Implementing AI: Analyzing NBA Data to Uncover Insights

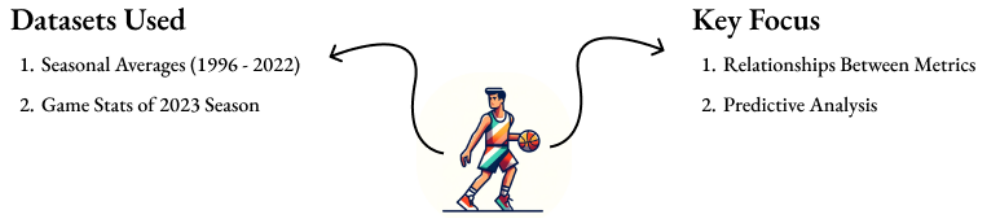
Understand Data Science Techniques Through NBA Case Studies

Pragnesh Anekal

April 18, 2024

Introduction

This concise guide explores the application of AI to NBA statistics, providing insights and predictive analytics centered around player performance metrics. Through a detailed analysis of two primary datasets, this book aims to not only enhance understanding of the game but also to equip readers with the skills to predict future outcomes based on statistical analysis.



- **Datasets Used:**

    1. **Seasonal Averages (1996 - 2022):** Analyze 26 years of NBA data, focusing on player performance across various seasons.
    2. **Game Stats of 2023 Season:** Deliver game-by-game statistical analysis for the 2023 NBA season.

- **Key Analytical Focus:**

    1. **Relationships Between Metrics:** Examine how on-court statistics such as points, rebounds, and others relate to each other for individual players.
    2. **Predictive Analysis:** Implement Bayesian techniques to forecast a player's future performance metrics, such as scoring averages.

This book is designed for readers interested in the intersection of sports analytics and data science, offering a theoretical foundation to the concepts of data science and statistical learning.

Understanding Relationships Through Linear Regression

Before diving into the specifics, I want to emphasize that throughout this and future sections, I will be deviating from the problem to explain some concepts which I found difficult to understand to best attempt to clear the air for this specific problem. To start things off, let's dive into linear regression.

What is Linear Regression?

It is a statistical method that establishes the relationship between a dependent variable (response), and one or more independent variables (predictors). It aims to predict the value of the response variable by finding the best linear combination of the predictors. It is given by the equation,

$$y = f(x) + \epsilon \tag{1}$$

where ϵ is the error associated with the prediction. In essence, we are constructing a line that best fits the data points, helping us make informed predictions or understand relationships within our data.

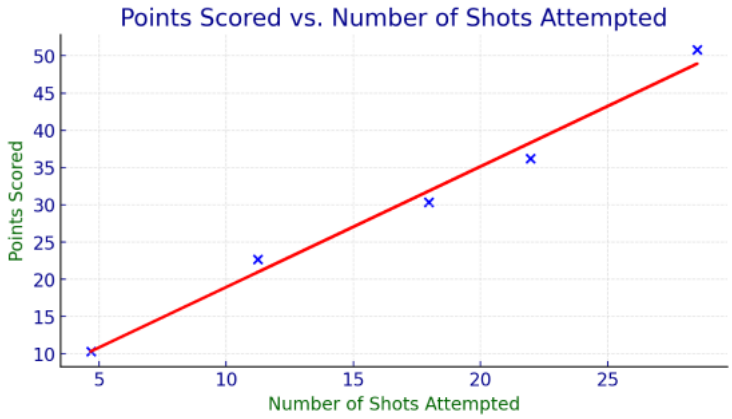


Figure 1: Linear Regression

Let's use the above example to understand linear regression more clearly:

- **Objective:** Predict the points scored (**response variable**) by a player based on the number of shots they attempt (**predictor variable**).
- **Method:** Establish a linear relationship between these variables. It involves finding a regression line that best fits our data. The **red line** in the graph represents this regression line.

Steps Taken

To explore the relationships between points scored and other on-court factors, below are the steps taken:

1. **Feature Selection and Distribution Analysis**
2. **Model Development**
3. **Making Conclusions with Coefficients**

Feature Selection and Distribution Analysis

Feature selection is crucial in isolating the most influential factors affecting points scored, which improves model accuracy and simplifies the interpretation of results. We plot distributions to visualize the behavior of variables, helping us identify patterns or outliers and understand the data's range and central tendencies.

Note: We have removed non-relevant factors like age, draft number, and other off-court stats. While it's true that high draft picks or players from universities like Duke might score more, our focus here is strictly on what a player can influence directly on the court. We're essentially asking, **"Can a player control this during a game?"** For instance, a player doesn't choose their draft position or college, so we've excluded such factors from our analysis.

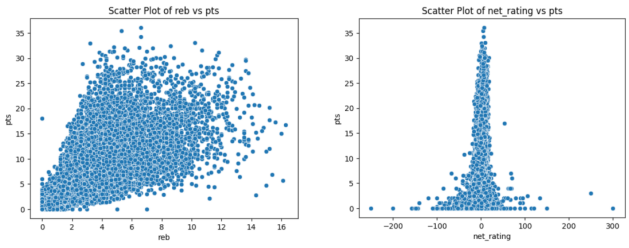


Figure 2: Scatter Plots

Scatter Plots: Represent the correlations between different basketball statistics and points scored, highlighting the strength and nature of their relationships. The plot on the left-hand side indicates a positive correlation where points tend to increase as rebounds increase. The plot on the right illustrates

points tightly clustered around the central axis, with fewer outliers, indicating a less clear relationship with net rating.

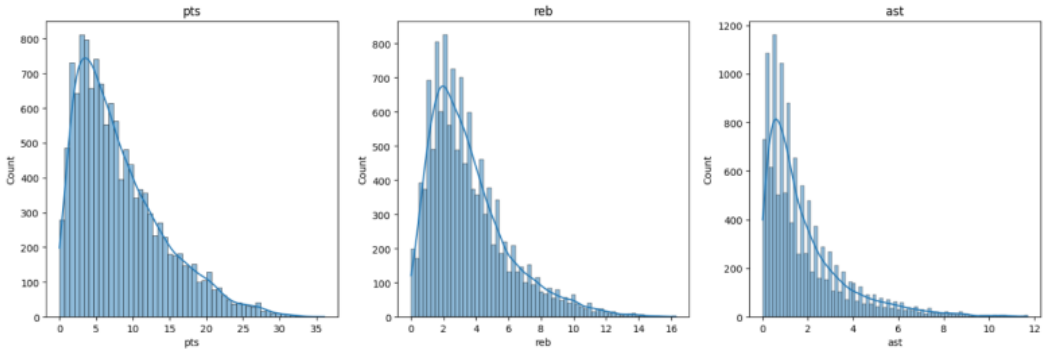


Figure 3: Histogram

Histogram: Represent the distribution of a dataset by showing the frequency of observations within different ranges of values. In the figure, the distribution for points scored (pts), rebounds (reb) and assists (ast) appears right-skewed, indicating a higher frequency of lower values. This is expected since it's typical to see that most players average lower values in statistical categories, while only a few star players reach higher averages.

For a complete visual of all distributions, refer to the Jupyter notebook associated with this textbook, which can be found on my GitHub.

Encoding Categorical Columns

Although most columns in our dataset are numerical, categorical columns also exist and require encoding to convert them into numerical formats for machine learning algorithm compatibility. This encoding process can inadvertently introduce multicollinearity.

What is multicollinearity?: A statistical phenomenon in which two or more predictor variables in a regression model are highly correlated, meaning that one can be linearly predicted from the others with a substantial degree of accuracy. This can create problems in the regression model, as it becomes difficult to determine the individual effect of each predictor variable on the outcome.

Method Used: Dummy Encoding

Dummy encoding converts categorical variables into a binary matrix representation suitable for machine learning algorithms, deliberately dropping one category to avoid multicollinearity. This ensures model coefficients remain interpretable. Below is an example showing how player names are encoded both before and after applying dummy encoding:

Before:

Player
Stephen Curry
LeBron James
Kobe Bryant

After:

Stephen Curry	**LeBron James**	**Kobe Bryant (Dropped)**
1	0	*0*
0	1	*0*
0	0	*0*

Note: The column for Kobe Bryant is dropped to prevent multicollinearity, as his presence is implied when both Stephen Curry and LeBron James columns are zero.

Model Development

The model development follows these steps to ensure reliable predictions:

1. Split dataset into training and holdout test sets to differentiate between model training and validation phases.

2. Conduct Ridge regression on the training data, standardizing features and utilizing `RidgeCV` to find the optimal regularization parameter (alpha) through cross-validation.

3. Apply `cross_val_score` for comprehensive cross-validation on the full training dataset to evaluate the model with the selected alpha.

4. Evaluate the model's performance on the holdout test set for an objective assessment of its predictive power.

5. Finally, for deployment or additional predictions, retrain the model on the entire dataset with the previously determined optimal alpha.

To broaden our understanding of model efficacy, we also incorporate AutoML training using H2O's AutoML to benchmark nonlinear models against our linear model.

Splitting the Dataset

Splitting the dataset involves dividing the data into separate sets to train and evaluate the machine learning model.

```
# Split data into training and holdout set
X = nba_df.drop(columns='pts')
y = nba_df['pts']

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

Listing 1: Python code for splitting the dataset

The code snippet above separates the dataset into training and testing sets, with 20% of the data reserved as a holdout set for model evaluation, ensuring the same split every time with a fixed random state.

Optimal Alpha for Ridge Regression via 5-Fold Cross Validation

To proceed with this section, we must first grasp two fundamental concepts: **cross-validation** and **regularization**.

Cross-validation: To assess how the results of a statistical analysis (in our case linear regression) will generalize to an unseen data set, especially useful in situations where the goal is prediction. It involves partitioning a sample of data into complementary subsets, systematically analyzing multiple 'folds' of data to validate the model on one subset while training the model on the other subsets.

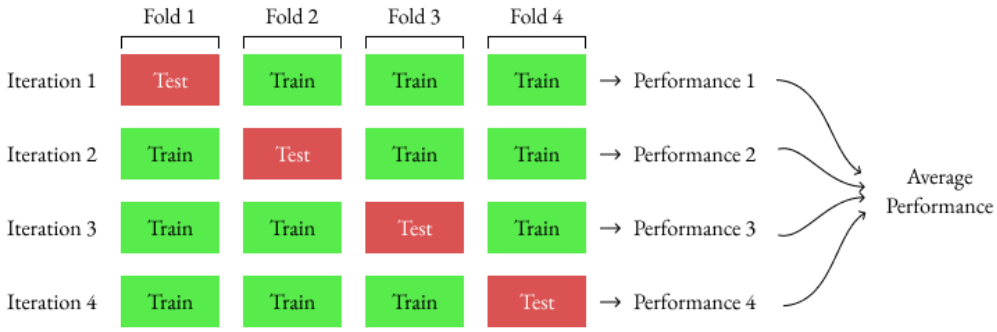


Figure 4: Cross Validation

Regularization: Used to prevent overfitting by penalizing large coefficients in a regression model. It introduces a penalty term to the loss function, constraining or regularizing the coefficient estimates towards zero.

Ridge regression, a type of regularized linear regression, includes an L2 penalty term which constrains the coefficients. This is represented mathematically as:

$$\text{Ridge Regression: } \min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - X_i\beta)^2 + \alpha \sum_{j=1}^{p} \beta_j^2 \right\} \tag{2}$$

Here, β represents the vector of coefficients, α is the regularization parameter that multiplies the penalty term, and p is the number of predictors. By choosing an optimal value for α, Ridge regression works to minimize the residual sum of squares (RSS) subject to a penalty on the size of coefficient estimates.

From Where Did This Equation Turn Up?

You may question the transition from equation 1 presented earlier to this current equation and its connection to linear regression. Let's revisit the example from Figure 1 to demystify this equation's function.
Here's what each part means in plain language:

- The sum $\sum_{i=1}^{5}(\text{points\_scored}_i - \text{shots\_attempted}_i \times \beta)^2$: We're adding up the squares of the differences (errors) between the actual points scored and the points predicted by our model for each of the five data points. We square these differences to penalize larger errors more severely and ensure all errors are positive.
- β: This is what we're solving for - it's the effect of scoring one more point in a game. In simple linear regression, we'd only minimize the errors, but...
- $\alpha \times \beta^2$: ...Ridge regression adds this extra piece to the puzzle. It's a penalty that gets bigger the larger our coefficient β is. The α is a number we choose in advance, deciding how harshly we penalize bigger coefficients. If α is zero, we're back to normal linear regression.
- The min implies that we want what is enclosed in the {}, which is the residuals to be as low as possible.

```
1 # Create a pipeline that standardizes, then runs RidgeCV
2 ridge_pipeline = Pipeline([
3     ('ridge_scaler', StandardScaler()),
4     ('ridge_cv', RidgeCV(cv=5))
5 ])
6
7 # Fit the model to the training data
8 ridge_pipeline.fit(X_train, y_train)
```

Listing 2: Python code for performing regularization

In the code:

- Line 3: Initializes a scaler that will standardize the features by removing the mean and scaling to unit variance.
- Line 4: Sets up RidgeCV, a Ridge regression model with built-in cross-validation to determine the optimal regularization parameter.

- Line 8: Fits the pipeline to the training data, which applies standardization followed by the RidgeCV model to train.

Use Cross Validation to Evaluate the Model

Cross-validation is utilized to evaluate the model's performance on the entire training set to ensure the robustness of the model.

```
# Step 3: Evaluate the Model with Cross-Validation
ridge_cv_scores = cross_val_score(ridge_pipeline, X_train,
    y_train, cv=5)
# Print R2 value for each fold from training set
for score in ridge_cv_scores:
  print(score)
```

Listing 3: Python code to evaluate the Ridge regression model with cross-validation

The output for each fold's R^2 value is as follows:

```
0.9124625863336421
0.8989462112191654
0.9120816756343475
0.9013414186268957
0.899532468665649
```

This output represents the R^2 value for each of the five folds in the cross-validation process, indicating consistent performance across different subsets of the training data.

What is R^2?

R^2, or the coefficient of determination, measures the proportion of variance in the dependent variable that is predictable from the independent variables. It is defined by:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and $\bar{y}$ is the mean of the actual values.

Assess Model on Holdout Set

This stage evaluates the trained model using the holdout set to estimate its performance on new, unseen data.

```python
# Fit model on training data using chosen alpha value and test
    it on holdout set
ridge_final_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('ridge', Ridge(alpha=ridgecv_model.alpha_))
])

ridge_final_pipeline.fit(X_train, y_train)

# Step 4: Evaluate on Holdout Test Set
ridge_test_score = ridge_final_pipeline.score(X_test, y_test)

# R2 value for test set
print(ridge_test_score)
```

Listing 4: Python code to fit and test the Ridge regression model on the training and holdout sets

The code snippet constructs a pipeline comprising a standard scaler and a Ridge regression model with an optimally tuned alpha parameter obtained from previous cross-validation steps. It then fits this model to the training data. After fitting, the model is evaluated on the holdout set by calculating the R^2 score, which quantifies how well the model predicts unseen data.

Train Model and Analyze Coeffecients

Outlines the process of training the Ridge regression model and analyzing the importance and influence of each feature within the model.

```python
ridge_final_pipeline.fit(X, y)

# Extract final model from pipeline
ridge_model = ridge_final_pipeline.named_steps['ridge']

# Get the coefficients from the model
coefficients = ridge_model.coef_

# Assuming 'feature_names' matches the columns used in X_train
feature_names = X.columns

# Combine the feature names and their corresponding
    coefficients into a dictionary
feature_coefficients = dict(zip(feature_names, coefficients))

# Now, if you want to display them, for example, sorted by the
    absolute value of the coefficient:
sorted_feature_coefficients = sorted(feature_coefficients.items
    (), key=lambda item: abs(item[1]), reverse=True)
```

```
17
18 # Print out the feature names along with their coefficients
19 print("Feature coefficients from Ridge Regression:")
20 for feature, coeff in sorted_feature_coefficients[:8]:
21     print(f"{feature}: {coeff}")
```

Listing 5: Python code to fit Ridge regression model and analyze coefficients

The code above fits the Ridge regression model to the data and extracts the model's coefficients. These coefficients are then associated with their respective features and sorted by their absolute values to highlight the most influential factors. Here are the top coefficients as output:

```
Feature coefficients from Ridge Regression:
reb: 3.1079144172468096
ast: 2.6948509357707513
usg_pct: 1.964154688535398
ast_pct: -0.993785525266237
dreb_pct: -0.9210067359761196
oreb_pct: -0.6429650266327452
ts_pct: 0.49102150644819237
gp: 0.28333912796628064
```

When we examine the coefficients produced by the Ridge regression model, we observe several intriguing relationships:

- **Rebounds (reb):** With the highest positive coefficient, this suggests a strong positive relationship with scoring averages. Players who tend to get more rebounds are likely to have a higher scoring average.

- **Assists (ast):** This statistic also has a significant positive coefficient, indicating that players who average more assists are likely to score more points.

An Intuitive Thought

As a basketball enthusiast, I find it intriguing that assisting, i.e. passing the ball to a teammate, can inherently improve a player's scoring ability, which seems counterintuitive. This suggests deeper relationships at play, as players who are ball dominant tend to average higher assists, rebounds, and points because they control the ball most of the time.
However, the coeffecients represent the change in the dependent variable (response variable) for a one-unit change in the independent variable (predictor) while keeping other variables constant. For example, for every 1 rebound a player secures, they score 3 more points provided all other stats remain constant.

- **Usage percentage (usg_pct):** A positive coefficient here implies that players who are more involved in plays (usage rate) are predicted to have higher scoring averages.

These findings indicate not only the direct impact of these stats on scoring but also hint at more complex interdependencies between a player's various roles and responsibilities on the court.

Understanding Predictions

While the primary focus of this section isn't on predictions, we still assess them to determine if linear regression is suitable for our model. This involves validating several linear regression assumptions, such as autocorrelation and normality of residuals.

- **Residuals** are the differences between the observed values and the values predicted by the model.
- **Autocorrelation** occurs when residuals are not independent of each other, meaning past values influence future values, which can distort the regression model if present.
- **Normality of residuals** implies that the residuals are normally distributed, an assumption underpinning many statistical tests involving regression models.

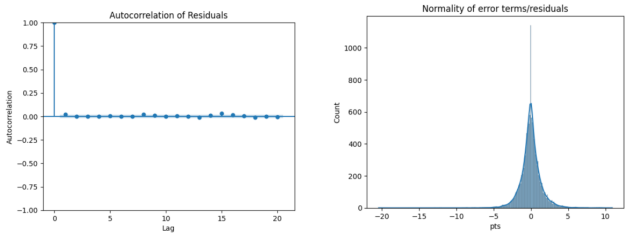


Figure 5: Autocorrelation and Normality of residuals

Quick Modeling: AutoML

AutoML automatically selects the best algorithms and tuning parameters. In this context, we use AutoML to explore how non-linear models perform and to compare their predictions against our linear model from the previous section.

```
# Set predictor and response variable for H2O dataframe
response = "pts"
predictors = list(nba_data.columns)
predictors.remove('pts')

# Initialize AutoML instance with run time of 5 minutes and 5-
    fold cross validation
aml = H2OAutoML(max_runtime_secs=300, seed=1, nfolds=5)

# Train the AutoML instance to obtain leaderboard
aml.train(x=predictors, y=response, training_frame=nba_data)
```

Listing 6: Python code to train AutoML models on NBA data

This Python code snippet sets up and trains an AutoML model to predict NBA player points using a dataset. It initializes an H2OAutoML instance with a 5-minute runtime limit and 5-fold cross-validation to efficiently find the best model.

The AutoML models achieved an R^2 score of 0.98, indicating high accuracy through 5-fold cross-validation. However, non-linear models make it challenging to interpret the predictions and the relationships between predictors and the response variable. Even though the regularized linear model scored a bit lower with an R^2 of 0.92, it's the preferred choice because our main goal is to understand the relationships between the predictors and the response variable.

Predicting Future Stats Through Bayesian Methods

In this section, we're going to take a crack at guessing how well a player will do in upcoming games, based on what they've done in the past. We'll be using Bayesian methods, which are really useful for this kind of thing because they let us combine what we already know with the latest information we have.

These methods help in dealing with all the what-ifs and maybes that come with making predictions, especially in sports where anything can happen. We're going to explore how Bayesian thinking can not only help us make better guesses about future games but also get a grip on why this approach is so handy when we're trying to figure out the future of players' stats. Bayesian methods are based on Bayes' Theorem, so let's start by understanding what that is.

What is Bayes' Theorem?

Bayes' Theorem is a mathematical formula that updates the probability of an event based on new evidence or information. Picture it this way: if you have an initial hunch about something and then something new pops up, Bayes' Theorem helps you get a new and improved hunch. The equation is as follows:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} \tag{4}$$

- H is the **hypothesis** under consideration.
- E is some new **evidence** observed.
- $P(H|E)$ is the **probability** of hypothesis H given the evidence E.

The following is a classic example for Bayes' theorem:

"Steve is very shy and withdrawn, invariably helpful but with very little interest in people or in the world of reality. A meek and tidy soul, he has a need for order and structure, and a passion for detail. Is Steve a librarian or a farmer?"

Seems likely to be a librarian based on this description. However, this is exactly what Bayes' theorem tells us not to do - New evidence does not **completely determine evidence** in a vacuum. It must **update prior beliefs**.

Let's solve this step by step:

1. Let's assume that the prior probability of being a librarian is very low because there are far fewer librarians in the population compared to farmers. (**Prior -** P(H))

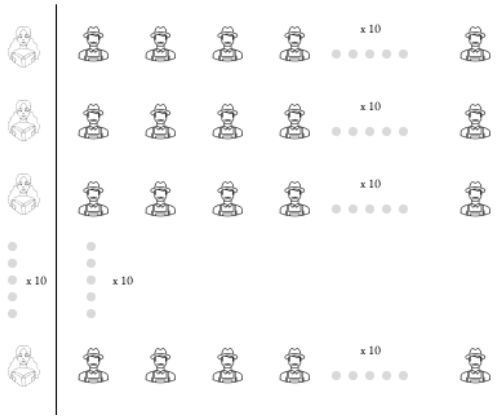


Figure 6: # of Librarians to Farmers - 0.1

2. Say, 4 librarians fit Steve's personality description out of the 10, and 20 farmers fit the description out of the 200. (**Likelihood -** $P(E|H)$)

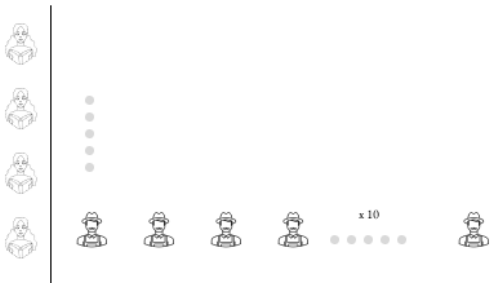


Figure 7: # of Librarians and Farmers that fit the description 4:10

3. Given the information provided, 4 librarians fit Steve's personality description out of a larger population of 24 total librarians and farmers . Therefore, the probability that Steve is a librarian given the description is $\frac{4}{24}$. (**Posterior -** $P(H|E)$)

By applying Bayes' Theorem with the given data, we've adjusted our initial assumption based on Steve's description and population ratios. This exercise demonstrates the power of Bayesian inference to shift our judgment in light of new evidence.

Bayesian Inference for Player Performance

We perform Bayesian inference through the same three fundamental steps: assuming a **prior** probability distribution, defining a **likelihood** distribution based on observed data, and using these to compute the **posterior** distribution.

```
def bayesian_inference(player_name, data):
    # Initialize a PyMC model context
    model = pm.Model()

    specific_player_column = f'Player_{player_name}'

    # Filter data for only the specific player
    player_data = nba_df[nba_df[specific_player_column] == 1]

    # Limit the analysis to the most recent 5 games
    recent_games_data = player_data.tail(5)

    with model:
        posteriors = {}  # Store posterior distributions

        # Define priors and likelihoods for each statistic
        for col in numerical_columns:
            # Prior distribution based on player's historical performance
            mu_prior = pm.Normal(f'{col}_mu', mu=player_data[col].mean(), sigma=player_data[col].std())

            # Likelihood based on observations from recent games
            observed = pm.Normal(f'{col}_observed', mu=mu_prior, sigma=player_data[col].std(), observed=recent_games_data[col])

            # Store the prior for access during sampling
            posteriors[col] = mu_prior

        # Sampling from the posterior distribution
        trace = pm.sample(2000, tune=1000)

        # Update predictions based on the posterior means
        for col in numerical_columns:
            mean_value = trace.posterior[f'{col}_mu'].mean().values
            data[col] = mean_value
```

Listing 7: Python code to train AutoML models on NBA data

This function starts by filtering data for a specific player and focuses on their most recent performances to predict future outcomes.

- **Personalized Priors:** The priors are personalized based on the player's historical performance data.
- **Likelihoods from Recent Games:** Likelihoods are specifically derived from the most recent (5) games, rather than the entire historical dataset. This recent data provides a relevant snapshot of the player's current form and condition.
- **Sampling and Updating Predictions:** The function performs sampling from the defined distributions to compute the posterior, which in turn updates the predictions for the player's future performance based on both the priors and the new evidence from recent games.

The following Python code demonstrates the prediction of points scored by Stephen Curry in a game in Miami on March 26th using a pre-trained Ridge regression model:

```
pts_scored = ridge_final_pipeline.predict(new_df.drop(columns='PTS'))
```

Listing 8: Python code to train AutoML models on NBA data

This resulted in the following output:

```
Stephen Curry @ MIA on 26th March: 25
```

The actual points scored by Stephen Curry in that game was 17 points. There is large difference in points scored and the predicted value giving a lot of room for improvement. The approach to improving our predictions is discussed in the next section.

Improving Predictions

Improving the accuracy of predictions in Bayesian models can significantly enhance decision-making processes. The approach is to refine both the prior and likelihood components of our model to yield better posterior distributions and thus more reliable predictions.

- **Refining Prior Distributions:** Currently, all priors are assumed to have normal distributions. Re-evaluating and assigning more accurate

probability distributions based on deeper historical analysis or expert knowledge can enhance the appropriateness of the priors. For instance, using a log-normal or exponential distribution for positively skewed data, such as scoring rates, may reflect the underlying realities more accurately.

- **Expanding the Likelihood Considerations:** In the existing model, only the most recent five games are considered for constructing the likelihood. To improve the model, one could expand the likelihood distribution to include:

    1. Games played against the same opponent, which might provide insights into player strategies and performance consistency against specific teams.
    2. Home versus away games data to factor in the 'home advantage' or performance variations when on the road.
    3. Critical games or playoff performances, where player behavior might change under pressure.

Each of these methods offers a pathway to refine our Bayesian inference model.

Wrapping Up

As we come to a close for analyzing the NBA data, it's clear that the insights gained extend beyond the hardwood. I believe two main takeaways exist from my learning from this project:

- **Linear Regression:** I delved into the relationships between various on-court metrics and their collective influence on player performance, leveraging linear regression to untangle these complex interactions.
- **Bayesian Techniques:** Our exploration of Bayesian inference techniques underscored the dynamic nature of prediction, balancing prior knowledge with emerging data to refine our forecasts continuously.