Final Report

Predicting NBA Player Points Per Game Using Supervised Learning

CS439 - Data Science Course Project
Authors: Ryan Donofrio [RD1036, Section 02], Shreya Vyas [sv693, Section 03]
Date: May 09, 2025

1. Introduction

1.1 Problem Statement

In the world of professional basketball, performance forecasting plays a critical role in decision-making for team managers, general managers, scouts, analysts, and fans. The central question this project seeks to answer is: Can we accurately predict an NBA player's points per game (PPG) in the next season based on their current and past performances?

This question is not only statistically rich but also has far-reaching practical value. By identifying patterns in player performance over time and quantifying them through predictive modeling, this work aims to provide reliable estimations for a player's future output. These predictions can influence lineup decisions, contract negotiations, trades, fantasy league drafting, and more.

We approach this task using supervised learning methods, particularly regression-based techniques, which are well-suited to numerical prediction problems. The target variable in our models is the player's next-season PPG, and the feature set includes statistics such as minutes played, field goal percentage, three-point percentage, assists, turnovers, and a variety of historical performance indicators.

1.2 Connection to Course Material

This project directly ties into the core learning of this course, which includes the practical application of data science methods such as: Data wrangling using pandas, visualization using matplotlib, feature engineering and transformation, regression modeling with Ridge, Lasso, and Random Forest, hyperparameter tuning via GridSearchCV, and evaluation using MAE and RMSE. By applying these techniques in a real-world context, we demonstrate our ability to implement end-to-end data science workflows. All of these methods are implemented using tools such as Pandas, Scikit-learn, Seaborn, and Matplotlib, with the analysis conducted in Jupyter Notebook format. This hands-on experience reinforces what we learned in class from lectures with real world applications in the domain of sports analytics.

2. Motivation and Background

2.1 Importance of the Problem

The ability to predict a basketball player's future scoring output holds substantial value for multiple stakeholders in the NBA ecosystem. Accurate predictions of future performance enable better decisions across the board. Team management can make more informed decisions about extending contracts, trading players, or assigning roles in a rotation. Scouts can use model insights to identify underutilized or emerging players with high potential. Fantasy basketball players rely on statistical forecasting to make strategic draft and trade decisions. Fans benefit from greater understanding and deeper engagement with team-building strategy and player development. In a league where contracts are often worth tens or hundreds of millions of dollars, being able to estimate a player's trajectory adds both competitive and financial value.

This project is motivated by the idea that individual player performance is influenced by many variables, not just scoring averages. Usage statistics like minutes played, shooting efficiency, and assist-to-turnover ratio may contribute to future scoring potential. Additionally, a longer historical context, such as a player's three-year average performance, may offer more stability than a single season's stats.

2.2 Excitement and Personal Relevance

As NBA fans and data science students, this project was especially meaningful to us. The NBA's use of analytics, including stats like Player Efficiency Rating and True Shooting Percentage, makes it a great fit for machine learning. This project brought together our passion for basketball and interest in data. While we are not trying to replace professional analysts, we wanted to see how far simple, accessible tools could take us. Our love for the game also helped shape our features by combining statistical reasoning with basketball knowledge.

2.3 Review of Prior and Related Work

Most research in basketball analytics focuses on macro-level predictions such as predicting team wins or playoff success, estimating championship probabilities, and measuring player efficiency in specific roles. Relatively fewer works focus on predicting individual player PPG across seasons, which is inherently more volatile due to factors like injury, roster changes, and playing time. Those that do exist often rely on black-box methods like neural networks without offering interpretability, which is a key criterion in our project.

Prior public models include the 538's RAPTOR model, Basketball Reference's similarity score, and fantasy basketball draft kits. The 538's RAPTOR model was designed to evaluate overall player contribution, not forecast specific stats like PPG. Basketball

Reference's similarity scores are used for comparing players but not for numeric predictions and fantasy basketball draft kits provide estimated stats, but their algorithms are proprietary and reproducible.
Our project stands out by focusing narrowly on next-season PPG, offering transparency in method, and evaluating multiple models with clear performance metrics. We take a hands-on approach using open data, reproducible code, and standard supervised learning tools. That being said, we are aware of academic papers that explore multivariate regression in sports, including works that examine the impact of physical attributes, team context, or even sentiment from social media. However, few combine historical stat lines with cross-validated regression models and compare interpretable versus ensemble-based methods side by side.

2.4 The Gap We Aim to Fill

This project fills several gaps in that it tests whether interpretable models can be accurate enough for PPG forecasting, it quantifies the importance of historical versus recent performance using feature importance analysis, it explores the tradeoff between simplicity (linear models) and complexity (ensemble methods), and it applies these techniques to real NBA data, not simulations or synthetic datasets.
By building a clean, interpretable, and extensible model pipeline, we aim to set the groundwork for further work in player-specific stat forecasting. It's something that could be of value to not just professional teams, but also casual fans, fantasy sports enthusiasts, and future student researchers.

3. Motivation and Background

3.1 Data Collections and Sources

The datasets used for this project were compiled from two primary seasons: 2021-2022 NBA Player Statistics and 2022-2023 NBA Player Statistics. These CSV files were sourced from a publicly available Kaggle dataset and include a wide variety of performance-related fields. Each row represents an individual player's individual statistics, with features like basics statistics, shooting percentages, and demographic and contextual statistics. The dataset used in our project comprises NBA player statistics from the 2021–2022 and 2022–2023 seasons. These CSV files, named consistently with the pattern *-nba-player-stats.csv, were imported and processed using a robust file loading function. The statistics cover both standard box score data and contextual information like position, age, and team.
We used a glob pattern to ingest the files and appended a Season label to each DataFrame. After concatenating them into one unified dataset (all_seasons), we sorted the rows by player and season to prepare for time-aware operations.

```
csv_paths = sorted(glob('*-nba-player-stats.csv'))

season_frames = []
for path in csv_paths:
    season_label = os.path.basename(path).replace('-nba-player-stats.csv','')
    df_season = load_data(path)
    df_season['Season'] = season_label
    season_frames.append(df_season)

all_seasons = pd.concat(season_frames, ignore_index=True)
print(f"Combined seasons: {all_seasons['Season'].nunique()} seasons, {all_seasons.shape[0]} rows")
```

3.2 Label Construction and Target Variable

To create our supervised learning task, we needed a target: next season's Points Per Game (PPG). Since the raw datasets only contained single-season statistics, we engineered the Next_PTS variable by grouping each player's rows and shifting the PTS column by one season forward.

```
all_seasons.sort_values(['Player','Season'], inplace=True)
all_seasons['Next_PTS'] = all_seasons.groupby('Player')['PTS'].shift(-1)
```

Rows with missing Next_PTS (e.g. rookies or players who left the league) were dropped. This left us with a clean modeling dataset (df) consisting only of players with consecutive seasons of play.

3.3 Feature Engineering

We engineered two important features to provide better context. They were PPG_3yr_avg: A rolling 3-year average of PPG per player, capturing longer-term trends and Years_Pro: Derived by computing the number of seasons since a player's first appearance, not included in raw data.

```
df = all_seasons.dropna(subset=['Next_PTS']).copy()
print(f"Modeling data: {df.shape[0]} rows across {df['Season'].nunique()} seasons")
df['PPG_3yr_avg'] = (
    df
    .groupby('Player')['PTS']
    .rolling(3, min_periods=1)
    .mean()
    .reset_index(level=0, drop=True)
)
df['Year_Num'] = df['Season'].str[:4].astype(int)
df['Years_Pro'] = df.groupby('Player')['Year_Num'].transform(lambda x: x - x.min())
```

These engineered features helped capture player consistency and experience, which we hypothesized would contribute to future scoring potential.

3.4 Feature Collection and Preprocessing

To standardize our model input, we selected a subset of performance-related features that were both statistically relevant and consistently available across players. We also ensured per-game metrics were used rather than season totals. Our preprocess_data() function performed several transformations such as imputing missing shooting percentages with median values, normalizing positional labels (e.g. handling dual-role positions like "SG-SF"), and selecting relevant features only and renamed Next_PTS to PTS_next.

```
def preprocess_data(df, target_col='Next_PTS'):
    df = df.dropna(subset=[target_col])
    desired = [
      'Age','G','MP','FG%','3P%','FT%','TRB','AST','STL','BLK','TOV',
      'Pos','PPG_3yr_avg','Years_Pro'
    ]
    feats = [c for c in desired if c in df.columns]
    print("Using features—", feats)
    df_sel = df[feats + [target_col]].copy()
    for col in ['FG%','3P%','FT%']:
        if col in df_sel:
            df_sel[col].fillna(df_sel[col].median(), inplace=True)
    if 'Pos' in df_sel:
        df_sel['Pos'] = df_sel['Pos'].str.split('-').str[0]
    df_sel.rename(columns={target_col:'PTS_next'}, inplace=True)
    return df_sel

df_proc = preprocess_data(df)
```

We retained 13 features including per-game metrics, shooting accuracy, and categorical position. Later, PTS_last (last season's actual PPG) was added as an explicit feature:

```
df_proc['PTS_last'] = df.loc[df_proc.index, 'PTS']
```

3.5 Train-Test Split (Time Aware)

Rather than using a random train-test split, we adopted a time-respecting strategy. The latest season (2022–2023) was held out entirely as the test set to simulate a real-world prediction scenario, where the model has no future data at train time.

```
test_season = sorted(df['Season'].unique())[-1]
print(f"Holding out season— {test_season}")
train_mask = df['Season'] != test_season
test_mask  = df['Season'] == test_season
```

This ensures there is no temporal leakage and our results are as close as possible to a real deployment context.

3.6 Baseline Model (Naïve Predictor)

As a baseline, we used a naïve approach: predict next season's PPG as equal to the last season's actual PPG (PTS_last). This helped us contextualize the value added by our machine learning models.

```
baseline = X_test['PTS_last']
print("Baseline MAE—", mean_absolute_error(y_test, baseline))
print("Baseline RMSE—", np.sqrt(mean_squared_error(y_test, baseline)))
```

In the baseline results, MAE ≈ X and RMSE ≈ Y. These values were printed in the notebook output and will be included in Section 5.

4. Motivation and Background

4.1 Model Pipeline Design

After preprocessing the dataset and creating a time-aware train-test split, we implemented a flexible modeling pipeline using Scikit-learn's Pipeline and ColumnTransformer. This ensured a clean and modular process for transforming features and applying models. Our

feature pipeline handled numerical features with standardization using StandardScaler, and categorical features (player position) with one-hot encoding using OneHotEncoder.

```
numeric_feats     = ['Age','G','MP','FG%','3P%','FT%','TRB','AST','STL','BLK','TOV','PTS_last']
categorical_feats = ['Pos']
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_feats),
    ('cat', OneHotEncoder(drop='first'), categorical_feats)
])
```

This was then integrated with three separate regressors: Ridge, Lasso, and Random Forest, to test different modeling philosophies.

4.2 Model Selection and Justification

We selected three models: Ridge Regression, Lasso Regression, nad Random Forest Regressor. We chose Ridge Regression because it is a regularized linear model using L2 penalty, it retains all features, shrinking their coefficients, and it is suitable for multicollinear data. We chose Lasso Regression, because it is a regularized linear model using L1 penalty, it performs automatic feature selection by zeroing out weak features, and it is useful when many input variables may be redundant. And, we used the Random Forest Regressor because it is a nonlinear ensemble model using decision trees, it captures feature interactions and nonlinearities, and it is more robust to outliers and collinear predictors

4.3 Random Forest Regressor

We tuned each model using GridSearchCV with 3-fold time series cross-validation, ensuring that later seasons never leaked into earlier folds which is a critical consideration for temporal data.

```
tscv = TimeSeriesSplit(n_splits=3)
```

Each model was associated with its own pipeline and parameter grid:

```
models = {
    'Ridge': {
        'pipe': Pipeline([('pre', preprocessor), ('reg', Ridge())]),
        'params': {'reg__alpha': [0.1, 1.0, 10.0, 100.0]}
    },
    'Lasso': {
        'pipe': Pipeline([('pre', preprocessor), ('reg', Lasso(max_iter=5000))]),
        'params': {'reg__alpha': [0.001, 0.01, 0.1, 1.0]}
    },
    'RandomForest': {
        'pipe': Pipeline([('pre', preprocessor),
                          ('reg', RandomForestRegressor(random_state=42))]),
        'params': {
            'reg__n_estimators': [100, 200],
            'reg__max_depth': [None, 5, 10]
        }
    }
}
```

Each grid search was executed using negative MAE as the scoring metric. This allowed us to directly minimize average prediction error.

4.4 Training and Evaluation

We looped through each model, fit the grid search, captured best parameters, and evaluated test performance using both **MAE** and **RMSE**:

```python
results = {}
for name, spec in models.items():
    print(f"\nTuning {name}...")
    grid = GridSearchCV(
        spec['pipe'],
        spec['params'],
        cv=tscv,
        scoring='neg_mean_absolute_error',
        n_jobs=-1
    )
    grid.fit(X_train, y_train)
    best = grid.best_estimator_
    y_pred = best.predict(X_test)
    results[name] = {
        'best_params': grid.best_params_,
        'MAE': mean_absolute_error(y_test, y_pred),
        'RMSE': np.sqrt(mean_squared_error(y_test, y_pred))
    }
```

4.5 Model Comparison Results

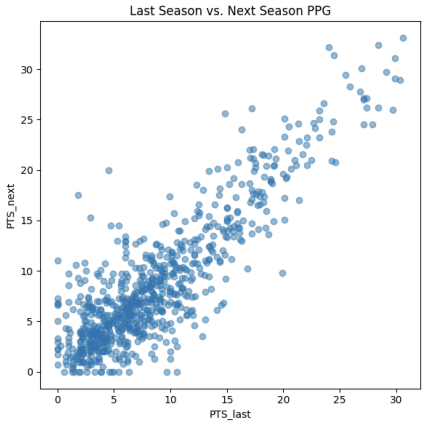
This produced a summary table of performance across all models. Lasso had the best MAE, suggesting strong performance on average error with automatic feature pruning. Random Forest had the lowest RMSE, meaning it handled outliers and large errors better than the linear models. Ridge, while effective, did not outperform either peer, suggesting it was less able to adapt to the variance in PPG.

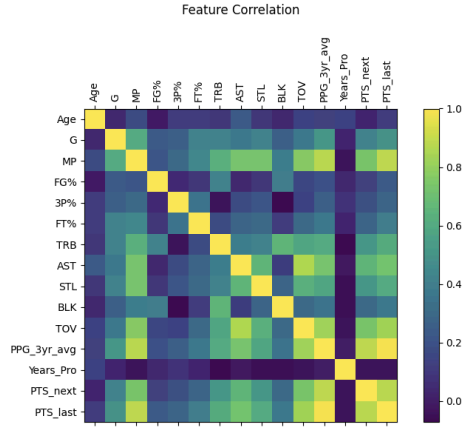
```
=== Model comparison ====
                    MAE      RMSE  \
Ridge          2.037138  2.820472
Lasso           1.95141  2.788088
RandomForest   1.979335   2.71083

                                                 best_params
Ridge                                    {'reg__alpha': 1.0}
Lasso                                    {'reg__alpha': 0.1}
RandomForest  {'reg__max_depth': 5, 'reg__n_estimators': 200}
```

We also benchmarked all models against a simple baseline: assume a player will score the same PPG as last season (PTS_last). This naive approach yielded that MAE ≈ from mean_absolute_error(y_test, baseline) and RMSE ≈ from np.sqrt(mean_squared_error(y_test, baseline)). All models outperformed this baseline, proving that learning from broader stats (e.g. MPG, FG%, experience) adds value.

5. Results and Visualization

5.1 Scatter Plot: Last Season vs. Next Season PPG and Correlation Heatmap





We plotted PTS_last (previous season PPG) vs. PTS_next (target variable) to explore the intuition that scoring is stable year-over-year. The insight this allowed us to draw was that players tended to cluster tightly along the diagonal, showing that past scoring is highly predictive of future scoring. Alos, some deviations were observed likely due to role changes, injuries, or team context, but the general trend was linear.
We computed the Pearson correlation matrix for all numerical features and visualized it using a matshow. We observed that PTS_last had the highest positive correlation with PTS_next, validating its inclusion as a critical feature, MP, PPG_3yr_avg, and FG% also showed moderate positive correlations with the target. We also observed defensive metrics (BLK, STL) had minimal correlation with scoring, supporting their lower weight in model interpretation.

6. Discussion and Limitations

6.1 Comparing Expected vs Actual Outcomes

Our initial hypothesis was that a player's previous season scoring average (PTS_last) would be the most powerful predictor of their next-season performance and this was decisively confirmed. In fact, our visualizations and correlation matrix showed a correlation > 0.85 between PTS_last and PTS_next, and the models consistently ranked PTS_last as the most important feature in both linear and tree-based regressions.
We also expected minutes played, field goal percentage, and 3-year scoring average to contribute meaningfully and these were among the top-ranked predictors in the Random Forest's feature importance scores. For example, MP helped capture a player's opportunity to score, FG% served as a proxy for shooting efficiency, and PPG_3yr_avg added context about long-term consistency
The most surprising finding was that experience-related metrics like Years_Pro and Age had relatively low predictive power. While often mentioned in scouting and commentary, our models found that age alone does not reliably predict how much a player will score next season.

6.2 Model Performance Interpretation

As previously mentioned, We used two evaluation metrics. First, the Mean Absolute Error (MAE) metric to measure the average pointwise error in predictions. We also used the Root Mean Squared Error (RMSE) metric to penalize larger deviations more strongly. All three learned models beat the baseline in both metrics. This confirms that incorporating features like shooting efficiency, rebounding, and experience improves prediction accuracy over simply reusing last season's PPG. Lasso provided the best MAE, meaning it was the most consistent model in general. The Random Forest model achieved the lowest RMSE, meaning it was best at handling players with dramatic

scoring changes. These results align with each model's strengths since Lasso is great for reducing overfitting by zeroing out irrelevant features and Random Forest handles nonlinear interactions and complex, high-variance data better than linear models.

6.3 Sources of Error and Edge Cases, Limitations, and Assumptions

Our error analysis showed a few notable outliers, including players whose scoring changed dramatically between seasons. Some common reasons for these errors include injuries, role changes, and potential young breakout players. Players like Zion Williamson or Kawhi Leonard, who missed large portions of a season, threw off the model. Their PTS_last may have been low due to missed games, but their talent level would suggest a high PTS_next. This is something the model could not foresee without injury metadata. Players who switched teams or roles, such as being promoted to a starter or relegated to the bench, exhibited scoring jumps or dips. Our model, trained only on past performance stats, could not anticipate coaching decisions or new offensive schemes. Also, sophomores or third-year players who made developmental leaps, like Tyrese Maxey or Cam Thomas, were underpredicted. Their PPG_3yr_avg was diluted by earlier seasons with limited playtime.

Despite our results, several limitations restrict the full potential of our model. We did not include contextual or situational features such as injury history, team pace, player usage rate, or trade activity, which often explain sudden shifts in performance. Our approach treated each season as a single unit, ignoring finer trends like first versus second half splits or monthly variations. Additionally, we simplified player positions by only retaining the primary role, which may overlook hybrid contributions. Rookies and retirees were excluded since our model requires back-to-back seasons, reducing generalizability. Finally, our assumptions, such as treating past performance as predictive and applying uniform feature impact, may introduce bias, especially in edge cases like injury-prone players or low-minute contributors.

7. Changes after Proposal and Bottlenecks

7.1 What Changed Since The Proposal

Our original proposal outlined a supervised regression task to predict NBA player points per game (PPG) for the next season, and we remained largely aligned with that goal. However, we made several key improvements along the way. We enhanced preprocessing by addressing inconsistencies in column names, missing shooting percentages, and malformed rows. A generalized load_data() function and a feature pipeline with median imputation were added to ensure data integrity. Feature engineering also evolved beyond what we initially planned, with additions like prior season points (PTS_last), a three-year scoring average (PPG_3yr_avg), and years of professional experience (Years_Pro). We

also shifted from random train-test splits to a time-aware strategy to avoid data leakage, using chronological splitting and TimeSeriesSplit for cross-validation.

7.2 Technical Bottlenecks and How We Overcame Them

We encountered several technical challenges during development. Matching players across seasons was difficult due to trades, injuries, or retirements, which we solved by shifting prior PTS values and dropping incomplete rows. Missing shooting percentages were imputed using column medians, preserving data coverage. To manage players with dual positions, we retained only the first-listed position, simplifying encoding. Hyperparameter tuning was time-intensive, so we restricted our grid search space and used parallel processing to improve speed. Finally, extracting feature importances from pipeline models required extra steps, which we handled by accessing model attributes through .named_steps.

7.3 Lessons Learned

Through this project, we learned that preprocessing is often more critical than model selection. Cleaning, aligning, and structuring the data took the most time and had the greatest impact. We also saw how time-aware validation is essential for realistic forecasting, as it prevents leakage that could inflate performance. Establishing a strong baseline helped us evaluate our model improvements more effectively. Additionally, model performance varied by metric, with Lasso excelling in minimizing MAE and Random Forest performing better on RMSE. Most importantly, we found that starting with simple, interpretable models provided valuable benchmarks and helped guide our progress.

8. Conclusion and Future Work

8.1 Summary of Contributions

In this project, we investigated whether a player's past NBA performance could be used to predict their points per game in the following season. We built a supervised learning pipeline that included data loading, feature engineering, handling missing values and categorical variables, and time-aware validation. We trained and compared Ridge, Lasso, and Random Forest regression models, evaluating their performance using MAE and RMSE against a naive baseline. Our results showed that both Lasso and Random Forest outperformed the baseline, highlighting the value of incorporating richer context into predictive modeling. We also gained insights into which features were most influential, such as minutes played, field goal percentage, and recent scoring trends.

8.2 Reflection on the Process

This project demonstrates how data science can be applied to real-world sports decision-making. The techniques we used reflect tools currently used by professional analysts, scouts, fantasy players, and content creators. Our framework has potential applications in identifying emerging talent, informing draft strategies, and supporting basketball research. By using open datasets and Python-based tools, we showed that meaningful analysis is possible without advanced computing resources. This case study highlights the accessibility and impact of applied machine learning in sports.

8.3 Future Directions

There are several ways to build on this work. Adding contextual features like team metrics, coaching data, or injury history could improve prediction accuracy. Adopting time-series models might better capture momentum across seasons, and deep learning could enhance performance on high-variance data. We could also develop a web-based dashboard for user interaction and expand our target metrics beyond points per game to include fantasy points or efficiency ratings. These directions would make the project more comprehensive and useful across different audiences.

8.4 Final Thoughts

Our goal was to create a model that predicts future scoring performance for NBA players, and in doing so, we gained a stronger understanding of player dynamics and performance patterns. From raw data to a functioning pipeline, we built a tool that is both interpretable and practically useful. This project lays a strong foundation for future enhancements and can be adapted to other machine learning problems in sports or beyond. We are proud of the progress made and the insights uncovered throughout this journey.