Men's March Madness 2022 Machine Learning Predictions

Samuel Johnson
Utah State University
Logan, Utah, United States
sam.johnson1357@gmail.com

Abstract

Keywords: basketball, neural networks, data mining, machine learning, classification, march madness

ACM Reference Format:
Samuel Johnson. 2022. Men's March Madness 2022 Machine Learning Predictions. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Every year Kaggle holds a competition to see who can predict the outcomes of the annual March Madness tournament with the most accuracy. The challenge is notoriously difficult. Every year there are so-called "Cinderellas", which are teams that do extraordinarily well despite low expectations. In this paper I will perform data exploration, fit a random baseline model, and then fit some machine learning models.

This research isn't particularly useful in the grand scheme of things. It is, however, useful for reasons of entertainment and gambling. Every year millions of people fill out brackets in an attempt to predict the outcome of every game correctly. Additionally, many place bets on the outcomes of games. The goal of this research is to see if we can improve March Madness predictions using data from the regular season and machine learning techniques.

1.1 Evaluation Metric

The metric of evaluation for the competition is the log loss, also known as cross entropy. The formula for binary log loss is given below.

$$\text{LogLoss} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \cdot \log_e(\hat{y_i}) + (1-y_i)\cdot \log_e(1-\hat{y_i})\right]$$

Below we also plot a visual of the log loss metric for cases where the true class if positive (1) and negative (0).

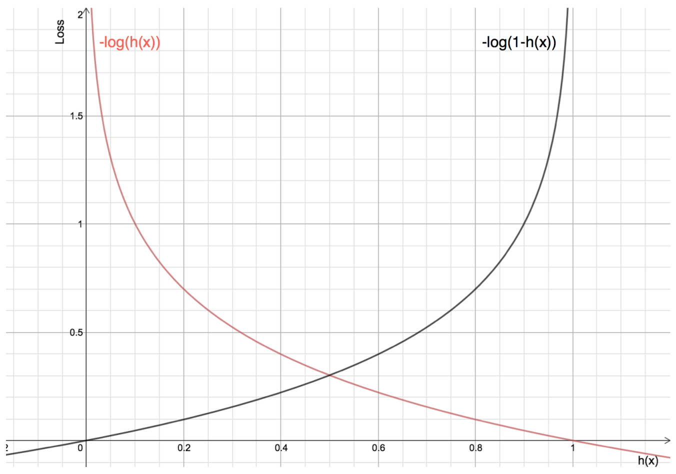


Figure 1. Log loss visualization

If the correct classification is 1, we can see that the log loss reaches a value of 0 when our predicted probability is 1. On the other hand, the log loss approaches infinity as our predicted probability approaches 0.

If the correct classification is 0, we can see that the log loss reaches a value of 0 when our predicted probability is 0. On the other hand, the log loss approaches infinity as our predicted probability approaches 1.

Basically, misclassifying an observation by a large amount is exponentially more harmful than misclassifying an observation by a small amount.

1.2 Description of the Data

Kaggle provided historical game data for several years for both regular season games and tournament games. The features for the winning team are given below. Note that the same features exist for the losing team in each game and they are appended with an *L* instead of a *W*. Some of these features were used by the model we ended up with while most of them were ignored.

- *WTeamID*: The ID

- *WScore*: The number of points scored
- *WFGM*: Field goals made
- *WFGA*: Field goals attempted
- *WFGM3*: Three pointers made
- *WFGA3*: Three pointers attempted
- *WFTM*: Free throws made
- *WFTA*: Free throws attempted
- *WOR*: Offensive rebounds
- *WDR*: Defensive rebounds
- *WAst*: Assists
- *WTO*: Turnovers committed
- *WStl*: Steals
- *WBlk*: Blocks
- *WPF*: Personal fouls committed
- *Seed*: The seed of the team entering the tournament
- *538Ranking*: Proprietary ranking of fivethirtyeight.com.

2 Data Exploration

We can see in figure 2 that the number of basketball teams has increased steadily over the past few decades. This will result in intense competition for the current year.

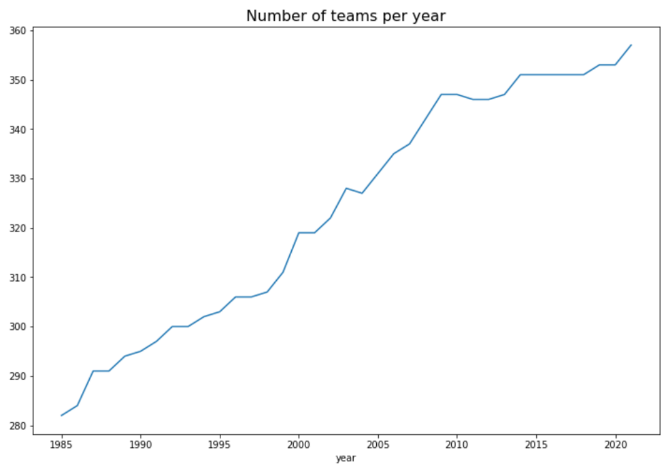


Figure 2. Number of teams per year

We can see in figure 3 that the mean percentage of field goals is around 50% for winning teams and 40% for losing teams. Making a high percentage of shots substantially increases a team's chance of winning.

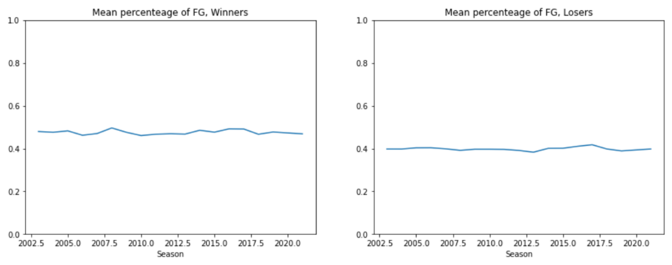


Figure 3. Mean percentage of field goals

We can see in figure 4 that the likelihood of a team winning in the first two rounds of the tournament steadily increases the lower their seed. Thus, seed is an excellent predictive variable of tournament success.

It's also worth noting that a surprisingly large number of seed 15 teams are winning in the second round. These are the "Cinderella" teams that we mentioned earlier.

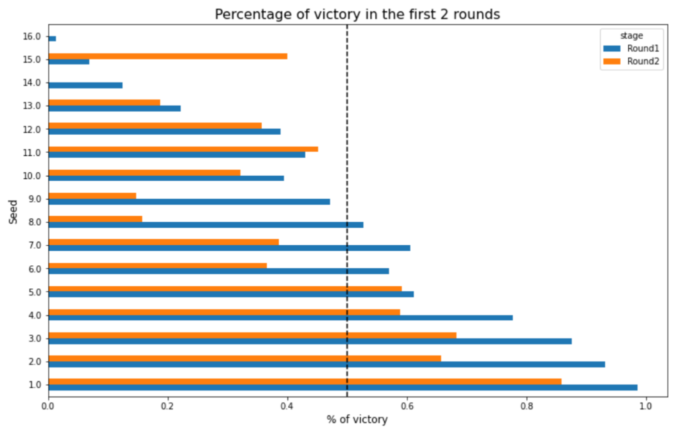


Figure 4. Percentage victory by seed

We can see in figure 5 that the point difference between any two teams during the regular season is usually quite small. In fact, half of the games ended with a point difference of less than ten. Nearly all of the games ended with a point difference of less than 30.

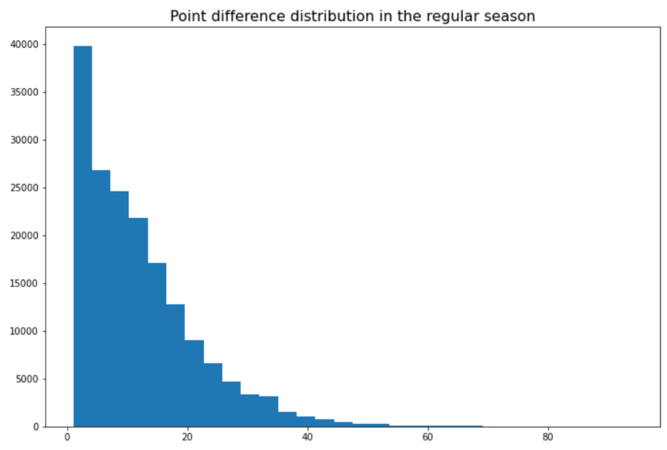


Figure 5. Point difference distribution

We also investigated how playing at home vs away affects the point difference. Figure 6 shows that the home team gets substantially more points on average than the away team. This advantage appears to disappear when playing on a neutral court.

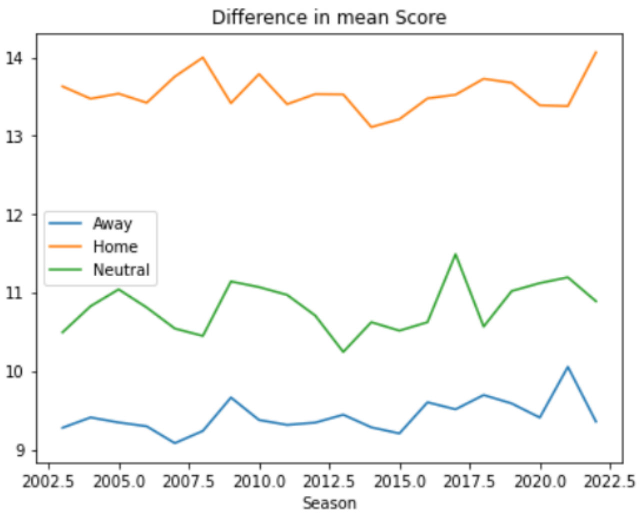


Figure 6. Home vs away point differences

3 Feature Engineering

We used the features provided in the dataset to perform feature engineering in order to create simplistic yet useful features.

The following calculations were performed to create the *WinRatio* variable and the *GapAvg* variable, which will be key features later on.

$AverageScoreGap = \frac{1}{n} * \sum_{i=1}^{n} WScore_i - LScore_i$
$GapWins = AverageScoreGap$ for winning games
$GapLosses = AverageScoreGap$ for losing games
$WinRatio = \frac{NumWins}{NumWins+NumLosses}$
$GapAvg = \frac{NumWins*GapWins-NumLosses*GapLosses}{NumWins+NumLosses}$

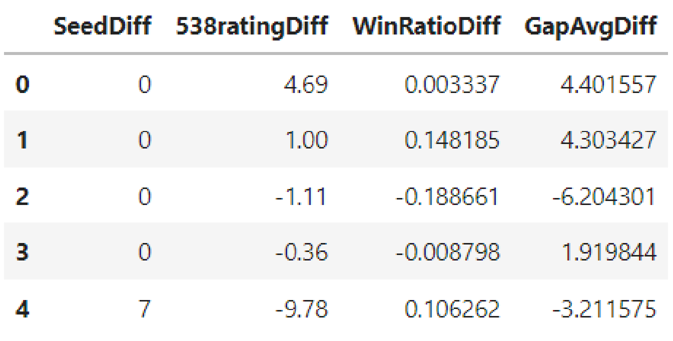
Then we separate *GapAvg*, *Seed*, and *WinRatio*, and *538rating* into two separate variables for winning and losing games respectively. The resulting variables are given below:

- *GapAvgW* is the gap average for winning games.
- *GapAvgL* is the gap average for losing games.
- *SeedW* is the seed of the winning team.
- *SeedL* is the seed of the losing team.
- *WinRatioW* is the win ratio of the winning team during the season.
- *WinRatioL* is the win ratio of the losing team during the season.
- *538ratingW* is the rating of the winning team during the season.
- *538ratingL* is the rating of the losing team during the season.

To create a learnable feature space, we subtract the losing team features from the winning team features and create difference features. The following features have now been created:

$SeedDiff = SeedW - SeedL$
$538ratingDiff = 538ratingW - 538ratingL$
$WinRatioDiff = WinRatioW - WinRatioL$
$GapAvgDiff = GapAvgW - GapAvgL$

We now have a simplified dataset of just four features that we will use to build a March Madness classification model. An image of what the dataset looks like in Python can be seen in figure 7.



	SeedDiff	538ratingDiff	WinRatioDiff	GapAvgDiff
0	0	4.69	0.003337	4.401557
1	0	1.00	0.148185	4.303427
2	0	-1.11	-0.188661	-6.204301
3	0	-0.36	-0.008798	1.919844
4	7	-9.78	0.106262	-3.211575

Figure 7. Dataset Image

The label we are trying to predict is *Win*, which takes on a value of one when the first listed team won and a value of zero when the second listed team won.

4 Baseline Models

The first baseline model I submitted was a model that simply predicted a 0.5 probability for every matchup. The model scored a log loss of 0.693. The second model I submitted predicted a random probability between 0 and 1 for each matchup. This resulted in a log loss of 1.042.

5 Machine Learning Model Evaluation

In an effort to achieve maximal prediction accuracy, we fitted five different machine learning models: logistic regression, decision tree, random forest, gradient boosting machine, and neural network. We then sought to tune the hyperparameters of each model using 10-fold cross validation. We go over the results below.

5.1 Logistic Regression

The first model we chose was a simple logistic regression model. We tuned the model using the hyperparameter C, which employs regularization on the dataset. The higher the value of C, the smaller the regularization effect. Conversely, the lower the value of C, the greater the regularization effect.

We ran cross validation using 10 different values of C ranging from 0.01 to 5 in equally spaced intervals. In figure 8 we show the effect of C on the cross validated log loss. We can see that as C increases, the log loss increases slightly. Thus, we conclude that a highly regulating C value of 0.01 works the best, yielding an impressive log loss value of 0.551.

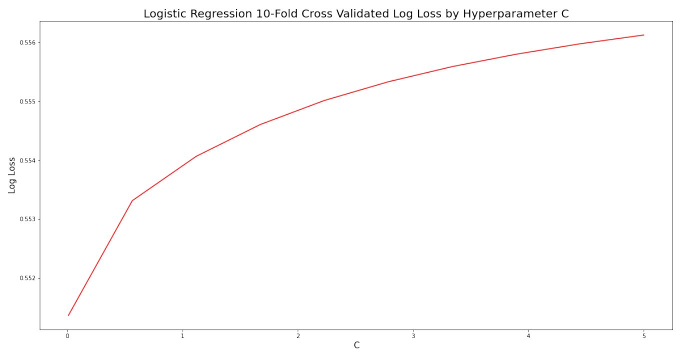


Figure 8. Logistic Regression Cross Validation

5.2 Decision Tree

The next model we employed was the decision tree model. The relevant hyperparameter for the decision tree was the max depth. In general, it is expected that as we allow a tree to grow deeper, it will eventually overfit to the training data and be unable to generalize to unseen testing data.

We ran cross validation on max depth values going from one to nine. In figure 9 we can see the effect that the max depth had on the cross validated accuracy. We can see that the log loss decreases slightly as we move from a max depth of 1 to a max depth of 2. After that, it consistently rises as the max depth increases. After we reach a max depth of 9, it is clear that we are massively overfitting the model as log loss exceeds 7. The best max depth value is 2, which leads to a log loss of 0.579.

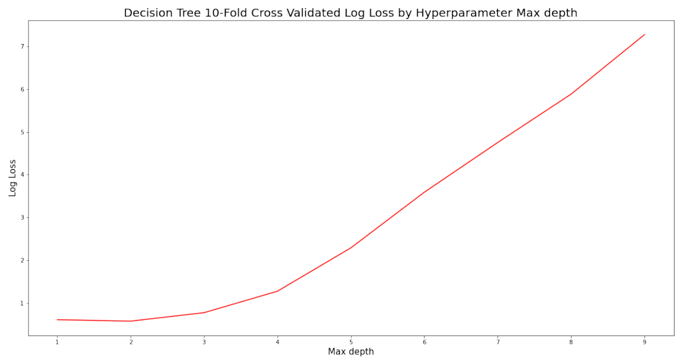


Figure 9. Decision Tree Cross Validation

We plot the chosen decision tree model in figure 10. Interestingly, it uses only one attribute, *538RatingDiff*, to determine which team is likely to win. Apparently the other features in the model provide very little additional discriminatory power.

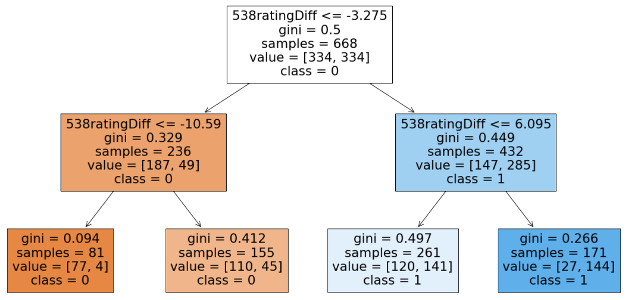


Figure 10. Decision Tree Graphic

5.3 Random Forest

The next model we tried was the random forest classifier. The relevant hyperparameter for the random forest is the number of estimators, which is simply the number of uncorrelated decision trees that are built to make predictions on future unseen data.

We ran cross validation on the following hyperparameters: 10, 60, 110, 160, and 210. Our cross validated results can be seen in figure 11. We can see that log loss remains very high when there are only 10 estimators. After we increase the number of estimators to 60, the log loss rapidly decreases, after which it remains constant. Log loss is ultimately lowest at 0.587 with 210 estimators.

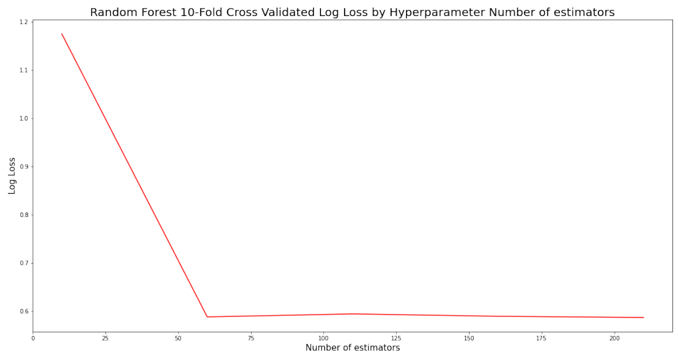


Figure 11. Random Forest Cross Validation

5.4 Gradient Boosting Machine

The next model we tried was the gradient boosting machine. The gradient boosting machine works by fitting a decision tree to make a prediction. Then it calculates the error and fits another decision tree to predict the residual of the original prediction. It continues *boosting* for *n* iterations. Thus, the relevant hyperparameter is the number of boosting rounds.

We used the following values in our hyperparameter optimization search: 10, 60, 110, 160, and 210. The results of the optimization can be seen in figure 12. Surprisingly, the model performs best with only 10 estimators. After we increase from 10, the model begins overfitting to the training data and we begin to see somewhat higher cross validation log loss. Thus, the best hyperparameter is 10 with a resulting log loss of 0.582.

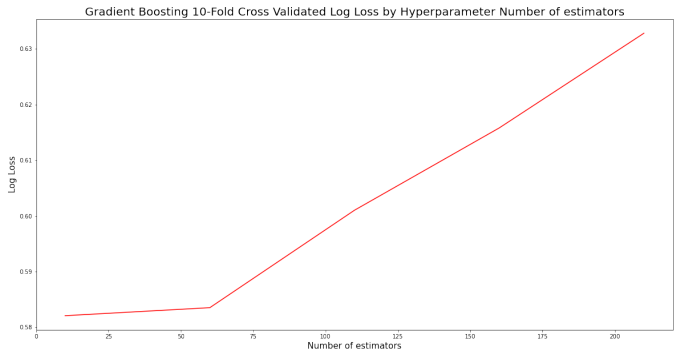


Figure 12. Gradient Boosting Cross Validation

Figure 13 shows the importance of the various features calculated by the chosen random forest model. They are calculated based on the total reduction of uncertainty over all of the fitted trees. We can see that the *538RatingDiff* is once again the most important attribute. The other attributes don't appear to be very important.

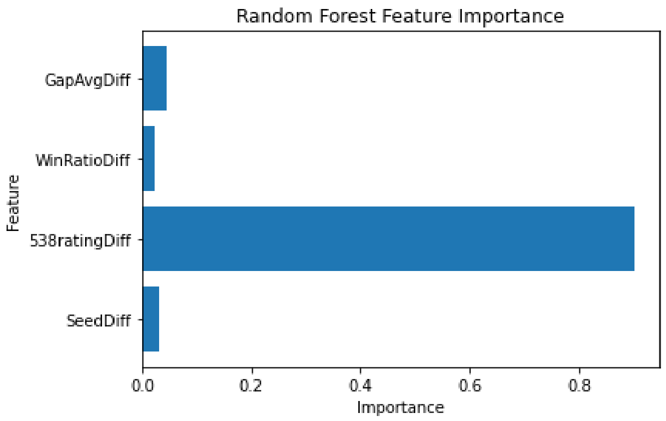


Figure 13. Gradient Boosting Feature Importance

5.5 Neural Network

The next model we used was a neural network. We used the ReLU Activation function, an Adam optimizer, an L2 regularization term of 0.0001, a batch size of 200, a learning rate of 0.001, and training for 200 iterations.

Due to limited computational power, the only hyperparameter that we varied was the number of layers and the size of each layer in the neural network. We randomly selected ten different neural network structures. The structures were required to have between 1 and 4 layers. Additionally, each layer was required to have between 50 and 150 neurons.

For each structure, we calculated the cross validated log loss and represented the results as a horizontal bar chart in figure 14. We can see that the results between the different models don't differ too much in their accuracy with each model's log loss hovering around 0.6. Interestingly, we can see that the models with only one layer seemed to perform better than those with multiple layers. Ultimately, the model that performed the best was the model with 89 neurons and just one hidden layer. It yielded a log loss of just 0.555, putting it almost on par with logistic regression.

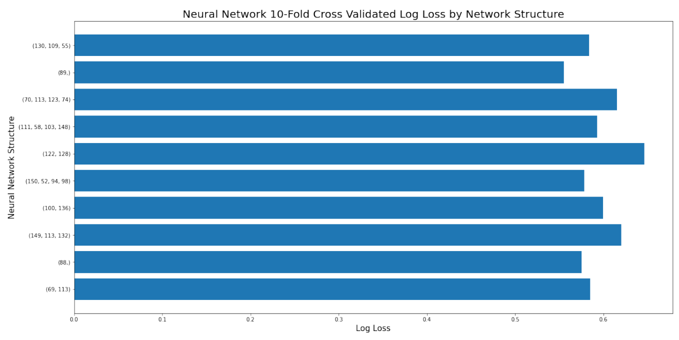


Figure 14. Neural Network Cross Validation

5.6 Model Results Comparison

Below, we compare the highest cross validated accuracy results from each of our models.

Model	Lowest Log Loss
Logistic Regression	0.551
Decision Tree	0.579
Random Forest	0.587
Gradient Boosting Machine	0.582
Neural Network	0.555

It is interesting to note that logistic regression actually performed the best with a log loss of just 0.551. This is somewhat surprising as logistic regression is perhaps the simplest classification method in machine learning. We would normally expect more sophisticated methods like random forest, gradient boosting, or neural networks to have higher performance. In this case, however, with only four features, it makes sense that simple models would do well.

5.7 Kaggle Competition Results

Because the logistic regression model was our top performing model using cross validation, that was the model we chose to submit to the competition. We ended up doing quite well, scoring in the top third of teams.

6 Conclusion & Future Work

In this paper we analyzed data resulting from NCAA level basketball games in an effort to see if we could predict the winners of the annual march madness tournament. We analyzed the data, performed feature engineering, and then built models to accurately classify the data.

In the end, we did moderately well in the Kaggle competition. We achieved excellent performance with the Logistic Regression model, which returned a log loss of 0.551. For reference, this is about 70% accuracy in terms of correctly predicting the winner of a given basketball game. We note that predicting basketball games is notoriously difficult, as there are frequent upsets and so-called Cinderellas. Perhaps it is possible to get slightly higher accuracy with better feature engineering but most likely not significantly better.

Hopefully this research can help basketball fans to fill out their brackets more accurately and help sports gamblers to make more informed bets. If could also have the possible effect of making gambling markets more efficient if it becomes more widely adopted.

In the Introduction we described the various attributes included in the dataset. We opted only to use differences in scores, the percentage of won games, the 538Rating, and the seed. For future research we might include more attributes like data on blocks, field goals, three pointers, steals, rebounds, and assists. Other features might help as well like the records of the coaches, the age of the players, trends over time, and more. There is definitely a lot of future work to be done in the area.

7 References

Some plots in the Data Exploration section were borrowed from the following Kaggle notebook:

Rahman, K. (2022, March 9). *2022 March Mania - Quick EDA & FE.* Kaggle. Retrieved April 25, 2022, from https://www.kaggle.com/code/kalilurrahman/2022-march-mania-quick-eda-fe

Some of the feature engineering ideas were borrowed from the following Kaggle notebook:

Viel, T. (2022, February 25). *Using last year's 2nd Place.* Kaggle. Retrieved April 25, 2022, from https://www.kaggle.com/code/theoviel/using-last-year-s-2nd-place/notebook