF) Removal, 0 variables were removed since all variables had a VIF score lower than 10. In backwards selection, 14 out of 19 variables were removed since they weren't significant at the 95% confidence level.

Please see Table 39 for the final outputted Ordinary Linear Regression Model. Our final improved model contains 5 variables. Pclass describes the class the passenger was in, SibSp describes how many spouses/ siblings that passenger had aboard the Titanic, other variables are self explanatory.

Variable	Estimate Coef	Std. Error	Significance
(Intercept)	1.9541	0.0384	¡ 2e-16
Pclass.2	-0.2196	0.0496	1.16e-05
Pclass.3	-0.3805	0.0446	¡ 2e-16
Sex.male	-0.4636	0.03652	¡ 2e-16
Age	-0.0869	0.0186	3.76e-06
SibSp.3	-0.334	0.1341	0.0131

Table 39: Ordinary Linear Regression Model Variables

The R^2 value for the model above is 0.3724, the MSE of the model is 0.1526 and the AIC is 521.6136. When predicting, the MSE is 0.128 and, as seen in Table 40, our Exact prediction accuracy is 0.8249 (we ignore the prediction within the confidence interval due to the context of our dependant variable **survived**). Looking at Table 40, we can see that out Ordinary Linear Regression model preforms 32% better than random choice (RC) and 21% better than mode choice (MC). Overall a good model for **Survived** and we have a fair amount of room to tweak this model to achieve a higher prediction accuracy.

Model	Exact
Ordinary Linear	0.8249
Compared to RC	0.3249
Compared to MC	0.2087

Table 40: Ordinary Linear Regression Predictions for Titanic

5.3.5 Elastic Net Regression

Our final model, Elastic Net Regression, ran successfully. In Table 41 we can see the list of variables which our best Elastic Net Regression model considered for prediction/modelling. Similar to the Ordinary Linear Regression model, Pclass (A passengers class), Sex & Age were considered. Unlike Ordinary Linear Regression, Fare was considered and SibSp (passengers amount of spouses/siblings on-board) wasn't.

Variable	Estimate Coef
(Intercept)	1.7498
Pclass.3	-0.1451
Sex.male	-0.4166
Age	-0.0157
Fare	0.0367

Table 41: Elastic Net Regression Model Variables

The best Elastic Net Model chosen by MSE had an α of 1. This model's predictions had an MSE of 0.13. As seen in Table 42, the model had an exact prediction accuracy of 0.8192, slightly worse than Ordinary Linear Regression. Overall a good model of **Survived** however not our best. The model could be improved by tweaking however these is very little that can be done compared to other models.

Model	Exact
Elastic Net	0.8192
Compared to RC	0.3192
Compared to MC	0.203

Table 42: Elastic Net Regression Predictions for Titanic

5.4 Conclusions

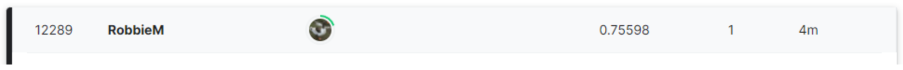
All our models were created successfully and preformed better than the RC and MC models, expect for K-Means. In Table 43 we are comparing all of the models made during the **automodel** run of the Titanic data-set. We can see that the Regression models seem to be best for modelling our dependant variable, **Survived**, with Ordinary Linear Regression being the best fit by Exact Prediction accuracy. If we were to carry on the analysis from here, it would be best to attempt fitting a better Ordinary Linear Regression model.

Model	Exact
OLR	0.8249
ENR	0.8192
CART	0.8136
kNN	0.7627
Mode Choice (MC)	0.6162
K-Means	0.5926
Random Choice (RC)	0.5

Table 43: Elastic Net Regression Predictions for Titanic

Since this data came from Kaggle, it's possible for us to use our best model made (which was our OLR model) on a testing set of data provided by Kaggle. To achieve this, we ran automodel with the same randomSeed but only used 1 test observation (which is done by setting **testPercent** to -1). Then based on the resulting model, we formatted the testing data provided by Kaggle into the formatted which our OLR model accepted. The code that generated this output can be found in Section 10.

As seen in Figure 9, at the time of submission, the OLR model generated by **automodel** scored a 0.75598 (76% accuracy in prediction) and was ranked 12289 out of roughly 14500 entries. The model's prediction accuracy is rather good but compared to other models made with the data, there still a fair amount of improvement that could be done. This represents that even though *automodel* generates useful models, the models shouldn't be used as a final consideration for the data-set.



12289	RobbieM	0.75598	1	4m

Figure 9: Kaggle leader-board Snippet

6 Iris Analysis

This is the results seen when running to following instance of **automodel**.

```
iris.r = autoModel("Species", iris, randomSeed = 1, corrConfLevel = 0.99)
```

We have tweaked the **corrConfLevel** since our brief correlation analysis on the iris data showed there was high correlation between variables which we wish to keep. This means for this run, we are expecting better results from our clustering/ classification models.

6.1 Data Processing

The Iris data-set didn't require any pre-processing to run. Since Iris is in-built into R, we don't need to load in any data and can call it within **automodel** without any hassle.

6.2 Evaluation Method

Our dependant variable, **Species**, is a categorical variable that takes 3 unique values. For this reason we will evaluate our results by only using the exact prediction accuracy of each model (MSE can be misleading given the regression models predict on a continuous scale). To understand these, please see Section 3.4.6.

We will compare the accuracy of predictions with the initial models/statistics seen in Table 44.

Model	Exact
Random Choice (RC)	0.3333

Table 44: benchmark Accuracies Iris

- **Random Choice**: This is the accuracy of predicting the species of iris (**Species**) by complete random. This can be viewed as predicting *Setosa* with a 33% chance, *Versicolor* with a 33% chance and *Virginica* with a 33% chance.

We do not use Mode Choice to predict like in the other runs (See Section 4.3 & 5.2) since Mode Choice (MC) is the same as Random Choice (RC). This is because there is a equal number of each species of iris within the Iris data (50 observations of each).

6.3 Run Results

The following is what we expect to see in the console when running the Iris data-set though the **autoModel** function:

- Console outputted legend telling us how the dependant variable (**Species** in this case) has been encoded
- A legend stating how our dependant variable has been encoded
- Console output telling us the changes that are made to the data for it to be cleaned
- K-Means model with prediction results
- kNN model with prediction results and training data used
- CART model with prediction results
- Ordinary Linear Regression model with prediction results
- Elastic Net Regression model with a list of fits attempted & prediction results (based on the best model)
- Console output of the prediction tables, MSE and prediction accuracies of each model

The data-set was processed and we received a full output from **automodel**. It took a total of 0.56 seconds to process and the results were output into the variable **iris.r**. We will go over each model made by **automodel** model and their respective evaluation metrics. The full processing of the data can be found in Section 10

6.3.1 K-Means

Our K-Means model was created successfully with a total amount of clusters being 3. From having a look at the results table in the console output, we can see that K-means has identified a cluster that only contains observations of **Species** *Setosa*, this being Cluster 3. Cluster 1 and Cluster 2 are also pretty well defined with Cluster 1 mainly being **Species** *Versicolor* and Cluster 2 mainly being **Species** *Virginica*.

From Table 45, we can see that our K-Means model preforms 56% better than random choice (RC). This is a good result and suggest that the data does indeed naturally cluster per iris species.

Model	Exact
K-Means	0.8933
Compared to RC	0.56

Table 45: K-Mean Predictions for Iris

6.3.2 kNN

Our kNN model successfully ran and considered 3 neighbors per testing observation. The MSE of of the model was 0.1053. From having a look at results table/confusion matrix, we can see that:

- The model predicted *setosa* correctly 15 times
- The model predicted *versicolor* correctly 8 times
- The model predicted *virginica* correctly 13 times
- The model predicted *virginica* in-correctly 2 times

From Table 46, we can see that out kNN model preforms 61% better than random choice (RC). Overall a very good model for *Species* and little needs to be done to improve the model.

Model	Exact
kNN	0.9474
Compared to RC	0.6141

Table 46: kNN Predictions for Iris

6.3.3 CART prediction model

Our CART model successfully ran and produced the following tree

```
n= 112

node), split, n, deviance, yval
      * denotes terminal node

 1) root 112 71.964290 2.017857
   2) Petal.Length< -0.7409513 35  0.000000 1.000000 *
   3) Petal.Length>=-0.7409513 77 19.220780 2.480519
     6) Petal.Length< 0.5619447 37  0.972973 2.027027 *
     7) Petal.Length>=0.5619447 40  3.600000 2.900000
      14) Petal.Length< 0.7318877 11  2.545455 2.636364 *
      15) Petal.Length>=0.7318877 29  0.000000 3.000000 *
```

The CART model gave the importance in each variable for predicting **Species** as follows:

- **Petal.Length**, the length of the petal in centimeters, importance of 68.44586
- **Petal.Width**, the width of the petal in centimeters, importance of 64.71597
- **Sepal.Length**, the length of the sepal in centimeters, importance of 46.59445
- **Sepal.Width**, the width of the sepal in centimeters, importance of 28.65781

The MSE of the model when predicting the testing data is 0.0406. Looking at Table 47, we can see that the CART model had an exact prediction accuracy of 0.9474 (94%). We can see that out CART model preforms 61% better than random choice (RC). Overall a very good model for *Species* and little needs to be done to improve the model.

Model	Exact
CART	0.9474
Compared to RC	0.6141

Table 47: CART Predictions for Iris

6.3.4 Ordinary Linear Regression

The process of generating an Ordinary Linear Regression from the full model to the improved model ran successfully. Our initial full model had an AIC of -12.4783. In Variance Inflation Factor (VIF) Removal, **Petal.Length** was removed since it had a VIF score of 33.0637. The resulting model after VIF had an AIC of 2.4362. In backwards selection, **Sepal.Width** and **Sepal.Length** were removed since they weren't significant at the 95% confidence level.

Please see Table 48 for the final outputted Ordinary Linear Regression Model. Our final improved model contains only Petal.Width. Since this model only has on variable in it, we can say that there is no multicollinearity between predictor variables.

Variable	Estimate Coef	Std. Error	Significance
(Intercept)	2.01155	0.02379	¡ 2e-16
Petal.Width	0.79787	0.02492	¡ 2e-16

Table 48: OLR Model Variables

The R^2 value for the model is 0.9022, the MSE of the model is 0.0623 (based on the training set) and the AIC is 12.878. When predicting, the MSE is 0.0416 and, as seen in Table 49, our exact prediction accuracy is 1 (we ignore the prediction within the confidence interval due to the context of our dependant variable **Species**). Looking at Table 49, we can see that out Ordinary Linear Regression model preforms 67% better than random choice (RC). Overall the model predicts perfectly, however, this may be because of the Train Test Split or the number of observations so the accuracy should be taken with caution.

Model	Exact
Ordinary Linear	1
Compared to RC	0.6667

Table 49: Ordinary Linear Regression Predictions for Titanic

6.3.5 Elastic Net Regression

Our final model, Elastic Net Regression, ran successfully. In Table 50 we can see the list of variables which our best Elastic Net Regression model considered for prediction/modelling. In general, ENR considered all of the variables when predicting **Species**

Variable	Estimate Coef
(Intercept)	2.0068
Sepal.Length	0.0267
Sepal.Width	-0.0662
Petal.Length	0.3283
Petal.Width	0.3814

Table 50: Elastic Net Regression Model Variables

The best Elastic Net Model chosen by MSE had an α of 0.1. This models predictions had an MSE of 0.0455. As seen in Table 51, the model had an exact prediction accuracy of 0.9737 (97%), slightly worse than Ordinary Linear Regression. Overall seemly good model for *Species* however since the model has considered all the predictor variables, we would assume the model has high multicollinearity and therefore varies drastically with small changes.

Model	Exact
Elastic Net	0.9737
Compared to RC	0.6404

Table 51: Elastic Net Regression Predictions for Iris

6.4 Conclusions

All our models were created successfully and preformed better than the RC model. In Table 52 we are comparing all of the models made during the **automodel** run of the Iris data-set. Surprisingly, our best model for prediction was Ordinary Linear Regression (OLR) with a model only using **Petal.Width**. Our classification models, K-Means and kNN still did well and showed that the data does seem to cluster based on **Species**. Generally, to better improve our calculated accuracies, more observations of iris flowers would be needed.

Model	Exact
OLR	1
Elastic Net	0.9737
CART	0.9474
kNN	0.9474
K-Means	0.8933

Table 52: Elastic Net Regression Predictions for Iris

7 Discussion

Now that we have seen what the **automodel** package is and how it works with different data-sets, we will discuss on why this data and methodology was used. We will also make mention to further improvements that could be done to **automodel** and give reasons why.

7.1 Data

To demonstrate **automodel**, 3 different data-sets were used.

Firstly, the Understanding Society data-set (see Section 2.1) was the main data source used within this dissertation. This is because of various reasons:

- When initially working on this dissertation, the goal was to fit a model that best described the GHQ score within Understanding Society. Naturally this meant **automodel** was first built using this data as a training ground.
- The Understanding Society data is incredibly large with data sources reaching up to 400MB+ in size. This gives us very large data-sets for us to work with meaning that when building **automodel**, we can account for issues that occur when using large amounts of data.
- The people that assisted in providing context to the Understanding Society data had academic links to the data-set. This meant that when building **automodel**, their wealth of knowledge could be used to understand the context around the large amount of data.

Secondly, the Titanic data-set (see Section 2.2) was used to show the how data assumed to have a linear relationship looks when using the **automodel** function. This is because of various reasons:

- Since the data-set was provided by Kaggle, the results could be uploaded to Kaggle leader-board to compare how the models generated compared with other built models submitted to Kaggle.
- The data-set included data which would test the cleaning, transforming and modelling functions to a good degree. Example: The variable **Name** is a string variable were each observation is a passengers name. This **Name** would test our cleaning methods and then our modelling methods.

- Due to it's smaller size compared to Understanding Society, this data-set became very useful to quickly preform tests on the **automodel** package after an update. A average run of the Titanic data-set took about 1 second (compared to Understanding Societies 20+ minutes)

Lastly, the Iris data-set (see Section 2.3) was used to demonstrate how well the **automodel** package represents data that is known to already have natural clustering.

- The data-set is easily recognized which gives readers a good benchmark to compare the results of their individual analysis to the **automodel** package.
- Since the variables were highly correlated with each other, this run was designed to favour our classification models and see if they were able to out preform the regression models in the model evaluation metrics.
- Due to it's smaller size compared to Understanding Society, runs with this data can be preformed very quickly, giving the reader another data-set like the Titanic data-set (see Section 2.2) to play around with.

7.2 Methodology

In this dissertation lots of different cleaning, transforming and modelling methods have been used within **automodel**. The reason for why these methods were chosen is all relatively the same; during university or outside study, these were the methods that were used the most/ were already known at a very basic level. Further investigation has been done on these methods so that an in depth understanding of them was learnt. Therefore instead of discussing why these certain methods were used, we will go over some criticisms of their usage in this dissertation.

7.2.1 Data Type Checks

Our method of casting string variables which have more than **catLevels** levels to a numerical variable lead to some strange results. Example: In our Titanic run, **Name** is considered a significant variable when considered as a continuous variable. This result isn't the correct way to model the variable **Name** and given our prior context on it, could cause and over-fit model.

7.2.2 Missing Value Imputation

Our method of Single Imputation using CART keeps time complexity low and ensures the function runs smoothly, however, this method doesn't account for the known variation that comes with imputing missing values. To account for this Multiple Imputation is generally used (more on this in Section 7.3).

7.2.3 Low Level Removal

Even though a user can adjust the value for this variable, the default fun of automodel will discriminate against factor levels which have low observations. Depending on the data-set parsed, this may be an undesired bias.

7.2.4 Correlation Checks

When checking for correlation, we discriminate against variables that come earlier in the data-set. This gives the user the functionality to have some manual input on what variables get removed at this stage however it comes at a trade-off. To achieve our result we now have a bias in our correlation selection method which could lead to more variables being removed than necessary. Another method that could of been introduced in for Section 3.2.10 is a check on the correlation coefficients between the predictor variables and the dependant variable. Then, another selection method could be done of these correlation coefficients to refine the predictor variables chosen for analysis. Example: We keep predictor variables if they have a correlation coefficient with the dependant variable of 0.5 or higher.

7.2.5 Dummy Variables

If we have a categorical variable, we won't create a dummy variable to represent the first ordered level since it's represented using the other dummies made (if all other dummies are 0, this represents the first ordered level). This reduces the amount of variables within the data-set analyzed. This may be an issue though when preforming variable selection in regression models (like OLR). Example: the first ordered level in a categorical with more than 2 levels is very significant in predicting our dependant variables whilst the other levels aren't. When encoding, our first level won't be represented as a single dummy variable but instead as a combination of all other dummies made from the same variable. This then makes the significance/importance of that first ordered level spread across the combination of dummy variables that describe it which can lead to this highly significant level to be considered insignificant due to it's other levels.

7.2.6 Factorization

The **autoModel** function decides on if a variable is a factor based on the function variable **catLevels**, which by default is 15. If there is a categorical variable within the data that has more levels than **catLevels**, it will be treated as continuous.

7.2.7 Train Test Split

To be able to get a prediction accuracy for our models, we split our data-set into a training and testing set. Since there's a random element to this process the resulting training and testing sets can have a large impact on the model created, causing variation in results each run. To minimise this, k-fold cross validation could be done to see how the models preform for each fold (more on this in Section 7.3)

7.2.8 K-Means

In our K-Means model, we decide clusters based on the number of unique levels within the dependant variable (if the dependant variable is continuous, **catLevels** clusters are made). This can lead to a bad representation of the data as clusters. Instead, the number of clusters should be decided though an evaluation method (More on this in Section 7.3). Our usage of K-Means as a prediction model means we can compare the models to the results of other models used by **automodel** (though exact prediction accuracy). Even though useful, this is an unorthodox way of using K-Means and the primary focus should be on the natural groups/clusters that form within the data-set. When modelling a continuous variable, our results tables and our exact prediction accuracies provide misleading results.

7.2.9 kNN

For our kNN model, we choose the number of neighbours to consider by taking the square root of the number of observations present ($\sqrt{n}$). This is a good estimate for the number of neighbours needed however it may not be the best. Instead, a selection method for k should be introduced where k is decided by calculating and error rate per k value and picking the k with the lowest error rate. When modelling a continuous variable, our results tables and our exact prediction accuracies provide misleading results.

7.2.10 Ordinary Linear Regression

For our backwards selection process, we choose our variables based on their statistical significance when modelling our dependant variable. Doing this can lead to having variables which are all considered statistically significant but the overall model not considered statically significant (more on this in Section 7.3). Also, our selection process could be more efficient if the selection process used was step-wise selection. This could increase accuracy but due to the time complexity this method brings, we chose to use backwards selection instead.

7.2.11 Elastic Net Regression

When choosing out λ to use in our penalty function, we preform cross validation then pick the λ which the smallest value. Instead, we could of potentially picked the λ of the smallest model that is within 1 standard error of the smallest value of λ. We chose not to since picking the smallest λ after cross validation is simpler to understand and produces the most accurate model (that comes with a trade off of more dimensions).

7.3 Future Improvements

Since there's been a time constraint on the completion of this dissertation, there are still many ways which the **automodel** package could be improved. Below I will talk about these in brief detail:

- **automodel** could also find the best value for k within K-Means clustering. By using method such as *elbow* or *average silhouette* we could instead find the best value for k in K-Means instead of it being an arbitrary amount that can be tuned by the user.

- In Ordinary Linear Regression, we improve our models using backwards selection on the statistical significance of each predictor variable at the **modelSigLevel** confidence level. If our final model includes lots of predictor variables, the entire model will be considered statistically significant at the product of all the confidence levels in each predictor variable. For example, if a model contains two variables that are statistically significant at exactly the 95% confidence level, the model will be considered statistically significant at the $95\% \times 95\% = 90.25\%$ confidence level. For this reason, using statistical significance as our metric for selection can lead to some models with overall, aren't that statistically significant.

  To get around this issue, we should use another metrics for our selection process. One example would be using the Akaike information criterion (AIC) to select our models instead. Instead of selecting a model based on the statistical significance of the variables included, the model is picked on the best overall performance at predicting the dependant variable (based on the AIC).

- In Ordinary Linear Regression, there is a lack of investigation into interactions between variables. To improve this, we could make the Ordinary Linear Regression model consider the interactions between predictor variables.

- Further investigation could be done on if the relationships between the dependant variables and predictor variables aren't linear. modelling terms as their squares, x^2, or as exponential, $\exp^x$, could lead to us finding further interesting relationships within the users data.

- For large data-sets, running the **autoModel** function can take quite a long time. We could improve this run-time by preforming some general quality of life improvements on the code which means that large calculations don't have to occur as often (such as in VIF or backwards selection, we don't recalculate the model after each variable removal, instead in jumps of 2 or 3).

- When processing the full Ordinary Linear Regression model, we have a step where we have to remove variables that the function deemed as unusable/ as NA. It's unknown the exact reasons why some variable appear like this since it's quite contextual to the data inputted, thare are some theories behind why though:

  – After correlation checks, some variables are considered so highly correlated, the lm() function doesn't model them. It's hard to know if this is the case though since there doesn't seem to be a defined limit to the correlation lm() checks for.

  – Some variables in a data-set can be composites of other variables in the data-set. For example if you have a variable called *score* that was the sum of 10+ other variables within the data-set, you may not get high correlation between *score* and these other variables however score can still turn up NA after calling lm() (creating a OLR model). It was very hard to think of a general function to tackle this issue and will be considered in the future

- There is no formal checks for outliers within the data. Besides the removal of categorical variable levels with low observations (see Section 3.2.7), we do not remove any observations if they are considered to be an outlier. This can results in very poor predictions when modelling and therefore should be accounted for correctly (using leverage or the interquartile range).

- Out function doesn't address the normally of the data/ how normally distributed each variable is. An option for a user to address this situation should be included as some model assumptions may want their variables to be considered as normally distributed.

- In testing, it's possible for a user to be shown an error message telling then to increase the **automodel** variable **obsPerLevel** (if a user couldn't find a good Train Test split, this would occur). After doing so the user could then be hit with another error message telling them to lower **obsPerLevel** (After low level observation removal, there may not be enough observations to continue). This can make the journey confusing and also seemingly impossible to run and therefore we should consider manually tuning parameters in those cases where an error occurs.

- Instead of having abstract numerical labelling in the clustering/classification models for categorical dependant variables, we should give them understandable labels. Say we were trying to predict a variable that as originally "Survived" and "Not Survived", we should aim to label our prediction tables as such (our current solution is outputting a legend for the user to follow at the start of the function).

- When predicting a binary variable using linear regression can lead to some misleading results (predicting 0.33 instead of 0 or 1). Instead we should be fitting a logistic regression model to predict variables of this nature.

- Instead of removing levels from categorical variables that have low observations, we could instead consider other options. One idea would be merging similar levels together if they have low observations, this is quite a hard task to achieve without any context of the data-set.

- To decrease the variance in the resulting models, k-folds cross validation could be done on the Train Test split. This would result in **automodel** modelling the data k times on k equally distributed folds of the data-set, then, the models results could be collated to get an average and variance across the results.

- K-Means and kNN aren't very good at representing the predictions of continuous variables. The predictions tables aren't going to format correctly since it's difficult to predict a continuous variable exactly. The SSR results for both algorithms are still valuable given the context of a continuous variable prediction.

- There is a lack of graphical output from the function. It would be hard to decide what graphs to include since there are a-lot of features which are worthy of plotting. We don't want to plot to much since it could end up breaking up the flow of the function's run-time. It's a hard one to decide on what's best to do

- The make future updates and collaboration easier, the **automodel** package should be re-designed in a modular style. This would mean that each method used would be their own function within the **automodel** package instead of being all bundled in the **autoModel** function. The **autoModel** would then call to these functions in order of best execution.

8 Conclusion

In this dissertation, we have investigated multiple different mathematical methods, created a function to use them all automatically and then analyze the results when this function was applied to different data-sets. This process has demonstrated that it is indeed possible to automate the modelling process. It has also demonstrated a technical understanding of the mathematical principles and concepts behind the techniques used to preform data science and analytics. This knowledge has then been successfully translated to the statistical coding language R where the working package, **automodel**, has been made.

The results from the **autoModel** function showed that even though the model performances were deemed useful, they aren't to be considered the best models for the specific data it's model from. Therefore the best usage of **automodel** is to use it in preliminary analysis to help advise in-depth analysis that utilizes the context of the data-set being analyzed. The audience that could get the most from this package would be those already with analytical knowledge so that the results from **automodel** can be interpreted correctly. If a person was to use **automodel** without this knowledge, there is the risk that the large quantity of output could lead to some poor/ incorrect conclusions. The package also doesn't provide a good learning experience and therefore it's recommended that those with less knowledge in data science and analytics initially preform analysis without **automodel**.

The **automodel** package in it's current stage works, but should undergo a fair amount of refinement. To improve, the features listed in Section 7.3 Future Improvements could be implemented. Hopefully this package will be continually worked on beyond this dissertation to further improve the usefulness for those looking to preform data science and analytics.

Overall this dissertation has been an in-depth learning experience and has generated a function, **automodel**, which can be used as a useful tool for analysis.

9 Bibliography

References

[1] Understanding Society (1978). **"Understanding Society: About the Study"**. Website: https://www.understandingsociety.ac.uk/about/about-the-study

[2] Understanding Society (1978). **"Understanding Society: Survey Timeline"**. Website: https://www.understandingsociety.ac.uk/documentation/mainstage/survey-timeline

[3] Goldberg D (1978). **"GHQ-12: A quick screener for survey use"**. Description: https://eprovide.mapi-trust.org/instruments/general-health-questionnaire#basic_description

[4] Glen_b (2014). **"How to interpret a QQ plot"** Post: https://stats.stackexchange.com/questions/101274/how-to-interpret-a-qq-plot

[5] Kaggle (2019). **"Titanic - Machine Learning from Disaster"** Article: https://www.kaggle.com/c/titanic/data

[6] Edgar Anderson (1988) **"iris: Edgar Anderson's Iris Data"** Documentation: https://www.rdocumentation.org/packages/datasets/versions/3.6.2/topics/iris

[7] Dr Mercedes Torres-Torres; Dr Michael Pound (2019). **"Data Analysis - Computerphile"**. Series start: https://www.youtube.com/watch?v=NxYEzbbpk-4&list=PLJ1bsfrnyoq63ncJB19w3nFm4i4K04mNq

[8]

[9] Isabella Ghement (2019). **"Multiple Imputation by Chained Equations (MICE) Explained"**. Post: https://stats.stackexchange.com/questions/421545/multiple-imputation-by-chained-equations-mice-explained

[10] Stef van Buuren (2012). **"Flexible Imputation of Missing Data, Second Edition"**. Book: https://stefvanbuuren.name/fimd/sec-MCAR.html

[11] Nunan D, Bankhead C, Aronson JK (2019). **"Catalogue of Bias Collaboration: Information bias"**. Catalogue Of Bias 2017: https://catalogofbias.org/biases/information-bias/

[12] Nunan D, Bankhead C, Aronson JK (2017). **"Catalogue of Bias Collaboration: Selection bias"**. Catalogue Of Bias 2017: http://www.catalogofbias.org/biases/selection-bias/

[13] Paul T. Barrett, Paul Kline (1981). **”Observation to Variable Ratio”** Paper: https://www.pbarrett.net/publications/Observation-to-Variable-Ratio-Barrett-and-Kline-1981.pdf

[14] Songhao Wu (2020). **”Multicollinearity in Regression”** Post: https://towardsdatascience.com/multi-collinearity-in-regression-fe7a2c1467ea

[15] Cuemath (2022) **”Transitive Relations”** Page: https://www.cuemath.com/algebra/transitive-relations/

[16] amoeba (2015). **”Making sense of principal component analysis, eigenvectors & eigenvalues”** Post: https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues

[17] Wiki, Unknown (2022). **”Eigendecomposition of a matrix”** Page: https://en.wikipedia.org/wiki/Eigendecomposition_of_a_matrix

[18] Nour Al-Rahman Al-Serw (2021). **”K-nearest Neighbor: The maths behind it, how it works and an example”**. Article: https://medium.com/analytics-vidhya/k-nearest-neighbor-the-maths-behind-it-how-it-works-and-an-example-f1de1208546c

[19] Avinash Navlani (2018). **”KNN Classification Tutorial using Scikit-learn”**. Tutorial: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

[20] Josh Starmer (2019). **”Regression Trees, Clearly Explained!!!”** Video: https://www.youtube.com/watch?v=g9c66TUylZ4

[21] Dr. GP Pulipaka (2016). **”An essential guide to classification and regression trees in R Language”** Article: https://medium.com/@gp_pulipaka/an-essential-guide-to-classification-and-regression-trees-in-r-language-4ced657d176b

[22] Cettt (2019). **”How is Pr(¿—t—) in a linear regression in R calculated?”** Post: https://stackoverflow.com/questions/55969011/how-is-prt-in-a-linear-regression-in-r-calculated

[23] Josh Starmer (2018). **”Machine Learning Fundamentals: Bias and Variance”**. Video: https://www.youtube.com/watch?v=EuBBz3bI-aA

[24] Josh Starmer (2018). **”Regularization Part 2: Lasso (L1) Regression”**. Video: https://www.youtube.com/watch?v=NGf0voTMlcs

[25] Josh Starmer (2018). **”Regularization Part 1: Ridge (L2) Regression”**. Video: https://www.youtube.com/watch?v=Q81RR3yKn30

[26] Josh Starmer (2018). **”Regularization Part 3: Elastic Net Regression”**. Video: https://www.youtube.com/watch?v=1dKRdX9bfIo

[27] Josh Starmer (2018). **”Machine Learning Fundamentals: Cross Validation”**. Video: https://www.youtube.com/watch?v=fSytzGwwBVw

[28] Newcastle University (2022). **”Coefficient of Determination, R-squared”** Site: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html

[29] Alexandre Zajic (2019). **”Introduction to AIC — Akaike Information Criterion”** Article: https://towardsdatascience.com/introduction-to-aic-akaike-information-criterion-9c9ba1c96ced

10 Appendix