# Predicting Rain in Australia

Aaron Kittel

5/11/2021

## Introduction

The goal of this project is to develop a model which can predict whether it will rain in Australia tomorrow with more accuracy than random guessing. The data used was taken from Kaggle. It contains 145,460 observations with the following variables:

```
rain_data %>% colnames()
```

```
##  [1] "Date"          "Location"      "MinTemp"       "MaxTemp"
##  [5] "Rainfall"      "Evaporation"   "Sunshine"      "WindGustDir"
##  [9] "WindGustSpeed" "WindDir9am"    "WindDir3pm"    "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"   "Humidity3pm"   "Pressure9am"
## [17] "Pressure3pm"   "Cloud9am"      "Cloud3pm"      "Temp9am"
## [21] "Temp3pm"       "RainToday"     "RainTomorrow"
```

"RainTomorrow" is the target variable which we will be predicting. Using this data, we will create a naive prediction model and use its accuracy as the baseline we will attempt to outperform by training various machine learning models.

## Analysis

As we can see from exploring the variables, every observation contains values for "RainToday" and "RainTomorrow". This is significant because it eliminates the need to treat the data as a time series and analysis may be done using random samples. Consequently, we begin our analysis by randomly partitioning the data into training and test sets using an 80/20 split. We then proceed to create our naive prediction by guessing with 50/50 odds whether it will rain tomorrow or not and check its accuracy.

```
guess_y_hat <- factor(sample(c("Yes","No"), nrow(test_set), replace = TRUE)) %>%
  factor()
confusionMatrix(guess_y_hat, test_set$RainTomorrow)$overall['Accuracy']
```

```
## Accuracy
## 0.498711
```

Knowing there isn't a daily 50% chance of rain in Australia, we might consider adjusting our naive forecast to make guesses a bit more grounded in reality. To do so, we check the percentage of days in our training dataset which experienced rain.

```
chance_of_rain <- mean(train_set$RainToday == "Yes")
chance_of_rain
```

```
## [1] 0.2187132
```

We then use this figure to adjust our naive prediction model; guessing it will rain with odds more commensurate with reality.

```
guess_y_hat <- factor(sample(c("Yes","No"), nrow(test_set), replace = TRUE, prob = c(chance_of_rain, 1 -
  factor()
confusionMatrix(guess_y_hat, test_set$RainTomorrow)$overall['Accuracy']
```

```
##  Accuracy
## 0.6548998
```

At this point we can see the accuracy increases with a smaller chance of guessing "Yes" for rain tomorrow. This stands to reason because we have calculated the chance of rain to be approximately 22%. This means that if we were to only guess "No" for rain every day without exception, our accuracy would be 1 - 0.22, or 0.78. To test this, we adjust our naive prediction to guess "No" 100% of the time.

```
guess_y_hat <- factor(sample(c("Yes","No"), nrow(test_set), replace = TRUE, prob = c(0,1))) %>%
  factor(levels = c("No", "Yes"))
confusionMatrix(guess_y_hat, test_set$RainTomorrow)$overall['Accuracy']
```

```
##  Accuracy
## 0.7808408
```

Despite its simplicity, this method results in the highest accuracy and so it will serve as our baseline naive prediction model. However, a more thorough observation of the confusion matrix reveals a concern.

```
confusionMatrix(guess_y_hat, test_set$RainTomorrow)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    No   Yes
##        No  22717  6376
##        Yes     0     0
##
##                Accuracy : 0.7808
##                  95% CI : (0.776, 0.7856)
##     No Information Rate : 0.7808
##     P-Value [Acc > NIR] : 0.5034
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
```

```
##          Pos Pred Value : 0.7808
##          Neg Pred Value :    NaN
##              Prevalence : 0.7808
##          Detection Rate : 0.7808
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : No
##
```

This model has a sensitivity of 100% and specificity of 0%, meaning we're predicting the days with no rain correctly 100% of the time and the days with rain 0% of the time. Obviously this is unacceptable despite the high accuracy of the model. Consequently, we will use the F-Score to compare our models since it takes both the sensitivity and specificity into consideration.

```
F_meas(guess_y_hat, reference = test_set$RainTomorrow)
```

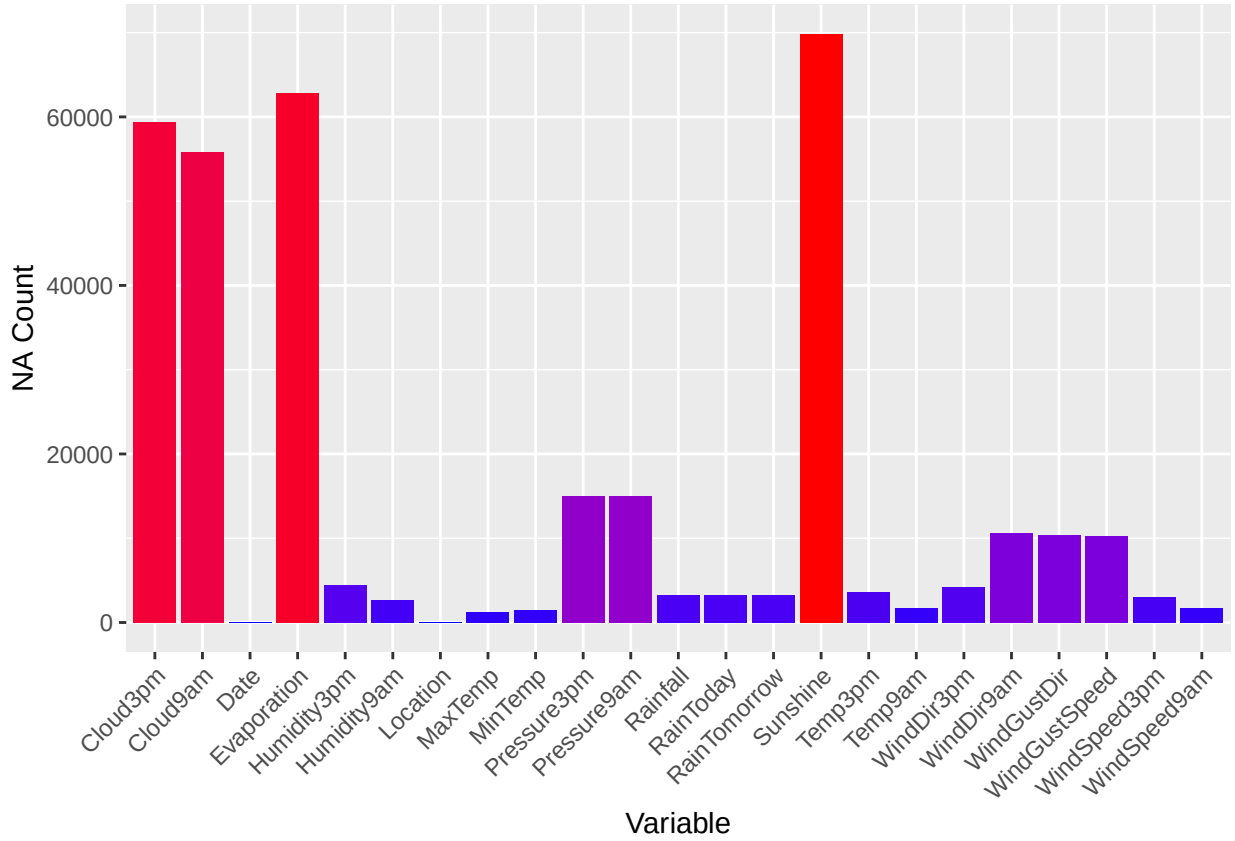
```
## [1] 0.876935
```

So an F-Score of 0.876935 is the metric we will attempt to beat.

| Method | F-Score |
|---|---|
| Naive Model | 0.876935 |

Before we can begin training any machine learning models, we need to clean our data. We begin this process by observing how many missing observations ("NA") appear for each variable.

```
na_count <- sapply(rain_data, function(x) sum(length(which(is.na(x)))))
data.frame(names = names(na_count), na_count) %>%
    ggplot(aes(x = names, y = na_count, fill = na_count)) +
    geom_bar(stat = "identity") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1),
          legend.position = "none") +
    xlab("Variable") +
    ylab("NA Count") +
    scale_fill_gradient(low = "blue", high = "red")
```




We can clearly see four variables with significant amounts of missing observations: Cloud3pm, Cloud9am, Evaporation, and Sunshine. We will simply omit these variables from our models since attempting to infer the missing values or omitting such a large amount of observations may be detrimental to our results. We will, however, infer the missing observations in the RainToday and RainTomorrow variables by making the reasonable assumption that a lack of report of rain means no rain for that day. We will then omit the remaining observations with missing values and factorize all discrete variables.

With our data cleaned, we can begin training machine learning models. The first we will train is a linear regression model.

```
linear_fit <- train(RainTomorrow ~ ., data = train_set, method = "glm")
```

We now acheive an F-Score of 0.8993637.

In this model, we included all variables for training. As we progress through more complicated machine learning models we will find that doing so is too computationally taxing for most personal computers, so at this point it would be prudent for us to consider which variables serve as the most important predictors and only include those going forward. We can accomplish this by using the varImp function.

```
varImp(linear_fit)
```

This tells us that the RainToday, Humidity3pm, and Rainfall variables account for the majority of our linear regression model's predictive power. We will use only these variables while training our remaining sample of machine learning models.

```
lda_fit <- train(RainTomorrow ~ RainToday + Humidity3pm + Rainfall, data = train_set, method = "lda")
dt_fit <- train(RainTomorrow ~ RainToday + Humidity3pm + Rainfall, data = train_set, method = "rpart")
rf_fit <- train(RainTomorrow ~ RainToday + Humidity3pm + Rainfall, data = train_set, method = "rf")
```

| Method | F-Score |
|---|---|
| Naive Model | 0.876935 |
| Linear Regression Model | 0.899364 |
| Linear Discriminant Analysis | 0.897190 |
| Decision Tree Model | 0.899777 |
| Random Forest Model | 0.898139 |

It's apparent that a decision tree achieves the highest F-Score. A review of the training function's documentation shows that this model has one tuning parameter: the Complexity Parameter. We will attempt to optimize this parameter in order to increase our model's F-Score.

```
dt_fit <- train(RainTomorrow ~ RainToday + Humidity3pm + Rainfall, data = train_set, method = "rpart",
                tuneGrid = data.frame(cp = seq(0, 0.05, 0.005)))
```

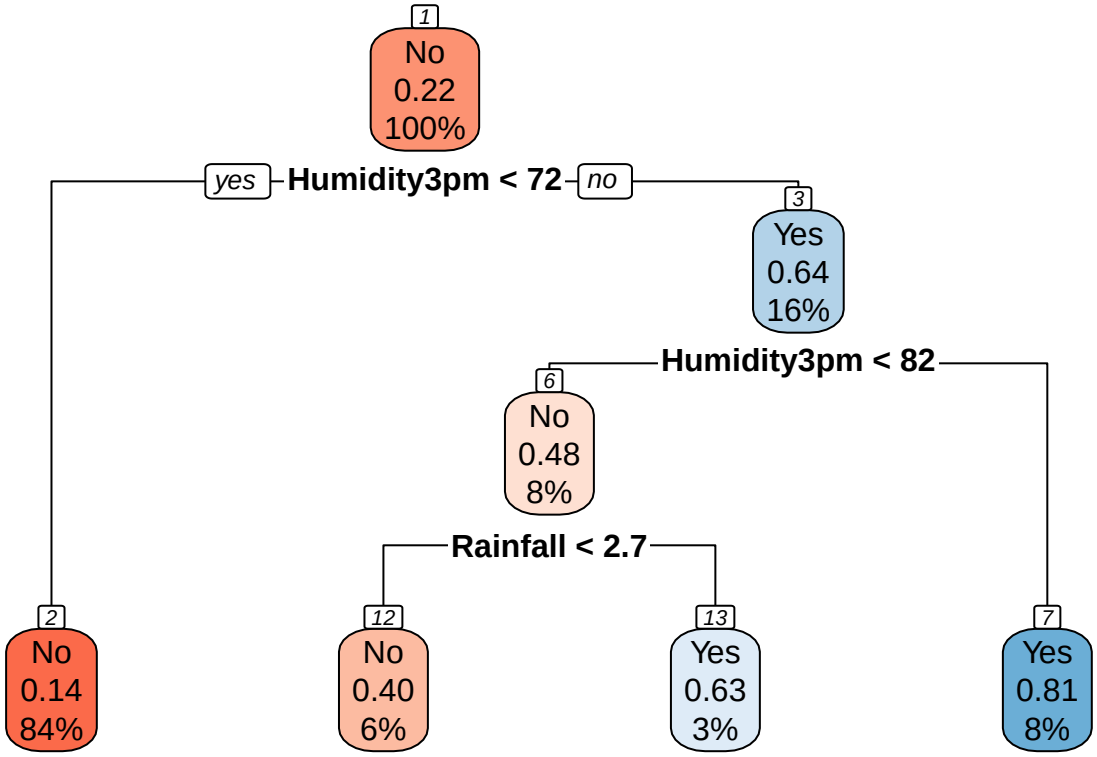
Doing so shows that the optimized model, using a Complexity Parameter of 0.005, still results in an F-Score of 0.899777.

## Results

Our analysis determined that a decision tree model provided the greatest increase in predictive power over the naive model by increasing the F-Score from 0.876935 to 0.899777, and accuracy from 0.7799 to 0.8319. We also found that the default decision tree parameters were already optimized as optimizing the model's Complexity Parameter resulted in an identical F-Score.

The final model is visualized below.

```
rpart.plot(dt_fit$finalModel, box.palette="RdBu", nn=TRUE)
```




## Conclusion

We were able to develop a machine learning model that provides a moderate increase in accuracy when predicting whether or not it will rain tomorrow in Australia. Perhaps the most notable limitation of this model lies in the fact that it can predict rain only for the country as a whole and not particular cities. While the city variable was determined to be statistically insignificant by our analysis, a rain prediction model which does not consider specific location could be regarded as largely useless. Future analysis should group observations by the location variable or even consider building separate models for each city.

Additional analysis might also include the transformation, standardization, or binning of continuous variables prior to the determination of their importance. Such techniques could reveal additional insights in our data that may have been missed.

Overall, this decision tree model provided a marked increase in rain prediction accuracy and could serve as a respectable baseline metric for future models to improve upon.