100 MILE TRAIL RACE FINISH TIME PREDICTOR

Russell Taylor
Student ID 001441098
C969 Computer Science Capstone
Western Governors University
December 10, 2020

CONTENTS

FIGURES

A1: LETTER OF TRANSMITTAL

Create a letter of transmittal (a single-page cover letter) to the client's senior leadership (non-technical background). This needs to be brief and concise addressing the following: summary of the problem, recommendation of the solution (data product and type), description of how the proposed solution benefits the client, objectives of the project, total funding requirement, and expertise of developer relevant to the solution. The letter of transmittal should include all elements typically found in artifacts of business communication, including a subject line, date, inside address, greeting, complimentary close, and signature.

Subject: **100 Mile Trail Race Finish Time Predictor**
Date: **December 10, 2020**

UltraSignup.com
1012 Taos Ranch Court
Reno, NV 89511

To the senior leadership of UltraSignup.com:

Summary of the Problem
UltraSignup.com has become not only the world's premier registration company for ultramarathons and trail races around the globe, the aggregated data it provides to both race directors and runners is an incredibly valuable feature. In particular, its Runner Ranking system allows for such useful features as the Target Finish Time assigned to each runner in a given race.

However, these features are clearly not as precise as they could be. The simple arithmetic used to calculate both the Runner Rank and Target Finish Time results in sometimes massive disparities between the Target Finish Time and a runner's actual finish time. While the existing formulas are an important first step, an improved model would greatly benefit both race directors and runners. Moreover, this data is highly sought after by ultra runners for training purposes, for improved chances of successfully completing the race, and for enhanced runner safety during the race.

Recommendation of the Solution
Greater precision in these calculated numbers and therefore greater value provided to both race directors and runners can readily be achieved through the use of Machine Learning. Machine Learning benefits from massive amounts of data, something UltraSignup.com already has at its disposal. That data, including actual finish times for prior races, can be used to generate a formula for predicting future finish times for individual runners.

Description of How the Proposed Solution Benefits the Client

The success of UltraSignup.com's simpler Runner Ranking and Target Finish Times are proof of the value of this concept. Improved Target Finish Time accuracy would benefit race directors and runners in 3 important ways:

1. Enhanced runner safety: Runners will be able to plan more carefully for unexpected circumstances during the race. For example, they will be less likely to find themselves alone on the trail at sundown without a flashlight.
2. Increased finish rates: Runners will have a reliable metric from which to establish a more realistic pace, especially in the early portions of the race.
3. Improved training plans: Runners will be able to tailor their training plans to more accurate expectations of what will transpire on race day.

Because this improvement in accuracy will provide immediate benefit to UltraSignup.com's clients, it will naturally lead to increased revenues.

Objectives of the Project

This project aims to produce a model for generating Target Finish Times for individual runners in a given race that have a high level of accuracy and a low mean absolute error.

Total Funding Requirement

This project, when fully built out, will still be rather straightforward to implement. It requires expertise in Machine Learning. But an initial version can be implemented in approximately 100 work-hours. A fully evaluated, more comprehensive version, trained on a larger dataset and taking into account a larger array of feature variables would take approximately another 100 work-hours. For a Software Engineer trained in Machine Learning making $50 per hour, total cost would be $5,000.00 for the prototype version and another $5,000.00 for the fully built-out version.

Expertise of the Developer Relevant to the Solution

The Software Engineer responsible for building this project has expertise in Machine Learning, Python, Jupiter Notebooks, Pandas, Numpy, Matplotlib, Scikit-learn, and basic web development skills needed to deploy the resulting product.

Thank you for your time and consideration.

Respectfully,
Russell Taylor

A2. NON-TECHNICAL PROJECT RECOMMENDATION

A project recommendation to convince those same senior, non-technical executives to implement the data product you are proposing to design to meet their business need.

1. PROBLEM SUMMARY

A summary of the problem that the application will solve. This should include a comprehensive description of the setting so it's clear how the solution meets the client's need(s). This is a scope statement complete with what will be included and incidental items that will not.

Ultramarathon trail racing is a sport that is rapidly growing in popularity. An article in the popular magazine Runner's World even asked the question "Is 100 miles the new marathon?" (https://www.runnersworld.com/racing/is-100-miles-the-new-marathon). Ultra Running Magazine's current race calendar lists 236 unique 100 mile races across the United States and Canada (https://calendar.ultrarunning.com) and the sport is equally popular outside North America. While any race longer than the marathon distance (26.2 miles) is considered an Ultramarathon, the most popular distances are 50 kilometers (31.07 miles), 50 miles, 100 kilometers (62.14), and 100 miles. Several events, such as the famous Moab 240 Endurance Run, feature even longer distances. But completing the 100 mile distance is widely regarded as the gold-standard for serious ultra runners.

Many of these races pride themselves on their extreme difficulty. Several feature more that 29,000 ft of elevation gain and loss (the equivalent of climbing from sea level to the top of Mount Everest and descending back to sea level) over the course of the race, many are run in extreme weather conditions, at very high altitude, and in remote locations with limited access to emergency support.

But the pristine beauty of the trails and mountains where many of these races take place, the challenge of overcoming a feat that seems all but impossible, and the thrill of crossing the finish line draws runners from all all around the world and all walks of life to this exciting sport.

Approximately 70% of ultramarathons use the website UltraSignup.com to provide convenient services for runners such as race signup, communications about race updates and logistics, and open access to official race results.

One of the most unique and powerful features of UltraSignup.com for both race directors and runners is the Runner Rank assigned to each runner on the basis of previous race results. UltraSignup.com details the formula used to calculate the Runner Rank in the Frequently Asked Questions section of the website:

"Runner rank is simply an average of your past results in our database. Rank is calculated using past results in our database. For each race, we take the gender specific best time

(winner) and divide that time by each participant's time. The result is a value less than 100% with winners receiving the full 100%. The average of the participant's past races is their ranking" (http://ultrasignup.com/faqs.aspx).

Each runner's Runner Rank is the basis for various other features provided by the website. One such feature is called the Target Finish Time which is assigned to each runner and included on the list of entrants for each race prior to the start of the race.

Currently, a runner's Target Finish Time is calculated by multiplying the runner's Runner Rank by the current course record for the given race. Since the Runner Rank is an average of the runner's results in any previous races expressed as a percentage relative to the winning times in those races, this formula seems rather consistent.

However, UltraSignup.com has openly and unapologetically acknowledged the rough nature of their calculations. In particular, a given runner may be assigned a relatively high Runner Rank for a given race if the winner of that race finished in a relatively slow time. The same runner may finish another race of the same distance with the exact same finish time, but if the second race features a field of elite runners, the Runner Rank will be relatively slow.

For example, if a runner named Suzy finishes a 100 mile race, Race A, in 30 hours, and the winner of the race finishes in 24 hours, Suzy's Runner Rank for that race will be 80% (24 hours divided by 30 hours). This result will then be averaged among all of Suzy's other race results. However, if Suzy instead finishes a different 100 mile race, Race B, in 30 hours, but this time world record holder Camille Herron is also in the field and finishes the race in 12 hours, Suzy's Runner Rank for that race will be 40% (12 hours divided by 30 hours).

The result of such disparities is that UltraSignup.com's Runner Rank must be regarded as nothing more than a very rough estimate of the runner's actual performance. Nevertheless, many runners place a significant amount of pride in their UltraSignup.com Runner Rank. There is even a common tongue-in-cheek discussion among ultra runners about whether they would choose to date a person with a higher or lower UltraSignup.com Runner Rank than their own.

This project does not propose to change anything about UltraSignup.com's Runner Rank system. Rather, the focus of this project is on one of the features derived from the Runner Rank system, namely each runner's Target Finish Time which is assigned to them after registering for a given race and prior to the start of each race.

While the lack of precision in Runner Ranks can be a fun topic of conversation, the lack of precision in Target Finish Times has the potential for some very serious consequences. These potential consequences are primarily positive in nature and contingent on the implementation of a more accurate means of calculation. Therefore, I am not suggesting that

the Target Finish Time predictions be eliminated, but rather that they be replaced by a more accurate predictive model.

2. APPLICATION BENEFITS

A description of how the application benefits the client and how the application will support a decision-making process in the context of the business need. Where does the product fit in filling an identified gap?

The popularity of UltraSignup.com's simpler Runner Ranking and Target Finish Times have served as a proof of concept demonstrating the value of performance analytics such as these to both race directors and runners.

Runners find value in such metrics because of the benefits they provide for targeted training, as well as for planning and preparation for overcoming unexpected obstacles on race day. Because of such benefits to runners, race directors find such features beneficial for both stirring up excitement about the race and for generating revenue from the race. As a service purchased by race directors on behalf of runners, UltraSignup.com sees a direct increase in both visibility and positive reputation in the running community, which results in increased revenues.

These facts have already been tested and demonstrated by the currently available features. However, as discussed above there exists significant room for improvement and along with it, significant potential upside for both UltraSignup.com and its users.

While increased precision of Runner Ranks themselves would add limited value and therefore Runner Ranks are sufficient as is, there are far greater potential benefits to providing more precise Target Finish Times.

Before discussing specific potential benefits, it should be noted, that precise race finish time predictions are in very high demand. Many precise mathematical models currently exist for road races at shorter distances. Among the most popular are:

- Runner's World's Race Finish Time Predictor (https://www.runnersworld.com/uk/training/a761681/rws-race-time-predictor)
- McMillan Running's Race Calculator (https://www.mcmillanrunning.com)
- Jack Daniels' VDOT Running Calculator (https://runsmartproject.com/calculator)
- SportTracksLabs Race Finish Time Predictor (https://sporttracks.mobi/labs/race-finish-time-predictor)

Demand among runners for tools such as these is extremely high. However, due to the extreme variations in race conditions and race difficulties among trail ultramarathons, such tools are severely lacking for these races. UltraSignup.com's own Target Finish Times are a

case in point. Most ultra runners do not take the current Target Finish Times seriously because they consistently prove to be wildly inaccurate, when compared to actual finish times.

The benefits of a more precise model are therefore clear:

1. High demand for **such features will generate revenue as long as there is a perceived benefit for runners**, regardless of whether there is any measurable benefit to runners. In a sport dominated by individuals with an unquenchable need for improvement and achievement, such metrics will remain in high demand.

Nevertheless, I contend that there actually are real benefits to improved metrics:

2. The most significant of these, especially from the perspective of race directors and regulatory agencies is **enhanced runner safety**. A 100 mile race in extreme conditions commonly lasts for 36 hours. Much can change during that time. As an anecdotal example, in a 100 mile race I finished last February in northern Washington state, I failed to account for the high latitude and early sunset on the second day of racing, leaving me without a flashlight and in the pitch dark at the very end of the race. Thankfully, I encountered some kind volunteers along the course who graciously let me borrow a flashlight. But a mistake such as this could easily have been the difference between crossing the finish line and a devastating (and preventable) DNF, or even worse.

3. Race prediction calculators for road races such as those listed above are often **used by runners to tailor and fine tune their training**. The specifics of training plans (how far, how fast, and how often to run during training) are often contingent on a runner's predicted pace. The benefits of such an approach are well documented and well understood. The same model can be, and often is, applied on an individual basis by runners in trail ultramarathons. But a tool that brings such metrics to a wider audience will allow many more runners to more train more deliberately.

4. A related benefit is the potential for **increased race finish rates and improved finish times**. The DNF (an acronym for "did not finish") rates in 100 mile ultramarathons are extremely high. In the 100 mile race mentioned above, the DNF rate was over 50%. One of the major causes of this is runners who begin the race far too fast. It is the norm to find among 100 mile race results a trend of runners beginning a 100 mile race at a 9-10 minute per mile pace and ending the same race at a 20-25 minute per mile pace. The far better racing strategy is to begin the race significantly slower than seems reasonable for the runner's fitness level. This almost always results in a faster finish time since during the later miles of the race the runner's pace is less likely to diminish to an incredibly slow slog. A precise prediction of a runner's finish time is a crucial metric both strategically and emotionally for a runner to exercise patience during the early miles of a 100 mile race.

3. APPLICATION DESCRIPTION

A description, using technical details, of the data solution application and how it aligns with the client's business priorities you're planning to solve.

The proposed solution to this problem is to use **Machine Learning** in order to more accurately predict runner finish times. Machine Learning is a subfield of Computer Science that creates models for predicting outcomes based on current data. Since UltraSignup.com has abundant data at its disposal, this approach is especially well suited to fit UltraSignup.com's business needs.

A traditional computer program is given input data, then the programmer writes a computer function that processes the input data, and the computer then provides output data. Machine Learning is different. With Machine Learning the computer is given input data and the expected output data, then the computer produces an algorithm that transforms the given input data into output data. The produced algorithm can then be used to predict outputs for any given input data.

In order to solve this problem, a Machine Learning model will be trained using a sample dataset taken from UltraSignup.com's existing database, namely race results going back 5 years for several 100 mile races. This dataset will be split into two parts: **training data** and **test data**. The training data will be used to train the Machine Learning model. The test data will be used to test the Machine Learning model and make any needed changes to optimize it so that it will produce more accurate results. Once the Machine Learning model is optimized, it can be used to predict a runner's finish time when given any appropriate input data.

The following data taken from UltraSignup.com's existing database will be used to train the Machine Learning model:

1. The race cutoff time (the amount of time given to runners to complete the race)
2. The location where the runner lives
3. The runner's age
4. The runner's gender
5. The runner's UltraSignup.com Rank
6. The runner's finish time for the given race

The runner's finish time will be removed from the test data set, and the Machine Learning model will then predicts each runner's finish time based on the first five features.

Each of these features are important for various reasons, but some are more impactful than others. The race cutoff time is necessary in order to account for runner finish times that vary widely because some races have a cutoff time of 30 hours or less, while others because of their extreme difficulty have a cutoff time of 48 hours. The most important feature for

prediction is the runner's Ultrasignip.com rank since it takes into account the runner's performance in all previous races in the entire Ultrasignup.com database.

Once this model is trained, tested, and optimized UltraSignup.com users will be able to input a race (with a specified cutoff time), their location, age, gender, and UltraSignup.com Rank, and the model will provide an accurate prediction of the runner's finish time for that race.

4. DATA DESCRIPTION

A description of the data that will be used to construct the data product including its type (nominal, real numbers, etc...), origin, and structure. Identify the independent and dependent variables. Comment on the anomalies of the dataset and the limitations of the data collected.

Ultrasignup.com has made race results publicly available for dozens of 100 mile races with data going back as far as 1974 (the Western States Endurance Run is the oldest 100 mile race in the United States and the results for every year it has been run can be found here: https://ultrasignup.com/register.aspx?did=41765). This project is only a prototype Machine Learning model for predicting race results, and therefore will not make use of the entire UltraSignup.com database. Rather, a smaller dataset will be collected from UltraSignup.com which includes official race results from 16 of the largest and most popular 100 mile races across the United States. The dataset will include the most recent 5 years of results for each race. This will total to 80 individual events and over 12,000 runner results. The included races are listed below:

1. Angeles Crest 100 Mile Endurance Run
   Wrightwood, CA (https://ultrasignup.com/register.aspx?did=79033)
   2019 (155 finishers)
   2018 (100 finishers)
   2017 (115 finishers)
   2016 (130 finishers)
   2015 (98 finishers)

2. The Bear 100 Endurance Run
   Logan, UT (https://ultrasignup.com/register.aspx?did=79739)
   2020 (207 finishers)
   2019 (204 finishers)
   2018 (221 finishers)
   2017 (222 finishers)
   2016 (166 finishers)

3. Cascade Crest 100 Mile Endurance Run
   Easton, WA (https://ultrasignup.com/register.aspx?did=74710)
   2019 (128 finishers)
   2018 (147 finishers)

2017 (110 finishers)
2016 (127 finishers)
2015 (99 finishers)

4. Cruel Jewel 100
   Blairsville, GA (https://ultrasignup.com/register.aspx?did=79152)
   2019 (136 finishers)
   2018 (100 finishers)
   2017 (77 finishers)
   2016 (87 finishers)
   2015 (48 finishers)

5. HardRock 100 Endurance Run
   Silverton, CO (https://ultrasignup.com/register.aspx?did=79879)
   2018 (114 finishers)
   2017 (126 finishers)
   2016 (112 finishers)
   2015 (126 finishers)
   2014 (100 finishers)

6. HURT 100
   Honolulu, HI (https://ultrasignup.com/register.aspx?did=75477)
   2020 (63 finishers)
   2019 (69 finishers)
   2018 (78 finishers)
   2017 (54 finishers)
   2016 (53 finishers)

7. IMTUF (Idaho Mountain Trail Ultra Festival) 100
   McCall, ID (https://ultrasignup.com/register.aspx?did=74620)
   2020 (90 finishers)
   2019 (119 finishers)
   2018 (125 finishers)
   2017 (86 finishers)
   2016 (79 finishers)

8. The Javelina Jundred
   Fountain Hills, AZ (https://ultrasignup.com/register.aspx?did=74613)
   2020 (160 finishers)
   2019 (425 finishers)
   2018 (368 finishers)
   2017 (348 finishers)
   2016 (285 finishers)

9. Kettle Moraine 100 Endurance Trail Races
   LaGrange, WI (https://ultrasignup.com/register.aspx?did=70464)
   2019 (166 finishers)
   2018 (162 finishers)
   2016 (133 finishers)
   2015 (164 finishers)
   2014 (finishers)

10. The Old Dominion 100 Mile Cross Country Run
    Fort Valley, VA (https://ultrasignup.com/register.aspx?did=80893)
    2019 (60 finishers)
    2018 (55 finishers)
    2016 (31 finishers)
    2013 (finishers)
    2012 (finishers)

11. Orcas Island 100 Miler
    Olga, WA (https://ultrasignup.com/register.aspx?did=65809)
    2020 (52 finishers)
    2018 (69 finishers)
    2017 (45 finishers)
    2016 (49 finishers)

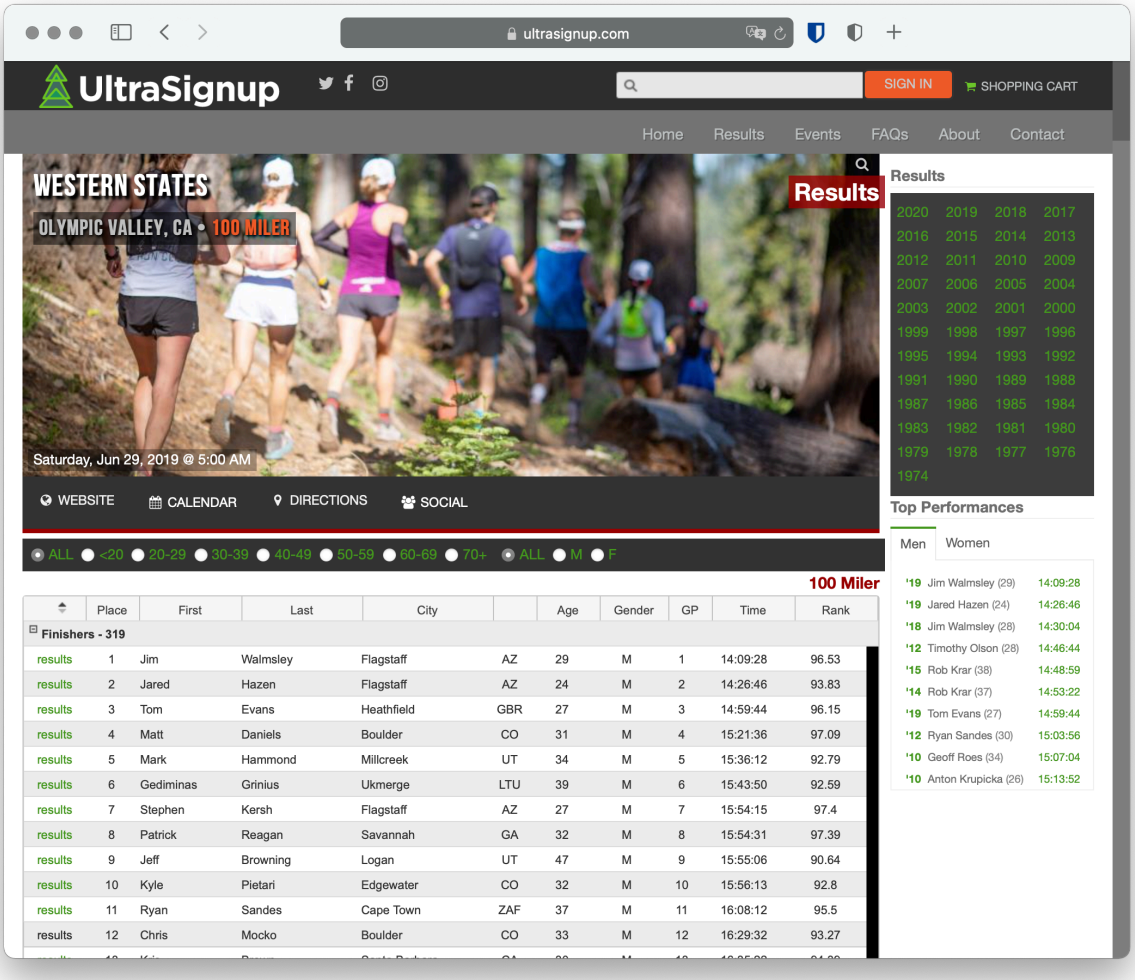
12. Run Rabbit Run 100 Mile Race
    Steamboat Springs, CO (https://ultrasignup.com/register.aspx?did=79897)
    2019 (178 'tortoise' finishers, 40 'hare' finishers)
    2018 (167 'tortoise' finishers, 40 'hare' finishers)
    2017 (145 'tortoise' finishers, 38 'hare' finishers)
    2016 (203 finishers)
    2015 (153 finishers)

13. Tunnel Hill 100
    Vienna, IL (https://ultrasignup.com/register.aspx?did=74148)
    2020 (36 finishers)
    2019 (225 finishers)
    2018 (203 finishers)
    2017 (181 finishers)
    2016 (101 finishers)

14. Wasatch Front 100 Mile Endurance Run
    Kaysville, UT (https://ultrasignup.com/register.aspx?did=79429)
    2019 (208 finishers)

2018 (187 finishers)
2016 (213 finishers)
2015 (203 finishers)
2014 (finishers)

Figure 1: UltraSignup.com Website Results Page for the 2019 Western States 100 Mile Endurance Run



15. Western States 100 Mile Endurance Run
    Olympic Valley, CA (https://ultrasignup.com/register.aspx?did=79446)
    2019 (319 finishers)
    2018 (299 finishers)
    2017 (248 finishers)
    2016 (279 finishers)
    2015 (254 finishers)

16. The Yeti 100 Mile Endurance Run
    Abingdon, VA (https://ultrasignup.com/register.aspx?did=79658)
    2020 (145 finishers)
    2019 (162 finishers)
    2018 (179 finishers)
    2017 (136 finishers)
    2016 (82 finishers)

All 16 races have a distance of approximately 100 miles. These specific races were chosen with consideration of the popularity of the race, the historical significance of the race, geographical distribution, and the availability of race results on UltraSignup.com. Western States is the oldest 100 mile race in the United States and widely regarded as the most prestigious. It maintains a list of qualifying races (runners must complete one of the qualifying races in order to register for Western States) which consists of the largest 100 mile races in each of five different regions across the United States (https://www.wser.org/qualifying-races). The above races were selected from the list of Western States qualifying races. Among them are races from 11 different US states. The list also includes both extremely challenging mountain races such as HardRock and flat, fast races such as Tunnel Hill where multiple world records have been set. The cutoff times for these races range from 30 hours to 48 hours depending on the difficulty of the race.

The most recent 5 years of available and complete results will be included for each race. Nine of the sixteen races were cancelled in 2020. Two were cancelled in 2019 due to inclement weather (HardRock and Orcas Island). Three races had incomplete data for certain years (Kettle Moraine 2017, Old Dominion 2014, 2015, 2017, Wasatch Front 2017). And one race has only 4 years worth of data available (Orcas Island).

The UltraSignup.com results for each race include the following fields: Race Name, Year, Place, First Name, Last Name, City, Location, Age, Gender, Gender Place, Finish Time, and UltraSignup.com Rank. Of these fields, only four will be used as input for the Machine Learning model, namely Location, Age, Gender, and UltraSignup.com Rank. Each of these fields is **dependent** on the runner, and **independent** of the race, and will therefore allow the model to be used to predict finish times for any 100 mile race, rather than requiring a specific race as input. This will be important in order to predict finish times for runners running new races in their first year. However, because race finish times are directly dependent on a race's cutoff time, the official cutoff time for each race will be added to the UltraSignup.com data as a fifth input field for the Machine Learning model. Place and Gender Place are too specific to be used as input information for the Machine Learning model. A runner's Finish Time will be the Machine Learning model's output.

The UltraSignup.com data is **structured** as simple spreadsheets, and will be imported into the Machine Learning model from csv (comma separated values) files.

Race results are reported to UltraSignup.com by race directors and are generally carefully checked for consistency and accuracy. The overall quality of the data is very high. However, several **anomalies** do currently exist in the data. One important factor will be the exclusion of results for runners who either did not finish (DNF) the race or did not start (DNS) the race. Since the Machine Learning model will be focused on predicting finish times rather than whether or not a runner will finish, both DNFs and DNSs will simply be excluded.

Additionally, the Location field is used for either the runner's home state or home country for international runners. However, some races simply leave this field blank for international runners. A few races exclude this field altogether. Where this data is blank for only specific runners, the designation INTL will be used instead.

Finally, nine anomalous finish times have been discovered in the data. In each case, a runner's result shows a finish time of less than 10 hours (the current world record stands at 11:19:13 for men and 12:42:39 for women). These results will be removed from the data used by the Machine Learning model.

5. OBJECTIVE AND HYPOTHESIS

The objectives represent the primary desired outcomes for the project and application. State what you want the application to achieve in relation to the goals. The hypotheses should be structured as a proposed explanation for a phenomenon(s) based on what the application will produce. State the "if" condition and the "then" outcome which is the prediction of general outcomes. You'll describe the validity of your hypothesis in Prompt D.

The primary **objective** of this project is simple and clear: to produce more accurate race finish time predictions for 100 mile trail races. Such predictions will allow for more precise training plans, better race day preparation, and enhanced on-course safety for runners.

More specifically, the aim of this project is to provide race finish time predictions that are accurate on average to within 1 hour of the actual finish time for a given runner. This goal seems reasonable considering that for a runner who finishes HardRock in 48 hours, a prediction that is off by 1 hour is still 97.9% accurate. For runners with a finish time near world record pace at 12 hours, a 1 hour discrepancy still amounts to 91.7% accuracy.

The working **hypothesis** for this project is that if a Machine Learning model is provided with a runner's location, age, gender, UltraSignup.com rank, and the cutoff time for the chosen race, then the model can predict the finish time for that runner to within 1 hour of the runner's actual finish time on average.

This aim of the current project is **limited** to providing a first generation prototype Machine Learning model that can predict runner finish times with greater accuracy that UltraSignup.com's current tool. A later generation model that makes use of UltraSignup.com's

entire database and uses a more complete set of input data fields will be needed in order to produce a fully functioning race predictor suitable for public use.

6. METHODOLOGY

An outline of the project methodology. Choose an industry standard methodology (e.g. ADDIE, Agile, or SDLC) that you'll use to manage your project. Describe why that methodology is appropriate and then indicate what parts of the project will align with the methodology phases.

This project will follow a standard Waterfall Software Development Life Cycle (SDLC) methodology. Since it is a relatively small and straightforward project, team-oriented methodologies designed for more complex projects such as Agile methodologies are not ideal.

The SDLC consists of the following steps:

1. **Feasibility analysis**. This is where the problem and the proposed solution are analyzed to determine the cost benefit ratio between the investment of time and resources and the likelihood of producing a viable product. This document aims to assert that the proposed solution is indeed feasible.

2. **Requirement specification**. This is where the stakeholders' requirements for the final solution are determined. Review of this document will facilitate and initiate discussions between the stakeholders and developers. A final requirement specification document will be signed prior to commencement of the project.

3. **Design**. This is where the proposed solution is designed including the details of how the solution will be implemented. The specific tools and technologies are chosen. The exact steps in the development process specific to this application are determined.

4. **Development**. This is where the solution is created by coding the proposed application. For this project, development will include data collection, data cleaning, and model creation and training.

5. **Testing**. This is where the Machine Learning model will be evaluated using carefully selected and appropriate metrics

6. **Deployment**. This is where the final product will be delivered to the customer and made available for use as intended. Because the current project is a prototype this step will consist of delivering the working product to the technical team at UltraSignup.com.

7. **Maintenance**. As new data becomes available as races are added to UltraSignup.com's database, that data will reflect changes in the sport of ultra running as well as

circumstances such as modifications to races due to global health restrictions in 2020. The current project will therefore need to be continually updated with the latest data and the Machine Learning model tuned to that data in order to provide the best results.

Because this project is small and straightforward, it is reasonable to approach each of these steps in a simple sequential order, namely using the Waterfall method. However, projects rarely follow a sequential flow, it is difficult to state all requirements explicitly in advance, and unexpected modifications are inevitable.

However, much of the potential risk posed by Boehm's first law ("Errors are most frequent during the requirements and design activities and are the more expensive the later they are removed.") can be mitigated through careful and thorough up front planning. In the case of this project, it was decided as a result of such planning that a limited prototype that uses a subset of UltraSignup.com's data and a limited number of input fields is a more feasible first version of this project rather than a final full-featured version. The limited scope of this project will therefore contribute to its success.

Nevertheless, some elements of iterative and agile methodologies will naturally be incorporated into the production of this application. In particular, it will be created using a test-first design methodology. Test Driven Development (TDD) makes test specifications the critical design activity, clearly defines what success means, and writes code with the specific aim of passing the tests. Such an approach clearly deviates from the strict waterfall methodology which technically places testing after development. However, in practice, very few software applications are created without some deviation from the waterfall method where appropriate.

7. FUNDING REQUIREMENTS

The project's funding requirements. Taking into consideration environment, personnel, and any licensing or programming tools required what would be the overall cost involved. Be sure this amount matches the letter of transmittal.

This project, when fully built out, will still be rather straightforward to implement. It requires that the developer or developers have expertise in Machine Learning, but the initial prototype version can be implemented in approximately **100 work-hours**.

A full-featured version of the application, trained on a comprehensive dataset and accounting for a larger array of feature variables would require an additional 100 work-hours.

This project can be built on any existing computer and the specific tools needed to build the the application (Anaconda, Jupyter Notebooks, Pandas, NumPy, Matplotlib, and Scikit-Learn) are all available for free.

For a Software Engineer trained in Machine Learning making **$50 per hour**, the total cost of this project will be **$5,000.00 for the prototype version** and another $5,000.00 for the fully built-out version.

8. STAKEHOLDERS IMPACT

Stakeholders are individuals or organizations with a vested interest in the project including being affected as a result of project execution or project completion. They may also influence the project's objectives and outcomes. Stakeholders may be internal, or external (e.g. customers), or members of a community. Describe the impact of your solution might have on the project stakeholders.

The stakeholders for this project include four distinct groups:

1. **Runners**. The primary goal of this project is aimed at the needs of runners. Improved accuracy in race finish time predictions will enable targeted training plans that prepare runners with maximum efficiency for the precise pacing and circumstances they will face on race day. Improved preparation and more accurate expectations about where the runner will be at any given time during the day will aid in making sure that drop bags, supplies, and meetups with crew members are where they need to be when the runner needs them. Accurate race finish time predictions will also encourage more even pacing throughout the race and will lead to improved finish times and higher finish rates. The added awareness and more reliable expectations about pacing during the race will also enhance the runner's safety and prevent situations where runners are caught without the necessary supplies.

2. **Race Directors**. Race directors work to make race day as smooth, enjoyable, and successful as possible for runners. Therefore the race director's interests are aligned with the runners' interests. So when runners are better prepared and better equipped with accurate expectations on race day, they are more likely to provide positive feedback and encourage others to run the race in the future. But perhaps the most significant benefit to race directors is the potential for enhanced runner safety. While all runners sign legal liability release forms prior to engaging in events such as these with a high potential risk, race directors still face liability in other ways, including any potential legal fees, the race's reputation in the running community, a sense of moral responsibility for the runners' wellbeing, and potential denial of permits, licenses, and approvals from regulatory agencies in the future.

3. **Regulatory Agencies**. Races such as these are often run on public lands and require permits, licenses, and/or approval from the relevant regulatory agencies. Any potential loss of life during such an event will reflect poorly on the agencies that manage the land and grant permits for the events to take place. Moreover, if a runner were to become lost, the cost of a search-and-rescue operation can be significant. Therefore, any additional

assurances that a race director can provide regarding runner safety will be welcomed by such agencies.

4. **UltraSignup.com** and other similar companies provide services to race directors and runners including race registration, payments, communication of race updates and information, live tracking of runners during the race, and official race results listings. UltraSignup.com has successfully distinguished itself from its competitors primarily because of its unique Runner Rank feature. It also offers related features such as Target Finish Time as part of each race's entrants list. But this feature is notoriously inaccurate and can be improved. Improvement of this feature will provide immediate value to both runners and race directors, and will therefore increase revenues for UltraSignup.com or any such service that implements the results of this project.

9. DATA PRECAUTIONS

Identify the data from the dataset(s) that are sensitive and/or protected. Then review the general guidelines related to working with that data. Discuss the general guidelines related to working with the sensitive data. Precautions may include data preparation, data analysis, data storage, access to data, and data dissemination. Explain the relevance between the precautions and application treatment of said data. If these considerations do not apply to the project and the data you are using, review the ethical and legal precautions commonly used when working with sensitive data. Obvious situations would be health care (HIPAA), education (FERPA), or payment (PCI DSS).

This project will use only data that is already freely available to the public on UltraSignup.com. As part of the race registration process, participants in races are required to provide written consent to the public dissemination of this data. As such, **no proprietary, personal, or sensitive data will be used** in this project except as has been explicitly authorized by the runners whose names appear in the data. For these reasons, this project will pose no legal liability to UltraSignup.com beyond any liability already incurred since this data is publicly posted on their website. Moreover, it should be noted that UltraSignup.com makes no claims as to the exclusive use of this data for their own purposes. It is thus freely available to be used in projects such as the present one, whether or not UltraSignup.com ultimately decides to incorporate this project into their product offerings.

10. DEVELOPER'S EXPERTISE

Provide information (e.g. academic training and professional expertise) about the software developer and how he/she is qualified to complete the project in a timely fashion. Be sure to relate it to the solution you're proposing. This information may be real or hypothetical to fit the project topic and scenario you have created.

As the primary software developer heading up this project, I am uniquely qualified for a project such as this. I have a **Bachelor of Science in Computer Science** including specific

experience and training in both **Machine Learning** as well as the tools that will be used to implement this project.

Moreover, I have **worked with large, complex sets of data** in the past as an academic researcher specializing in Ancient Middle Eastern Languages and contract developer for Oaktree Software, Inc. As part of that work, I created and maintained complex, highly-specialized right-to-left language datasets optimized for natural language processing. Those datasets enabled comprehensive syntactic tagging of ancient texts, and led to several new discoveries about ancient middle eastern linguistics, history, and culture.

Finally, as **an ultra runner** who has completed many races including the 100 mile distance, I will bring domain expertise to this project. I have first hand knowledge of the types of features and data runners want, need, and are willing to pay for. And I will build this project from the ground up with that knowledge and experience at its core.

B. TECHNICAL PROJECT RECOMMENDATION

Write a proposal for the technically savvy IT professional leadership of your client. This will contain industry appropriate jargon and sufficient technical details to describe the proposed project and resulting application. Remember, you're establishing the technical context for your project and what it will accomplish for the client.

1. PROBLEM STATEMENT

Describe the problem or opportunity you are solving. This could relate to the need for accurate analysis or predictive representation of the data.

The UltraSignup.com website backend calculates race finish time predictions for runners who are registered for 100 mile trail races. However, the current formula used by UltraSignup.com to calculating these predictions is based on simple arithmetic averages, and produces results that are notoriously inaccurate. This project will use Machine Learning to generate finish time predictions with a Mean Absolute Error of less than 1 hour.

2. CUSTOMER SUMMARY

Provide a description of the customers or users of the application and how this product will fulfill their needs. What is the environment where the application will be used and what special skill sets might be necessary?

The eventual end-users of this application are runners who would like an accurate prediction of their finish time for a given 100 mile trail ultramarathon. A final, fully built-out version of this application will be integrated into the UltraSignup.com website's backend and will replace the Target Finish Time feature displayed on each Race Entrants webpage. As a component of UltraSignup.com's backend, the feature would require no special environment setup, software, or skills on the part of the end user except for basic ability to use a web browser or mobile app.

However, the current project aims to create a first version of the race finish time predictor. It will be a prototype Machine Learning model **designed to be used by UltraSignup.com's technical team** for evaluation purposes. As such its user interface will require a basic knowledge of Anaconda and ability to open and run a simple Jupyter Notebook. Anaconda is freely available for most operating systems, including Linux, MacOS, and Windows.

3. EXISTING SYSTEM ANALYSIS

Summarize the technology environment prior to project initiation and the desired state of the environment upon project completion. This is a systems analysis for the project deliverables. This might also include a review of the existing functional gaps in the data products you are replacing.

UltraSignup.com's current Target Finish Time is calculated based on each runner's UltraSignup.com Runner Rank and the Runner Rank of each other runner registered for the given race.

An UltraSignup.com Runner Rank is a calculated average of a runner's past race results in the UltraSignup.com database. For each past race, a unique Rank for that particular race is assigned to the runner. For the given race, a runner's Rank is calculated by taking the finish time of the winner of the race and dividing it by the runner's time. The result is a value expressed as a percentage. If the runner is the winner of the race, the value is 100%. Otherwise the runner's percentage is some value less than 100%. Because each race has a male and a female winner, the Rank for runners who identify themselves as male and female are calculated separately based on the overall male and female winners.

$$\textbf{RaceRank (\%)} = \textbf{(WinnerTime / RunnerTime)} \times \textbf{100}$$

The overall Runner Rank for each runner is then calculated by taking the average Rank assigned to that runner for each of the past races they have run which are present in the UltraSignup.com database.

$$\textbf{OverallRank} = \left(\sum_{i=1}^{n} \textbf{RaceRank}(i) \right) / n, \text{ where } n = \textbf{NumFinishes}$$

An UltraSignup.com Target Finish Time is calculated by multiplying the runner's overall average Runner Rank by the current course record finish time for the given race.

$$\textbf{TargetFinish} = \textbf{OverallRank} \times \textbf{CourseRecord}$$

These calculations provide a very rough estimate of a runner's predicted finish time for a given race. However, both the Runner Rank calculations and the Target Finish Time calculations are heavily dependent on the finish time of the race winner. This results in a sizable **disparity** between races where the winner is an average runner versus a race that has serval elite runners in the field. Both scenarios occur regularly in trail ultramarathons. If a runner has only run smaller races in the past with zero elite runners, that runner may have a very high Runner Rank. If that runner then signs up for a race featuring several elite runners, their very high Runner Rank will be multiplied by the very fast course record for the new race, and the resulting predicted finish time may be many hours faster than the runner is actually capable of running.

For the reasons outlined above, the existing system has garnered a reputation among runners as being notoriously inaccurate and unreliable. The current system makes no attempt to use more advanced technologies such as Machine Learning and instead relies on simple arithmetic.

This project proposes to fill that gap. Upon completion of this project, it will be possible to input a runner's location, gender, age, Runner Rank, and the cutoff time for the race, and to receive as output a predicted finish time with a Mean Absolute Error of less than 1 hour.

4. DATA

Expand on the nature of the data you're using including: how it is to be collected, what techniques will be required to make the data usable, how you'll deal with undesirable exceptions or data anomalies (outliers, incomplete data).

The data that will be used in this project is freely available in json format on UltraSignup.com's website. For an eventual, fully built-out version of this application it will be desirable to include in the dataset all available race results in UltraSignup.com's database. In order to do so, the developers will need to access the database using UltraSignup.com's DBMS.

Without such access, the data could be obtained by writing a Python script to scrape the entire UltraSignup.com website and import data from json files. However, UltraSignup.com does have in place web crawler tracking software in place to prevent too many automated requests. Therefore, more complex methods would have to be employed in order to access the entire database, such as using VPNs and scraping the website in smaller chunks at a time from multiple IP addresses. Such methods could take a significant amount of time, resources, and effort, and may have unintended ethical or legal consequences.

No such techniques will be necessary for this project since it represents only an initial prototype version of the desired race prediction functionality. For that reason, it will be sufficient to use a subset of UltraSignup.com's database. Because **the data is easily accessible in HTML5 format** using standard http protocols in a web browser, it will take a developer no longer than an hour to either copy and paste the race results data into a spreadsheet or else automate the process by **reformatting the data using a simple script** or RegEx find-and-replace commands.

The **outcome** of that process will be a set of **csv files** which contain individual runner results for multiple races. Each included race will have its own spreadsheet, and each spreadsheet will contain results for all included years of that race. Each row of the spreadsheet will include the race result for a single runner. Each column will include a field of data for that runner. The included fields will be: Race, Year, Cutoff, Place, First, Last, City, Location, Age, Gender, GP, Time, and Rank.

A few of these fields must be **added** to the dataset. The Race, Year, and Cutoff fields are not currently present in the UltraSignup.com data. Rather, these pieces of data are listed at the top of the webpage for each race. They will, however, need to be added **as data fields in the csv files** since results from multiple races and race years will be combined into a single dataset in the csv files and in the Pandas DataFrames in later stages of this project. This can easily be accomplished by importing the data into DBMS software such as **MySQL**

Workbench, adding fields to the tables, and then exporting the result as csv files. Or, because the prototype dataset is a manageable size, it will be possible to manually add columns to the spreadsheet using Microsoft Excel or Apple's Numbers and fill them with the appropriate race data as each csv is being manually created. In either case, care must be taken to ensure that these new data fields are input accurately and that no anomalies are introduced into the data as a result of this process.

Once the data is consolidated into csv files, it will then be **imported** into a **Pandas DataFrame**. Pandas will be used to further **parse and clean the data**. Three types of anomalies exist in the data and will need to be addressed:

Records which Contain Missing or Erroneous Data
The first step in parsing and cleaning the data will involve excluding individual runner results that either do not contain a finish time or contain erroneous finish times. Because the runner's finish time is the value that the Machine Learning model will predict, accurate data in this field is essential and any records containing inaccuracies must be excluded.

The first anomaly to address involves results that do not contain a finish time. Many races include records for **runners who either did not start (DNS) or did not finish (DNF) the race**. These results may be excluded either from the csv files, or after the data is imported into a Pandas DataFrame. We will choose to exclude them using Pandas.

The second anomaly requiring that individual records be excluded involves finish times that are **outliers** and clearly reflect inaccurate reporting of data. Finish **times that are lower than the current world record** times for men (11:19:13) and women (12:42:39) respectively will be excluded. Finish **times that are higher that the official cutoff time of the race**, and not part of a cluster of results reflecting a change in policy for a given year, will also be excluded.

Fields which Contain Missing, Erroneous, or Extraneous Data
The second step in parsing and cleaning the data will involve handling **fields** with missing, erroneous, or extraneous data.

First, the City and Location fields warrant special attention. For some races, these fields are **missing entirely**. Such races have been excluded from the dataset for this project. Other races have missing data in these fields only where a runner is **from a country outside the United States**. Because the City field will not be used in the Machine Learning model for this project, there is no need to alter null City values. However, the Location field will be used as a feature variable and therefore must be normalized. Since null fields in the data are present only where a runner is from an international location, it will be sufficient to simply **fill** any null fields with the designation 'INTL'. A more precise approach in future versions of this application will be to identify the runner's country based on the city data provided and fill the null fields with appropriate data. But no attempt will be made to do so in the initial version.

Second, multiple fields in the UltraSignup.com data are extraneous and will need to be excluded from the Machine Learning model's training and test datasets. The most significant among these are fields which contain specific data that will result in **overtraining the Machine Learning model**. The runner's First name, Last name, and City will be excluded for this reason, as well as the runner's final Place in the given race and the runner's gender-specific place in the given race (GP). The latter two fields are clear examples of how overtraining a Machine Learning model is detrimental to the model's effectiveness. If the model is trained on data that contains the runner's Place, that model will not be able to accurately predict a runner's finish time where this information is lacking.

For the same reason, the Race and Year fields will also be excluded. This will allow the model to remain abstract enough to predict a runner's finish time in **any** 100 mile trail race where only the runner's Age, Gender, Location, and Rank, and the race's Cutoff are provided as inputs. The result will be a more versatile and useful model for predicting runner results.

Incompatible Data Types
Incompatible data types exist in the current data in two forms.

First, the runner finish times will be stored and imported as strings. Each value must therefore be converted using Pandas from a **string** to a **Timedelta**. Then, because the Machine Learning models we will use require numerical inputs, it will be necessary to convert each **Timedelta** to an **integer** representing the number of seconds it took for the runner to complete the race.

Second, **categorical fields** must also be converted to a **numerical** data type. This can be done in two ways. It can be achieved using the Pandas `Categorical()` method and then isolating the `cat.codes` as a field of integer values. Or, it can be achieved using the Pandas `OneHotEncoder()` and `ColumnTransformer()` methods. This method produces sparse data by creating a separate field for each category and assigning a boolean value to each record. Either approach will work for our predictive model. But only the former will be suitable for our descriptive model.

The result of the parsing and cleaning described above will be a normalized dataset containing only numerical datatypes which is suitable as input for our Machine Learning models.

5. PROJECT METHODOLOGY

Elaborate on your industry standard methodology by providing specific details about what aspects your project will be managed by each of the methodology phases. How will the methodology guide and support the data product design and development? Include levels of testing (e.g. unit, integration, system, and acceptance) as part of the methodology.

The Waterfall methodology used in this project is straightforward and well understood.

1. **Feasibility analysis**. This document provides a detailed analysis of the cost-benefit ratio between the investment of time and resources and the likelihood of producing a viable product. The details of this analysis are presented here in the form of a detailed plan, budget, and timeline.

2. **Requirement specification**. This document outlines the specific problem to be solved and a detailed proposal of how that problem will be solved. The technical team and senior leadership at UltraSignup.com will review this document and a final requirement specification document will be signed prior to commencement of the project.

3. **Design**. This document outlines the details of the proposed product design including the programming environment, the specific technologies to be used for development and deployment, and a detailed outline of the product code.

4. **Development**. The solution will be created by coding the proposed application and will include data collection, data cleaning, model creation and training, and model evaluation.

   Data collection for the prototype version of this application will include gathering data from the UltraSignup.com website. Data will be collected from the HTML5 webpages themselves or from the underlying json files using a Python script. This data will be stored in csv files and imported into the Jupyter Notebook as a Pandas DataFrame.

   Data cleaning will take place in the Jupyter Notebook by using the tools offered by Pandas to manipulate the data, including filling null fields, deleting rows and columns containing erroneous data, and creating additional fields as needed.

   Model creation and training will be carried out using Scikit-Learn. Both descriptive Randomized PCA and non-descriptive Random Forest Regressor algorithms will be trained and tested on the input data. The Randomized PCA algorithm will reduce the number of features needed for the predictive Random Forest Regressor algorithm and will therefore improve both the efficiency and accuracy of the results.

   Model evaluation will assess the accuracy of the Machine Learning models using the given input data and the chosen features. The regression algorithm will be evaluated using the R^2 coefficient of determination regression score function as well as the algorithm's mean absolute error.

5. **Testing**. This is where the Machine Learning model will be evaluated using carefully selected and appropriate metrics.

   Unit testing is the first level of testing and will be performed by the developer. Each unit of the Jupyter Notebook will be run in isolation from the other units to ensure that it is functioning properly. Because the testing is limited to a particular unit, exhaustive testing

will be possible. This will allow errors to be detected at an early stage which will save time and money.

Integration testing is the second level of testing in which individual modules in the Jupyter Notebook will be run in conjunction with one another with the aim of detecting any interfacing issues between the modules. Big bang integration testing will be used for this project.

System testing is the third level of testing where the complete project will be tested as a whole. The focus will be on whether the application meets the business requirements laid out in the requirement specification document. This testing will be carried out in an environment similar to the production environment.

Acceptance testing is the final level of testing and will be carried out by UltraSignup.com's technical team once the product is delivered. Since the product will be a prototype, the entire lifecycle of the first phase product will effectively be an alpha test (https://artoftesting.com/levels-of-software-testing).

6. **Deployment**. This is where the final product will be delivered to the customer and made available for use as intended. Because the current project is a prototype this step will consist of delivering the working product to the technical team at UltraSignup.com.

7. **Maintenance**. As new data becomes available as races are added to UltraSignup.com's database, that data will reflect changes in the sport of ultra running as well as circumstances such as modifications to races due to global health restrictions in 2020. The current project will therefore need to be continually updated with the latest data and the Machine Learning model tuned to that data in order to provide the best results.

6. PROJECT OUTCOMES

Describe the deliverables associated with the design and development of the application. List and describe those deliverables. Also, include examples to help clarify what specific type of artifacts will qualify.

The deliverables associated with the design and development of this application will include both Project Deliverables and Product Deliverables. The **Project Deliverables** will include:

Letter of Transmittal to the client's senior leadership.

Non-Technical Project Recommendation containing a problem summary, application benefits, application description, data description, objective and hypotheses, methodology, funding requirements, stakeholders impact, data precautions, and developer's expertise.

Technical Project Recommendation containing a problem statement, customer summary, existing system analysis, data, project methodology, project outcomes, implementation plan, evaluation plan, resources and costs, and timeline and milestones.

Post-Implementation Report containing the project purpose, datasets, data product code, hypothesis verification, effective visualizations and reporting, accuracy analysis, application testing, application files, user's guide, and summation of learning experience.

The **Product Deliverables** will include:

Jupyter Notebook which will serve as this project's self-contained, functional **user interface** and includes interactive features that allow the user to access, manipulate, and display the project's descriptive and non-descriptive **data methods**, **datasets**, **analytics**, **graphical visualizations**, and **source code**.

Input data in the form of multiple csv files.

Log files for reporting and debugging purposes.

7. IMPLEMENTATION PLANS

Explain how the project will be implemented. This has to do with how the software application will be put into the production environment, not how it will be created.

Strategy for Implementation

This project aims to produce a working prototype Machine Learning model that will predict runner finish times in 100 mile trail races to within 1 hour of the runner's actual finish time on average. As a prototype, its end user is the technical team at UltraSignup.com. Therefore, the strategy for implementation is relatively simple. It will consist of delivering a working Jupyter Notebook with careful formatting, documentation, and interactive features enabled to the technical team at UltraSignup.com together with a user guide detailing the steps to open, run, and interact with a Jupyter Notebook on their local machine. The runtime environment for Jupyter Notebooks is Anaconda which is freely available and cross-platform, so this project may be run on any local machine running any major operating system.

Phases of Rollout

A projected final version of this app will require at least two phases of rollout.

The **first phase** will be the result of this project and will be a working prototype Machine Learning model using a limited dataset of approximately 12,000 samples. It will be designed for use by the technical team at UltraSignup.com.

A future **second phase** of the project should incorporate a larger dataset, preferably UltraSignup.com's entire database. Including the entire database will improve the accuracy of

the Machine Learning model. It will also require the implementation of additional functionality. UltraSignup.com's database includes trail races of all distances, so the model will need to account for various distances, whereas the prototype will only include data for 100 mile trail races. Additionally, the database contains data of varying qualities, so added parsing and cleaning functionality will be necessary. Access to the larger database, will bring with it more detailed data about individual runners. This data may be used to increase the number of relevant feature variables, and tune the model so that it achieves greater accuracy. With a significantly larger dataset, it will also be necessary to more heavily optimize the Machine Learning model do that its analysis can be completed within a reasonable amount of time.

A **final phase** of the project, which may be completed in conjunction with the second phase, will involve incorporating the Machine Learning model into the UltraSignup.com website's backend. I do not have access to the UltraSignup.com infrastructure, and as such, can only guess which technologies are currently in use. However, since this model will be built using Python and the Scikit-Learn Machine Learning libraries, one of the best options for integrating the model into the existing website will be to use Flask. Once the model is successfully deployed as a replacement for UltraSignup.com's existing Target Finish Time functionality, it will be possible to use the same Machine Learning model as a backend for new interactive features. In particular, it will be possible to create a feature that allows a runner to input any data about themselves, either manually or from their existing UltraSignup.com profile, and the Cutoff time for any trail 100 mile trail race, and receive back an accurate prediction of their finish time. Additional input features could be added such as race Distance, race Elevation Gain and Loss, race Altitude, etc.

Details for Levels of Testing and Final Distribution

Unit testing is the first level of testing and will be performed by the developer. Each unit of the Jupyter Notebook will be run in isolation from the other units to ensure that it is functioning properly. Because the testing is limited to a particular unit, exhaustive testing will be possible. This will allow errors to be detected at an early stage which will save time and money.

Integration testing is the second level of testing in which individual modules in the Jupyter Notebook will be run in conjunction with one another with the aim of detecting any interfacing issues between the modules. Big bang integration testing will be used for this project.

System testing is the third level of testing where the complete project will be tested as a whole. The focus will be on whether the application meets the business requirements laid out in the requirement specification document. This testing will be carried out in an environment similar to the production environment.

Acceptance testing is the final level of testing and will be carried out by UltraSignup.com's technical team once the product is delivered. Since the product will be a prototype, the entire lifecycle of the first phase product will effectively be an alpha test (https://artoftesting.com/levels-of-software-testing).

Final distribution will involve delivering the product to the customer and making it available for use as intended. Because the current project is a prototype, this step will consist of delivering the working product to the technical team at UltraSignup.com.

Dependencies and Milestones

This project's Development will be completed in several steps, each of which depends on the previous one. These steps are dependent on the successful completion of the Feasibility Analysis, Requirement Specification, and Design stages of the SDLC. The completion of each of the following steps will be considered a milestone for this project.

1. **Data collection** for the prototype version of this application will include gathering data from the UltraSignup.com website. Data will be collected from the HTML5 webpages themselves or from the underlying json files using a Python script. This data will be stored in csv files and imported into the Jupyter Notebook as a Pandas DataFrame.

2. **Data cleaning** will take place in the Jupyter Notebook by using the tools offered by Pandas to manipulate the data, including filling null fields, deleting rows and columns containing erroneous data, and creating additional fields as needed.

3. **Model creation and training** will be carried out using Scikit-Learn. Both descriptive Randomized PCA and non-descriptive Random Forest Regressor algorithms will be trained and tested on the input data. The Randomized PCA algorithm will reduce the number of features needed for the predictive Random Forest Regressor algorithm and will therefore improve both the efficiency and accuracy of the results.

4. **Model evaluation** will assess the accuracy of the Machine Learning models using the given input data and the chosen features. The regression algorithm will be evaluated using the R^2 coefficient of determination regression score function as well as the algorithm's mean absolute error.

5. **Post-Implementation Report** will be completed prior to delivery of the product. This report will provide detailed technical specifications for the working product.

6. **Delivery** of the working product.

Deliverables

Tangible deliverables include both Project Deliverables and Product Deliverables. Project deliverables include a Letter of Transmittal, a detailed Non-Technical Project

Recommendation, a detailed Technical Project Recommendation, and a Post-Implementation Report. Product Deliverables include the project's Jupyter Notebook which serves as an interactive, functional user interface, the input data csv files, and log files.

Intangible deliverables include the potential for increased user satisfaction with the client's product offerings and improved reputation in the running community. Such intangible benefits can reasonably be expected to lead to increased revenue.

User testing

The prospective user of the initial prototype which will be produced by this project is the technical team at UltraSignup.com. Therefore user testing will involve setting up the Anaconda environment on the local machines used by the UltraSignup.com technical team, running the provided Jupyter Notebook, ensuring the accuracy of the provided results, and testing the predictive Machine Learning model using input data.

8. EVALUATION PLAN

Describe the methods for validating and verifying that the developed application meets the requirements and subsequently the needs of the client. Discuss your approach to quality assurance and what criteria that is based on. Consider industry standards, regulatory requirements, and organizational policies. What metrics will be used, how measured, and what measurement indicates success. What are the plans for the testing cycles and closures?

This project has a single aim: to produce accurate race finish time predictions using Machine Learning. Therefore the evaluation plan consists primarily of technical Machine Learning evaluation metrics. Provided these metrics are within acceptable ranges, the product will have demonstrated itself to be working properly. The same results should function properly on the client's local machines in order to allow for future stages of the project including deployment to UltraSignup.com's website as an end-user feature. The evaluation metrics to be used are:

Explained Variance will be used to evaluate the descriptive Randomized PCA algorithm. Explained variance is a measure of the relative impact of each input feature on the predictive Machine Learning model. It can be used to reduce the number of features needed to create an accurate model, which improves the efficiency of the program. If the variance of a given feature is low, it does not contribute much to the predictive model (https://towardsdatascience.com/principal-component-analysis-explained-d404c34d76e7). Variance is measured by generating Principle Components (eigenvectors) which are combinations of features of the original dataset, then calculating the ratio of the related eigenvalue and sum of the eigenvalues of all eigenvectors. Let the given eigenvalue be λ_i and the sum of all eigenvalues be represented as $\lambda_1 + \lambda_1 + \ldots + \lambda_n$. Then the formula used to calculate the Explained Variance for the given Principle Component is (https://vitalflux.com/pca-explained-variance-concept-python-example):

$$\frac{\lambda_i}{\lambda_1 + \lambda_1 + \ldots + \lambda_n}$$

R^2 Coefficient of Determination will be used to evaluate the non-descriptive / predictive Random Forest Regressor algorithm. This metric "represents the proportion of variance (of y) that has been explained by the independent variables in the model." It provides "a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance" (https://scikit-learn.org/stable/modules/model_evaluation.html). If $y_p(i)$ is the predicted value of the *i*th sample and $y_a(i)$ is the actual value of the *i*th sample, then the formula for calculating the R^2 coefficient of determination is:

$$R^2(y_p, y_a) = 1 - \frac{\sum_{i=1}^{n} (y_a(i) - y_p(i))^2}{\sum_{i=1}^{n} (y_a(i) - z)^2} \quad \text{where } z = 1/n \sum_{i=1}^{n} y_a(i)$$

This score can have a maximum value of 1 and may be negative because the model can be arbitrarily worse. The closer the score is to 1, the better the model is able to predict the actual outcome (https://medium.com/analytics-vidhya/coefficient-of-determination-r²-a337b8d3e9e7).

Mean Absolute Error will also be used to evaluate the non-descriptive / predictive Random Forest Regressor algorithm. The result will be expressed in hours and minutes and will represent the mean difference between the predicted finish time and actual finish time for each sample. If $y_p(i)$ is the predicted value of the *i*th sample and $y_a(i)$ is the actual value of the *i*th sample, then the formula for calculating the Mean Absolute Error is (https://scikit-learn.org/stable/modules/model_evaluation.html):

$$MAE(y_p, y_a) = 1/n \sum_{i=1}^{n} |y_a(i) - y_p(i)|$$

9. RESOURCES AND COSTS

Elaborate/itemize the costs required to test and complete the project in a production environment.

Programming Environment

Provide a clear picture of what hardware and software are required to complete the project.

This product will be created on a MacBook Pro running MacOS Big Sur 11.0.1. But it will be possible to create the project on any local machine running any major operating system

including Windows, MacOS, or any Linux distribution. It will be assumed that the developer already has access to hardware in order to design and create this application.

Anaconda must be installed on the local machine in order to create and run the Jupyter Notebook which will serve both as this application's design environment and its user interface. Anaconda is open source and freely available for individual use or $14.95 per month for commercial use (https://www.anaconda.com/pricing). The free individual version will be sufficient for creating and designing the prototype version of this application.

Programming environment total cost: **$0.00**

Environment Costs

Provide an explanation of the costs associated with the application. Some might be startup, first-time costs while others might be a percentage of licensing costs. Environment costs are relatively minimal. The environment where the system resides in a shared environment where costs are shared by the organizations.

This product can be run on any local machine running any major operating system including Windows, MacOS, or any Linux distribution. It will be assumed that the user already has access to hardware in order to run this application.

Anaconda must be installed on the local machine in order to run the Jupyter Notebook which will serve as this application's user interface. Anaconda is open source and freely available for individual use or $14.95 per month for Commercial use (https://www.anaconda.com/pricing). The free individual version will be sufficient for creating and designing the prototype version of this application.

Environment total cost: **$0.00**

Human Resource Requirements

What is the time and cost for the labor to complete the application?

This project, even when fully built out, will be rather straightforward to implement. It requires that the developer or developers have expertise in Machine Learning, but the initial prototype version can be implemented in approximately **100 work-hours**.

A full-featured version of the application, trained on a comprehensive dataset and accounting for a larger array of feature variables would require an additional 100 work-hours.

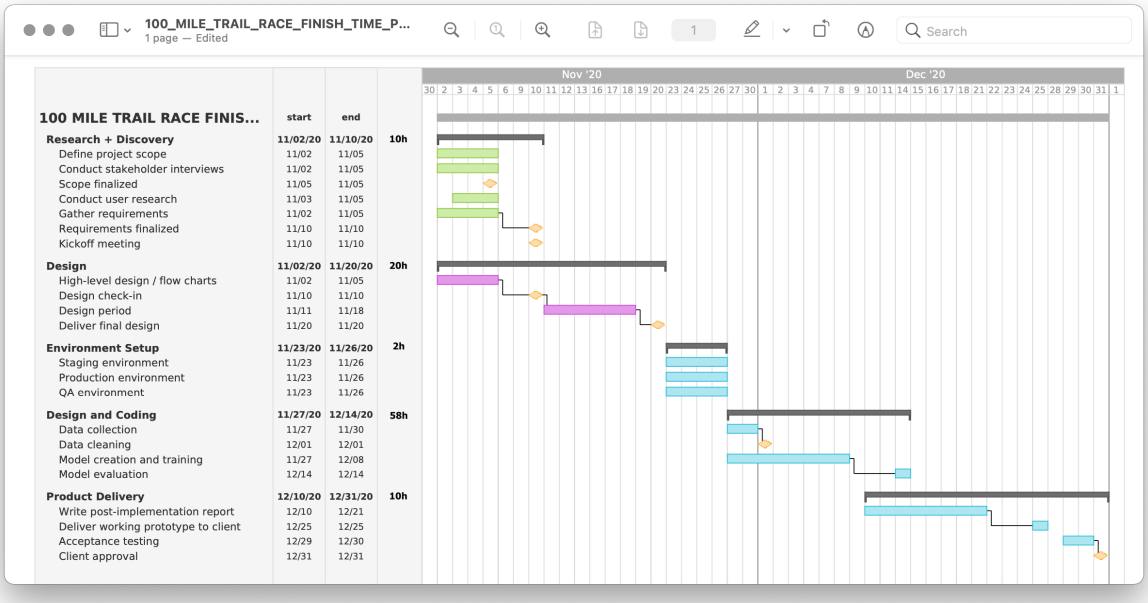
For a Software Engineer trained in Machine Learning making $50 per hour, the total cost of this project will be $5,000.00 for the prototype version and another $5,000.00 for the fully built-out version.

Human resource total cost: **$5,000.00**

10. TIMELINE AND MILESTONES

Generate a projected timeline, including milestones, start and end dates, duration for each milestone, dependencies, and resources assigned to each task. Use a table to display your timeline material.

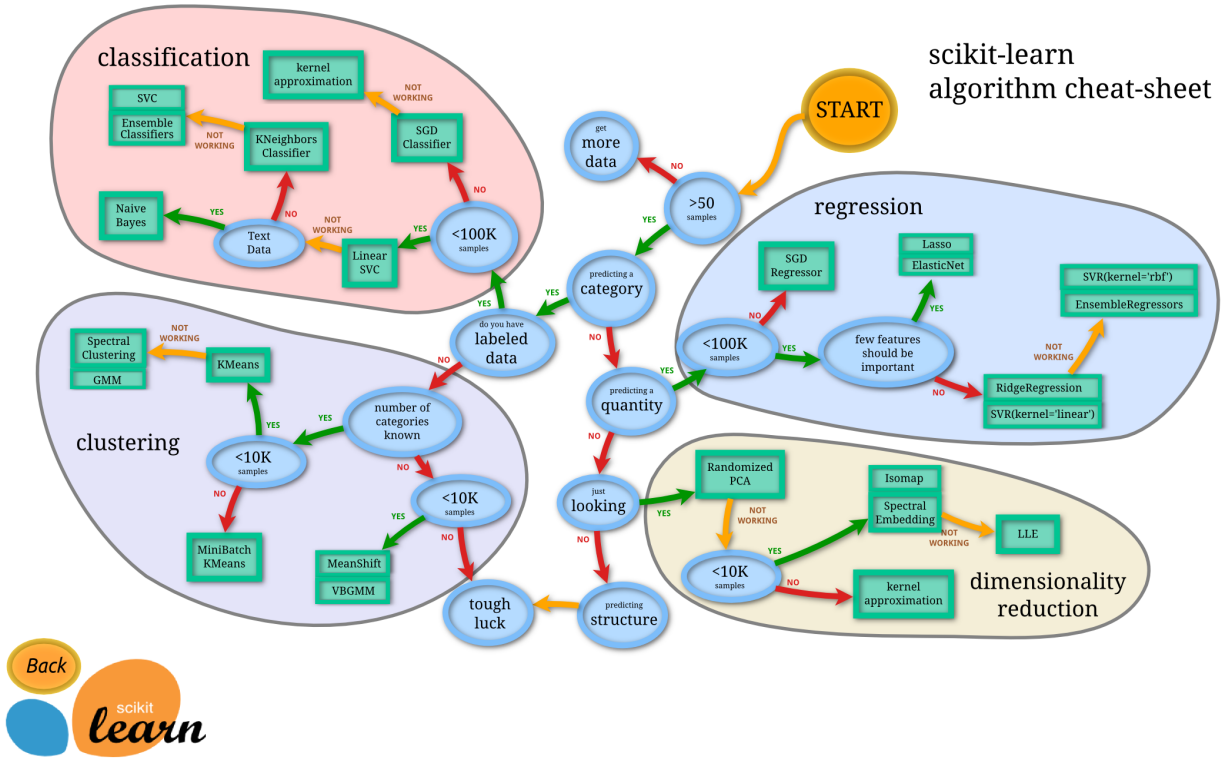
Figure 2: Timeline and Milestones Gantt Chart



C. FULLY FUNCTIONAL DATA PRODUCT

Design and develop a fully functional data product (application) addressing your identified business problem or organizational need. Your ability to explain how your application meets said need is essential so remember it's the overall context what your application is providing.

Figure 3: Scikit-Learn Machine Learning Map (source: https://scikit-learn.org/stable/tutorial/machine_learning_map)



The aim of this project is "predicting a quantity" which means that the appropriate descriptive and non-descriptive data methods may be found on the right side of the above chart.

1. DESCRIPTIVE DATA METHOD: RANDOMIZED PCA

Provide one descriptive method that discerns relationships and characteristics of the past data in at least three forms of visualization. The descriptive method should be in the domains of cluster or association analysis. Be sure to include a justification of the methods selected in relationship to the goal(s) of the project. Examples of acceptable descriptive methods: K-means clustering, Hierarchical clustering, Other clustering methods (max distance, min distance, etc.), PCA (all variables need to be numeric), MCA (all variables need to be factorized), FAMD, Logistic regression with interpretations of estimated coefficients, Multi-linear regression with interpretations of estimated coefficients.

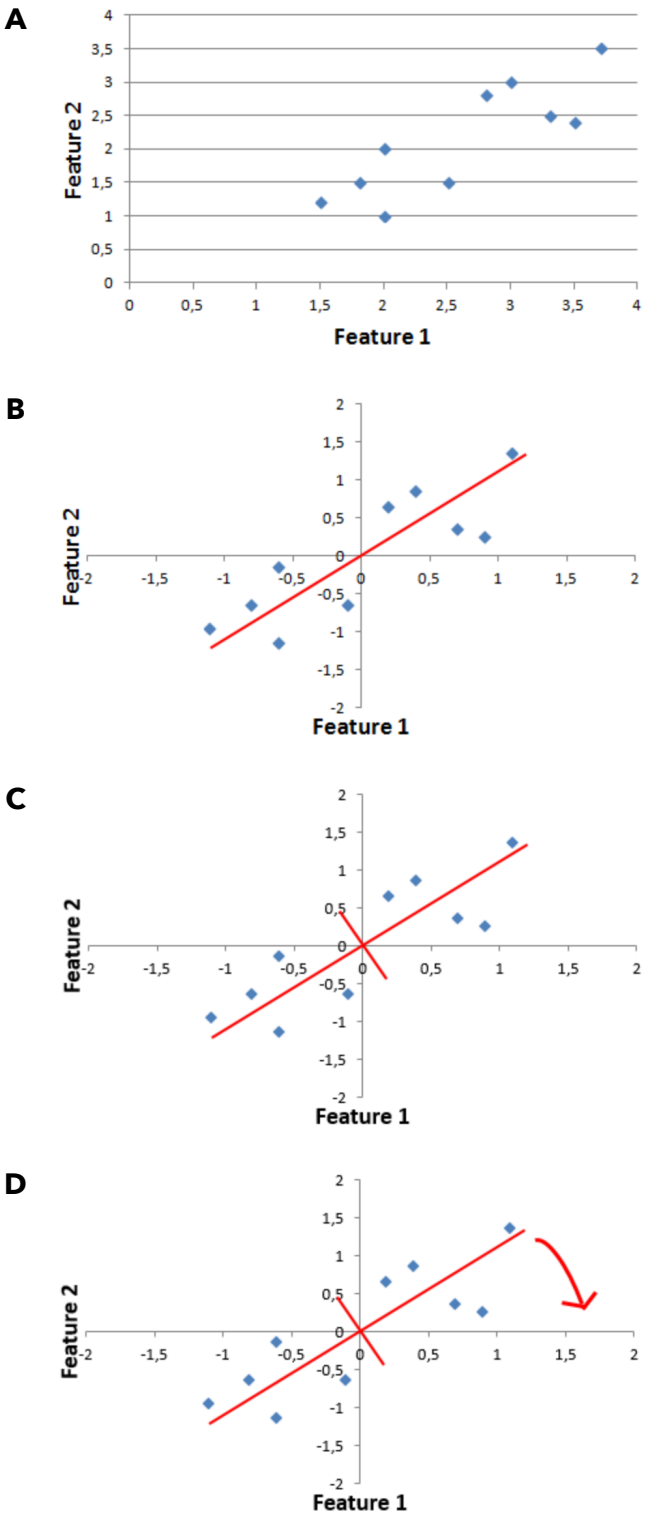
Descriptive data methods include Clustering and Dimensionality Reduction and are found in the lower half of the above chart. Clustering algorithms are appropriate for projects that aim to predict a category, while **Dimensionality Reduction** algorithms are appropriate for projects that aim to **predict a quantity**. Since this project aims to predict finish times, the appropriate non-descriptive data method is a Dimensionality Reduction algorithm. The first choice of Dimensionality Reduction algorithms is **Randomized Principle Component Analysis (PCA)**. Since this project's dataset includes greater than 10,000 samples, Kernel Approximation would also be appropriate. However, since Randomized PCA did produce favorable results it was not necessary to seek out an alternative. Therefore, this project implements a Randomized PCA algorithm using Scikit-Learn's `PCA()` method.

Principle Component Analysis is an **unsupervised learning algorithm** which finds the relations among features within a dataset. PCA is widely used as a **preprocessing** step for predictive algorithms, and it has been used for that purpose in this project. The goal of the PCA algorithm is to identify which Principle Components have a greater or lesser impact on the predictive model so that only those features which are most useful can then be used as input for the predictive model. This increases both the efficiency of the predictive model and potentially improves its accuracy as well.

Figure 4A represents a comparison of any two input features of the dataset. PCA begins by shifting the data points so that the center of data is at the origin (Figure 4A to Figure 4B). The linear red line in Figure 4B is the first Principle Component (PC1) and is determined such that the distances from the data points to the line are minimized. The shorter linear red line in Figure 4C is the second Principle Component (PC2) and is orthogonal to PC1. The entire graph is then rotated so that PC1 becomes the x-axis and PC2 becomes the y-axis. Through this process, it becomes clear that PC1 explains much more variance than PC2 (https://towardsdatascience.com/principal-component-analysis-explained-d404c34d76e7). For detailed graphs of the data features used in this project, see **Data Visualization** below.

PCA is one of the two main methods to reduce the number of features. It derives new features from the existing ones while keeping as much information as possible. This process is also called **feature extraction**. The other method for reducing the number of features is **feature selection** which aims to find the most informative features or eliminate uninformative features. Feature selection can be done manually or using software tools. Both methods were used on the dataset for this project, and it was determined that feature selection produced better results.

Figure 4: Principle Component Analysis (source: https://towardsdatascience.com/principal-component-analysis-explained-d404c34d76e7)



The implementation of the PCA algorithm using Scikit-Learn required seven steps (Figure 5).

1. The first step was to **remove extraneous columns** which contained non-numeric data that was clearly not useful as input data for this project. The columns that were removed were the runner's First and Last names and City, and also the runner's Place and gender-specific place (GP). Each of these features would clearly result in overtraining the predictive model if they were included in the input dataset.

2. Because PCA requires numeric input, the second step was to **convert categorial features to numerical form**.

3. The third step was to **split data into features and labels**. Features are the input data columns or 'x' variables, and labels are the output or predicted data columns or 'y' variables. For our model, the features are Race, Year, Cutoff, Location, Age, Gender, and Rank. The label or value to be predicted is the runner's finish Time. The individual records are also shuffled in this step in order to avoid artificially inflated results.

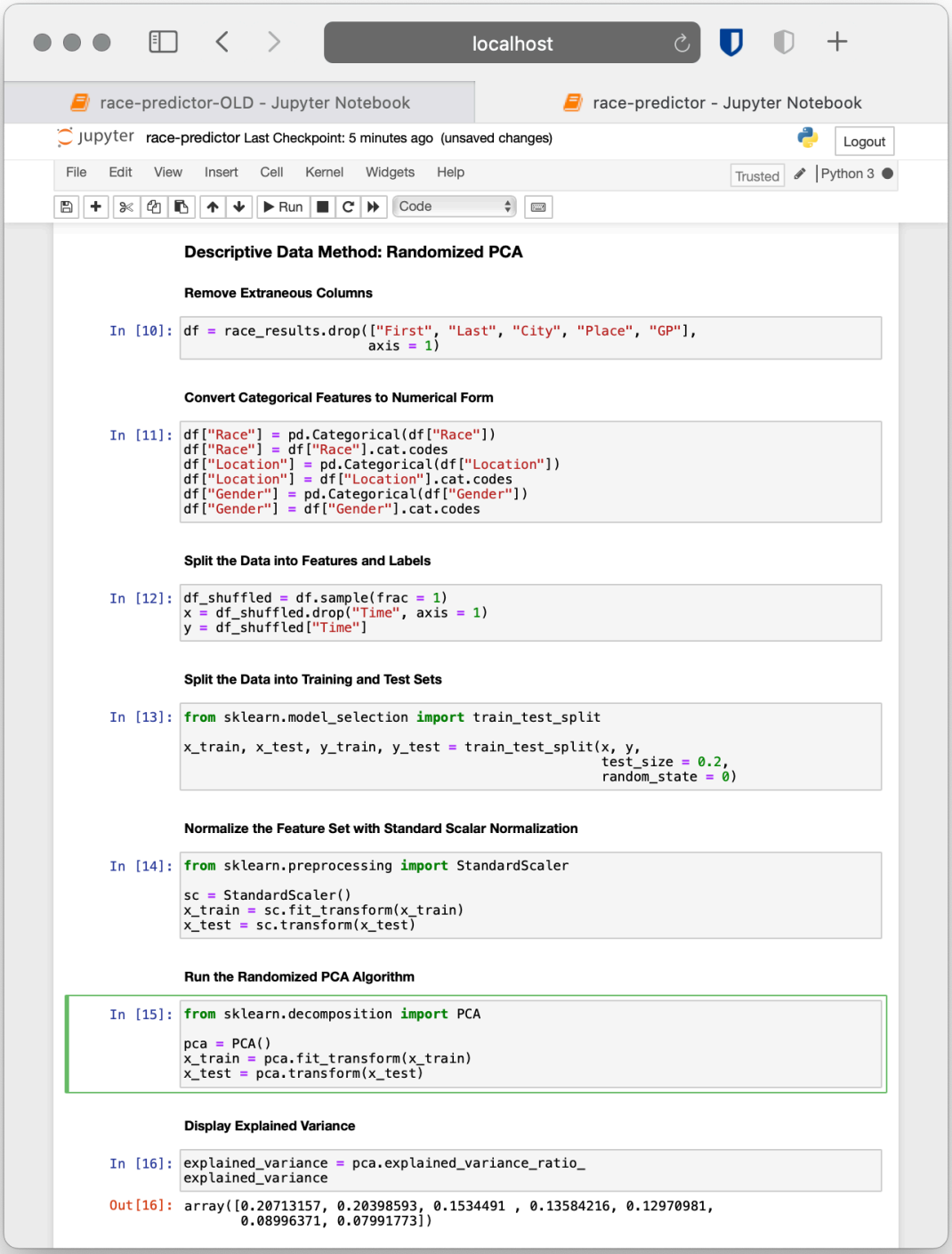
4. The fourth step is to **split the data into training and test sets**. The split used is 0.8 / 0.2 meaning that 80% of the samples are used to train the model, and the remaining 20% are used to test the model.

5. Because PCA works best with normalized datasets, the fifth step is to **normalize the feature set with standard scaler normalization** so that the mean of the data points is 0 and the variance is 1. This is equivalent to the shift from Figure 4A to Figure 4B above. This can be done easily using Scikit-Learn's `StandardScaler()` method.

6. The sixth step is to **run the PCA algorithm**. Scikit-Learn makes this process simple. We create a `PCA()` object and then fit the data points to it. First, we must initialize the PCA class by passing the number of components to the constructor. Then we call the fit and and transform methods and passing the feature set to these methods. The transform method returns the specified number of principal components.

7. The final step is to **display the explained variance**. The result is an array which contains variance ratios for each Principle Component. The output for our data is: [0.20713157, 0.20398593, 0.1534491 , 0.13584216, 0.12970981, 0.08996371, 0.07991773]. This indicates that the first two Principle Components are each responsible for approximately 20% variance in the dataset. The remaining Principle Components are responsible for approximately 15%, 14%, 13%, 9% and 8% variance in the dataset respectively. Using this information we see that there is no clear differentiation between high impact Principle Components and those that contribute less to the variance. Nevertheless it is possible to test the predictive model with various combinations of principle components. As discussed above, it was determined that the best predictive results were produced using manual feature selection, rather than feature extraction.

Figure 5: Randomized PCA Algorithm Implementation Using Scikit-Learn



2. NON-DESCRIPTIVE DATA METHOD: RANDOM FOREST REGRESSION

Provide one non-descriptive where a decision or trend could be inferred. It could include pruning algorithm, discriminate analysis, regression analysis (linear, logistic), Bayesian methods, neural network, or support vector machines. Be sure to include a justification of the methods selected in relationship to the goal(s) of the project. Examples of appropriate non-descriptive methods: Logistic regression, Decision trees, Random forest, Neural network, Multi-linear regression (if it includes a strong justification – like ease of interpretability).

Non-descriptive data methods include Classification and Regression and are found in the upper half of the above chart. Classification algorithms are appropriate for projects that aim to predict a category, while **Regression** algorithms are appropriate for projects that aim to **predict a quantity**. Since this project aims to predict finish times, the appropriate descriptive data method is a Regression algorithm. Since this project's dataset includes less than 100,000 samples, and since few features should be important, either Ridge Regression or **Ensemble Regression algorithms** would be appropriate. Therefore, this project implements an Ensemble Regression algorithm, namely **Random Forest Regression**, using Scikit-Learn's `RandomForestRegressor()` method.

"Random Forest is an ensemble machine learning technique capable of performing both regression and classification tasks using multiple decision trees and a statistical technique called bagging." It builds **multiple decision trees** and then **merges their predictions together** "to get a more accurate and stable prediction rather than relying on individual decision trees" (https://gdcoder.com/random-forest-regressor-explained-in-depth). Randomness is introduced into Random Forest algorithms at two key stages: 1. **Random sampling of training data** when building trees, and 2. **Random subsets of features** for splitting nodes.

The use of multiple trees rather than a single tree prevents overfitting. A single tree can simply grow until it has leaf nodes for every result. The result of such a model is that it will make predictions for data it has seen before with 100% accuracy and will struggle to make predictions for data it has not seen. It will also learn from not only the good data, but also any noise in the data. Rather, a Machine Learning model should aim to generalize the data and strike a good balance between **variance** (or overfitting) and **bias** (or over-generalizing). By training the model on multiple trees generated from subsets of the training data, the Random Forest Regression algorithm accomplishes this both effectively and elegantly. "The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant hence yielding an overall better model" (https://scikit-learn.org/stable/modules/ensemble.html).

The implementation of a Random Forest Regression algorithm in Scikit-Learn is again relatively straightforward. It involves six steps:

1. The first step is to **remove extraneous columns**. Here we trained the model using both PCA feature extraction and manual feature selection and found manual **feature selection** to produce better results for our dataset. Therefore, the Race, Year, First, Last, City, Location, Place, and GP fields are removed in this step. The First, Last, City, Place, and GP fields were found to result in overtraining when included in the input dataset. The Race, Year, and Location fields were found to have minimal impact on the model's output and have been excluded so that the model will accurately predict race results independently of any specific race.

2. The second step is to **split data into features and labels**. Features are the input data columns or 'x' variables, and labels are the output or predicted data columns or 'y' variables. For our model, the features are Cutoff, Age, Gender, and Rank. The label or value to be predicted is the runner's finish Time. The individual records are also shuffled in this step in order to avoid artificially inflated results.

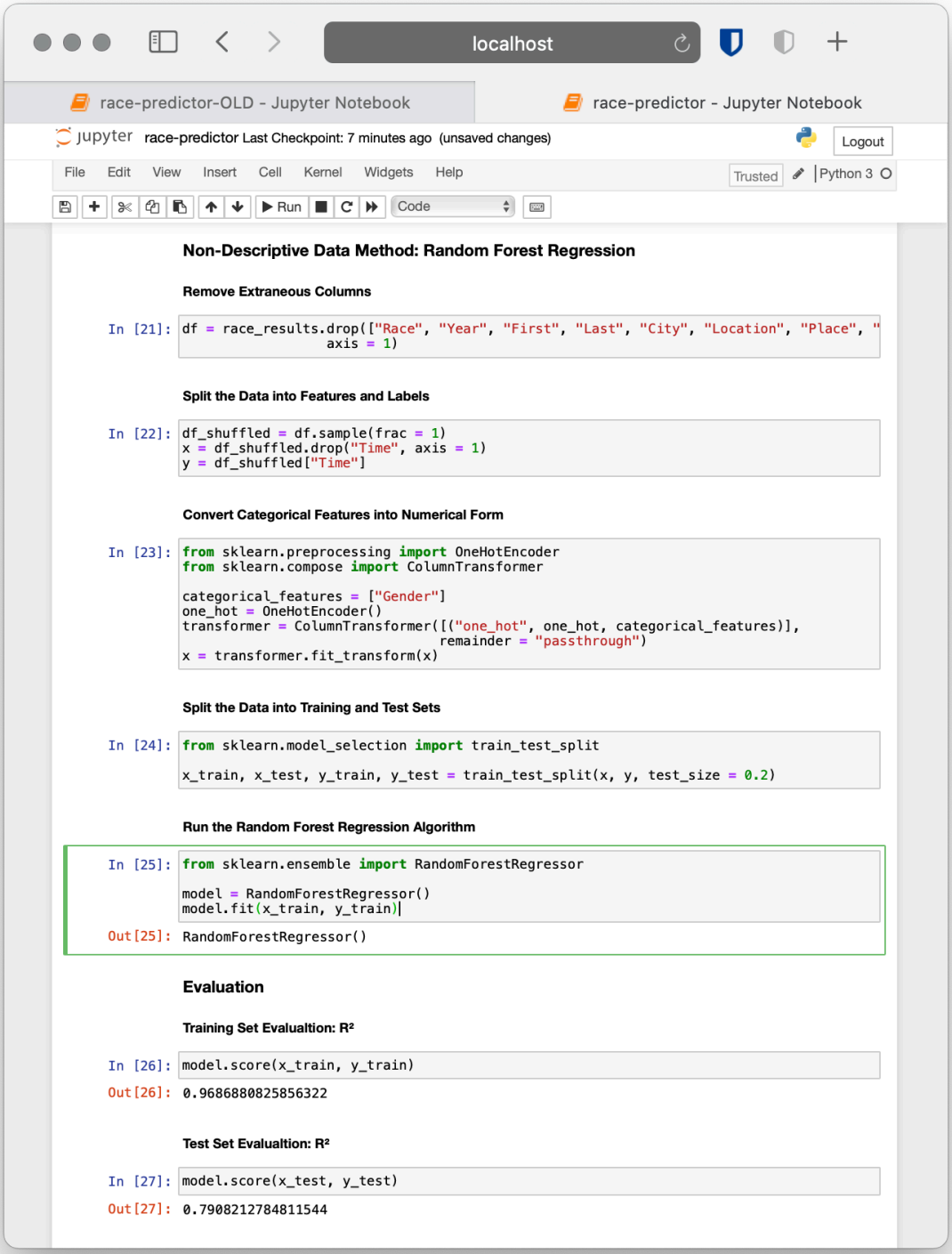
3. The third step is to **convert categorial features to numerical form**. The Random Forest Regression algorithm requires that all input be in numerical form. Here we use Scikit-Learn's `OneHotEncoder()` and `ColumnTransformer()` methods to generate new features for each possible category filled with boolean values for each sample.

4. The fourth step is to **split the data into training and test sets**. The split used is 0.8 / 0.2 meaning that 80% of the samples are used to train the model, and the remaining 20% are used to test the model.

5. The fifth step is to **run the Random Forest Regression algorithm**. Scikit-Learn makes this process simple. We create a `RandomForestRegressor()` object and then fit the data points to it. First, we must initialize the RandomForestRegressor class by passing the number of components to the constructor. Then we call the fit method and pass the feature set to it.

6. The final step is to call the `score()` method which outputs the R^2 coefficient of determination for both the training set and the test set. The **training set evaluation** should output a result close to 100% (our model outputs 97%) since the model has access to both the features and labels for the training set while the **test set evaluation** is a reliable metric of how accurately the machine learning model was able to blindly predict finish times (our model has an R^2 coefficient of determination of 79%). Further evaluation and display of the results was also done and is discussed in detail below.

Figure 6: Random Forest Regressor Algorithm Implementation Using Scikit-Learn



3. DATASETS

The use of dataset(s) is a critical element and involves the gathering and measuring of information on targeted variables in a systematic fashion. This could be student collected (Please consider IRB ramifications.) or publicly accessible such as websites (e.g. Kaggle.com), governmental (e.g. Department of Labor), or software related (e.g. GitHub.com). Be sure to consider the methodology used including possible disadvantages and challenges.

The data that will be used in this project is freely available at UltraSignup.com's website. For this project, the data was collected by manually copying and pasting race results from the HTML5 webpages for the specific races included in this project's dataset. This methodology is obviously limited by its inability to scale efficiently. For the amount of data collected for this project, it was determined to be relatively simple and efficient to collect the data manually. However, for future versions of this application, it will be desirable to include much larger quantities of data. In order to do so, it will be necessary to write a simple Python script to scrape the UltraSignup.com website and automate the collection of data. Nevertheless, UltraSignup.com does have in place web-crawler detection systems, so workarounds will be required. The preferable solution would be to have direct access to UltraSignup.com's proprietary database.

Details regarding the dataset used in this project, including the decision-making process for inclusion or exclusion of specific races and years, and a description of the various anomalies present in the data and how they are handled, may be found in sections A4 and B4 above. The dataset itself may be found in the 'race_results' folder among the included files.

4. ANALYTICS

Using the given data, your application needs to enable decisions to be formulated or support for given trends to be provided.

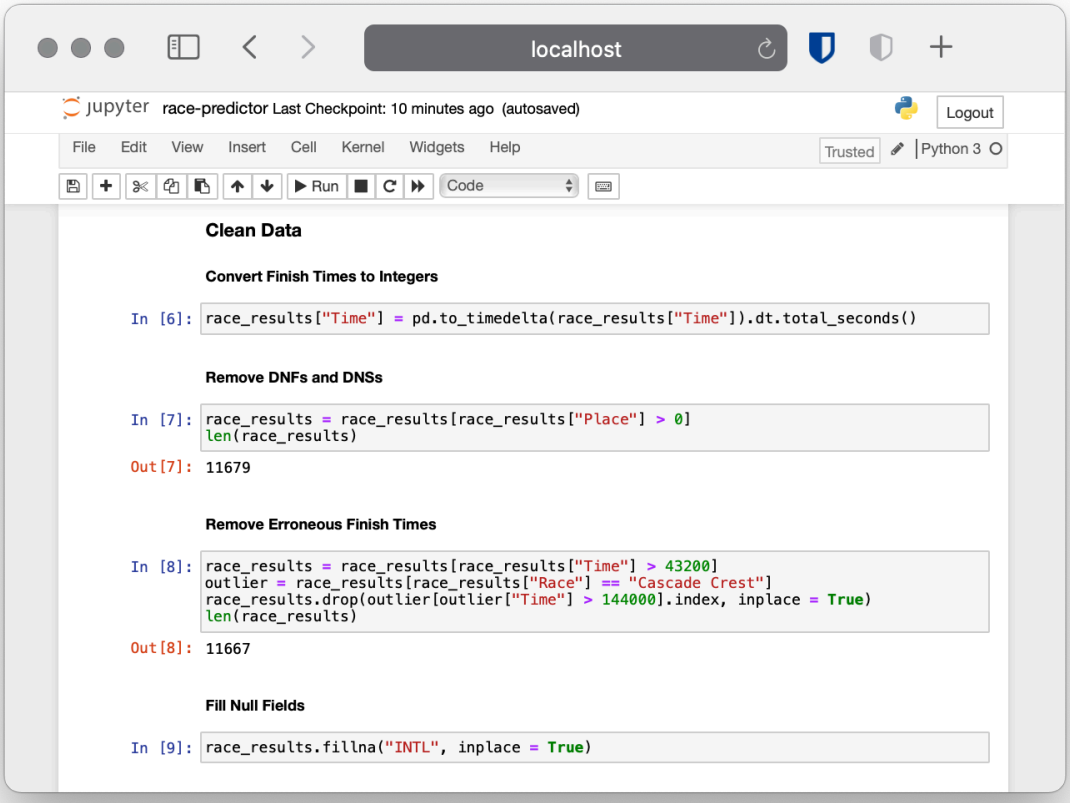
The predictive Random Forest Regressor algorithm produces race finish time predictions for individual runners with a coefficient of determination of 79% and a mean absolute error of approximately 2 hours. This means that a runner can input their Age, Gender, UltraSignup.com Rank, and the Cutoff time for any 100 mile trail race, and receive a race finish time prediction which they can reasonably expect to be within approximately 2 hours of their actual finish time, and a confidence of around 79%. These numbers were achieved with a limited set of data and limited number of input features. The accuracy of the predictions can likely be improved further in future versions by incorporating a larger dataset and more detailed input features.

5. DATA CLEANING

If applicable, create a function that will make the data usable prior to actually being used by the application. Things such as featuring, parsing, cleaning, and wrangling the datasets.

The data cleaning needed for this project's dataset and the specific methods used to parse and clean the data are discussed in detail in **section B4 above**. Specific modifications included adding the Race, Year, and Cutoff fields, excluding samples with **missing or erroneous finish times**, filling or excluding fields which contain anomalies such as **null location fields for international runners**, and **converting all fields to numeric datatypes**. Most of the parsing and cleaning of data was done using **Pandas DataFrames** and **NumPy**.

Figure 7: Data Cleaning Implementation Using Pandas and NumPy



The specific steps taken to clean the data are shown in Figure 7.

1. The first step is to **convert finish times to integers**. In order to convert the values accurately, they are first converted from strings to Timedeltas using the Pandas `to_timedelta()` method, then from Timedeltas to integers using the NumPy `total_seconds()` method. The result is an integer value representing the number of seconds it took for the runner to complete the race.

2. The second step is to **remove DNFs and DNSs**. Many races include records for runners who either did not start (DNS) or did not finish (DNF) the race. These records can easily be

identified and isolated because they contain a finish time equal to 0. Therefore all samples with a finish time of 0 are removed from the dataset.

3. The third step is to **remove erroneous finish times**. Here we check for finish times that are outliers and clearly reflect inaccurate reporting of data. Finish times that are lower than the current world record times for men (11:19:13) and women (12:42:39) respectively are excluded. Finish times that are higher that the official cutoff time of the race, and not part of a cluster of results reflecting a change in policy for a given year, are also excluded. The latter case only occurs once among the 'Cascade Crest' results.

4. The fourth step is to **fill null fields**. Several races have missing Location data for international runners. In these cases, it is appropriate to fill these fields with the value 'INTL'. This value is then treated as a category along with the individual state designations when the field converted to a categorical feature.

After the steps shown in Figure 7 are completed, the categorial fields are then converted to numeric datatypes as shown in Figures 5 and 6 above. The addition of the Race, Year, and Cutoff fields is completed previously during the data collection process.

6. DATA VISUALIZATION

You need at least three real-time (e.g. using the GUI/dashboard) formats to visualize the data in a graphic format. Look at things like charting, mapping, color theory, plots, diagrams, or other methods (tables must include heat mapping). These, in conjunction with or as a part of your GUI, would enable users to explore or inspect the data characteristics.

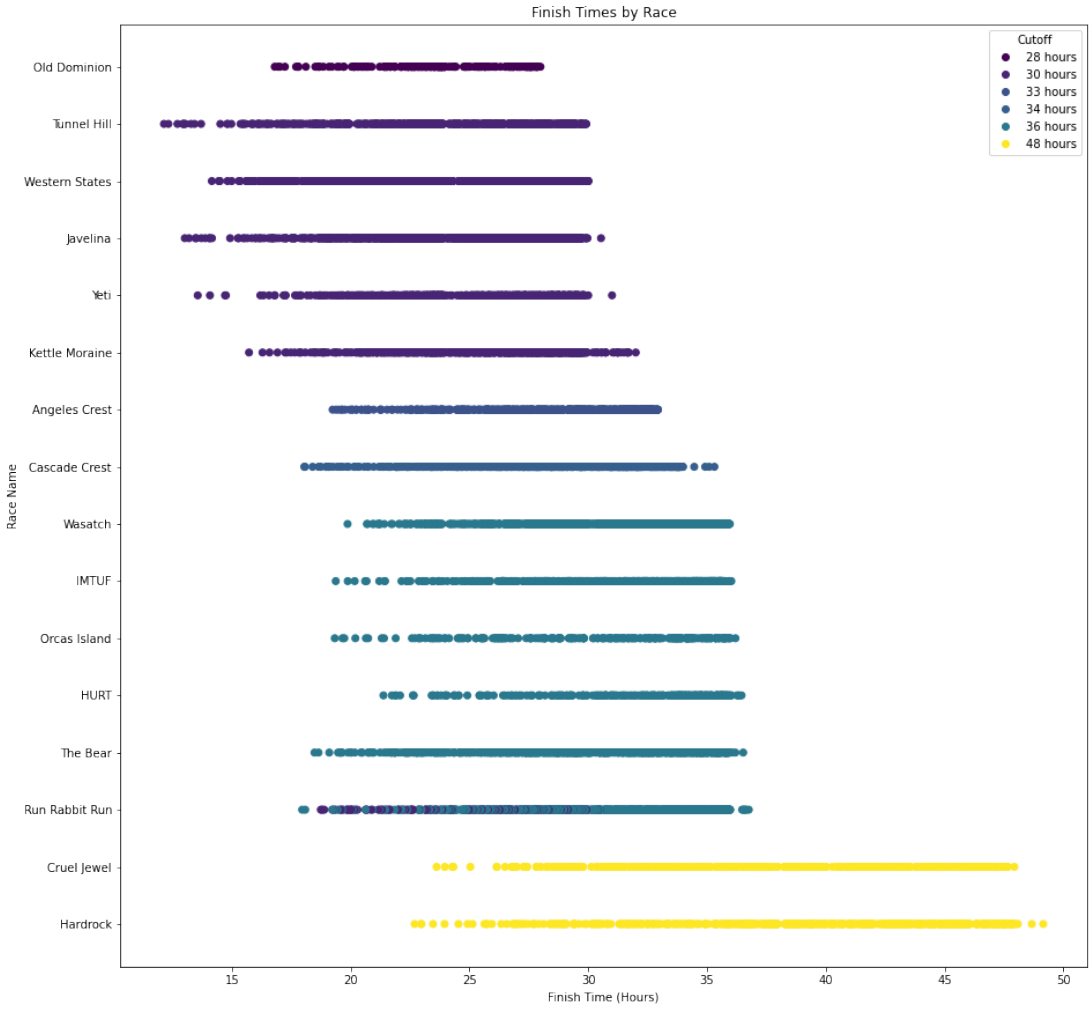
The data visualizations included in the project fall into two categories. The first three are illustrate the descriptive modeling of the data, while the second three illustrate the results of the predictive model.

Each of the visualizations below **can be easily modified and updated in real-time** within the Jupyter Notebook. The possible customizations are endless, and with even a passing familiarity with Python and Matplotlib, they can be done with ease.

Real-time modifications may include adjusting the x-axis, y-axis, and heat-mapping color variables, adjusting the scale, size, and/or colors of the visualizations, adjusting the labels, gridlines, and/or graph style, and finally the graphs can be updated with new data with the press of a single key combination. While a moderate amount of technical skill is required, the target user for this prototype version of the app is the technical team at UltraSignup.com which can be reasonably expected to be competent with these basic technologies. Furthermore, the level of control and customization is significantly improved versus a more simplistic user-interface.

Descriptive Visualizations

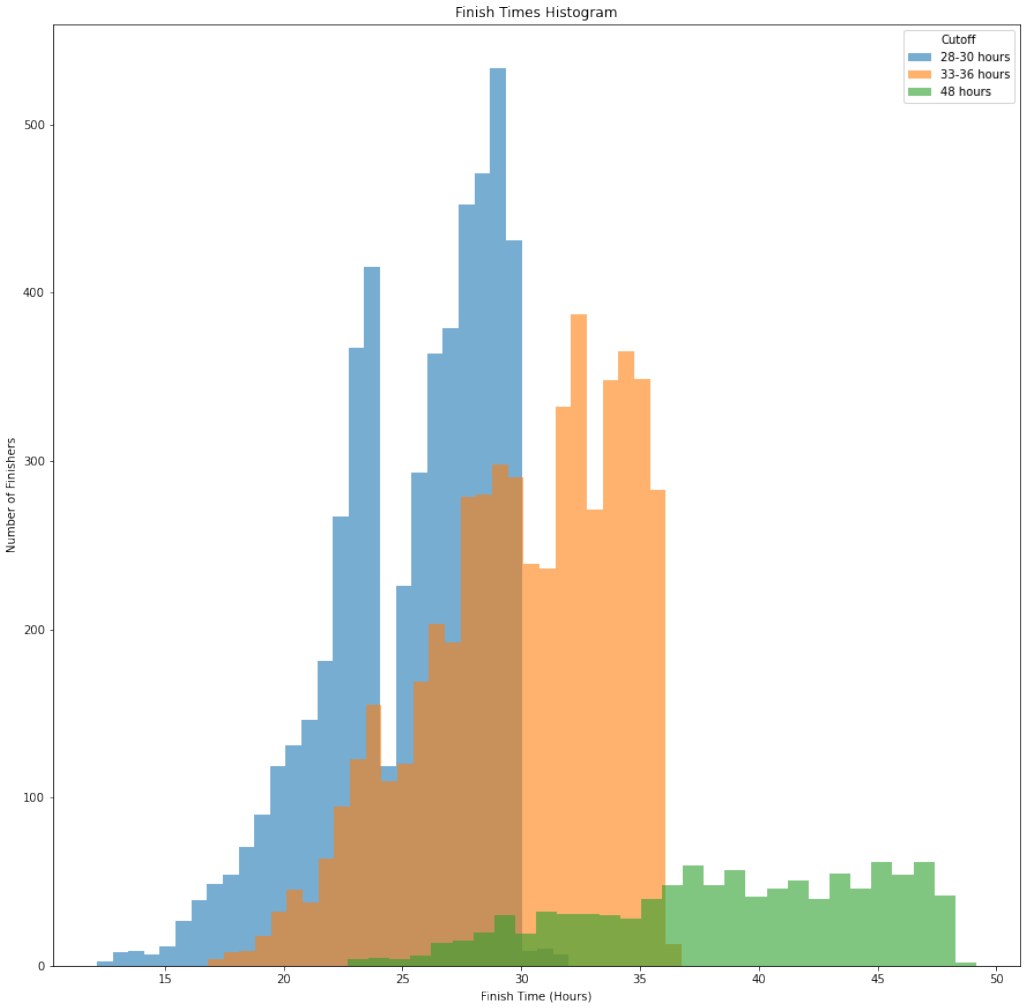
Figure 8: Finish Times by Race



The first figure displays the names of each Race along the y-axis and the finish Time in hours along the x-axis. The data is represented as a scatter plot, but because of the large number of runners, the results appear mostly as solid lines. The results are colored according to the Cutoff times for each race, with purple representing a cutoff of 30 hours, various shades of blue representing cutoff times between 33 and 36 hours, and yellow representing cutoff times of 48 hours.

The less dense results on the left side of the figure illustrate the intuitive fact that faster finish times are less common than slower finish times. The significant difference between the winning time for Tunnel Hill and the winning times for Cruel Jewel and HardRock reflect the significant range of difficulties represented by these races in spite of the fact that they are all 100 mile races.

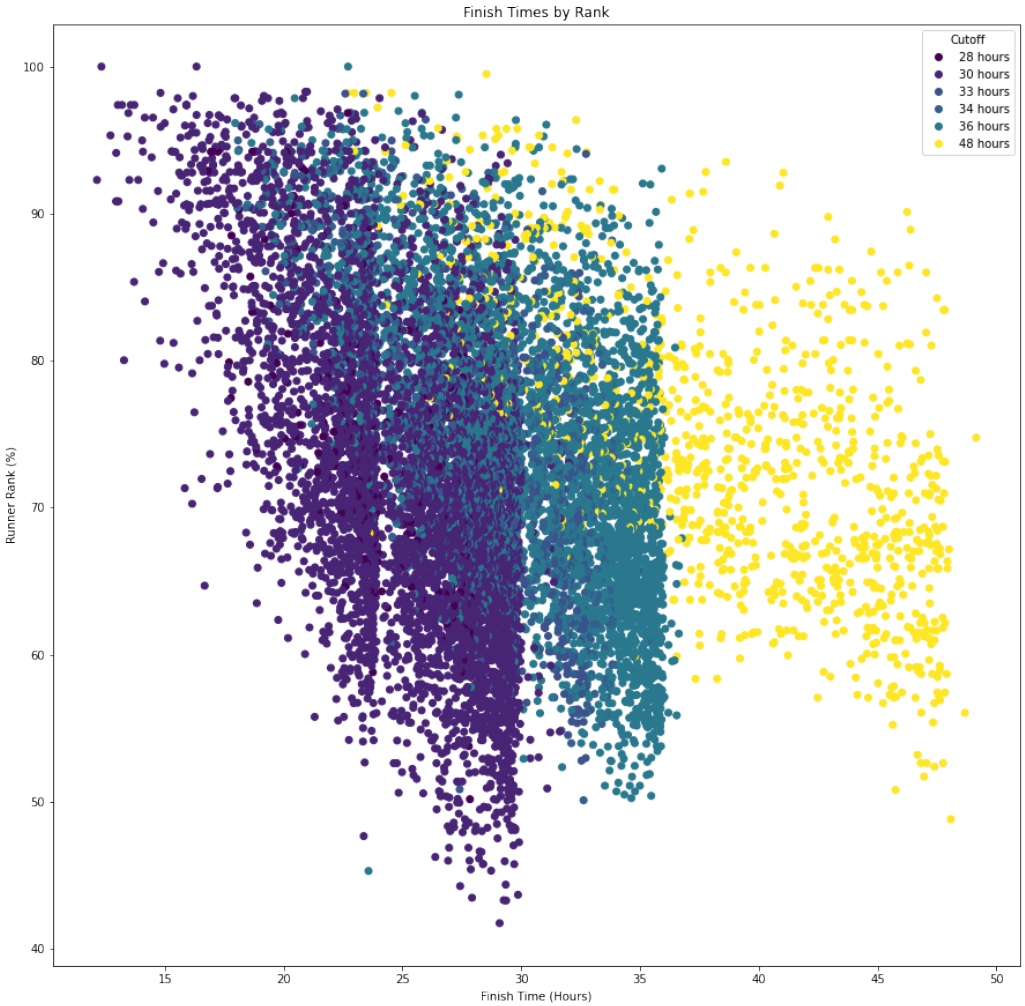
Figure 9: Finish Times Histogram



The second figure displays a histogram of finish times with Time in hours along the x-axis and the number of runners who finished in that amount of time along the y-axis. The results are colored according to race Cutoff times, with blue representing cutoffs of 30 hours or less, orange representing cutoffs of 33–36 hours, and green representing cutoffs of 48 hours.

This figure reveals an important feature of 100 mile ultramarathons, namely the somewhat arbitrary motivations of many runners. The blue results contain two peaks, and that is neither an error nor a coincidence. The first peak is at 24 hours and represents the fact that many runners set a goal to finish in less than 24 hours. Similarly, races with 36 hour cutoffs have many runners who aim to finish in 24, 30, or 32 hours.

Figure 10: Finish Times by Runner Rank



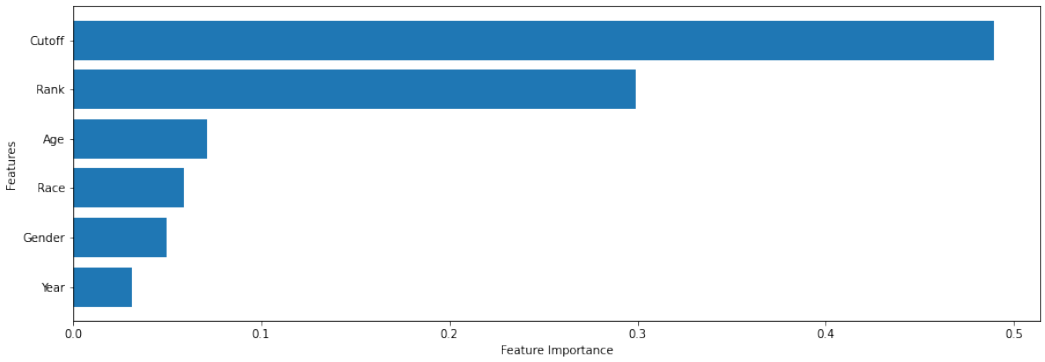
The third figure plots finish Time in hours along the x-axis and each runner's UltraSignup.com Rank along the y-axis. The results are once again colored according to race Cutoff times. This is a significant graph for the current project since Rank is one of the primary determining features of the predictive model.

This figure reveals some interesting trends. The purple results are from races with a cutoff of 30 hours. As you would expect, runners with faster finish times tend to have a higher UltraSignup.com Rank. It is also not surprising that runners with a low UltraSignup.com Rank are not among the fastest finish times. Thus the trend from upper left to lower right is expected. This trend confirms that the present dataset is a useful feature for our predictive model.

However, this figure also reflects a large number of runners with high UltraSignup.com Rank and slow finish times. This occurs because UltraSignup.com takes into account shorter races when calculating a runner's Rank. Several of these runners may be first time 100 mile race finishers. It also reflects the fact that 100 mile races are incredibly unpredictable, and many runners end up finishing with much slower times than anticipated. This illustrates both a significant challenge for our machine learning model and also illustrates perhaps the primary benefit of a more complex predictive model such as Random Forest Regression. Our model will take into account many of the complex realities of ultramarathon trail running that simple arithmetic cannot adequately reflect.

Predictive Visualizations

Figure 11: Feature Importance



Feature Importance is a metric provided by the Scikit-Learn `feature_importances_` instance variable of the `RandomForestRegressor()` class. It measures the relative importance of each feature variable in the input dataset to the Random Forest Regression predictive model. Figure 11 shows that the most important variable for the predictive model is Cutoff. This is to be expected since differing cutoff times have a significant impact on Finish Times, as illustrated in Figure 8 above. The second most important feature is Rank. This too was anticipated by the data represented in Figure 10 above. The remaining features have a relatively small impact on the predictive model. This is confirmed by the fact that when they adding or removing these features from the input dataset results in only minimal changes to the R^2 coefficient of determination and mean absolute error. Feature Importance is an important metric for tuning the machine learning model through feature selection.

Figure 12: Actual Finish Time / Predicted Finish Time

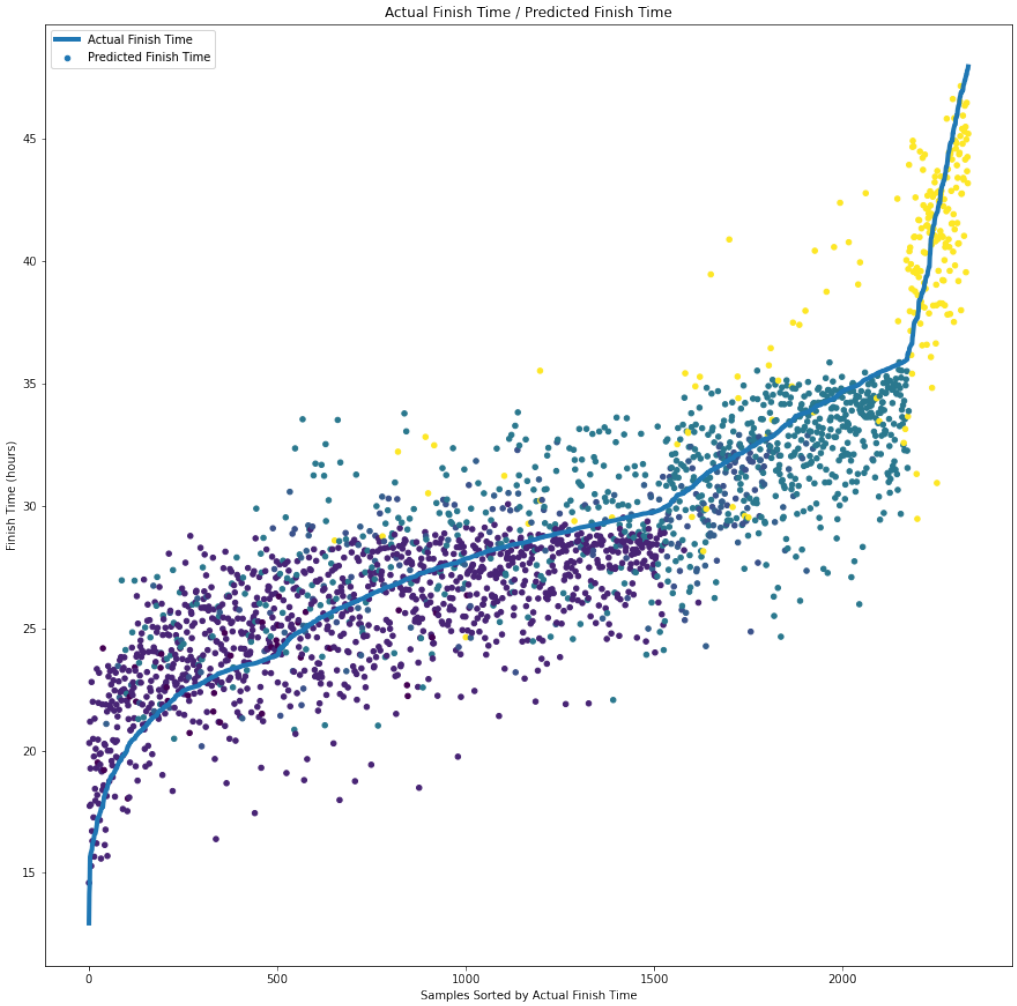


Figure 12 presents the results of the Random Forest Regression algorithm trained on our dataset. The figure plots the samples in the test dataset sorted by Actual finish time along the x-axis, and the finish Time in hours along the y-axis. The blue line represents the sorted Actual finish times from fastest to slowest. The scatter plot represents the corresponding Predicted finish times. The scatter plot is colored according to race Cutoff times like the descriptive figures above. Figure 12 demonstrates that the Random Forest Regression Algorithm successfully fits the Predicted data to the Actual data. Nevertheless, it is also clear that there is room for improvement, as a significant number outliers lie outside the target mean absolute error for this project.

Figure 13: Mean Absolute Error

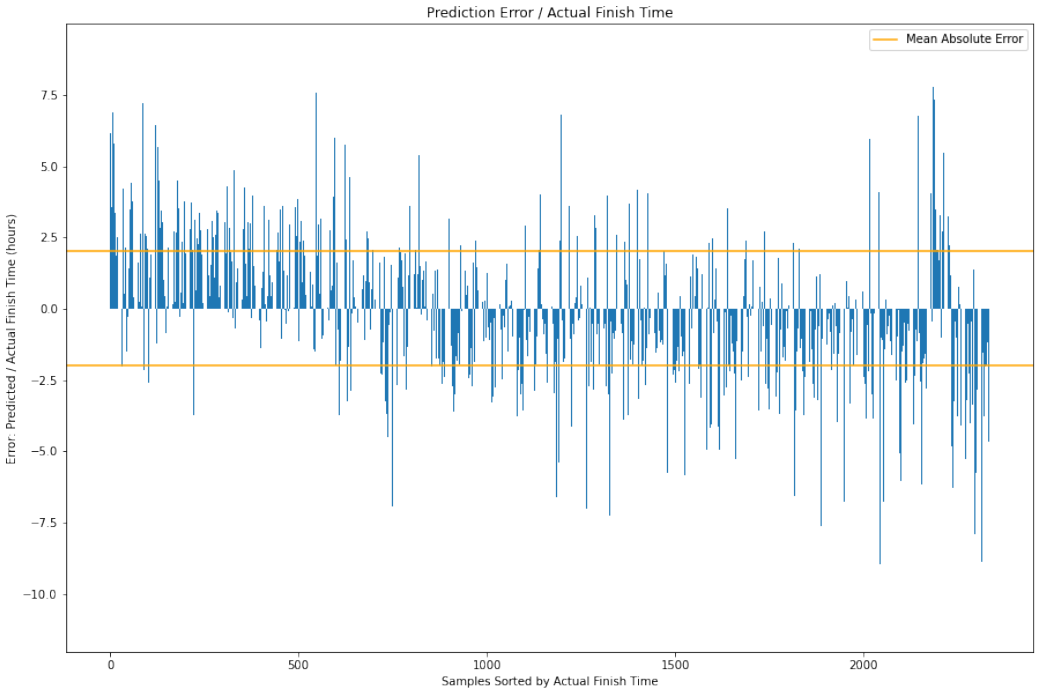


Figure 13 plots plots the samples in the test dataset sorted by Actual finish time along the x-axis, and the error for each sample along the y-axis. Additionally the calculated Mean Absolute Error (MAE) is represented by two orange horizontal lines. This figure shows that the majority of Predicted finish times fall within the MAE, but a significant number of Predicted finish times have an error that is far higher than the MAE.

Several patterns exist among the Predicted finish times with a very high error. Predicted finish times tend to be too high on the left side of the graph and tend to be too low on the right side of the graph. This indicates that the Random Forest Regression algorithm's **bias** may be too high. The trend is especially prevalent where the Actual finish times are very fast (on the left side of Figure 13) and will result in predictions for elite runners that are consistently too high.

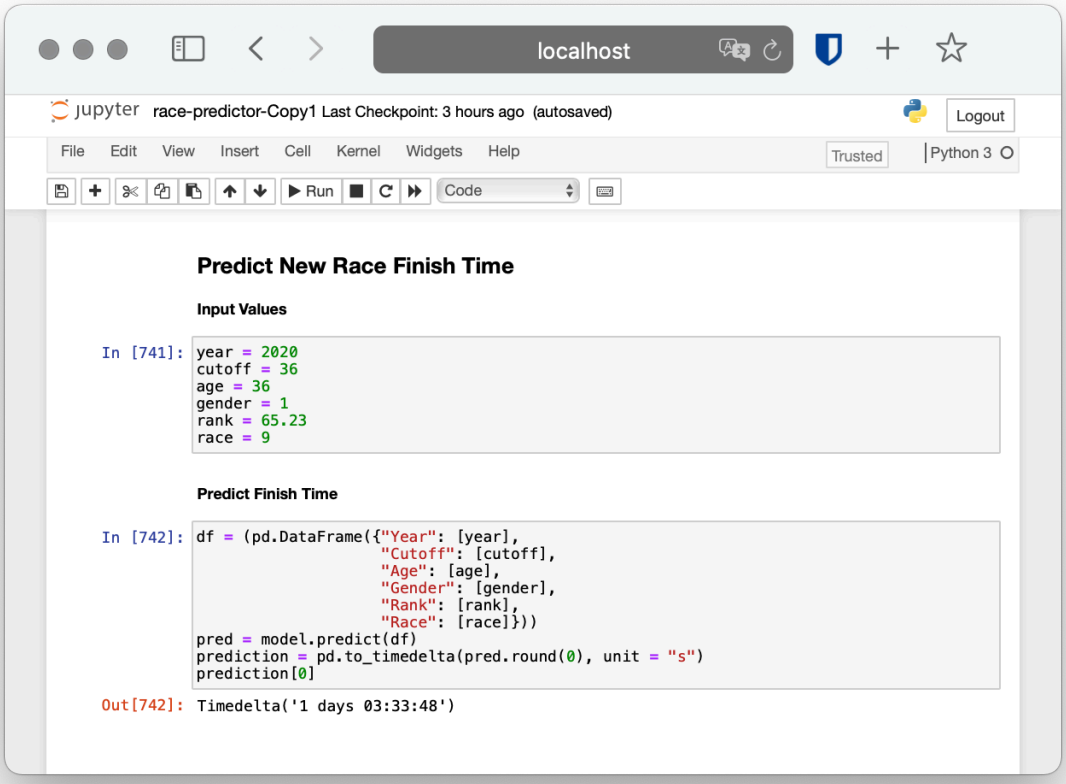
There also exist spikes in error at certain points. The cause of these spikes becomes apparent when comparing Figures 12 and 13, which have the same scale along the x-axis. On the right side of Figure 13, there is a spike in Predicted finish times that are too high immediately after the results for the races with a 36 hour Cutoff are no longer present. This shows the model over-compensating for the 36 hour Cutoff races in the 48 hour Cutoff race results. A similar spike occurs in Predicted finish times that are too low at 24 hours. This corresponds to the

artificially high number of runners that finish just under 24 hours, as evident in Figure 9. Once again, it is clear that the model would benefit from hyper-tuning in order to produce slightly higher **variance**.

7. REAL-TIME QUERIES

As part of your GUI enable users to access and manipulate data real-time including data maintenance.

Figure 14: Predicting Race Finish Times from User Input



The Jupyter Notebook includes functionality to allow the predictive model to be used to generate a new predicted Finish times based on previously unseen user input. The user may update any of the green values in the upper box shown in Figure 14. The trained predictive model is then called in the lower box in Figure 14 and the output is converted to a Timedelta datatype. The result is a predicted Finish time in days, hours, minutes and seconds.

The input data shown in Figure 14 represents a 36 year old male runner with an UltraSignup.com rank of 65.23 who plans to run the 2020 Orcas Island 100 Miler (race 9)

which has a cutoff time of 36 hours. The model predicts that this runner will complete the race in 27 hours, 33 minutes, and 48 seconds.

The entire Jupyter Notebook is fully interactive and grants the user control over the training and test data, the process of cleaning that data, tuning and evaluating the Machine Learning algorithms, all of the data visualizations, and use of the Machine Learning model to predict finish times for new data input by the user. This flexibility allows for easy access, manipulation, and maintenance of the application.

8. ADAPTIVE ELEMENT

If appropriate for the business need, provide the implementation of machine-learning methods and algorithms to enable the application to improve with experience. Examples include learning associations, classification, statistical arbitrage, prediction, extraction, and regression.

The Machine Learning models used in this application are inherently adaptive. Both the PCA algorithm and the Random Forest Regression algorithm are designed to seamlessly adapt to new data inputs and updated datasets. These features "can be used to create a data pipeline for continual improvement and adaptation to new information." Additionally, the simplicity of Scikit-Learn implementation of these algorithms allows for versioning the system. The code only needs to be written once and can be reused to train the models on new datasets. It should be kept in mind that the effectiveness of newly trained models using this code will depend on the proper selection of features as well as careful hyper-parameter tuning. Nevertheless, the present code will allow for "a seamless transition of models that can be deployed or rolled back as new models are trained and evaluated."

9. OUTCOME ACCURACY

Provide functionalities that evaluate the accuracy of the information/outcomes given by the application. What are the parameters for valid output data and how will those be checked by the application?

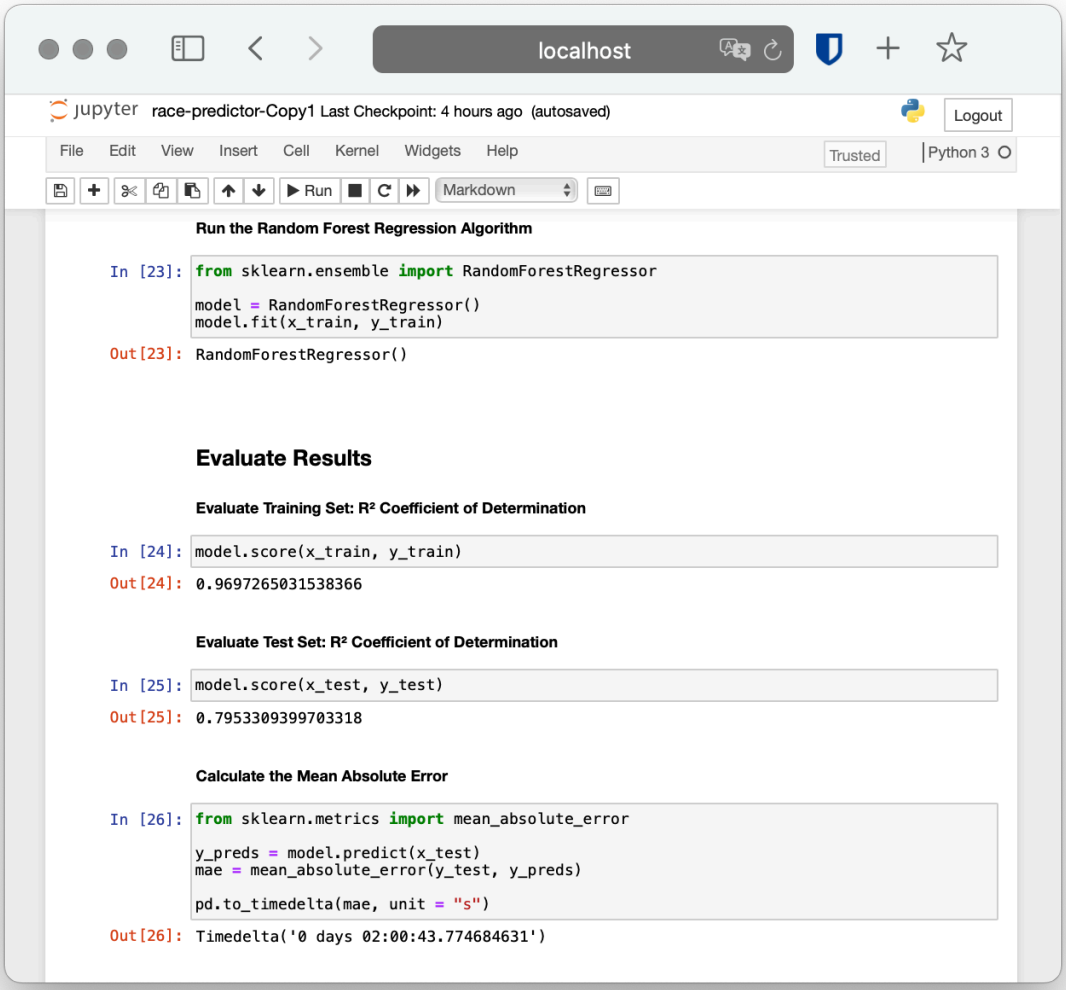
The descriptive and predictive models are evaluated using four distinct metrics during and after training. **Explained Variance** evaluates the descriptive Randomized PCA algorithm. Explained variance is a measure of the relative impact of Principle Components on the model. This is accomplished by generating eigenvectors which are combinations of features of the original dataset, then calculating the ratio of the corresponding eigenvalue and sum of the eigenvalues of all eigenvectors. Similarly, Feature Importance evaluates the effectiveness of the selected features for the current predictive model trained on the current dataset. This metric may be used to adjust which features are selected and improved the accuracy of the predictive model.

The **R^2 Coefficient of Determination** is used to evaluate the output data from the predictive Random Forest Regressor algorithm in order to ensure the validity of the output data. This

metric “represents the proportion of variance (of y) that has been explained by the independent variables in the model.” It provides “a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance” (https://scikit-learn.org/stable/modules/model_evaluation.html).

The R^2 Coefficient of Determination for the Random Forest Regression model trained on the current dataset, with the selected features, and default hyper-parameter tuning is **79.5%** on the test dataset. This means that the current model is 79.5% more accurate on average than outputs chosen at random. While this metric can likely be improved with a model trained on a larger dataset, more comprehensive set of features, and appropriately tuned hyper-parameters, it improves significantly on the primitive model currently in use by UltraSignup.com and is therefore well within an acceptable range for the prototype application produced in this project.

Figure 15: Evaluation of the Random Forest Regression Algorithm



Finally, **Mean**

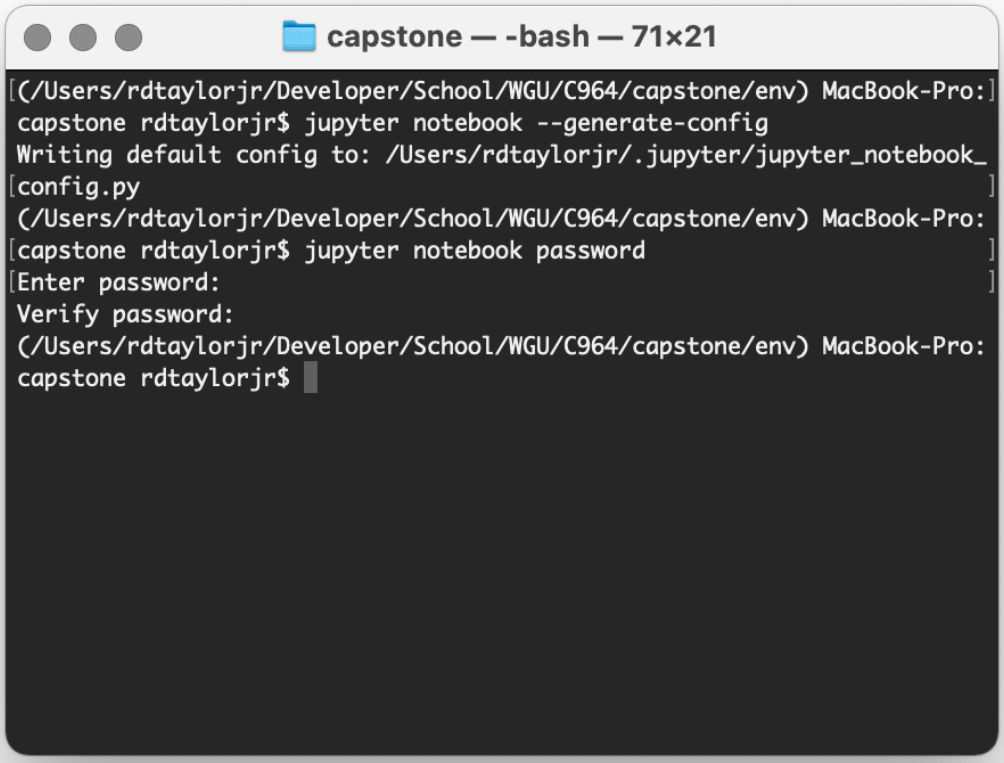
Absolute Error is calculated from the output data of the Random Forest Regressor algorithm. It is used to check that the algorithm is producing results that are within the specified parameters.

The Mean Absolute Error for the Random Forest Regression model trained on the current dataset, with the selected features, and default hyper-parameter tuning is **2 hours, 0 minutes, and 44 seconds**. This means that, on average, predicted times produced by this application will be accurate to within 2 hours of a runner's actual finish time. This value is larger than the target value of 1 hour arbitrarily proposed in the Project Recommendation documentation. Nevertheless, it too offers a significant improvement over the primitive prediction model currently in use by UltraSignup.com and is therefore also within an acceptable range for the prototype application produced in this project.

10. SECURITY MEASURES

Include industry-appropriate security features that will control access to the data and/or, how the data is stored or transmitted. The security features should be appropriate for the data product and the sensitivity of the data it interacts with. For example, with a web application this requirement might be satisfied by implementing username/password authentication.

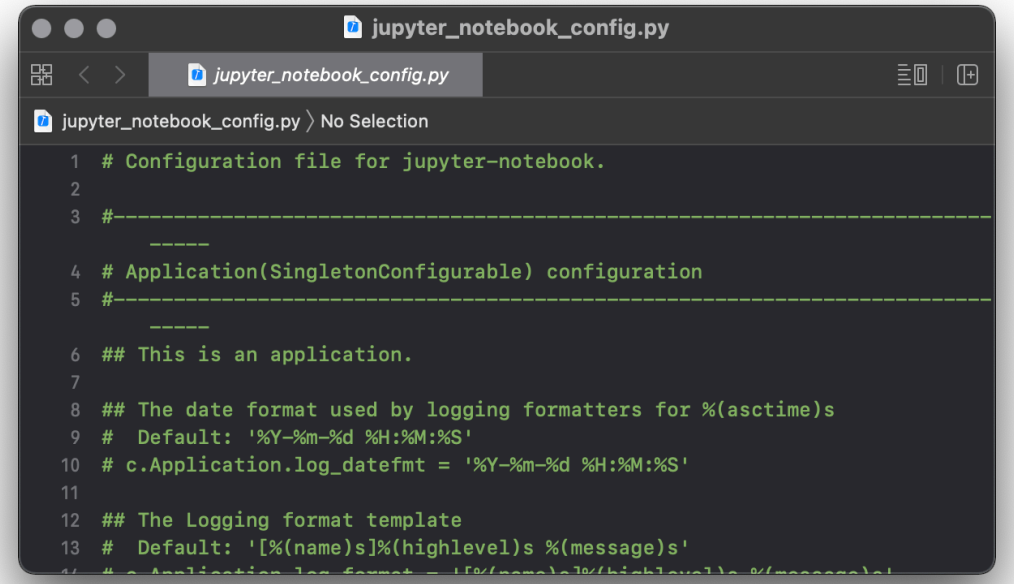
Figure 16: Using the CLI to Password Protect the Jupyter Notebook



Access to the Jupyter Notebook has been **secured with a password** from the Command Line Interface (CLI) by modifying the jupyter_notebook_config.py file. The password is stored in a json file named jupyter_notebook_config.json. Prior to storing the password in this file, the password is **salted** and **hashed**.

The config file and json file should both be placed in the /Users/username/.jupyter directory on the local machine where the Jupyter Notebook is running. The config file may be generated using the `jupyter notebook --generate-config` command in the CLI, and the password may be updated using the `jupyter notebook password` command. Once the password is updated by the user, the property `c.NotebookApp.allow_password_change` in the config file will need to be set to `False`.

Figure 17: The Jupyter Notebook Configuration File



The password is currently set to 'capstone' and password changes are allowed. The user will need to **choose a more secure password** and **set the allow password change property to false** from the CLI (https://thequickblog.com/how-to-password-protect-jupyter-notebook).

Figure 18: The Jupyter Notebook Hashed Password Json File

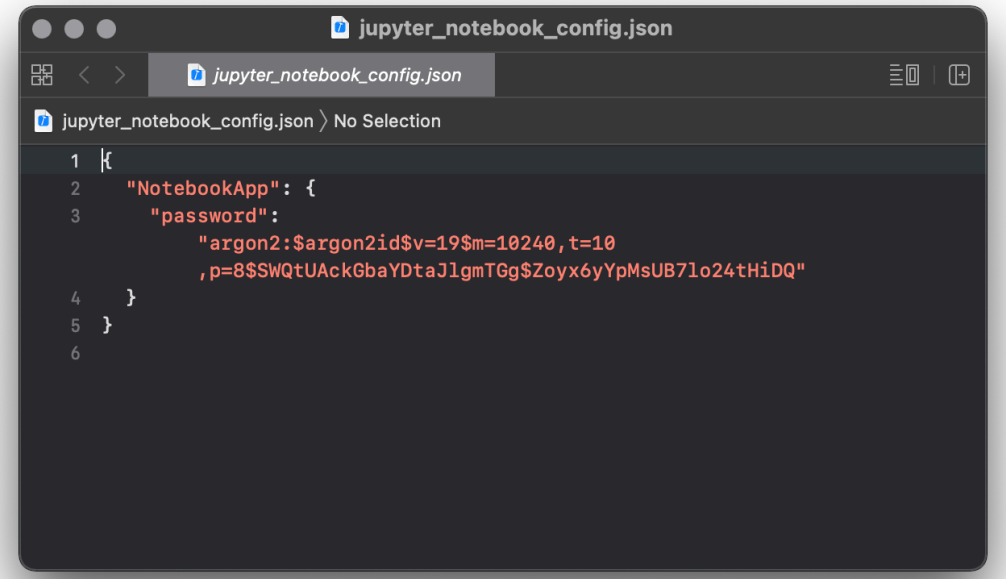
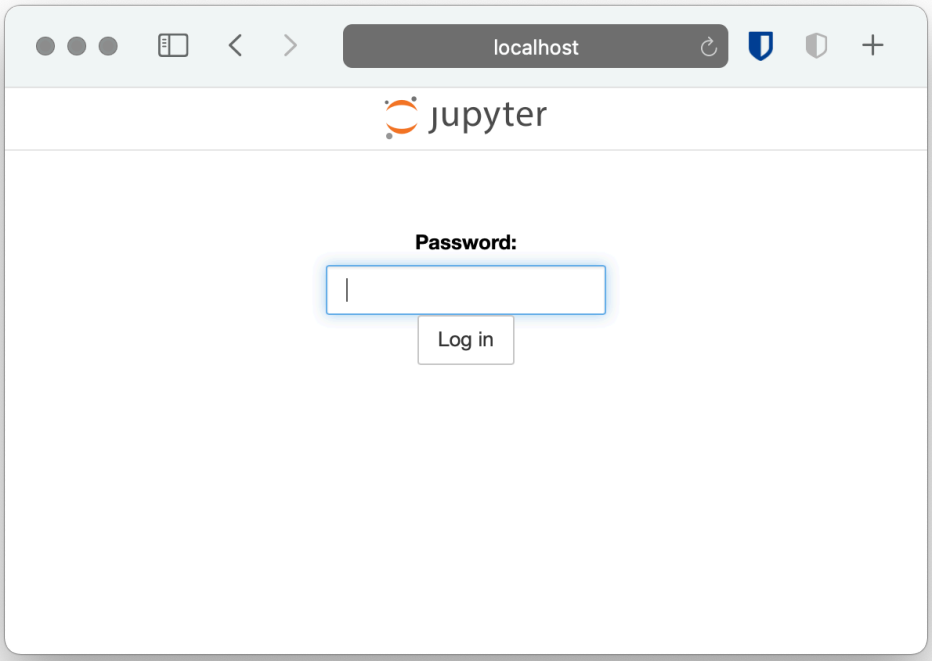


Figure 19: The Jupyter Notebook User Interface Password Prompt

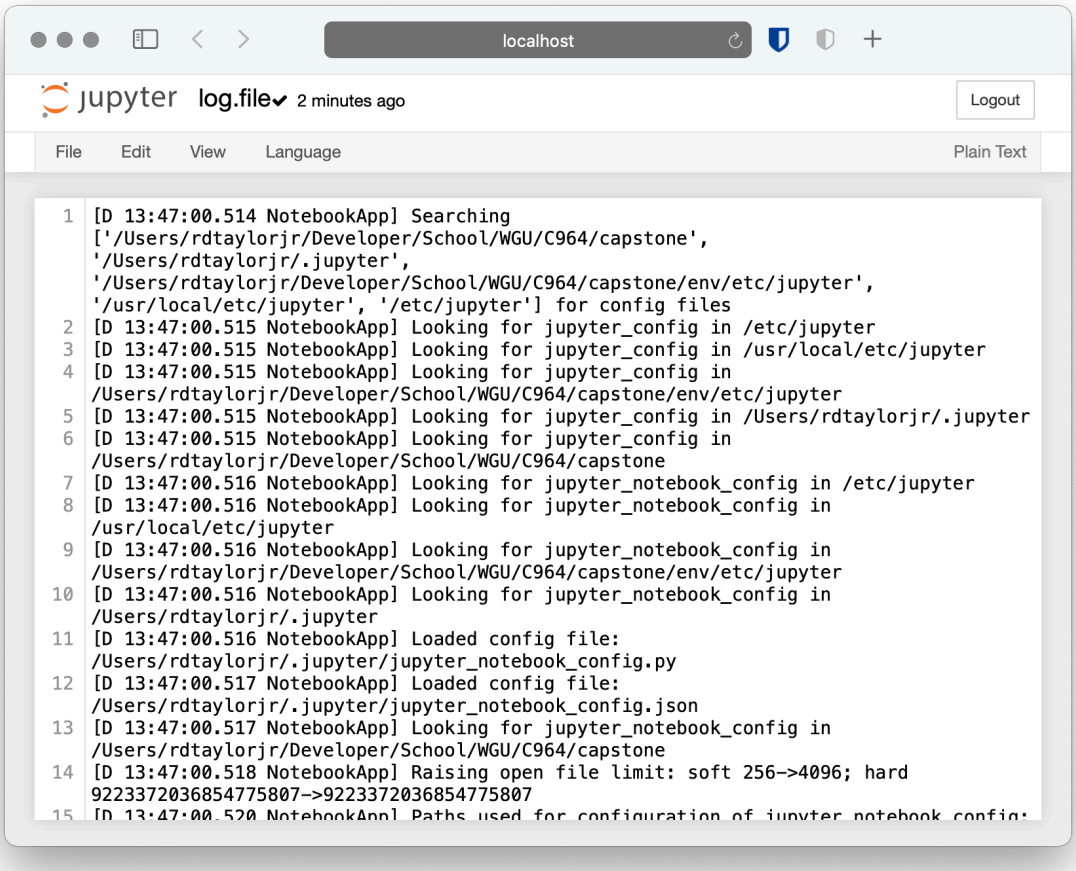


11. PRODUCT HEALTH MONITORING

Include functionality that will enable the application's "health" and reliability to be monitored. It should answer the questions, "Is the application performing correctly after being deployed?" For example, the use of displays or logs to quickly discover, isolate and solve problems that can negatively affect the application's performance and accuracy. If the data continues to accumulate, is it still accurate? If the code gets corrupted and needs to be fixed, is there a debugger to address that? With a system that evaluates the statistical or the practical significance of the product, there should be a monitoring tool.

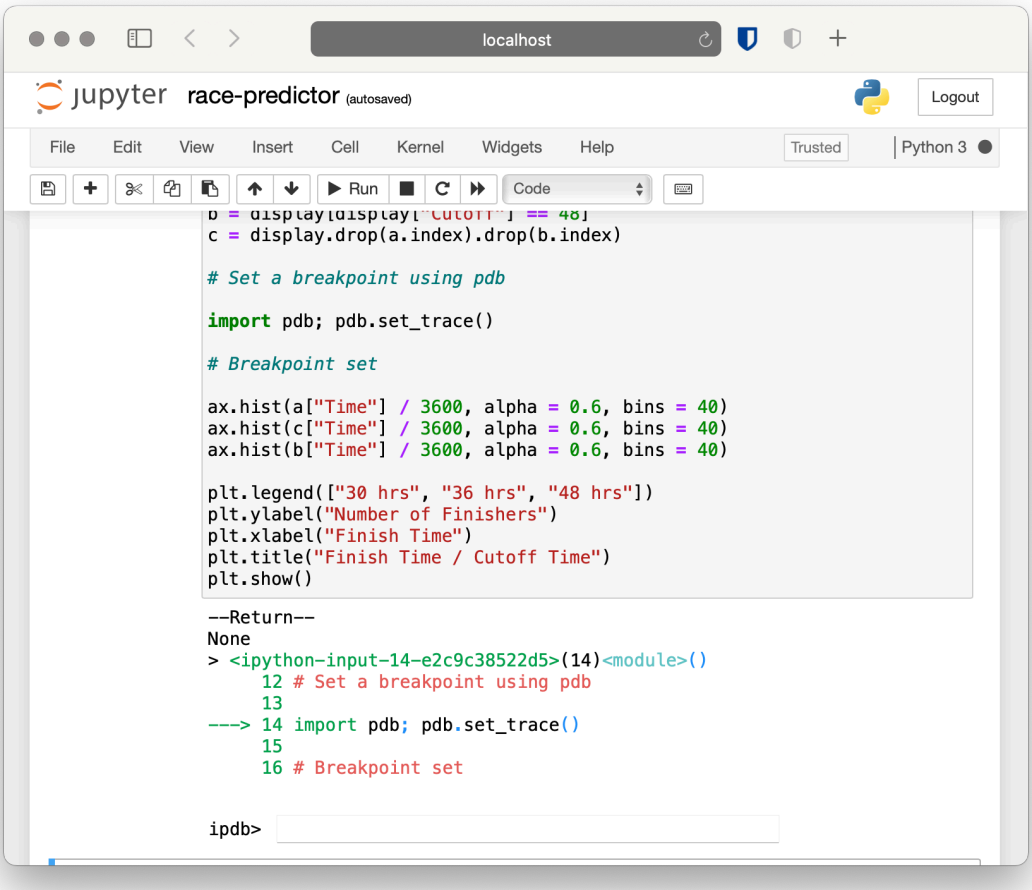
The health of the app may be monitored by accessing the **Jupyter Notebook Debug Log** via the command line with the command `jupyter notebook --debug > log.file 2>&1`. This command creates and stores a log file named log.file in the notebook's env directory. The log file confirms that the application is performing correctly after being deployed, and tracks any errors that may occur. This allows the user to quickly discover, isolate and solve problems with the Jupyter Notebook itself (https://stackoverflow.com/questions/33632529/how-to-enable-and-access-debug-logging-for-notebook-and-ipython-kernel).

Figure 20: The Jupyter Notebook Debug Log File



For debugging code that gets corrupted and needs to be fixed, we have a few different options available, including Python's `IPython.core.debugger.set_trace` and `IPython.core.debugger.Tracer` classes. The most convenient approach, and the one used to debug this project is **Python's built-in `pdb` debugger** which allows us to set a breakpoint and create an interactive debugging prompt. That prompt allows us to "print our variables, evaluate code to inspect the current stack, etc" (https://davidhamann.de/2017/04/22/debugging-jupyter-notebooks).

Figure 21: Setting a Breakpoint Using Python's `pdb` Debugger



12. DASHBOARD

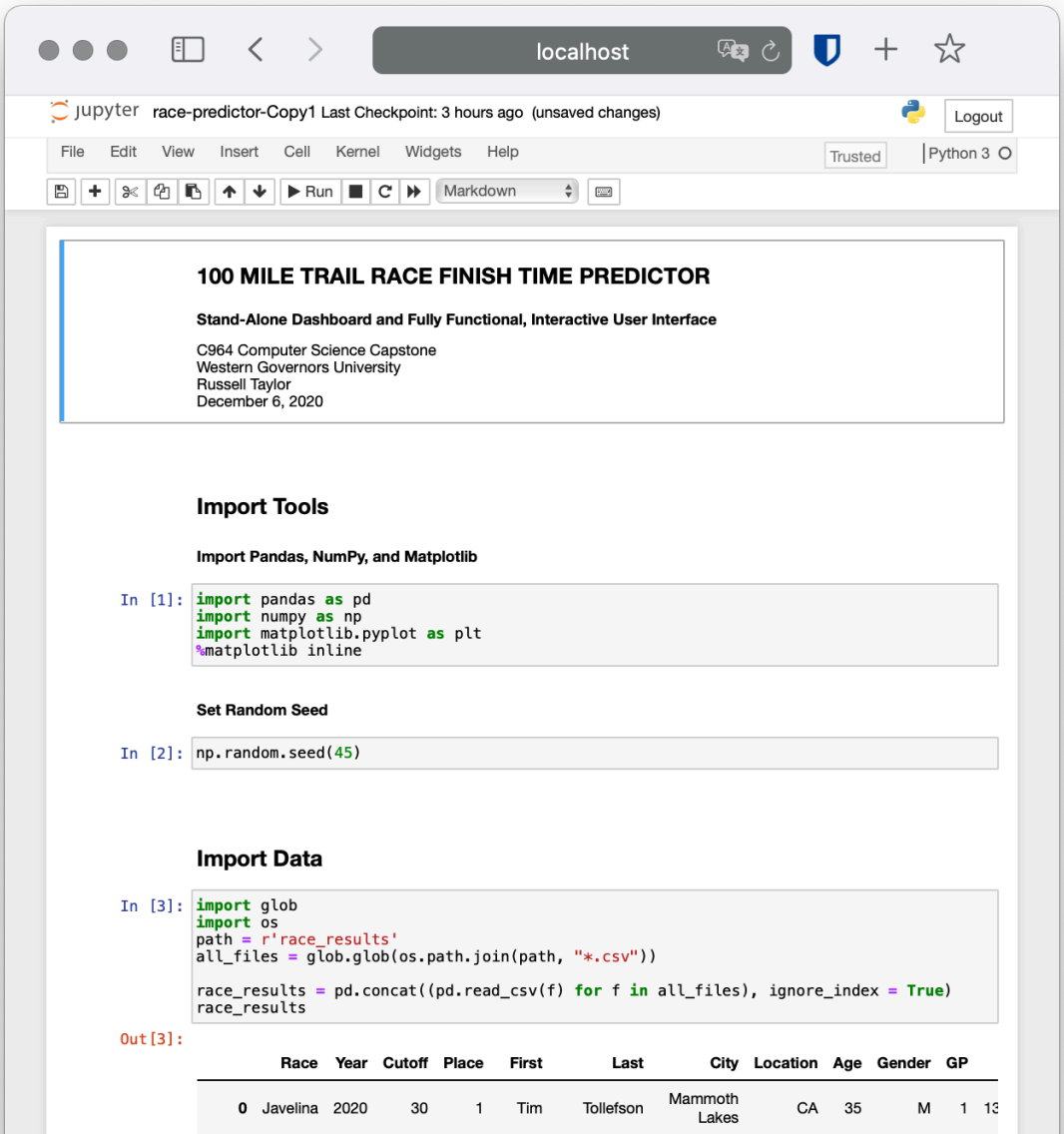
Include a user-friendly, functional dashboard that enables the query and display of the data, as well as other functionality described in this section. This could be stand-alone, Web-based, or a mobile application interface.

Since the end user of this initial version of the race predictor app is the technical team at UltraSignup.com, the appropriate user-friendly, functional dashboard for this project is the

Jupyter Notebook. This is **a stand-alone interface** that provides a significant amount of flexibility and user interaction with both the data and the code used to generate Machine Learning models. The Jupyter Notebook allows the user to query and/or manipulate the data in any way they see fit. Additionally, the user-friendly layout and workflow of Jupiter Notebooks has allowed me to clearly format and present both the data and the code in a way that is eminently readable and useable. Each section of code may be modified and run in real time by the user. All of the graphical elements for this project, namely the Matplotlib interactive figures, are included in the Jupiter Notebook and can be viewed and modified by the user with a significant amount of flexibility.

Screenshots of the Jupyter Notebook stand-alone dashboard are shown in Figure 22.

Figure 22: The Jupyter Notebook Stand-Alone Dashboard



1	Javelina	2020	30	2	Nick	Coury	Scottsdale	AZ	33	M	2	14
2	Javelina	2020	30	3	Nicole	Bitter	Phoenix	AZ	38	F	1	15
3	Javelina	2020	30	4	Darce	Claus	Bonners Ferry	ID	49	F	2	15
4	Javelina	2020	30	5	Sean	Van Horn	Carbondale	CO	34	M	3	16
...	...	...	...	...	...	...	...	...	...	...	...	
12269	Angeles Crest	2015	33	94	Peter	Hulbert	Edina	MN	36	M	74	32
12270	Angeles Crest	2015	33	95	Tim	Steele	San Ramon	CA	53	M	75	32
12271	Angeles Crest	2015	33	96	John	Kawaharada	Los Angeles	CA	47	M	76	32
12272	Angeles Crest	2015	33	97	Ryan	Launder	Landers	CA	39	M	77	32
12273	Angeles Crest	2015	33	98	Susan	Owen-Mccollum	La Canada-Flintridge	CA	45	F	21	32

12274 rows × 13 columns

Clean Data

Convert Finish Times to Integers

In [4]:
```
race_results["Time"] = pd.to_timedelta(race_results["Time"]).dt.total_seconds()
```

Remove DNFs and DNSs

In [5]:
```
race_results = race_results[race_results["Place"] > 0]
len(race_results)
```

Out[5]: 11679

Remove Erroneous Finish Times

In [6]:
```
race_results = race_results[race_results["Time"] > 43200]
outlier = race_results[race_results["Race"] == "Cascade Crest"]
race_results.drop(outlier[outlier["Time"] > 144000].index, inplace = True)
len(race_results)
```

Out[6]: 11667

Fill Null Fields

In [7]:
```
race_results.fillna("INTL", inplace = True)
```

Descriptive Data Method: Randomized PCA

Remove Extraneous Columns

In [8]:
```
df = race_results.drop(["First", "Last", "City", "Place", "GP"],
                       axis = 1)
list(df.columns)
```

Out[8]: ['Race', 'Year', 'Cutoff', 'Location', 'Age', 'Gender', 'Time', 'Rank']

Convert Categorical Features to Numerical Form

In [9]:
```
df["Race"] = pd.Categorical(df["Race"])
df["Race"] = df["Race"].cat.codes
df["Location"] = pd.Categorical(df["Location"])
df["Location"] = df["Location"].cat.codes
df["Gender"] = pd.Categorical(df["Gender"])
df["Gender"] = df["Gender"].cat.codes
```

Split the Data into Features and Labels

In [10]:
```
df_shuffled = df.sample(frac = 1)
x = df_shuffled.drop("Time", axis = 1)
y = df_shuffled["Time"]
```

Split the Data into Training and Test Sets

In [11]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.2,
                                                    random_state = 0)
```

Normalize the Feature Set with Standard Scalar Normalization

In [12]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Run the Randomized PCA Algorithm

In [13]:
```
from sklearn.decomposition import PCA

pca = PCA()
x_train = pca.fit_transform(x_train)
x_test = pca.transform(x_test)
```

Display Explained Variance

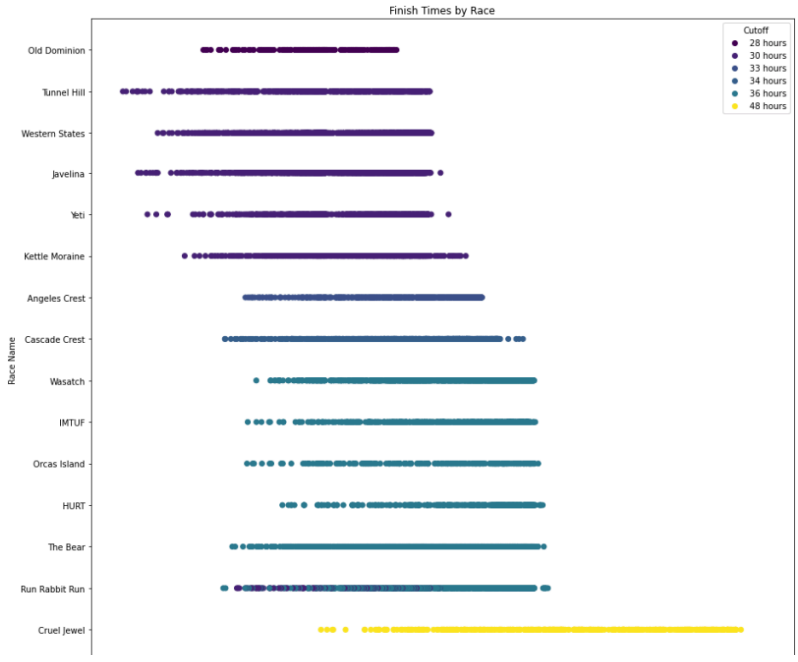
In [14]:
```
explained_variance = pca.explained_variance_ratio_
explained_variance
```

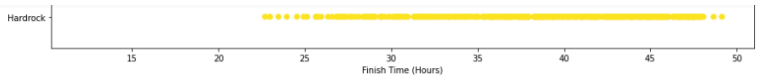
Out[14]:
```
array([0.20713157, 0.20398593, 0.1534491 , 0.13584216, 0.12970981,
       0.08996371, 0.07991773])
```

Display Data

Display Finish Times by Race

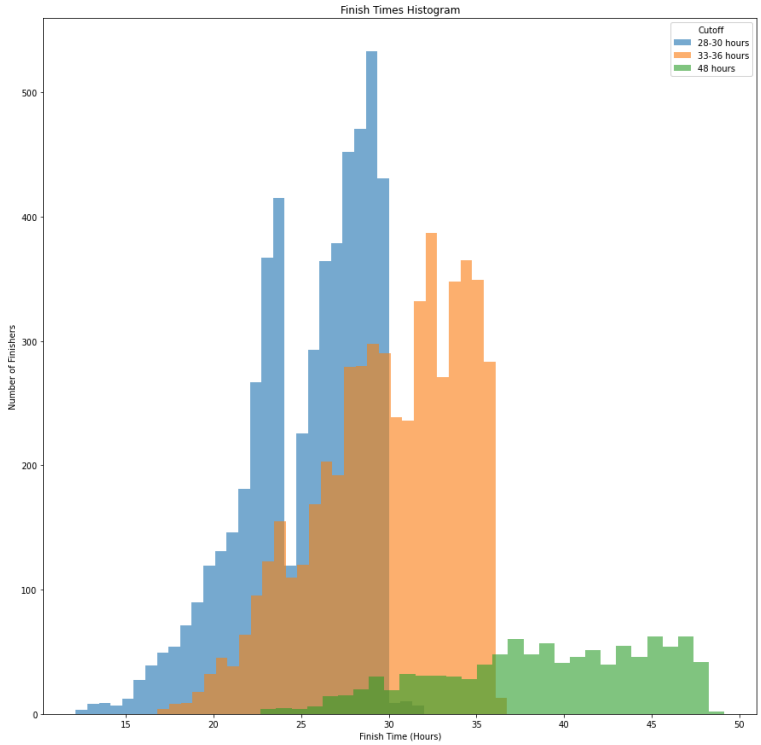
In [15]:
```
display = race_results.sort_values(by = "Time", ascending = False)
fig, ax = plt.subplots(figsize = (15, 15))
scatter = ax.scatter(display["Time"] / 3600,
                     display["Race"],
                     c = display["Cutoff"])
ax.set(title = "Finish Times by Race",
       xlabel = "Finish Time (Hours)",
       ylabel = "Race Name")
plt.legend(scatter.legend_elements()[0],
           ("28 hours", "30 hours", "33 hours", "34 hours", "36 hours", "48 hours"),
           title = "Cutoff")
fig.savefig("images/finish_times_by_race.png")
plt.show()
```





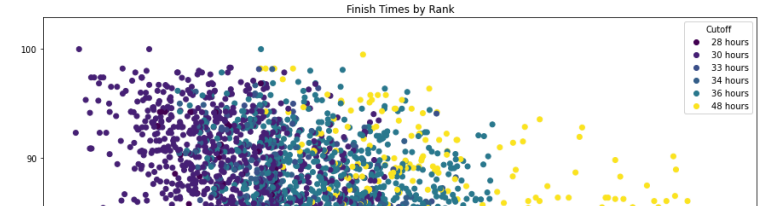
Display Finish Times Histogram

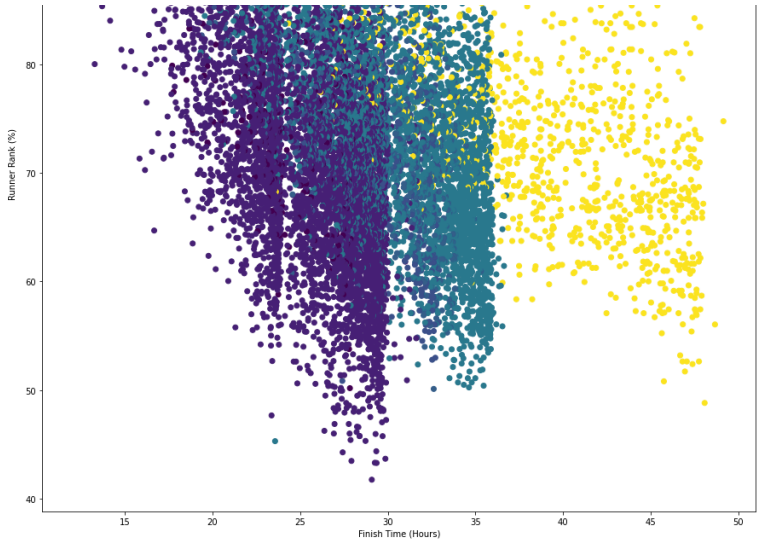
In [16]:
```
display = race_results
fig, ax = plt.subplots(figsize = (15, 15))

a = display[display["Cutoff"] == 30]
b = display[display["Cutoff"] == 48]
c = display.drop(a.index).drop(b.index)

ax.hist(a["Time"] / 3600, alpha = 0.6, bins = 30)
ax.hist(c["Time"] / 3600, alpha = 0.6, bins = 30)
ax.hist(b["Time"] / 3600, alpha = 0.6, bins = 30)

plt.legend(["28-30 hours", "33-36 hours", "48 hours"], title = "Cutoff")
plt.ylabel("Number of Finishers")
plt.xlabel("Finish Time (Hours)")
plt.title("Finish Times Histogram")
fig.savefig("images/finish_times_histogram.png")
plt.show()
```



Display Finish Times by Runner Rank

In [17]:
```
display = race_results.sort_values(by = "Time", ascending = False)
fig, ax = plt.subplots(figsize = (15, 15))
scatter = ax.scatter(display["Time"] / 3600,
                     display["Rank"],
                     c = display["Cutoff"])
ax.set(title = "Finish Times by Rank",
       xlabel = "Finish Time (Hours)",
       ylabel = "Runner Rank (%)")
plt.legend(scatter.legend_elements()[0],
           ("28 hours", "30 hours", "33 hours", "34 hours", "36 hours", "48 hours"),
           title = "Cutoff")
fig.savefig("images/finish_times_by_rank.png")
plt.show()
```





Non-Descriptive Data Method: Random Forest Regression

Remove Extraneous Columns

In [18]:
```
df = race_results.drop(["First", "Last", "Location", "City", "Place", "GP"],
                       axis = 1)
list(df.columns)
```

Out[18]:
```
['Race', 'Year', 'Cutoff', 'Age', 'Gender', 'Time', 'Rank']
```

Convert Categorical Features to Numerical Form

In [19]:
```
df["Race"] = pd.Categorical(df["Race"])
df["Race"] = df["Race"].cat.codes
df["Gender"] = pd.Categorical(df["Gender"])
df["Gender"] = df["Gender"].cat.codes
```

Split the Data into Features and Labels

In [20]:
```
df_shuffled = df.sample(frac = 1)
x = df_shuffled.drop("Time", axis = 1)
y = df_shuffled["Time"]
```

Convert Categorical Features into Numerical Form

In [21]:
```
# from sklearn.preprocessing import OneHotEncoder
# from sklearn.compose import ColumnTransformer

# categorical_features = ["Gender"]
# one_hot = OneHotEncoder()
# transformer = ColumnTransformer([("one_hot", one_hot, categorical_features)],
#                                 remainder = "passthrough")
# x = transformer.fit_transform(x)
```

Split the Data into Training and Test Sets

In [22]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

Run the Random Forest Regression Algorithm

In [23]:
```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor()
model.fit(x_train, y_train)
```

Out[23]:
```
RandomForestRegressor()
```

Evaluate Results

Evaluate Training Set: R² Coefficient of Determination

In [24]:
```
model.score(x_train, y_train)
```

Out[24]:
```
0.9697265031538366
```

Evaluate Test Set: R² Coefficient of Determination

In [25]:
```
model.score(x_test, y_test)
```

Out[25]:
```
0.7953309399703318
```

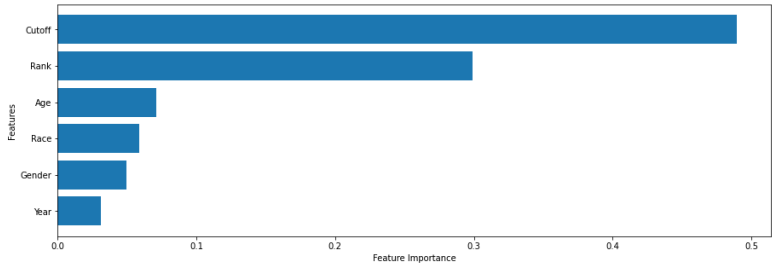
Calculate the Mean Absolute Error

In [26]:
```
from sklearn.metrics import mean_absolute_error

y_preds = model.predict(x_test)
mae = mean_absolute_error(y_test, y_preds)

pd.to_timedelta(mae, unit = "s")
```

Out[26]:
```
Timedelta('0 days 02:00:43.774684631')
```

Display Feature Importance

In [27]:
```
def plot_features(columns, importances):
    df = (pd.DataFrame({"features": columns,
                        "feature_importances": importances})
          .sort_values("feature_importances")
          .reset_index(drop = True))

    # Plot the DataFrame
    fig, ax = plt.subplots(figsize = (15, 5))
    ax.barh(df["features"], df["feature_importances"])
    ax.set_ylabel("Features")
    ax.set_xlabel("Feature Importance")

    fig.savefig("images/feature_importance.png")

plot_features(x_train.columns, model.feature_importances_)
```



Display Results

Create Results DataFrame

In [28]:
```
df = x_test.copy()
df["Actual"] = y_test / 3600
df["Predicted"] = y_preds / 3600
df["Difference"] = df["Predicted"] - df["Actual"]
df["AbsDifference"] = abs(df["Difference"])
df
```

Out[28]:

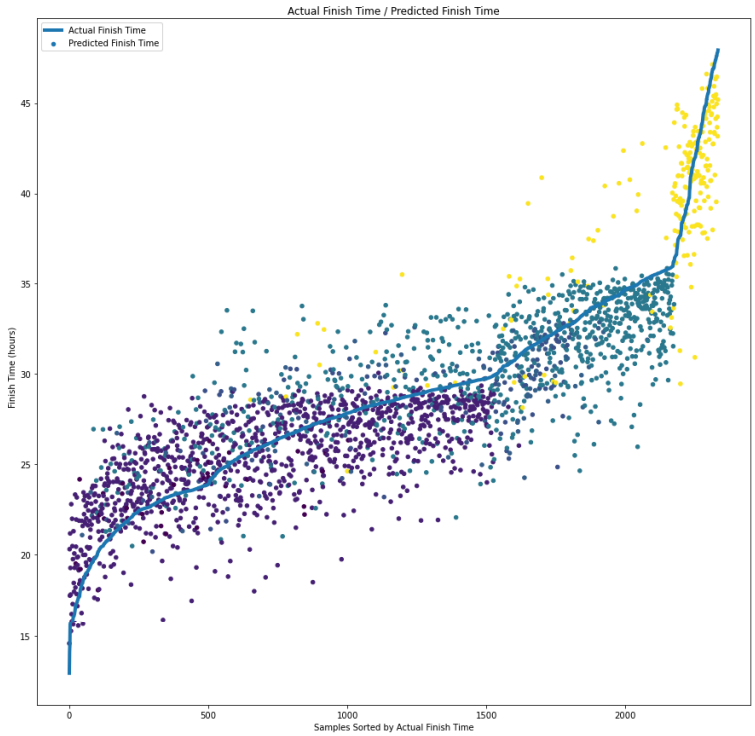
	Race	Year	Cutoff	Age	Gender	Rank	Actual	Predicted	Difference	AbsDifference
3319	12	2016	30	64	1	52.33	26.750833	28.273347	1.522514	1.522514
8977	10	2016	36	22	1	94.29	21.636667	21.885636	0.248969	0.248969
7498	13	2018	36	48	0	81.54	32.303056	33.342919	1.039864	1.039864
899	6	2018	30	33	0	74.39	28.665556	26.206964	-2.458592	2.458592
10053	1	2018	34	62	1	67.85	31.195556	30.069789	-1.125767	1.125767
...	...	...	...	...	...	...	...	...	...	...
9684	5	2017	36	37	1	70.38	29.196389	32.929372	3.732983	3.732983

8054	13	2015	36	52	1	64.62	33.803056	33.890519	0.087464	0.087464
2453	9	2018	36	30	0	80.66	29.298889	33.197044	3.898156	3.898156
2230	7	2014	30	37	1	72.78	23.195000	24.969694	1.774694	1.774694
5788	14	2018	30	41	1	95.81	22.067778	18.344278	-3.723500	3.723500

2334 rows × 10 columns

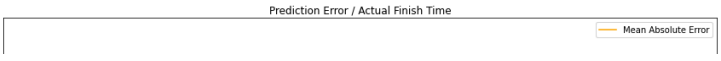
Display Actual Finish Time / Predicted Finish Time

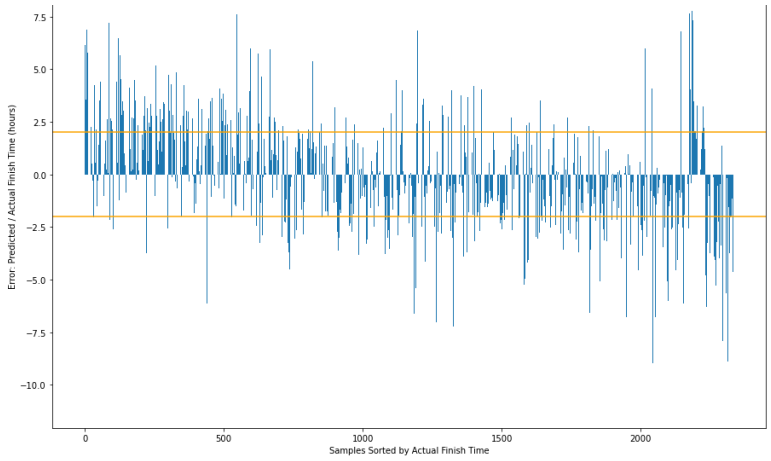
In [29]:
```
display = df.sort_values(by = ["Actual"])
display.reset_index(drop = True, inplace = True)
fig, ax = plt.subplots(figsize = (15, 15))
ax.plot(display.index,
        display["Actual"],
        linewidth = 4,
        label = "Actual Finish Time")

ax.scatter(display.index,
           display["Predicted"],
           s = 20,
           c = display["Cutoff"],
           label = "Predicted Finish Time")

ax.set(title = "Actual Finish Time / Predicted Finish Time",
       xlabel = "Samples Sorted by Actual Finish Time",
       ylabel = "Finish Time (hours)")
plt.legend()
fig.savefig("images/actual_predicted_finish_time.png")
plt.show()
```



Display Mean Absolute Error

In [30]:
```
display = df.sort_values(by = ["Actual"])
display.reset_index(drop = True, inplace = True)
fig, ax = plt.subplots(figsize = (15, 10))

ax.bar(display.index,
       display["Difference"])

plt.axhline(y = mae / 3600, color = "Orange", label = "Mean Absolute Error")
plt.axhline(y = -(mae / 3600), color = "Orange")

ax.set(title = "Prediction Error / Actual Finish Time",
       xlabel = "Samples Sorted by Actual Finish Time",
       ylabel = "Error: Predicted / Actual Finish Time (hours)")
plt.legend()
fig.savefig("images/mean_absolute_error.png")
plt.show()
```





Predict New Race Finish Time

Input Values

In [31]:
```
year = 2020
cutoff = 36
age = 36
gender = 1
rank = 65.23
race = 9
```

Predict Finish Time

In [32]:
```
df = (pd.DataFrame({"Year": [year],
                    "Cutoff": [cutoff],
                    "Age": [age],
                    "Gender": [gender],
                    "Rank": [rank],
                    "Race": [race]}))
pred = model.predict(df)
prediction = pd.to_timedelta(pred.round(0), unit = "s")
prediction[0]
```

Out[32]:
```
Timedelta('1 days 03:33:48')
```

Save and Load Trained Model

Save Trained Model

In [33]:
```
import pickle
```

In [34]:
```
pickle.dump(model, open("models/race_predictor_random_forest_model.pkl", "wb"))
```

Load Trained Model

In [35]:
```
aded_pickle_model = pickle.load(open("models/race_predictor_random_forest_model.pkl", "
aded_pickle_model.score(x_test, y_test)
```

Out[35]:
```
0.7953309399703318
```

D. POST-IMPLEMENTATION REPORT

Create a post-implementation report that addresses the various aspect of the project, providing sufficient detail so a reader, unfamiliar with the project, can understand what you have accomplished.

1. PROJECT PURPOSE

Reiterate the business or organization problem that this application solved. How did the application address the "vision" or expectations of the client? Summarizes the technical functionality and end-user requirements that were met.

The purpose of this project is to use Machine Learning to predict runner finish times for 100 mile trail races. Currently, UltraSignup.com uses simple arithmetic to calculate rough estimations of a runner's finish time, but these estimations are notoriously inaccurate. A more accurate predictive model will allow for more accurate estimations which will provide value to UltraSignup.com's customers. Those customers include Race Directors and runners. The demand for accurate race finish time predictions is well documented, and there currently does not exist in the market a robust predictive model for 100 mile trail race finish times. Such a model will have many immediate benefits including enhanced runner safety, the potential for more targeted training and race-day preparation, increased finisher rates, and greater notoriety for UltraSignup.com in the trail running community. Each of these will lead to increased revenues for UltraSignup.com.

This solution has been implemented in Python using its robust data libraries including Anaconda, Jupyter Notebooks, Pandas, NumPy, Matplotlib, and Scikit-Learn. A working prototype has successfully been created that uses a subset of UltraSignup.com's available 100 mile trail race data. Both a descriptive Principle Component Analysis (PCA) algorithm and a predictive Random Forest Regression algorithm have been trained and tested on the provided dataset. The result is a functioning Machine Learning model that can predict a runner's finish time for a given 100 mile trail race with 79.5% confidence and to within an average of 2 hours of the runner's actual finish time.

2. DATASETS

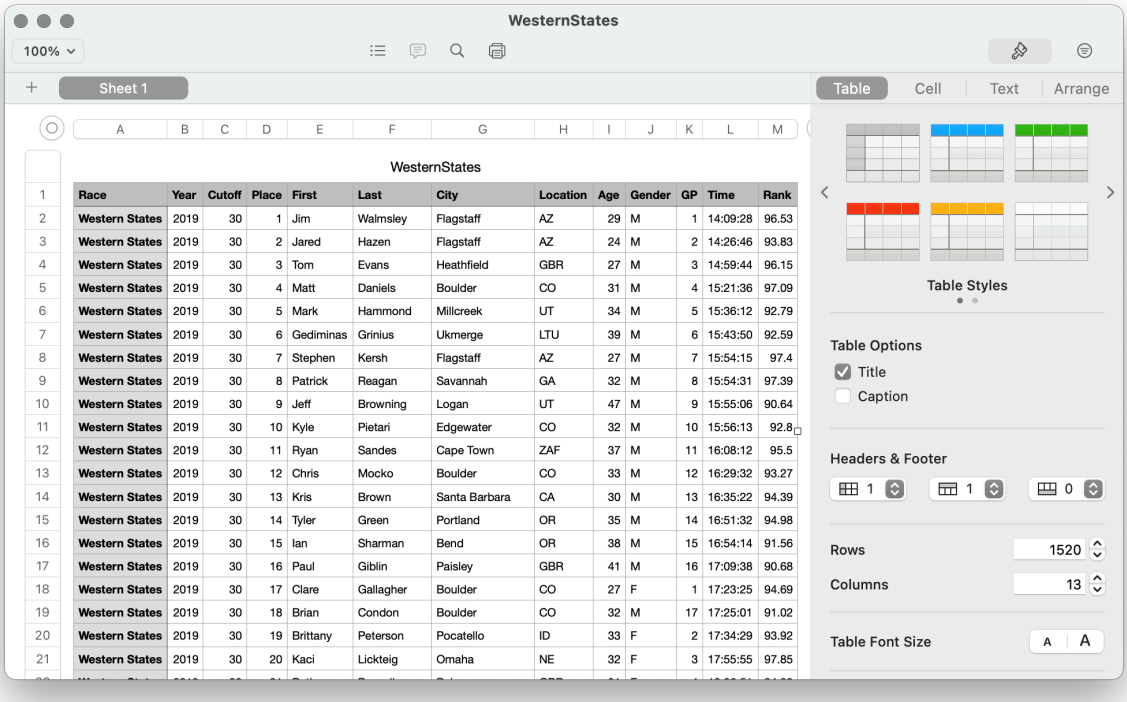
Provide examples of the raw datasets used by the application. If applicable, also include the cleaned datasets along with a sample of the code (or executable file) used to clean the data. If the data was cleaned or manipulated in any way, provide explanations of the interventions. If not, state so and why.

The dataset used to create the prototype Machine Learning model was taken from UltraSignup.com's Race Results webpages and includes the most recent **5 years of available results** for **16 different 100 mile trail races**. The raw data contains over 14,000 individual race results and the parsed and cleaned data contains **11,667 individual race results**.

The raw dataset consists of 16 **comma-separated-values (csv) files**. These files may be found in the 'race_results' folder delivered with this project. The 16 races and corresponding raw data csv files are as follows:

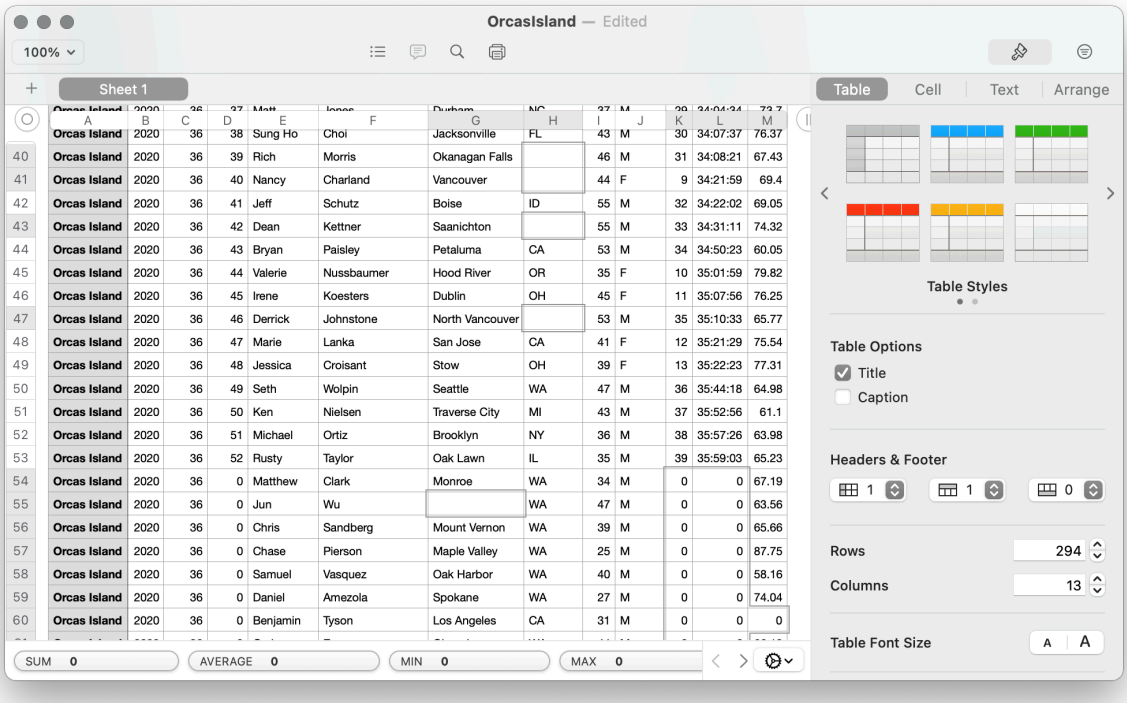
Angeles Crest 100 Mile Endurance Run	AngelesCrest.csv
Cascade Crest 100 Mile Endurance Run	CascadeCrest.csv
Cruel Jewel 100	CruelJewel.csv
HardRock 100 Endurance Run	HardRock.csv
HURT 100	HURT.csv
IMTUF (Idaho Mountain Trail Ultra Festival) 100	IMTUF.csv
The Javelina Jundred	JavelinaJundred.csv
Kettle Moraine 100 Endurance Trail Races	KettleMoraine.csv
The Old Dominion 100 Mile Cross Country Run	OldDominion.csv
Orcas Island 100 Miler	OrcasIsland.csv
Run Rabbit Run 100 Mile Race	RunRabbitRun.csv
The Bear 100 Endurance Run	TheBear.csv
Tunnel Hill 100	TunnelHill.csv
Wasatch Front 100 Mile Endurance Run	WasatchFront.csv
Western States 100 Mile Endurance Run	WesternStates.csv
The Yeti 100 Mile Endurance Run	Yeti.csv

Figure 23: Western States 100 Mile Endurance Run Raw Data



	Race	Year	Cutoff	Place	First	Last	City	Location	Age	Gender	GP	Time	Rank
2	Western States	2019	30	1	Jim	Walmsley	Flagstaff	AZ	29	M	1	14:09:28	96.53
3	Western States	2019	30	2	Jared	Hazen	Flagstaff	AZ	24	M	2	14:26:46	93.83
4	Western States	2019	30	3	Tom	Evans	Heathfield	GBR	27	M	3	14:59:44	96.15
5	Western States	2019	30	4	Matt	Daniels	Boulder	CO	31	M	4	15:21:36	97.09
6	Western States	2019	30	5	Mark	Hammond	Millcreek	UT	34	M	5	15:36:12	92.79
7	Western States	2019	30	6	Gediminas	Grinius	Ukmerge	LTU	39	M	6	15:43:50	92.59
8	Western States	2019	30	7	Stephen	Kersh	Flagstaff	AZ	27	M	7	15:54:15	97.4
9	Western States	2019	30	8	Patrick	Reagan	Savannah	GA	32	M	8	15:54:31	97.39
10	Western States	2019	30	9	Jeff	Browning	Logan	UT	47	M	9	15:55:06	90.64
11	Western States	2019	30	10	Kyle	Pietari	Edgewater	CO	32	M	10	15:56:13	92.8
12	Western States	2019	30	11	Ryan	Sandes	Cape Town	ZAF	37	M	11	16:08:12	95.5
13	Western States	2019	30	12	Chris	Mocko	Boulder	CO	33	M	12	16:29:32	93.27
14	Western States	2019	30	13	Kris	Brown	Santa Barbara	CA	30	M	13	16:35:22	94.39
15	Western States	2019	30	14	Tyler	Green	Portland	OR	35	M	14	16:51:32	94.98
16	Western States	2019	30	15	Ian	Sharman	Bend	OR	38	M	15	16:54:14	91.56
17	Western States	2019	30	16	Paul	Giblin	Paisley	GBR	41	M	16	17:09:38	90.68
18	Western States	2019	30	17	Clare	Gallagher	Boulder	CO	27	F	1	17:23:25	94.69
19	Western States	2019	30	18	Brian	Condon	Boulder	CO	32	M	17	17:25:01	91.02
20	Western States	2019	30	19	Brittany	Peterson	Pocatello	ID	33	F	2	17:34:29	93.92
21	Western States	2019	30	20	Kaci	Lickteig	Omaha	NE	32	F	3	17:55:55	97.85

Figure 24: Orcas Island 100 Miler Raw Data with Missing and Null Fields



	A	B	C	D	E	F	G	H	I	J	K	L	M
40	Orcas Island	2020	36	39	Rich	Morris	Okanagan Falls		46	M	31	34:08:21	67.43
41	Orcas Island	2020	36	40	Nancy	Charland	Vancouver		44	F	9	34:21:59	69.4
42	Orcas Island	2020	36	41	Jeff	Schutz	Boise	ID	55	M	32	34:22:02	69.05
43	Orcas Island	2020	36	42	Dean	Kettner	Saanichton		55	M	33	34:31:11	74.32
44	Orcas Island	2020	36	43	Bryan	Paisley	Petaluma	CA	53	M	34	34:50:23	60.05
45	Orcas Island	2020	36	44	Valerie	Nussbaumer	Hood River	OR	35	F	10	35:01:59	79.82
46	Orcas Island	2020	36	45	Irene	Koesters	Dublin	OH	45	F	11	35:07:56	76.25
47	Orcas Island	2020	36	46	Derrick	Johnstone	North Vancouver		53	M	35	35:10:33	65.77
48	Orcas Island	2020	36	47	Marie	Lanka	San Jose	CA	41	F	12	35:21:29	75.54
49	Orcas Island	2020	36	48	Jessica	Croisant	Stow	OH	39	F	13	35:22:23	77.31
50	Orcas Island	2020	36	49	Seth	Wolpin	Seattle	WA	47	M	36	35:44:18	64.98
51	Orcas Island	2020	36	50	Ken	Nielsen	Traverse City	MI	43	M	37	35:52:56	61.1
52	Orcas Island	2020	36	51	Michael	Ortiz	Brooklyn	NY	36	M	38	35:57:26	63.98
53	Orcas Island	2020	36	52	Rusty	Taylor	Oak Lawn	IL	35	M	39	35:59:03	65.23
54	Orcas Island	2020	36	0	Matthew	Clark	Monroe	WA	34	M	0	0	67.19
55	Orcas Island	2020	36	0	Jun	Wu		WA	47	M	0	0	63.56
56	Orcas Island	2020	36	0	Chris	Sandberg	Mount Vernon	WA	39	M	0	0	65.66
57	Orcas Island	2020	36	0	Chase	Pierson	Maple Valley	WA	25	M	0	0	87.75
58	Orcas Island	2020	36	0	Samuel	Vasquez	Oak Harbor	WA	40	M	0	0	58.16
59	Orcas Island	2020	36	0	Daniel	Amezola	Spokane	WA	27	M	0	0	74.04
60	Orcas Island	2020	36	0	Benjamin	Tyson	Los Angeles	CA	31	M	0	0	0

3. DATA PRODUCT CODE

Review the functionality of the code used to perform the analysis of the data and how it provided the necessary functionality of descriptive and predictive outputs. You will also need to submit the entire, functional source code.

The entire, functional source code may be found in the included file 'race-predictor.ipynb'. Screenshots may also be viewed in Section C12, Figure 22 above. The product code consists of ten parts:

1. **Import Tools**. The needed Python data libraries are imported into the Jupyter Notebook including Pandas, NumPy, and Matplotlib. Additionally, a random seed is set for use later in the project by the Scikit-Learn algorithms.

2. **Import Data**. The raw csv data files are imported into a Pandas DataFrame. The Python `glob()` method is used to recursively find the data files, then the contents of each file is concatenated to the end of the DataFrame. The result is a single DataFrame containing the entire dataset.

3. **Clean Data**. Three types of anomalies are present in the raw dataset: incompatible data types, records which contain missing or erroneous data, fields which contain missing, erroneous, or extraneous data. First, the finish times are converted from strings to Timedeltas and from Timedeltas to integers, and the remaining string fields are converted to categorical features and then to numerical datatypes using `cat.codes`. Next, records with missing or erroneous finish times are eliminated from the dataset. Finally, fields with null values, such as the Location for international runners, are filled.

4. **Descriptive Data Method: Randomized PCA**. Principle Component Analysis is used as a preprocessing step for the predictive algorithm and aims to reduce the number of features needed in order to produce accurate predictive results with maximum efficiency. First, the data is split into labels and features. Features are the input data columns or 'x' variables, and labels are the output or predicted data columns or 'y' variables. Next the data is split into training and test sets with 80% of the data allocated to the training set and 20% allocated to the test set. Then the feature set is normalized with standard scaler normalization so that the mean of the data points is 0 and the variance is 1. Then the PCA algorithm is fit to the data and transforms it using the Scikit-Learn `PCA` class and `fit_transform()` and `transform()` methods. The output of the PCA algorithm is a set of Principle Components which may be used as input for the predictive algorithm. Finally, the explained variance of each Principle Component is displayed.

5. **Display Data**. Three descriptive visualizations are created from the Pandas DataFrame using Matplotlib in order to analyze the existing data and determine the best approach to predictive modeling for this data's unique features. For each visualization, the data is sorted if needed, figures and axes are created using PyPlot's object-oriented method, then the specific graph type is initialized with the appropriate variables including heat-mapping, the axis and figure labels are set, an image file of the figure is saved, and the figure is displayed inline in the Jupyter Notebook.

6. **Non-Descriptive Data Method: Random Forest Regression**. Random Forest Regression is used to predict finish times based on a set of feature variables. First, the data is split into labels and features. Features are the input data columns or 'x' variables, and labels are the output or predicted data columns or 'y' variables. Next the data is split into training and test sets with 80% of the data allocated to the training set and 20% allocated to the test set. Then the Random Forest Regression algorithm is fit to the data using the Scikit-Learn `RandomForestRegressor` class and `fit()` method. The output of the Random Forest Regression algorithm is a set of predicted finish times for each sample in the test dataset.

7. **Evaluate Results**. The results of the predictive algorithm are evaluated using three different metrics. The first is the R^2 coefficient of determination. This is implemented using the Scikit-Learn `score()` method. The second is the mean absolute error, implemented using the `mean_absolute_error()` method. The third is feature importance, implemented using the `feature_importances_` value.

8. **Display Results**. In order to display the results of the predictive model, a new DataFrame is created which contains the original features, the actual finish times, and the predicted finish times for the test dataset. Two visualizations are used to display the results. For each visualization, the data is sorted if needed, figures and axes are created using PyPlot's object-oriented method, then the specific graph type is initialized with the appropriate variables including heat-mapping, the axis and figure labels are set, an image file of the figure is saved, and the figure is displayed inline in the Jupyter Notebook.

9. **Predict New Race Finish Time**. An interface is provided in the Jupyter Notebook for the user to input any values for the required feature variables, and receive as output the corresponding finish time prediction from the predictive Machine Learning model. First the input values are defined and stored in a DataFrame. Then the predictive model is run using the DataFrame as input, and the finish time prediction is produced.

10. **Save and Load Trained Model**. A last step is to save the working Machine Learning model to file for external use using Pickle, and to test that the Pickle model can be reimported and runs properly.

4. HYPOTHESIS VERIFICATION

Discuss if and why the initially established project hypothesis was accepted or rejected based on the use of the application. Did your assumptions about the phenomenon included (described) in the data prove true or not. Rejected hypothesis are okay but explain why that was so.

This project's **stated hypothesis** in the Non-Technical Project Recommendation document was that "if a Machine Learning model is provided with a runner's location, age, gender, UltraSignup.com rank, and the cutoff time for the chosen race, then the model can predict the finish time for that runner to within 1 hour of the runner's actual finish time on average" (Section A5 above).

This hypothesis has been **accepted** in whole except for the specific requirement that the Mean Absolute Error be less than 1 hour. The actual Mean Absolute Error for this project's predictive algorithm currently stands at twice the hypothesized value, almost exactly **2 hours**. Nevertheless, this result is a significant improvement over the primitive arithmetic estimation currently in use by UltraSignup.com and is therefore deemed to be an acceptable outcome for the initial version of this application. Further improvements to this application in future versions including, but not limited to, a larger dataset, more comprehensive feature variables, and hyper-parameter tuning where appropriate, can reasonably be expected to produce a model with improved accuracy and efficiency, as well as improved metrics.

5. EFFECTIVE VISUALIZATIONS AND REPORTING

Provide a description of how your visualizations and elements effectively told an accurate story about the data. This needs to include items such as how your application supported data preparation, data analysis, and data summary. It should also include how your display techniques clearly explained any phenomenon detection if appropriate.

Detailed descriptions of the contribution of each visualization are provided in Section C6 above, including how each visualization supported data preparation, data analysis, or data summary, as well as the various phenomena detected in the data because of the visualizations.

Figure 8: Finish Times by Race supported **data preparation** by exposing several outlier finish times that were either faster than the current world records, or slower than the cutoff time for the given race. It also supported **data analysis** by highlighting the significant dependency of individual finish times on the race cutoff time.

Figure 9: Finish Times Histogram supported **phenomenon detection** by illuminating the large numbers of finishers at several arbitrary finish times, primarily 24 hours, 30 hours, and 32 hours, even in races where the cutoff times were much longer. This phenomenon impacted the final predictive model's over-bias and helped to highlight the need for hyper-parameter tuning in order to increase the model's variance.

Figure 10: Finish Times by Runner Rank supported **data analysis** and **phenomenon detection** by clarifying the correlation between higher Ranks and faster finish times, as well as lower Ranks and slower finish times. However, while lower Ranks almost never correspond to faster finish times, higher Ranks frequently correspond to slower finish times. This is an important phenomenon for the predictive model to recognize and account for in order to achieve a high level of accuracy.

Figure 11: Feature Importance supported **data preparation** and **data summary** since it clearly identifies which features are have the greatest impact on the predictive model and therefore allows for more informed feature selection.

Figure 12: Actual Finish Time / Predicted Finish Time supported **data analysis** and **data summary** as it shows the generalized correlation between actual finish times and predicted finish times. It also shows the correlation between race cutoff time and the predicted and actual finish times.

Figure 13: Mean Absolute Error supported **data summary** and **phenomenon detection**. This figure was crucial for identifying the slight over-bias of the predictive model and need for greater variance. The phenomena detected in Figure 8 were used to identify and explain otherwise unexpected spikes in error which appeared in this figure.

6. ACCURACY ANALYSIS

Assess how accurate your application is at presenting the data and providing predictive outcomes. Provide an example of what the data showed and explain why those offer representative artifacts of the application's accuracy.

Two metrics were used to assess the accuracy and efficiency of the predictive Random Forest Regression algorithm.

The first metric used was **R² Coefficient of Determination** implemented using the Scikit-Learn `score()` method. This is a simple and effective metric particularly useful for regression algorithms. It "represents the proportion of variance (of y) that has been explained by the independent variables in the model." It provides "a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance" (https://scikit-learn.org/stable/modules/model_evaluation.html). The R² Coefficient of Determination for the Random Forest Regression model trained on the current dataset, with the selected features, and default hyper-parameter tuning is **79.5%** on the test dataset.

The second metric used was **Mean Absolute Error** implemented using the Scikit-Learn `mean_absolute_error()` method. Mean Absolute Error is calculated by determining the difference between each predicted finish time and its corresponding actual finish time in the test dataset. Because the predicted finish time may be higher or lower than the actual finish time, the absolute value is taken, and then the mean of all Absolute Errors is calculated. The Mean Absolute Error for the Random Forest Regression model trained on the current dataset, with the selected features, and default hyper-parameter tuning is **2 hours, 0 minutes, and 44 seconds**. This means that, on average, predicted times produced by this application will be accurate to within 2 hours of a runner's actual finish time. This value is larger than the target value of 1 hour proposed in the Project Recommendation documentation. Nevertheless, it offers a significant improvement over the primitive prediction model currently in use by UltraSignup.com and was therefore deemed to be an acceptable outcome for this project.

7. APPLICATION TESTING

Clearly describe the different levels of testing you used to confirm the functionality of your application. These could include unit, integration, system, and acceptance. Also, did you conduct any beta tests or usability tests for your project? Summarize those tests and explain how results were used to modify/calibrate the product during its development.

Throughout the development of this application, it was tested manually by running each module of code in the Jupyter Notebook. More extensive formal unit testing, integration testing, system were deemed unnecessary because this is a relatively small project and because of the overall simplicity of the project code. However, acceptance testing was done in the form of alpha testing by a small group of volunteers, including both technical and non-technical participants. Verbal feedback was provided by each participant regarding the user interface's ease of use, the presence of any bugs (one compatibility issue on a Linux machine

was discovered and corrected), and correctness of the program output in different user environments.

8. APPLICATION FILES

Provide a comprehensive inventory of the files required to execute your application. Include a clear description of the interdependencies and file hierarchy. Describe how those files will be organized for submission. Include those files in the submission.

The application files will be contained within a single zip file named **C964_Task2_Russell_Taylor.zip**. Within the zip will be a single Jupyter Notebook file, this pdf document, and four directories:

race-predictor.ipynb
C964_Task2_Russell_Taylor.pdf
log.file

The **env** directory contains the Anaconda environment files and libraries necessary in order to run the Jupyter Notebook. This is where the Pandas, NumPy, Matplotlib, and Scikit-Learn libraries are located. These files require a significant amount of disk space and so are not included in the files submitted for this project.

The **images** directory contains png image files of each of the figures included in this document and the visualizations exported from the Jupyter Notebook:

fig1_ultrasignup_website_results_page.png
fig2_timeline_and_milestones_gantt_chart.png
fig3_scikit-learn_machine_learning_map.png
fig4_principle_component_analysis.png
fig5_randomized_pca_algorithm.png
fig6_random_forest_regressor_algorithm.png
fig7_data_cleaning.png
fig8_finish_times_by_race.png
fig9_finish_times_histogram.png
fig10_finish_times_by_runner_rank.png
fig11_feature_importance.png
fig12_actual_finish_time_and_predicted_finish_time.png
fig13_mean_absolute_error.png
fig14_predicting_race_finish_times_from_user_input.png
fig15_evaluation_of_the_random_forest_regression_algorithm.png
fig16_using_the_cli_to_password_protect_the_jupyter_notebook.png
fig17_the_jupyter_notebook_configuration_file.png
fig18_the_jupyter_notebook_hashed_password_json_file.png
fig19_the_jupyter_notebook_user_interface_password_prompt.png

fig20_the_jupyter_notebook_debug_log_file.png
fig21_setting_a Breakpoint_using Pythons_pdb_debugger.png
fig22_the_jupyter_notebook_stand_alone_dashboard.png
fig23_western_states_100_mile_endurance_run_raw_data.png
fig24_orcas_island_100_miler_raw_data.png
fig25_using_the_cli_to_install_anaconda.png
fig26_using_the_cli_to_open_the_jupyter_interface.png
fig27_the_jupyter_interface.png

The **models** directory contains the exported Pickle file which is the trained and tested Random Forest Regression model used in this project:

race_predictor_random_forest_model.pkl

The **race_results** directory contains the 16 raw csv data files which comprise the dataset used in this project:

AngelesCrest.csv
CascadeCrest.csv
CruelJewel.csv
HardRock.csv
HURT.csv
IMTUF.csv
JavelinaJundred.csv
KettleMoraine.csv
OldDominion.csv
OrcasIsland.csv
RunRabbitRun.csv
TheBear.csv
TunnelHill.csv
WasatchFront.csv
WesternStates.csv
Yeti.csv

9. USER'S GUIDE

Include a brief manual concerning the installation and use of your application. Be sure to describe all steps necessary to establish an environment capable of running your application. Provide clear, concise steps of how a user would execute the application and produce the results you've described in your documentation. Carefully consider and describe the procedural aspect of the application including know areas where certain crucial steps can affect the performance of the application. You must ensure that anybody can run the application. Please include details on the technology context that is required for your

application to properly execute.

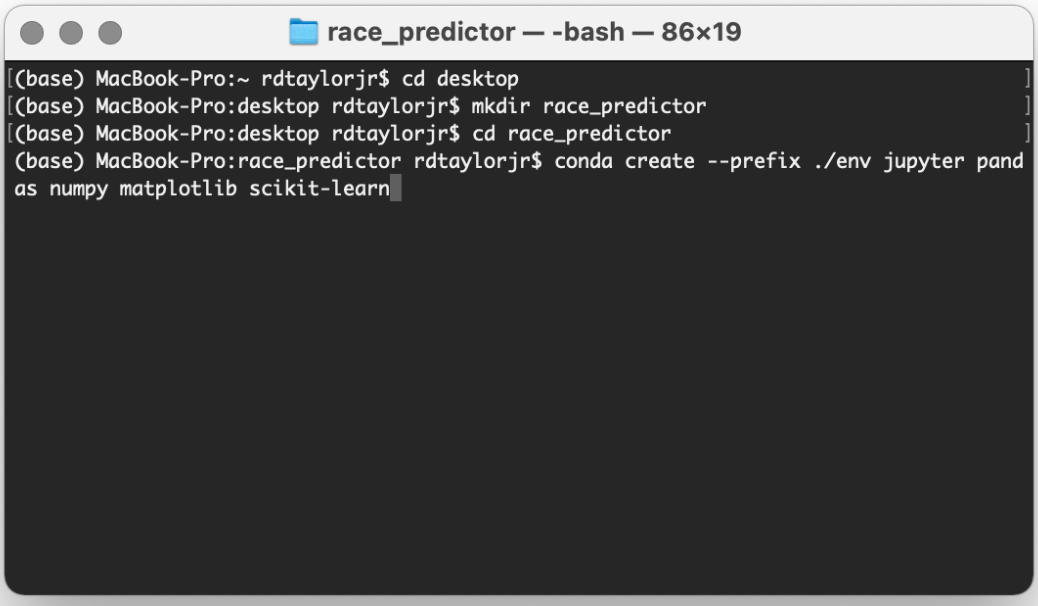
Please follow these steps to install an run this application:

1. Visit https://docs.conda.io/en/latest/miniconda.html and download the version of Miniconda that is appropriate for your Operating System and version of Python. This is a free download.

2. Open the downloaded file and follow the instructions to install Miniconda3 on your computer.

3. Open the Terminal. Type the command `cd desktop` to change directories to the Desktop . Type the command `mkdir race_predictor` to create a new directory called **race_predictor**. Type the command `cd race_predictor` to change directories to the new **race_predictor** directory. Type the command `conda create --prefix ./env jupyter pandas numpy matplotlib scikit-learn` to create a new Conda environment. Conda will download and install the required libraries. When it asks if you would like to proceed, type `y` for yes.

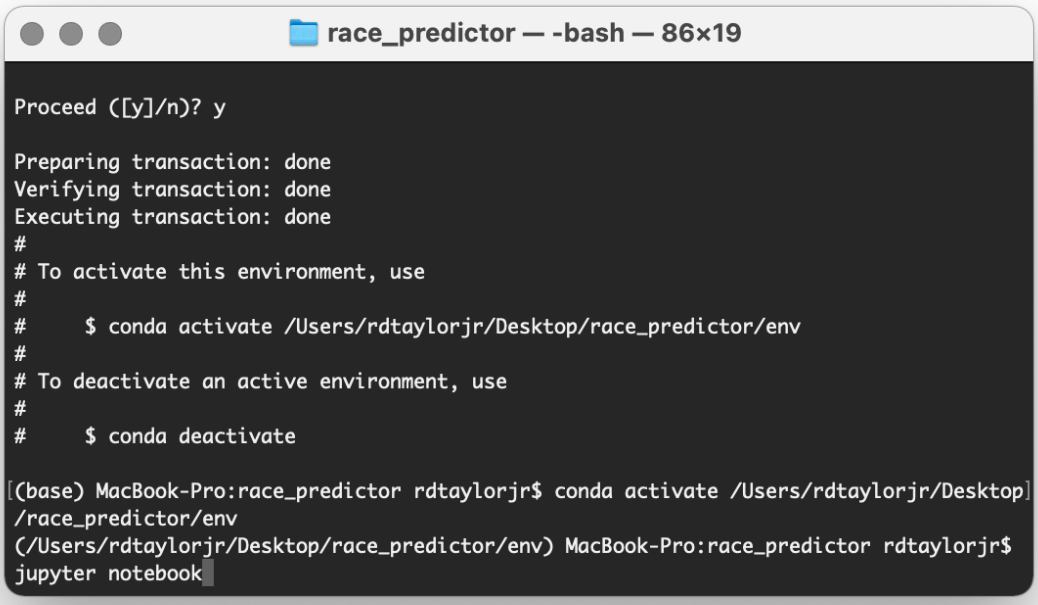
Figure 25: Using the CLI to Install a New Anaconda Environment



4. Activate the new Conda environment by typing the command `conda activate /Users/rdtaylorjr/Desktop/race_predictor/env`. Be sure to replace the username `rdtaylorjr` with your username.

5. Type the command `jupyter notebook`. This will open the Jupyter interface in your web browser.

Figure 26: Using the CLI to Open the Jupyter Interface



6. Enter the password 'capstone' to open the Jupyter Interface. It is strongly recommended that you change this password following the instructions in Section C10 above.

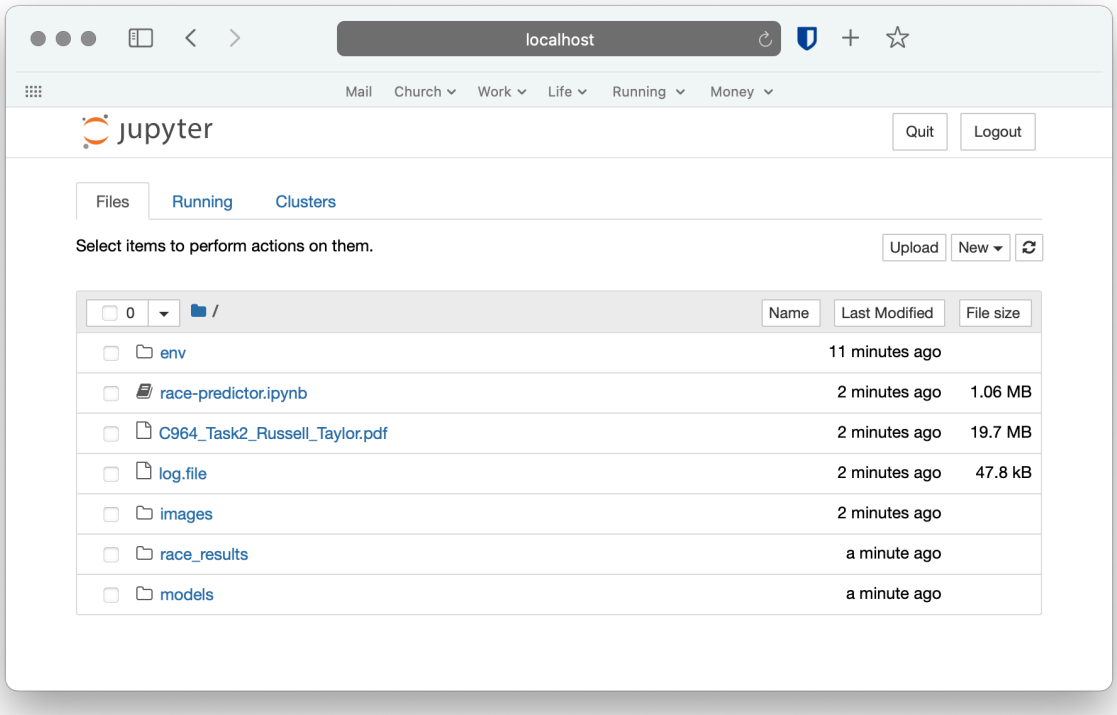
7. Download and unzip the **C964_Task2_Russell_Taylor.zip** file. In the Jupyter Interface, click 'Upload', navigate to the unzipped files, and click 'OK' to upload them.

8. Click on **race-predictor.ipynb** in the Jupyter Interface to open the Jupyter Notebook for this project.

9. Click on each cell of the Jupyter Notebook and press CMD + Enter to run the code in that cell. Ensure that the code in all previous cells has run before you run the code in the next cell.

10. Any of the Python code may be modified to import different datasets, to alter the visualizations, or to modify or tune the Machine Learning models.

Figure 27: The Jupyter Interface



10. SUMMATION OF LEARNING EXPERIENCE

Describe how your prior experience assisted you with the capstone project then review the additional professional development you had to pursue to complete the capstone. What assistance did you seek out from other individuals? How has the experience contributed to your concept of life-long learning?

I am proficient in Python and as part of this degree completed C950 Data Structures and Algorithms II which required a significant amount of coding in Python. For this course I participated in the entire Udemy course, "Complete Machine Learning & Data Science Bootcamp 2021," by Andrei Neagoie and Daniel Bourke (https://www.udemy.com/course/complete-machine-learning-and-data-science-zero-to-mastery). This gave me the basic knowledge of Machine Learning and the Python data libraries I needed to complete this project.

This degree has been part of my preparation for a career in software development, and has provided a valuable opportunity to learn new technologies that I had not used in the past. I also intend to pursue a MS in Computer Science at Georgia Tech where I will study Machine Learning in far greater depth, so I am grateful for this introduction to these concepts. I love learning and am excited to use this course as a foundation to learn more about the many other Machine Learning algorithms beyond the supervised regression algorithm that I implemented in this project.

SOURCES

Stef Bernosky, "DFL > DNF > DNS?," Towards Data Science, October 8, 2017, https://towardsdatascience.com/dfl-dnf-dns-60969b9e995d, accessed December 9, 2020.

Stef Bernosky, "ultrasignup," Github Repository, https://github.com/ophiolite/ultrasignup, accessed December 9, 2020.

Georgios Drakos, "Random Forest Regressor Explained in Depth," *GDCoder*, June 4, 2019, https://gdcoder.com/random-forest-regressor-explained-in-depth, accessed December 22, 2020.

"Frequently Asked Questions," UltraSignup, http://ultrasignup.com/faqs.aspx, accessed December 9, 2020.

David Hamann, "Debugging Jupyter Notebooks," April 22, 2017, davidhamann.de/2017/04/22/debugging-jupyter-notebooks, accessed December 14, 2020.

"How to Enable and Access Debug Logging for Notebook and IPython Kernel," Stack Overflow, November 10, 2015, stackoverflow.com/questions/33632529/how-to-enable-and-access-debug-logging-for-notebook-and-ipython-kernel, accessed December 14, 2020.

David Kiefer, "Is 100 Miles the New Marathon?," Runner's World, May 18, 2015, https://www.runnersworld.com/racing/is-100-miles-the-new-marathon, accessed December 9, 2020.

Ajitesh Kumar, "PCA Explained Variance Concepts with Python Example," August 8, 2020, https://vitalflux.com/pca-explained-variance-concept-python-example, accessed December 21, 2020.

Rodrigo Leite, "Coefficient of Determination (R^2)," *Analytics Vidhya*, December 10, 2020, https://medium.com/analytics-vidhya/coefficient-of-determination-r²-a337b8d3e9e7, accessed December 21, 2020.

Andrei Neagoie and Daniel Bourke, "Complete Machine Learning & Data Science Bootcamp 2021," Udemy Course, https://www.udemy.com/course/complete-machine-learning-and-data-science-zero-to-mastery, accessed December 9, 2020.

"Qualifying Races," *Western States 100 Mile Endurance Run*, https://www.wser.org/qualifying-races, accessed December 20, 2020.

"Race Calendar," *Ultra Running Magazine*, https://calendar.ultrarunning.com, accessed December 9, 2020.

"Race Finish Time Predictor," *Runner's World*, https://www.runnersworld.com/uk/training/a761681/rws-race-time-predictor, accessed December 18, 2020.

Kuldeep Rana, "Levels of Testing," *The Art of Testing*, July 4, 2020, https://artoftesting.com/levels-of-software-testing, accessed December 21, 2020.

Sanjog Slgdel, "How to Password Protect Your Jupyter Notebook," *The Quick Blog*, May 24, 2020, thequickblog.com/how-to-password-protect-jupyter-notebook, accessed December 14, 2020.

"Scikit-Learn Ensemble Methods," scikit-learn.org/stable/modules/ensemble.html, accessed December 21, 2020.

"Scikit-Learn Metrics and Scoring: Quantifying the Quality of Predictions," https://scikit-learn.org/stable/modules/model_evaluation.html, accessed December 21, 2020.

Soner Yildirim, "Principal Component Analysis – Explained," *Towards Data Science*, February 24, 2020, https://towardsdatascience.com/principal-component-analysis-explained-d404c34d76e7, accessed December 21, 2020.