**Introduction**

This project aims to categorize which of the over 1000 Pokémon, including alternate forms, are considered viable or not. This project will simplify the complexities of competitive Pokémon down to three things: the Pokémon’s type, the Pokémon’s stats, and the Pokémon’s ability. This is a gross oversimplification of the system as move-sets, the general availability of Pokémon at the time, and hold items, along with a myriad of other extraneous game systems at play can affect a Pokémon’s place in the meta. This project mainly serves as conduit to explore classification data mining techniques.

The metric for viability used for this project is the Smogon tiering system as of the beginning of December 2023. Smogon tiers are a grassroots community run system where tiering starts at OU, or Standard Play, followed by UU, RU, NU, PU, and finally ZU. Every month Pokémon who do not meet a specific usage threshold for its given tier will be dropped down to a lower tier. Pokémon that are too strong for their tier but do not meet the usage thresholds for the tier above are placed onto the Ban List or BL for its tier, i.e. OUBL, UUBL etc. There is also a tier above OU known as Ubers where exceptionally strong Pokémon are placed in and a tier above that known as Anything Goes or AG where Pokémon that completely break the game resides in.

There also exists two separate tiers that exist independently of the standard tiering system. Little Cup or LC is its own separate game where first stage evolution Pokémon, with exceptions, can battle against each other. The final tier is Not Fully Evolved or NFE consists of middle stage evolution Pokémon who are considered ineligible to participate in LC but do not see play in the standard tiering system because they are just seen as lesser versions of their fully evolved counterparts, also with some exceptions.

Originally, I wanted to be able to sort Pokémon into specific tiers, but my lack of knowledge on how to do that for this project forced me to change the scope of the project to a simpler binary classification project; Viable and Unviable. I partitioned the tier from AG to RUBL as viable and the rest as unviable. In retrospect, I should have removed all LC and NFE from the ranking because they are not part of the standard tiering system. I should also consider removing AG and potentially Ubers because these Pokémon are considered way too powerful for standard play. This is something I will consider when I revisit this project later.

**Data Cleaning**

The data was compiled at the beginning of December 2023, just before the release of the Pokémon Scarlet and Violet DLC, The Indigo Disk. As a result, known Pokémon that would become released later that month, such as Iron Crown and Raging Bolt, needed to be removed from the dataset because all the relevant information needed to analyze whether those Pokémon were viable or not was not available to the public at the time. Pokémon that were never available to use by the player, such as Eternamax Eternatus, or were available, but in a game whose battle system is not like the rest of the series, such as Partner Pikachu, also needed to be removed. When I revisit the project, I plan on removing Pokémon who have not been available in SV from the data set to be used as testing data. I used whatever ranking they had when they were last available, but I realized after the fact that I should’ve removed them initially because Pokémon metas are drastically different from generation to generation.

The first main hurdle that I encountered when cleaning the data was figuring out how to dummy out the variables for Typing and Abilities. There are 18 types, and over 300 Abilities that Pokémon can have. Figuring out how to handle this data was crucial if I were to implement adding moves to the data set down the lines considering that there are over 1000 unique moves. The main problem that I encountered concerning types and abilities was that a Pokémon can have a Secondary Type in addition to its Primary Type and three different abilities; it doesn’t matter which slot the typing or ability is in, I just matters if the Pokémon has the typing or ability or not. As a result, I couldn’t just dummy out the variables the way I was taught in school otherwise I would get dummy variables such as PrimaryType\_Fire, SecondaryType\_Fire, Ability1\_Intimidate, Ability2\_Intimidate etc) as opposed to just Type\_Fire or Ability\_Intimidate. After some research I learned how to splice PrimaryType and Secondary Type, and Ability1, Ability2, and HiddenAbility together into Type and Ability, where each separate entry was separated by a comma. This allowed me to use the split variable in the dummy variables function to assign each separate instance into its own dummy variable.

The next problem I encountered was how to find high correlation between variables when I had to sort through over 300 variables. In the end, I found an article with code created by Katherine Williams published in a Data Science article that had the solution I was looking for. After running the code, I found that variables that had the highest correlation were mostly Abilities that only Pokémon of a specific type are given, two signature Abilities that both belong to the same Pokémon, and Base State Total or BST with any specific stat or two different stats in general. The only pair of variables that had a correlation over .80 was the Abilities Aura Break and Power Construct, which are the signature abilities of Zygarde. I decided to keep in the data point anyways because I didn’t think it would affect things that badly because it only barely tiptoed over the .80 threshold.

A screen shot of a computer

Description automatically generated

**TO DO: Rerun Models Removing LC and NFE. Also remove entries that are outdated, from older gens.**

**Logistic Regression**

I ran into two main issues while trying to perform Logistic Regression on this dataset. The first one was trying to factorize all the categorical variables which took up the overwhelming majority of my data. I resolved this by going into Excel, copying and transposing all the relevant column names and used concatenation to generate all the commands I needed. The second and more pressing issue was that it took 12 hours to run all three models that I had planned to create, the longest being the backwards selection model. I had to run the models again after realizing I made a mistake. As a result, I learned how to save and load models so I wouldn’t have to go through the entire tedious process again. This also made me consider learning how to run multiple threads in order to speed up the model creation process when I go back to add moves.

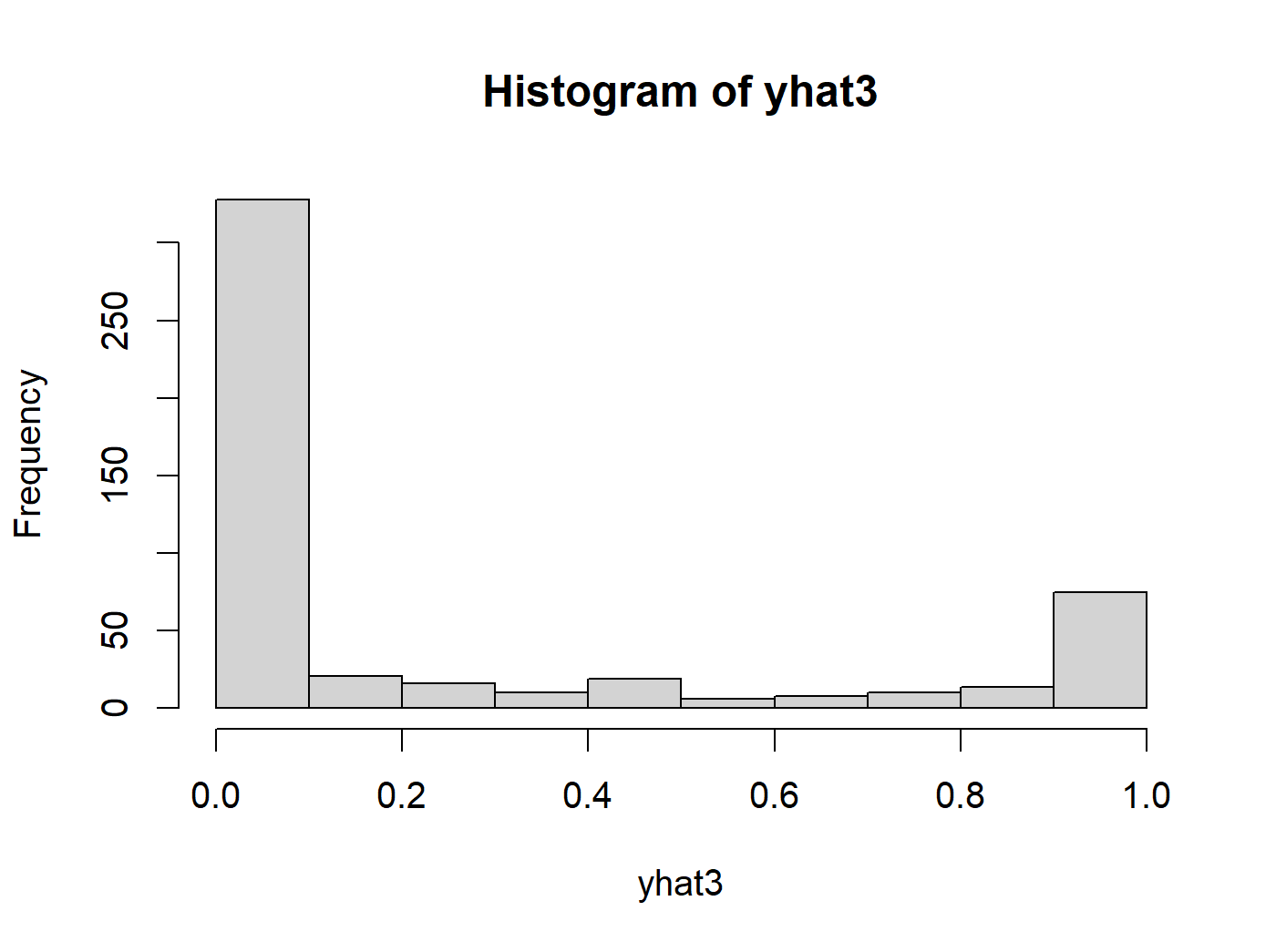
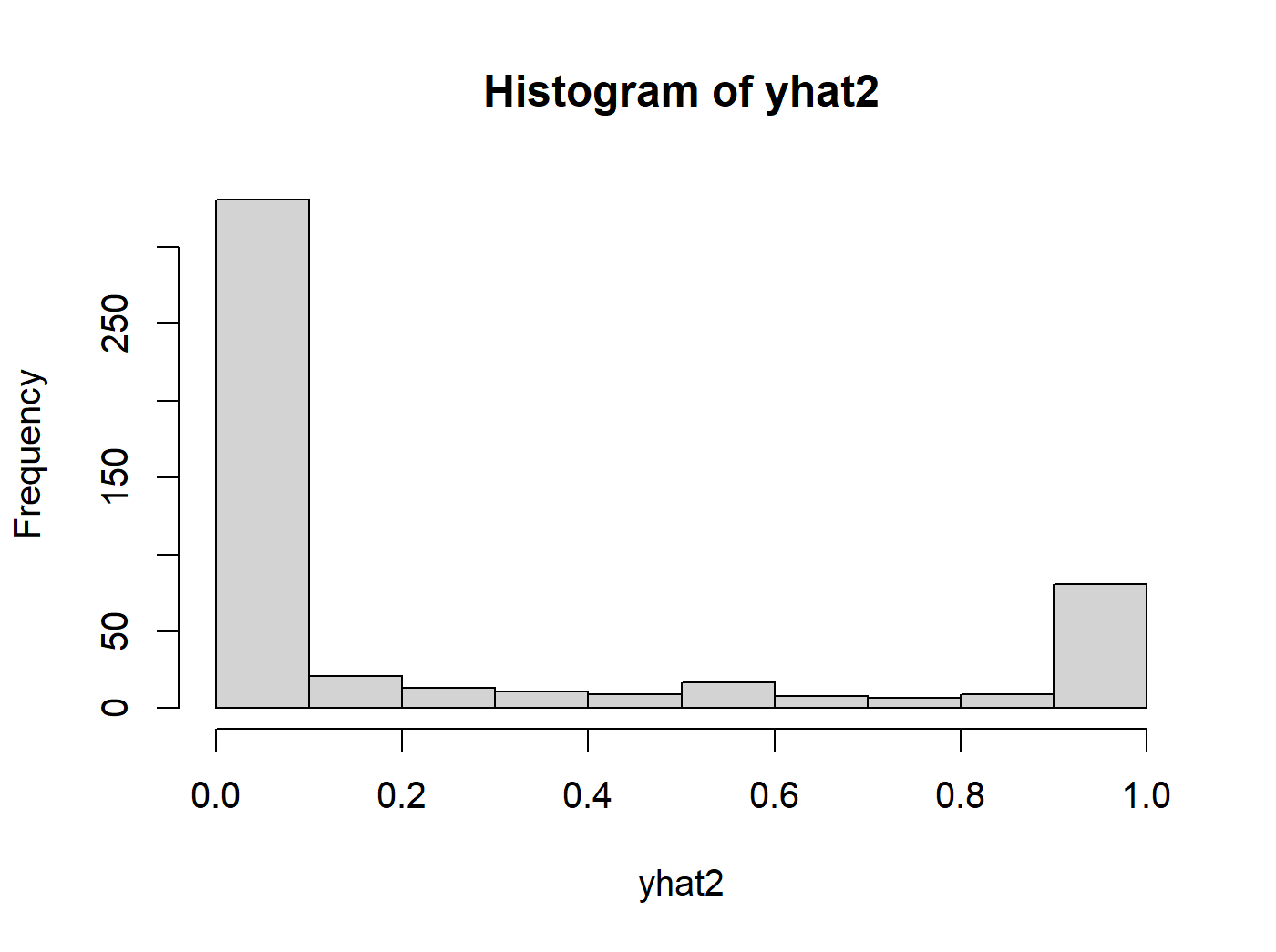
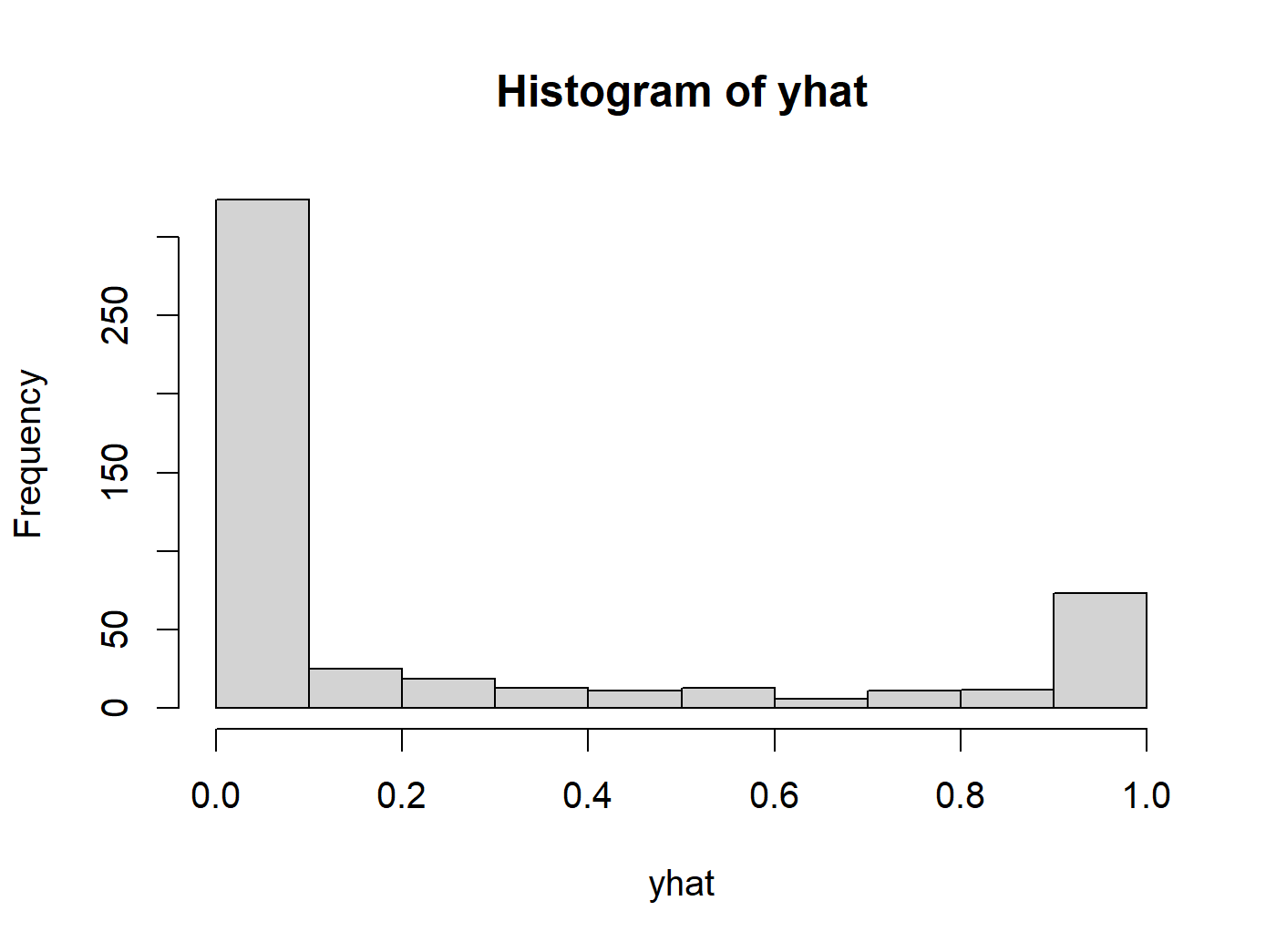
# Results

|  |  |  |  |
| --- | --- | --- | --- |
| Forward | ytrue.class |  |  |
|  |  | 1 | 0 |
| yhat.class | 1 | 102 | 13 |
|  | 0 | 24 | 368 |
|  |  |  |  |
| ERR | 7.30% |  |  |
| Sensitivity | 80.95% | PPV | 88.70% |
| Specificity | 96.59% | NPV | 93.88% |

|  |  |  |  |
| --- | --- | --- | --- |
| Backwards |  | ytrue.class | |
|  |  | 1 | 0 |
| yhat2.class | 1 | 107 | 16 |
|  | 0 | 19 | 365 |
|  |  |  |  |
| ERR | 6.90% |  |  |
| Sensitivity | 84.92% | PPV | 86.99% |
| Specificity | 95.80% | NPV | 95.05% |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Forwards | Backwards | Stepwise |
| Ability |  |  |  |
| 1 | Ripen | Pickup | Pickup |
| 2 | Regenerator | Regenerator | Regenerator |
| 3 | Weak Armor | Static | Insomnia |
| 4 | Toxic Chain | Insomnia | Weak Armor |
| 5 | Swarm | Ripen | Anticipation |
| 6 | Drizzle | Steam Engine | Toxic Chain |
| 7 | Static | Weak Armor | Clear Body |
| 8 | Unaware | Swarm | Swarm |
| 9 | Clear Body | Unaware | Unaware |
| 10 | Anticipation | Anticipation | Static |
| Types |  |  |  |
| 1 | Poison | Water | Poison |
| 2 | Fighting | Fighting | Water |
| 3 | Steel | Dark | Fighting |
| 4 | Water | Steel | Steel |
| 5 | Dark | Poison | Dark |
| 6 | Fairy | Fairy | Fairy |
| 7 | Ground | Ground | Ground |
| 8 | Ghost | Ghost | Fire |
| 9 | Fire | Flying | Ghost |
| 10 | Normal |  |  |
| Stats |  |  |  |
| 1 | Speed | Speed | Speed |
| 2 | Sp. Attack | Attack | Sp.Atk |
| 3 | Sp. Def | Sp.Def | Defense |
| 4 |  | HP |  |
| 5 |  | Sp.Atk |  |
| 6 |  | Defense |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Stepwise |  | ytrue |  |
|  |  | 1 | 0 |
| yhat3.class | 1 | 101 | 12 |
|  | 0 | 25 | 369 |
|  |  |  |  |
| ERR | 7.30% |  |  |
| Sensitivity | 80.16% | PPV | 89.38% |
| Specificity | 96.85% | NPV | 93.65% |



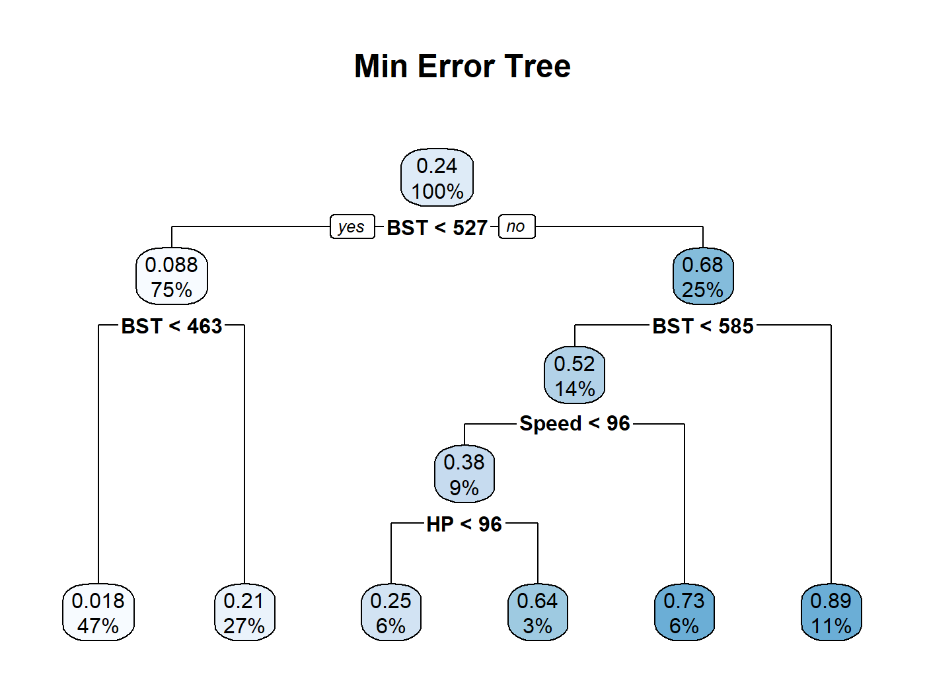
For this project, I decided to leave the cutoff at the default 0.5. I chose to run three different selection models: forwards, backwards, and stepwise. They all performed pretty similarly with forwards and stepwise having an ERR of 7.3% and backwards having an ERR of 6.9%. However, one caveat I should mention is that backwards selection took almost half a day to run while forwards and stepwise took only a few hours in comparison. All three models had the same specificity, being able to detect 95-96% of Pokémon being unviable given that they were indeed unviable. Meanwhile backwards selection had a higher sensitivity at 84.9%, which is about 5% higher than the other two models. While in the backwards selection was the most accurate, I wouldn’t voluntarily run it again unless I learned how to use threading in R to speed up the process since I don’t feel like that the extra accuracy is worth the amount of time it took to generate the model.

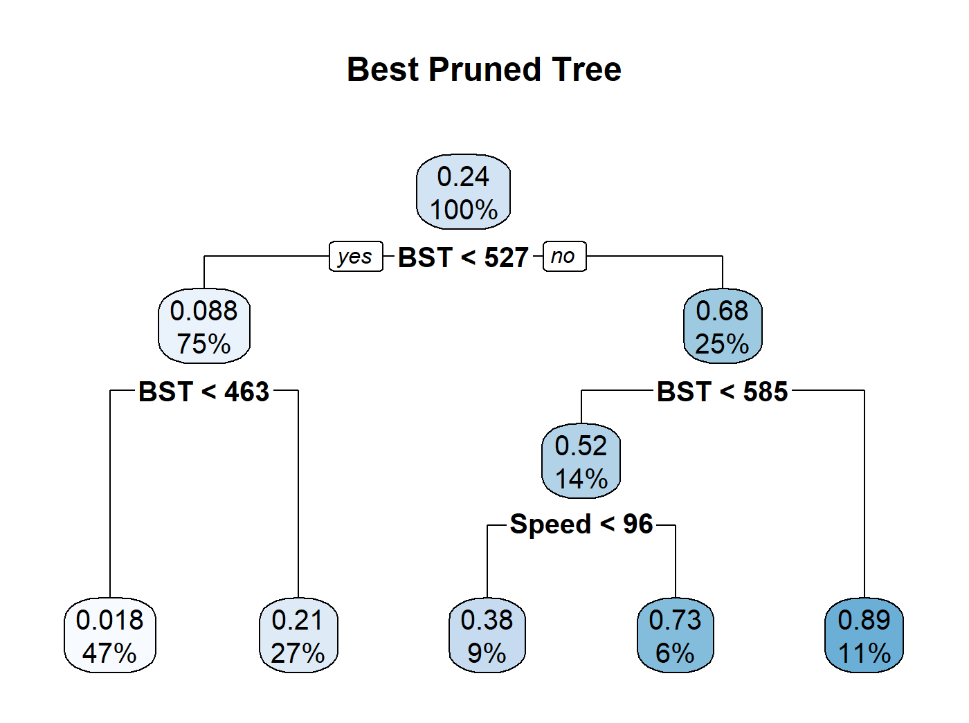
The choice of abilities that had the biggest impact on a Pokémon’s viability among all three models was strange from the viewpoint of a dedicated Pokémon fan such as myself. Abilities that have no effect in battle such as Pickup and Honey Gather were considered to have a positive effect on a Pokémon’s viability while abilities that most players would consider good such as Snow Warning or Skill Link were a detriment instead. Regardless of these odd glaring errors, the model was generally able to correctly predict which abilities were good, such as Regenerator and Unaware, and which ones were bad, such as Slow Start and Truant. There are several reasons I could think as to why there is quite a bit of variability in what abilities were considered good or not. First, Pokémon can have a choice of up to three different abilities, but certain Pokémon might have one very good ability and two awful ones that it would never use and vice versa, which means that common abilities could have their viability ranking skewed based on how the abilities are distributed. Another reason is that certain abilities are what we would call ‘Signature Abilities’ meaning that only a small group of related Pokémon have them. This means that the sample size is too small for the models to be able to make any meaningful predictions unless the Ability in question had a huge and obvious effect on the ranking of a Pokémon such as the ability Slow Start.

When it came to types, all three models consistently put Poison, Water, Fighting, Steel, and Dark as the top 5 types with Fairy and Ground scoring 6th and 7th. The models generally did a good job with Water, Steel, Fairy, and Ground being considered really good, but Poison and Dark are usually considered middling types. The models, however, were only able to pick out types that have a positive effect on a Pokémon’s viability but not pick out which types generally would be considered to have a negative effect such as Ice or Bug.

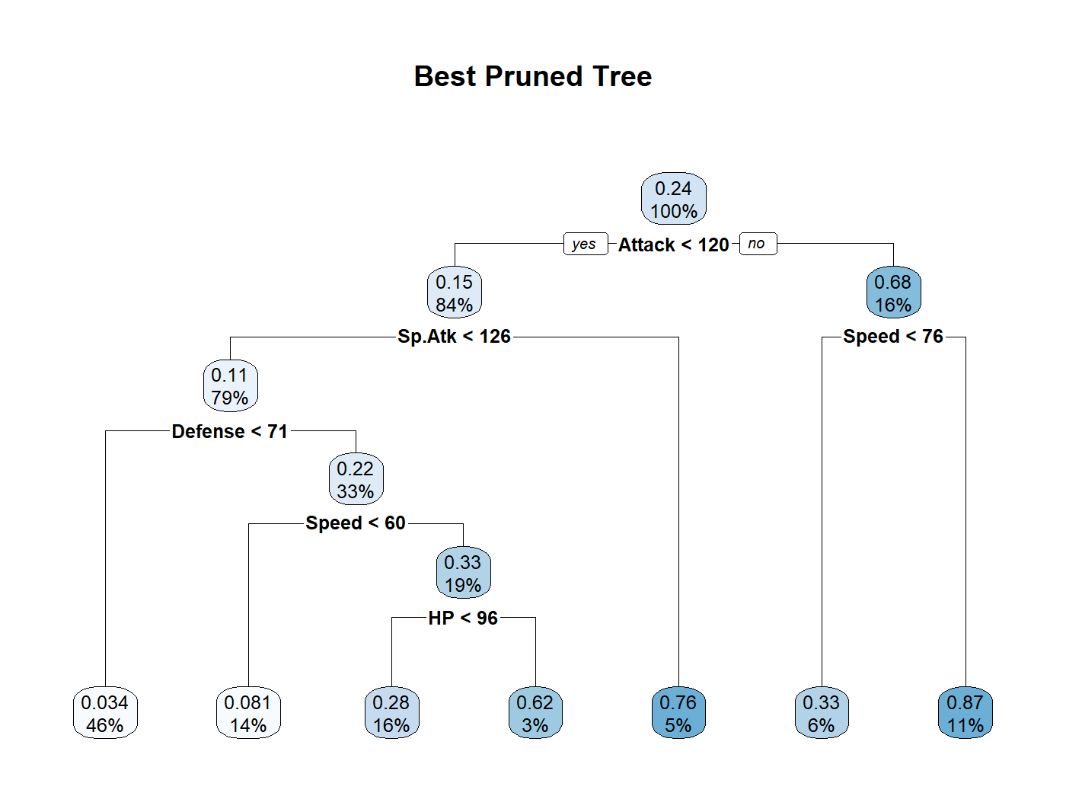
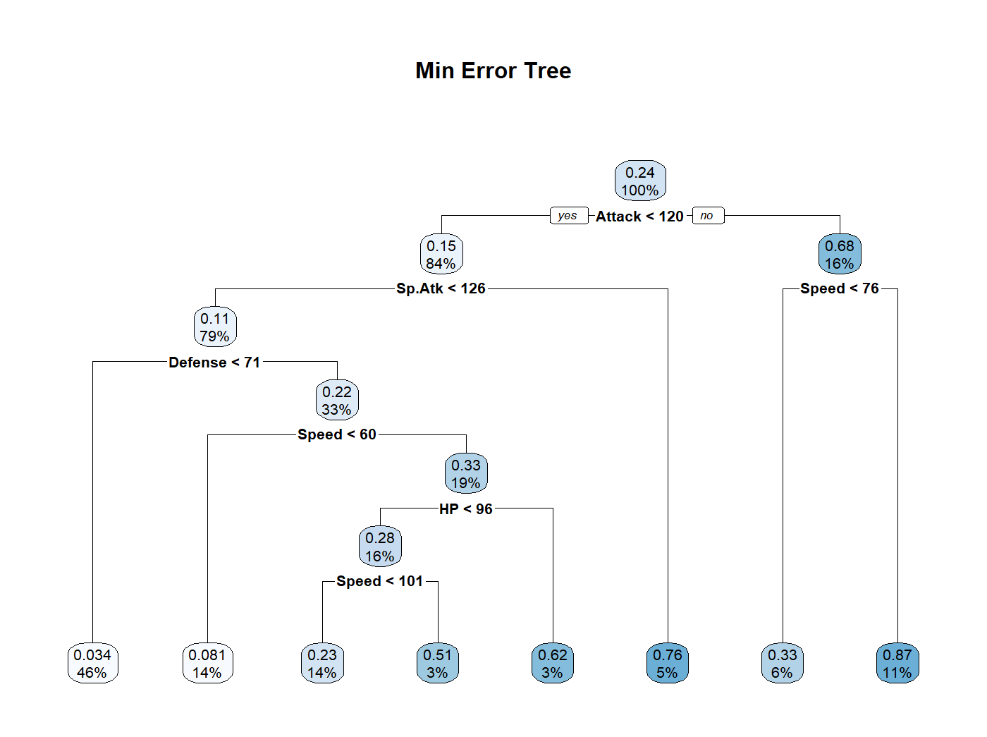
Lastly when it came to stats, the three models were able to consistently pick out Speed as the best stat overall. However, there was no consensus on the other 5 stats, with each model having a wildly different take on the matter.

**Classification and Regression Trees**





This portion of the project was mainly conducted to get myself reacquainted with CART before revising this project later to delve into more robust algorithms such as Random Forest. The main thing that surprised me was that no matter how much I messed with he controls in R, the CART trees mainly consisted of comparing Base Stat Totals; I was hoping for a more comprehensive tree. After removing BST from the data set and rerunning my analysis I was able to come up with a more interesting tree, but it consisted entirely of making comparisons based on the Pokémon’s specific stats instead.

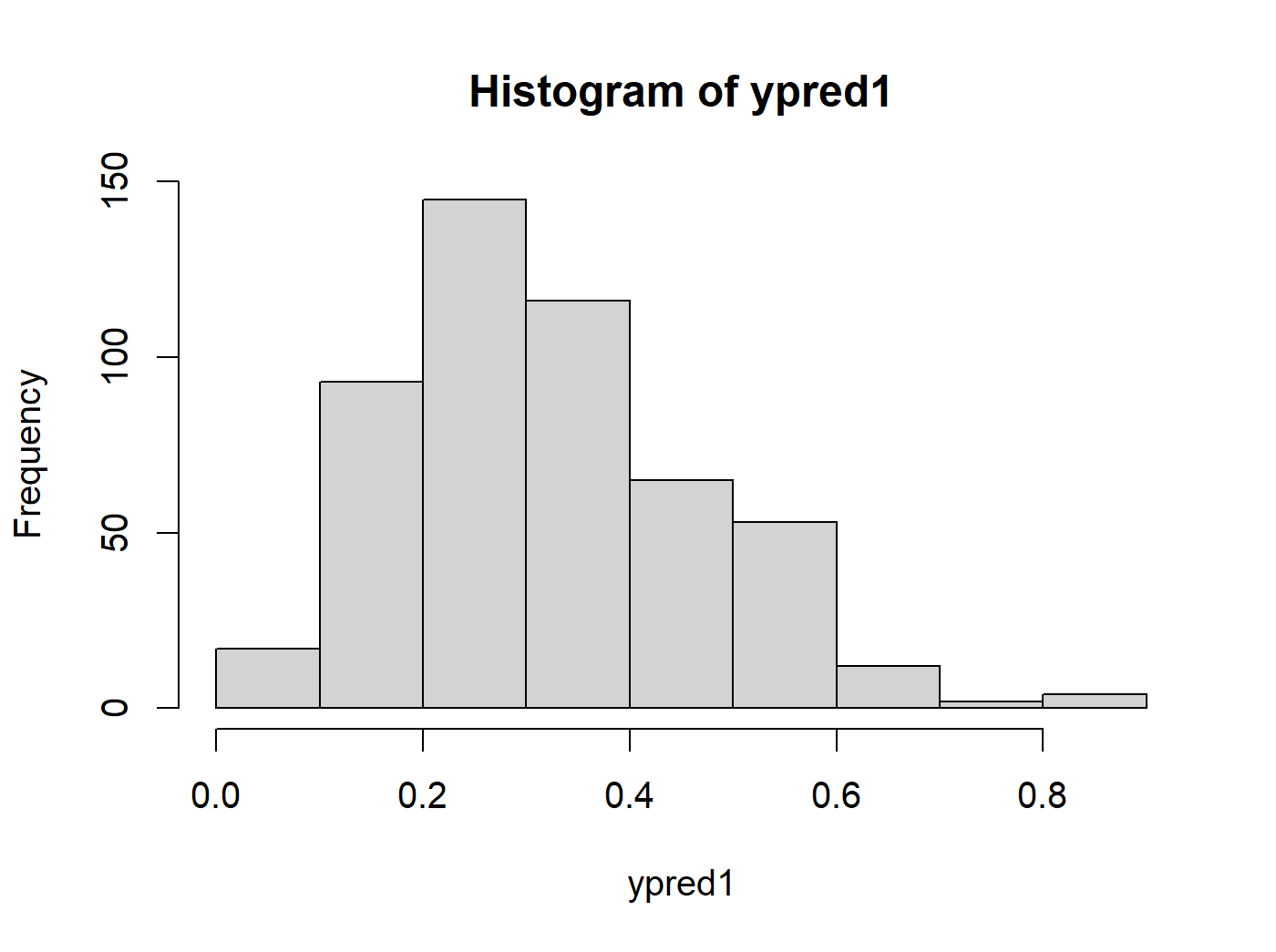


When it came to the ERR of all 4 tree instances, it hovered around 11-12% with sensitive and specificity hovering around 60-70% and 90-95% respectively. As a result of this light experiment, I am considering removing BST in future iterations of this project.

**K-nearest Neighbors**

A screen shot of a computer

Description automatically generated



A computer code with numbers and letters

Description automatically generated

My last classification model was K-nearest neighbors. What I found most interesting about this model was that it had to worse error over all at about 14%. This was mainly because it had a very low sensitivity but a high specificity, meaning that this model was not able to accurately detect which Pokémon were viable while being much better at predicting which Pokémon were not viable than the other models. I believe that the reason for this dichotomy was that one, most Pokémon are actually not viable in general so there’s a lot more neighbors to compare as opposed to viable Pokémon and two, most viable Pokémon have very specific niches that they fill that most other Pokémon cannot thus making it harder to make comparison between viable Pokémon.