What it Takes to Finish Western States

Tomas Castillo

January 29, 2017

Overview of Project and Western States

I wanted to focus on a project in an area that I have a passion for, along with one that could lead to further applications. I decided to go for data surrounding ultramarathons, specifically 100 mile races.

There are a few different sites that I could pull aggregate data from, but I wasn't able to properly scrape that kind of data. Instead, I was able to take data from what is the gold standard for 100-mile races - the Western States Endurance Run (WSER).

The race has some interesting lore behind it, but it is consistently one of the most competitive races year after year (it was ranked #1 for 2016, and again in 2015 by ultrarunning.com), and was also the very first 100-mile ultramathon. It is most often a gold standard to compare other 100-mile races to. Therefore, I figured it would be a good race to evaluate for my project.

Ultramarathon 101

While most may be familiar with a traditional marathon of 26.2 miles, an ultramarathon is any foot race over 26.2 miles. Most take place on the trails of state and federal lands (sometimes looped, other times point-to-point), and often involve lots of vertical climbing and descending to go with the horizontal distance (known as "vert" for vertical gain).

Additionally, for a 100-mile race, the body does not contain enough readily available resources for the mind and muscles to last the duration of these events, which can take anywhere from 14 hours for the superhumans, to 24 and 36 hours and beyond; therefore, aid stations are setup along the course to fuel and hydrate runners, and serve as a chance for rest if needed.

Lastly, while the point of a race is to go as fast as possible, 100 miles is a long distance and requires smart planning and execution, otherwise finishing may not even be an option. For some runners that may mean to run up the hills and spend little time at aid stations, while for others it is wiser to power-hike up a hill and conserve energy for flatter and downhill sections.

Origin and Wrangling of Data

WSER not only includes the final times and overall places for each race, but also has a detailed results page that includes all splits into the aid stations for each year.

The format of the file was not always consistent and required a fair amount of wrangling.

Once I pulled each year, I was able to normalize each aid station based on its name, distance from the start, and whether the measurement was for a runner coming into an aid station, or leaving. Below is an example of what the final data looked in excel before I exported it to a `*.csv` and imported it to R.

Formatting in R

Once in R, it was necessary to wrangle the data into a format that would be most conducive to analysis. Information pertaining to the runner's overall stats (name, age, gender, overall time, place, etc.) were formatted in a style that was easy to work with. Information detailing the aid stations was not always in a

Year	Google Doc	Excel	Text File	PDF
2016		x		
2015		x		
2014		x		
2013		x		
2012	x	x		
2011	x	x		
2010	x	x		
2009	x	x		
2007	x	x	x	
2006	x	x	x	
2005	x	x	x	
2004	x	x	x	
2003			x	x
2002			x	x
2001			x	
2000			x	
1999			x	
1998			x	
1997			x	
1996			x	
1995			x	
1994			x	
1993			x	
1992			x	
1991			x	
1990			x	
1989			x	
1988			x	
1987			x	
1986			x	
1983				x
1982				x
1981				x

Figure 1: WSER Splits Page

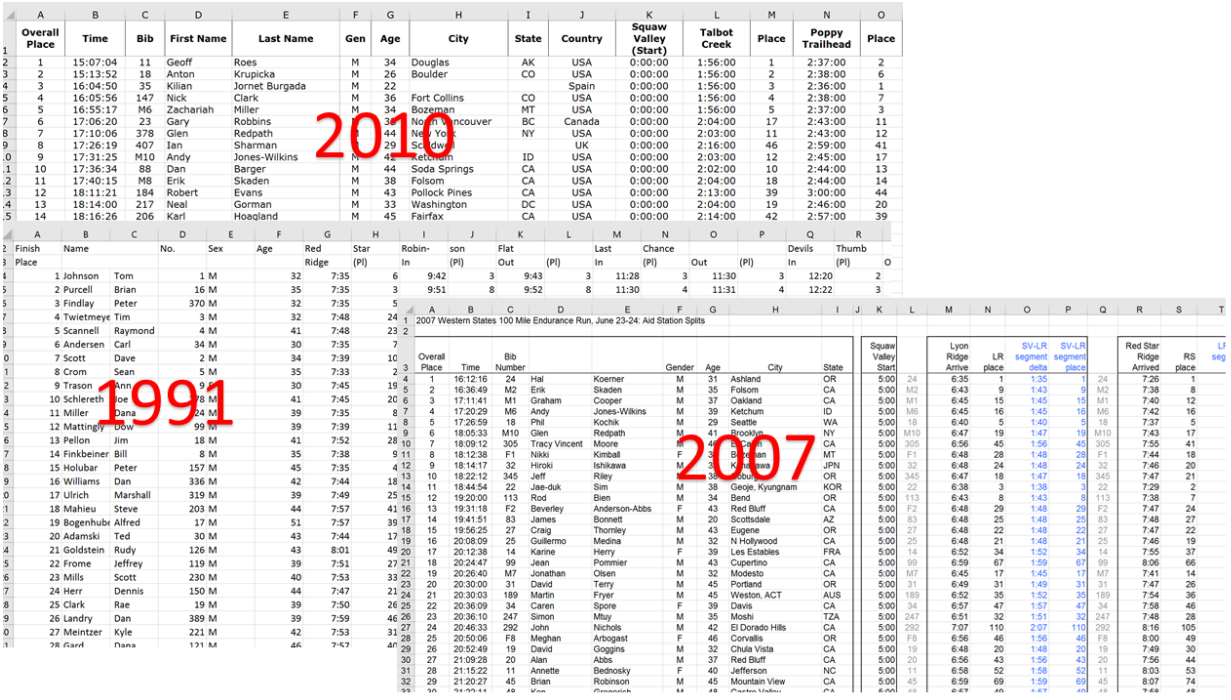


Figure 2: Example of Splits Data

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
27			Last Na	First Name	Bib	Sex	Age	City	State	Talbot Creek (snow)		Poppy (snow)		Lyon Ridge		Red Star		Duncan Canyon	
28	Year	Place	Last Na	First Name	Bib	Sex	Age	City	State	t15.2_in	p15.2_in	t20.3_in	p20.3_in	t10_in	p10_in	t16_in	p16_in	t23.8_in	p23.8_in
29	1986	1	Jones	Chuck	248	M	27									7:39	9		
30	1986	2	Pellon	Jim	2	M	36									7:37	7		
31	1986	3	King	Jim	1	M	29									7:28	2		
32	1986	4	Tanzer	Herb	6	M	34									7:37	8		
33	1986	5	Loesch	John	294	M	42									7:40	11		
34	1986	6	William	Dan	466	M	37									7:40	12		
35	1986	7	Twietn	Timothy	10	M	27									7:54	48		
36	1986	8	Latime	Doug	8	M	48									7:49	33		
37	1986	9	Moore	Craig	339	M	33									7:45	21		
38	1986	10	Steven	Dave	13	M	33									7:56	51		
39	1986	11	Pomro	Jim	375	M	39									7:49	29		
40	1986	12	Robert	David	333	M	33									7:41	13		
41	1986	13	Jones	Martin	249	M	43									7:55	49		
42	1986	14	Volken	Rob	452	M	55									8:06	82		
43	1986	15	Roth	David		M	32									7:44	20		
44	1986	16	D'Onof	Kathy	122	F	21									7:49	31		
45	1986	17	Daniels	Roger	148	M	50									7:40	10		
46	1986	18	Savage	Charles W	14	M	38									7:59	58		
47	1986	19	Moody	Kimberly	338	F	31									7:51	43		
48	1986	20	Bogenh	Alfred	94	M	46									8:03	66		
49	1986	21	Turney	Chris	446	M	28									7:42	16		
50	1986	22	Johnso	Vicki	16	F	43									8:05	77		
51	1986	23	Leighto	Gard	283	M	52									7:51	42		
52	1986	24	Austrh	Bjorg	20	F	43									8:02	65		
53	1986	25	Kissel	Bill	267	M	33									7:47	26		
54	1986	26	Park	James Dav	357	M	30									8:03	67		
55	1986	27	Gray	David	195	M	32									8:08	88		
56	1986	28	La Belle	Bruce	21	M	30									7:33	6		
57	1986	29	Rotkis	Tom	17	M	38									7:59	59		
58	1986	30	Gard	Dana	41	M	41									7:49	35		
59	1986	31	Larsen	Dane	276	M	35									7:51	41		
60	1986	32	Hamm	Mary	208	F	26									8:12	97		

Figure 3: Final XLSX Output

consistent layout, and required a method to normalize the data. Below is an example of my methodology for how I categorized the aid stations.

AS Name	Variable Name	Distance (miles)	In or Out	Type of Measurement
Red Star Ridge	`t16_in`	16	In	Time
Red Star Ridge	`p16_in`	16	In	Place
Duncan Canyon	`t23.8_in`	23.8	In	Time
Duncan Canyon	`p23.8_in`	23.8	In	Place
Miller's Defeat	`t34.4_in`	34.4	In	Time
Miller's Defeat	`p34.4_in`	34.4	In	Place
Miller's Defeat	`t34.4_out`	34.4	Out	Time
Miller's Defeat	`p34.4_out`	34.4	Out	Place

Once the dataset was in a somewhat manageable wide format (one runner per line), there were still lots of steps to go through. Different challenges included accounting for the way the time was recorded at aid stations on different years (i.e. time of day vs. elapsed time since the start of the race), removing missing data, writing some fun `REGEX`, and doing a bit of feature engineering. I tried to comment on the code to hopefully make it clear what my steps were.

Ultimately, I was able to get data back into a wide format where each line represents a single racer per year with all relevant information for the racer for that particular race year. The final data frame only contains the time a racer came into an aid station due to the fact that the exit times weren't as consistent as entry times.

Code for Wrangling

Code for the wrangling process is contained in the appendix and in the R-Markdown file used to create this document, which can be viewed on my github page.

Final Data Set

Below is an outline of the finalized data frame that is used for analysis. I am excluding the list of all the aid stations, but will include that in the appendix, along with an data table showing the long data frame structure.

Variable Name	type	Explanation
`key`	chr	Unique ID of year and overall place (e.g. 1995-3)
`YearInt`	num	Year of race (1986-2016)
`Year`	factor	Year of race as a factor
`Place`	int	Overall place
`Last.Name`	chr	Last name of runner
`First.Name`	chr	First name of runner
`Bib`	chr	Bib number of runner (mostly numbers, some alpha numeric)
`Sex`	chr	Male or Female (M or F)
`Age`	int	Age of runner for given year
`ageGroup`	num	Index number for 7 bins - (0,20,30,40,50,60,70,200)
`timeElapsed_MIN`	num	Overall time for race (NA if DNF)
`DNF`	num	Finish status - "Finished" = 0, "DNF" = 1
`qRank.overall`	int	Decile rank based on overall time vs all years
`pace.overall`	num	Overall pace for duration of race (assumes distance of 100 miles)
`qRank.year`	int	Decile rank based on overall time vs given year
** AID STATIONS **	num	Time elapsed (in minutes) to reach given aid station
`pace.avg`	num	Average pace between each aid station
`pace.sd`	num	Standard deviation of the paces between each aid station
`as.max`	num	Maximum number of aid stations given runner checked in to
`bibNo`	num	Reformatted `Bib` removing the ALPHA characters
`sex.male`	num	Reformatted `Sex` - "Female" = 0, "Male" = 1

Exploratory Data Analysis

It is always important to evaluate the data to ensure, first and foremost, none of the data are abnormal outliers. After that, I wanted to explore the different variables and what they showed, and how they are related. I have a separate post on the exploration of the data, but have a included it here as well. In an effort to conserve space, I have supressed the code output so all that is shown are the graphs.

Timing into Aid Stations

Below are a few of the different graphs that explore what sorts of times runners were hitting going into the second aid station for different years.

Something to note on figure 04 is how a few years seem to have abnormally fast 2nd aid stations. This is due to snow on the course and the race altering the traditional path, and the 2nd aid station being a bit closer to the start.

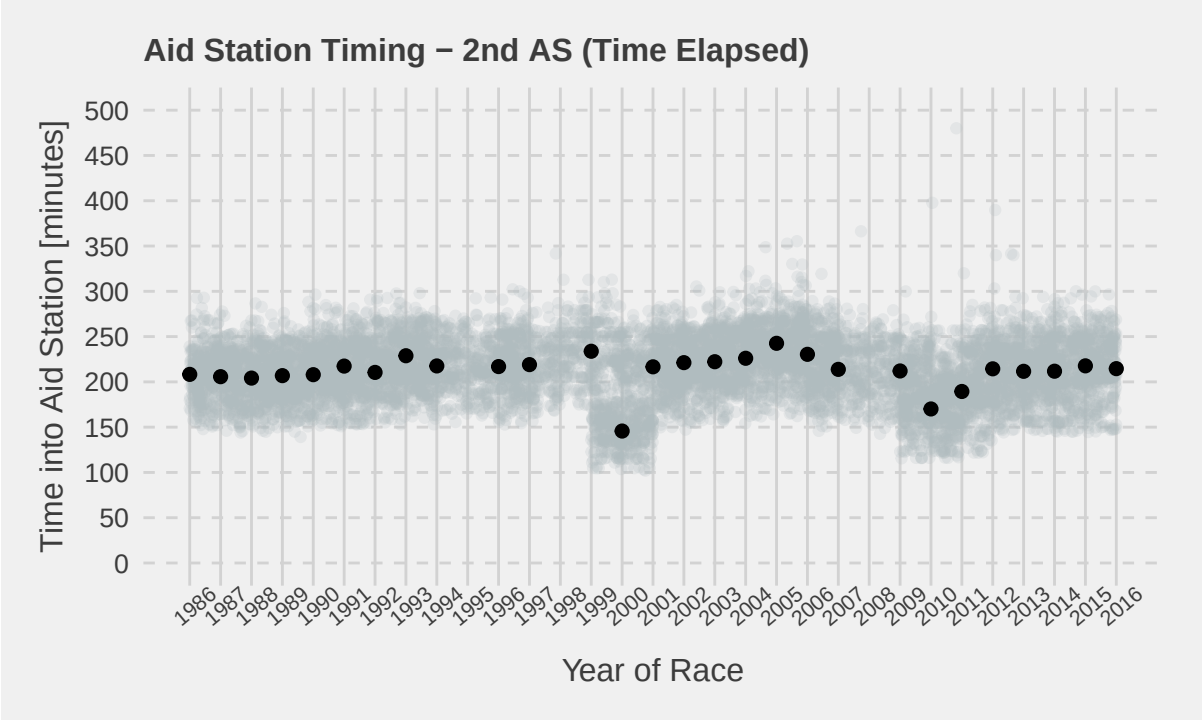


Figure 4: Time elapsed for all runners reaching the 2nd aid station

Figure 05 builds on the above and plots each aid station for all the years, with the black dot representing the mean for the aid station at the given distance.

Finishing Times

At the outset of this project, the goal was to capture finish times and evaluate that. The project has since evolved, but there are still good data with some interesting trends with regard to total finishing times. Below are a few of the explorations into the different times, categorized a few different ways.

Figure 06 shows a base plot of the total time it took to finish.

The first thing to notice is how there is a change in the shape of the curve about halfway up. The change in shape is the effect of runners trying to hit the 24-hour cutoff (1440 minutes).

By ranking each runner per year based on their overall time compared to all times recorded, a set of 10 quantile groups (deciles) can be established. By categorizing by this factor, the categories are more or less what would be expected (figure 07).

To understand what type of pace is required to get into each of the decile rankings, a separate plot for mean finishing times of each group will be useful. Figure 08 is an example of a take-away that a potential entrant can evaluate and perhaps apply to their training if they wish to make a certain group (without taking anything else into consideration, that is).

One last graph (figure 09) that I found rather interesting was comparing a runner's decile rank for the specific year (e.g. in 1995, the 45th place runner ranked in the 2nd decile) vs what their time would have them ranked for all years (e.g. the same runner is now ranked in the 3rd decile).

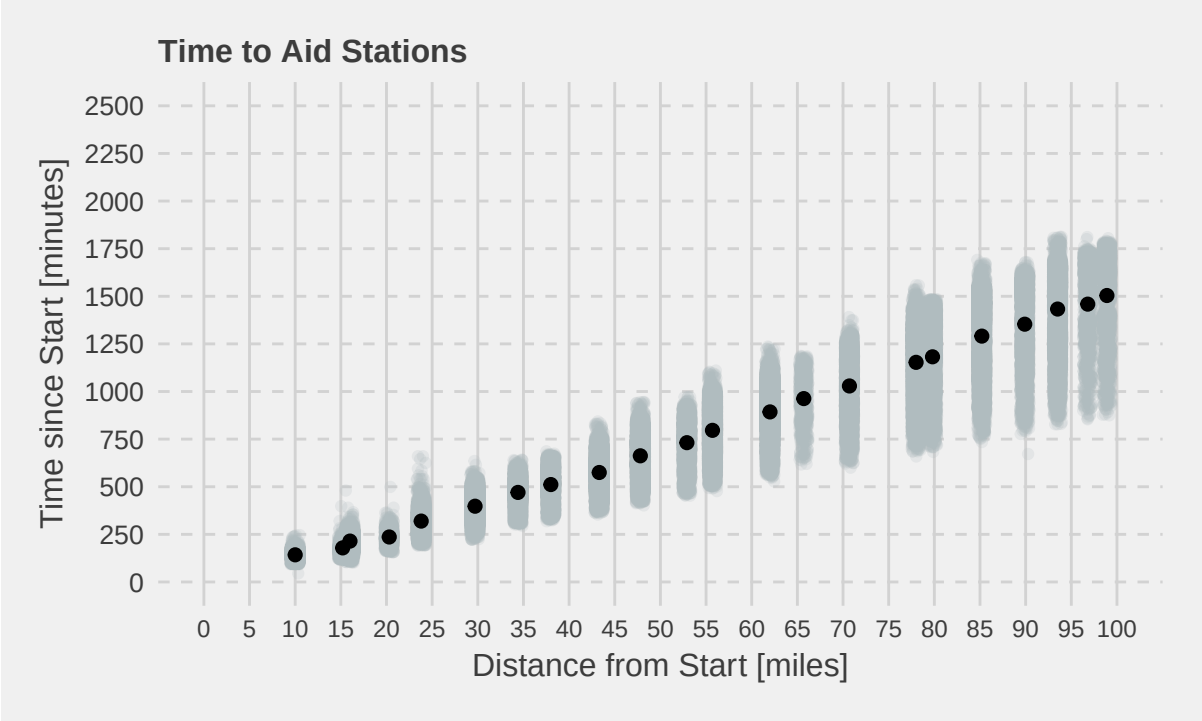


Figure 5: Time elapsed into each aid station

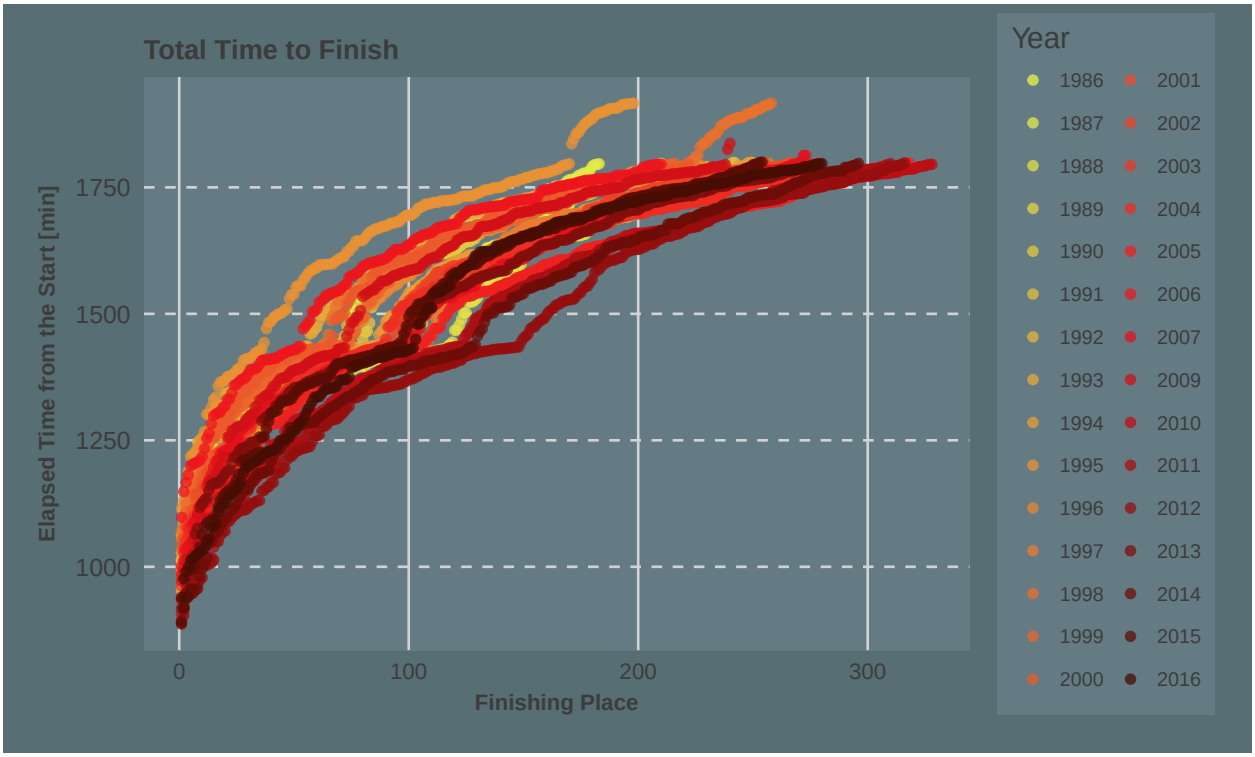


Figure 6: Total finishing times for all years

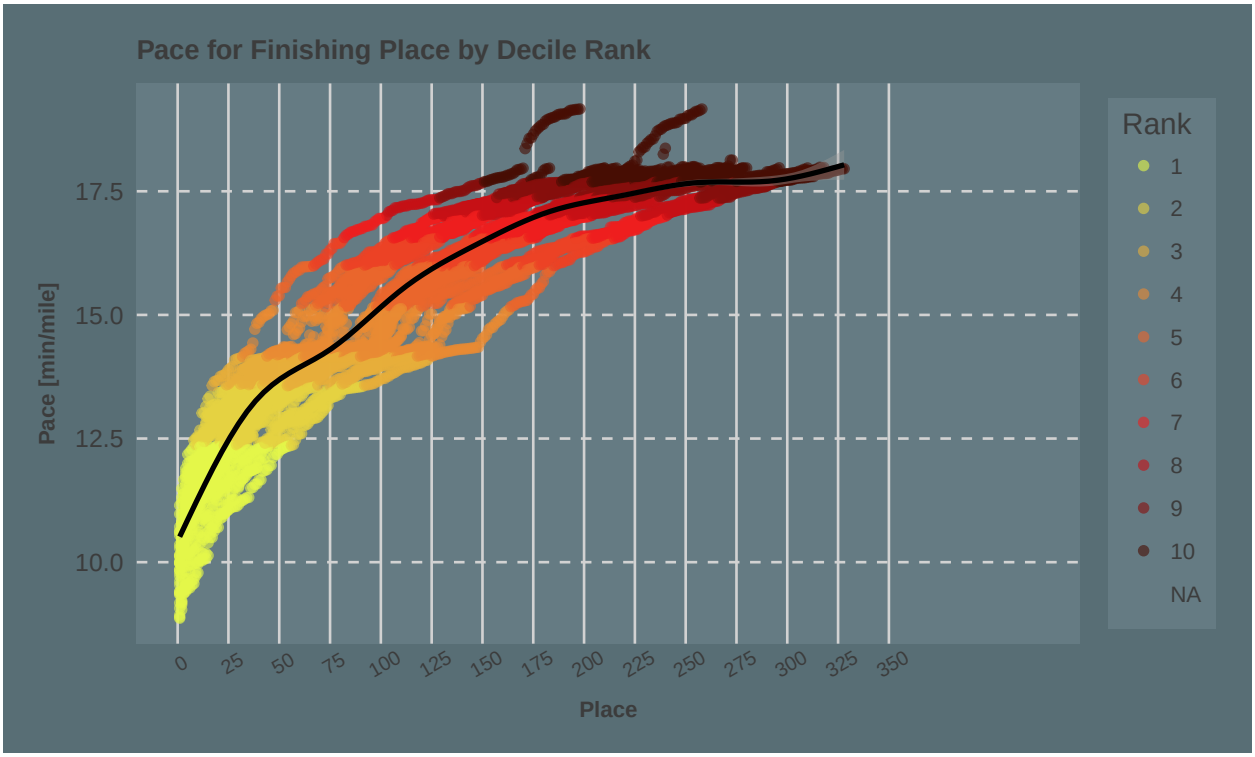


Figure 7: All finishers colored by their ranking

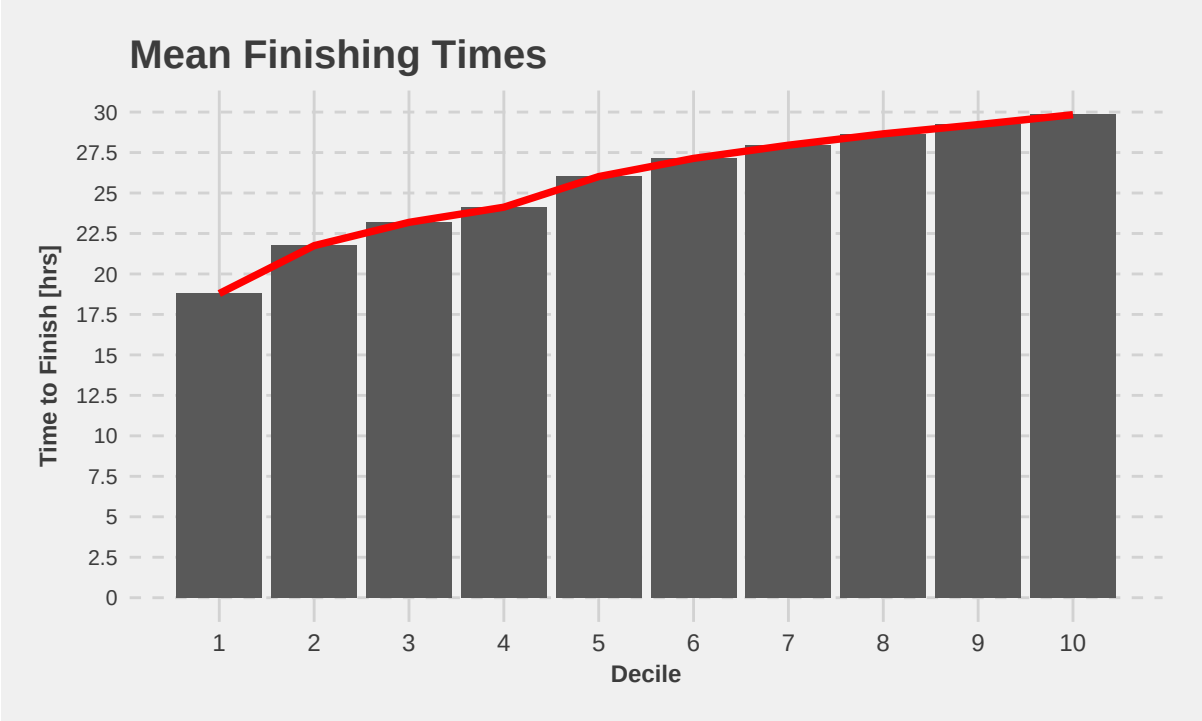


Figure 8: Finishing times for each decile ranking

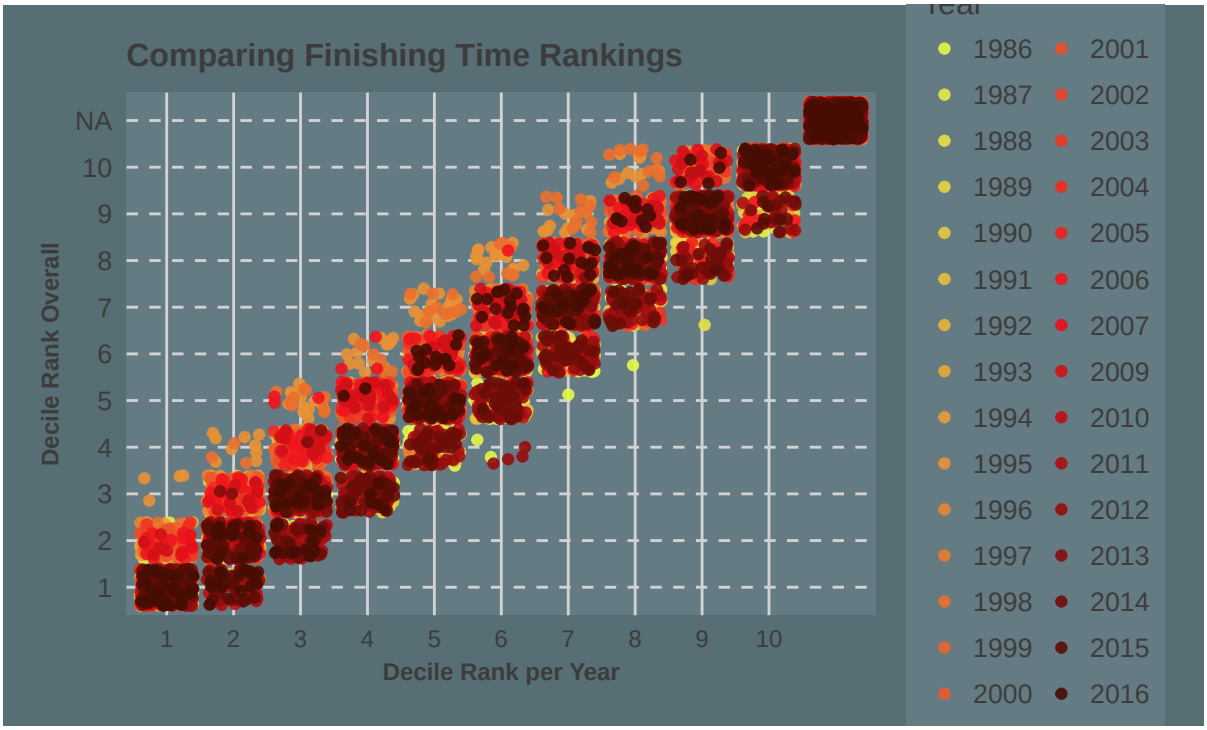


Figure 9: Rank in given year vs. overall rank

Further Categorization by Age and Gender

To tell the whole story, it will be necessary to understand how different categories may change the overall time, or allow a better look at how certain runners finish. I broke out both age and gender to help understand this a bit better.

As is evident from figure 10, the split between Men and Women is quite one-sided (in aggregate, 18% of entrants are female). Because of this, I did not include gender into building my final model due to a heavy bias towards males. Figure 11 shows the times for the two compared as well.

Another variable to consider with each runner is age. I broke out each age into age groups usually used in races in figure 12 (box plot seemed the most apt here). What is interesting is the number of runners in each category (figure 13). The cause for a small number in the younger (and faster) age groups, I suspect, has to do with the average age of ultra runners, and lottery process to get into the race.

Overall vs. DNF

All the plots above are essentially an exploration of those that finished the race. While there is still valuable information in the plots and associated data above, what really interested me was what distinguished a person from DNF'ing (Did Not Finish) and finishing.

First, figure 14 shows the histogram grouped by those that finished and DNF'd.

Overall - the DNF rate is 32%. To continue exploring this further, it was necessary to look at how fast runners went into the aid stations and how their rank may have changed over time.

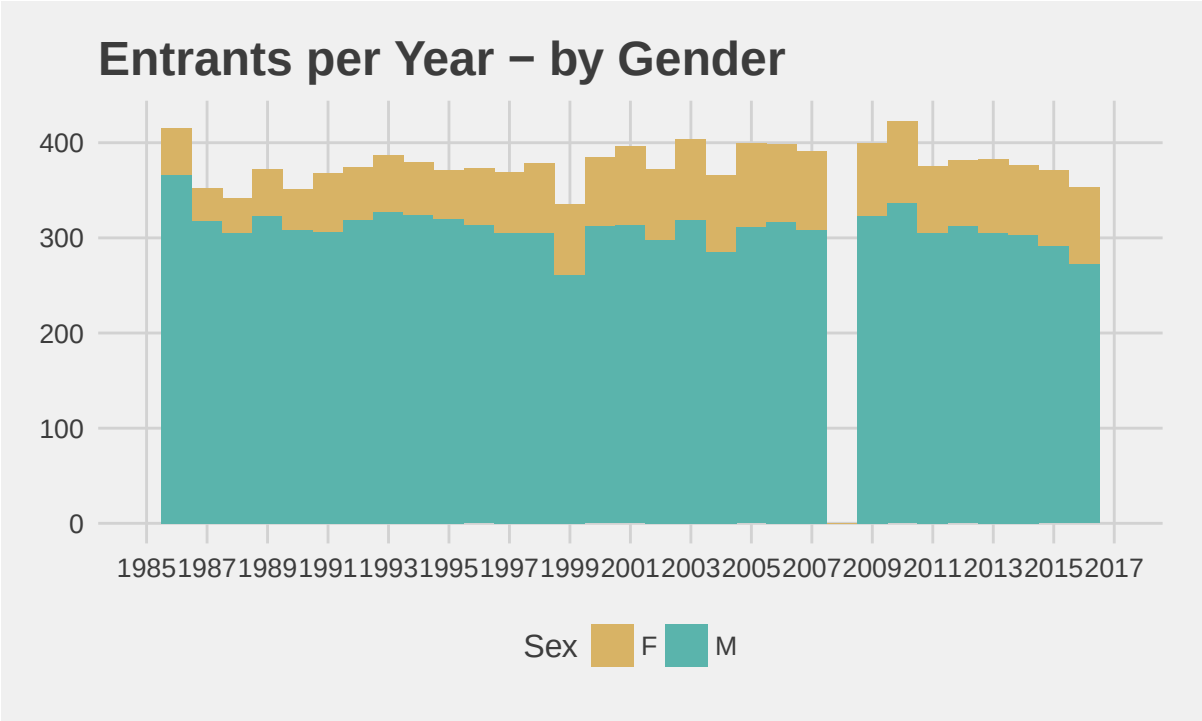


Figure 10: Histogram of gender split

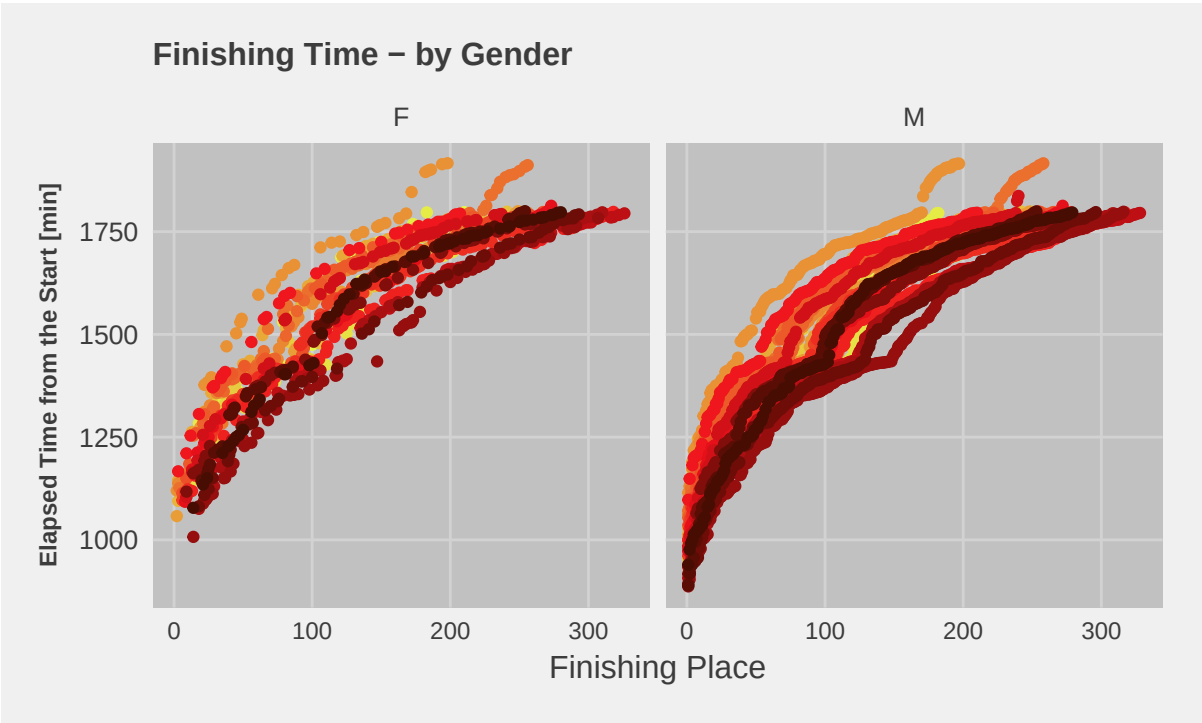


Figure 11: Splitting finish times by gender

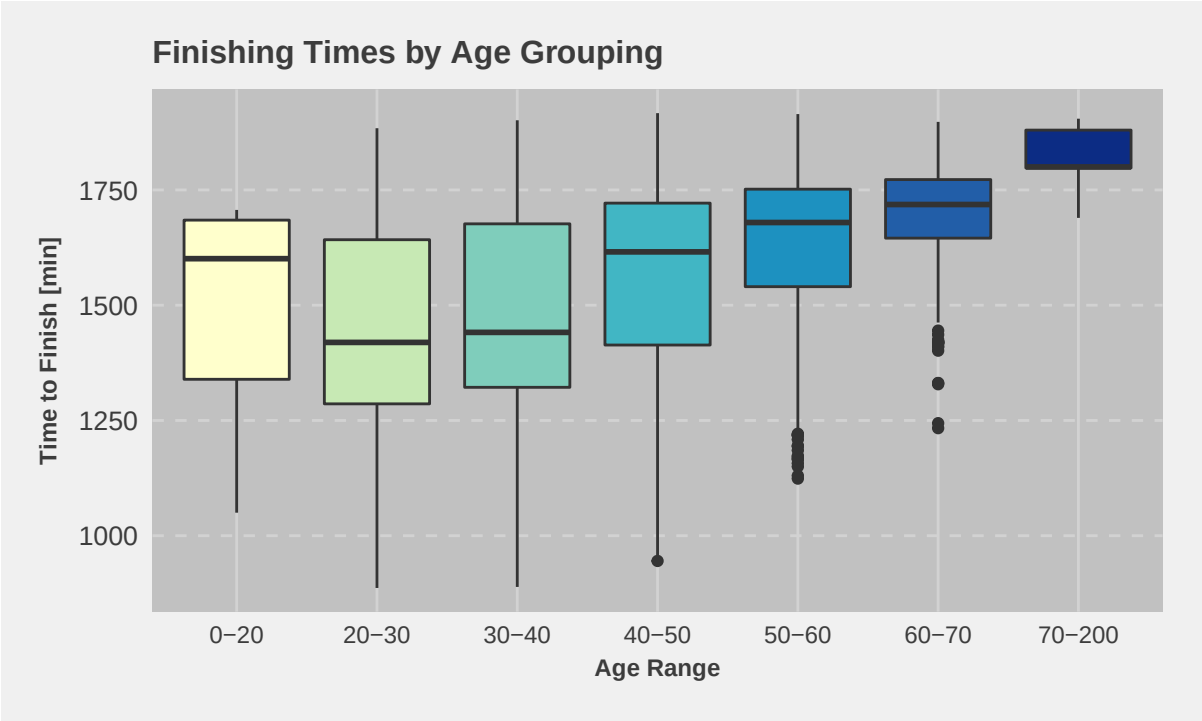


Figure 12: Box plot of finish times

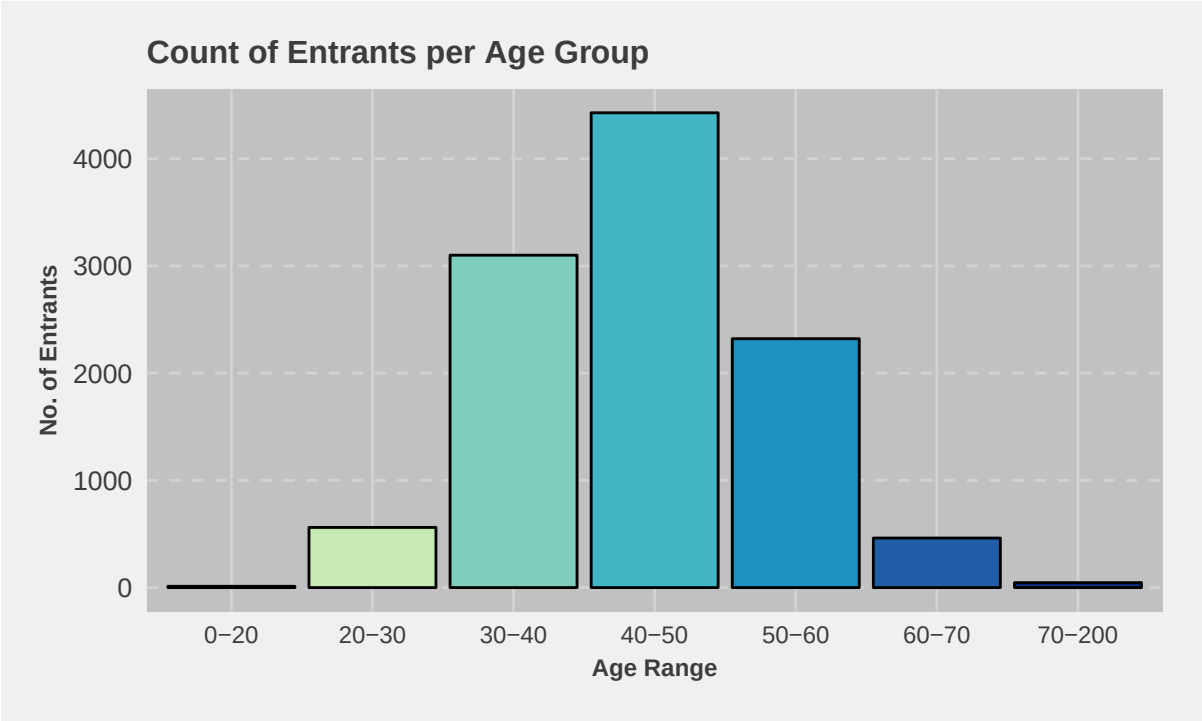


Figure 13: Age group histogram

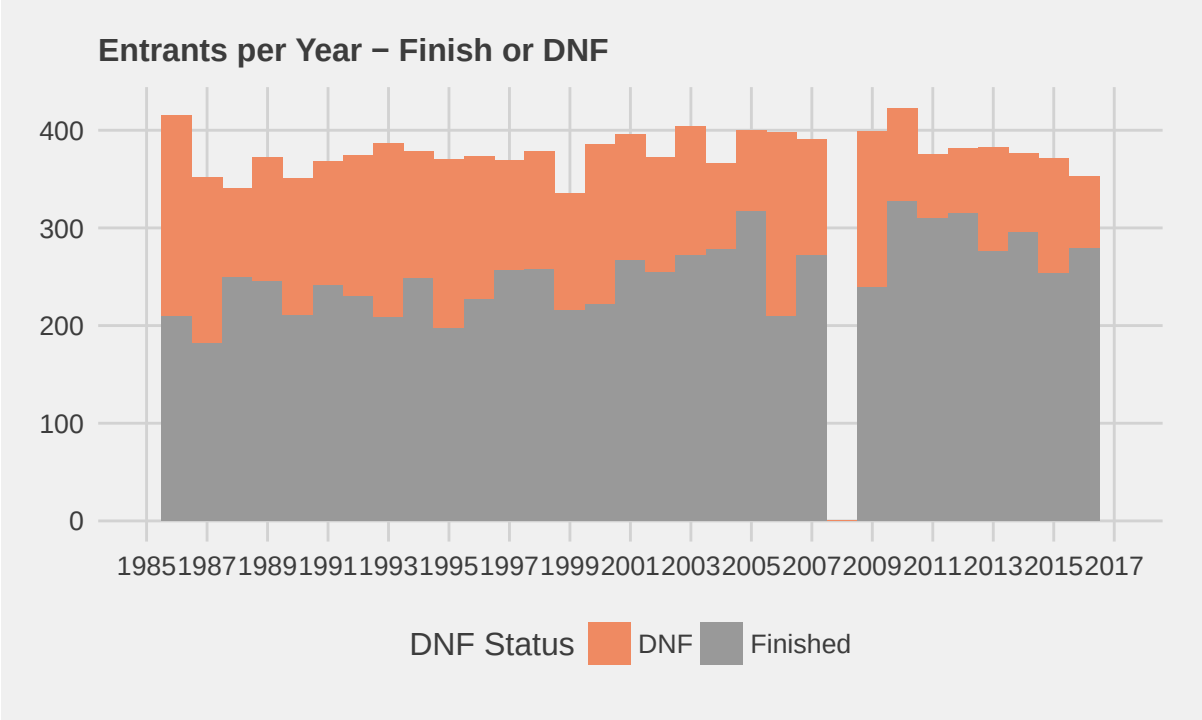


Figure 14: DNF Histogram

Aid Stations and the Story Within

I was able to break out each runner's times into aid stations and extract some valuable information from that. I could evaluate a runner's decile rank at each aid station (`qRank.as`), overall pace at each aid station (`total distance to aid station/time elapsed = pace.as`), and the differential pace to reach one aid station from the previous (`diff.pace`). From there, I was able to produce summary statistics showing the average pace into each aid station (`pace.avg`), the maximum number of aid stations a runner entered (`as.max`), and the standard deviation of a runner's pace through all the aid stations (`pace.sd`).

By plotting each runner's (x-axis) pace into each aid station, figure 15 gives a good idea of the speed with which each place (finisher or not) ran with. Keep in mind, for every tick on the x-axis, there are multiple paces for each aid station that person hit, along with multiple runners at each place for the different year.

By then breaking out each runner into their respective decile ranking, it becomes easier to see how the pace over time changes with each group in figure 16.

Adding a color group to the data for a runner's rank into each aid station helps to show in figure 17 how top runners stayed in the top rankings throughout the race (same color along a vertical section), while those in the DNF section seem to have more color change throughout their grouping. Keep in mind, again, that the x-axis represents the finishing place, while the color represents a runner's decile rank into multiple aid stations for a single year' race.

When I extracted out the average pace between aid stations, I was then able to compare how the pace of those that finished differed from those that did not (figure 18).

The reason there are so many different pace ranges for those that DNF'd is the number of aid stations that they may have gone through. Sometimes a runner will only go through the 1st or 2nd aid station before callling it quits, and may have gone too fast. Whereas, those that finished seem to have a fairly defined curve by making it through each aid station.

The standard deviation became of more interest to me as that feature may help reveal how consistent they

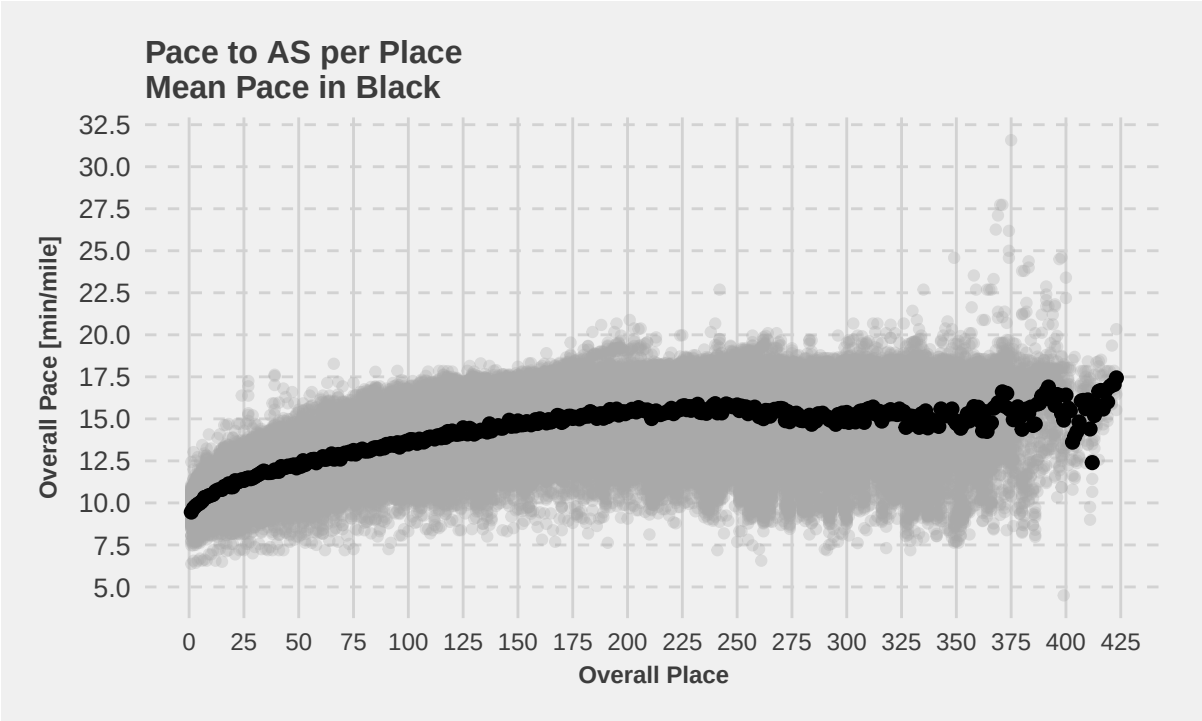


Figure 15: Mean pace per runner for every aid stations

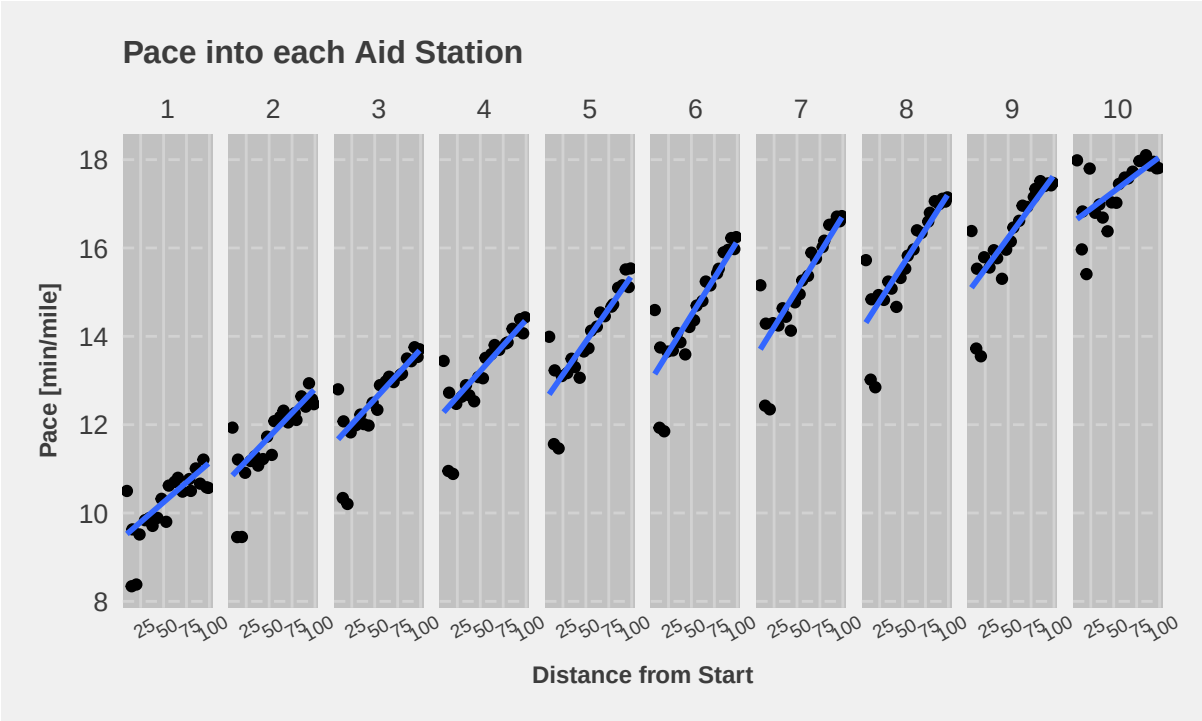


Figure 16: Aggregate of pace into aid stations by decile rank

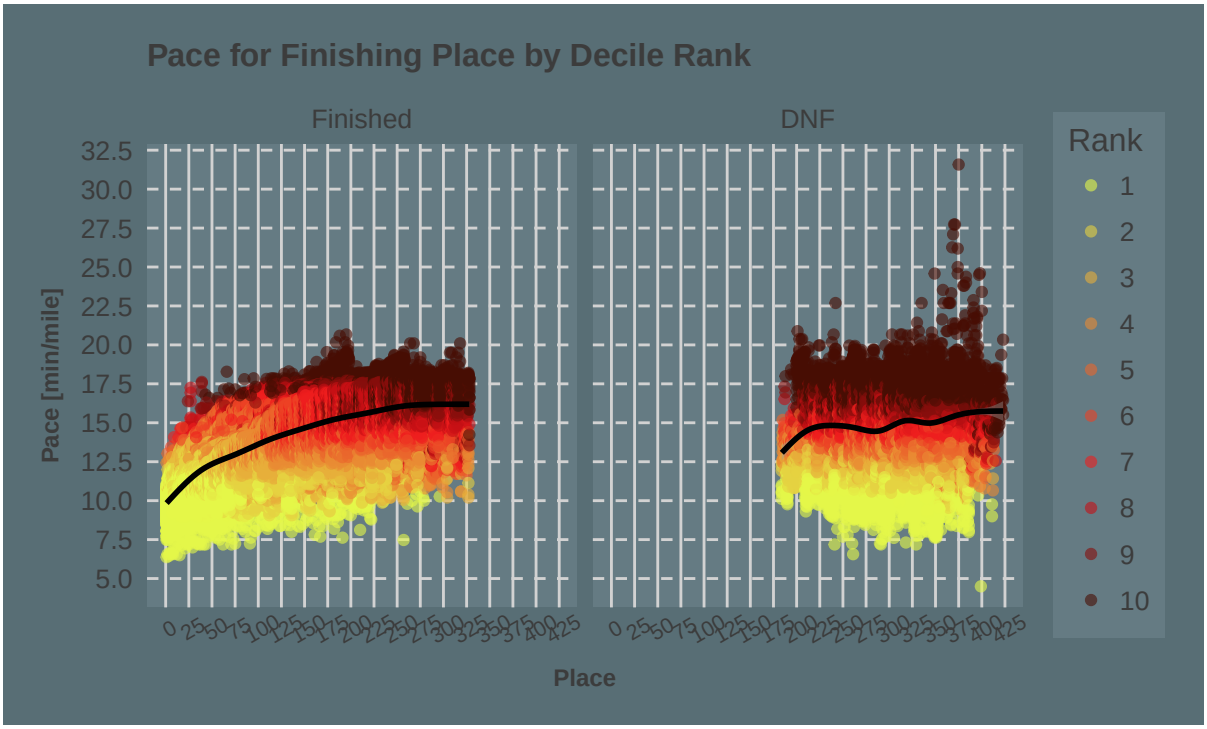


Figure 17: Individual paces for all years grouped and colored by rank into aid stations

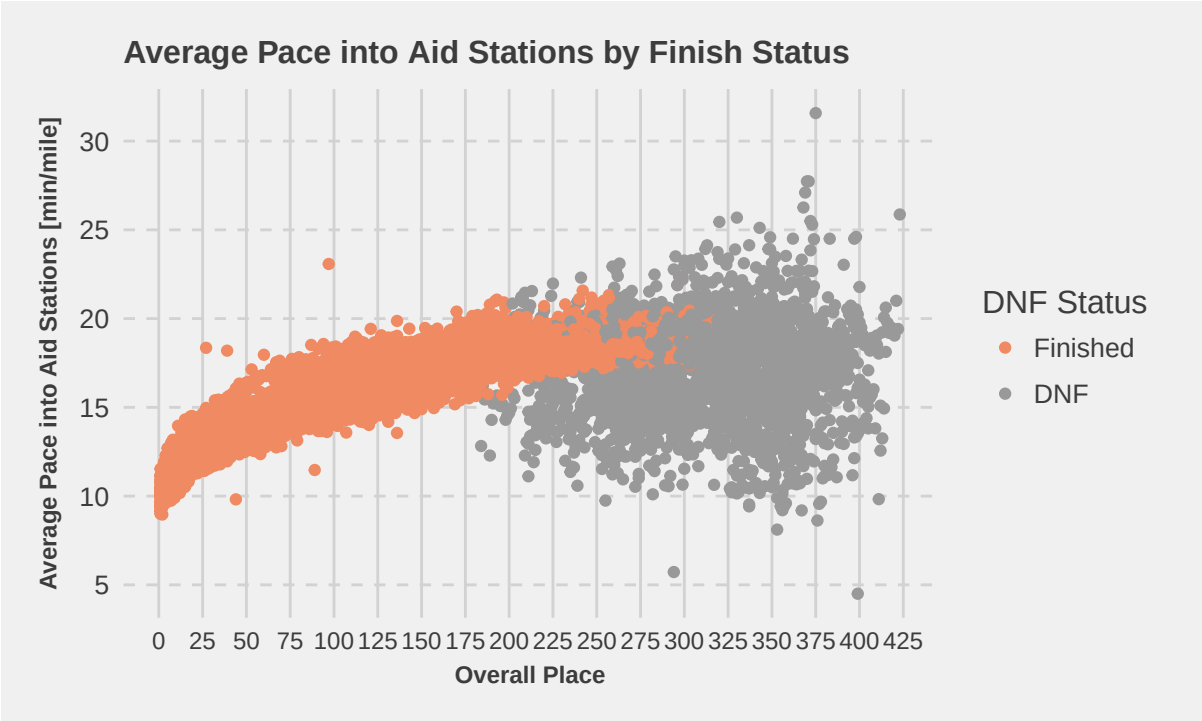


Figure 18: Average pace per runner into aid stations

were and whether or not a runner was about to DNF or able to make it through - shown in figure 19.

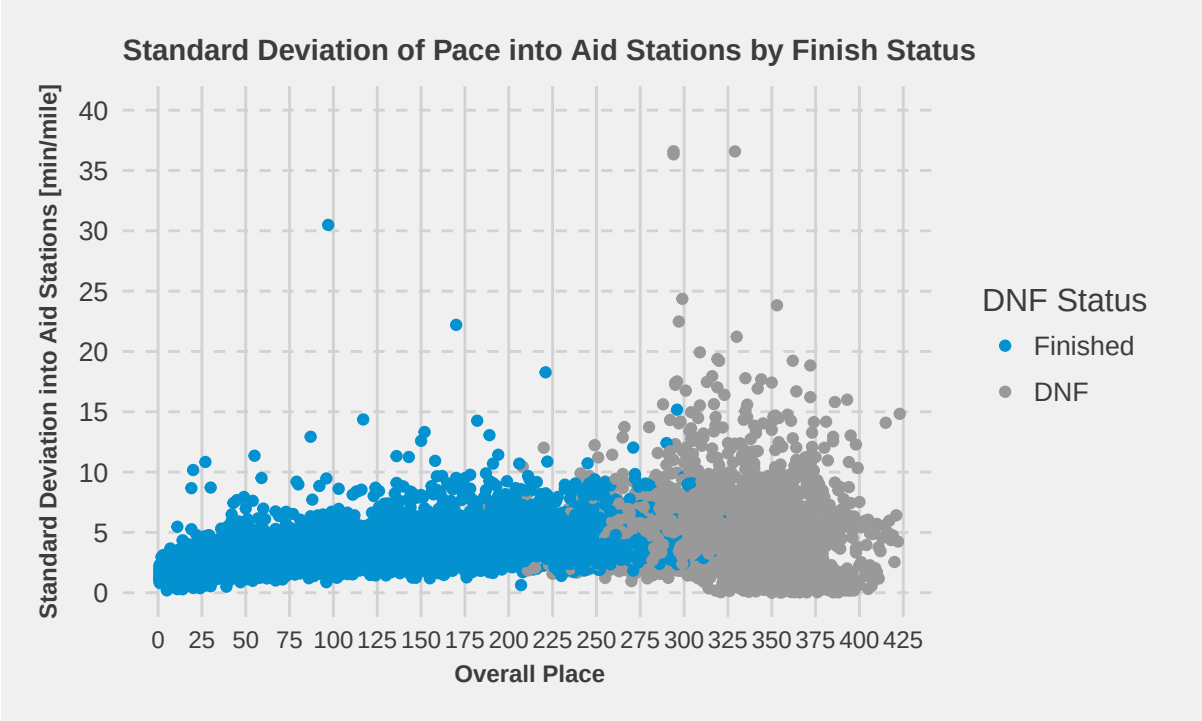


Figure 19: Standard Deviation for Single Runner's Pace to Aid Stations

The standard deviation of a runners pace maintains a similar curve to it, while those that DNF'd have no pattern to them.

The last few graphs on aid station timing and pacing became the focus for my final project. I wanted to be able to find the pattern and factors that are most significant, and to know what was the major cause for a person finishing vs not finishing the race.

WSER Finishing Model

By establishing a baseline for what the data can do, there are a few different things that can be predicted.

1. Predict whether or not a racer will finish.
2. Predict the different times a racer will show up to an aid station along the course based on their previous times.
3. Predict the final finish times and overall place.

The main goal with my project is to predict whether or not a runer will finish. We'll make the model based on the pre-race information (age, sex, bib) along with information from along the course to help determine if the runner will make it or not. While the other two aspects for prediction in the race would be extremely helpful, they will need to be addressed at a later time.

Taking in the different variables and engineered features added and shown above, it will be important to select the most impactful features and ones that minimize multicolinearity.

Below is a graph of variable importance using the function `boruta` from the package `Boruta` (more information can be found here). I won't get into the details, but it goes through a Random Forest model to parse out the importance of the different factors and affect they have on the output variable.

```
## [1] "YearInt"  "Place"    "Age"      "ageGroup" "pace.avg" "pace.sd"
## [7] "as.max"   "bibNo"
```

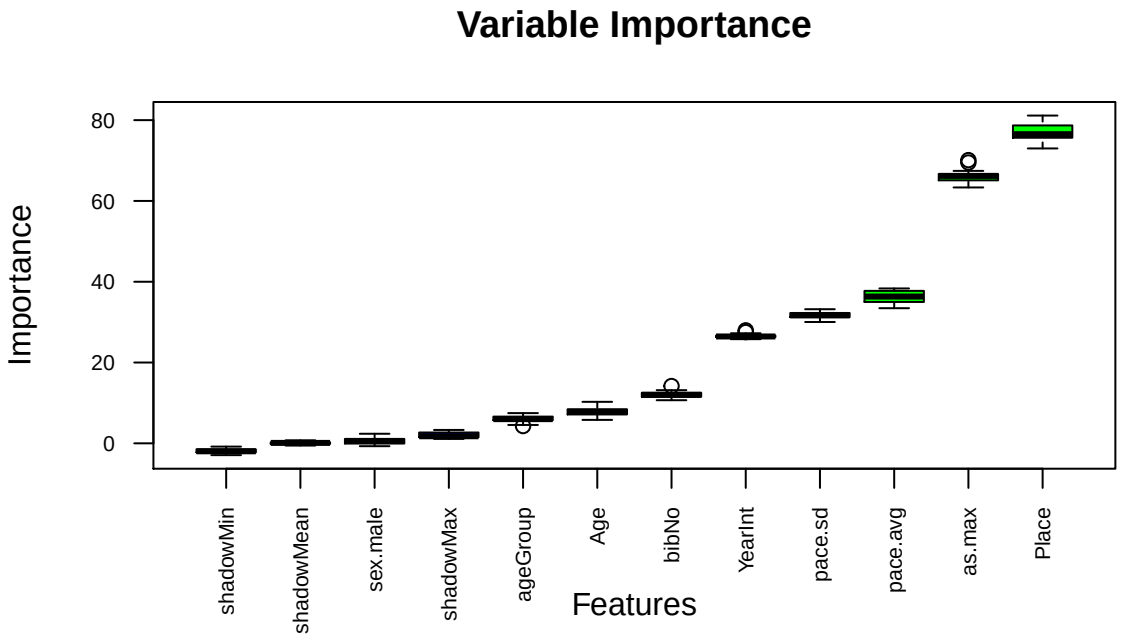


Figure 20: Variable Importance

Now, a correlation plot to show how those features are related, so as to better select the proper variables.

Selecting Features

From the `corrplot` and `variable importance` plot above, the two most impactful features are `Place` and `as.max`. It can be easy to just pick those to use in the model, but we'll need to understand what they mean and if we should actually use them.

Because `as.max` represents the total number of aid stations a runner hits, it makes sense that if a runner made it to more aid stations, they are likely to finish. So this variable, while highly correlated with DNF, can show that basically if a runner makes it to, let's say, aid station #8, they have a more likely chance of finishing the race.

While useful and interesting, the number of aid stations varied between the years due to various reasons, figure 17 shows the different aid stations over the years. Because of how the number of aid stations changes, the variable `as.max` would not accurately model whether or not a runner will finish. This also sheds light on why `as.max` and `YearInt` are highly correlated and why `YearInt` (or `Year`) will be excluded as well.

`Place` is the runner's final finishing placement, and therefore is similar to a runner's DNF status since a DNFers `Place` is given after all finisher's `Place` is assigned.

`ageGroup` is just a larger sub-category of `Age` and only one of those variables will be included. Races typically categorize based on age group, but since `Age` seems to have more importance, that will be the variable we use there.

`bibNo` is a runner's bibNo. While there really isn't much of a correlation here, there are certain bib numbers that are reserved for elites or those that get into the race a certain way. Therefore, `bibNo` offers a chance of

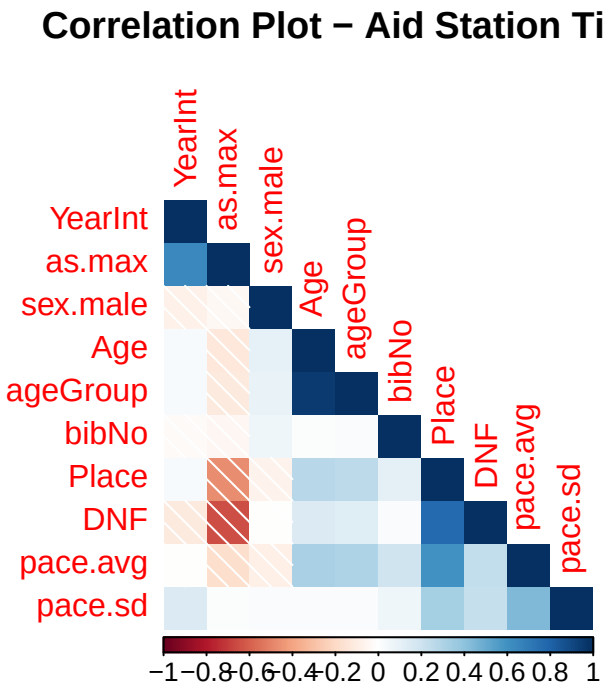


Figure 21: Correlation Plot

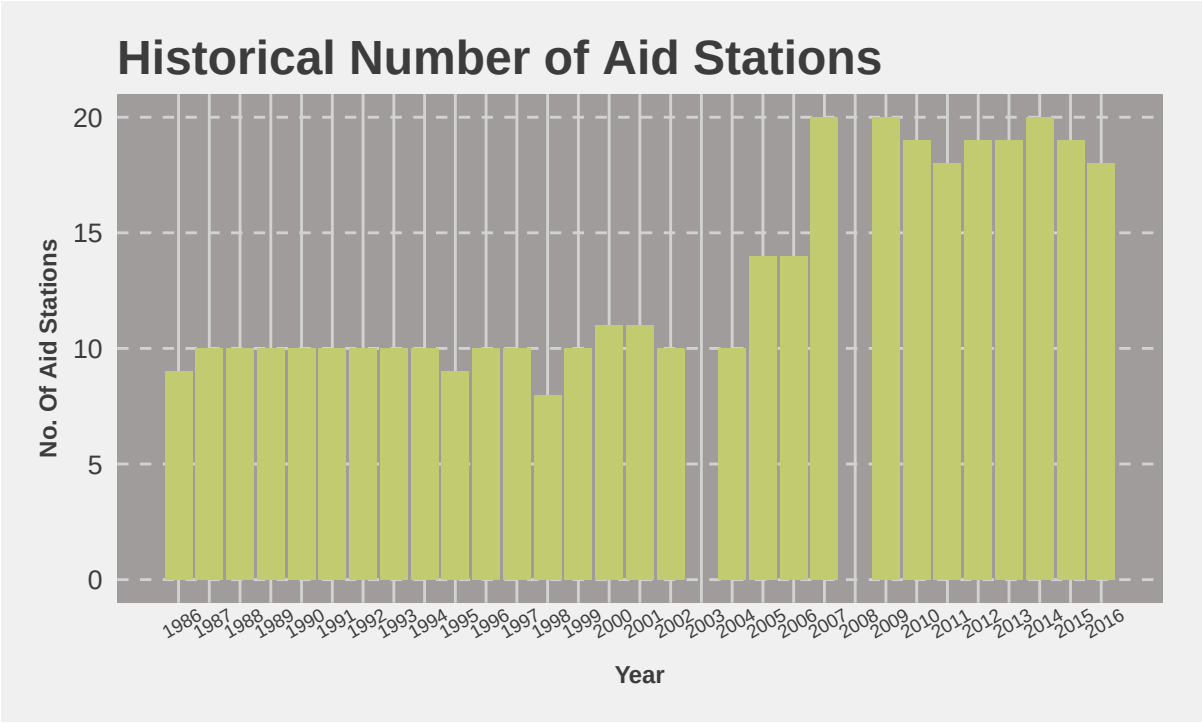


Figure 22: Aid Stations by Year

helping to add in some predictive power to those with the *fast* bibs.

The last variable to be used here has to do with pace. Both `pace.avg` and `pace.sd` are taken from a runner's time between each of the aid stations they reached. Calculating `pace.avg` just takes the distance between each of the aid stations and gets a pace for each section, then takes the mean for each runner. `pace.sd` is the standard deviation of those results different paces as well.

From here, the goal will be to come up with a model that uses the data above, and best predicts whether or not a runner will finish the Western States Endurance Run.

Base Prediction To Beat

To start, a baseline prediction for anyone starting the race. This way, any predictive models should be able to improve upon the accuracy of this basline.

Taking all the results and tabulating those that finished (`0`) and did not (`1`).

```
table(wser.cor$DNF)
```

```
##
##    0    1
## 7308 3470
```

That means the base model will predict a runner will finish with an accuracy of *67.8%*.

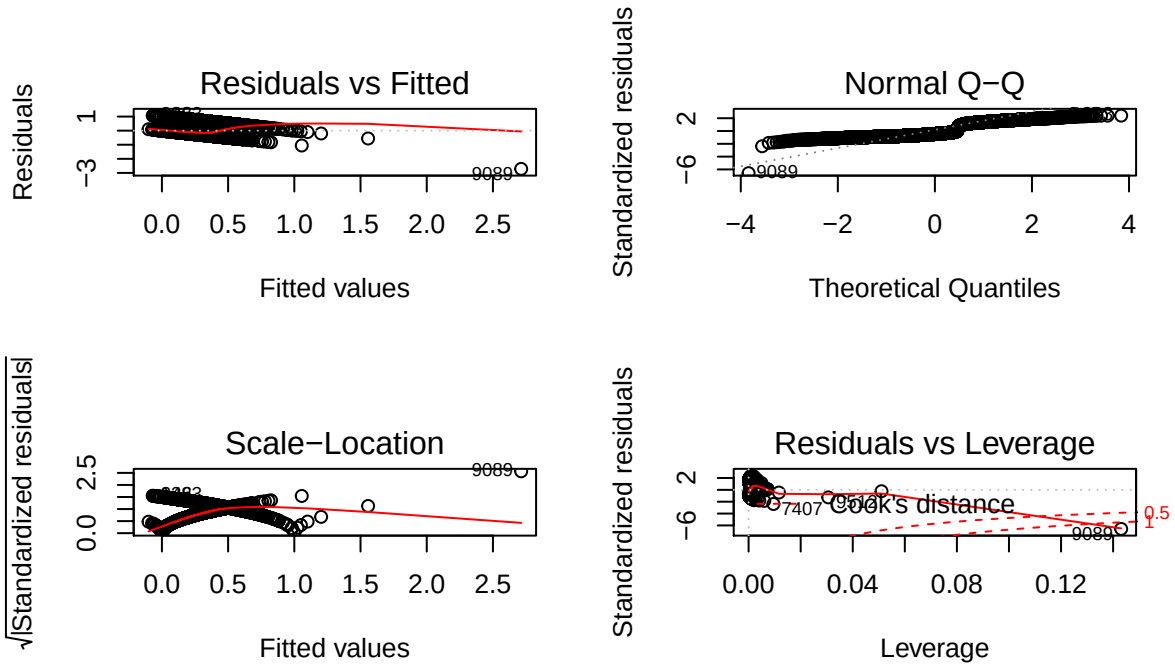
Linear Regression

The goal will see how much better we can predict a finish. First we'll start with a basic linear model.

```
## Linear Model
dnf.lm <- lm(DNF ~ bibNo + Age + pace.avg + pace.sd, data = dnfTrain)
summary(dnf.lm)
```

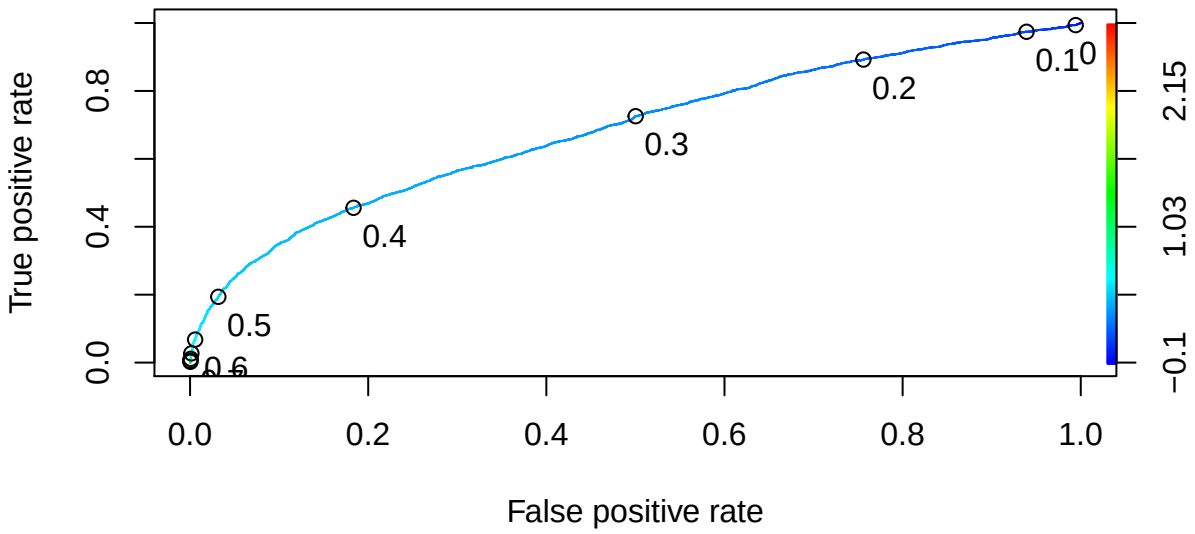
```
##
## Call:
## lm(formula = DNF ~ bibNo + Age + pace.avg + pace.sd, data = dnfTrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.7133 -0.3439 -0.2087  0.5129  1.0693
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.398e-01  3.798e-02 -14.215   <2e-16 ***
## bibNo       -9.266e-05  3.930e-05  -2.358   0.0184 *
## Age          5.445e-03  5.847e-04   9.312   <2e-16 ***
## pace.avg     3.082e-02  2.577e-03  11.959   <2e-16 ***
## pace.sd      3.377e-02  2.577e-03  13.104   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4453 on 8078 degrees of freedom
## Multiple R-squared:  0.09207,    Adjusted R-squared:  0.09162
## F-statistic: 204.8 on 4 and 8078 DF,  p-value: < 2.2e-16
```

```
par(mfrow = c(2,2))
plot(dnf.lm)
```




```
dnf.lm.train <-stats::predict(dnf.lm, type = 'response')

rocrPred <- ROCR::prediction(dnf.lm.train, dnfTrain$DNF)
rocrPerf <- performance(rocrPred, 'tpr', 'fpr')
par(mfrow = c(1,1))
plot(rocrPerf, colorize = TRUE,
     print.cutoffs.at = seq(0, 1, 0.1),
     text.adj = c(-0.2, 1.7))
```




```
dnf.lm.test <- stats::predict(dnf.lm, dnfTest)

table(dnfTest$DNF, dnf.lm.test > 0.41)
```

```
##
##     FALSE TRUE
##   0  1522  305
##   1   504  364
```

```
#lm.rmse <- sqrt(mean((dnfTest$DNF - dnf.lm.test)^2))
acc <- round(sum((dnf.lm.test > 0.41) == dnfTest$DNF)/length(dnfTest$DNF),3)
#print(paste("RMSE for Linear Model:", round(lm.rmse, 3)))
print(paste("Accuracy for Linear Model:", acc))
```

```
## [1] "Accuracy for Linear Model: 0.7"
```

The linear model is a first shot at determining a decent regression model for the data here. There are limitations to a linear model, and it is not the best at a binary classification. The accuracy on the test data is what will be used to compare all the different models.

Logistic Regression

Logistic regression may offer a better fit than linear regression as it has the capability of classifying an output into 1 of 2 groups.

```
#####
# GLM Model k-Fold Cross Validation
#####

ctrl <- trainControl(method = "repeatedcv", number = 10,
                                            repeats = 3)
```

```
set.seed(352)
dnf.glm <- train(factor(DNF) ~ bibNo + Age + pace.avg + pace.sd,
                    data = dnfTrain,
                    method = "glm",
                    family = "binomial",
                    trControl = ctrl)

## Coefficients
exp(coef(dnf.glm$finalModel))
```

```
## (Intercept)       bibNo         Age    pace.avg     pace.sd
## 0.005015575 0.999624519 1.027587173 1.172675155 1.202075378
```

```
## Apply Model to prediction
dnf.glm.pred <- predict(dnf.glm, newdata = dnfTest)

## Summary of Results
cm <- confusionMatrix(dnf.glm.pred, dnfTest$DNF)
cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1738  661
##          1   89  207
##
##                Accuracy : 0.7217
##                  95% CI : (0.7044, 0.7386)
##     No Information Rate : 0.6779
##     P-Value [Acc > NIR] : 4.692e-07
##
##                   Kappa : 0.2294
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9513
##             Specificity : 0.2385
##          Pos Pred Value : 0.7245
##          Neg Pred Value : 0.6993
##              Prevalence : 0.6779
##          Detection Rate : 0.6449
##    Detection Prevalence : 0.8902
##       Balanced Accuracy : 0.5949
##
##        'Positive' Class : 0
##
```

That seems to give us pretty good results! The model predicts with an accuracy of 0.7217 But can we do better?

Support Vector Machine

For the Support Vector Machine, we'll use the base parameters to give an idea to how SVM performs. There are a few different kernel methods to choose when creating the SVM, but here we stuck with a fairly basic

Radial Basis Function put through 10-fold cross validation to help with overfitting (`svmRadial`).

```
## Setup model

## NOT USED ##
#set.seed(352)

#dnf.svm <- ksvm(factor(DNF) ~ bibNo + Age + pace.avg + pace.sd,
#                data = dnfTrain)

###############

## Base SVM Model

set.seed(352)
dnf.svm <- train(factor(DNF) ~ bibNo + Age + pace.avg + pace.sd,
                 data = dnfTrain,
                 method = "svmRadial",
                 trControl = trainControl(method = "repeatedcv", number = 10,
                                          repeats = 3))


dnf.svm.pred <- predict(dnf.svm, dnfTest)

dnf.svm
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 8083 samples
##    4 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 7275, 7275, 7274, 7275, 7275, 7275, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7610217  0.3691622
##   0.50  0.7650634  0.3815908
##   1.00  0.7677439  0.3900921
##
## Tuning parameter 'sigma' was held constant at a value of 0.3280702
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were sigma = 0.3280702 and C = 1.
```

```
table(dnfTest$DNF, dnf.svm.pred)
```

```
##    dnf.svm.pred
##        0    1
##   0 1705  122
##   1  534  334
```

```
acc <- round(sum(dnf.svm.pred == dnfTest$DNF)/length(dnfTest$DNF),3)
names(acc) <- "accuracy"
acc
```

```
## accuracy
##    0.757
```

With the SVM model, we're able to see much different results. Potentially due to the fact that the final classification we're seeking (Finish or DNF), has some heavy outliers that a linear model won't necessarily capture, and SVM has the ability to classify a bit better.

While there are additional modifications that can be made towards the model in terms of tuning, for now, we will stick with the base model with only minor modifications to the model.

Regression Trees

There are two separate classification tree algorithms we can use. The first is the standard CART method (Classification And Regression Tree). The second is the Random Forest. We'll utilize both approaches.

CART

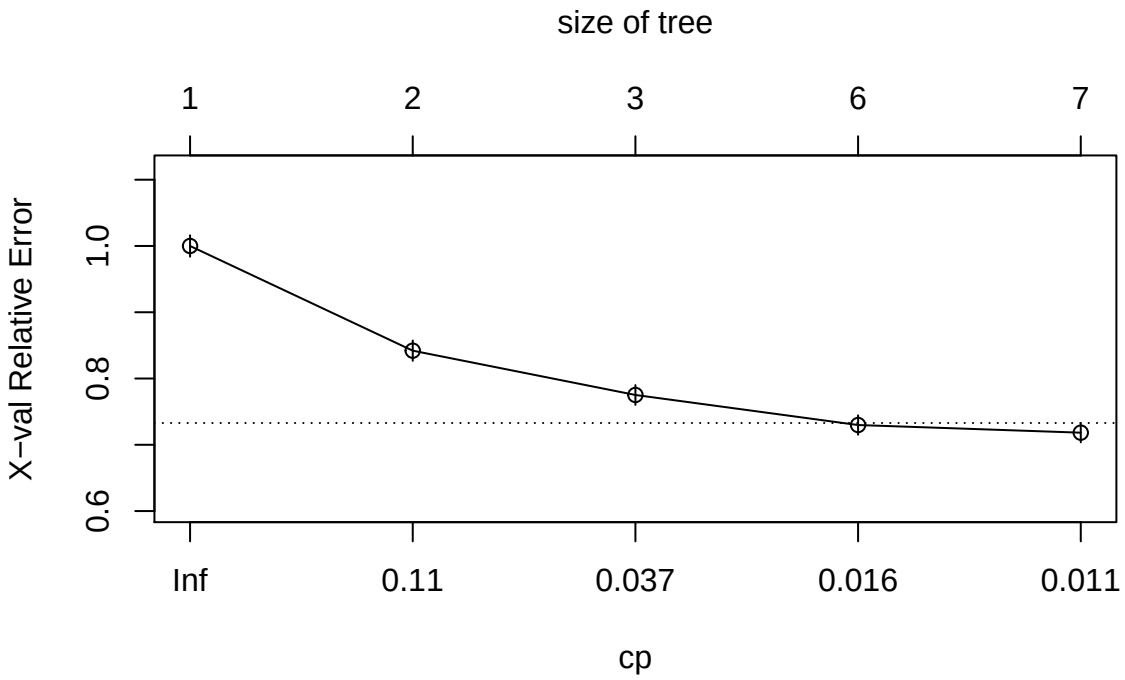
For the CART method, I ended up using two different libraries to compute the model for CART. The reason is that while both are using `rpart`, `dnf.tree` is called directly through `rpart` and saved as an `rpart` object, therefore giving access to the `rattle` library and having a nice plot output.

The other `rpart` model is created using `caret` with 10-fold cross validation. While the method starts the same, the two must go through some sort of selection process or automatic methods that the other is not, as the two models are not equal. Quite honestly, I'm not sure where they differ and that will take further exploration and reading to understand. I'll show the comparison below, but despite them not being equal, I use the `rpart` to plot the decision tree, and `caret::rpart` to compare to other models.

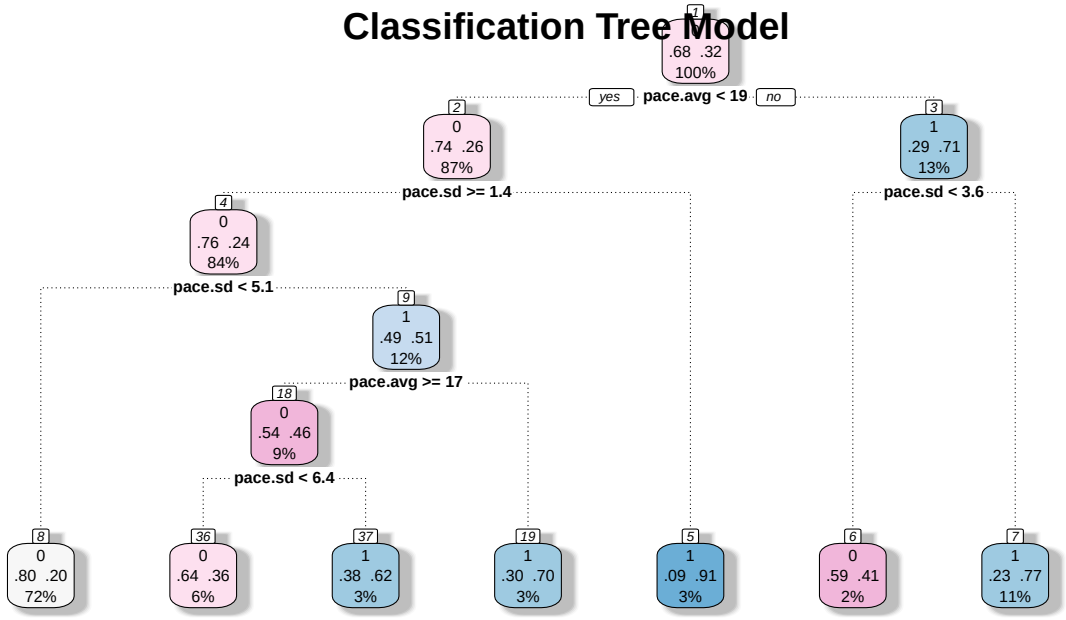
```
# Classification Tree
set.seed(352)
dnf.tree <- rpart(factor(DNF) ~ bibNo + Age + pace.avg + pace.sd,
                  data = dnfTrain)

set.seed(352)
dnf.tree.t <- train(factor(DNF) ~ bibNo + Age + pace.avg + pace.sd,
                    data = dnfTrain,
                    method = "rpart",
                    trControl =
                    trainControl(method = 'repeatedcv', number = 10, repeats = 3))

plotcp(dnf.tree)
```




```
#summary(dnf.tree.p)
fancyRpartPlot(dnf.tree, main = "Classification Tree Model", palettes = "PiYG")
```




```
### Applying Model to test data
dnf.tree.pred <- predict(dnf.tree, dnfTest)

#tree.rmse <- sqrt(mean((dnfTest$DNF - dnf.tree.pred)^2))
#print(paste("RMSE of Classification Tree: ", round(tree.rmse,3)))
```

```
## Accuracy of Model
## Split Point for Probability is centered around 0.5
dnf.tree.pred <- as.data.frame(dnf.tree.pred)
dnf.tree.pred$DNF <- factor(ifelse(dnf.tree.pred$`0` > 0.5, 0, 1))
ref <- factor(dnfTest$DNF)

cartTree.cm <- confusionMatrix(data = dnf.tree.pred$DNF, reference = ref)
cartTree.cm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1669  482
##          1  158  386
##
##                Accuracy : 0.7625
##                  95% CI : (0.746, 0.7785)
##     No Information Rate : 0.6779
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3971
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9135
##             Specificity : 0.4447
##          Pos Pred Value : 0.7759
##          Neg Pred Value : 0.7096
##              Prevalence : 0.6779
##          Detection Rate : 0.6193
##    Detection Prevalence : 0.7981
##       Balanced Accuracy : 0.6791
##
##        'Positive' Class : 0
##
```

```
### Comparison to CARET rpart
dnf.predCaret <- predict(dnf.tree.t, dnfTest)
caretTree.cm <- confusionMatrix(data = dnf.predCaret, ref)
```

Comparison of the accuracy of the two models:

rpart	caret::rpart
0.7625	0.744

Random Forest

```
######################
## Random Forest
######################

set.seed(352)
dnf.randForest <- train(factor(DNF) ~ bibNo + Age + pace.avg + pace.sd,
```

```
                  data = dnfTrain,
                  method = "rf", trControl =
                    trainControl(method = 'repeatedcv', number = 10, repeats = 3),
                  prox = TRUE, allowParallel = TRUE)

dnf.randForest
```

```
## Random Forest
##
## 8083 samples
##    4 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 7275, 7275, 7274, 7275, 7275, 7275, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.7738041  0.4410562
##   3     0.7715363  0.4361602
##   4     0.7687715  0.4311074
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
getTrainPerf(dnf.randForest)
```

```
##   TrainAccuracy TrainKappa method
## 1     0.7738041  0.4410562     rf
```

```
## Applying Model to test data
dnf.randForest.pred <- predict(dnf.randForest, dnfTest)

## Calculating Accuracy
confusionMatrix(dnf.randForest.pred, reference = ref)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1621  427
##          1  206  441
##
##                Accuracy : 0.7651
##                  95% CI : (0.7486, 0.781)
##     No Information Rate : 0.6779
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4236
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8872
##             Specificity : 0.5081
##          Pos Pred Value : 0.7915
```

```
##          Neg Pred Value : 0.6816
##              Prevalence : 0.6779
##          Detection Rate : 0.6015
##    Detection Prevalence : 0.7599
##       Balanced Accuracy : 0.6977
##
##        'Positive' Class : 0
##
```

Neural Network

It may not be completely applicable to use a neural network for this model with the limited variables and binary classification, but nonetheless, it is worthwhile to go through this model and see what results we can find. We'll just use the base setup for a neural net from the package `nnet::nnet`.

The model will is using `caret` to make the predictions and we'll set it up using 2 hidden layers.

```
#######################
## Neural Net Model
#######################

set.seed(352)
dnf.net <- train(factor(DNF) ~ bibNo + Age + pace.avg + pace.sd,
                 data = dnfTrain,
                 method = "nnet", hidden = 2,
                 trControl = trainControl(method = "repeatedcv", number = 10, repeats =3))
```

```
## Summary and Accuracy of Trained Model
print(dnf.net$results)
```

```
##   size decay  Accuracy     Kappa AccuracySD    KappaSD
## 1    1 0e+00 0.6981732 0.1029974 0.02795958 0.13906743
## 2    1 1e-04 0.7034457 0.1266763 0.03026887 0.14916982
## 3    1 1e-01 0.7132640 0.1764392 0.02652622 0.12929744
## 4    3 0e+00 0.7216777 0.2223620 0.02824119 0.13894274
## 5    3 1e-04 0.7125182 0.1772707 0.03100919 0.15217772
## 6    3 1e-01 0.7429575 0.3202540 0.01911735 0.08127053
## 7    5 0e+00 0.7213064 0.2171297 0.03075339 0.14848608
## 8    5 1e-04 0.7258026 0.2521194 0.02561403 0.12250097
## 9    5 1e-01 0.7428348 0.3244633 0.01721762 0.05536130
```

```
# Applying Model to Test Data
dnf.net.pred <- predict(dnf.net, newdata = dnfTest)

## Confusion Matrix of Prediction
confusionMatrix(dnf.net.pred, reference = ref)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1671  527
##          1  156  341
##
##                Accuracy : 0.7466
##                  95% CI : (0.7297, 0.7629)
```

```
##     No Information Rate : 0.6779
##     P-Value [Acc > NIR] : 4.018e-15
##
##                   Kappa : 0.3463
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9146
##             Specificity : 0.3929
##          Pos Pred Value : 0.7602
##          Neg Pred Value : 0.6861
##              Prevalence : 0.6779
##          Detection Rate : 0.6200
##    Detection Prevalence : 0.8156
##       Balanced Accuracy : 0.6537
##
##        'Positive' Class : 0
##
```

Comparing Results

Because I was able to build the models using `caret`, each model is stored similarly across the different methods and therefore I am able to compare them using a few built in functions. Below is a representation of those different models, and their accuracies compared.

```
results <- resamples(list(Forest = dnf.randForest, nnet = dnf.net, SVM = dnf.svm, GLM = dnf.glm, CART =
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: Forest, nnet, SVM, GLM, CART
## Number of resamples: 30
##
## Accuracy
##          Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## Forest 0.7550  0.7652 0.7723 0.7738  0.7811 0.7933    0
## nnet   0.6782  0.7331 0.7458 0.7430  0.7519 0.7822    0
## SVM    0.7512  0.7600 0.7661 0.7677  0.7785 0.7896    0
## GLM    0.7067  0.7168 0.7240 0.7239  0.7311 0.7376    0
## CART   0.7370  0.7466 0.7519 0.7534  0.7574 0.7775    0
##
## Kappa
##          Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## Forest 0.4010  0.4238 0.4342 0.4411  0.4606 0.4918    0
## nnet   0.0000  0.2812 0.3345 0.3203  0.3719 0.4373    0
## SVM    0.3306  0.3627 0.3845 0.3901  0.4174 0.4486    0
## GLM    0.1734  0.2165 0.2340 0.2331  0.2524 0.2709    0
## CART   0.3097  0.3521 0.3899 0.3808  0.4068 0.4706    0
```

```
## Dot Plot of Accuracy
scales <- list(x = list(relation="free"), y = list(relation="free"))
dotplot(results, scales = scales)
```

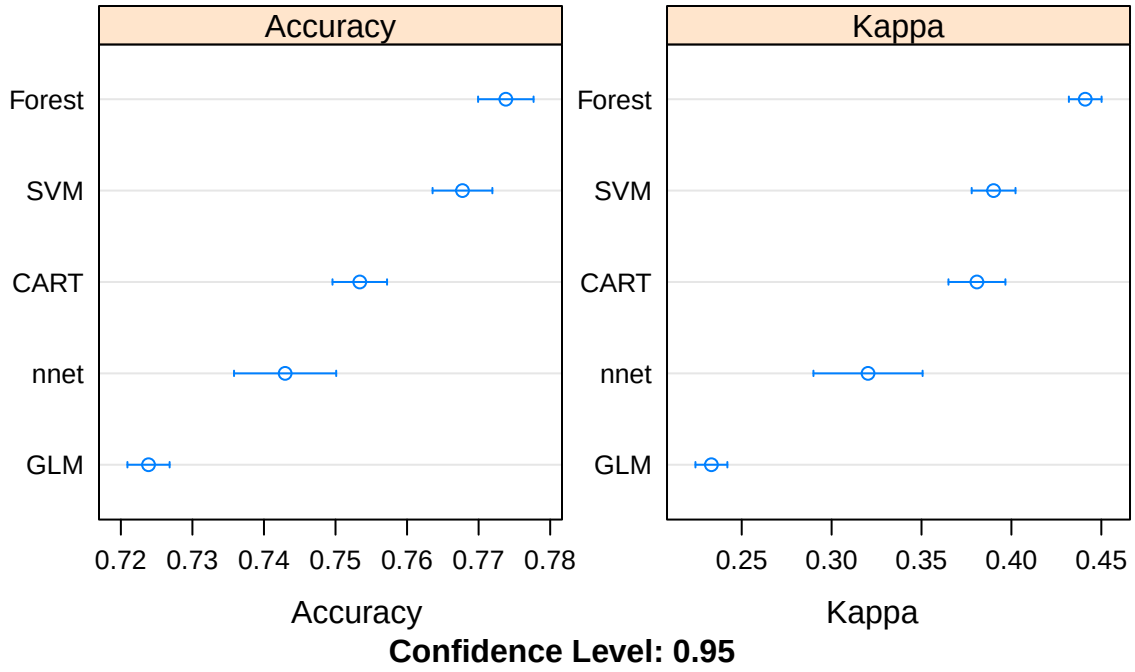


Figure 23: Comparison of Model Results

Summary of the Model

By looking at the results table and comparison graphs, the final results show us that, based on pure numbers, the random forest model gets us the best results of predicting a finish at the Western States. While the difference in accuracy is not all that different, it is still noticeable, especially from the GLM model or the baseline of 68%.

The only downside is I was not able to replicate the decision tree properly using `caret` and `rpart`, however, the `rpart` model has very similar accuracy to the `randomForest`. The reason I find this a downside is the decision tree is easy to follow with *YES* or *NO* questions.

Based on the ease of reading the decision tree, and only a 1% difference in accuracy, I would recommend using the decision tree for atheletes curious about what their training might lead to and how to plan a race.

For instance, if a runner were to have a training program put together where by the end of it, they knew they could maintain a 15 min/mile, while fluctuating 4 min/mile depending on the vert or descent of a particular section, they could accurately predict they would have an 80% of a solid finish.

However, if the training has been slacking and maintaining even a 19 min/mile is tough, it is better to stay as consistent with the slow pace as possible (57%).

Future Enhancements and Final Thoughts

Going forward, I think there is still a lot of work that can be done with this data and lead to further insights.

1. While this excercise above goes through a few different machine learning methods, I pretty much used the base tuning parameters, aside from putting each one through a 10-fold cross validation. Further exploration of the data would include analzying and understanding all the relationships between the

features even more to get a better understanding of their geometry and dimensionality. This would allow for finer tuning, and even an enhancment of the feature space too.

2. A few other things can be done with the data scraping aspect of the project. While I was able to extract what I deemed the most useful data, more feature engineering can take place with some more of the fields (e.g., `% of total aid stations`, `time spent at aid station`), along with capturing a few more of the fields I had to exclude due to length of time to data mine (e.g. `state` and `country`).

3. Another way to split and perform regression on the data would be to use clustering and understand the difference between the elites from the normal runners, or if they differ at all aside from their shear speed.

4. Further data to gather, not present in this dataset, would be the addition of elevation data along the course and training data. Both of these would be extremely helpful, and would take this model to a new step.
   By adding in elevation data, it can help to compare how pace changes with grade, and provide further insight. This would open the door even more by allowing a comparison of not only this single race, but being able to compare this race with others based on relative pace per grade. While this information is available, it would require a significant more amount of time to integrate, but I do hope to do so in the future.

5. Training data is probably the biggest determining factor for whether or not a runner will finish. Training data would give the ability to create a model and determine what sort of training is most apt for this race, and could even allow for a better understanding of ultra marathons as a whole. By creating a model that is inclusive of elevation gain and miles per week, coaches and runners could use training tools such as Strava and Training Peaks to really hone in their approach for a specific race.

6. Predict finishing time and place and anticipated arrival at aid stations.

Application in the Real World

What this model shows is the ability to focus in on specific variables and help a runner determine whether or not they have the tools to finish. But this information is sometimes only applicable during race day.

By combining this model with training information, the model would have much greater power as it now has a better piece of information that could be applied prior to the run, instead of during it. Online platforms such as Training Peaks and Strava offer training plans along with helping athletes track workouts and give metrics.

A few ways this project could integrate with their system:

1. Through their training plans they offer, a user could enter in their goal of finishing the Western States Endurance Run. This model would then be able to take their Age information, and then bsaed on speed and different splits and standard deviations over time, they would accumulate more and more data points to give indicators to how they would do in the race.

2. The trainng platforms could retroactively ask users that have participated in WSER to share their training data so as to incorporate that into building a better model.

3. Coaches (or those self-coached) could take and analyze their own data through whatever program works best for them, and compare their average paces over distance, along with standard deviation of different splits from different races, and get a gauge for how they might finish at WSER.

I hope to be able to continue the development of this project along with the accuracy and usefulness of this model.

Acknowledgements

My sincerest thanks goes out to Matt Fornito for helping me to refine this project and turn it into something meaninful and fun, and hopefully insightful to some. I know I have enjoyed it, and couldn't have gotten this far without his mentorship.

Secondly, I want to thank YOU reading this now. If you made this far (or just skipped to the bottom), you are reading this, and that is 1 more person than I thought would read this. Thank you.

Please don't hesitate to add commentary or thoughts on future improvements.