Predicting Weather Forecast

Kruger 60 A

Research Skills: Programming with R

Name	u-number	e-mail
Vicky Mekes	u980683	v.j.b.mekes@tilburguniversity.edu
Sema Karan	u924823	s.karan@tilburguniversity.edu
Shelly van Erp	u1266624	s.d.a.vanerp@tilburguniversity.edu

Introduction

Weather forecasting has been attempted to be predicted formally since 19th century (wikipedia). Traditionally, this has been done through physical simulations in which the atmosphere is modeled as a fluid (Holmstrom, Liu & Vo, 2016). The samples from the present state of atmosphere are collected and prediction has been done by computing the numerical equations fluid dynamics & thermodynamics with the samples collected. Later on, a lot work and methods have been used to predict weather forecast with machine learning models since they are more robust to perturbations. In this project, a few different machine learning models are attempt to predict whether it'll rain tomorrow or not and the amount of rain that will rain tomorrow.

Data Description

The dataset used was retrieved from the competition website Kaggle. The original dataset contains 24 different variables about weather in Australia. Every observation represents one day at an Australian city and 20 of the variables represent specific measurements like humidity, wind speed, minimum and maximum temperature and so on. The dataset contains two target variables as presented as a challenge by Kaggle: a binary variable that states if it would rain the next day and a continuous variable that states the amount of rain that would fall the next day.

Research Questions

For the two target variables presented in the previous section, two separate research questions were constructed:

- R1: Which weather features can most accurately predict whether it will rain tomorrow?
- R2: How accurate can weather measurements from today predict the amount of rain that will fall tomorrow?

For both research questions, different types of modeling and feature selection methods were tried to find the optimal solution.

Packages Used

`caret` was used to make a test en training set for our dataset. Furthermore, we build a KNN model to predict research question 1. `tidyverse` is used to benefit from `dplyr` packages which is used to select and made new variables, subsetting or deleting certain unnecessary variables and also benefit from `ggplot2` which is used for visualization both in modelling and EDA parts, besides we followed tidy data principles with " %>% " in `tidyverse` package. `ggfortify` was needed for the PCA analysis. The package assists the `ggplot2` package and makes it possible to use the `autoplot()` function. The function made it possible to easily plot the PCA graph. `leaps` and `MASS` are used by Caret to do the forward and backward feature selection methods. `MLmetrics` was used to get the mean squared error for all linear models tried to answer research question 2, to be able to compare the model performances. `neuralnet` The package neuralnet was used to build a simple neural network to answer question 2 and compare the results with the simple linear regression models. `Boruta` package is built to find important variables based on Boruta algoithm which is a wrapper built around random forest classification algorithm. It tries to bring variables as *important*, *unimportant* and *tentative*. We used the package to answer research question 1.

```
rm(list=ls(all=TRUE))
library(caret)
library(dplyr)
library(tidyverse)
library(ggplot2)
library(ggfortify)
library(leaps)
library(MASS)
library(mice)
library(MLmetrics)
library(neuralnet)
library(Boruta)
```

Data Processing

First the data is loaded and structure is viewed to see what type of variables are available.

```
weather <- read.delim("weatherAUS.csv", sep=",", stringsAsFactors = FALSE)
str(weather) # dataset has int, num and chr values
```

```
## 'data.frame':    145460 obs. of  24 variables:
##  $ Date         : chr  "2008-12-01" "2008-12-02" "2008-12-03" "2008-12-04" ...
##  $ Location     : chr  "Albury" "Albury" "Albury" "Albury" ...
##  $ MinTemp      : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
##  $ MaxTemp      : num  22.9 25.1 25.7 28 32.3 29.7 25 26.7 31.9 30.1 ...
##  $ Rainfall     : num  0.6 0 0 0 1 0.2 0 0 0 1.4 ...
##  $ Evaporation  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ Sunshine     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ WindGustDir  : chr  "W" "WNW" "WSW" "NE" ...
##  $ WindGustSpeed: int  44 44 46 24 41 56 50 35 80 28 ...
##  $ WindDir9am   : chr  "W" "NNW" "W" "SE" ...
##  $ WindDir3pm   : chr  "WNW" "WSW" "WSW" "E" ...
##  $ WindSpeed9am : int  20 4 19 11 7 19 20 6 7 15 ...
##  $ WindSpeed3pm : int  24 22 26 9 20 24 24 17 28 11 ...
##  $ Humidity9am  : int  71 44 38 45 82 55 49 48 42 58 ...
##  $ Humidity3pm  : int  22 25 30 16 33 23 19 19 9 27 ...
##  $ Pressure9am  : num  1008 1011 1008 1018 1011 ...
##  $ Pressure3pm  : num  1007 1008 1009 1013 1006 ...
##  $ Cloud9am     : int  8 NA NA NA 7 NA 1 NA NA NA ...
##  $ Cloud3pm     : int  NA NA 2 NA 8 NA NA NA NA NA ...
##  $ Temp9am      : num  16.9 17.2 21 18.1 17.8 20.6 18.1 16.3 18.3 20.1 ...
##  $ Temp3pm      : num  21.8 24.3 23.2 26.5 29.7 28.9 24.6 25.5 30.2 28.2 ...
##  $ RainToday    : chr  "No" "No" "No" "No" ...
##  $ RISK_MM      : num  0 0 0 1 0.2 0 0 0 1.4 0 ...
##  $ RainTomorrow : chr  "No" "No" "No" "No" ...
```

- **Variable classes**

The variables as presented in the original dataset were all integer, numerical and character values. We decided to convert all variables that contained character values to factors, except for the variable Date. This was done as character values are difficult to use in further analysis to answer the research questions.

```
weather$RainToday <- factor(weather$RainToday)
weather$RainTomorrow <- factor(weather$RainTomorrow)
weather$WindDir3pm <- factor(weather$WindDir3pm)
weather$WindDir9am <- factor(weather$WindDir9am)
weather$WindGustDir <- factor(weather$WindGustDir)
weather$Location <- factor(weather$Location)
```

- **Adding new variables**

As the Date variable is coded like *yyyy-mm-dd*, it was not very helpful to use this variable as a predictor, therefore we decided to break the variable down and make two new variables with it: *Month* and *Season*. First, we added the column Month by taking a substring from the *Date* column, after that we wrote our own function (`conver_season`) to convert a column with month numbers to the Australian season and we saved those in a new column: *Season*.

```
#Creating a new column with only the month number
weather$Season <- substr(weather$Date, 6, 7)
weather$Season <- as.numeric(weather$Season)
weather$Season[weather$Season == 12] <- 0

convert_season <- function(Month) {
  Month[Month <= 2] <- "Summer"
  Month[Month >= 3 & Month <= 5] <- "Autumn"
  Month[Month >= 6 & Month <= 8] <- "Winter"
  Month[!(Month %in% c("Autumn", "Winter", "Summer"))] <- "Spring"
  Month
}

weather$Season <- convert_season(weather$Season)
weather$Season <- factor(weather$Season )
```

- **Missing data**

The dataset contained quite a lot of missing data, especially for four of the variables: *Evaporation, Sunshine, Cloud9am and Cloud3pm.* Values of those four variables were missing for more than one third of the observations. Before deleting those variables from the dataset, we looked at some graphs to see if the variables could be good predictors for the target variables. Although *Cloud9am and Cloud3pm* seemed to be reasonably good predictors we decided to delete all four of them from consideration, as the fraction of data missing was too big.

For the other variables we tried to impute the missing values with multiple imputation using the `mice` package. Unfortunately, due to the size of the dataset the imputation was very computationally expensive so we decided to omit the observations that contained missing values. This still resulted in a dataset of **112.925 observations** (against 145.460 observations from the original dataset), which we think should be enough to get reasonable good results.

```
sapply(weather, function(x) sum(is.na(x))) # shows absolute numbers of missing values
```

```
##          Date      Location       MinTemp       MaxTemp      Rainfall
##             0             0          1485          1261          3261
##   Evaporation      Sunshine   WindGustDir WindGustSpeed    WindDir9am
##         62790         69835         10326         10263         10566
##    WindDir3pm  WindSpeed9am  WindSpeed3pm   Humidity9am   Humidity3pm
##          4228          1767          3062          2654          4507
##   Pressure9am   Pressure3pm      Cloud9am      Cloud3pm       Temp9am
##         15065         15028         55888         59358          1767
##       Temp3pm     RainToday       RISK_MM  RainTomorrow        Season
##          3609          3261          3267          3267             0
```

```
colMeans(is.na(weather)) # shows percentage of missig values per column
```

```
##          Date      Location       MinTemp       MaxTemp      Rainfall
##    0.00000000    0.00000000    0.01020899    0.00866905    0.02241853
##   Evaporation      Sunshine   WindGustDir WindGustSpeed    WindDir9am
##    0.43166506    0.48009762    0.07098859    0.07055548    0.07263853
##    WindDir3pm  WindSpeed9am  WindSpeed3pm   Humidity9am   Humidity3pm
##    0.02906641    0.01214767    0.02105046    0.01824557    0.03098446
##   Pressure9am   Pressure3pm      Cloud9am      Cloud3pm       Temp9am
##    0.10356799    0.10331363    0.38421559    0.40807095    0.01214767
##       Temp3pm     RainToday       RISK_MM  RainTomorrow        Season
##    0.02481094    0.02241853    0.02245978    0.02245978    0.00000000
```

```
weather_clean <- subset(weather, select = -c(Evaporation, Sunshine, Cloud3pm, Cloud9am))
weather_clean <- na.omit(weather_clean)

sapply(weather_clean, function(x) sum(is.na(x)))
```

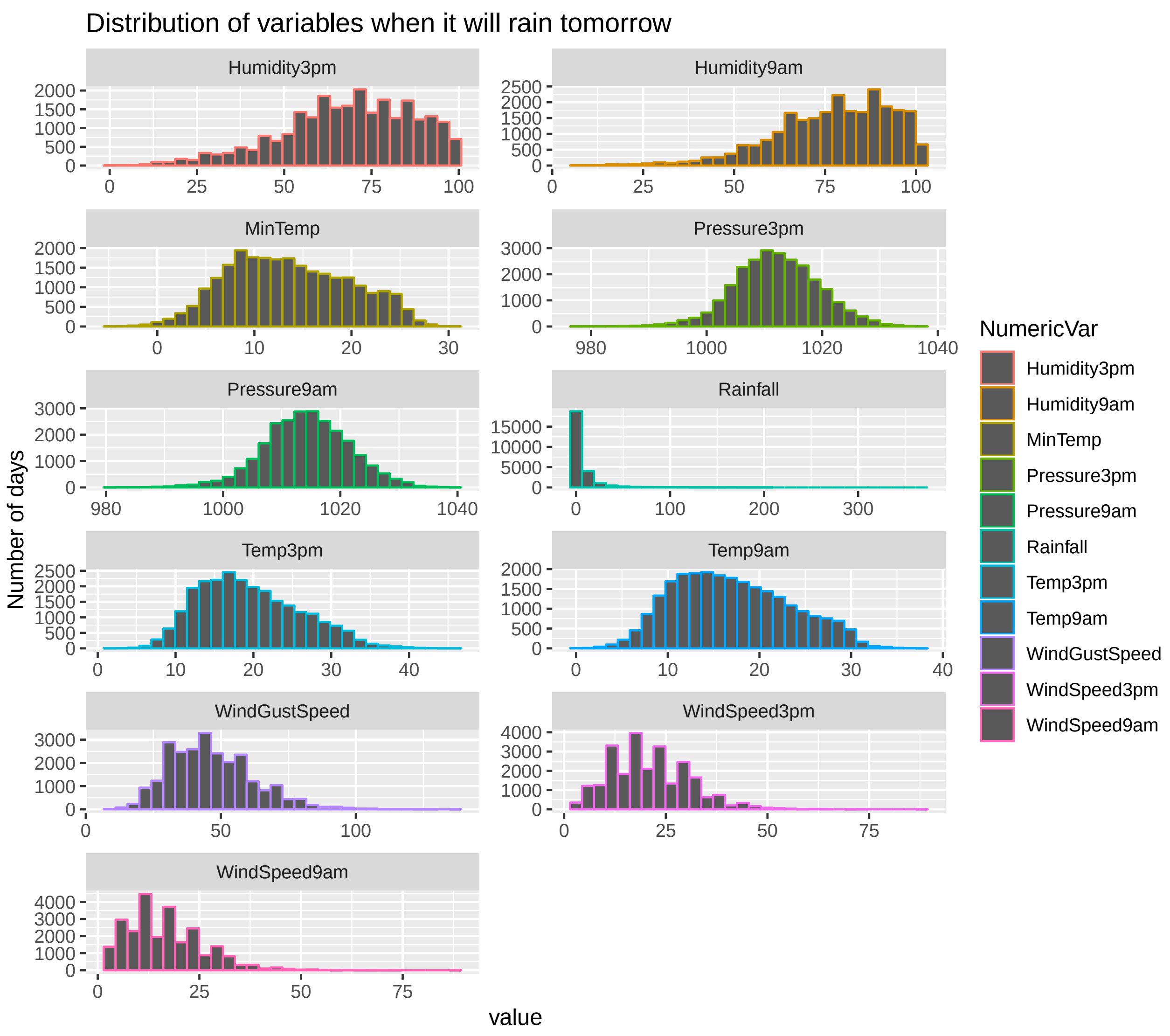
```
##          Date      Location       MinTemp       MaxTemp      Rainfall
##             0             0             0             0             0
##   WindGustDir WindGustSpeed    WindDir9am    WindDir3pm  WindSpeed9am
##             0             0             0             0             0
##  WindSpeed3pm   Humidity9am   Humidity3pm   Pressure9am   Pressure3pm
##             0             0             0             0             0
##       Temp9am       Temp3pm     RainToday       RISK_MM  RainTomorrow
##             0             0             0             0             0
##        Season
##             0
```

Exploratory Data Analysis (EDA)

The graphs show the distribution of the numberic values per each binary target value. It can be observed that skewnesses of **Humidity3pm** and **Humidty9am** are changing based on tomorrow is rainy or not. Pressure and temparature distributions are also slightly different. That might help to predict whether it'll rain tomorrow or the amount of rain will it rain.

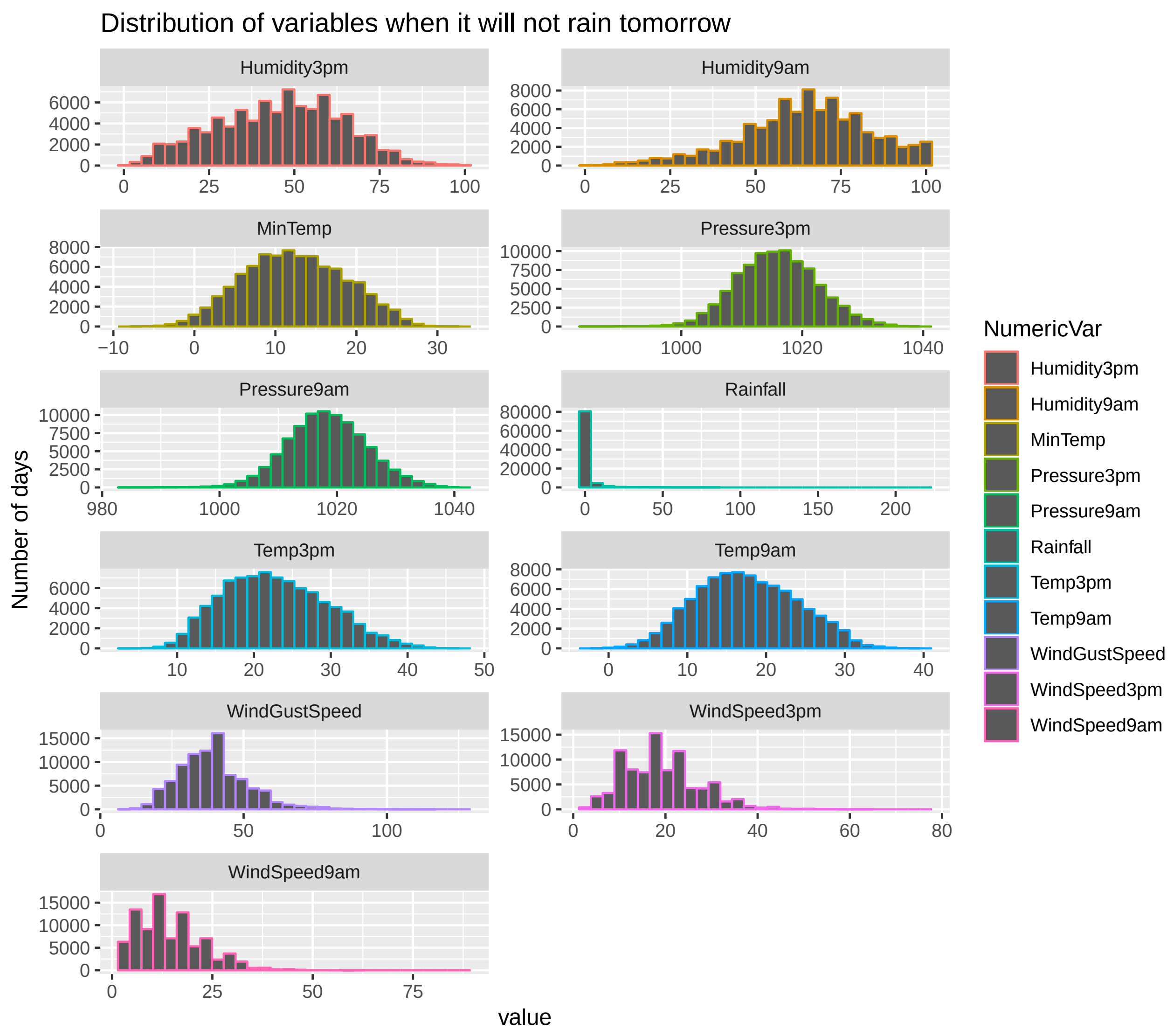
```
num_values_paired <- weather_clean %>%
  gather("NumericVar", "Value", c(3,5,7, 10:17))

dist_yes <- ggplot(data = num_values_paired %>% filter(RainTomorrow == "Yes"),
              aes(x = Value, color = NumericVar)) +
  geom_histogram() +
  scale_y_continuous("Number of days") +
  scale_x_continuous("value") +
  facet_wrap( .~ NumericVar, ncol = 2, scales = "free") +
  ggtitle("Distribution of variables when it will rain tomorrow")
dist_yes
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
dist_no <- ggplot(data = num_values_paired %>% filter(RainTomorrow == "No"),
              aes(x = Value, color = NumericVar)) +
  geom_histogram() +
  scale_y_continuous("Number of days") +
  scale_x_continuous("value") +
  facet_wrap( .~ NumericVar, ncol = 2, scales = "free") +
  ggtitle("Distribution of variables when it will not rain tomorrow")
dist_no
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Modelling Research Question 1

1. Logistic Regression & Burato Algorithm for Feature Selection

When we approach the problem as a **binary classification proplem** to predict whether it'll rain tomorrow or not, the algorithm will be **logistic regression** to apply on the problem. Since the research question has two predictable outcome which are "yes" or "no".

Target variable: *RainTomorrow* Predictor variables: all columns excluding *Date, Rainfall and RISK_MM*

```
weather_clean_r1 <- subset(weather_clean, select = -c(Date, RISK_MM))
weather_clean_r1$RainTomorrow <- relevel(factor(weather_clean_r1$RainTomorrow),
                                         ref = "Yes")
```

Splitting data to train and test

```
set.seed(1)
trn_index = createDataPartition(y = weather_clean_r1$RainTomorrow, p = 0.70, list = FALSE)
trn_weather = weather_clean_r1[trn_index, ]
tst_weather = weather_clean_r1[-trn_index, ]
```

Fitting training data to logistic regresssion

```
set.seed(1)
weather_lgr = train(RainTomorrow ~ ., method = "glm",
  family = binomial(link = "logit"), data = trn_weather,
  trControl = trainControl(method = 'cv', number = 5))
weather_lgr
```

```
## Generalized Linear Model
##
## 79049 samples
##    18 predictor
##     2 classes: 'Yes', 'No'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 63239, 63239, 63240, 63239, 63239
## Resampling results:
##
##   Accuracy   Kappa
##   0.8526104  0.5224735
```

```
set.seed(1)
predicted_outcomes <- predict(weather_lgr, tst_weather)
predicted_outcomes[1:10]
```

```
##  [1] No  No  No  No  Yes No  No  No  No  No
## Levels: Yes No
```

Once the training data is fitted on logistic regression with all features the prediction accuracy is 85% which is kind of good accuracy but definitely could be better by eliminating the unimportant variables which can be noise for the model.

```
accuracy <- sum(predicted_outcomes == tst_weather$RainTomorrow) /
  length(tst_weather$RainTomorrow)
accuracy
```

```
## [1] 0.8559747
```

Zooming a bit more to the accuracy to understand at which class, the logistic resgression is better to predict (yes or no), actually it's more important to know for us when we predict when it'll rain because in the context it makes more sense to be prepared for rain. So, we're more interested in **recall score (sensitivity)** than **accuracy**, meaning that out of all the positive classes, how much we predicted correctly. It should be high as possible. Regarding the model that we fitted, *sensitivity (52%)* is much lower than *accuracy(85%)*.

```
weather_confM <- confusionMatrix(predicted_outcomes, tst_weather$RainTomorrow)
weather_confM
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   Yes    No
##        Yes  3913  1287
##        No   3592 25084
##
##                Accuracy : 0.856
##                  95% CI : (0.8522, 0.8597)
##     No Information Rate : 0.7785
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5309
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.5214
##             Specificity : 0.9512
##          Pos Pred Value : 0.7525
##          Neg Pred Value : 0.8747
##              Prevalence : 0.2215
##          Detection Rate : 0.1155
##    Detection Prevalence : 0.1535
##       Balanced Accuracy : 0.7363
##
##        'Positive' Class : Yes
##
```

So, to find the important features to increase recall & accuracy, we can use **Boruta** package that will help to classify features as important and unimportant. Then the important ones can be fitted to the linear regression again and see if the scores improve. Then, we'll be able to answer research question 1 by identifying more important variables to predict whether it'll rain tomorrow or not.

Classifying features as important or unimportant with `Boruta`

Since Boruta algorithm is a wrapper built on random forest classification, if we use the all data set, it'll be computationally heavy and time-consuming, so, the data will be sampled to make it faster because the original purpose of the project is to demonstrate our R skills rather than computational work.

```r
# Sample Data
set.seed(1)
sample_index = createDataPartition(y = weather_clean_r1$RainTomorrow, p = 0.01,
                                list = FALSE)
sample_weather = weather_clean_r1[sample_index, ]



set.seed(1)
boruta_weather_train <- Boruta(RainTomorrow ~ .,
                                  data = sample_weather,
                                  doTrace = 0)
boruta_weather_train
```

```
## Boruta performed 99 iterations in 1.055885 mins.
##  14 attributes confirmed important: Humidity3pm, Humidity9am,
## MaxTemp, MinTemp, Pressure3pm and 9 more;
##  3 attributes confirmed unimportant: Location, Season, WindDir3pm;
##  1 tentative attributes left: WindSpeed9am;
```

It seems that Location, Season, WinDir3pm and WindSpeed9am features are unimportant and they'll be removed from the features to fit the model again.

Fitting only important features to logistic regression again

```r
weather_clean_r1_1 <- subset(weather_clean, select = -c(Date, RISK_MM,Location, Season, WindDir3pm,
                                                        WindSpeed9am, WindDir9am))
weather_clean_r1_1$RainTomorrow <- relevel(factor(weather_clean_r1_1$RainTomorrow),
                                          ref = "Yes")
set.seed(1)
trn_index = createDataPartition(y = weather_clean_r1_1$RainTomorrow, p = 0.70, list = FALSE)
```

```r
trn_weather = weather_clean_r1[trn_index, ]
tst_weather = weather_clean_r1[-trn_index, ]

set.seed(1)
weather_lgr = train(RainTomorrow ~ ., method = "glm",
  family = binomial(link = "logit"), data = trn_weather,
  trControl = trainControl(method = 'cv', number = 5))


set.seed(1)
predicted_outcomes <- predict(weather_lgr, tst_weather)
predicted_outcomes[1:10]
```

```
##  [1] No  No  No  No  Yes No  No  No  No  No 
## Levels: Yes No
```

```r
accuracy <- sum(predicted_outcomes == tst_weather$RainTomorrow) /
  length(tst_weather$RainTomorrow)
accuracy
```

```
## [1] 0.8559747
```

```r
weather_confM <- confusionMatrix(predicted_outcomes, tst_weather$RainTomorrow)
weather_confM
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   Yes    No
##        Yes  3913  1287
##        No   3592 25084
##                                           
##                Accuracy : 0.856           
##                  95% CI : (0.8522, 0.8597)
##     No Information Rate : 0.7785          
##     P-Value [Acc > NIR] : < 2.2e-16       
##                                           
##                   Kappa : 0.5309          
##  Mcnemar's Test P-Value : < 2.2e-16       
##                                           
##             Sensitivity : 0.5214          
##             Specificity : 0.9512          
##          Pos Pred Value : 0.7525          
##          Neg Pred Value : 0.8747          
##              Prevalence : 0.2215          
##          Detection Rate : 0.1155          
##    Detection Prevalence : 0.1535          
##       Balanced Accuracy : 0.7363          
##                                           
##        'Positive' Class : Yes             
## 
```

After eliminating the unimportant variables based on `Burota` package, we observed that neither accuracy nor sensitivity score has changed, so it still supports our model that at least it's proved that Location, Season, WinDir3pm and WindSpeed9am features are not important predictors to predict it'll rain tomorrow since it didn't decrease the model accuracy by removing them.

K-Nearest Neighbor(KNN) & Principal Component Analysis (PCA)

Besides a logistic regression, we also wanted to test other options with our dataset. The idea was to do a PCA for feature selection, and then fit a KNN model.

Splitting data to test and train

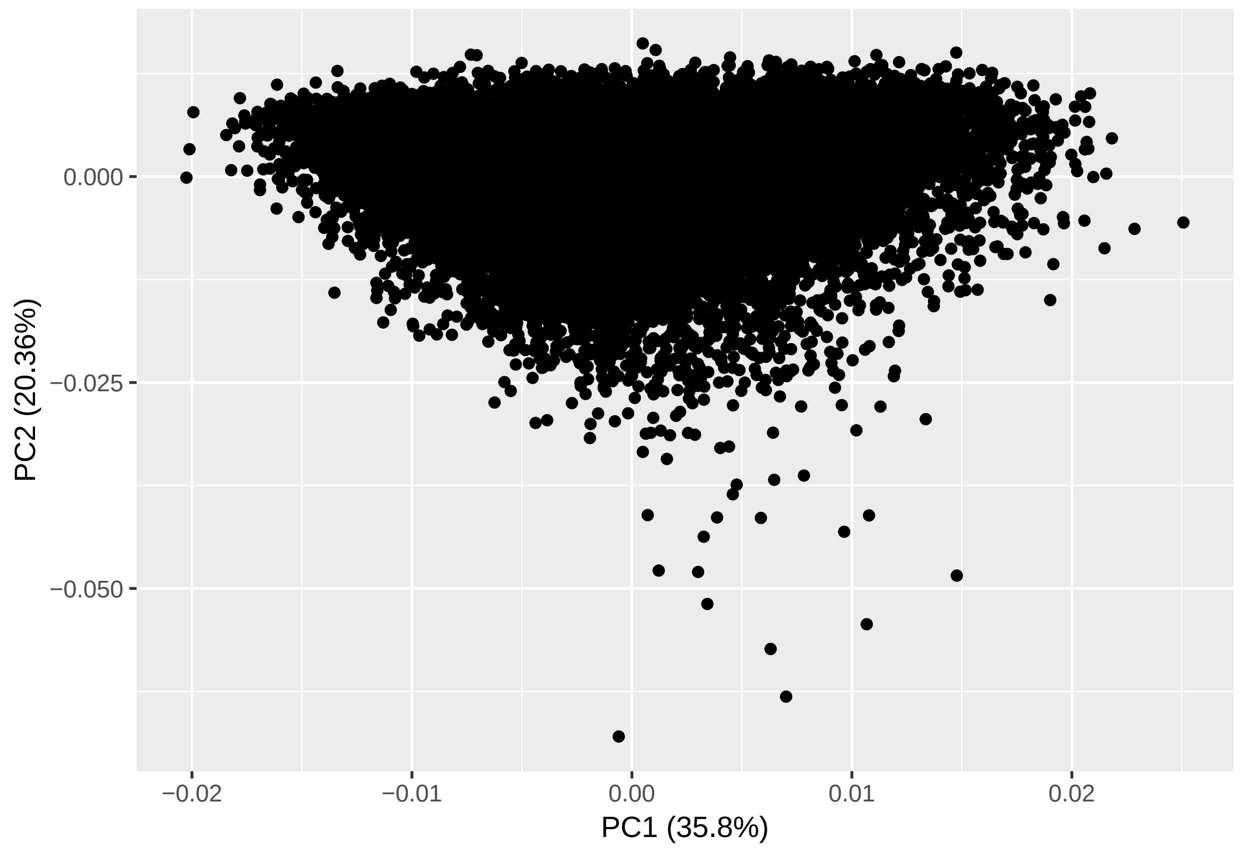
```
set.seed(1)
weather_frac = sample_frac(weather_clean, size = 0.2, replace = FALSE)

trn_index <- createDataPartition(weather_frac$RainTomorrow, p = 0.7, list = FALSE)
trn_weather <- weather_frac[trn_index, ]
tst_weather <- weather_frac[-trn_index, ]
```

PCA for feature selection

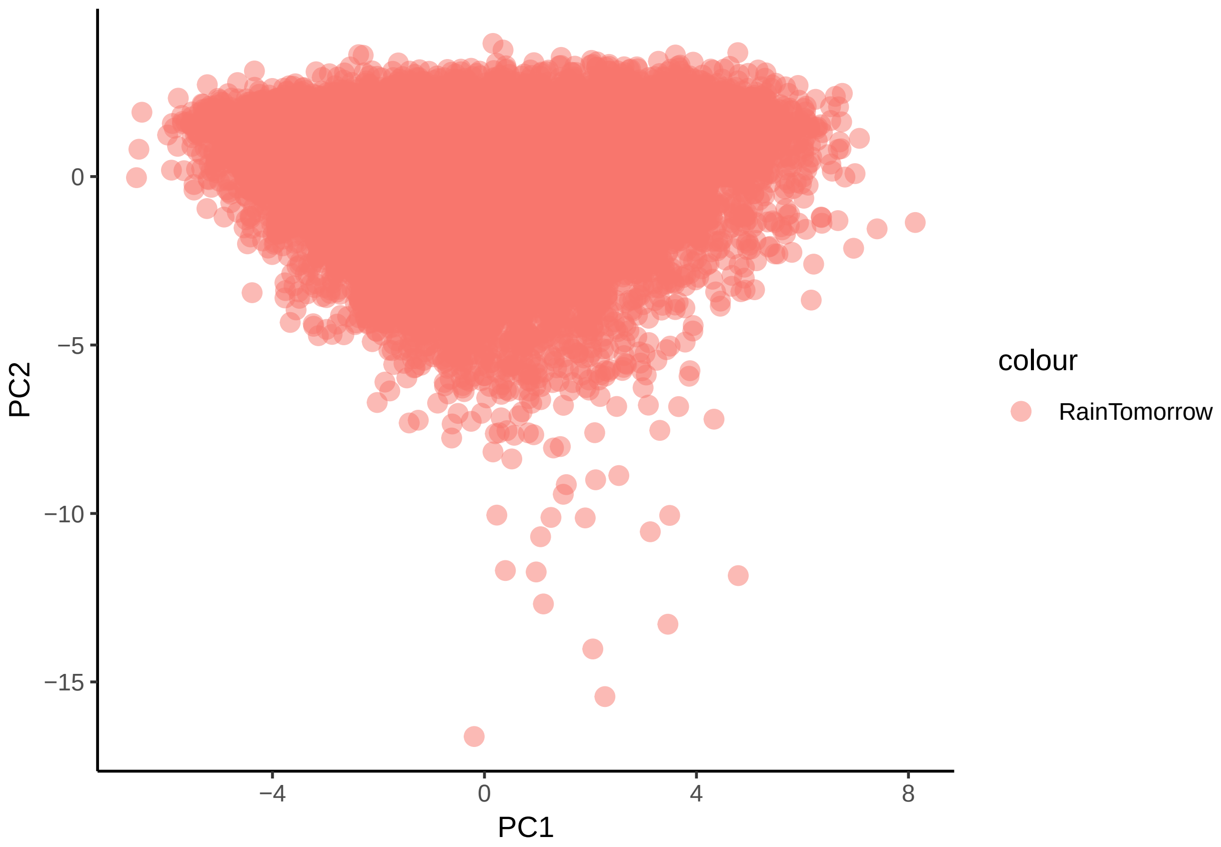
```
pca_weather <- prcomp(na.omit(weather_frac[,c(3:5,7,10:17,19)]),
                      center = TRUE,
                      scale. = TRUE)
summary(pca_weather)
```

```
## Importance of components:
##                          PC1    PC2    PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.157 1.6270 1.3421 1.02957 0.92551 0.83305 0.71411
## Proportion of Variance 0.358 0.2036 0.1386 0.08154 0.06589 0.05338 0.03923
## Cumulative Proportion  0.358 0.5617 0.7002 0.78176 0.84766 0.90104 0.94026
##                            PC8     PC9    PC10    PC11    PC12    PC13
## Standard deviation     0.58037 0.51128 0.32782 0.18209 0.16339 0.10488
## Proportion of Variance 0.02591 0.02011 0.00827 0.00255 0.00205 0.00085
## Cumulative Proportion  0.96618 0.98628 0.99455 0.99710 0.99915 1.00000
```

```
saved_pca <- pca_weather$x[, 1:13]
saved_pca <- cbind(saved_pca, RainTomorrow = factor(weather_frac$RainTomorrow))

autoplot(pca_weather) # quick plot without color and edits
```



```r
ggplot(saved_pca, aes(x = PC1,y = PC2, color = "RainTomorrow"))+
  geom_point(size=3,alpha=0.5)+
  theme_classic()
```



As the code shows, the PCA works fine, but the plotting does not. The PCA shows 13 components. RainTomorrow as factor has a different amount of rows than the saved_pca variable. This is why the color of the ggplot is off, as we unfortunately unable to fix this properly.

We wanted to follow up with a KNN model. As seen below, this is a regular model without using the PCA components. Multiple error warnings showed that the datapoints of the components were too close together to build a KNN model.

Fitting the data onto KNN model

```
## Training a basic KNN model for target RainTomorrow
knn_weather <- train(RainTomorrow ~. -Date, method = "knn", data = trn_weather,
                     trControl = trainControl(method = 'cv', number = 10),
                     na.action = na.omit)
knn_weather
```

```
## k-Nearest Neighbors
##
## 15810 samples
##    20 predictor
##     2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 14229, 14229, 14229, 14229, 14229, 14229, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.8756483  0.5918052
##   7  0.8767868  0.5914922
##   9  0.8764073  0.5878667
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
# k = 9 proves to be the most accurate (0.892)
```

The regular KNN model has high accuracy on the train set, where k = 9 is the most accurate (0.892).

Modelling Research Question 2

Different linear models were tried and compared to predict the target variable **RISK_MM** (amount of rain that will fall tomorrow). First a linear regression model was tried, including all variables of the dataset, except Date. Furthermore, forward and backwards feature selection was tried using leapforward and leapbackward when training a model with the caret package in combination with the leap package. Finally a simple neural network was tried using the neuralnet package, with a minimum amount of steps and repetitions, as a neural network is very computationally expensive.

The data was sampled into a smaller set to decrease running time of the models.

```
set.seed(1)
weather_frac = sample_frac(weather_clean, size = 0.2, replace = FALSE)

weather_frac_r2 <- subset(weather_frac, select = -c(Date, RainTomorrow))
```

```
validation_index <- createDataPartition(weather_frac_r2$RISK_MM, p=0.70,
  list=FALSE)
validation <- weather_frac_r2[-validation_index,]
training <- weather_frac_r2[validation_index,]
```

Linear Regression Without Feature Selection

```
full_linear <- lm(RISK_MM ~., data= training)
summary(full_linear)
```

```
##
## Call:
## lm(formula = RISK_MM ~ ., data = training)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.613  -2.465  -0.662   1.110 212.897
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                39.004927  11.869943   3.286 0.001018 **
## LocationAlbury             -0.171153   0.542923  -0.315 0.752580
## LocationAliceSprings        0.783891   0.534482   1.467 0.142495
## LocationBadgerysCreek      -0.413007   0.550498  -0.750 0.453120
## LocationBallarat           -2.666379   0.546768  -4.877 1.09e-06 ***
## LocationBendigo            -0.511548   0.517992  -0.988 0.323384
## LocationBrisbane           -0.726748   0.539049  -1.348 0.177612
## LocationCairns             -1.375663   0.568761  -2.419 0.015587 *
## LocationCanberra           -0.417340   0.536397  -0.778 0.436555
## LocationCobar             0.757176   0.512455   1.478 0.139549
## LocationCoffsHarbour      0.418198   0.553922   0.755 0.450274
## LocationDartmoor         -2.004525   0.562671  -3.563 0.000368 ***
## LocationDarwin           -1.112891   0.545973  -2.038 0.041530 *
## LocationGoldCoast        -0.429653   0.555892  -0.773 0.439589
## LocationHobart           -2.077852   0.535633  -3.879 0.000105 ***
## LocationKatherine        -2.894170   0.846453  -3.419 0.000630 ***
## LocationLaunceston       -1.206474   0.618485  -1.951 0.051111 .
## LocationMelbourne        -1.727594   0.570333  -3.029 0.002457 **
## LocationMelbourneAirport -1.634892   0.526110  -3.108 0.001890 **
## LocationMildura           0.563324   0.514851   1.094 0.273907
## LocationMoree            -0.374715   0.527643  -0.710 0.477611
## LocationMountGambier     -1.854381   0.524189  -3.538 0.000405 ***
## LocationNhil             -1.205997   0.615163  -1.960 0.049961 *
## LocationNorahHead        -2.293605   0.536456  -4.275 1.92e-05 ***
## LocationNorfolkIsland    -2.465016   0.534269  -4.614 3.98e-06 ***
## LocationNuriootpa        -0.865331   0.522240  -1.657 0.097547 .
## LocationPearceRAAF       -1.173081   0.541836  -2.165 0.030402 *
## LocationPerth            -0.441274   0.508916  -0.867 0.385909
## LocationPerthAirport     -0.953912   0.503959  -1.893 0.058398 .
## LocationPortland         -2.300831   0.521786  -4.410 1.04e-05 ***
## LocationRichmond         -0.807413   0.566323  -1.426 0.153971
## LocationSale             -1.895627   0.537829  -3.525 0.000425 ***
## LocationSydney           -1.035325   0.566155  -1.829 0.067464 .
## LocationSydneyAirport    -1.375220   0.534407  -2.573 0.010081 *
## LocationTownsville       -1.017379   0.571950  -1.779 0.075294 .
## LocationTuggeranong      -0.225032   0.548296  -0.410 0.681504
## LocationUluru            -0.316342   0.652150  -0.485 0.627629
## LocationWaggaWagga       -0.192413   0.516411  -0.373 0.709454
## LocationWalpole          -2.647451   0.548194  -4.829 1.38e-06 ***
## LocationWatsonia         -1.545136   0.524573  -2.946 0.003229 **
## LocationWilliamtown      -0.716469   0.559575  -1.280 0.200431
## LocationWitchcliffe      -1.126282   0.550029  -2.048 0.040608 *
## LocationWollongong       -2.266696   0.529286  -4.283 1.86e-05 ***
## LocationWoomera           0.278131   0.514839   0.540 0.589047
## MinTemp                   0.015950   0.029142   0.547 0.584164
## MaxTemp                   0.070734   0.054228   1.304 0.192119
## Rainfall                  0.157547   0.007911  19.916  < 2e-16 ***
## WindGustDirENE            0.153924   0.327482   0.470 0.638344
## WindGustDirESE           -0.223384   0.326218  -0.685 0.493501
## WindGustDirN             -0.055454   0.358917  -0.155 0.877215
## WindGustDirNE             0.075922   0.353065   0.215 0.829741
## WindGustDirNNE           -0.430003   0.369655  -1.163 0.244745
## WindGustDirNNW           -0.185100   0.392839  -0.471 0.637515
## WindGustDirNW             0.043683   0.371019   0.118 0.906277
## WindGustDirS              0.190929   0.346830   0.550 0.581986
## WindGustDirSE             0.115765   0.326221   0.355 0.722695
## WindGustDirSSE            0.066258   0.341457   0.194 0.846144
## WindGustDirSSW           -0.031406   0.350559  -0.090 0.928615
## WindGustDirSW            -0.126003   0.350228  -0.360 0.719020
## WindGustDirW             -0.262801   0.352528  -0.745 0.455996
## WindGustDirWNW           -0.119912   0.367232  -0.327 0.744028
## WindGustDirWSW            0.073374   0.351230   0.209 0.834525
## WindGustSpeed             0.144344   0.007078  20.394  < 2e-16 ***
## WindDir9amENE              0.542958   0.310442   1.749 0.080313 .
## WindDir9amESE              0.433296   0.316591   1.369 0.171134
## WindDir9amN               -0.613079   0.316410  -1.938 0.052689 .
## WindDir9amNE               0.514722   0.327133   1.573 0.115639
## WindDir9amNNE              0.416348   0.331560   1.256 0.209235
## WindDir9amNNW             -0.810917   0.349301  -2.322 0.020270 *
## WindDir9amNW              -0.901689   0.347614  -2.594 0.009497 **
## WindDir9amS               -0.076351   0.326614  -0.234 0.815169
## WindDir9amSE              -0.347962   0.311422  -1.117 0.263870
## WindDir9amSSE             -0.310416   0.320682  -0.968 0.333066
## WindDir9amSSW             -0.606373   0.345027  -1.757 0.078858 .
## WindDir9amSW              -0.231783   0.348029  -0.666 0.505429
## WindDir9amW               -0.866110   0.352934  -2.454 0.014137 *
## WindDir9amWNW             -0.402991   0.360154  -1.119 0.263183
## WindDir9amWSW             -0.371058   0.353193  -1.051 0.293467
## WindDir3pmENE             -0.881767   0.337792  -2.610 0.009053 **
## WindDir3pmESE             -0.122977   0.327195  -0.376 0.707032
## WindDir3pmN               -0.140311   0.361913  -0.388 0.698250
## WindDir3pmNE              -1.684882   0.348328  -4.837 1.33e-06 ***
## WindDir3pmNNE             -1.213031   0.372474  -3.257 0.001130 **
## WindDir3pmNNW             -0.320688   0.376567  -0.852 0.394443
## WindDir3pmNW              -0.456650   0.373849  -1.221 0.221922
## WindDir3pmS               -0.012308   0.337996  -0.036 0.970953
## WindDir3pmSE              -0.429504   0.318086  -1.350 0.176947
## WindDir3pmSSE             -0.049591   0.339422  -0.146 0.883841
## WindDir3pmSSW              0.347042   0.355752   0.976 0.329318
## WindDir3pmSW              -0.166025   0.357681  -0.464 0.642532
## WindDir3pmW                0.039140   0.355787   0.110 0.912404
## WindDir3pmWNW             -0.044210   0.364505  -0.121 0.903466
## WindDir3pmWSW              0.051801   0.352493   0.147 0.883168
## WindSpeed9am               0.006768   0.009679   0.699 0.484401
## WindSpeed3pm              -0.096298   0.009731  -9.896  < 2e-16 ***
## Humidity9am               -0.012653   0.006359  -1.990 0.046636 *
## Humidity3pm                0.163588   0.007113  22.998  < 2e-16 ***
## Pressure9am                0.622177   0.040574  15.334  < 2e-16 ***
## Pressure3pm               -0.670817   0.040079 -16.738  < 2e-16 ***
## Temp9am                   -0.053525   0.046088  -1.161 0.245518
## Temp3pm                   -0.004542   0.059953  -0.076 0.939610
## RainTodayYes               0.471608   0.173765   2.714 0.006654 **
## SeasonSpring              -0.441409   0.164093  -2.690 0.007153 **
## SeasonSummer               0.217808   0.179485   1.214 0.224950
## SeasonWinter              -0.400754   0.189242  -2.118 0.034218 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.976 on 15705 degrees of freedom
## Multiple R-squared:  0.2263, Adjusted R-squared:  0.2212
## F-statistic: 44.17 on 104 and 15705 DF,  p-value: < 2.2e-16
```

```
MSE(y_pred = predict(full_linear, validation), y_true = validation$RISK_MM)
```

```
## [1] 59.34653
```

```
# MSE = 56.87, R2 = 0.27
```

Stepwise feature selection using Caret, Leaps and MASS

For the simple linear model all variables were included to predict RISK_MM. We also tried to implement backwards and forwards feature selection to check if leaving out features can lead to better predictions

- 1. Backward feature Selection

```
set.seed(1)
backwards_model <- train(RISK_MM ~. , data = training,
                     method = "leapBackward",
                     tuneGrid = data.frame(nvmax = 1:18),
                     trControl = trainControl(method = "cv", number = 10),
                     na.action = na.exclude)
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```
backwards_model$results
```

```
##    nvmax     RMSE   Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1      1 7.629633 0.05271570 3.176216 1.125572 0.01240481 0.1553364
## 2      2 7.629352 0.05280321 3.174591 1.125146 0.01239039 0.1550559
## 3      3 7.625690 0.05360128 3.173745 1.123843 0.01145819 0.1544331
## 4      4 7.621979 0.05454935 3.175074 1.123022 0.01192728 0.1556921
## 5      5 7.619561 0.05519086 3.172759 1.124580 0.01214816 0.1570521
## 6      6 7.603028 0.05957518 3.174631 1.127524 0.01353631 0.1570393
## 7      7 7.602971 0.05958458 3.174888 1.126441 0.01355569 0.1580846
## 8      8 7.603390 0.05949559 3.174136 1.126571 0.01365350 0.1584838
## 9      9 7.601737 0.05995186 3.172603 1.122183 0.01339476 0.1592889
## 10    10 7.598160 0.06075652 3.172812 1.120851 0.01246855 0.1595194
## 11    11 7.597004 0.06107688 3.171436 1.120116 0.01289170 0.1598675
## 12    12 7.597152 0.06104855 3.171816 1.120220 0.01292376 0.1599876
## 13    13 7.596697 0.06116075 3.170675 1.119754 0.01295282 0.1596706
## 14    14 7.596547 0.06122224 3.170597 1.119727 0.01315491 0.1591900
## 15    15 7.594944 0.06161241 3.168480 1.120001 0.01336925 0.1599359
## 16    16 7.594922 0.06169802 3.168058 1.118047 0.01355120 0.1594530
## 17    17 7.594926 0.06170483 3.167793 1.117793 0.01361541 0.1599390
## 18    18 7.592061 0.06252164 3.183630 1.114818 0.01290278 0.1621939
```

The results of the backwards feature selection model show that including all 18 variables leads to the lowest RMSE and the highest R squared score.

The backwards feature selection model has an MSE of

```
MSE(y_pred = predict(backwards_model, validation), y_true = validation$RISK_MM)
```

```
## [1] 71.514
```

- 2. Forward Feature Selection

```
forwards_model <- train(RISK_MM ~. , data = training,
                    method = "leapForward",
                    tuneGrid = data.frame(nvmax = 1:18),
                    trControl = trainControl(method = "cv", number = 10),
                    na.action = na.exclude)
```

```
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
## Reordering variables and trying again:
```

```
forwards_model$results
```

```
##    nvmax     RMSE   Rsquared      MAE   RMSESD RsquaredSD     MAESD
## 1      1 7.582508 0.05474692 3.175895 1.433014 0.01735202 0.1779703
## 2      2 7.582301 0.05483861 3.174417 1.432485 0.01755949 0.1769568
## 3      3 7.579663 0.05546339 3.169821 1.433044 0.01737813 0.1770036
## 4      4 7.578904 0.05572383 3.170492 1.435021 0.01777857 0.1775136
## 5      5 7.561437 0.06003377 3.176058 1.428901 0.01761640 0.1828253
## 6      6 7.561714 0.05998168 3.177195 1.427612 0.01757299 0.1817460
## 7      7 7.557641 0.06090816 3.175340 1.428575 0.01724883 0.1834513
## 8      8 7.557522 0.06088972 3.174216 1.429179 0.01714231 0.1852817
## 9      9 7.551915 0.06267829 3.171645 1.424088 0.01979855 0.1843580
## 10    10 7.552058 0.06264361 3.171472 1.424110 0.01980318 0.1843688
## 11    11 7.551656 0.06271658 3.170792 1.424412 0.01969466 0.1848665
## 12    12 7.551789 0.06267532 3.171601 1.423692 0.01965477 0.1853148
## 13    13 7.551474 0.06274626 3.171279 1.424325 0.01963012 0.1855148
## 14    14 7.551244 0.06282030 3.171416 1.424677 0.01968190 0.1855226
## 15    15 7.550750 0.06294223 3.170736 1.424176 0.01963997 0.1856800
## 16    16 7.550631 0.06303957 3.170834 1.422156 0.02000336 0.1849253
## 17    17 7.550680 0.06301787 3.170906 1.422298 0.01992060 0.1844607
## 18    18 7.548431 0.06356009 3.171678 1.423294 0.01933368 0.1835269
```

The results of the forward feature selection model also showed that including all 18 variables lead to the best prediction for RISK_MM, but the R squared score is slightely higher and the RMSE score is slighetly lower for forwards feature selection.

The forward feature selection model has an MSE of

```
MSE(y_pred = predict(forwards_model, validation), y_true = validation$RISK_MM)
```

```
## [1] 71.40349
```

Comparing the MSEs of the simple linear model and the models using backwards and forwards feature selection, the simple linear model still performs the best.

Deep Learning

In the next piece of code a simple neural network is implemented to try to get a smaller MSE score than the simple linear regression model.

- First all factor variables have to become dummy variables

```
n <- names(weather_frac_r2)
formula_bin <- as.formula(paste("~ ", paste(n, collapse= "+")))
binarized_train <- as.data.frame(model.matrix(formula_bin , data = training))
binarized_valid <- as.data.frame(model.matrix(formula_bin, data = validation))

binarized_train$`(Intercept)`<- NULL
binarized_valid$`(Intercept)`<- NULL
```

- Adding all variables in a formula to use in the neural network. The parameters 'rep' and 'stepmax' are kept very low to speed up the process time.

```
n_bin <-names(binarized_train)
formula_net <- as.formula(paste("RISK_MM ~", paste(n_bin[!n_bin %in% "RISK_MM"],
                          collapse = "+")))

nn <- neuralnet(formula_net, data=binarized_train, hidden=3, rep=50,
                linear.output=T, lifesign="full", algorithm="rprop+",
                threshold=0.01, stepmax = 1000)
```

```
## hidden: 3    thresh: 0.01    rep:  1/50    steps:      48    error: 493887.4038  time: 0.71 secs
## hidden: 3    thresh: 0.01    rep:  2/50    steps:      52    error: 493887.4038  time: 0.79 secs
## hidden: 3    thresh: 0.01    rep:  3/50    steps:      52    error: 493887.4038  time: 0.79 secs
## hidden: 3    thresh: 0.01    rep:  4/50    steps:      57    error: 493887.4038  time: 0.89 secs
## hidden: 3    thresh: 0.01    rep:  5/50    steps:      62    error: 493887.4038  time: 1.13 secs
## hidden: 3    thresh: 0.01    rep:  6/50    steps:      58    error: 493887.4038  time: 0.84 secs
## hidden: 3    thresh: 0.01    rep:  7/50    steps:      65    error: 493887.4038  time: 0.92 secs
## hidden: 3    thresh: 0.01    rep:  8/50    steps: stepmax    min thresh: 3959.194317
## hidden: 3    thresh: 0.01    rep:  9/50    steps:      54    error: 493887.4038  time: 0.8 secs
## hidden: 3    thresh: 0.01    rep: 10/50    steps: stepmax    min thresh: 120.6535465
## hidden: 3    thresh: 0.01    rep: 11/50    steps:      50    error: 493887.4038  time: 0.87 secs
## hidden: 3    thresh: 0.01    rep: 12/50    steps:      51    error: 493887.4038  time: 0.74 secs
## hidden: 3    thresh: 0.01    rep: 13/50    steps:      52    error: 493887.4038  time: 0.75 secs
## hidden: 3    thresh: 0.01    rep: 14/50    steps: stepmax    min thresh: 18.55614134
## hidden: 3    thresh: 0.01    rep: 15/50    steps:      68    error: 493887.4038  time: 1.08 secs
## hidden: 3    thresh: 0.01    rep: 16/50    steps:      60    error: 493887.4038  time: 1.1 secs
## hidden: 3    thresh: 0.01    rep: 17/50    steps:      61    error: 493887.4038  time: 0.94 secs
## hidden: 3    thresh: 0.01    rep: 18/50    steps:      57    error: 493887.4038  time: 0.93 secs
## hidden: 3    thresh: 0.01    rep: 19/50    steps:      62    error: 493887.4038  time: 0.89 secs
## hidden: 3    thresh: 0.01    rep: 20/50    steps:      59    error: 493887.4038  time: 0.89 secs
## hidden: 3    thresh: 0.01    rep: 21/50    steps:      56    error: 493887.4038  time: 0.85 secs
## hidden: 3    thresh: 0.01    rep: 22/50    steps:      57    error: 493887.4038  time: 1.07 secs
## hidden: 3    thresh: 0.01    rep: 23/50    steps:      47    error: 493887.4038  time: 0.69 secs
## hidden: 3    thresh: 0.01    rep: 24/50    steps: stepmax    min thresh: 4301.630098
## hidden: 3    thresh: 0.01    rep: 25/50    steps: stepmax    min thresh: 3641.430109
## hidden: 3    thresh: 0.01    rep: 26/50    steps:      58    error: 493887.4038  time: 0.87 secs
## hidden: 3    thresh: 0.01    rep: 27/50    steps: stepmax    min thresh: 4232.506632
## hidden: 3    thresh: 0.01    rep: 28/50    steps:      64    error: 493887.4038  time: 1.03 secs
## hidden: 3    thresh: 0.01    rep: 29/50    steps:      51    error: 493887.4038  time: 0.87 secs
## hidden: 3    thresh: 0.01    rep: 30/50    steps:      59    error: 493887.4038  time: 0.97 secs
## hidden: 3    thresh: 0.01    rep: 31/50    steps:      73    error: 493887.4038  time: 1.13 secs
## hidden: 3    thresh: 0.01    rep: 32/50    steps:      59    error: 493887.4038  time: 0.86 secs
## hidden: 3    thresh: 0.01    rep: 33/50    steps:      46    error: 493887.4038  time: 0.68 secs
## hidden: 3    thresh: 0.01    rep: 34/50    steps:      55    error: 493887.4038  time: 0.82 secs
## hidden: 3    thresh: 0.01    rep: 35/50    steps:      60    error: 493887.4038  time: 1.13 secs
## hidden: 3    thresh: 0.01    rep: 36/50    steps:      59    error: 493887.4038  time: 0.99 secs
## hidden: 3    thresh: 0.01    rep: 37/50    steps:      78    error: 493887.4038  time: 1.2 secs
## hidden: 3    thresh: 0.01    rep: 38/50    steps:      57    error: 493887.4038  time: 0.89 secs
## hidden: 3    thresh: 0.01    rep: 39/50    steps:      55    error: 493887.4038  time: 1.05 secs
## hidden: 3    thresh: 0.01    rep: 40/50    steps:      58    error: 493887.4038  time: 0.98 secs
## hidden: 3    thresh: 0.01    rep: 41/50    steps: stepmax    min thresh: 0.9297154123
## hidden: 3    thresh: 0.01    rep: 42/50    steps:      51    error: 493887.4038  time: 0.74 secs
## hidden: 3    thresh: 0.01    rep: 43/50    steps: stepmax    min thresh: 3645.055923
## hidden: 3    thresh: 0.01    rep: 44/50    steps:      53    error: 493887.4038  time: 0.77 secs
## hidden: 3    thresh: 0.01    rep: 45/50    steps:      46    error: 493887.4038  time: 0.66 secs
## hidden: 3    thresh: 0.01    rep: 46/50    steps:      57    error: 493887.4038  time: 0.82 secs
## hidden: 3    thresh: 0.01    rep: 47/50    steps:      54    error: 493887.4038  time: 1.07 secs
## hidden: 3    thresh: 0.01    rep: 48/50    steps:      66    error: 493887.4038  time: 1.11 secs
## hidden: 3    thresh: 0.01    rep: 49/50    steps:      58    error: 493887.4038  time: 1.03 secs
## hidden: 3    thresh: 0.01    rep: 50/50    steps:      69    error: 493887.4038  time: 1.28 secs
```

- Removing the target variable from the validation set to make predictions

```r
copy_bin_valid <- binarized_valid
copy_bin_valid$RISK_MM <- NULL

pred_nn <- neuralnet::compute(nn, copy_bin_valid)
predicted_values <- pred_nn$net.result
```

- Simple neural network resulted in a MSE = 75.35

```r
MSE(y_pred = predicted_values, y_true = validation$RISK_MM)
```

```
## [1] 75.35606684
```

Conclusion

- **Research Question 1** Regarding the logistic regression model and important feature detection method (Burota algorithm), we can predict with 85% accuracy whether it'll rain tomorrow. To answer research question, **"Which weather features can most accurately predict whether it will rain tomorrow?"**, we have 14 important variable that predicts the rain with 85% accuracy and we identified 4 unimportant features as Location, Season, WinDir3pm and WindSpeed9am that they don't change model accuracy. We assume to have better accuracy and recall score by using more advanced classification models which can be computationally heavy and also unsupervised models to identify best predictors.
- **Research Question 2**

When interpreting the model results for research question 2, the linear regression model including all variables from the dataset performed best in predicting the amount of rain tomorrow. To answer the research question, **"how accurate can weather measurements from today predict the amount of rain that will fall tomorrow?"**, our models did not predict the amount of rain very accurately. The best model had a high mean squared error and the R^2 score was very low, which shows that the model did not explain a lot of variability in the target variable RISK_MM and thus does not fit the data well. We expect to get more accurate results with a more computationally expensive neural network, when normalizing the numerical variables on beforehand and when increasing the amount of steps and repetitions the network may make.

Task Distribution

Report writing and data processing have been done together and the seperation of the other tasks are as below;

- Vicky Mekes worked on research question 1 by doing PCA &KNN
- Sema Karan worked on research question 1 by doing EDA, Logistic Reg., Burato feature selection
- Shelly van Erp worked on research question 2 by doing Forward/Backward Selection, Linear Model and Deep Learning.