TidyML

Create Dataset

We will first import the dataset from palmerpenguins library

```r
load_all()
```

```
i Loading TidyML
Loading required package: tidyverse

-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
df <- palmerpenguins::penguins %>%
  na.omit() %>%
  select(-year) %>%
  filter(species == "Adelie" | species == "Gentoo") %>%
  mutate(species = droplevels(species))
```

We have omitted the ***year*** column and selected only two species (*Adelie* and *Gentoo*) for binary classification.

Preprocessing Step

We will first preprocess the data set using the **transformer** function. We will pass the dataset along with the formula for our problem. The preprocessing step requires to specify which columns are going to be preprocessed:

- Numerical columns will be normalized by z-score
- Categorical columns will be one-hot encoded

In our case, we will preprocess all numerical columns and all categorical columns using the **all** keyword:

```
tidy_object = transformer(df,
                          "species ~ .",
                          norm_num_vars = "all",
                          encode_cat_vars = "all"
                          )
```

The function returns an object with the preprocessing information stored in it. We will need this object for the subsequent steps.

Model Definition

In this step will define the model we will use as well as specify the hyperparameters for the model. We will use the **create_models** function. We will pass:

- tidy_object
- name of the model we will use (**Random Forest**)
- Hyperparameter list (this can be ranges to tune or a fixed value)
- task (**classification**)

```
tidy_object <- create_models(tidy_object,
                             "Random Forest",
                             list(
                                    mtry = c(2,3),
                                    trees = 2
                             ),
                             "classification"
                             )
```

This will again return a **tidy_object** that we will need in the subsequent steps.

Hyperparameter Tuning

Now we will tune the hyperparameters with the **model_tuning** function. We can use either **Bayesian Optimization** or a grid search with cross validation (**Grid Search CV**). We will also specify the metric we want for model selection, in our case, the area under the receiver operation characteristic curve (**roc_auc**):

```
tidy_object <- model_tuning(tidy_object,
                            tuner = "Grid Search CV",
                            metrics = "roc_auc"
                            )
```

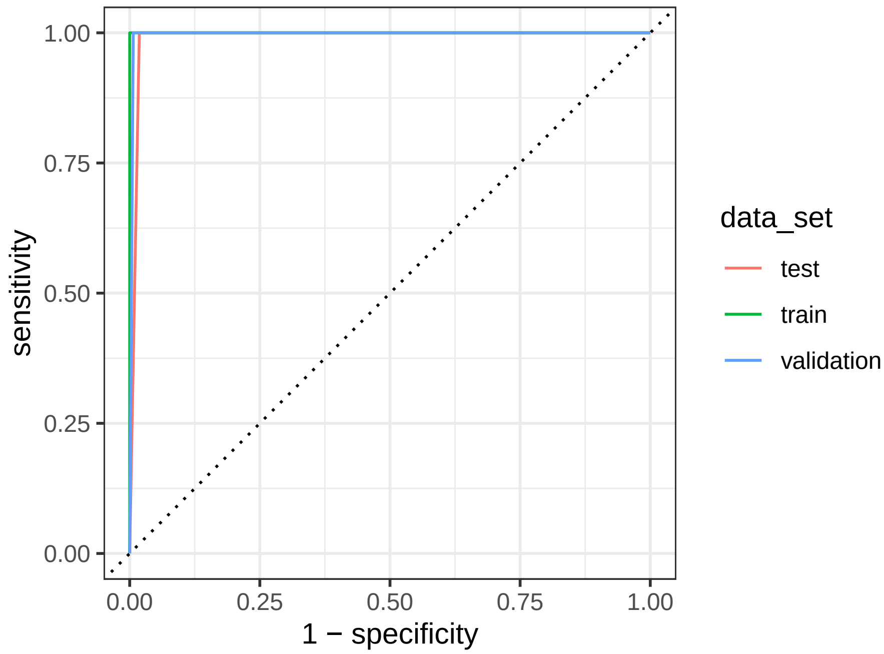
Results

Now we can extract the results from our model using the **get_results** function. This function allows to get the performance of our model using a wide variety of metrics as well as using visual plots. In our case we will want:

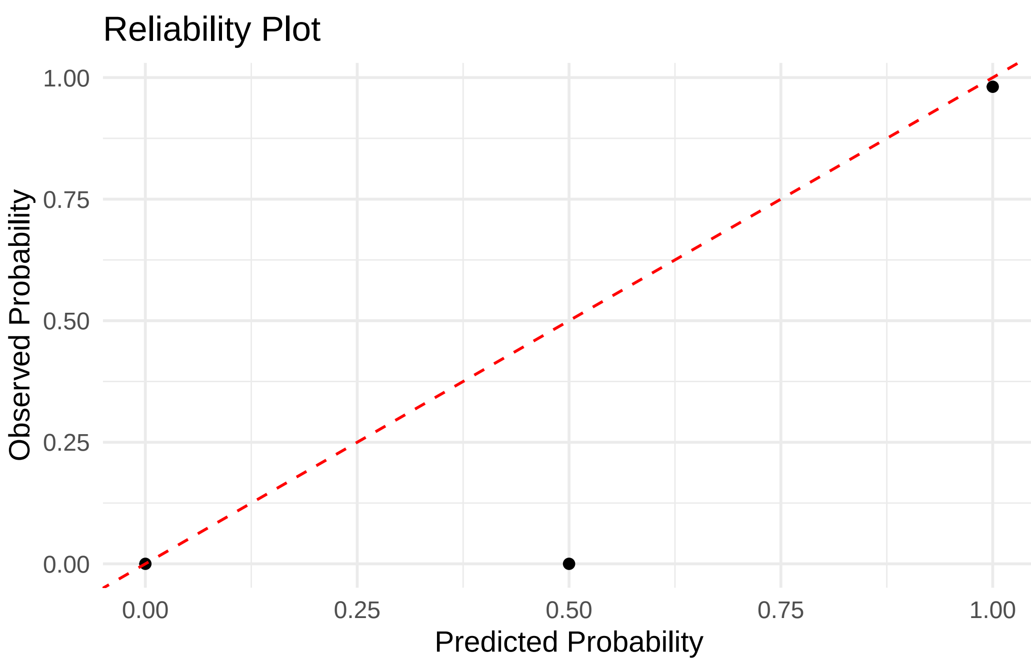
- **summary**
- Receiver Operation Characteristic curve (**roc_curve**)
- Confusion Matrix (**confusion_matrix**)
- Calibration curve also known as Reliability plot (**reliability_plot**)

```
results <- get_results(tidy_object,
                       summary = T,
                       roc_curve = T,
                       confusion_matrix = T,
                       reliability_plot = T)
```

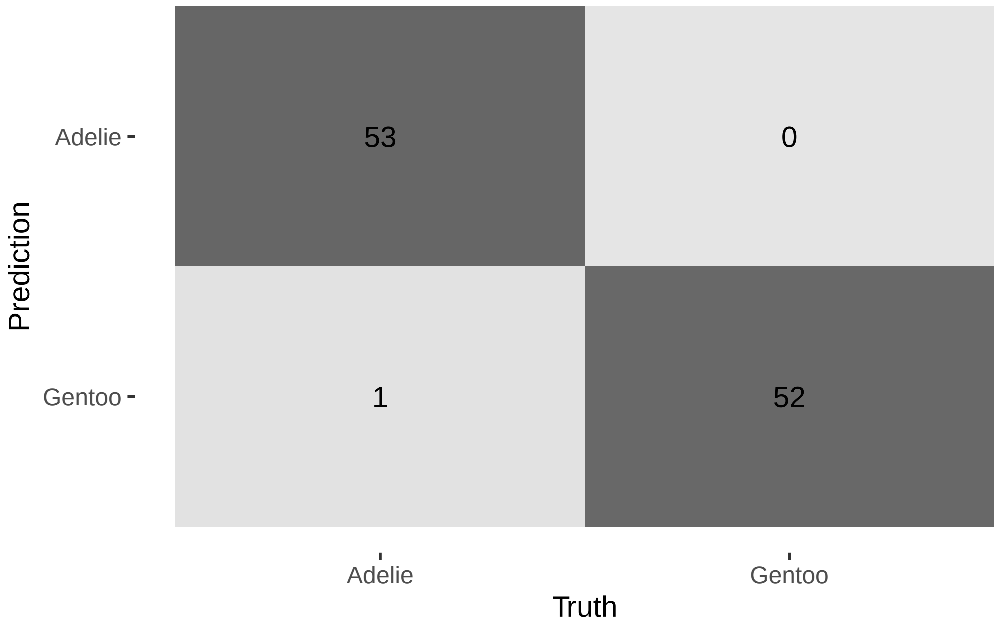
```
  Accuracy Balanced_Accuracy Precision    Recall Specificity Sensitivity
1 0.990566         0.9907407         1 0.9814815           1   0.9814815
      Kappa  F1_score       MCC   J_index Detection_Prevalence   AUC_ROC
1 0.9811321 0.9906542 0.9813068 0.9814815                  0.5 0.9907407
    AUC_PR Gain_Capture Brier_Score
1 0.990566    0.9814815   0.9764151
```



1.00
0.75
0.50
0.25
0.00
sensitivity
0.00
0.25
0.50
0.75
1.00
1 − specificity
data_set
test
train
validation



Reliability Plot
1.00
0.75
0.50
0.25
0.00
Observed Probability
0.00
0.25
0.50
0.75
1.00
Predicted Probability



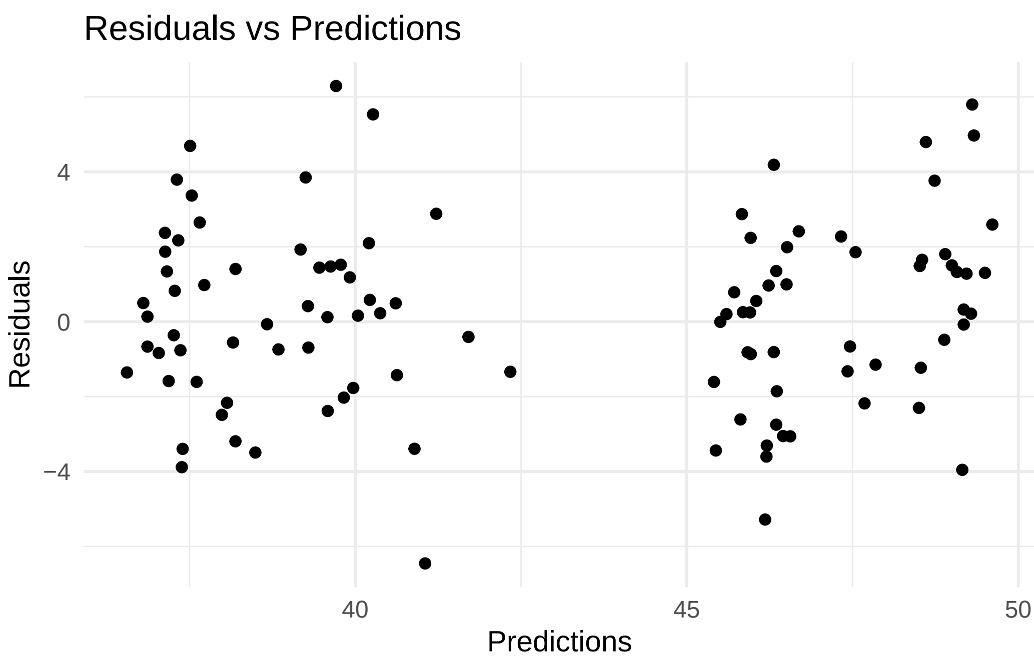
Using the pipe operator (%>%)

The previous process can be executed in a single "statement" using the %>% operator:

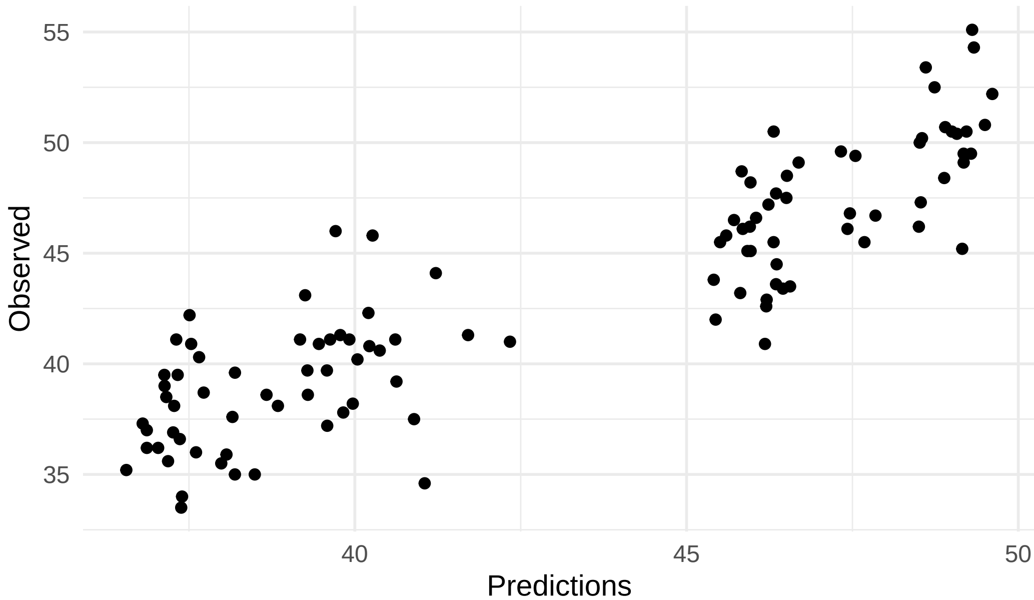
```
results <- transformer(
                          df,
                          "bill_length_mm ~ .",
                          norm_num_vars = "all",
                          encode_cat_vars = "all"
                          ) %>%

            create_models(
                             model_names = "Random Forest",
                             hyperparameters = list(
                                    mtry = c(2,3),
                                    trees = 100
                             ),
                             task = "regression"
                             ) %>%

            model_tuning(
                          tuner = "Grid Search CV",
                          metrics = "rmse"
                            ) %>%
```

```
get_results(summary = T,
            scatter_predictions = T,
            scatter_residuals = T,
            residuals_dist = T)
```

```
      RMSE      MAE     MAPE        MPE       CCC    SMAPE     RPIQ      RSQ
1 2.441733 1.940619 4.541062 0.06609305 0.8752672 4.538351 3.409464 0.783859
```



Observed vs Predictions



Residuals vs Predictions