Disney World Ride Wait Times

Gus Lipkin ~ glipkin6737@floridapoly.edu

April 27, 2022

Table of Contents

Introduction

Most people like going to Disney World for one reason or another. It could be the atmosphere, character meet and greets, themeing, fireworks, exclusive dining options, rides, or any number of other reasons. Of the people that don't like going to Disney World, I'd hazard a guess that the large crowds and long ride wait times are at the top of most people's lists. While Disney does offer a way to cut long lines through its Lightning Lane program, it is an additional cost, limited use item, and is not available for every attraction (Disney 2022). There are some techniques to avoid the lines such as arriving to the park early, getting in line during a parade or fireworks, or visiting at different times of day or year. These techniques can show some success, but once too many people catch on, then the technique becomes useless. This is similar to how on a three lane highway, one lane may be moving faster until everyone wants to get into the "fast" lane, and then that lane becomes the slow lane. There are many websites that provide some level of insight into current and historical ride wait times such as touringplans.com, wdwpassport.com, and thrill-data.com. However, none attempt to forecast ride wait times to give their users a competitive advantage in choosing their next ride so as to minimize ride wait times. Although my method has not yet been tested as of writing, I will analyze historical data and publish decision trees that will empower Disney goers to choose their next ride based on current park data that is immediately available to them.

Literature Review

In Spring 2020, Lipkin et al Lipkin, Skoglund, and Pierstorff (2020) worked to predict the best time of year and day to visit Disney World based on historical seasonal trends. Based on an extensive exploration of the data and data visualization techniques, they recommended that September was the best time of year to visit, particularly a Wednesday afternoon. They suggested that this was mostly due to the fact that nearly every school in the country is in session during this time. They also considered that it may also have to do with the weather still being Florida's hot and rainy season, which is less than ideal for outdoor theme parks. In their next steps, they suggested using supervised machine learning to predict actual ride wait times based on posted ride wait times.

Nearly a year earlier, Mendoza, Wu, and Leung (2019) also studied data from Touring Plans (2021), and went so far as to develop a multiple regression model that explained approximately 80% of daily wait time variation with 15 minute average error. They took particular note of the seasonal lows in September and highs in December, but did not go so far as to make suggestions on when to visit Disney World based on their analysis.

Data

Overview

The bulk of the data used in this analysis was provided by Touring Plans (2021). It consists of fourteen main datasets, one each for the fourteen different attractions with data recorded. Each of these consists of a park date variable based on the day which the park opened, especially important for days where parks are open past midnight, a datetime variable that says when the data was recorded, and an actual and predicted wait time variable. Due to limitations in how Touring Plans recorded the data, there is either only an actual wait time or posted wait time, but not both for any given date time.

Touring Plans also provided an expansive metadata dataset with information from each day. While I won't discuss every variable here, I will highlight the some variables of note. Included are the ticket season, proximity to a holiday, historical maximum and minimum temperature as well as precipitation, opening and closing times, time of any fireworks, parades, or other shows and proportion of schools in session in different regions across the United States.

To supplement the weather data provided in the Touring Plans metadata dataset, I downloaded hourly weather data for 28.3772° N, 81.5707° W, the coordinates for Disney World from copernicus.eu. Using the Touring Plans data as a reference, I downloaded precipitation and temperature data for every hour that the parks are open.

All Touring Plans data was in `csv` files. However, after downloading, I noticed that the data did not go as far back as the data had for Lipkin, Skoglund, and Pierstorff (2020) and found that data. I combined the old data and the new data and dropped duplicate rows. The new metadata file also included twelve new columns. The volume of data needed from Copernicus exceeded their single download limit so I split the download request roughly in half by year. Each download was a `netcdf` file. In all cases, the data was combined in R.

Summary Statistics

A more complete set of summary statistics are provided in my past paper, Lipkin, Skoglund, and Pierstorff (2020), but I have elected to update some statistics with the new data. The boxplot for actual ride wait times shows that most rides have median wait times under 30 minutes with the highest points reaching nearly 150 minutes for Flight of Passage. Because posted wait times are in 5 minute increments, the results outliers appear more uniform. There's a sort of bimodal split here with some rides hovering around 30 minutes and others around 60 minutes. However, there is an apparent difference between the two. When we overlay the actual and posted distributions, we see that there is a good split between the two.

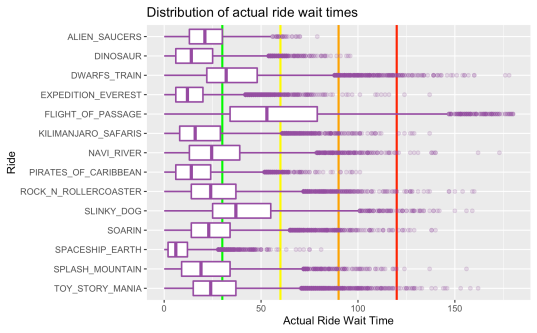


Figure 1: Actual Wait Times

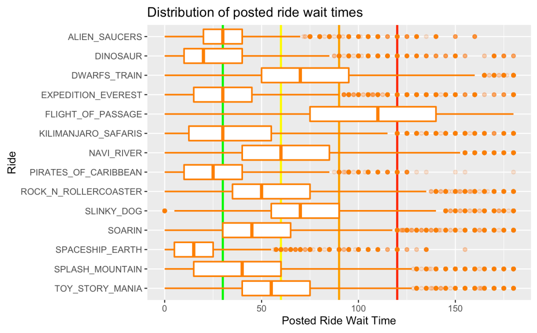


Figure 2: Posted Wait Times

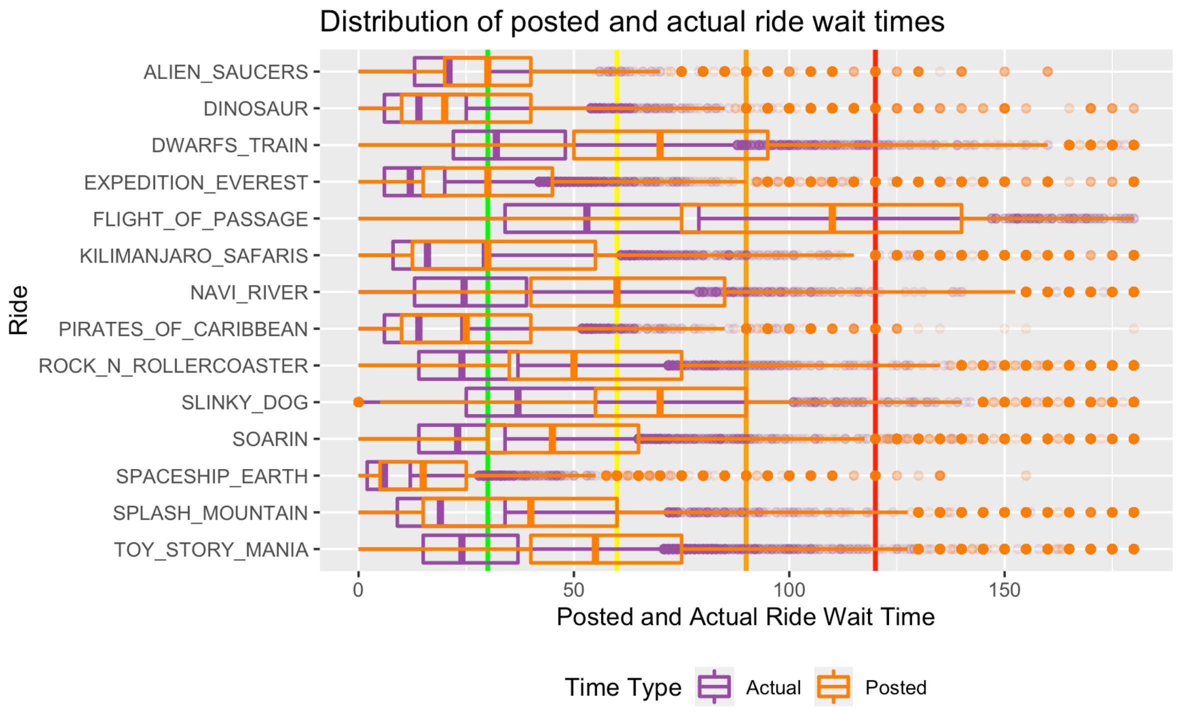


Figure 3:

The difference between the posted and actual ride wait times seems rather large in the merged boxplot. To determine if there is a true difference between the two datasets, we can perform a t.test. To prevent the t-test from misconstruing our time values, we must first remove outliers before performing the test. With a p-value less than $2.2 * 10^{-16}$, we can be reasonably sure that there is a difference between the posted and actual wait times. By checking back with the overlayed boxplot, we can see that posted wait times are frequently overestimates of the actual ride wait times. Nevertheless, this does not mean that the posted wait time cannot be a good predictor of the actual wait time.

```
t = -327.48, df = 135564, p-value < 2.2e-16

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

-21.93920 -21.67815

sample estimates:

mean of x mean of y

24.61871 46.42739
```

Research Design

Aggregating the Data

There are two main issues with the wait time data from Touring Plans. The first is that data is not collected at consistent intervals and the second is that if the actual wait time is recorded, the posted wait time is not also recorded at the same time. These two issues then compound into smaller issues. The first of which is that traditional time series techniques become much more difficult to apply if data is not recorded at consistent intervals. The second is that wait times cannot be well compared across rides because there may not be a recording at the same time. While there is an indicator that a ride may be closed, there are also sometimes large gaps between recordings that are not well explained. To combat this, I took the mean recording every five minutes. I chose five minute intervals for two reasons. The first is that while line length can change significantly in five minutes, it is not likely, and the second is that five minutes is just enough time to really hoof it across any one of the four parks.

Choosing a Model

In all models, I would want the actual wait time to be lagged so that I am predicting what the ride wait time will be in five minutes. This way, model users will be able to get to the line before other people that see the line is getting shorter.

In *Final Project for Intro to Data Science*, I suggested that linear regression would be an appropriate next step. A linear regression would be more flexible with the sparse data. However, variable selection would be difficult such a large number of independent variables. In addition, linear regressions can be difficult to understand and implement while on the go at a theme park. Of course, nowadays everyone does have a calculator in their pocket, but I digress.

Logistic regression would follow the same pitfalls as linear regression, but would give the user even less information. Rather than predicting an actual wait time, I would only be able to predict if the next wait time would be less than the current wait time or not. The same problem presents with other classifier methods such as support vector machines. Multiple logistic regression would give us the flexibility we are looking for, it takes away a sense of agency from the user because it would suggest which ride would have the shortest line, rather than what the wait time could be for each ride.

This leaves us with decision trees. There is no doubt that random forest would produce better results than a plain decision tree, but interpreting a random forest model is nearly impossible and it cannot be used for prediction without access to the model. Not to mention, the amount of computation time it could take to run. This is of special concern because although cell phones are becoming more and more powerful every year, most people hope for their phones to last all day at Disney World so they don’t need to find an outlet or carry an external battery with them.

Analysis

Analysis Process

Once the data was properly cleaned and aggregated, there were a few more steps needed before beginning the analysis. Another metadata dataframe was created with the ride name, opening date, whether or not the ride has a water splash aspect or is indoors, how long the ride lasts, the wait time per hundred people in line, and the park location.

With all my data prepared, my last step was to lead each actual wait time by one. This allows me to forecast the what the wait time will be in five minutes for each ride. From there, I split the data into a training and test set with an 80/20 split.

I used nested lapply statements to generate the decision trees in an efficient manner. The outer level is supplied a character vector that represents each of the four parks in Walt Disney World. The outer level then creates a vector of variables that are only relevant to the current park. For example, if the current park is Magic Kingdom, there are no rides from Animal Kingdom or any of the other parks included. The inner layer is then supplied a vector of ride names for the park given in the outer layer. Each ride is then used as the response variable in an rpart decision tree (Therneau and Atkinson 2022). At this point, two things are done. The decision tree is printed in graphical format, and the test set is used to predict new wait times. The inner lapply then returns a single row data.table with the ride name, mean squared error, mean accuracy as measured by the predicted value being within $\pm15\%$ of the actual wait time value, and the mean lower and mean upper limit for the 15%.

Results

Based on the mean accuracy from the process described in the previous section, my models are accurate within $\pm15\%$ .37 of the time. The highest accuracy is .34 for the Alien Saucers ride at Hollywood Studios and the lowest is .13 for Spaceship Earth.

Although it may seem strange to account for a 30% window and call that a success, in practice it makes quite a bit of sense. Take my least favorite ride, Expedition Everest, for example, the mean actual reported wait time is a little over twelve minutes. A 15% buffer on either side means that you could be waiting for between ten to fourteen minutes. In the grand scheme of things, no one is going to be particularly bothered by waiting an extra two minutes. Even for rides with longer wait times such as Slinky Dog Dash, what is an extra seven minutes on top of thirty-nine minutes?

Table 1: Summary of Decision Tree Results for Predicting Test Data

Ride Name	MSE	Accuracy	Error Direction	Mean Wait Time	Mean Lower Limit	Mean Upper Limit
DWARFS TRAIN	356.04	0.28	0.45	37.38	31.77	42.99
PIRATES OF CARIBBEAN	77.53	0.19	0.44	15.28	12.98	17.57
SPLASH MOUNTAIN	153.13	0.20	0.42	20.59	17.50	23.67
ALIEN SAUCERS	75.04	0.34	0.45	24.13	20.51	27.75
ROCK N ROLLERCOASTER	200.88	0.25	0.45	26.91	22.87	30.95
SLINKY DOG	291.05	0.30	0.45	41.77	35.51	48.04
TOY STORY MANIA	354.43	0.25	0.42	29.77	25.30	34.24
DINOSAUR	107.24	0.18	0.41	16.29	13.84	18.73
EXPEDITION EVEREST	76.91	0.19	0.42	14.55	12.37	16.73
FLIGHT OF PASSAGE	1391.43	0.30	0.45	70.96	60.31	81.60
KILIMANJARO SAFARIS	167.78	0.16	0.42	19.48	16.56	22.40
NAVI RIVER	207.13	0.25	0.45	29.37	24.97	33.78
SOARIN	136.73	0.26	0.44	25.58	21.74	29.42
SPACESHIP EARTH	34.37	0.13	0.39	7.66	6.51	8.81

The distribution in the levels of model accuracy is shown in Figure 1. The mean accuracy level is not particularly amazing. However, it is good that there are no outliers in the model accuracy levels. It does appear that the distribution is slightly right-skewed, but we would need more information to make a true assumption. It does follow that some rights might be more accurate than others.

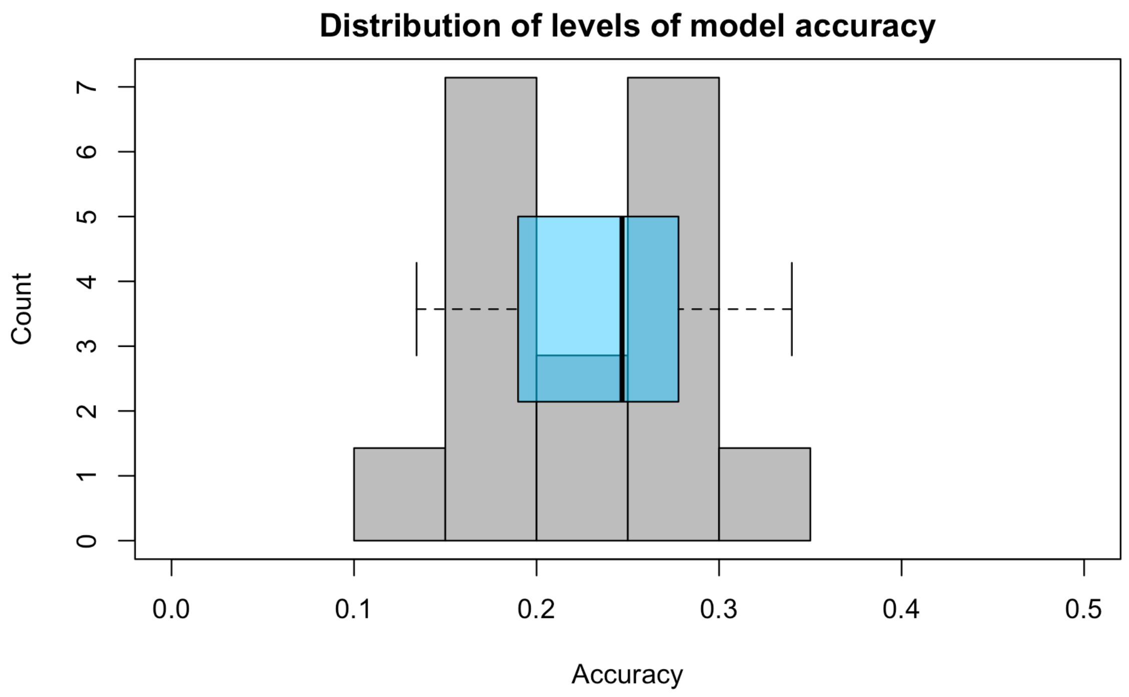


Figure 1: Distribution of levels of model accuracy

Next Steps

There are two projects I would like to take to further this project. The first is to create a shiny app so that Disney-goers can plug in current park information to get a recommendation on ride wait times. By making an app, I will be able to give users a ranking of shortest to longest wait times rather than them needing to check each ride at a time. The second project is to build the model to a stacking ensemble model that will hopefully allow the strengths of each underlying model to shine through. In addition, since the project will already be a shiny app, I will be able to use more involved models such as random forest, linear regression, and any other models I may feel appropriate.

Conclusion

Disney has entire teams of people working on just this kind of analysis and I'm only one person who has other things to do. Nevertheless, I am happy with my results. I had to overcome issues with sparse and inconsistent data and find a way to reasonably forecast ride wait times five minutes out. I was able to accomplish this with a reasonable measure of accuracy just over 23% using individual decision trees for each ride. That said, achieving this level of accuracy did require some sacrifices in terms of precision with a $\pm 15\%$ buffer. I am most pleased with my 34% accuracy for the Alien Saucers ride and plan to test out my theory as soon as I can.

References

2022. *Copernicus Climate Data Store*. Copernicus. https://doi.org/10.24381/cds.adbb2d47.

Disney. 2022. "Disney Genie Lightning Lane - Save Time Waiting in Line ..." 2022. https://disneyworld.disney.go.com/genie/lightning-lane/.

Dowle, Matt, and Arun Srinivasan. 2021. *Data.table: Extension of 'Data.frame'*. https://CRAN.R-project.org/package=data.table.

Lipkin, Gus. n.d. *Dewey: An r Library for a Variety of Things*.

Lipkin, Gus, Hailey Skoglund, and Alex Pierstorff. 2020. "Final Project for Intro to Data Science." *Disney_ds*, April. https://github.com/guslipkin/disney_ds.

Mendoza, Dayanira, Wenbo Wu, and Mark T Leung. 2019. "Predicting the Expected Waiting Time of Popular Attractions in Walt Disney World." *Journal of Undergraduate Research & Scholarly Work*. https://rrpress.utsa.edu/bitstream/handle/20.500.12588/99/JURSW_Vol_6.Mendoza.pdf?sequence=1.

R Core Team. 2020. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Therneau, Terry, and Beth Atkinson. 2022. *Rpart: Recursive Partitioning and Regression Trees*. https://CRAN.R-project.org/package=rpart.

Touring Plans. 2021. *Disney World Ride Wait Time Datasets*. *Touringplans.com*. https://www.touringplans.com/walt-disney-world/crowd-calendar/#DataSets.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Appendix A (Code and Output)

Load Libraries

```r
library(dewey)
library(data.table)
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
v ggplot2 3.3.5     v purrr   0.3.4
v tibble  3.1.6     v dplyr   1.0.8
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1
```

```
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::between()   masks data.table::between()
x dplyr::filter()    masks stats::filter()
x dplyr::first()     masks data.table::first()
x dplyr::lag()       masks stats::lag()
x dplyr::last()      masks data.table::last()
x purrr::transpose() masks data.table::transpose()
```

```r
library(tidync)

library(tree)
library(randomForest)
```

```
randomForest 4.7-1
```

```
Type rfNews() to see new features/changes/bug fixes.
```

```
Attaching package: 'randomForest'

The following object is masked from 'package:dplyr':

    combine

The following object is masked from 'package:ggplot2':

    margin
```

```r
library(caret)
```

```
Loading required package: lattice

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift
```

```r
library(rpart.plot)
```

```
Loading required package: rpart
```

Load initial data

Load File List and Create Helper Functions

```
# load the wait time files
files <-
  data.table("filePath" = grep("*\\.csv", list.files("data/"),
                               value = TRUE)) %>%
  .[, "rideName" :=  sub("(\\_old)?\\.csv", "", filePath)] %>%
  .[, rideName :=
      toupper(ifelse(rideName == "7_dwarfs_train", "dwarfs_train", rideName))]

round_time = function(x, precision, method = round) {
  if ("POSIXct" %in% class(x) == FALSE)
    stop("x must be POSIXct")

  tz = attributes(x)$tzone
  secs_rounded = method(as.numeric(x) / precision) * precision
  as.POSIXct(secs_rounded, tz = tz, origin = "1970-01-01")
}

longerData <- function(x) {
  rbindlist(
    list(x %>%
           .[, .(RIDENAME, date, datetime, SACTMIN)] %>%
           .[, `:=`(TYPE = "SACTMIN", WAITTIME = SACTMIN, SACTMIN = NULL)],
         x %>%
           .[, .(RIDENAME, date, datetime, SPOSTMIN)] %>%
           .[, `:=`(TYPE = "SPOSTMIN", WAITTIME = SPOSTMIN, SPOSTMIN = NULL)],
         x %>%
           .[, .(RIDENAME, date, datetime, CLOSED)] %>%
           .[, `:=`(TYPE = "CLOSED", WAITTIME = CLOSED, CLOSED = NULL)])
    )
}

dropDuplicated <- function(x) {
  x[!duplicated(x)]
}

roundTime <- 5
```

Load the Data

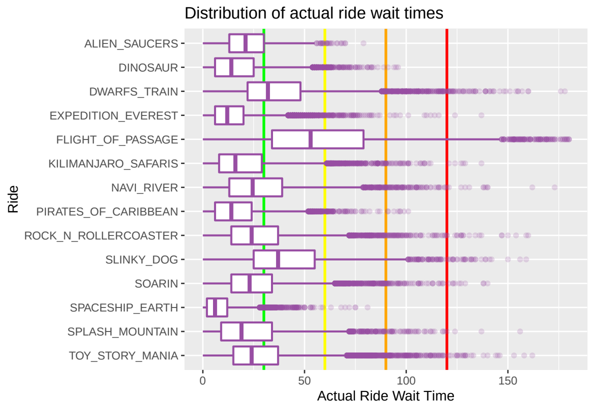
```r
dt <- unique(rbindlist(apply(files, 1, function(x) {
    fread(paste0("data/", x["filePath"])) %>%
    .[, CLOSED := ifelse(!is.na(SPOSTMIN) & SPOSTMIN == -999, 1, 0)] %>%
    .[, SPOSTMIN :=
        ifelse(!is.na(SPOSTMIN) & SPOSTMIN == -999, NA, SPOSTMIN)] %>%
    .[, RIDENAME := x["rideName"]]
  }), use.names = TRUE)) %>%
  longerData(.) %>%
  .[, `:=`(DATE = as.ordered(as.Date(date, format = "%m/%d/%Y")),
          date = NULL,
          DATETIME = round_time(datetime, 60*roundTime, floor),
          datetime = NULL)] %>%
  .[, `:=`(MONTH = month(DATETIME),
          DAY = mday(DATETIME),
          TIME = as.ITime(DATETIME),
          DATETIME = NULL)] %>%
  .[, WAITTIME := mean(WAITTIME, na.rm = TRUE),
    by = .(RIDENAME, TYPE, DATE, MONTH, DAY, TIME)] %>%
  .[TYPE == "CLOSED", WAITTIME :=
      ifelse(!is.nan(WAITTIME) & WAITTIME != 0, 1, 0)] %>%
  dropDuplicated(.) %>%
  dcast(., DATE + MONTH + DAY + TIME ~ RIDENAME + TYPE,
        value.var = "WAITTIME")

dt[dt == "NaN" | dt == "-Inf"] <- NA
```

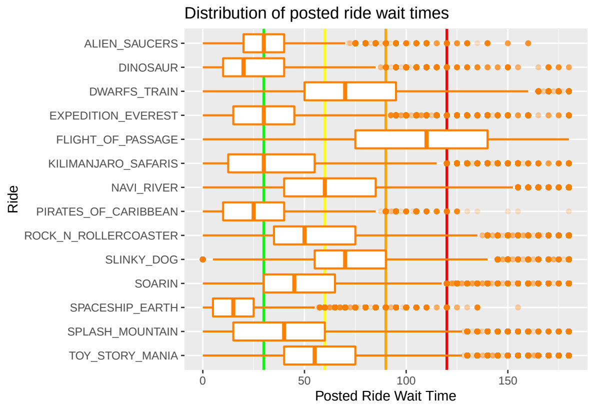
```r
RIDENAME <- toupper(c("dwarfs_train", "alien_saucers", "dinosaur",
               "expedition_everest", "flight_of_passage", "kilimanjaro_safaris",
               "navi_river", "pirates_of_caribbean", "rock_n_rollercoaster",
               "slinky_dog", "soarin", "spaceship_earth", "splash_mountain",
               "toy_story_mania"))
OPENDATE <- as.Date(c("2014/05/28", "2018/06/30", "1998/04/22", "2006/04/09",
                       "2017/05/27", "1998/04/22", "2017/05/27", "1973/12/17",
                       "1999/07/29", "2018/06/30", "2005/05/15", "1982/10/01",
                       "1992/07/17", "2008/05/31"))
SPLASH <- c(FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, TRUE,  FALSE, FALSE,
            FALSE, FALSE, TRUE,  FALSE)
INDOOR <- c(FALSE, FALSE, TRUE,  FALSE, TRUE, FALSE, TRUE,  TRUE,  TRUE,  FALSE,
            TRUE,  TRUE,  FALSE, TRUE)
AGEHIERARCHY <- c(10, 13,  4,  8, 11, 5, 12,  1,  6, 14, 7,  2,  3,  9)
DURATION <- c(3, 2.5, 3.5, 4, 6, 20, 5, 7.5, 1.5, 3, 8, 16, 18, 6.5)
WAITPERHUNDRED <- c(5, 10, 3, 4, 4, 4, 5, 1.5, 2.5, 5, 3, 3, 3.5, 4.5)
PARK <- toupper(c("mk", "hs", "ak", "ak", "ak", "ak", "ak", "mk", "hs", "hs",
                  "ep", "ep", "mk", "hs"))
dtMeta <- data.table(RIDENAME, OPENDATE, AGEHIERARCHY, SPLASH, INDOOR, PARK,
                     DURATION, WAITPERHUNDRED)
```

Summary Statistics (Wait Times)

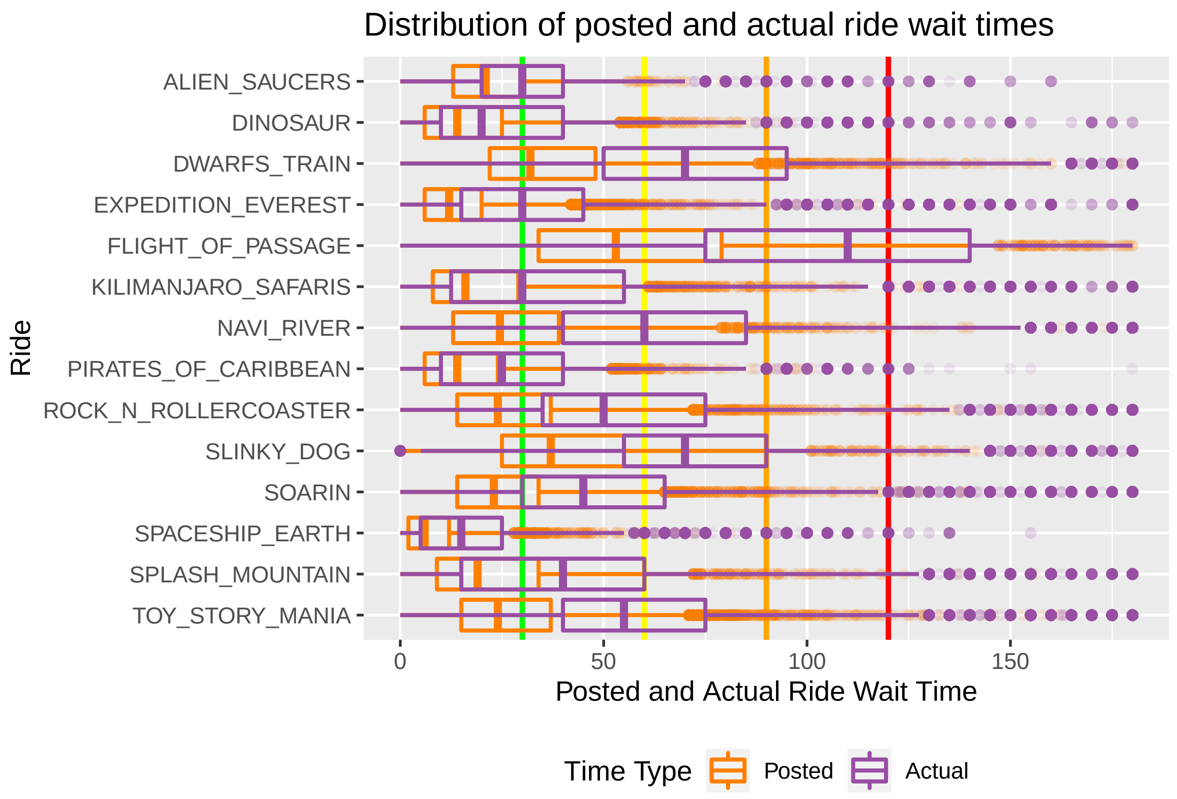
```r
dt %>%
  select(ends_with("_SACTMIN")) %>%
  `colnames<-`(., dtMeta$RIDENAME[order(dtMeta$RIDENAME)]) %>%
  filter(rowSums(!is.na(.)) != 0) %>%
  melt() %>%
  mutate(variable = factor(variable, levels = rev(unique(variable)))) %>%
  ggplot() +
  geom_vline(xintercept = 30, color = "green", size = 1) +
  geom_vline(xintercept = 60, color = "yellow", size = 1) +
  geom_vline(xintercept = 90, color = "orange", size = 1) +
  geom_vline(xintercept = 120, color = "red", size = 1) +
  xlim(c(0, 180)) +
  geom_boxplot(aes(y = variable, x = value), outlier.alpha = .2, size = .75,
               color = RColorBrewer::brewer.pal(9, "Set1")[4]) +
  labs(title = "Distribution of actual ride wait times") +
  xlab("Actual Ride Wait Time") +
  ylab("Ride") +
  theme(legend.position = "bottom")
```

```r
dt %>%
  select(ends_with("_SPOSTMIN")) %>%
  `colnames<-`(., dtMeta$RIDENAME[order(dtMeta$RIDENAME)]) %>%
  filter(rowSums(!is.na(.)) != 0) %>%
  melt() %>%
  mutate(variable = factor(variable, levels = rev(unique(variable)))) %>%
  ggplot() +
  geom_vline(xintercept = 30, color = "green", size = 1) +
  geom_vline(xintercept = 60, color = "yellow", size = 1) +
  geom_vline(xintercept = 90, color = "orange", size = 1) +
  geom_vline(xintercept = 120, color = "red", size = 1) +
  xlim(c(0, 180)) +
  geom_boxplot(aes(y = variable, x = value), outlier.alpha = .2, size = .75,
               color = RColorBrewer::brewer.pal(9, "Set1")[5]) +
  labs(title = "Distribution of posted ride wait times") +
  xlab("Posted Ride Wait Time") +
  ylab("Ride") +
  theme(legend.position = "bottom")
```





```
dt %>%
  select(ends_with(c("_SACTMIN", "_SPOSTMIN"))) %>%
  melt() %>%
  filter(!is.na(value)) %>%
  mutate(type = as.factor(ifelse(grepl("SACTMIN", variable),
                                 "Actual", "Posted"))) %>%
  mutate(variable = factor(str_remove(variable, "_SACTMIN|_SPOSTMIN"),
         levels = rev(sort(dtMeta$RIDENAME)))) %>%
  ggplot() +
  geom_vline(xintercept = 30, color = "green", size = 1) +
  geom_vline(xintercept = 60, color = "yellow", size = 1) +
  geom_vline(xintercept = 90, color = "orange", size = 1) +
  geom_vline(xintercept = 120, color = "red", size = 1) +
  xlim(c(0, 180)) +
  geom_boxplot(aes(y = variable, x = value, color = type), size = .75,
               position = "identity", outlier.alpha = .1, fill = "transparent") +
  scale_color_manual(values = RColorBrewer::brewer.pal(9, "Set1")[5:4],
                     name = "Time Type",
                     labels = c("Posted", "Actual")) +
  labs(title = "Distribution of posted and actual ride wait times") +
  xlab("Posted and Actual Ride Wait Time") +
  ylab("Ride") +
  theme(legend.position = "bottom")
```



Remove Outliers

```
removeOutliers <- function(x) {
  z <- abs(x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)
  x <- ifelse(is.na(x) | z < 3, x, NA)
  return(x)
}

cols <- grep(paste0(dtMeta$RIDENAME, collapse = "|"), colnames(dt),
             value = TRUE)
cols <- cols[!grepl("CLOSED", cols)]
dt <- dt[, (cols) := lapply(.SD, removeOutliers), .SDcols = cols]
```

T Test

```r
ttest <- dt %>%
  select(ends_with(c("_SACTMIN", "_SPOSTMIN"))) %>%
  melt() %>%
  filter(!is.na(value) & !is.nan(value)) %>%
  mutate(type = as.factor(ifelse(
    grepl("SACTMIN", variable), "Actual", "Posted")))

t.test(ttest[type == "Actual", .(value)], ttest[type == "Posted", .(value)])
```

```

    Welch Two Sample t-test

data:  ttest[type == "Actual", .(value)] and ttest[type == "Posted", .(value)]
t = -327.48, df = 135564, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -21.93920 -21.67815
sample estimates:
mean of x mean of y 
 24.61871  46.42739
```

Load and flesh out metadata

```
metadata <- fread("data/metadata/newMetadata.csv", na.strings = "") %>%
  .[, DATE := as.ordered(format(as.Date(DATE, format = "%m/%d/%y")))]
colnames(metadata) <- toupper(colnames(metadata))

tmp <-
  grep("OPEN|CLOSE|PRDDT[1-2]{1}|SHWNT[1-2]{1}|FIRET[1-2]{1}|PRDNT[1-2]{1}|SUNSET",
       colnames(metadata), value = TRUE)
metadata <- metadata[!duplicated(metadata$DATE)]
metadata[, (tmp) := lapply(.SD, as.ITime), .SDcols = tmp]
datetime <-
  data.table("DATE" = rep(metadata$DATE, each = 288),
             "TIME" = as.ITime(rep(seq(0*3600, 24*3600-1, by = 60*roundTime))))
shows <- grep("PRDDT[1-2]{1}|SHWNT[1-2]{1}|FIRET[1-2]{1}|PRDNT[1-2]{1}",
              colnames(metadata), value = TRUE)
showType <- c("PRDDT", "SHWNT", "FIRET", "PRDNT")

tmp <- lapply(toupper(unique(dtMeta$PARK)), function(x) {
  type <- grep(paste0(x, showType, collapse = "|"), shows, value = TRUE)
  type <- unique(str_extract(type, paste0(showType, collapse = "|")))
  lapply(type, function(y) {
    cols <- c("DATE", grep(paste0(x, y), shows, value = TRUE))
    y <- melt(metadata[, ..cols],
         measure.vars = cols[-1],
         variable.name = paste0(x, y),
         value.name = paste0(x, y, "TIME"))
    y <- merge(datetime, y,
          by.x = c("DATE", "TIME"),
          by.y = c("DATE", grep("TIME", names(y), value = TRUE)),
          all.x = TRUE)
    return(y[, !c("DATE", "TIME")])
  })
})
tmp <- rlist::list.cbind(unlist(tmp, recursive = FALSE))
cols <- unique(colnames(tmp))
tmp <- cbind(datetime, tmp[, ..cols])

cols <- !grepl(paste0(shows, collapse = "|"), colnames(metadata))
metadata <- merge(tmp, metadata[, ..cols], all.x = TRUE, by = "DATE") %>%
  .[TIME >= as.ITime("06:00:00") | TIME <= as.ITime("03:00:00")]

dt <- merge(dt, metadata, by = c("DATE", "TIME"), all.y = TRUE) %>%
  .[, `:=`(YEAR = NULL, MONTH = NULL, DAY = NULL)]
rm("tmp", "metadata")
```

Add in the weather data

```
nc <- tidync("data/w2012_2018.nc")
w2012_2018 <- nc %>%
  hyper_tibble() %>%
  data.table() %>%
  .[, `:=`(TIME = as.POSIXct(time * 60 * 60,
                             origin = "1900/01/01", tz = "EST"),
           temp = ((t2m - 273.15) * 9/5) + 32,
           rain = tp,
           longitude = NULL, latitude = NULL,
           t2m = NULL, tp = NULL, time = NULL)] %>%
  .[, `:=`(DATE = as.ordered(as.Date(TIME, format = "%m/%d/%y", tz = "EST")),
           hour = hour(as.ITime(TIME)),
           TIME = NULL)]

nc <- tidync("data/w2019_2021.nc")
w2019_2021 <- nc %>%
  hyper_tibble() %>%
  data.table() %>%
  .[, `:=`(TIME = as.POSIXct(time * 60 * 60,
                             origin = "1900/01/01", tz = "EST"),
           temp = ((t2m - 273.15) * 9/5) + 32,
           rain = tp,
           longitude = NULL, latitude = NULL,
           t2m = NULL, tp = NULL, time = NULL)] %>%
  .[, `:=`(DATE = as.ordered(as.Date(TIME, format = "%m/%d/%y", tz = "EST")),
           hour = hour(as.ITime(TIME)),
           TIME = NULL)]

w2019_2021 <- rbind(w2012_2018, w2019_2021)

nc <- merge(datetime[, `:=`(hour = hour(TIME))],
            w2019_2021, by = c("DATE", "hour"), all.x = TRUE) %>%
  .[, hour := NULL] %>%
  .[rowSums(is.na(.)) == 0]

dt <- merge(dt, nc, by = c("DATE", "TIME"), all.x = TRUE) %>%
  .[, DATE := as.ordered(format(as.Date(DATE), format = "%m-%d"))]

cols <- grep("CLOSED", colnames(dt), value = TRUE)
dt[, (cols) := lapply(.SD, function(x) {
  as.logical(case_when(
    is.na(x) ~ FALSE,
    x == 0 ~ FALSE,
    TRUE ~ TRUE
  ))
  }), .SDcols = cols]

rm("datetime", "files", "nc", "w2012_2018", "w2019_2021")
```

Lag the data

```r
# lag the actual data
cols <- paste0(dtMeta$RIDENAME, c("_SACTMIN"))
dt <- dt[, (cols) :=  shift(.SD, n = 1, type = "lead"),
         by = .(DATE), .SDcols = cols]
```

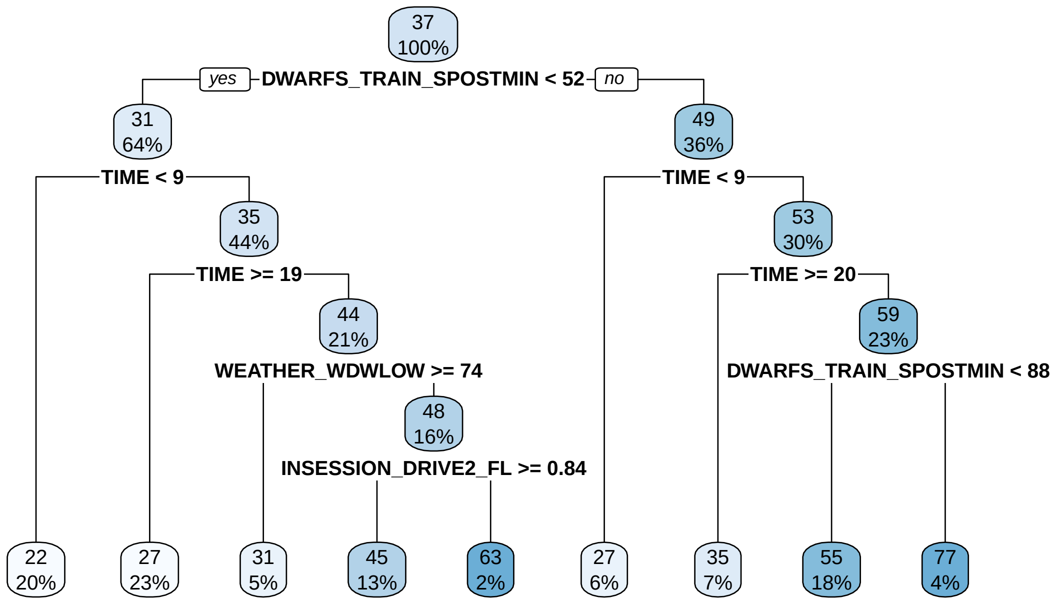
Split the data

```r
set.seed(2022)
rowPicker <- sample(c(TRUE, FALSE), nrow(dt), replace = TRUE, prob = c(.8, .2))

cols <- sapply(dt, function(x) { is.factor(x) | is.character(x) })
cols <- names(cols)[cols == TRUE & names(cols) != "DATE"]

dt2 <- dt
dt2 <- dt2[, (cols) := lapply(.SD, function(x = .SD, y = names(.SD)) {
  factor(x, levels = unique(x))
}), .SDcols = cols]

train <- dt2[rowPicker]
train[, TIME := as.numeric(as.numeric(TIME)  / 3600)]

test <- dt2[rowPicker]
test[, TIME := as.numeric(as.numeric(TIME)  / 3600)]

colnames(train) <- toupper(colnames(train))
colnames(test) <- toupper(colnames(test))
```

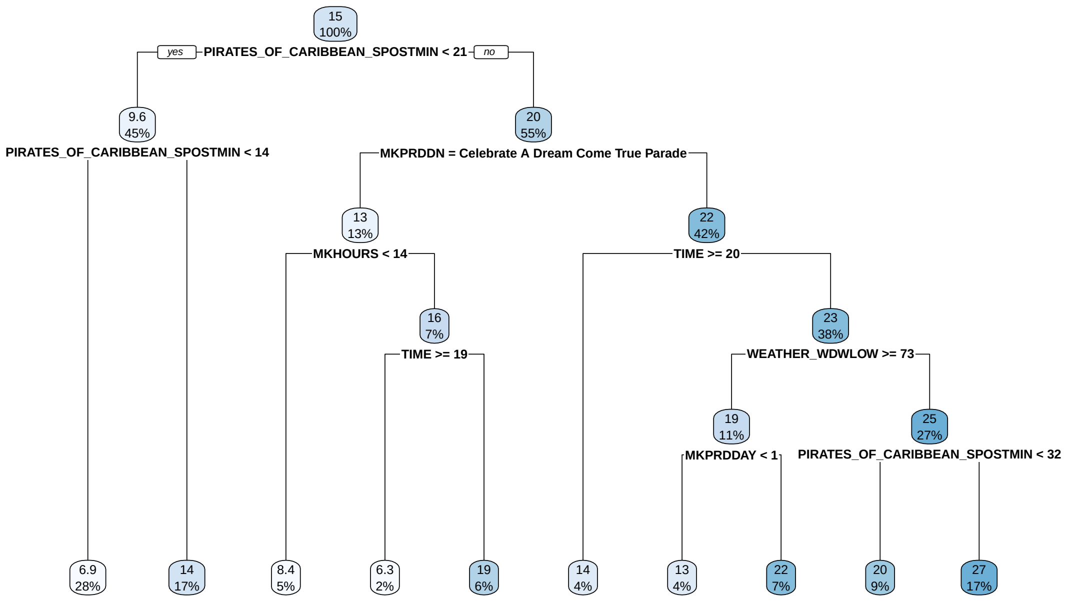
Create the trees

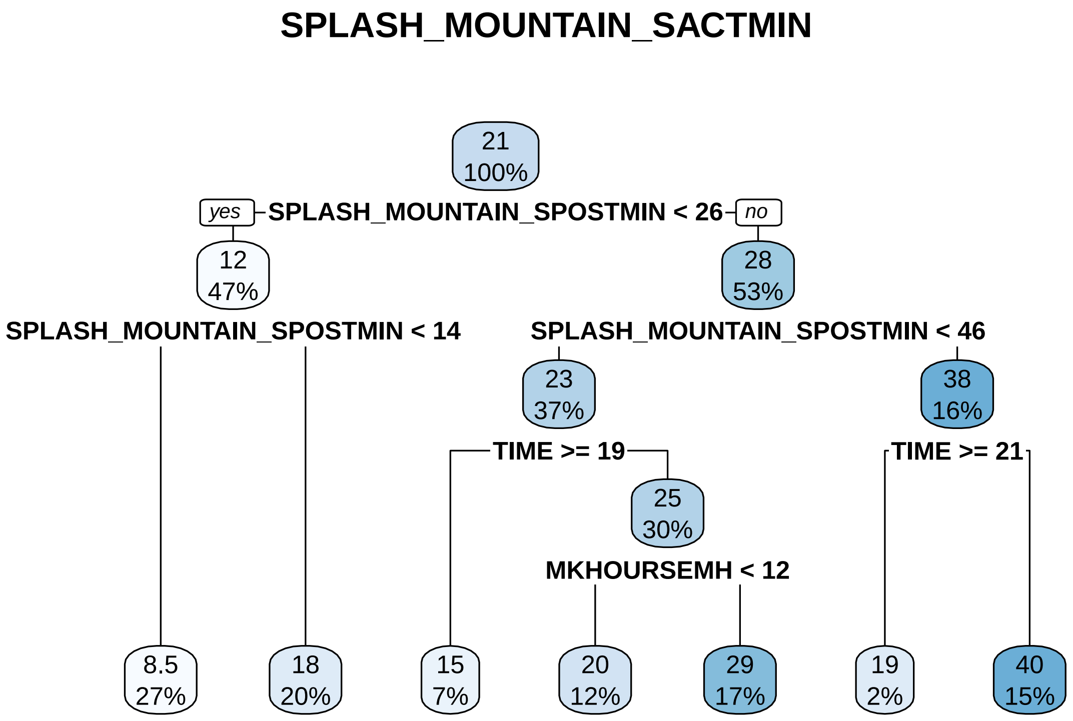
```r
cat("", file = "trees.txt")
tr <- rbindlist(lapply(unique(dtMeta$PARK), function(x) {
  # get columns for the park
  # drop the columns for rides not in the park
  rides <-
    colnames(train)[!grepl(paste0(unique(dtMeta$PARK[!dtMeta$PARK %in% x]),
                                  collapse = "|"), colnames(train))]
  rides <- rides[!grepl(paste0(dtMeta$RIDENAME[dtMeta$PARK != x],
                               collapse = "|"), rides)]
  # get the actual ride time variables
  actuals <- rides[grepl("(SACTMIN)", rides)]
  # return decision trees for the actual ride time variables
  return(rbindlist(lapply(actuals, function(y) {
    tr <- rpart(as.formula(paste(y, "~ .")), train[, ..rides])
    cat(paste0(y, "\n"), file = "trees.txt", append = TRUE)
    sink("trees.txt", append = TRUE); print(tr); sink()
    cat("\n\n", file = "trees.txt", append = TRUE)
    rpart.plot(tr, main = y)
    pred <- predict(tr, test)
    tmp <- data.table("actual" = test[[y]], "predicted" = pred)
    tmp <- tmp[!is.na(tmp$actual)][, `:=`(lower = actual * .85,
                                          upper = actual * 1.15)]
    tmp[, within :=
          ifelse(predicted >= lower & predicted <= upper, TRUE, FALSE)]
    tmpTbl <- table(tmp$within)
    tr <- data.table("ride" = y,
                     "mse" = mean((pred - test[[y]])^2, na.rm = TRUE),
                     "accuracy" = tmpTbl["TRUE"] / sum(tmpTbl),
                     "error" = mean(ifelse(test[[y]] - pred > 0, 1, 0),
                                    na.rm = TRUE),
                     "time" = mean(test[[y]], na.rm = TRUE),
                     "lower" = mean(tmp$lower),
                     "upper" = mean(tmp$upper))
    return(tr)
    })))
}))
```

DWARFS_TRAIN_SACTMIN

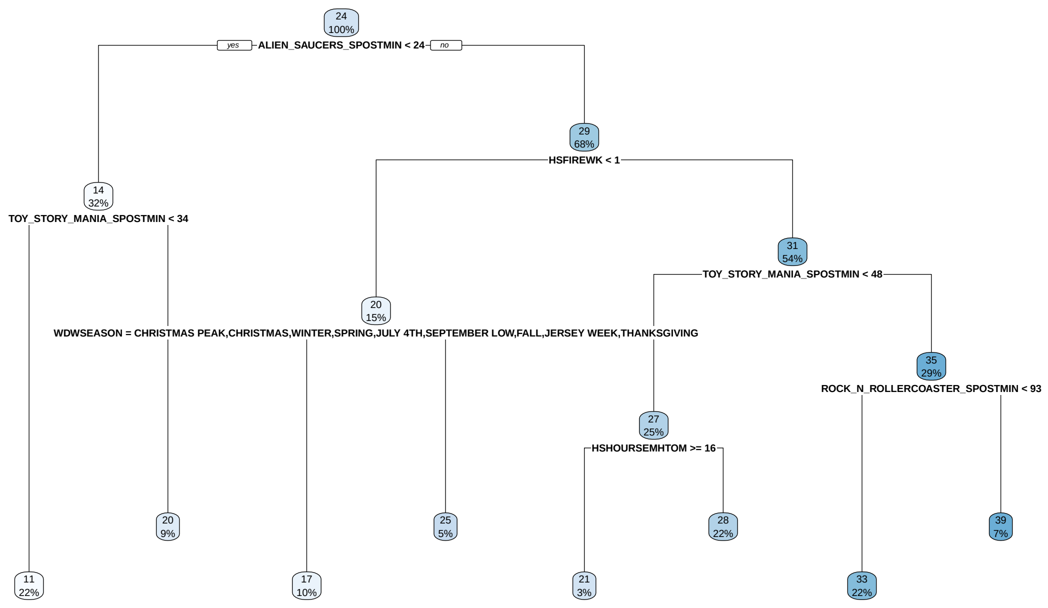


PIRATES_OF_CARIBBEAN_SACTMIN

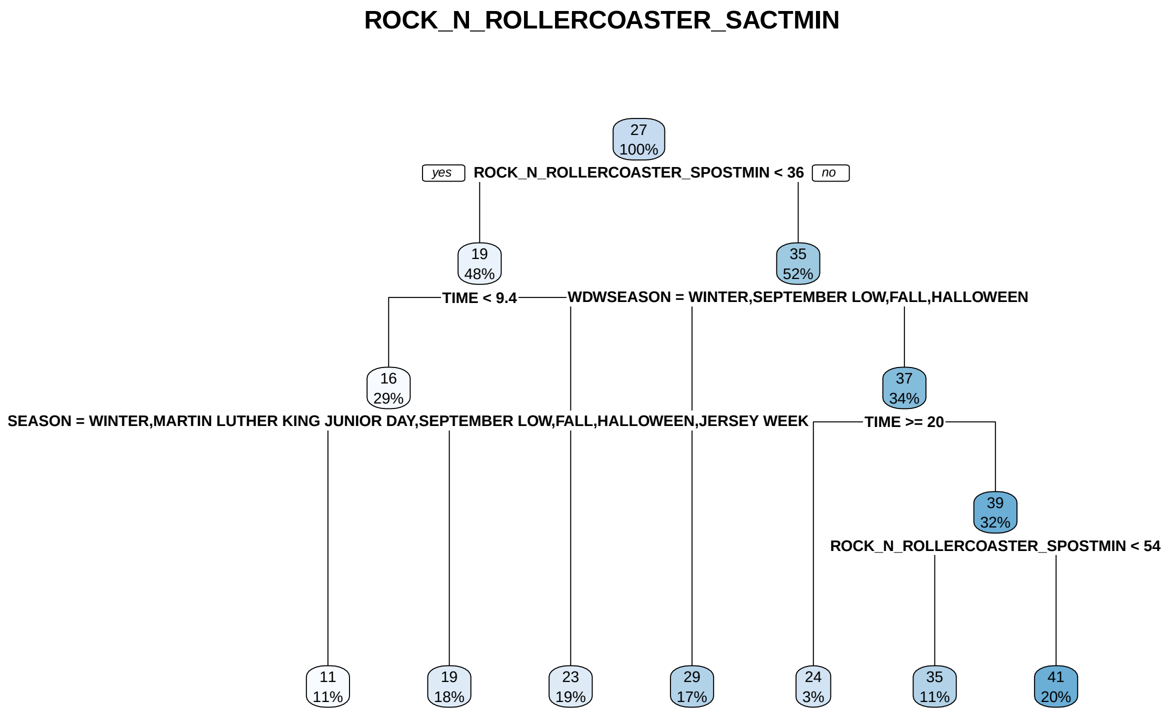



SPLASH_MOUNTAIN_SACTMIN
21
100%
yes
SPLASH_MOUNTAIN_SPOSTMIN < 26
no
12
47%
28
53%
SPLASH_MOUNTAIN_SPOSTMIN < 14
SPLASH_MOUNTAIN_SPOSTMIN < 46
23
37%
38
16%
TIME >= 19
TIME >= 21
25
30%
MKHOURSEMH < 12
8.5
27%
18
20%
15
7%
20
12%
29
17%
19
2%
40
15%

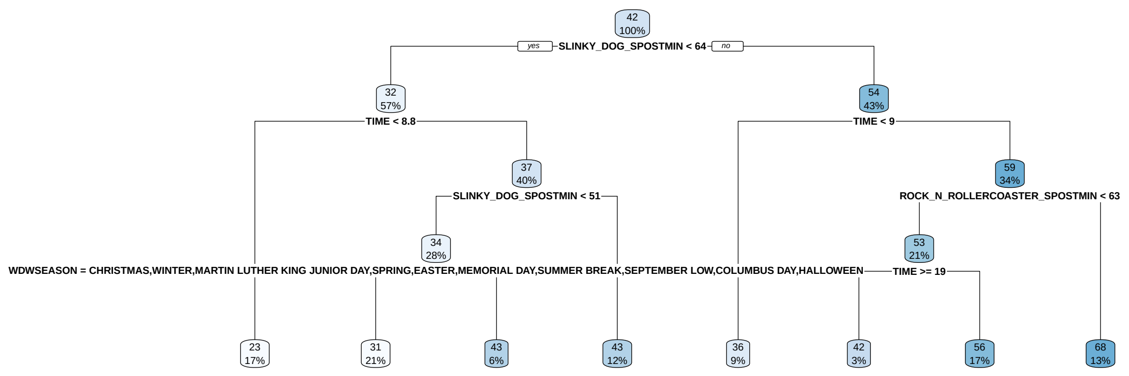
ALIEN_SAUCERS_SACTMIN



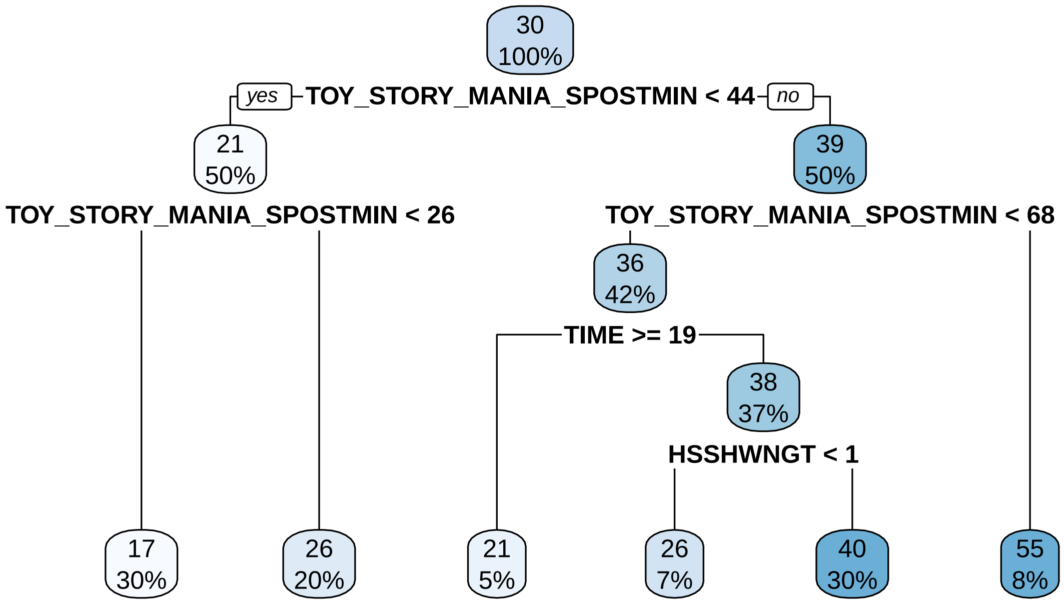
24
100%
yes
ALIEN_SAUCERS_SPOSTMIN < 24
no
29
68%
HSFIREWK < 1
14
32%
TOY_STORY_MANIA_SPOSTMIN < 34
31
54%
TOY_STORY_MANIA_SPOSTMIN < 48
20
15%
WDWSEASON = CHRISTMAS PEAK,CHRISTMAS,WINTER,SPRING,JULY 4TH,SEPTEMBER LOW,FALL,JERSEY WEEK,THANKSGIVING
35
29%
ROCK_N_ROLLERCOASTER_SPOSTMIN < 93
27
25%
HSHOURSEMHTOM >= 16
20
9%
25
5%
28
22%
39
7%
11
22%
17
10%
21
3%
33
22%

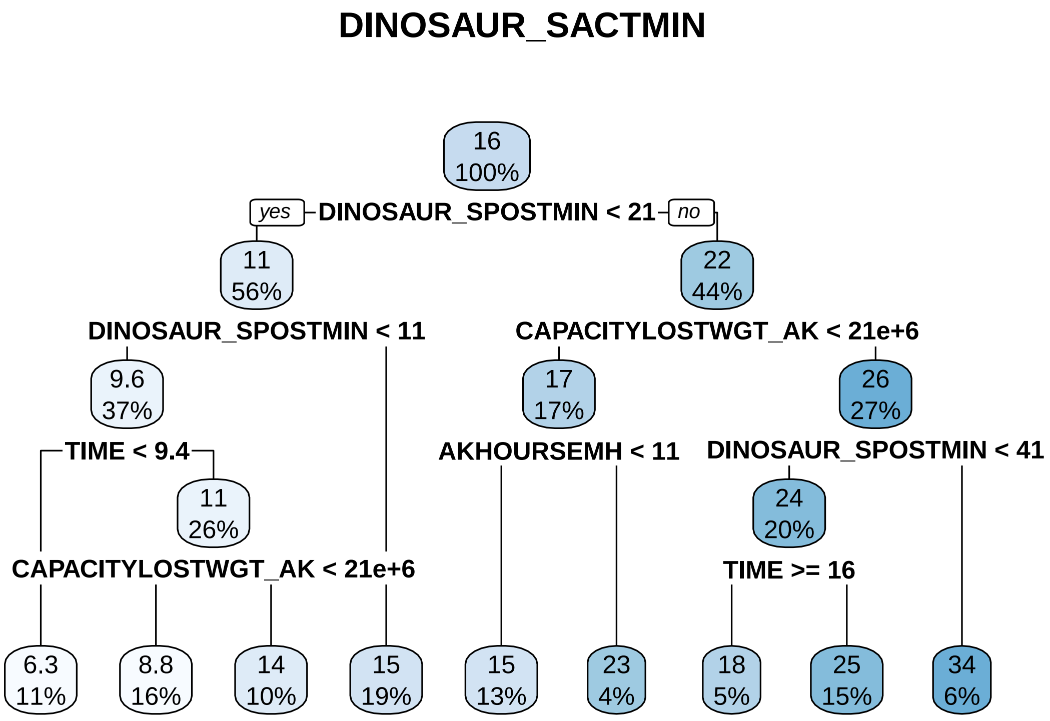

ROCK_N_ROLLERCOASTER_SACTMIN
27
100%
yes
ROCK_N_ROLLERCOASTER_SPOSTMIN < 36
no
19
48%
35
52%
TIME < 9.4
WDWSEASON = WINTER,SEPTEMBER LOW,FALL,HALLOWEEN
16
29%
37
34%
SEASON = WINTER,MARTIN LUTHER KING JUNIOR DAY,SEPTEMBER LOW,FALL,HALLOWEEN,JERSEY WEEK
TIME >= 20
39
32%
ROCK_N_ROLLERCOASTER_SPOSTMIN < 54
11
11%
19
18%
23
19%
29
17%
24
3%
35
11%
41
20%

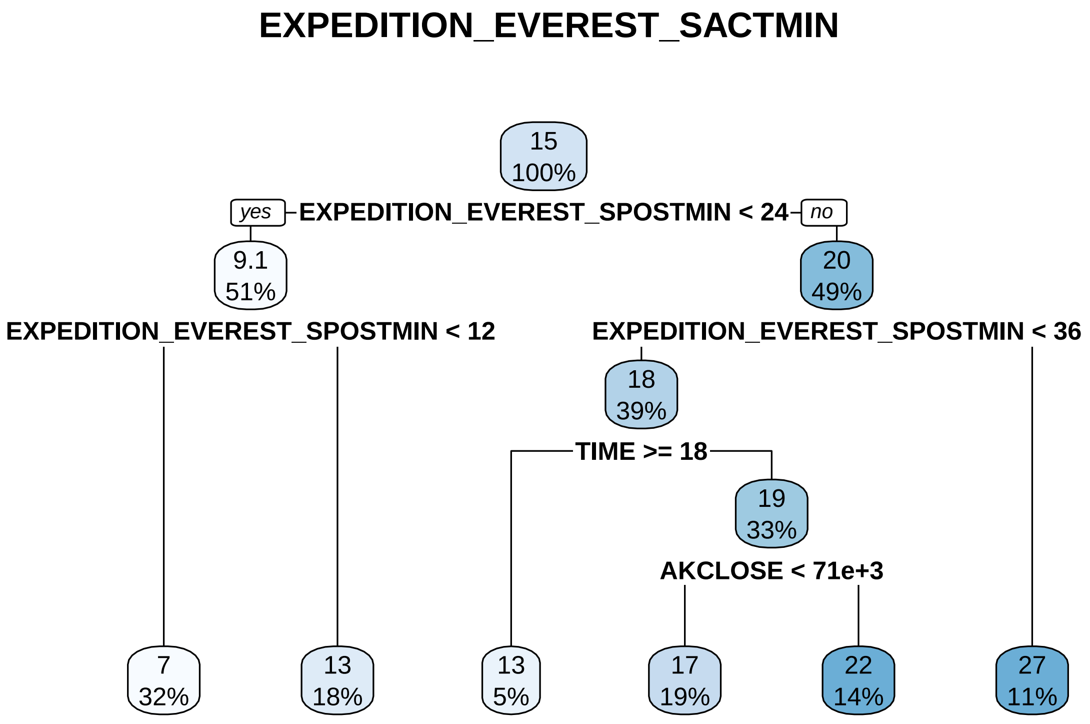
SLINKY_DOG_SACTMIN


42
100%
yes
SLINKY_DOG_SPOSTMIN < 64
no
32
57%
TIME < 8.8
54
43%
TIME < 9
37
40%
SLINKY_DOG_SPOSTMIN < 51
59
34%
ROCK_N_ROLLERCOASTER_SPOSTMIN < 63
34
28%
WDWSEASON = CHRISTMAS,WINTER,MARTIN LUTHER KING JUNIOR DAY,SPRING,EASTER,MEMORIAL DAY,SUMMER BREAK,SEPTEMBER LOW,COLUMBUS DAY,HALLOWEEN
53
21%
TIME >= 19
23
17%
31
21%
43
6%
43
12%
36
9%
42
3%
56
17%
68
13%

TOY_STORY_MANIA_SACTMIN

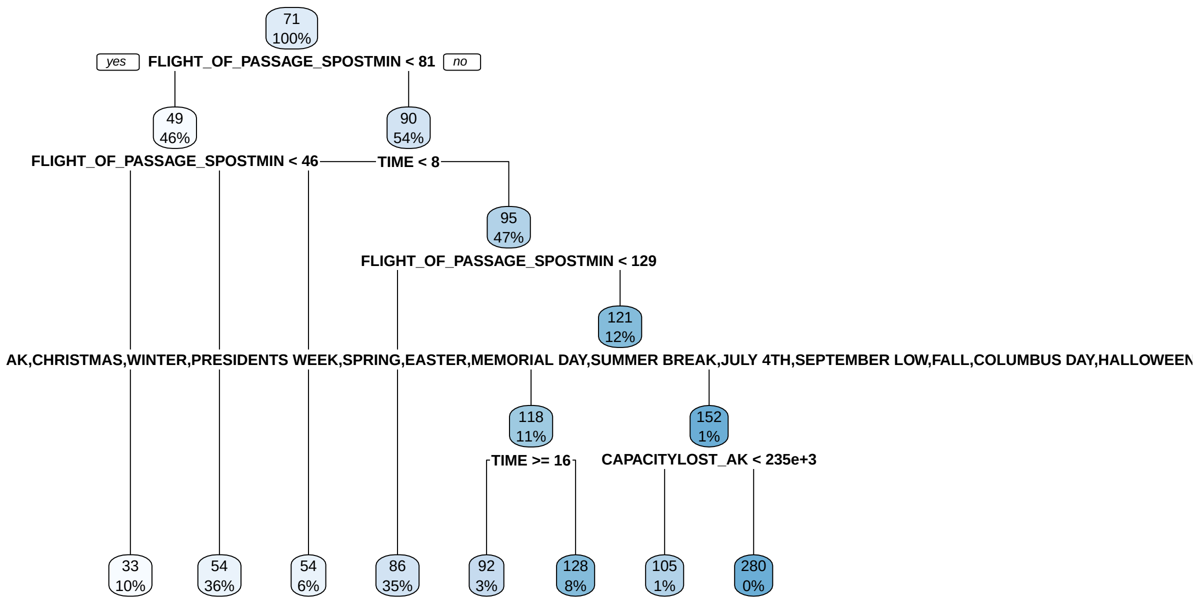

30
100%
yes
TOY_STORY_MANIA_SPOSTMIN < 44
no
21
50%
39
50%
TOY_STORY_MANIA_SPOSTMIN < 26
TOY_STORY_MANIA_SPOSTMIN < 68
36
42%
TIME >= 19
38
37%
HSSHWNGT < 1
17
30%
26
20%
21
5%
26
7%
40
30%
55
8%


DINOSAUR_SACTMIN
16
100%
yes
DINOSAUR_SPOSTMIN < 21
no
11
56%
22
44%
DINOSAUR_SPOSTMIN < 11
CAPACITYLOSTWGT_AK < 21e+6
9.6
37%
17
17%
26
27%
TIME < 9.4
AKHOURSEMH < 11
DINOSAUR_SPOSTMIN < 41
11
26%
24
20%
CAPACITYLOSTWGT_AK < 21e+6
TIME >= 16
6.3
11%
8.8
16%
14
10%
15
19%
15
13%
23
4%
18
5%
25
15%
34
6%

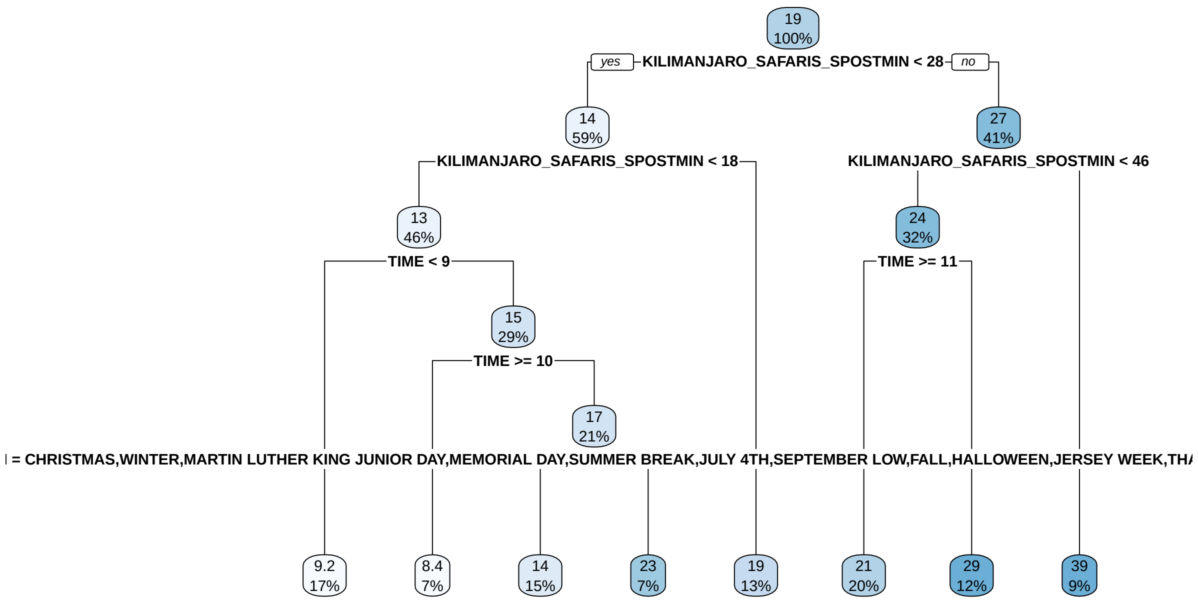


EXPEDITION_EVEREST_SACTMIN
15
100%
yes
EXPEDITION_EVEREST_SPOSTMIN < 24
no
9.1
51%
20
49%
EXPEDITION_EVEREST_SPOSTMIN < 12
EXPEDITION_EVEREST_SPOSTMIN < 36
18
39%
TIME >= 18
19
33%
AKCLOSE < 71e+3
7
32%
13
18%
13
5%
17
19%
22
14%
27
11%

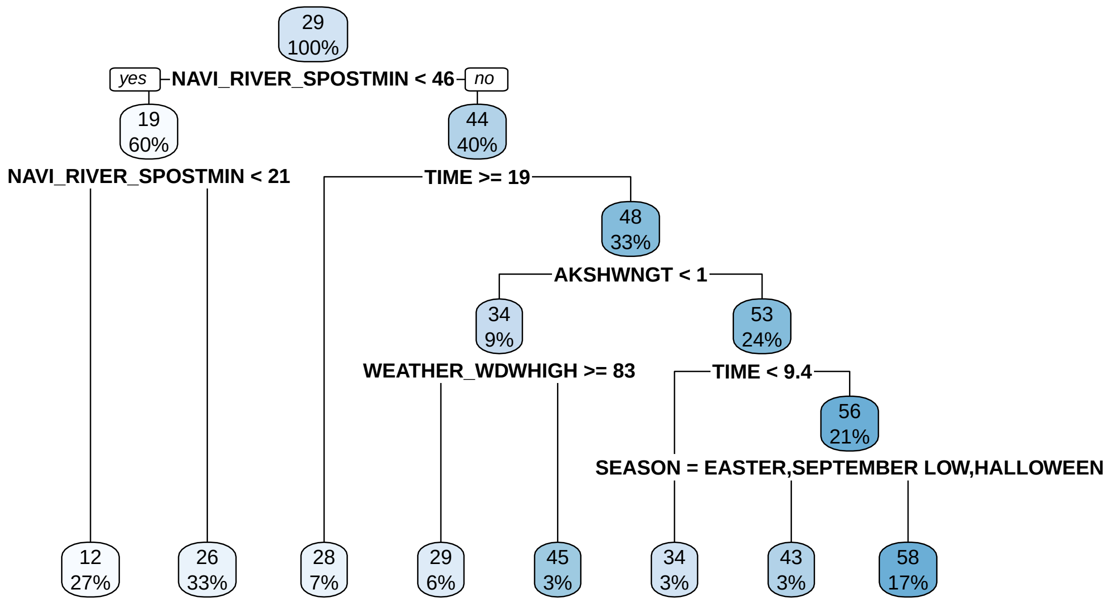
FLIGHT_OF_PASSAGE_SACTMIN

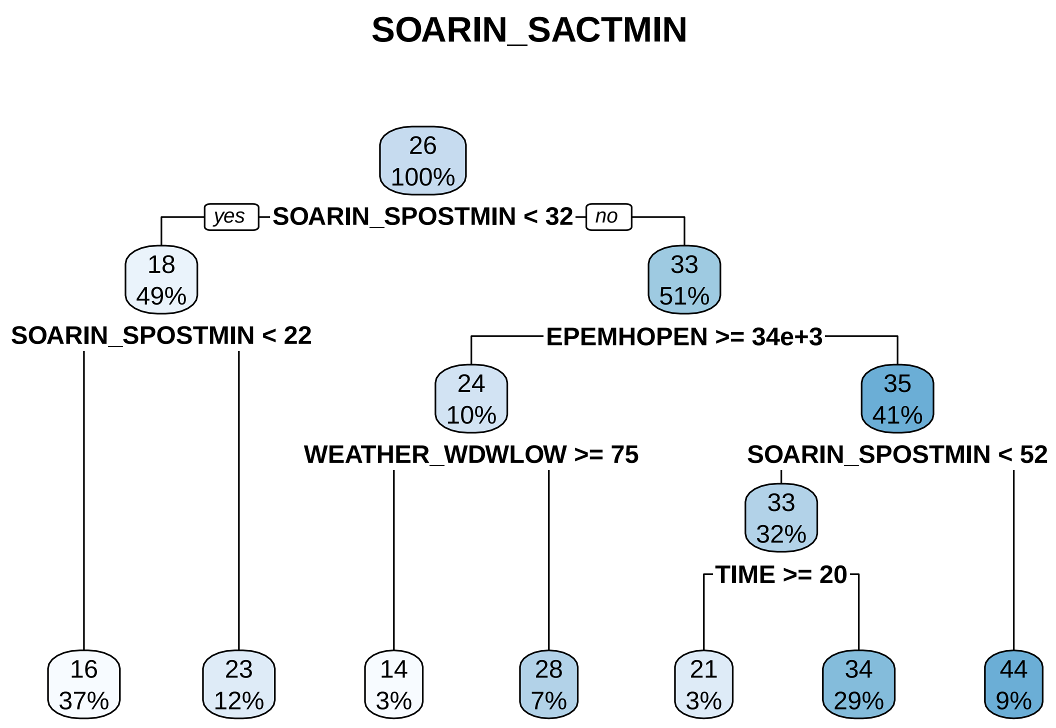

71
100%
yes
FLIGHT_OF_PASSAGE_SPOSTMIN < 81
no
49
46%
90
54%
FLIGHT_OF_PASSAGE_SPOSTMIN < 46
TIME < 8
95
47%
FLIGHT_OF_PASSAGE_SPOSTMIN < 129
121
12%
AK,CHRISTMAS,WINTER,PRESIDENTS WEEK,SPRING,EASTER,MEMORIAL DAY,SUMMER BREAK,JULY 4TH,SEPTEMBER LOW,FALL,COLUMBUS DAY,HALLOWEEN
118
11%
152
1%
TIME >= 16
CAPACITYLOST_AK < 235e+3
33
10%
54
36%
54
6%
86
35%
92
3%
128
8%
105
1%
280
0%

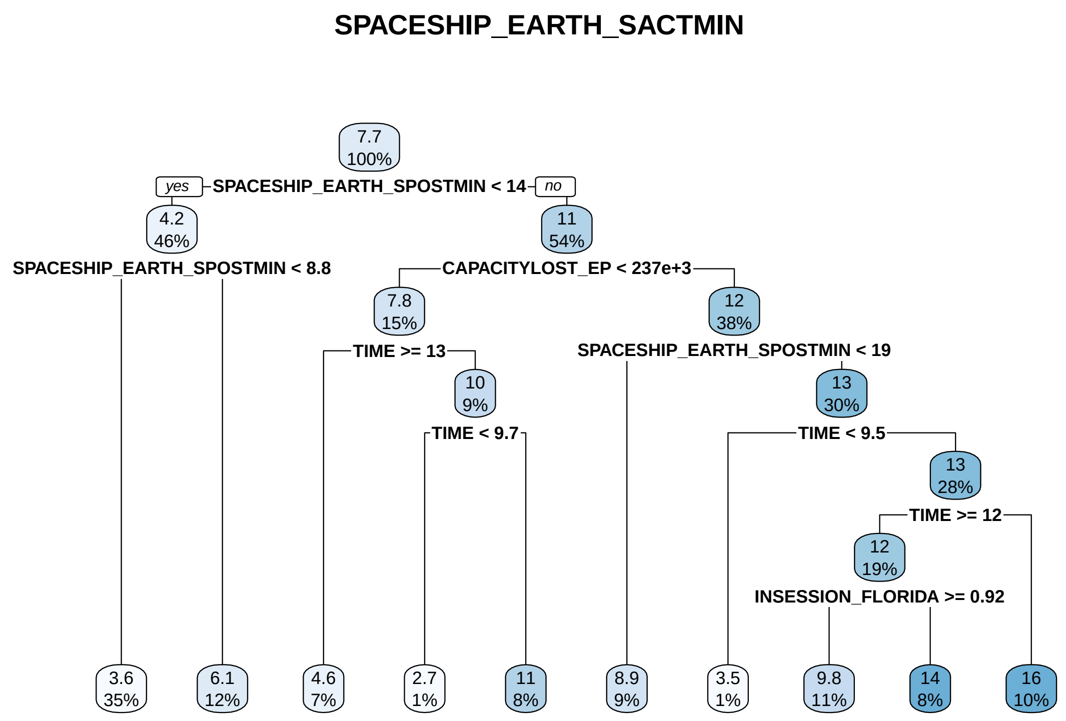
KILIMANJARO_SAFARIS_SACTMIN



NAVI_RIVER_SACTMIN




SOARIN_SACTMIN
26
100%
yes
SOARIN_SPOSTMIN < 32
no
18
49%
33
51%
SOARIN_SPOSTMIN < 22
EPEMHOPEN >= 34e+3
24
10%
35
41%
WEATHER_WDWLOW >= 75
SOARIN_SPOSTMIN < 52
33
32%
TIME >= 20
16
37%
23
12%
14
3%
28
7%
21
3%
34
29%
44
9%



```r
tr[order(-accuracy),]
```

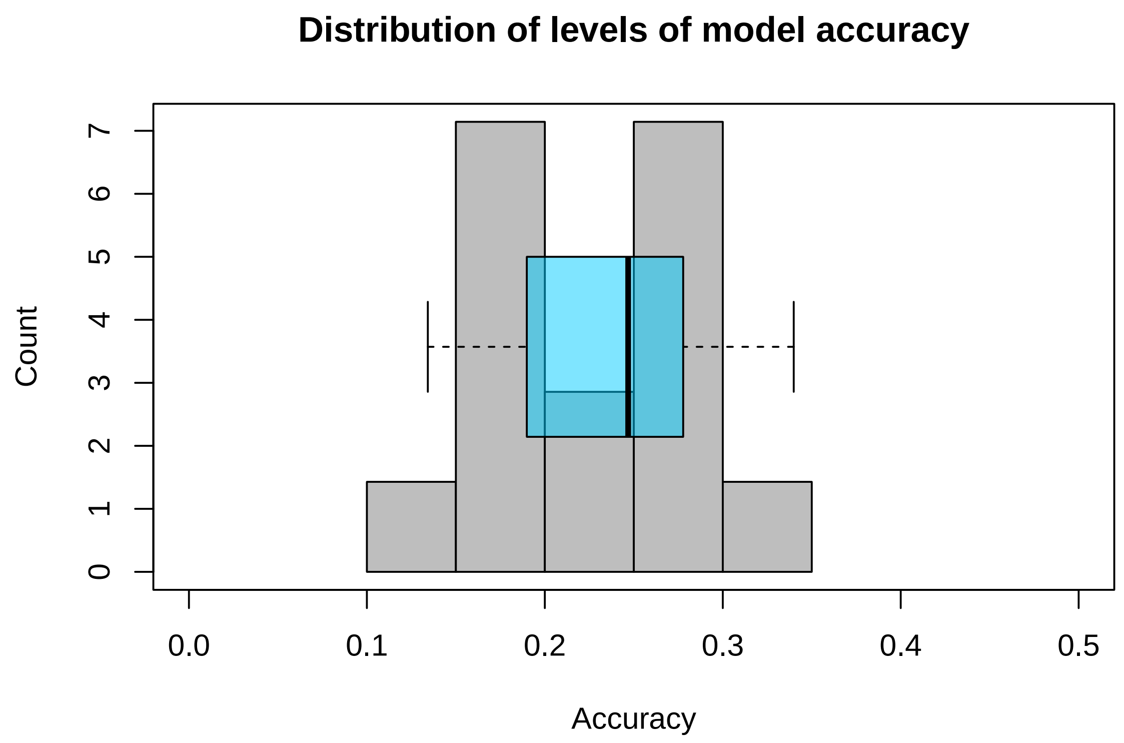
```
                             ride        mse  accuracy     error      time
 1:          ALIEN_SAUCERS_SACTMIN   75.03634 0.3398693 0.4545455 24.132006
 2:             SLINKY_DOG_SACTMIN  291.05262 0.2967509 0.4498195 41.771841
 3:      FLIGHT_OF_PASSAGE_SACTMIN 1391.42732 0.2966457 0.4517820 70.958100
 4:            DWARFS_TRAIN_SACTMIN  356.04344 0.2777460 0.4500286 37.380116
 5:                 SOARIN_SACTMIN  136.72691 0.2622899 0.4396105 25.581383
 6:             NAVI_RIVER_SACTMIN  207.13406 0.2518397 0.4468520 29.373876
 7:         TOY_STORY_MANIA_SACTMIN  354.42701 0.2473868 0.4232109 29.769946
 8: ROCK_N_ROLLERCOASTER_SACTMIN  200.87767 0.2462754 0.4492910 26.910324
 9:         SPLASH_MOUNTAIN_SACTMIN  153.13020 0.1950009 0.4229002 20.585891
10: PIRATES_OF_CARIBBEAN_SACTMIN   77.53222 0.1922266 0.4392703 15.275323
11:     EXPEDITION_EVEREST_SACTMIN   76.91368 0.1898715 0.4194596 14.547443
12:               DINOSAUR_SACTMIN  107.23556 0.1821600 0.4065177 16.285025
13:   KILIMANJARO_SAFARIS_SACTMIN  167.77709 0.1606648 0.4159082 19.481137
14:         SPACESHIP_EARTH_SACTMIN   34.37234 0.1342466 0.3907271  7.663962
```

```
        lower     upper
 1: 20.512205 27.751807
 2: 35.506065 48.037617
 3: 60.314385 81.601815
 4: 31.773099 42.987134
 5: 21.744176 29.418591
 6: 24.967794 33.779957
 7: 25.304454 34.235437
 8: 22.873776 30.946873
 9: 17.498007 23.673775
10: 12.984025 17.566622
11: 12.365327 16.729560
12: 13.842272 18.727779
13: 16.558966 22.403308
14:  6.514368  8.813556
```

```
fwrite(tr, "accuracy.csv")
mean(tr$accuracy)
```

```
[1] 0.2337839
```

```
# Histogram
hist(tr$accuracy, prob = TRUE,
     col = "grey",
     main = "", xlab = "", ylab = "",
     xlim = c(0, .5))
par(new = TRUE)
boxplot(tr$accuracy, horizontal = TRUE, axes = FALSE,
        ylim = c(0, .5),
        col = rgb(0, 0.8, 1, alpha = 0.5),
        xlab = "Accuracy", ylab = "Count",
        main = "Distribution of levels of model accuracy")
box()
```


Distribution of levels of model accuracy
Count
0
1
2
3
4
5
6
7
0.0
0.1
0.2
0.3
0.4
0.5
Accuracy