**UNIVERSIDAD DISTRITAL**
FRANCICO JOSÉ DE CALDAS
Acreditación institucional de alta calidad

Systems Analysis and Design Course

Group: 020-82

June 28, 2025


**UNIVERSIDAD DISTRITAL**
FRANCISCO JOSÉ DE CALDAS

David Felipe Ariza Ariza - 20221020029
Oscar Manuel Contreras Gacha - 20221020052
Julián David Méndez Lara - 20241020140

# NCAA Basketball Tournament - System Simulation

## 1 Introduction

This workshop builds upon the foundations established in Workshops 1 and 2, where an analysis of the NCAA basketball tournament was conducted to identify relevant data and design a system capable of predicting the outcomes of matches between participating teams. In this stage, the focus shifts to the implementation of a simulation, which serves as a critical tool for evaluating the system's architecture. By simulating scenarios, it becomes possible to assess the performance of the predictive model, detect potential errors, and identify areas for improvement. The simulation also provides insights into the reliability and adaptability of the system, contributing to the development of a more accurate and efficient solution.

## 2 Methodology

### 2.1 Data Preparation

The datasets were cleaned and reduced to only the most essential information. The data identified as fundamental included tournament results, regular season results, and team seed rankings. These elements were selected because they provide a clear overview of each team's historical performance. Tournament results reflect how teams perform under pressure in elimination games, while regular season results offer a broader picture of consistency and overall strength. Seed rankings, assigned before the tournament begins, indicate expert expectations and can serve as a useful reference for comparing actual outcomes. Focusing on these core datasets helps simplify the system without losing valuable insights, ensuring that the predictive model is built on relevant and meaningful information.

On the other hand, location-based data such as the cities where games were played was not included as fundamental information in the simulation. While game location can sometimes influence team performance due to factors like travel distance or home-court advantage, in the context of the NCAA tournament, most games are played at neutral sites. This reduces the potential impact of location on the outcome. Furthermore, including city data would have added complexity without significantly improving the model's predictive power. Since the goal of the simulation is to focus on performance-based metrics, the decision was made to prioritize historical results and seeding over geographical information.

### 2.2 Simulation Planning

The simulation will focus on evaluating the system's performance in predicting NCAA basketball game outcomes. Specifically, it will simulate the process from data ingestion through prediction generation to measure how effectively the system handles end-to-end workflows. The scenario includes repeated runs of model training and prediction phases to observe behavior over time and under varying data loads.

The simulation will activate each module of the system architecture defined in Workshop #2 as follows:

- ***Ingestion Module:*** Will be exercised by loading batches of historical and test data repeatedly.

- ***Feature Engineering Module:*** Will transform raw data into predictive features, testing computational cost and consistency.
- ***Training Module***: Will retrain models on different training sets to test accuracy drift and computational resource usage.
- ***Prediction Module:*** Will apply trained models to new data, evaluating accuracy and speed of inference.
- ***Output Writer Module:*** Will export prediction results, simulating external system communication and result formatting.

### 2.3 Simulation Implementation

To implement the simulation, each phase of the predictive system was translated into modular Python code using Jupyter Notebooks. These notebooks reflect the structure and limitations of the system design and simulate a workflow from data input to prediction output.

1. ***Ingestion:*** Loads historical NCAA data.
2. ***Feature Engineering***: Generates predictive features.
3. ***Training:*** Builds the machine learning model using the prepared data.
4. ***Predict:*** Uses the trained model to make predictions.
5. ***Output Writer:*** Saves the results for further analysis or presentation.

### 2.4 Executing the Simulation

Four simulations were carried out, each using a different set of variables.

The first simulation had a set of every variables (all team average stats), as our first simulation, we wanted to use all of those variables, the results of Kaggle was a score of 0.16093.

The second simulation had just a set of variables (some team average stats) such as Team 1 average score, Team 2 average score, T1 average point difference, T1 average FGM3(three points made by team 1), and so on.

The result of that simulation was not bad, but not as good as we wanted, the kaggle score was 0.16256.

For the next simulation (third simulation) we implemented the seeds of each team and seeds difference between the two teams, as a variable, and the same set of the first simulation too, and we found that these variables (seeds and seeds difference) were very important and had a great influence on the forecast. The result was 0.11929.

And for the last simulation (fourth simulation) we used all the stats, and the seeds, and to our surprise it got the best score, 0.11790. So we keep that set of variables.

## 3 Main results

We realized that, as mentioned in the last workshop, the seeds were a very important variable that has a great influence on the system, as shown in Figure 1 and Figure 2.

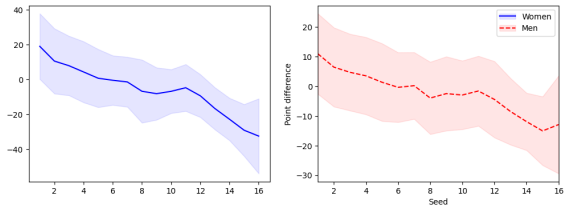


Figure 1: Point Difference vs Team 2 Seed

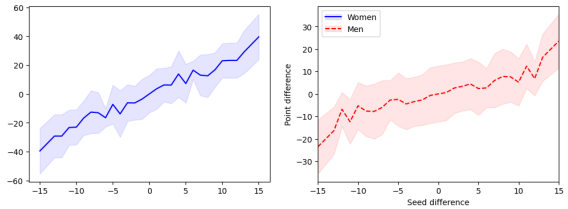


Figure 2: Point Difference vs Seeds Difference

The next thing we did, was a graphic of the point difference and its win probability, for men and women:

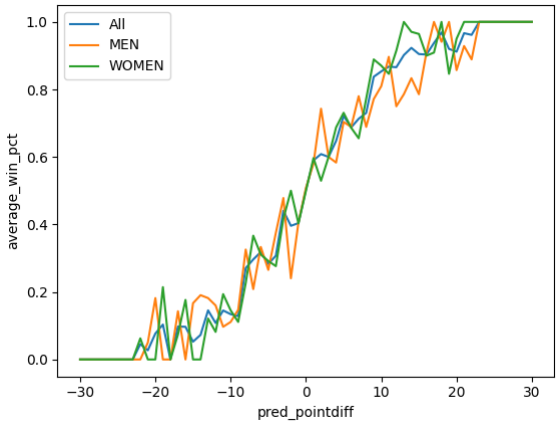


Figure 3: Average Win Prob vs Point Difference

And, as expected, the win probability increases as the point difference does, too.

And lastly we made another graphic , but with the predicted point difference vs the win probability, and with those graphics, we found the probability of each game, depending on the predicted point difference of them.

# 4 Code highlights

This section presents key code snippets from the Jupyter Notebooks used for the simulation. These highlights illustrate the most critical steps in our pipeline, from data ingestion to the final output, demonstrating the practical implementation of the system design outlined in Workshop #2.

## 4.1 Data Ingestion Module

This function represents the entry point of our simulation pipeline. Its primary responsibility is to load all the necessary raw datasets from the Kaggle competition files into memory. This modular

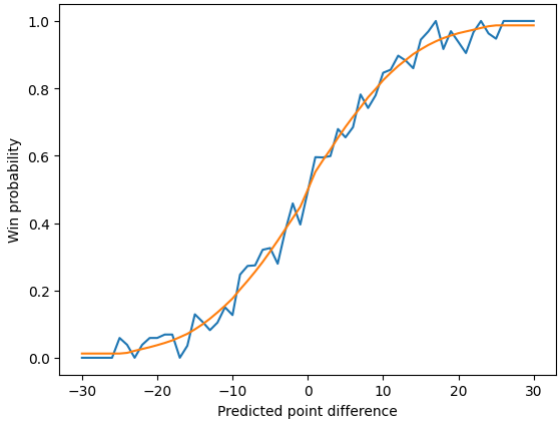


Figure 4: Win Prob vs Predicted Point Difference

approach ensures that all data sources are managed in a single, consistent location before any processing begins.

```python
def ingest_data(self):
    # Load the datasets for the simulation
    menRegularResults = pd.read_csv(f"{self.data_dir}MRegularSeasonDetailedResults.csv")
    menTourneyResults = pd.read_csv(f"{self.data_dir}MNCAATourneyDetailedResults.csv")
    menSeeds = pd.read_csv(f"{self.data_dir}MNCAATourneySeeds.csv")

    womenRegularResults = pd.read_csv(f"{self.data_dir}WRegularSeasonDetailedResults.csv")
    womenTourneyResults = pd.read_csv(f"{self.data_dir}WNCAATourneyDetailedResults.csv")
    womenSeeds = pd.read_csv(f"{self.data_dir}WNCAATourneySeeds.csv")
    self.data = [menRegularResults, menTourneyResults, menSeeds, womenRegularResults, womenTourneyResults, womenSeeds]
```

Figure 5: Ingestion

This code loads the detailed results and seed information for both the men's and women's tournaments. By encapsulating this logic in an ingest_data function, we create a clear and reusable component that forms the foundation for all subsequent feature engineering and training tasks

## 4.2 Feature Engineering

### 4.2.1 Data Preparation and Adjustment

he first stage of feature engineering involves preparing and standardizing the raw game data. The function highlighted below performs several critical tasks: it adjusts team statistics to account for overtime periods, ensuring fair comparison between games of different lengths. It also restructures the data to have a consistent "T1" and "T2" format and creates the initial PointDiff feature, which serves as the primary target for our predictive model.

```python
# Data preparation with adjustments for overtime and difference characteristics
def prepare_data(df):
    df = df[["Season", "DayNum", "LTeamID", "LScore", "WTeamID", "WScore", "NumOT", "LFGM", "LFGA", "LFGM3", "LFGA3",
             "LFTM", "LFTA","LOR", "LDR", "LAst", "LTO", "LStl", "LBlk", "LPF","WFGM", "WFGA",
             "WFGM3", "WFGA3", "WFTM", "WFTA", "WOR", "WDR", "WAst", "WTO", "WStl", "WBlk", "WPF"]]
    # adjustment factor for overtimes, as more stats are accumulated during overtimes
    adjOverTime = (40 + 5 * df["NumOT"]) / 40
    adjCols = ["LScore", "WScore", "LFGM", "LFGA", "LFGM3", "LFGA3", "LFTM", "LFTA", "LOR", "LDR", "LAst", "LTO",
               "LStl","LBlk", "LPF", "WFGM", "WFGA", "WFGM3", "WFGA3", "WFTM", "WFTA", "WOR", "WDR", "WAst", "WTO",
               "WStl", "WBlk", "WPF"]
    for col in adjCols:
        df[col] = df[col] / adjOverTime

    dfSwap = df.copy()

    # Replace W with T1_ and L with T2_ in column names
    df.columns = [x.replace("W", "T1_").replace("L", "T2_") for x in list(df.columns)]
    dfSwap.columns = [x.replace("L", "T1_").replace("W", "T2_") for x in list(dfSwap.columns)]
    output = pd.concat([df, dfSwap]).reset_index(drop=True)

    # Adding difference features
    output["PointDiff"] = output["T1_Score"] - output["T2_Score"]
    output["win"] = (output["PointDiff"] > 0) * 1
    output["men_women"] = (output["T1_TeamID"].apply(lambda t: str(t).startswith("1"))) * 1  # 0: women, 1: men
    return output
```

Figure 6: Adjustment and Point difference

This preparation is essential for creating a clean and reliable base dataset. The overtime adjustment, in particular, is a key step to prevent statistical skew from games that last longer than the

standard 40 minutes.

### 4.2.2 Seed Incorporation

As discovered during our simulations, team seeding is one of the most powerful predictors. This next code snippet is arguably the most impactful piece of feature engineering in our system. It merges the tournament seed information with the match data and calculates

Seed_diff, a direct comparison of the seeding between the two competing teams.

```
# adding team seeds and seed diff
seeds_T1 = seeds[["Season", "TeamID", "seed"]].copy()
seeds_T2 = seeds[["Season", "TeamID", "seed"]].copy()
seeds_T1.columns = ["Season", "T1_TeamID", "T1_seed"]
seeds_T2.columns = ["Season", "T2_TeamID", "T2_seed"]

tourneyData = tourneyData[["Season", "T1_TeamID", "T2_TeamID", "PointDiff", "win", "men_women"]]
tourneyData = pd.merge(tourneyData, seeds_T1, on=["Season", "T1_TeamID"], how="left")
tourneyData = pd.merge(tourneyData, seeds_T2, on=["Season", "T2_TeamID"], how="left")
tourneyData["Seed_diff"] = tourneyData["T2_seed"] - tourneyData["T1_seed"]
tourneyData
```

Figure 7: Adding team seeds and seed diff

The implementation of this feature was directly responsible for the most significant improvement in our model's performance, validating the hypothesis from our initial systems analysis. It confirms that a team's official ranking is a critical factor and must be included to achieve a competitive prediction score

## 4.3 Model Training Module

For model training, we implemented a robust "leave-one-season-out" cross-validation strategy. This technique is ideal for sports data as it simulates a real-world scenario where we predict a future tournament using only data from past seasons. This prevents data leakage and provides a more realistic estimate of the model's generalization performance.

```
from sklearn.metrics import mean_absolute_error

models = {}
oof_mae = []
oof_preds = []
oof_targets = []
oof_ss = []

# leave-one-season out models
for oof_season in set(tourneyData.Season):
    x_train = tourneyData.loc[tourneyData["Season"] != oof_season, features].values
    y_train = tourneyData.loc[tourneyData["Season"] != oof_season, "PointDiff"].values
    x_val = tourneyData.loc[tourneyData["Season"] == oof_season, features].values
    y_val = tourneyData.loc[tourneyData["Season"] == oof_season, "PointDiff"].values
    s_val = tourneyData.loc[tourneyData["Season"] == oof_season, "Season"].values

    dtrain = xgb.DMatrix(x_train, label=y_train)
    dval = xgb.DMatrix(x_val, label=y_val)
    models[oof_season] = xgb.train(
        params=param,
        dtrain=dtrain,
        num_boost_round = num_rounds,
    )
    preds = models[oof_season].predict(dval)
    print(f"oof season {oof_season} mae: {mean_absolute_error(y_val, preds)}")
    oof_mae.append(mean_absolute_error(y_val, preds))
    oof_preds += list(preds)
    oof_targets += list(y_val)
    oof_ss += list(s_val)

print(f"average mae: {np.mean(oof_mae)}")
```

Figure 8: Training Model

The code iterates through each season, using it as a validation set while training an XGBoost model on all other available seasons. The mean absolute error (MAE) is calculated for each season, and the out-of-fold (oof) predictions are stored for later analysis and calibration, ensuring our performance metrics are reliable.

## 4.4 Prediction

Instead of using the raw outputs from the XGBoost model directly, we implemented a calibration step to refine our predictions. Raw model outputs are not always well-calibrated probabilities. This code uses a UnivariateSpline to map our model's predicted point differences to a smoother, more reliable win probability, which can improve the final Brier score.

```python
from scipy.interpolate import UnivariateSpline
from sklearn.metrics import brier_score_loss


t = 25
dat = list(zip(oof_preds, np.array(oof_targets)>0))
dat = sorted(dat, key = lambda x: x[0])
pred, label = list(zip(*dat))
spline_model = UnivariateSpline(np.clip(pred, -t, t), label, k=5)
spline_fit = np.clip(spline_model(np.clip(oof_preds, -t, t)), 0.01, 0.99)
print(f"brier: {brier_score_loss(np.array(oof_targets)>0, spline_fit)}")
df["spline"] = spline_fit
xdf = df.clip(-30,30).groupby('pred_pointdiff')[['spline','label']].mean().reset_index()
```

Figure 9: Prediction procces

This post-processing step demonstrates a more advanced approach to refining predictions. By fitting a spline to our out-of-fold predictions, we generate a final probability that is better calibrated, which is crucial for metrics like the Brier score that penalize overconfident and incorrect predictions.

### 4.5 Output Writer Module

The final module in our pipeline is the Output Writer. This component is responsible for formatting the predictions according to the Kaggle competition's requirements and saving them to a CSV file.

```python
X['Pred'] = X.apply(lambda row: manual_overrides[row['ID']] if row['ID'] in manual_overrides else row['Pred'], axis=1)

X['Pred'] = X['Pred'].round(6)
X[['ID','Pred']].to_csv('..\predictions.csv',index=None)
```

Figure 10: csv with the results

This snippet handles the final touches: it rounds the predicted probabilities to the required precision and generates the predictions.csv file with the 'ID' and 'Pred' columns. An interesting feature of this module is the logic for manual_overrides, which was designed to allow for manual adjustments to specific game predictions if external analysis provided a reason to do so, adding a layer of expert-in-the-loop flexibility to the system.

## 5 Review of Findings

Based on the results of the four simulations and the visual and statistical analysis performed, several key findings emerged:

1. **Full Stats Set (Simulation 1):** Using all average team statistics yielded a moderate performance (Kaggle score: 0.16093). It established a baseline but showed that quantity does not guarantee quality in feature selection.

2. **Reduced Stats Set (Simulation 2):** Limiting the variables to a smaller, more curated set slightly worsened the performance (score: 0.16256), suggesting that some less obvious statistics still contributed value.

3. **Seeds and Seed Difference (Simulation 3):** Introducing seed data and seed difference resulted in a significant improvement (score: 0.11929). This reinforced the hypothesis that seed rankings are highly predictive in tournament outcomes.

4. **All Stats + Seeds (Simulation 4):** Combining full statistics with seed information gave the best result (score: 0.11790), confirming that both historical performance and tournament rank are essential for accurate predictions.

5. **Win Probability Correlation:** Graphs confirmed a strong relationship between point difference and win probability. This validated the use of `PointDiff` as a predictive target and justified its central role in model training.

6. **Seed Impact Visualization:** Visuals comparing seed values with point difference showed clear trends, highlighting the practical importance of seeding beyond just numerical input.