# Handin 3

Beate, Marcus, Jonas, Wilmer

2024-04-11

First load the following libraries

```
packages <- c("tidyverse", "tidymodels", "dplyr", "rsample", "ranger", "ggplot2")

for (package in packages) {
  if (!require(package, character.only = TRUE)) {
    install.packages(package)
    library(package, character.only = TRUE)
  }
}
```

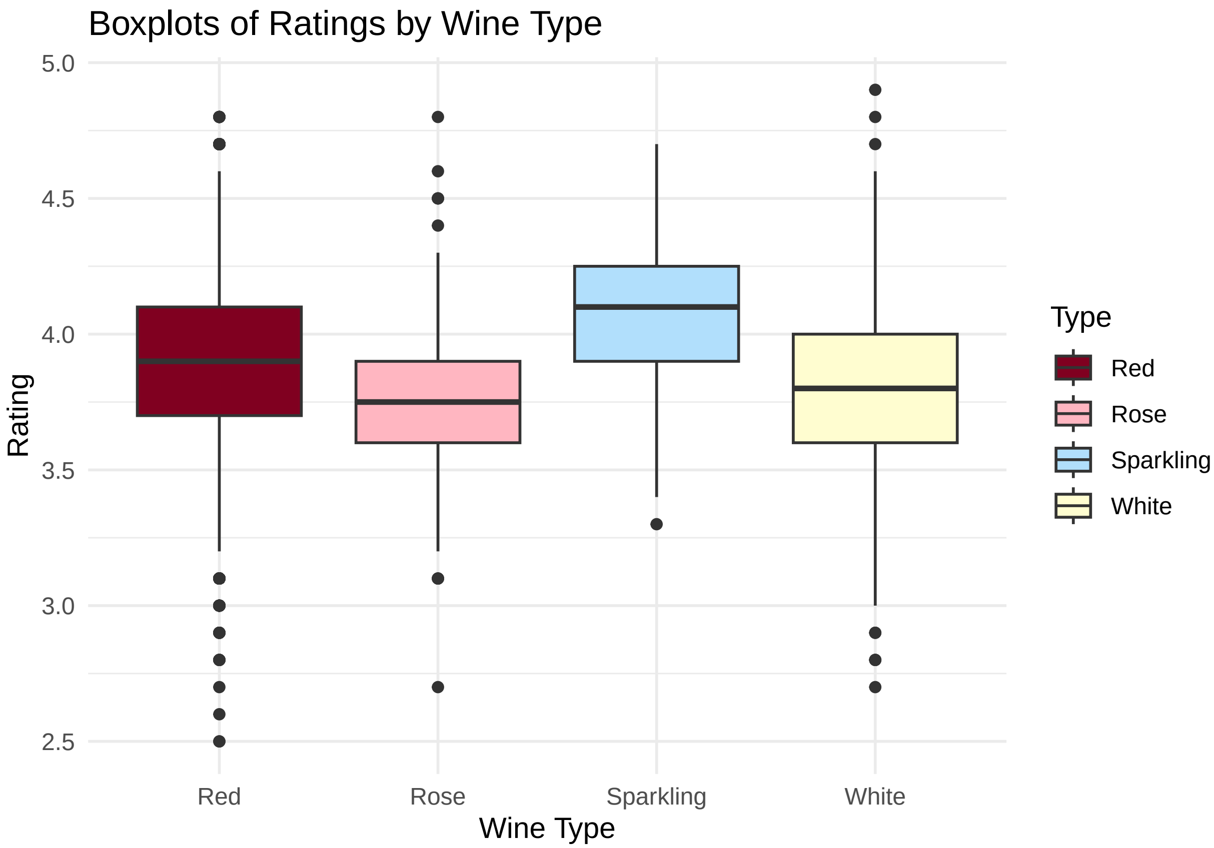
The data is fetched from "https://shorturl.at/crAK3". This dataset contains files for four wine styles: red, white, rosé, and sparkling. Each of these has eight columns, including the "NumberOfRatings' ' column indicating the count of people who rated the wine.

Each dataset is augmented with a "Type" column indicating the wine type, and unnecessary columns like "Name", "Winery", and "Region" are removed for analysis.

```
red <- read.csv("Red.csv")
white <- read.csv("White.csv")
rose <- read.csv("Rose.csv")
sparkling <- read.csv("Sparkling.csv")

red$Type <- "Red"
white$Type <- "White"
rose$Type <- "Rose"
sparkling$Type <- "Sparkling"

data <- rbind(red, white, rose, sparkling) %>% select(-Name, -Winery, -Region)
data$Country <- as.factor(data$Country)
data$Year <- as.factor(data$Year)
data$Type <- as.factor(data$Type)

data <- data %>% filter(Year != "N.V.")
head(data)
```

```
##   Country Rating NumberOfRatings Price Year Type
## 1  France    4.2             100 95.00 2011  Red
## 2  France    4.3             100 15.50 2017  Red
## 3   Italy    3.9             100  7.45 2015  Red
## 4   Italy    3.5             100  8.72 2019  Red
## 5 Austria    3.9             100 29.15 2016  Red
## 6  France    3.7             100 19.90 2017  Red
```

```
wine_colors <- c(
  "Red" = "#800020",
  "White" = "#FFFDD0",
  "Rose" = "#FFB6C1",
  "Sparkling" = "#b1dffc"
)
ggplot(data, aes(x = Type, y = Rating, fill = Type)) +
  geom_boxplot() +
  scale_fill_manual(values = wine_colors) +
  labs(x = "Wine Type", y = "Rating", title = "Boxplots of Ratings by Wine Type") +
  theme_minimal()
```




Here, we split the combined dataset into training and testing sets. This step ensures that our models are trained on a portion of the data and evaluated on unseen data.

```
set.seed(123)
cell_split <- initial_split(data, prop = 3 / 70, strata = "Type")

train_data <- training(cell_split)
test_data <- testing(cell_split)
```

We construct a random forest regression model with 1000 trees and fit it to the training data. Additionally, we perform cross-validation with 10 folds to assess the model's performance across different subsets of the data.

```
rf_mod <- rand_forest(mode = "regression", trees = 1000) %>%
  set_engine("ranger")
rf_wf <-
```

```
  workflow() %>%
  add_model(rf_mod) %>%
  add_formula(Rating ~ .)
rf_fit <- rf_wf %>% fit(data = train_data)

folds <- vfold_cv(train_data, v = 10)

rf_fit_rs <-
  rf_wf %>%
  fit_resamples(folds)

rf_metrics <- collect_metrics(rf_fit_rs)
rf_metrics
```

```
## # A tibble: 2 x 6
##   .metric .estimator  mean     n std_err .config
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.208    10 0.00602 Preprocessor1_Model1
## 2 rsq     standard   0.494    10 0.0322  Preprocessor1_Model1
```

The cross-validation results for the random forest model demonstrate a RMSE of 0.207 with a standard error of 0.00593 and an R-squared of 0.495, indicating strong predictive performance. For comparison, we specify and fit a linear regression model to the training data. Similar to the random forest model, cross-validation is performed to compare these models performance.

```
lm_recipe <- recipe(Rating ~ ., data = train_data)

lm_mod <- linear_reg() %>% set_engine("lm")

lm_wf <- workflow() %>%
  add_recipe(lm_recipe) %>%
  add_model(lm_mod)

lm_fit <- lm_wf %>% fit(data = train_data)

lm_fit_rs <- lm_wf %>% fit_resamples(folds)

lm_metrics <- collect_metrics(lm_fit_rs)
```

Here, we compare the performance metrics, between the random forest and linear regression models. This comparison provides insights into which model may be more suitable for predicting wine ratings.

```
comparison_metrics <- bind_rows(
  lm_metrics %>% mutate(Model = "Linear Regression"),
  rf_metrics %>% mutate(Model = "Random Forest")
)
comparison_metrics
```

```
## # A tibble: 4 x 7
##   .metric .estimator  mean     n std_err .config              Model
##   <chr>   <chr>      <dbl> <int>   <dbl> <chr>                <chr>
## 1 rmse    standard   0.249     3 0.0172  Preprocessor1_Model1 Linear Regression
## 2 rsq     standard   0.355     3 0.0586  Preprocessor1_Model1 Linear Regression
## 3 rmse    standard   0.208    10 0.00602 Preprocessor1_Model1 Random Forest
## 4 rsq     standard   0.494    10 0.0322  Preprocessor1_Model1 Random Forest
```

The Random Forest model has a lower RMSE and higher RSQ compared to the Linear Regression model, suggesting that it provides a more accurate prediction. \ Moreover, we further enhance the performance of our models by fine-tuning hyperparameters. Using grid search, we can fine-tune hyperparameters and select the best model configuration. By fine-tuning the Random Forest model, we get various combinations of hyperparameters. # https://juliasilge.com/blog/sf-trees-random-tuning/

```
tune_spec <- rand_forest(
  mtry = tune(),
  trees = 1000,
  min_n = tune()
) %>%
  set_mode("regression") %>%
  set_engine("ranger")

tree_wf <- workflow() %>%
  add_model(tune_spec) %>%
  add_formula(Rating ~ .)

tree_res <- tune_grid(
  tree_wf,
  resamples = folds,
  grid = 20
)
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
mtry_range <- tree_res$.metrics[[1]] %>%
  select(mtry) %>%
  summarise(
    min_mtry = min(mtry),
    max_mtry = max(mtry)
  )

min_n_range <- tree_res$.metrics[[1]] %>%
  select(min_n) %>%
  summarise(
    min_min_n = min(min_n),
    max_min_n = max(min_n)
  )

rf_grid <- grid_regular(
  mtry(range = mtry_range),
  min_n(range = min_n_range),
  levels = 5
)

tree_res <- tune_grid(
  tree_wf,
  resamples = folds,
  grid = rf_grid
)

tree_res %>%
  show_best(metric = "rmse")
```

```
## # A tibble: 5 x 8
```

```
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     3    20 rmse    standard   0.206    10 0.00565 Preprocessor1_Model13
## 2     3    29 rmse    standard   0.206    10 0.00548 Preprocessor1_Model18
## 3     4    29 rmse    standard   0.206    10 0.00517 Preprocessor1_Model19
## 4     3    11 rmse    standard   0.206    10 0.00564 Preprocessor1_Model08
## 5     4    20 rmse    standard   0.207    10 0.00530 Preprocessor1_Model14
```

The show_best() function is used to identify the model with the lowest RMSE across different configurations. We then select the best model and perform the final fit using the optimized workflow, ensuring that the model is trained on the entire training dataset.

```
best_tree <- tree_res %>%
  select_best(metric = "rmse")

final_wf <-
  tree_wf %>%
  finalize_workflow(best_tree)

final_fit <-
  final_wf %>%
  last_fit(cell_split)

final_fit %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       0.199 Preprocessor1_Model1
## 2 rsq     standard       0.560 Preprocessor1_Model1
```

These are the metrics of our selected configuration, which should be performing reasonably in predicting wine rating.