Report

1 Data Analysis

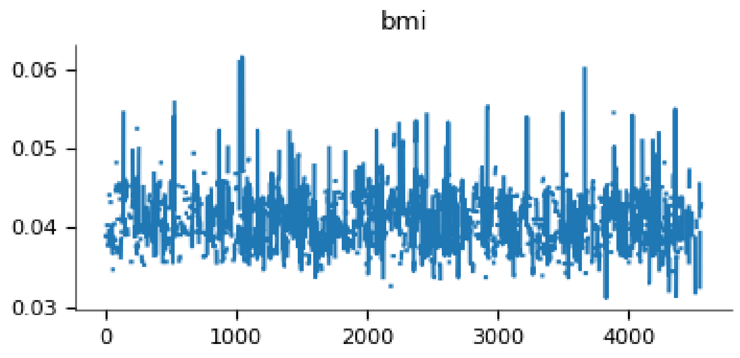
1.1 Exploratory Analysis and Visualizations

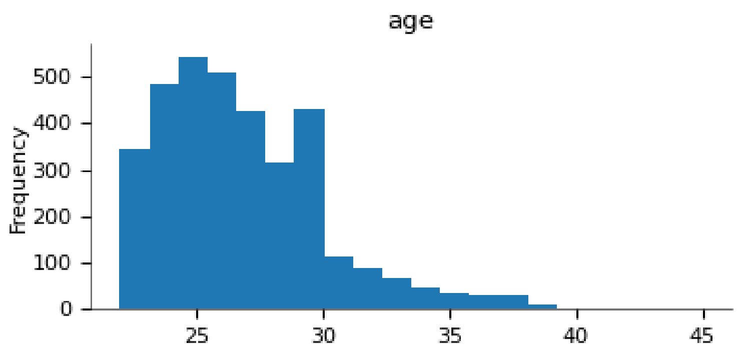
This dataset comprises individual and team information of NFL players. This information is ultimately used to calculate Projected fantasy points in PPR (Points Per Reception) format. There are some important points within this data that have prompted us to address some fundamental questions:

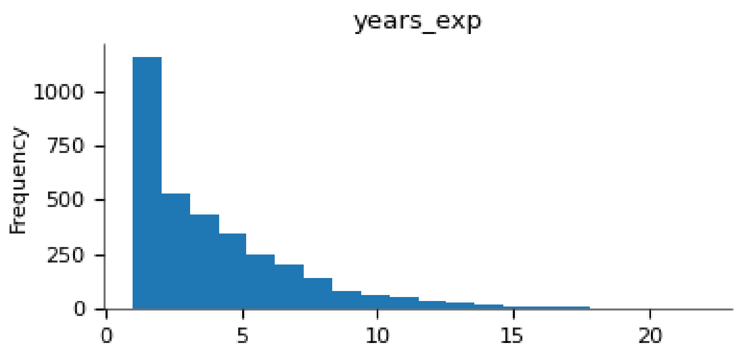
1. Is all of this data necessary for calculating and limiting Projected fantasy points in PPR?
2. Should players without certain characteristics be removed?
3. What are the most critical features for calculating these points?
4. What steps should be taken to predict Projected fantasy points in PPR for the next season?
5. Does the team information or personal information of the players have an impact?

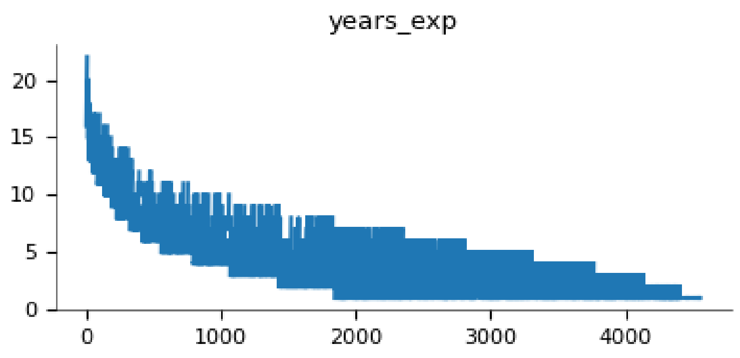
So, we began analyzing the data and seeking answers to these questions. Initially, we attempted to utilize tools to determine whether they could identify any meaningful relationships within our data or not

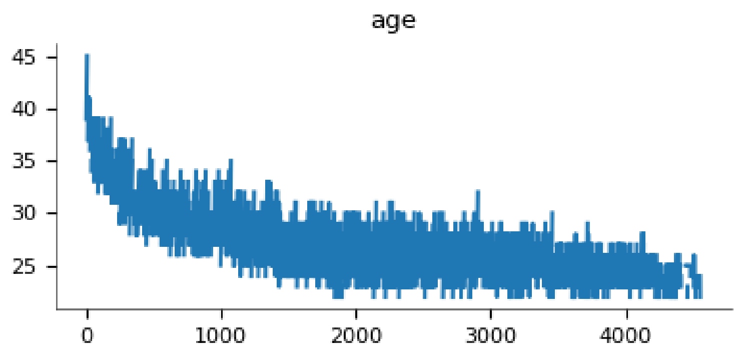
Here are some Analytical chart

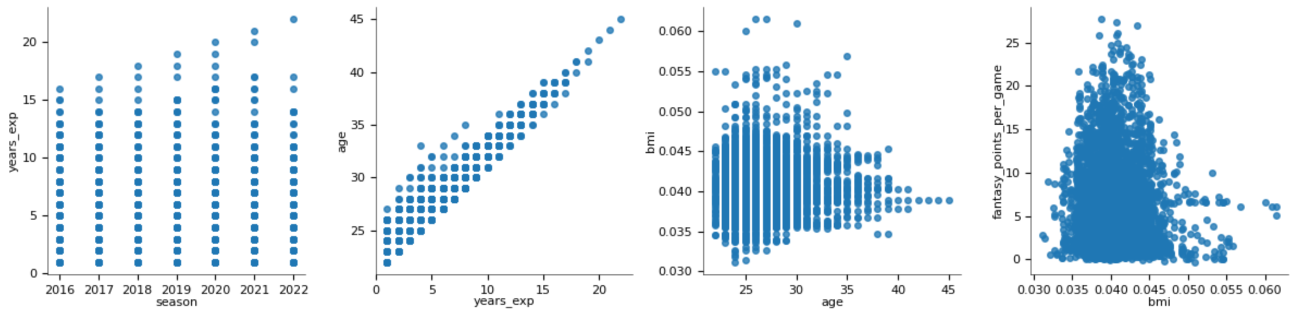












We soon realized that charts did not yield much useful information, which was somewhat expected. This was because the graphs either showed a direct correlation between variables, an inverse relationship, or, at times, no discernible connection at all. For instance, age and experience tended to increase together, as did the number of successful receptions and the fantasy_points_ppr score. Conversely, there were instances of the opposite trend, such as the increase in age leading to a decrease in the number of players of the same age in the league. In fact, these data didn't prove to be helpful for our analysis. Consequently, we made the decision to proceed methodically and conduct our analysis step by step.

Our initial analysis focused on determining which data is necessary for predicting Projected fantasy points in PPR. We recognized a significant challenge when attempting to predict a player's score for an upcoming season, such as the 2024 season. Since the season has not yet started, we lack information on the player's performance during that season. Therefore, we need to rely on personal data such as BMI, age, experience, and other relevant factors.

To elaborate, we initially explored whether there was any noticeable correlation between these personal data and their PPR fantasy points. However, we found limited or no significant correlation. As a result, we decided to utilize data from the previous season to make predictions. For instance, when predicting a player's performance in the 2024 season, we incorporate information from the 2023 season and the player's personal characteristics as features. The objective is to predict the player's points for the 2024 season.

We apply the same approach to our training data. For example, we consider the points earned in the 2016 season as sample features, and the final score for the 2017 season serves as the label. This method provides us with a dataset containing 5 to 6 seasons' worth of information for each player.

Finally, we came to the conclusion that the three main actions and the player's performance in each game plus general indormation can be good data for building the model. So we went to the formulas for calculating Rushing, Receiving, and Passing points.

1.2 Data Quality

A series of data does not help us due to being an identifier, so we delete them at the beginning.

```
columns_to_drop = ['name','team','draft_number',]
data.drop(columns=columns_to_drop, inplace=True)
```

But we keep the player_id for further processing.

Then we go to age, bmi exp. Age and exp of some players were blank, but we obtained this information according to the information of previous years.

In this code, it carefully adds one to the player's age from the previous year, if any.

```
j=0
for i in range(len(data)):
 if(math.isnan(data.iloc[i,3])):
        if(data.iloc[i-1,0] ==data.iloc[i,0]   ):
           data.iloc[i,3]=data.iloc[i-1,3]+1
```

We did the same process with exp. But we transferred exactly the same bmi of the previous year to the new year.

In this section, we made sure that we have the complete information of the players. So, with one command, we removed the players who did not have complete general information.

2 Feature Engineering

2.1 Feature Creation

We derived three new features which are as follows.

```
data['passing_fantasy_points'] = (data['passing_tds'] * 4) +
(data['passing_yards'] * 0.04) - (data['interceptions'] * 2)
data['rushing_fantasy_points'] = (data['rushing_tds'] * 6) +
(data['rushing_yards'] * 0.1)-(data['rushing_fumbles'] * 2)
data['receiving_fantasy_points'] = (data['receiving_tds'] * 6) +
(data['receiving_yards'] * 0.1) + (data['receptions'] * 1) -
(data['receiving_fumbles'] * 2)
```

These three features calculate the overall score of each player for the three main actions of passing, rushing, and receiving during a season.

2.2 Feature Selection

Finally, the following features were selected for the model.

```
per_game_columns = ['completions', 'attempts', 'passing_yards',
'passing_tds', 'interceptions',
'passing_air_yards','passing_yards_after_catch'     ,'rushing_yards',
'rushing_tds',
                     'targets', 'receptions', 'receiving_yards',
'receiving_tds', 'receiving_air_yards' ,'receiving_yards_after_catch' ]
```

```
features = ['years_exp', 'age', 'bmi',
            'passing_fantasy_points', 'rushing_fantasy_points',
'receiving_fantasy_points'] + [f'{col}_per_game' for col in per_game_columns]
```

3 Model build

3.1 Model selections

Why we use randomforrest regressor ? here is the answer

Random Forest Regressor is a machine learning algorithm that can be a good choice for various regression tasks, including predicting Projected fantasy points in PPR for NFL players. Here are several reasons why you might want to use a RandomForestRegressor:

1. Ensemble Learning: Random Forest is an ensemble learning technique that combines the predictions of multiple decision trees. This ensemble approach often results in more robust and accurate predictions compared to a single decision tree.

2. Non-Linearity: Random Forests can capture complex non-linear relationships between input features (e.g., player characteristics) and the output (fantasy points). This flexibility makes it suitable for modeling situations where the relationship is not straightforward or linear.
3. Handling of Large Feature Sets: If you have a dataset with a large number of features (e.g., player statistics, personal data), Random Forests can handle this effectively. They automatically select a subset of features for each tree, which helps mitigate the curse of dimensionality and overfitting.
4. Robustness to Outliers: Random Forests are less sensitive to outliers and noisy data compared to some other regression techniques. They can handle noisy data without significantly affecting the overall performance.
5. Feature Importance: Random Forests provide a measure of feature importance. This can help you identify which player characteristics have the most significant impact on predicting fantasy points. This information can be valuable for feature selection and model interpretation.
6. Reduced Overfitting: By aggregating predictions from multiple trees and introducing randomness during tree construction, Random Forests tend to have lower overfitting compared to a single decision tree. This makes them more generalizable to unseen data.
7. Easy to Use: Random Forests are relatively easy to use and require minimal hyperparameter tuning compared to some other complex algorithms. This makes them a good choice for practitioners looking for a robust solution without spending excessive time on fine-tuning.
8. Parallelization: Training individual decision trees in a Random Forest can be done in parallel, making it suitable for scaling on multi-core processors and distributed computing environments.

3.2 Model training

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
predictions = model.predict(X_test)
```

and there are explantion about this model

1. Model Initialization:
   - model = RandomForestRegressor(n_estimators=100, random_state=42): This line of code initializes a Random Forest Regressor model with certain hyperparameters.
   - n_estimators=100: This parameter specifies the number of decision trees (estimators) that will be used in the ensemble. In this case, 100 trees will be used.
   - random_state=42: The random_state parameter sets the random seed for reproducibility. Using the same seed ensures that the results will be the same each time you run the code. It's useful for debugging and reproducing results.
2. Training the Model:
   - model.fit(X_train, y_train): This line of code trains the Random Forest Regressor model on the training data.
   - X_train: This is the feature matrix containing the training data.
   - y_train: This is the target variable or label corresponding to the training data.

3. Making Predictions:

- predictions = model.predict(X_test): After training, the model is used to make predictions on the test dataset (X_test), which contains data that the model has not seen during training.

In summary, the code snippet demonstrates the process of training a Random Forest Regressor model, using it to make predictions on unseen data, and then evaluating the model's performance using the Mean Absolute Error metric. The choice of 100 estimators and a random state of 42 are hyperparameters that can be adjusted based on your specific dataset and requirements.

4 Model performance

4.1 Evaluation

We use mae for evaluation

```
# Evaluate the model
mae = mean_absolute_error(y_test, predictions)
print(f'Mean Absolute Error: {mae}')
```

Model Evaluation:

- mae = mean_absolute_error(y_test, predictions): This line calculates the Mean Absolute Error (MAE) between the true target values (y_test) and the predicted values (predictions). MAE is a common metric used to evaluate the performance of regression models.
- print(f'Mean Absolute Error: {mae}'): Finally, the code prints the MAE, providing an indication of how well the model's predictions match the actual target values. Lower MAE values indicate better predictive performance.

4.2 Interpretability

We get this at last

```
Mean Absolute Error: 4.317499831365936
```

Mean Absolute Error (MAE) is a measure of the average absolute errors between the actual target values (y_test) and the predicted values (predictions) generated by your model. In this case, a MAE of approximately 4.32 indicates that, on average, your model's predictions for fantasy points in PPR (Points Per Reception) for NFL players are off by roughly 4.32 points when compared to the actual values.

A lower MAE is generally desirable, as it suggests that your model is making more accurate predictions. However, the significance of the MAE depends on the specific context of your problem and the range of possible values for the target variable.

In many cases, achieving a MAE of approximately 4.32 would be considered a good result, especially if it aligns with the level of precision required for your application. It's important to consider the domain and practical implications of the error when interpreting MAE. If this level of error meets your requirements, your model appears to be performing well in predicting fantasy points in PPR for NFL players.

4.3 Reproducibility

It is enough to give this model the overall performance of a player's season plus the PPR fantasy points of the same season to predict the points of the next season.