STA 160 Final Project

Eric Kye
Student ID: 919455487

May 29, 2025

1 Introduction

Twenty drivers. Ten teams. Two drivers per team. One goal.

Few things are more exciting and anxiety-inducing than the sport of Formula 1. From the screeching roar of the engines to the blazing speeds of the cars going over 200 miles per hour, Formula 1 is the peak of motorsport. As a fan, you can feel the adrenaline coursing through the veins of the drivers as they risk their lives for a chance to be called the fastest driver in the world. In a sport where margins can decide victory or defeat, life and death, data is just as powerful for the drivers as horsepower.

Figure 1: Formula 1 cars driving during the Qatar Grand Prix.

The Kaggle F1nalyze competition challenges me to use historical race data to try and predict something as dynamic as the finishing position of the driver. The data gives me insight into driver and team attributes while also providing race/track-specific information. My specific goal in this project is to develop a predictive model that can accurately predict the position that a driver will finish in from 1 to 20.

[0]Source: Image adapted from the blog post "The Complete Beginners Guide to Formula 1" by F1Chronicle (`https://f1chronicle.com/a-beginners-guide-to-formula-1/`).

This challenge is intriguing to me because I am a huge Formula 1 fan, especially of Team Ferrari. I thought this was a chance to better understand and appreciate the technical side of the sport that I watch for fun. I also thought this would be a great way to use what I have learned in my undergraduate stats classes and apply it to something I am passionate about.

What makes this challenge important is that it is similar to the real problems that teams in F1 face. In F1, strategies such as planning pit stops and choosing tire types can help change race results. A team is only as successful as its analytics and engineering department. Formula 1 is as much engineering as it is good driving.

To take on this challenge, I have taken the classic statistical data science project approach. I started with data processing and cleaning and then explored the data to see if I could identify any patterns. Then I applied a wide range of machine learning algorithms to see what would best work in predicting driver positions. In the end, I interpreted the results to show the insights that the project has provided.

2 Data Overview and Exploratory Data Analysis (EDA)

The training dataset used in this project comes from the Formula 1 Datathon hosted on Kaggle. It contains decades' worth of Formula 1 World Championship data. These variables range from lap counts and grid positions to driver nationalities and constructor stats. This archive spans the rich history of F1 and is perfect for a predictive task like estimating driver finishing positions. For modeling purposes, the competition also provides separate validation and test datasets, each with around 350,000 entries.

Before attempting any modeling, the data needed significant cleaning. There were over 2.8 million race records and 55 variables. This was confusing to me at first because that just seemed like an impossible number of race records to have. After looking into the training dataset, it was clear that the dataset had many duplicate rows and inconsistencies. This was likely due to merging issues across tables and eras (F1 has over 70 years of history). To ensure cleaner inputs and no duplicates, I grouped the data by `raceId` and `driverId`. These were the core identifiers of a race driven by a specific driver. After cross-examining the table with historical results online, it seemed the most accurate duplicate record was the best record per driver per race. I sorted in order based on total points, race finish position, and season points to retain the most accurate result.

After cleaning, the training dataset was reduced to a much more manageable 21,998 rows with 16 columns. Looking at the features, I chose to drop 39 columns to make the dataset cleaner and easier to work with. Many of the columns I removed, like URLs, driver names, and session times (such as practice or qualifying dates), weren't useful for predicting race results. Some of them had a lot of missing data, while others were just ID numbers or repeated information in a different form. My goal was to keep only the columns that could actually help predict a driver's finishing position. I also wanted to try my best to ensure that the columns were numeric so they could help when working with different machine learning models. It was clear that these features would be things like starting grid, points, team, and race location. Removing unnecessary or messy columns helped simplify the dataset and made it easier to build a better model.

After cleaning all of those columns, the ones that remained were all complete. There were no missing values. I did keep three non-numeric features, as I believed that they would be useful to predict position and driver success. I kept driver nationality, constructor nationality, and `grand_prix`. I encoded them numerically to make them model-friendly. For example, "Abu Dhabi Grand Prix" was encoded as 0, while driver nationalities like "American" or "Finnish" also received integer labels. There were a number of 40 unique Grand Prix, 40 unique driver nationalities, and 24 unique constructor nationalities.

Finally, after cleaning the training dataset, I was able to explore the cleaned dataset. The cleaned dataset revealed some interesting trends. The majority of drivers in a race earn zero points, which aligns with how only the top 10 finishers are awarded points in Formula 1. The median value for points per race is 0, while the maximum is 25, indicating a skewed distribution. Similarly, most drivers have zero wins, with a few outliers achieving double-digit victories. The `points_y` column, which tracks cumulative season performance, ranges from 0 to over 400, clearly distinguishing dominant drivers from one-time participants. Also, `grid` position and `positionOrder` columns ranged widely (from 0 to 34 and 1 to 39, respectively). This is because the number of starting cars on the grid has shifted over the years. In an effort to improve safety, the modern size for the starting grid is 20.

When looking at the correlation matrix, I was surprised to find that few variables strongly correlated with final race position. However, `grid` (starting position) had a correlation of 0.36, `constructorId` had 0.22, and `driverId` had 0.28, showing the strongest associations. This intuitively makes sense, as you are more likely to finish better with a better starting position. When it comes to drivers and constructors, this positive relationship also makes sense. Drivers can only perform as well as the car they're given. Top constructors usually have more money to build better cars and attract the most talented drivers with higher contracts. This leads to a strong correlation between constructor quality and driver success.

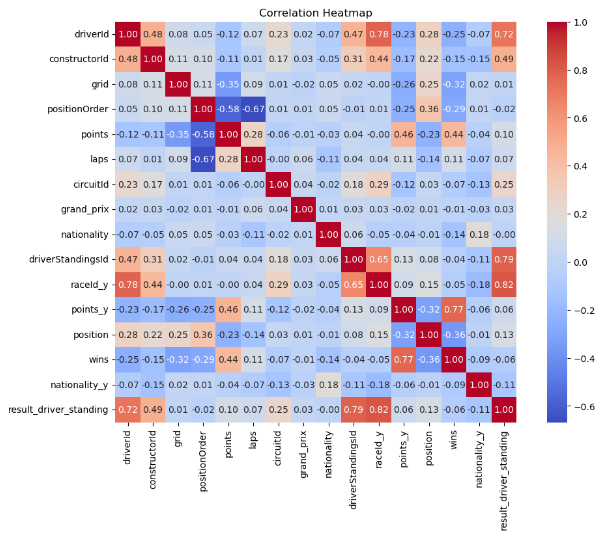
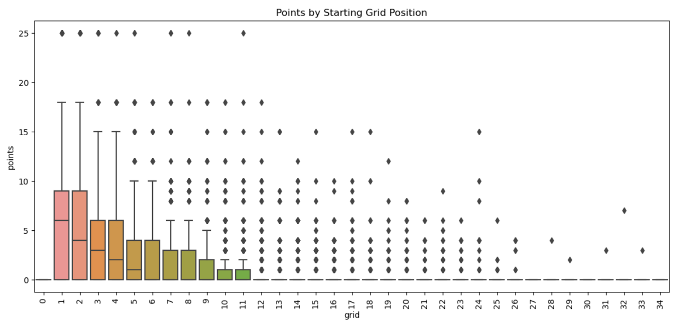
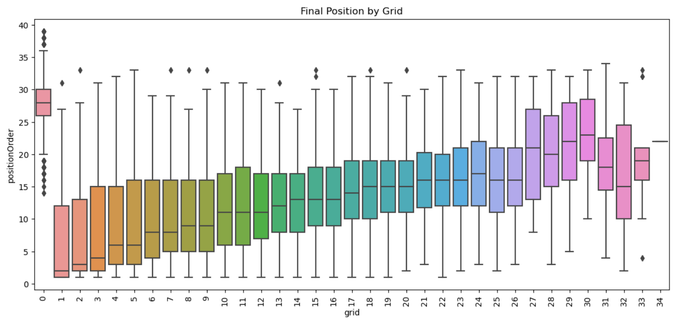


Figure 2: Correlation matrix showing relationships among key features.

To further understand these relationships, I visualized key patterns. The box plots comparing grid position to both points earned and final race position confirmed that drivers starting in the top three rows consistently performed better. There's a steep drop-off in median performance after grid position 10, highlighting the massive advantage of strong qualifying sessions to gain a better starting position on race day.



(a) Grid position vs. points earned.

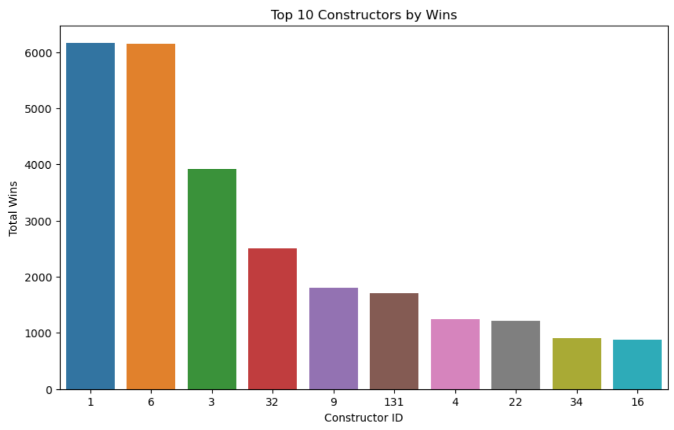


(b) Grid position vs. final race position.

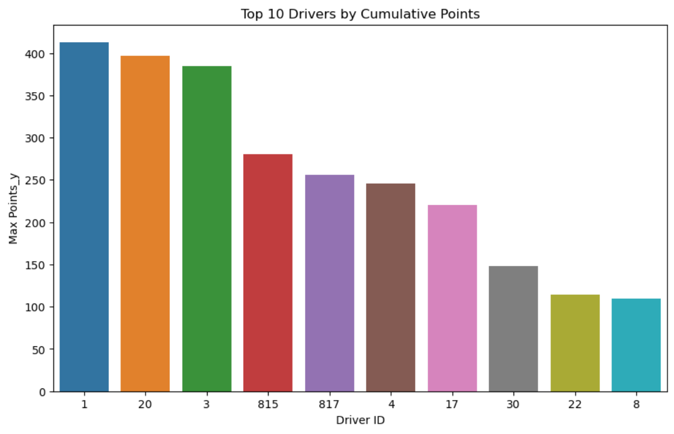
Figure 3: Boxplots showing the impact of starting grid on race outcomes.

I also created two bar charts to visualize the top 10 constructors by total wins and the top 10 drivers by total points. After cross-referencing the results with official F1 records, I was able to confirm that the data cleaning steps I took earlier didn't remove any valuable or accurate information. These charts paint a clear picture of Formula 1's dynasties. The top constructors included McLaren, Ferrari, Williams, Team Lotus, Red Bull, and Mercedes, along with historic names like Renault, Benetton, Brabham, and BAR. It's no surprise to see Ferrari and McLaren dominating the top, as they are historically two of the most prestigious teams, each surpassing 6,000 wins in the dataset. You can also see the rise of modern successful constructors like Red Bull and Mercedes making their way up the chart.

On the driver side, legends like Lewis Hamilton and Sebastian Vettel lead with over 400 cumulative points, closely followed by other all-time greats like Michael Schumacher, Fernando Alonso, and Nico Rosberg. These visualizations not only helped validate the cleaned dataset but also confirmed a key pattern in the sport: the best drivers tend to be paired with the best constructors. Many of the drivers in the top 10 raced for these dominant teams, showing how top-tier talent and top-tier engineering often go hand in hand to produce winning results.



(a) Top 10 Constructors by Wins.



(b) Top 10 Drivers by Points.

Figure 4: Bar charts highlighting Formula 1 dynasties among teams and drivers.

In the end, cleaning the data made it much easier to work with and more accurate. The trends I found, such as better starting positions and top teams that lead to better results, match the real F1 patterns. The visuals confirmed that the data was reliable, and now I am ready to move on to building models to predict race outcomes.

3 Methodology

After cleaning the training dataset, I moved on to the modeling part of the project. The main evaluation metric for the Kaggle Formula 1 Datathon was Root Mean Squared Error (RMSE). For modeling, I used the training dataset to fit each model and evaluated performance on the validation set. The test set was not used for evaluation because it doesn't include actual driver finishing positions. Instead, it was used only for final predictions submitted to Kaggle for actual competitors at the time of the Datathon.

Instead, to evaluate model performance, I relied on the validation set. Since the validation dataset contains both features and the actual target variable (position), it allowed me to test how well each model generalized to new data. This helped me compare models and tune thehyperparameters. I decided to try five different machine learning models that I learned from previous stats courses here in Davis: Random Forest, XGBoost, Gradient Boosting Regressor, Lasso, and Ridge Regression. Each model offered a different approach to the problem, and comparing them gave me a better understanding of how different models perform on F1 data.

I started with Random Forest because it handles structured data well and can capture complex relationships. Since F1 results depend on many interacting factors like driver, team, and starting position, Random Forest was a good starting point. This is because it can handle outliers and overlapping features better than many models.

Next, I used XGBoost, which is a boosting method that builds trees and learns from past mistakes. From past courses, I learned that this model is usually the most accurate. XGBoost also has regularization, which helps reduce overfitting. I also tried Gradient Boosting Regressor from scikit-learn. This gave me a way to compare how basic boosting performs compared to more advanced versions like XGBoost. If both perform similarly, it suggests that boosting in general is a good strategy for this kind of prediction task.

To add some contrast, I tested two linear models: Lasso and Ridge Regression. Lasso uses L1 regularization to shrink less important features to zero and help with feature selection. Ridge uses L2 regularization to deal with multicollinearity, which can happen in F1 data since many variables are related. These linear models served as simple baselines to see if the relationships in the data could be captured with more basic methods. After running each model, I compared its RMSE scores on the validation set. As expected, XGBoost and Random Forest performed the best, while the linear models had higher errors. This showed that more flexible and nonlinear models are better suited for predicting F1 race outcomes.

To improve XGBoost even further, I used 3-fold cross-validation to tune the hyperparameters. I tested different values for learning rate, tree depth, and the number of estimators. After finding the best combination, I retrained the model and used a feature importance plot to see which features had the biggest impact on predictions.

4 Results

As mentioned previously, after building and training the models, I evaluated their performance using Root Mean Squared Error (RMSE) on the validation set. This metric helped measure how close each model's predicted race positions were to the actual results. Among all five models, Random Forest performed the best with an RMSE of 4.47. To my surprise, the XGBoost model performed slightly worse at 5.37, while gradient boosting had a lower RMSE of 5.27. The two linear models, Lasso and Ridge Regression, had the highest errors at 7.013 and 7.03, confirming that the simpler, linear approaches weren't well-suited for the complexity of this dataset.

To improve XGBoost's performance, I performed 3-fold cross-validation and tuned several hyperparameters. After testing different combinations, the best set of parameters was a learning rate of 0.05, a max depth of 6, 100 estimators, and a subsample ratio of 1. These changes brought XGBoost's RMSE down from 5.37 to 4.68. However, this was still higher than the Random Forest model, suggesting that for this particular dataset, Random Forest was better at generalizing, while XGBoost may have overfitted the data.

I also generated a feature importance plot using the tuned XGBoost model to understand which variables had the biggest impact on predictions. The plot showed that `points_y` (season points) and `wins` were by far the most important features. This makes sense since drivers can only have season success by doing well each race day. Other key features included `positionOrder`, `driverId`, and `grid`, aligning with patterns found during the earlier data exploration. Less impactful but still useful features included `nationality`, `grand_prix`, and `constructorId`.

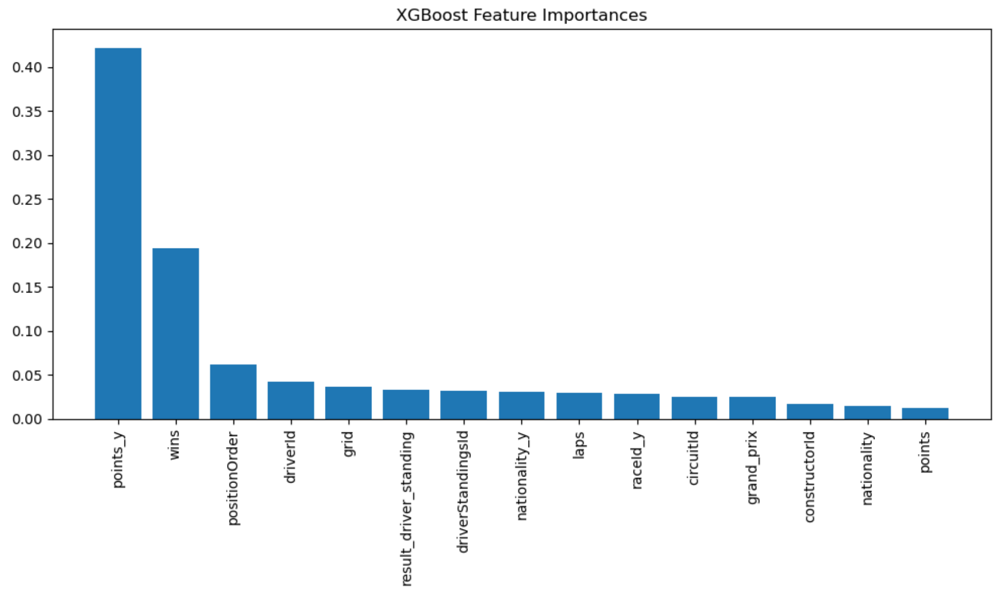


Figure 5: Feature importance plot from the tuned XGBoost model.

Overall, these results confirmed that the models were learning meaningful patterns from the data. The strong performance of tree-based models like Random Forest and XGBoost showed they were well-suited for capturing the complex, non-linear nature of Formula 1 race outcomes.

Visualizing Model Performance with XGBoost

To better understand how well the tuned XGBoost model performed, I created two visualizations and a sample table to evaluate the predicted finishing positions compared to the actual results. These visuals helped confirm some of the trends I was seeing in the RMSE values and added more context to how the model was behaving.

The first chart below shows a scatter plot of actual vs. predicted finishing positions. In an ideal world, all the points would line up perfectly on the red dashed line, showing that the model's prediction exactly matched the true race result. Although the predictions generally followed an upward trend, which means that the model correctly identified that drivers who finished higher were likely to finish higher again, there was a lot of spread around each actual position. This means that while the model gets the general ranking right, it still struggles to be precise, especially for drivers finishing in the middle or back of the pack.

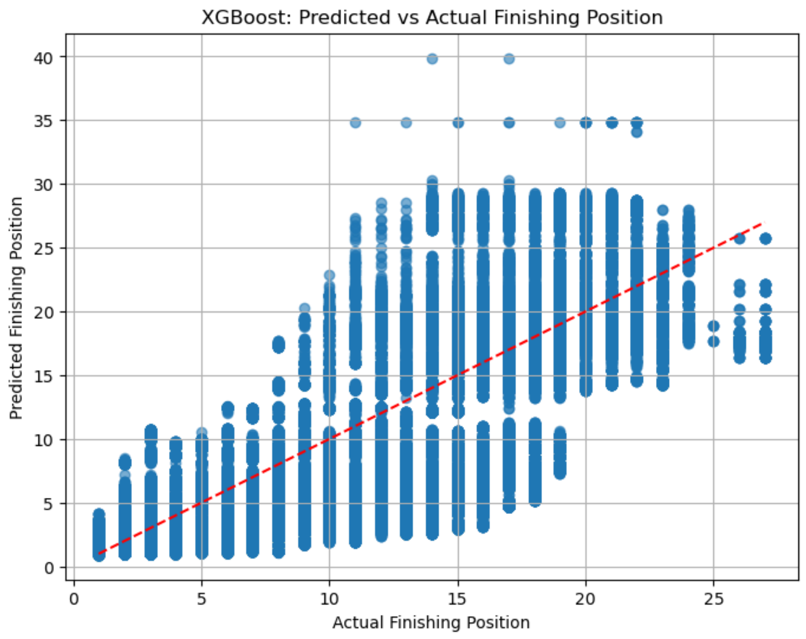


Figure 6: XGBoost: Predicted vs Actual Finishing Position

To dig a little deeper into where the model was going wrong, I also created a residual plot. This plot shows the difference between the actual and predicted positions for each driver. A perfect model would have all its residuals centered around zero. However, what I noticed was a clear pattern: the model tends to underestimate the best drivers (residuals above zero on the left side of the plot) and overestimate the lower-performing ones (residuals below zero on the right). This means that the model has a tendency to pull predictions toward the middle, and it is hesitant to predict that someone will finish first and also reluctant to say that someone will finish last. This makes sense since RMSE penalizes large errors, and the model might be playing it safe by avoiding extreme predictions.

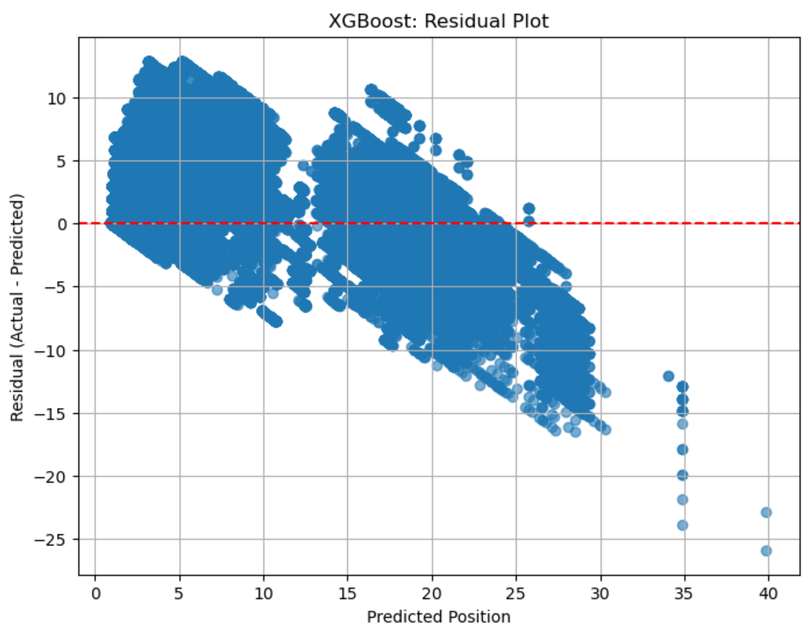
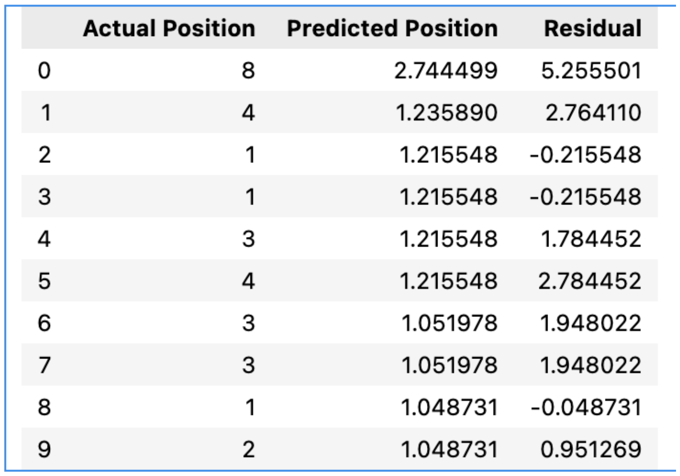


Figure 7: XGBoost: Residual Plot (Actual - Predicted)

Finally, to make the performance easier to interpret, I included a table that shows a sample of 10 predictions from the validation set. You can see that the predictions aren't random, and they're generally close to the actual finishing position. But there are still a few examples where the model predicted a driver to finish much higher than they did. This reinforces the idea that the model understands general race dynamics but struggles with nailing down the exact position when things get competitive or unpredictable.



	Actual Position	Predicted Position	Residual
0	8	2.744499	5.255501
1	4	1.235890	2.764110
2	1	1.215548	-0.215548
3	1	1.215548	-0.215548
4	3	1.215548	1.784452
5	4	1.215548	2.784452
6	3	1.051978	1.948022
7	3	1.051978	1.948022
8	1	1.048731	-0.048731
9	2	1.048731	0.951269

Figure 8: Sample of Actual vs. Predicted Finishing Positions with Residuals

These visualizations helped me better understand the strengths and weaknesses of the model. They also made it clear that there is still room for improvement, especially when it comes to modeling the unpredictable nature of a Formula 1 race.

5 Discussion and Reflection

One of the biggest challenges I faced in this project was during the data cleaning process. The dataset was huge, with over 2.8 million rows, and had a lot of duplicate and messy entries. I tried my best to clean it by grouping by race and driver, sorting based on points and positions, and cross-referencing with real race results online. I spent several hours on this step, and while I was able to clean up a lot of the issues, I know it wasn't perfect. Some entries still didn't fully make sense. For example, I remember seeing a driver finish first but gain only 2 points. In a perfect data science world, I would have looked through every row and checked it against official F1 results and removed the duplicates this way.

Even with those issues, I think the results still showed meaningful patterns. The models picked up on trends like how past season points and wins affect future race performance. It was also interesting to see that Random Forest actually did better than XGBoost, even though XGBoost is usually one of the best models in competitions. This might be because Random Forest handles messy data better, while XGBoost may have overfitted to some of the noise that was left in the dataset.

If I had had more time, I would have explored more ways to improve the model. I could have tried new features, like adding information about driver age, team experience, or whether a race was in the driver's home country. Just from my fan knowledge, in Formula 1, there can sometimes be "home Grand Prix advantages." For example, Lewis Hamilton tends to perform well at the British Grand Prix, and Ferrari drivers typically have a strong showing when racing in Italy. I also think it would have helped to check more carefully whether each model was making fair and consistent predictions.

Overall, this project showed me how important it is to have clean and well-organized data. The modeling part was important, but getting the data right made the biggest difference. Even though it was not perfect, I learned a lot from going through the full process and working with such a large and messy dataset.

6 Conclusion

This project gave me a chance to take a closer look at Formula 1 using real data and apply everything I've learned in all my statistics courses at Davis so far. The main goal was to build a model that could predict a driver's finishing position based on past performance, team information, and race details. Through cleaning, exploring, and modeling the data, I was able to uncover patterns that reflect what we often see in real races. These patterns are how starting position, season points, and team strength all play a crucial role in predicting driver success.

While the data cleaning process wasn't perfect, I still saw that the models were able to learn useful trends. The tree-based models like Random Forest and XGBoost performed the best, showing that more flexible approaches work better with complex sports data like this. The results helped answer my original question and also showed how important data quality is when doing any kind of prediction work.

Looking ahead, there are a lot of ways I could improve this project. With more time, I would explore new features, look deeper into model fairness, and maybe even try ensembling models to boost performance. Overall, this project was a valuable learning experience that allowed me to connect my passion as a Formula 1 fan with what I've been learning in class and apply it to a real-world problem.

Works Cited

1. Autosport. *Most successful F1 teams: Which team has the most Constructors' titles?* `[Online]`. Available at: `https://www.autosport.com/f1/news/most-successful-f1-teams-which-team-has-the-most-constructors-titles-5367606/5367606/` `[Accessed: 4 December 2024]`.
2. DataCamp. *XGBoost in Python: A Step-by-Step Tutorial.* `[Online]`. Available at: `https://www.datacamp.com/tutorial/xgboost-in-python` `[Accessed: 4 December 2024]`.
3. Formula1.com. *Race Result - Great Britain 1950.* `[Online]`. Available at: `https://www.formula1.com/en/results/1950/races/94/great-britain/race-result` `[Accessed: 4 December 2024]`.
4. Formula1.com. *What is Formula 1?* `[Online]`. Available at: `https://www.formula1.com/en/page/what-is-f1` `[Accessed: 4 December 2024]`.
5. Machine Learning Mastery. *Feature Importance and Feature Selection with XGBoost in Python.* `[Online]`. Available at: `https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/` `[Accessed: 4 December 2024]`.
6. MonsterWriter. *Using italics in LaTeX.* `[Online]`. Available at: `https://www.monsterwriter.com/italics-LaTeX.html/` `[Accessed: 4 December 2024]`.
7. NVIDIA. *XGBoost.* `[Online]`. Available at: `https://www.nvidia.com/en-us/glossary/xgboost/` `[Accessed: 4 December 2024]`.
8. Overleaf. *Bold, italics, and underlining.* `[Online]`. Available at: `https://www.overleaf.com/learn/latex/Bold,_italics_and_underlining/` `[Accessed: 4 December 2024]`.
9. Overleaf. *How do I adjust the font size?* `[Online]`. Available at: `https://www.overleaf.com/learn/latex/Questions/How_do_I_adjust_the_font_size%3F/` `[Accessed: 4 December 2024]`.
10. Overleaf. *Line breaks and blank spaces.* `[Online]`. Available at: `https://www.overleaf.com/learn/latex/Line_breaks_and_blank_spaces/` `[Accessed: 4 December 2024]`.
11. PCC Instructional Support. *Accessibility in Math and Science: LaTeX.* `[Online]`. Available at: `https://www.pcc.edu/instructional-support/accessibility/mathscience/latex/` `[Accessed: 4 December 2024]`.
12. Scikit-learn. *sklearn.preprocessing.LabelEncoder.* `[Online]`. Available at: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html` `[Accessed: 4 December 2024]`.
13. Seaborn. *seaborn.barplot.* `[Online]`. Available at: `https://seaborn.pydata.org/generated/seaborn.barplot.html` `[Accessed: 4 December 2024]`.
14. Seaborn. *seaborn.boxplot.* `[Online]`. Available at:

`https://seaborn.pydata.org/generated/seaborn.boxplot.html [Accessed: 4 December 2024].`

15. TeX Stack Exchange. *Forcing math mode to be on the same line.* `[Online]`. Available at: `https://tex.stackexchange.com/questions/89354/forcing-math-mode-to-be-on-the-same-line/ [Accessed: 4 December 2024].`
16. TeX Stack Exchange. *Is there a designated symbol for the negative sign in say -16?* `[Online]`. Available at: `https://tex.stackexchange.com/questions/79141/is-there-a-designated-symbol-for-the-negative-sign-in-say-16/ [Accessed: 4 December 2024].`
17. TeX Stack Exchange. *Quote marks are backwards using Texmaker + pdflatex.* `[Online]`. Available at: `https://tex.stackexchange.com/questions/52351/quote-marks-are-backwards-using-texmaker-pdflatex/ [Accessed: 4 December 2024].`
18. XGBoost. *Official Documentation (v3.0.0).* `[Online]`. Available at: `https://xgboost.readthedocs.io/en/release_3.0.0/ [Accessed: 4 December 2024].`

Appendix: Jupyter Notebook Code

The full Jupyter Notebook used is available at the following GitHub link:

`https://github.com/ekye0512/Formula1-STA-160/blob/main/sta160project.ipynb`

This notebook contains all code blocks and results discussed in the main report.