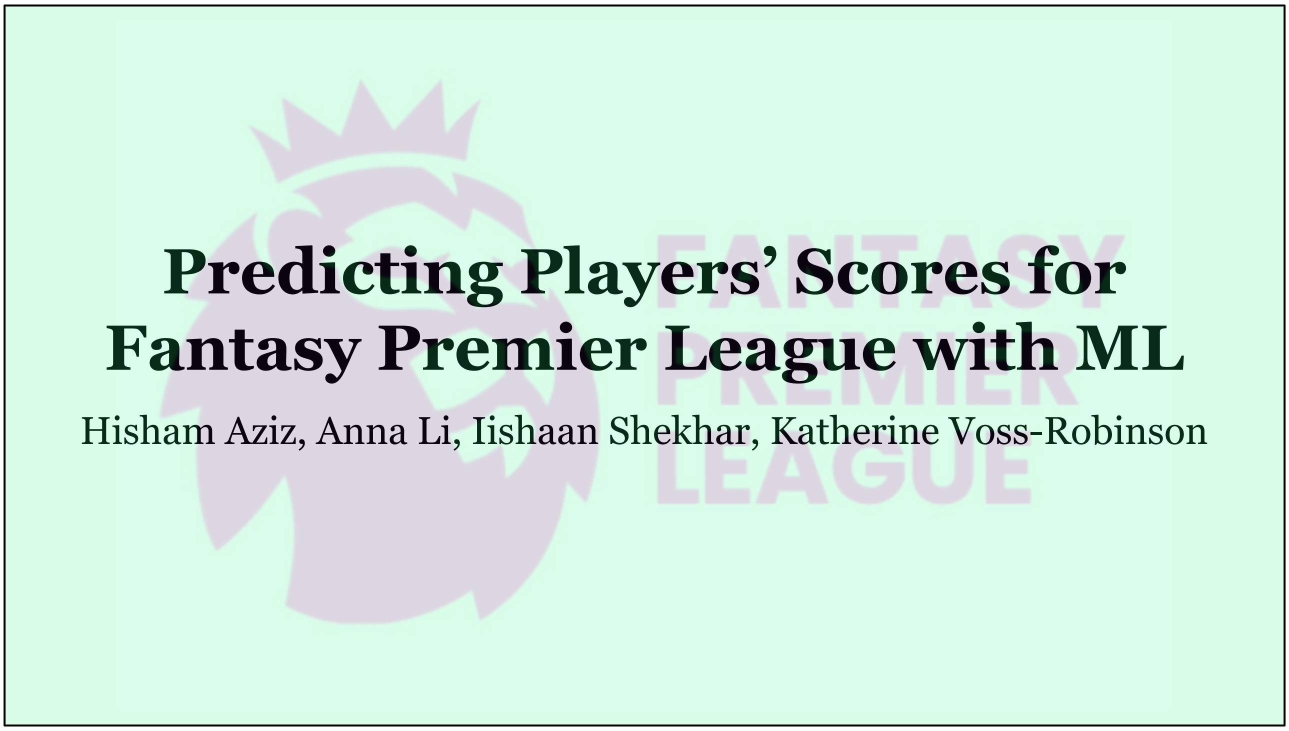
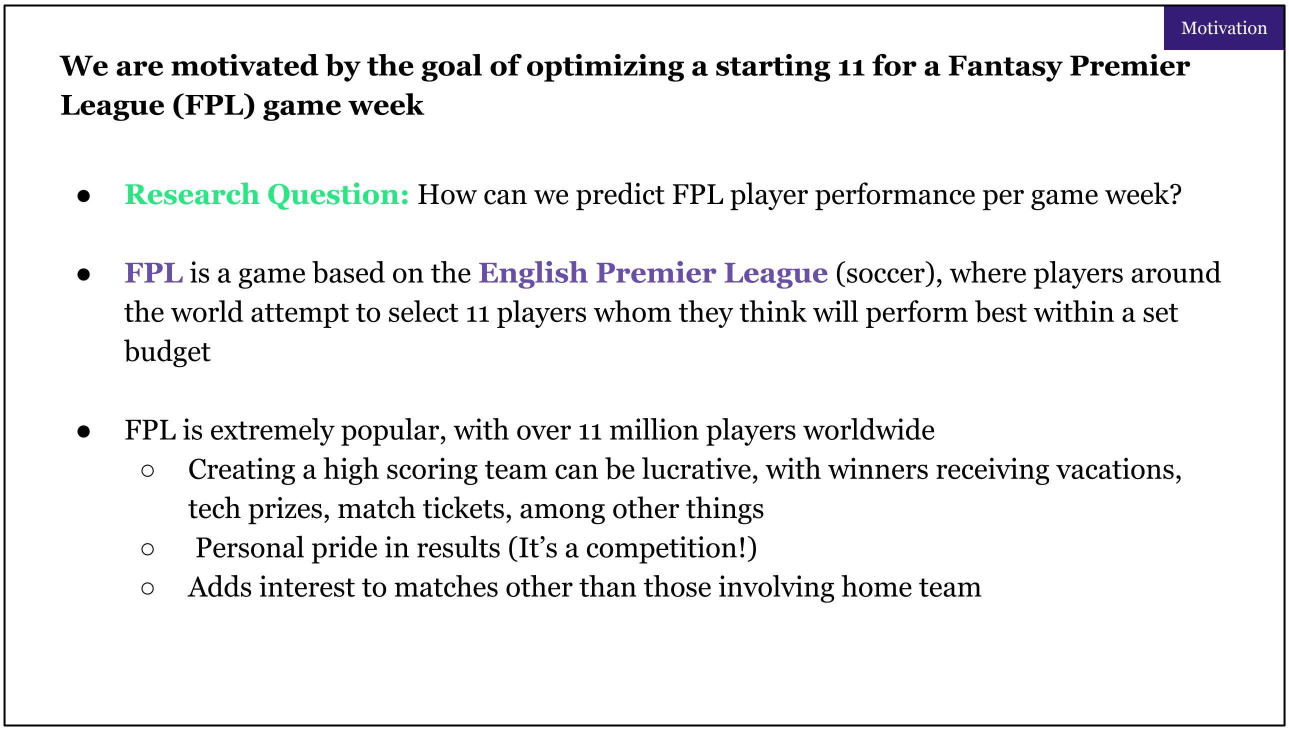


Predicting Players’ Scores for Fantasy Premier League with ML

Hisham Aziz, Anna Li, Iishaan Shekhar, Katherine Voss-Robinson

Speaker: Hisham



Motivation

We are motivated by the goal of optimizing a starting 11 for a Fantasy Premier League (FPL) game week

- **Research Question:** How can we predict FPL player performance per game week?
- **FPL** is a game based on the **English Premier League** (soccer), where players around the world attempt to select 11 players whom they think will perform best within a set budget
- FPL is extremely popular, with over 11 million players worldwide
  - Creating a high scoring team can be lucrative, with winners receiving vacations, tech prizes, match tickets, among other things
  - Personal pride in results (It's a competition!)
  - Adds interest to matches other than those involving home team

Speaker: Hisham
For our final project we chose to take up the task of using Machine Learning techniques to create the optimum FPL team. For those that are not aware, FPL, or Fantasy Premier League, is a highly popular online fantasy sports game based on the English Premier League, one of the world's most followed football leagues.

In FPL, participants assemble virtual teams of real Premier League players and earn points based on the real-life performances of those players. The game's popularity stems from its ability to engage football fans on a deeper level, allowing them to become virtual managers and strategists. This platform is so popular that there are actual pundits and writers that release weekly articles on different FPL strategies. The motivation of using ML to beat humans at a game of strategy is as old as the concept of AI itself, take the examples of Deep Blue, AlphaStar, and AlphaGo, three notable AI systems used to beat humans at Chess, Starcraft and Go.

Of course this a substantial task, so for the purpose of this class we took on only the first step: predicting individual player scores by training the model on previous seasons.
(next slide please)

We are not the only people who feel that this task is well-suited for ML

- Kaggle Fantasy Premier League
  - Contains consolidated historical Premier League data
- Predicting Fantasy Football Teams using Regression with Regularisation
  - Uses only linear regression but adds an optimization algorithm to select a full starting team
- FPL Predictions Using Same Season Data
  - Leverages data from within a season to predict performance with several advanced modeling techniques (e.g., FFNN, XGBoost, and LSTM)

Speaker: Hisham

Despite my rhetoric on the links between AI and gaming systems, at first we were unsure whether this task was well suited to be solved by ML. However, once we started doing some research we saw that there have been a host of other attempts to solve this problem and in fact there are data sets on Kaggle, consolidated for training models for this very purpose. (next slide please and over to you Anna)

To tackle this problem, we evaluated eight model architectures

- **Target Variable:** Total points in a given game week at the player level
- **Evaluation Methodology:** MSE and MAE (with explainability as a tie break)
- **Results:**
  - Created eight model architectures (simple linear regression, RNN/LSTM, gradient boosted regression, FFNN, KNN, decision tree, XGBoost, and random forest regression) in addition to our baseline
  - Our baseline had an MSE of 5.59 and an MAE of 1.60
  - All models beat our baseline, but our random forest model beat it by the most with an MSE of 3.85 and an MAE of 1.05

Speaker: Anna

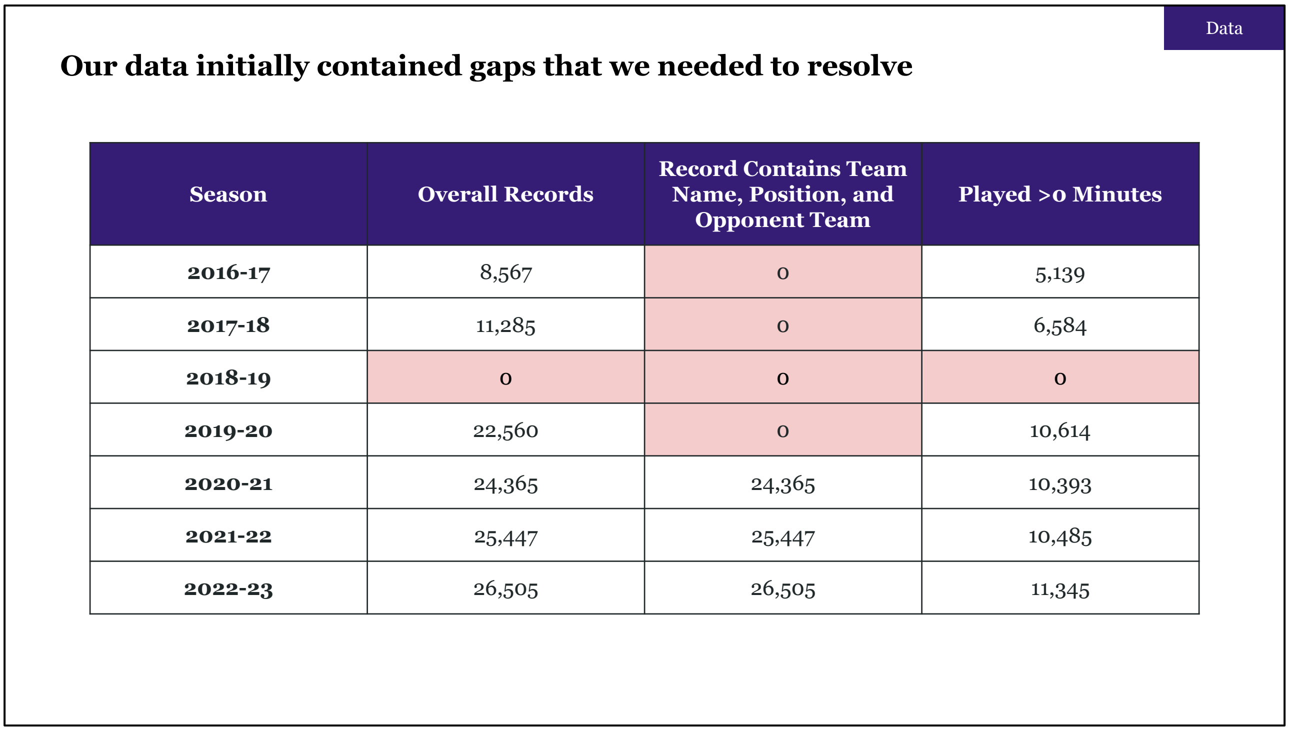
We utilized an FPL library containing game week-level data from 2016 through the present to build our models

Data Source: a GitHub repository from a user who did some cleaning and aggregation on FPL data

Columns:
In total, we have 37 columns (split between **categorical** and **continuous** variables) and ~140k rows (one for each player and game week in the data)

['season_x', 'name', 'position', 'team_x', 'clean_sheets', 'opponent_team', 'opp_team_name', 'round', 'was_home', 'GW', 'assists', 'bonus', 'bps', 'creativity', 'element', 'fixture', 'goals_conceded', 'goals_scored', 'ict_index', 'influence', 'kickoff_time', 'minutes', 'own_goals', 'penalties_missed', 'penalties_saved', 'red_cards', 'saves', 'selected', 'team_a_score', 'team_h_score', 'threat', 'total_points', 'transfers_balance', 'transfers_in', 'transfers_out', 'value', 'yellow_cards']

Speaker: Anna



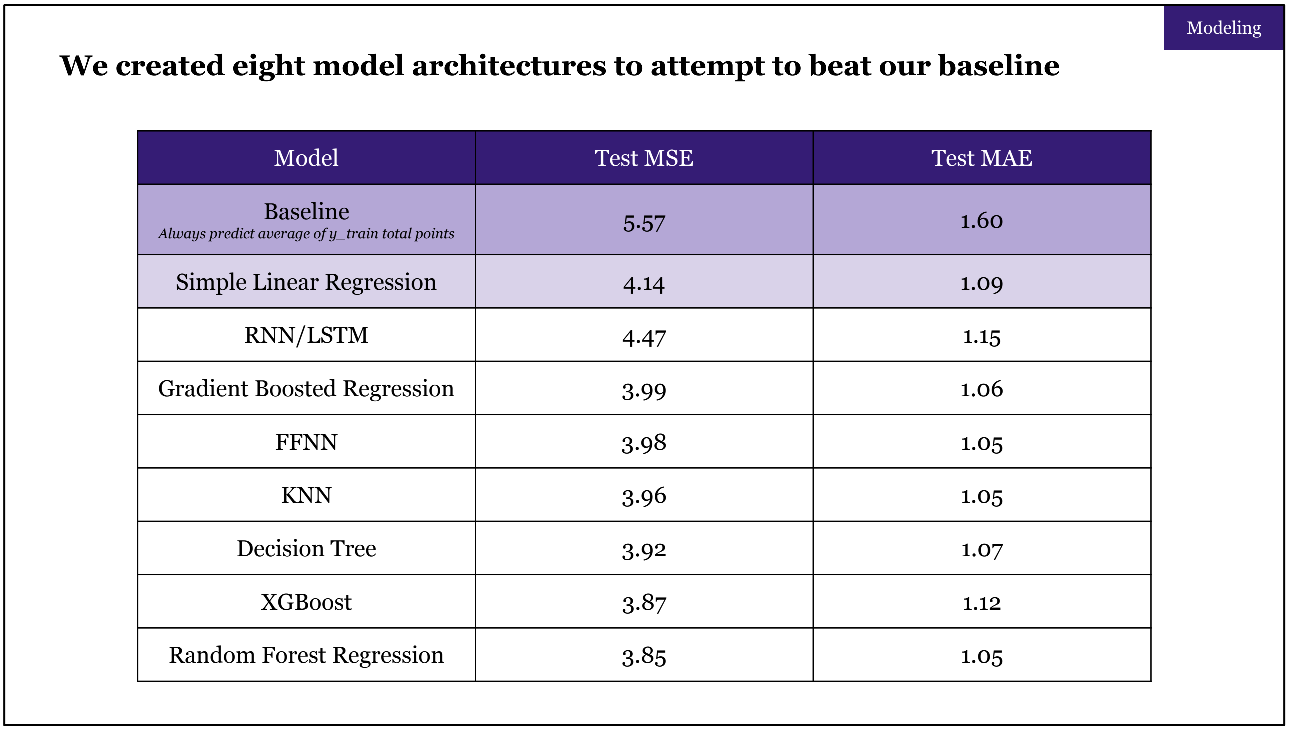
Data

Our data initially contained gaps that we needed to resolve

Season	Overall Records	Record Contains Team Name, Position, and Opponent Team	Played >0 Minutes
2016-17	8,567	0	5,139
2017-18	11,285	0	6,584
2018-19	0	0	0
2019-20	22,560	0	10,614
2020-21	24,365	24,365	10,393
2021-22	25,447	25,447	10,485
2022-23	26,505	26,505	11,345

Speaker: Anna

- Data in early seasons was sparse:
  - 2016-18 data is sparser due to data consolidation issues most likely (showed by >0 mins being less).
  - 2018-23 data seems reasonable-- but 2020-23 data has higher total counts which can partially be attributed to increase from 3 to 5 subs (from covid onwards)
- Data Preparation Steps:
  - Remove special characters in player names
  - Fill in missing season, team, position, and opponent team information
  - Transform “away and home team score” columns into “player and opponent team score” columns
  - Add opponent difficulty column
  - Add a sequence column
  - Transform week-level continuous variables into lagged averages
  - Standardize continuous variables
  - Transform categorical variables into one-hot encodings
  - Split data into train/validation/test sets*
    - Train: 2018-19, 2019-20, and 2020-21 seasons
    - Validation: 2021-22
    - Test: 2022-23



Modeling

We created eight model architectures to attempt to beat our baseline

Model	Test MSE	Test MAE
Baseline *Always predict average of y_train total points*	5.57	1.60
Simple Linear Regression	4.14	1.09
RNN/LSTM	4.47	1.15
Gradient Boosted Regression	3.99	1.06
FFNN	3.98	1.05
KNN	3.96	1.05
Decision Tree	3.92	1.07
XGBoost	3.87	1.12
Random Forest Regression	3.85	1.05

Speaker: Katherine

- Baseline created for simplicity
- MAE = average number of points the prediction is off
- Our table starts with linear regression because it's simplest and then descents in order of test MSE
  - All beat baseline, though random forest was best at predicting points for outlier players
- Why we thought each architecture could be a good choice:
  - **Linear regression** - might be a linear problem space; some problems don't need more complex architecture (also, it's easy to make this)
  - **Neural nets** - if problem is non-linear
    - **RNNs** are great at analyzing sequential data like ours
  - Given our complex space containing a high number of categorical and continuous variables, we thought **decision trees** could be a good choice
    - **Gradient boosting and random forest** are similar but can also use ensemble learning
  - **KNN** drives predictions via similarity to neighbors; our model might be able to profile successful and unsuccessful players to compare to
- Misc. details:
  - **RNN/LSTM** - Players often go through patches of good and bad form.

- RNN models excel in analyzing sequential data so they could be a good choice for predicting EPL fantasy football scores by effectively capturing any patterns of player performance.
- **Decision Trees -** given the high number of categorical and continuous variables, could be a good choice as it could model performance across a very complex space
- **Gradient Boosting (includes XGBoost), Random Forest** - Similar to decision trees, except models could be optimized with ensemble learning
- **FFNN** - if the problem space is nonlinear, this would be key
- **KNN -** predictions are driven by similarity to neighbors; similar players may score a similar number of points

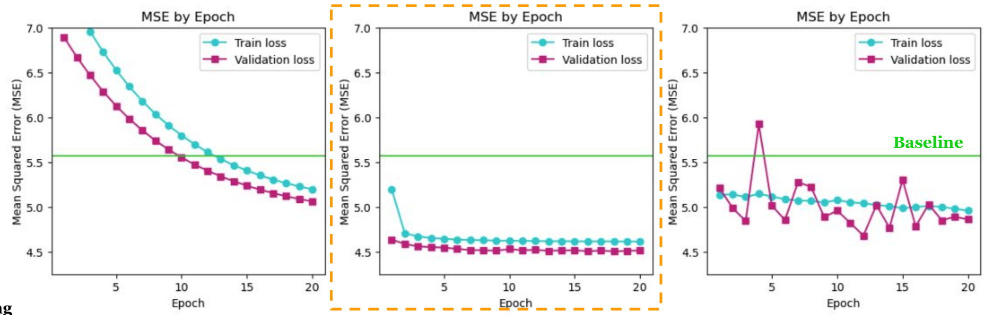
We will give more detail on the tuning we conducted to optimize five of these

	Model	Test MSE	Test MAE
	Baseline *Always predict average of y_train total points*	5.57	1.60
1	Simple Linear Regression	4.14	1.09
2	RNN/LSTM	4.47	1.15
	Gradient Boosted Regression	3.99	1.06
3	FFNN	3.98	1.05
4	KNN	3.96	1.05
	Decision Tree	3.92	1.07
	XGBoost	3.87	1.12
5	Random Forest Regression	3.85	1.05

Speaker: Katherine

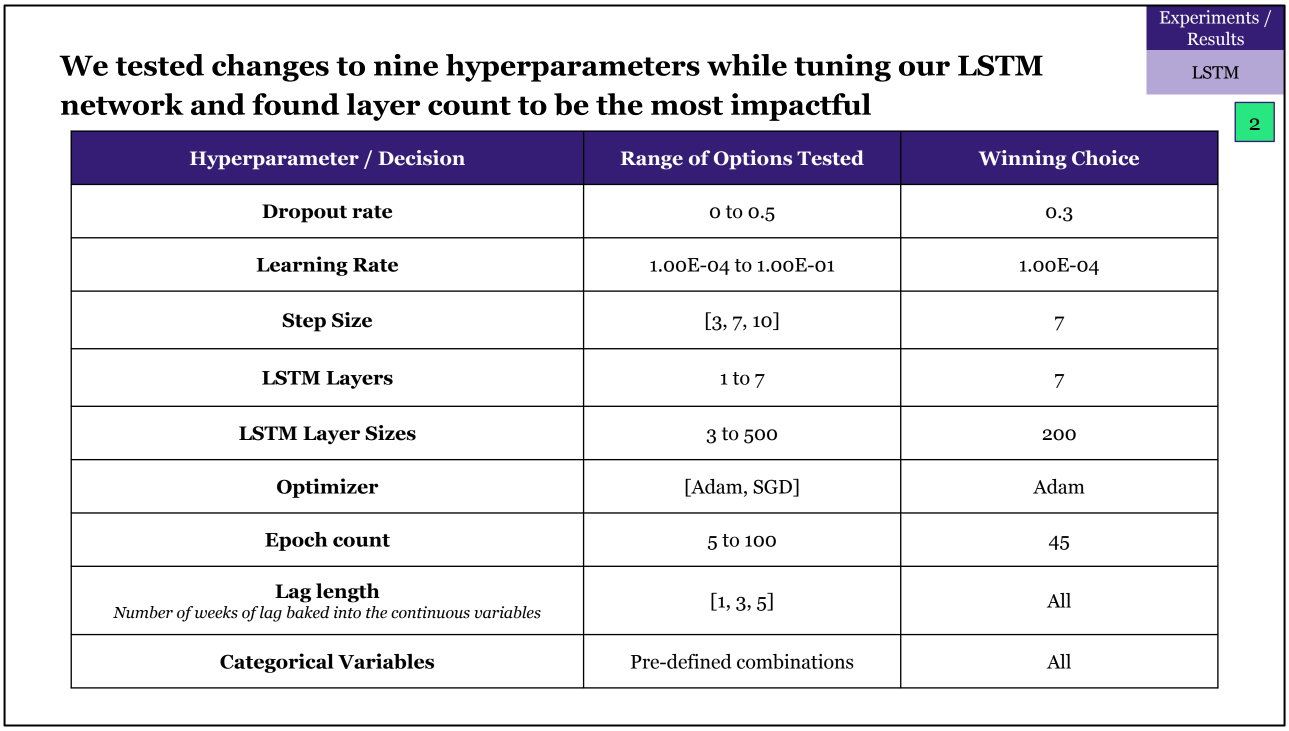
We tuned our optimizer, learning rate, and epoch count to generate our final linear regression model

Hyperparameter	Range of Options Tested	Winning Choice *Circled in green below*
Optimizer	[Adam, SGD]	Adam
Initial Learning Rate	5.00E-06 to 5.00E-02	5.00E-04
Learning Rate Schedule *Decay steps and rate*	Steps: 1k to 100k Rate: 0.75 to 0.95	Steps: 10k Rate: 0.9
Epoch count	5 to 100	20



Initial learning rate: **5.00E-06** **5.00E-04** **5.00E-02**

Speaker: Anna



We tested changes to nine hyperparameters while tuning our LSTM network and found layer count to be the most impactful

Hyperparameter / Decision	Range of Options Tested	Winning Choice
Dropout rate	0 to 0.5	0.3
Learning Rate	1.00E-04 to 1.00E-01	1.00E-04
Step Size	[3, 7, 10]	7
LSTM Layers	1 to 7	7
LSTM Layer Sizes	3 to 500	200
Optimizer	[Adam, SGD]	Adam
Epoch count	5 to 100	45
Lag length *Number of weeks of lag baked into the continuous variables*	[1, 3, 5]	All
Categorical Variables	Pre-defined combinations	All

Speaker: Hisham

- The next architecture we tried was a recurrent neural network using LSTM nodes. This architecture provides us with even more hyper parameters than the Feed Forward Neural Network architecture so tuning was quite a lengthy process.

For this particular project we found that:

- L1 Regularisation was not found to have any positive impact.

- LSTM layers are often accompanied with a Dropout layer, but we achieved the best results with 0.0 Dropout Rates at lower epochs, however, because our training loss rate showed potential for improvement at higher epochs we re-introduced dropout to fight overfitting in our final model.

- Learning rate of 0.001 had the best results and our final model had over 2.2M trainable parameters.

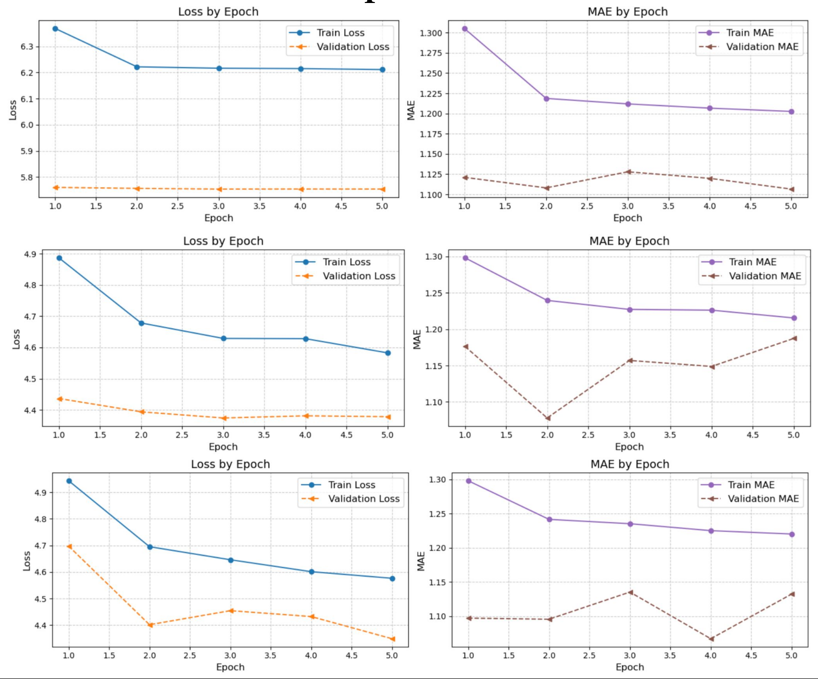
The 7 layer LSTM model had the best performance

Number of LSTM Layers:

3

5

7

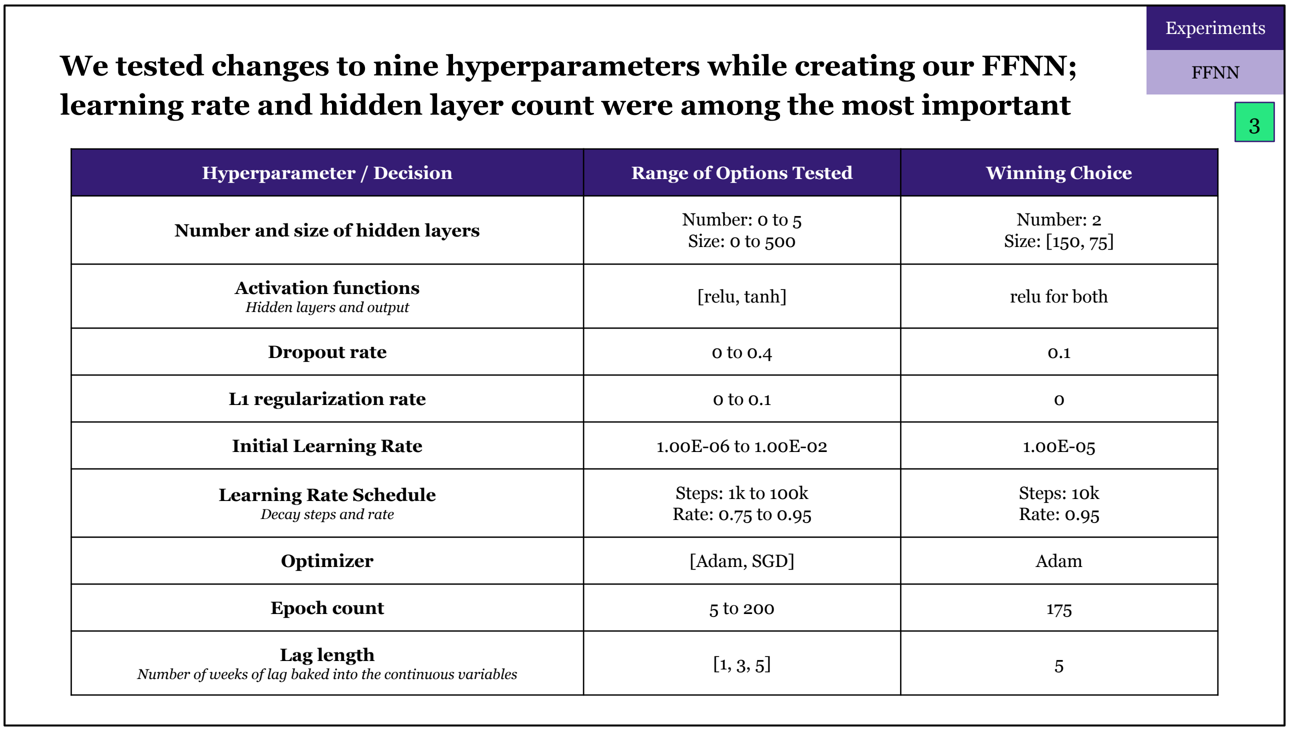


Speaker: Hisham

The most significant result of tuning this model was that we achieved our best results with deeper architectures. Here we show results from training using 3, 5 and 7 LSTM layers with a fixed layer size and time step sequence length.

Further, it is not shown here but an increase the LSTM layer nodes was also seen to have an impact on training loss. Subsequently, we increased layer size to 200 for the final model.

Surprisingly results did not match up to even those achieved by the linear regression model, showing that LSTM was not an ideal fit for our setup. (next slide and over to Katherine)



We tested changes to nine hyperparameters while creating our FFNN; learning rate and hidden layer count were among the most important

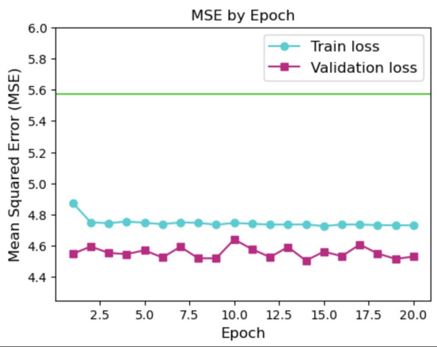
Hyperparameter / Decision	Range of Options Tested	Winning Choice
Number and size of hidden layers	Number: 0 to 5 Size: 0 to 500	Number: 2 Size: [150, 75]
Activation functions *Hidden layers and output*	[relu, tanh]	relu for both
Dropout rate	0 to 0.4	0.1
L1 regularization rate	0 to 0.1	0
Initial Learning Rate	1.00E-06 to 1.00E-02	1.00E-05
Learning Rate Schedule *Decay steps and rate*	Steps: 1k to 100k Rate: 0.75 to 0.95	Steps: 10k Rate: 0.95
Optimizer	[Adam, SGD]	Adam
Epoch count	5 to 200	175
Lag length *Number of weeks of lag baked into the continuous variables*	[1, 3, 5]	5

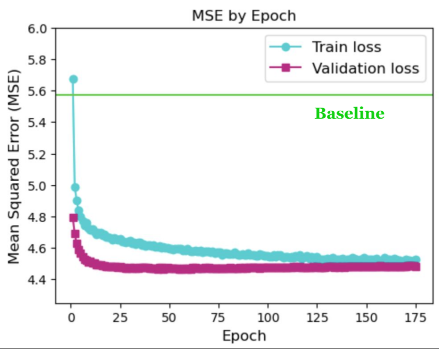
Speaker: Katherine

- Note: what I cover here will not be exhaustive; these are illustrative charts to give a high-level overview of how I arrived at my chosen hyperparameters
- Also note that this process was very iterative; I'm depicting an order here, but I didn't push and pull in a linear way during the actual analysis
- Some of these (activation functions, optimizer) were unsurprising
- Model consistently underfit, leading to low dropout and no L1 regularization
- Several things here (learning rate schedule, lag length) didn't help much

We found that a lower learning rate, higher epoch count, and the presence of some hidden layers improved our FFNN

Learning Rate	1.00E-2	1.00E-5
Hidden Layer Size(s)	[ ]	[150, 75]
Epoch Count	20	175
Test MSE	4.14	3.98





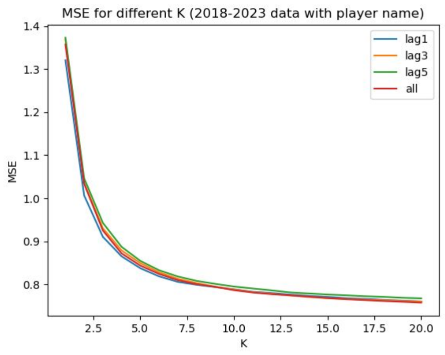
Speaker: Katherine

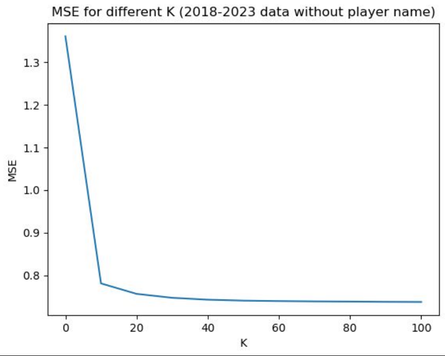
- Note: the underfitting we saw never went away
- BUT adding hidden layers, lowering learning rate, and adding epochs all helped
- Even our worst model beat the baseline

KNN

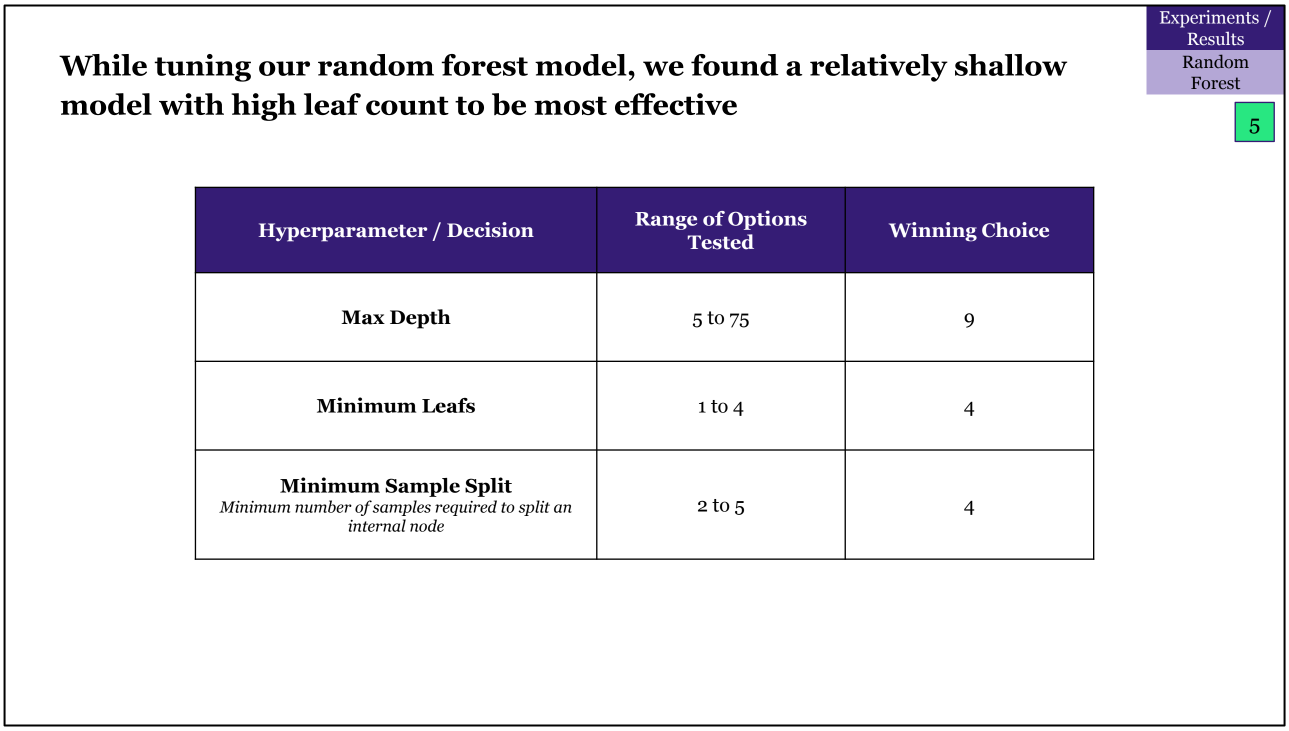
Model experimentation:

- No true validation set
    - Utilize cross validation instead
- Datasets with and without player name
- Datasets using different lag variables
- Different K
- Different train/ test sets





Speaker: Anna



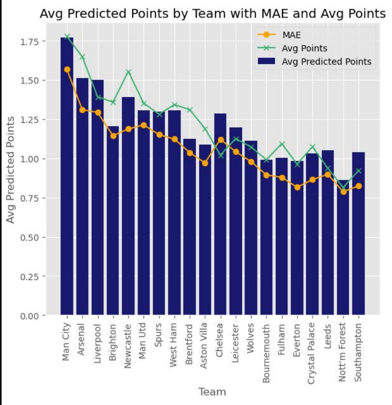
While tuning our random forest model, we found a relatively shallow model with high leaf count to be most effective

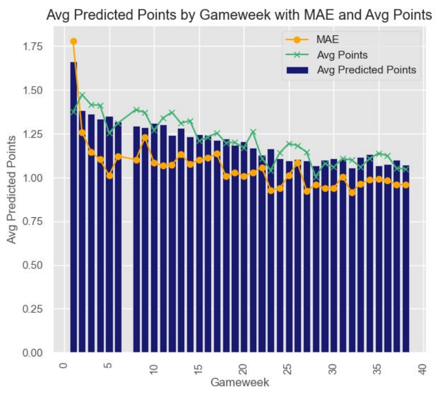
Hyperparameter / Decision	Range of Options Tested	Winning Choice
Max Depth	5 to 75	9
Minimum Leafs	1 to 4	4
Minimum Sample Split *Minimum number of samples required to split an internal node*	2 to 5	4

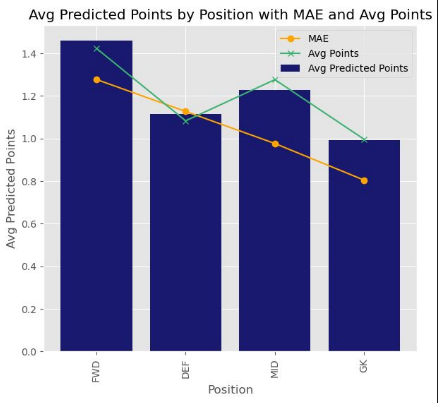
Speaker: Iishaan

- Suggests that lower dimensional models significantly outperformed high dimensional models, but higher number of leafs performed better for Random Forests than Decision Trees
-

Our model does account for position and team differences in predictions but shows higher MAE for higher-scoring teams and positions







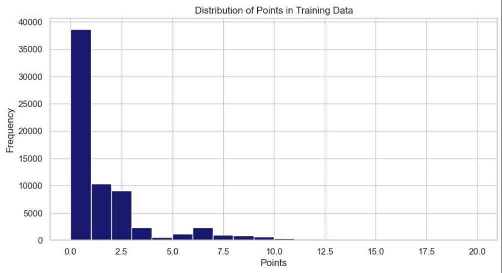
Speaker: Iishaan

- Predicted points by team, position, and gameweek with MAE and MSE

Conclusions

Key results:

- All of our models outperformed our baseline, but our random forest model had the lowest MSE and MAE (3.85 and 1.05 respectively)
  - Reducing the MAE of predictions by .5, given the tight distribution of expected points (shown right), has a ***significant effect*** on our ability to predict overall player performance



What have we learned?

- Our winning model performs better for average players than outlier players (e.g. highest performing players, as top performers are one-tailed)

How might we build on this work in the future?

- Build on our model to create a model that fully manages your FPL team, within dedicated FPL budgets
  - Could predict cost efficiency (points / $) rather than just total points
- Alternative twist: Instead of predicting points for all players, pick top 50 players from the previous seasons and try to model each player individually

Speaker: Iishaan

Thank you for listening!

Any questions?

Speaker:

Team Contributions

Github Repo: *https://github.com/annali-2/datasci207_final_AHIK/tree/main*

Whole Group:

- Data processing
- Slides
- Baseline and linear regression models
- Providing feedback and suggestions to teammates

Individual Contributions:

- KNN: Anna
- RNN/LSTM: Hisham
- Decision Tree, Random Forest, Gradient Boosting (includes XGBoost): Iishaan
- FFNN: Katherine

Sources

- Fantasy Premier League, Official Fantasy Football Game of the Premier League. (n.d.). Premier League. https://fantasy.premierleague.com/prizes
- Šuľan, M. (2023, July 27). How many people play FPL? (2023/24). FPL Reports. https://www.fantasyfootballreports.com/how-many-people-play-fpl/
- (FPL image) NCFSC Fantasy Premier League Competition. (n.d.). Norwich City Fans Social Club. Retrieved December 12, 2023, from https://www.ncfsc.co.uk/fantasy-premier-league-competition/
- (Project data) Anand, V. (2022). Fantasy-Premier-League. GitHub. https://github.com/vaastav/Fantasy-Premier-League/tree/master
- Fantasy Premier League - Dataset. (n.d.). Www.kaggle.com. Retrieved December 12, 2023, from https://www.kaggle.com/datasets/ritviyer/fantasy-premier-league-dataset/data
- Placidi, J. (2023, April 27). Predicting Fantasy Football Teams using Regression with Regularisation. GitHub. https://github.com/JoshuaPlacidi/Fantasy-Football-Team-Predictions/tree/master
- solpaul. (2023, October 27). FPL Prediction. GitHub. https://github.com/solpaul/fpl-prediction

Appendix

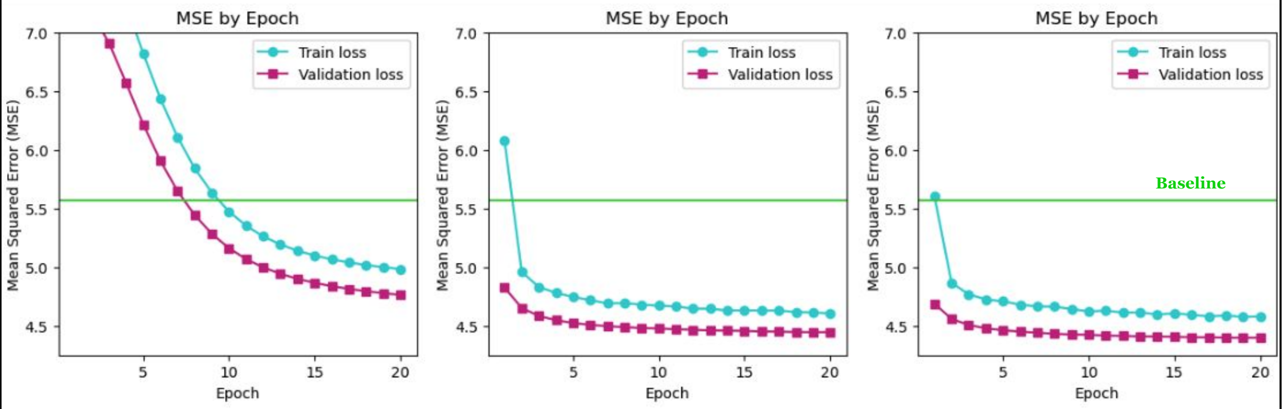
Experiments / Results

FFNN

3

The presence of hidden layers reduced loss, but no amount of model complexity resolved the underfitting

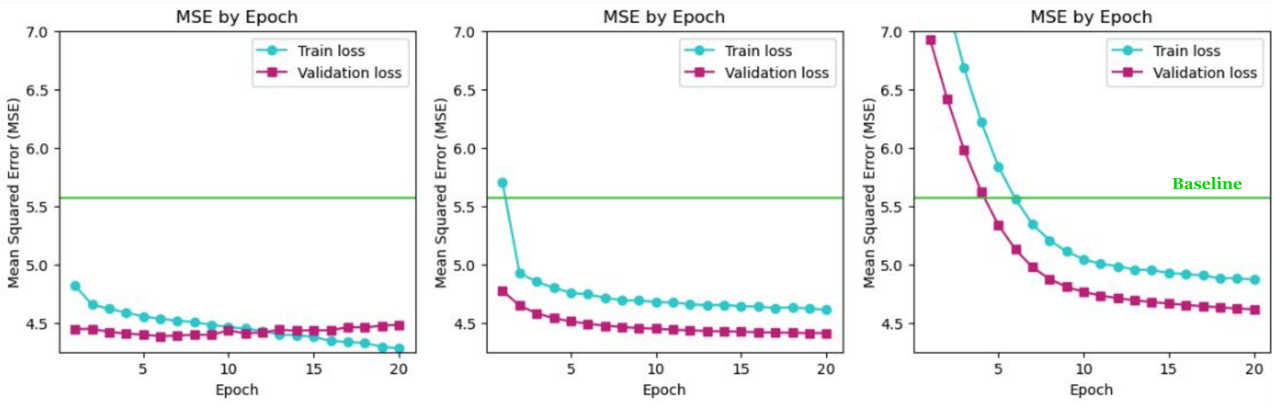
Hidden Layer Size(s)	None	[150, 75]	[300, 150, 75]
Final Train / Val Loss	4.98 / 4.76	4.60 / 4.45	4.58 / 4.40



Speaker: Katherine

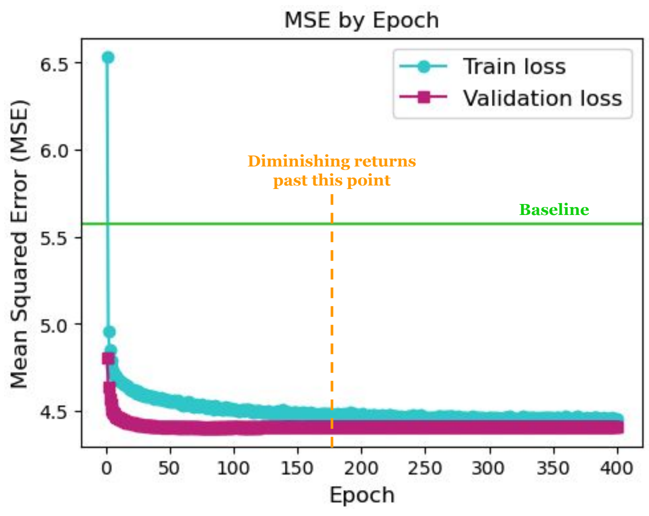
A learning rate of 1.0E-5 generates an ideal learning curve

Initial Learning Rate	1.0E-4	1.0E-5	1.0E-6
Final Train / Val Loss	4.32 / 4.54	4.62 / 4.45	4.89 / 4.65



Speaker: Katherine

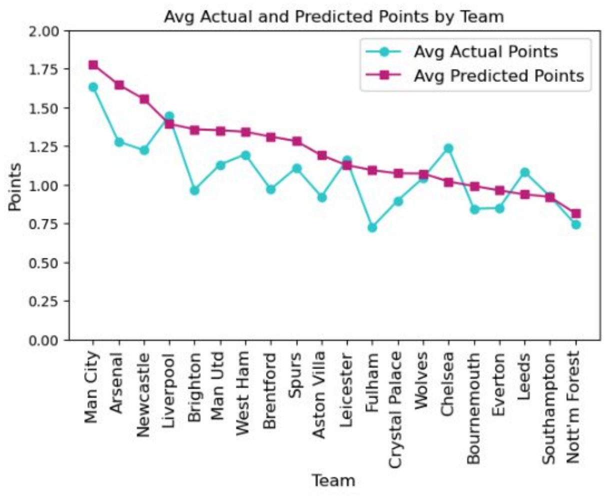
We cut off our training at 175 epochs due to diminishing returns from further training

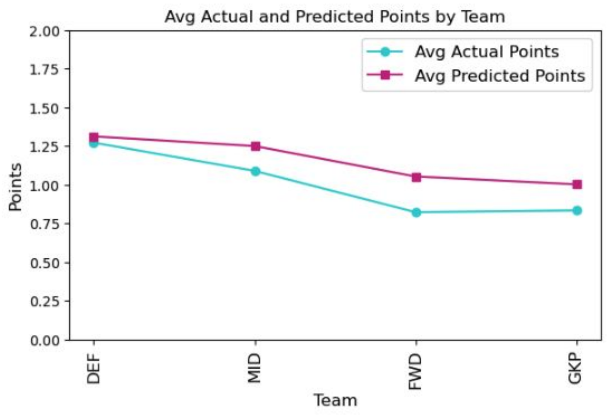


Epoch Count	Final Train / Val Loss
175	4.45 / 4.42
400	4.46 / 4.46

Speaker: Katherine

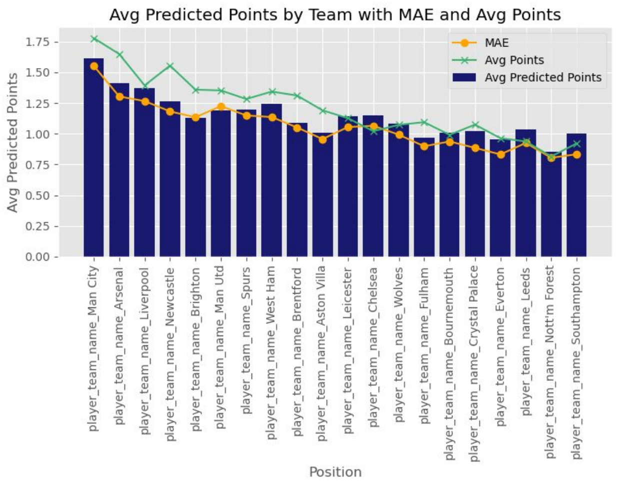
Our FFNN does take into account team and position differences, though imperfectly

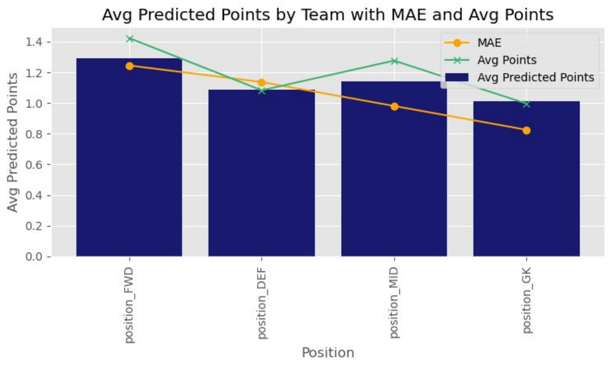




Speaker: Katherine

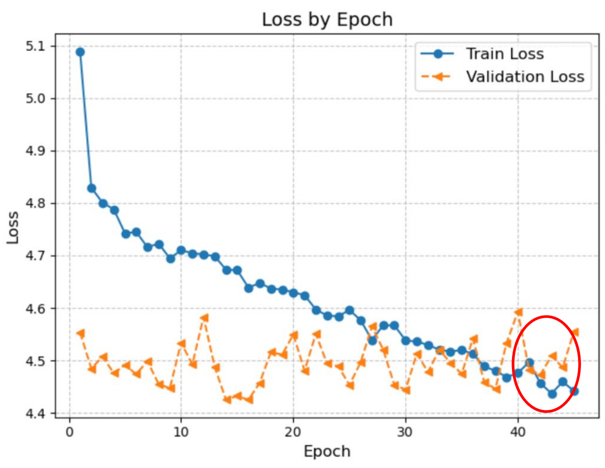
Our KNN also takes into account team and position differences, though imperfectly

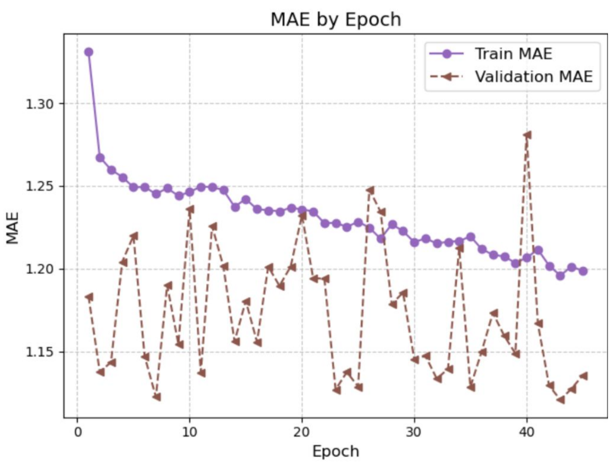




Speaker: Anna

We cut off our training at 45 epochs due to overfitting beyond this point





The model starts to overfit around epoch 40.

Absolute Error continues to improve with every epoch

Speaker: Hisham

Gradient Boosted Regression

Hyperparameter / Decision	Range of Options Tested	Winning Choice (Gradient Boosted)	Winning Choice (XGBoost)
Number of estimators	5 to 150	103	40
Max Depth	5 to 150	7	7
Minimum Samples Leaf	1 to 4	1	N/A
Minimum Samples Split	2 to 5	5	N/A
Learning Rate *Named as eta in the XGB model*	.001 to .1	N/A	.06
Maximum Leaves	0 to 4	N/A	0
Gamma *Minimum loss reduction required to make a further partition on a leaf node of the tree*	0 to 5	N/A	2

Speaker: Iishaan