

# Workshop 3

## Collegiate basketball tournaments

### Systems Analysis and Design

#### Universidad Francisco José de Caldas

---

Member #1:	Carlos Andres Celis Herrera
Member #2:	Xiomara Salome Arias Arias
Member #3:	Nicolas Romero Rodriguez
Project:	Systems Engineering
Professor:	Carlos Andres Sierra Virgües

---

## 1 Introduction

As a continuation of the work done for designing and implementing our predictive system for the NCAA 2025 basketball tournament we aim to generate simulations for various components of the system, these simulations would serve as a tool to evaluate how the system would behave under various conditions before making a full deployment. This report has its foundations on a previous system analysis and a architectural design phase, and seeks to evaluate how this proposed system would function at the moment of solving this real-life problem.

This document will explore various processes of the forecasting system such as data ingestion, preparation, feature engineering and prediction engine, including metrics to make an accurate evaluation of these components' performance, while also showcasing the application of systems engineering principles that ensure adaptability and flexibility on the proposed design, while also ensuring that the model's behavior can be meaningfully interpreted and improved. Ultimately, the simulation not only measures the technical efficacy of the prediction system but also identifies opportunities for refinement to improve its performance and output.

### 1.1 Methodology

The simulations for the project follow the same structure of the general system's flow, summarized as follows

#### 1. Data Preparation:

- Collect historical team statistics from Kaggle datasets.
- Filter out outdated information.
- Clean inconsistent or missing data.
- Engineer features such as seed difference, scoring margin, and efficiency metrics.

#### 2. Model Training and Baseline Prediction:

- Train the classification model using the processed data.
- Generate win probabilities for all valid matchups in the NCAA 2025 tournament.
- Store predictions as the baseline output.

#### 3. Perturbation Scenario:

- Apply controlled random noise ( $\pm 2\%$  to  $\pm 5\%$ ) to selected input variables.
- Rerun the prediction pipeline on the modified inputs.

#### 4. Analysis and Comparison:

- Compare outputs from both scenarios.
- Analyze average probability shifts, percentage of large deviations, and visualize prediction distributions through graphics.

These phases allow us to evaluate the system's predictive stability, sensitivity to variation, and resilience in the presence of uncertain input data.

## 2 Data Classification & selection

Continuing with the competition solution and following the analysis and design of the basketball tournament system carried out in workshops 1 and 2. The study and preparation of historical data, which will be used to create a prediction model and at the same time predict the results of the NCAA tournament games. Sonas et al. (2025), it's a very important step for the project.

In order to produce a dataset usable for training the machine learning model the competition website provides 36 CSV files containing historical information of past tournaments. These files contain a large amount of information that dates from many years ago. Exploration of this data allows us to determine which of the files are convenient to include in the model in order to generate the most accurate predictions possible while reducing the system's complexity as much as possible in order to improve its performance and make these predictions feasible.

The following is an explanation of which of the 36 csv files were selected for the prediction engine, what information is contained within them and why was it selected

With that in mind, the Kaggle competition page provides a series of .csv documents containing a large amount of historical data on college basketball games and teams, dating back many years. With this historical data, strategies are explored and developed to predict the results of March Madness. To develop our prediction model in accordance with the competition's system design, the use of the following .csv documents is essential. These are:

- **MNCAATourneySeeds.csv and WNCAATourneySeeds.csv:** These files identify the pre-seeded teams in each NCAA tournament. The Seed field includes a region prefix (W, X, Y, Z) + number (01–16) + optionally a letter (a/b) for play-in games. The seed is the number assigned to a team within a tournament region, ranging from 1 (strongest) to 16 (weakest). It is useful for calculating seed difference, one of the most relevant features in prediction.

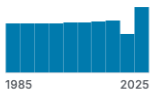

# Season	Seed	TeamID
	W01 2% W02 2% Other (2546) 97%	
1985	W01	1207
1985	W02	1210
1985	W03	1228
1985	W04	1260
1985	W05	1374
1985	W06	1288
1985	W07	1393

Figure 1: Tourney Seeds Data Structure

- **MSeasons.csv and WSeasons.csv :** These files identify the different seasons included in the historical data. With DayZero, you can convert DayNum into an actual date, facilitating temporal analysis. With this, you can standardize all seasons so that games coincide on the same scale of days.


DayZero	RegionW	RegionX	RegionY	RegionZ
	East 90% Atlanta 5% Other (2) 5%	West 29% Midwest 29% Other (17) 41%	Midwest 59% South 17% Other (10) 24%	West 61% South 17% Other (9) 22%
10/29/1984	East	West	Midwest	Southeast
10/28/1985	East	Midwest	Southeast	West
10/27/1986	East	Southeast	Midwest	West
11/02/1987	East	Midwest	Southeast	West
10/31/1988	East	West	Midwest	Southeast
10/30/1989	East	Midwest	Southeast	West
10/29/1990	East	Southeast	Midwest	West
11/04/1991	East	West	Midwest	Southeast

Figure 2: Seasons Data Structure

- **MGameCities.csv** and **WGameCities.csv** : These files identify all games since the 2010 season, along with the city where they were played. Regular season games, the NCAA tournament, and other postseason tournaments are included. The CRType column (Regular / NCAA / Secondary) allows you to distinguish between different types of games, which is useful if you want to study home field advantage, long trips, or city altitude.

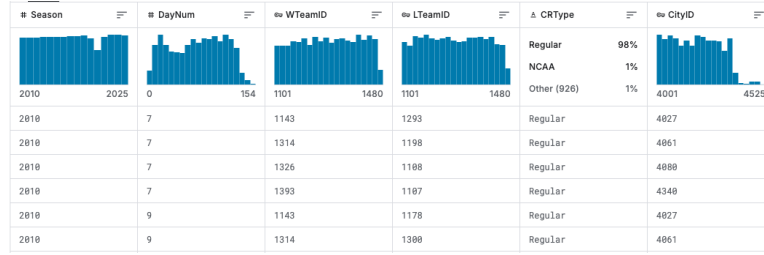


Figure 3: Data Structure for Cities and Teams

- **MTeams.csv** and **WTeams.csv** : These files identify the different university teams present in the dataset. Columns such as FirstD1Season and LastD1Season help filter active teams.

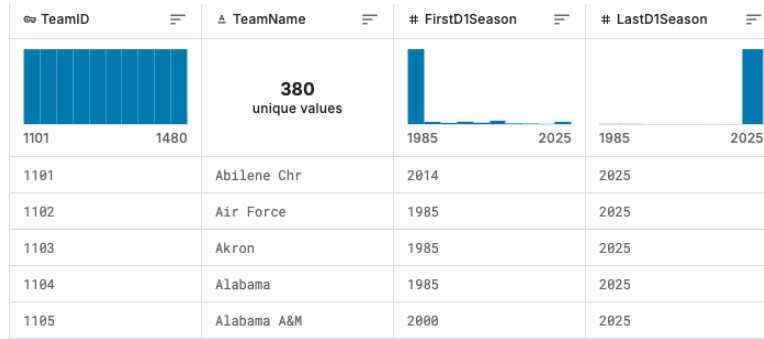


Figure 4: Data Structure for Teams

- **MRegularSeasonDetailedResults.csv** and **WRegularSeasonDetailedResults.csv** : These files provide team statistics for many regular seasons with historical data. Statistics by team and by game: rebounds, steals, turnovers, free throws, etc. This is the key file for building performance features.

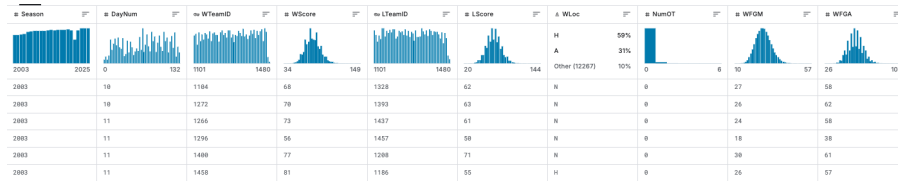


Figure 5: Data Structure of Match Statistics

- **MNCAATourneyCompactResults.csv** and **WNCAATourneyCompactResults.csv** : These files identify the results of the NCAA tournament game by game for all seasons of historical data. It contains NCAA tournament results exclusively and is useful for training classification models with actual “who won” results.

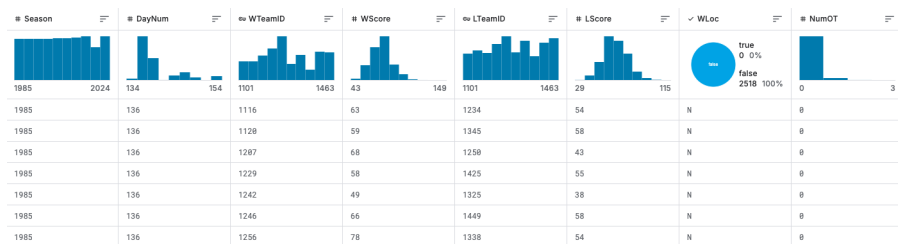


Figure 6: Data Structure of Match Statistics

- **SampleSubmissionStage1.csv:** This file illustrates the submission format for the Stage 1 “preparation” competition. The ID field is in the format SSSS\_XXXX\_YYYY, where: SSSS = season XXXX, YYYY = team IDs.


ID	# Pred
507108 unique values	 0.5 0.5
2021_1101_1102	0.5
2021_1101_1103	0.5
2021_1101_1104	0.5
2021_1101_1105	0.5
2021_1101_1106	0.5
2021_1101_1107	0.5
2021_1101_1108	0.5
2021_1101_1110	0.5

Figure 7: Data Structure of Solution

## 2.1 Data Characteristics

The dataset provides detailed historical information on basketball games focused on the NCAA men’s and women’s college tournament in the United States, used primarily to model and predict results for the March Madness competition. Its main features are summarized below:

- **Temporary coverage:** Data from the 1985 season to 2025 (depending on the file).
- **Level of granularity:** Match-level data (compact and detailed), team by season, and tournament structure.
- **Volume:** Thousands of matches per season, with the possibility of generating combinations of hypothetical matchups.
- **Variables:** Mix of categorical variables (team IDs, seeds, results), numerical variables (points, rebounds, turnovers), and temporal variables (day number, season).
- **Joints:** The keys **TeamID**, **Season** and **DayNum** allow multiple files to be linked together.
- **Data distribution:** Low seeds win more frequently; some variables have a normal distribution (points), others are skewed (losses).

The decision of using these specific files its so that we can use specific features that would be the key of the prediction engine, these features and how to calculate their values are listed in the table 1.

Feature	Description / Calculation
Seed difference	Difference between the seeds assigned to each team: SeedB – SeedA
Win-Loss Ratio	Total number of wins divided by total number of games played
Average points scored	Average number of points scored per game
Average points allowed	Average number of points conceded per game
Offensive efficiency	Points scored divided by estimated possessions
Defensive efficiency	Points allowed divided by estimated possessions
Scoring margin	Average point differential per game (scored – conceded)

Table 1: Summary of *features* used as model input

## 3 Simulation Planning

### 3.1 Scenario Description

This simulation focuses on evaluating the behavior of a predictive system when estimating win probabilities for all possible matchups in the NCAA 2025 tournament. The task aligns with the core challenge of the March Machine Learning Mania 2025 competition, which requires generating predictions for thousands of hypothetical games that could occur during the tournament.

The system receives historical and statistical data as input—such as offensive and defensive efficiency, seed difference, and prior performance—and produces as output the probability that Team A defeats Team B.

Two evaluation conditions are considered:

- A **baseline scenario**, where predictions are generated using clean, unaltered input data.
- A **perturbed scenario**, where controlled variations (random noise) are introduced to input variables to simulate real-world uncertainty, errors, or noise in the data.

This approach allows us to assess the model's robustness and sensitivity to small changes in the data, helping identify whether such perturbations significantly affect predictions. The simulation replicates a real life setting in which data is not always precise or stable and examines whether the system maintains coherence and reliability under uncertainty.

### 3.2 System Design Alignment

This scenario is closely aligned with the system architecture outlined in Workshop 2, activating the following components:

- **Data Ingestion:** Loads historical data from Kaggle .csv files, including team statistics, seed history, and game outcomes.
- **Data Preprocessing:** Filters data by season, handles missing or inconsistent values, and standardizes formats.
- **Feature Engineering:** Constructs predictive variables such as seed difference, win-loss ratio, average scoring margin, and efficiency metrics.
- **Prediction Engine:** Trains and applies a machine learning model (e.g., logistic regression) to estimate the probability of Team A beating Team B based on historical features.
- **Storage Layer:** Stores model outputs under both baseline and perturbed input conditions for later analysis.
- **Analytics and Reporting:** Visualizes distributions of predicted probabilities, compares shifts caused by noise, and summarizes model performance.

**Note:** The bracket simulator module is excluded from this simulation since the focus is on individual matchup prediction rather than full tournament progression.

### 3.3 Systems Engineering Principles

The design of this simulation incorporates various systems engineering principles, including:

- **Modularity:** Each component (ingestion, preprocessing, prediction, analysis) has a single responsibility and can be developed or improved independently.
- **Fault Tolerance:** The system handles missing or noisy data and tests the model's robustness through perturbed inputs.
- **Scalability:** Capable of generating predictions across thousands of matchups, and so if the number of possible matchups were to increase the accuracy of the system would not be affected.
- **Performance Metrics:**
  - Total execution time for generating predictions.
  - Average shift in predicted probabilities under noise.

- Percentage of matchups with shifts greater than  $\pm 0.10$ .
- Comparison of prediction distributions between baseline and perturbed data.
- **Simplicity and Traceability:** Built using standard Python tools (`pandas`, `scikit-learn`, etc.) with reproducible workflows.

## 4 Implementation

This following section aims to showcase important elements of the simulations code that give a better understanding of these tests functioning.

### 4.1 Data ingestion

This part of the code represents the data ingestion process in which the previously selected files enter the system, which include:

- The regular season results
- Tournament results
- Team seeds across tournaments
- Format required for the project

In order to make the prediction engine more accurate all of the data previous to 2015 is discarded, and from the remaining data, the system extracts from them the most important variables contained within the files such as the teams id's and their respective seeds across tournaments

```
def load_data(self):
    """Load seeds and tournament results from CSV files"""
    self.seeds = pd.read_csv("Data/MNCAATourneySeeds.csv")
    self.seeds["SeedNum"] = self.seeds["Seed"].str.extract(r"(\d+)").astype(int)
    self.results = pd.read_csv("Data/MNCAATourneyCompactResults.csv")
    print("Data loaded successfully.")
    print(self.seeds.head())
    print(self.results.head())
```

Figure 8: Data ingestion process

### 4.2 Feature engineering

Another important part of the code is generating a neutral matchup dataset for training, this dataset is the one used to train the predictive model, here each historical game is restructured into a matchup between a team A and a team B, each of them containing a binary variable named winner (0 means the team lost (1 means the team wins). In order to maintain a balance in the system is necessary to represent each possible matchup twice in the dataset in the first instance of the matchup team A is marked as the winner and in the second instance is marked as the loser, this way we avoid generating bias towards specific teams.

### 4.3 Prediction engine

As a first approach towards the use of logistic regression the system initially uses a single feature for the model, in this case the sole parameter for the prediction is the difference between seeds, represented by a single subtraction between team A and B seeds. To implement this we used the functions related to logistic regression contained in the sklearn library, this functions also allowed to generate some accuracy metrics.

```

# First version: winner is TeamA
win = merged.copy()
win["TeamA"] = win["WTeamID"]
win["TeamB"] = win["LTeamID"]
win["Winner"] = 1

# Second version: loser is TeamA
lose = merged.copy()
lose["TeamA"] = lose["LTeamID"]
lose["TeamB"] = lose["WTeamID"]
lose["SeedDiff"] = -lose["SeedDiff"]
lose["Winner"] = 0

```

Figure 9: Creation of neutral dataset

```

def train_model(self):
    """Train logistic regression model and report performance"""
    X_train, X_test, y_train, y_test = train_test_split(self.X, self.y, test_size=0.2, stratify=self.y)
    model = LogisticRegression()
    model.fit(X_train, y_train)
    self.model = model

    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[: , 1]
    acc = accuracy_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_prob)

    print(f"Accuracy: {acc:.4f}")
    print(f"AUC Score: {auc:.4f}")

```

Figure 10: Training process

#### 4.4 simulation experiments

For this tests there were chosen two possible scenarios, the baseline simply implements the information from the dataset with no additions. This case applies the trained logistic regression model to all possible team matchups in the NCAA 2025 tournament. For each unique combination of TeamA vs TeamB, we compute the seed difference and use it to estimate the probability that TeamA wins. This case aims to represent an ideal situation with no noise in the data

```

def predict_baseline(self):
    """Generate baseline predictions for all possible 2025 matchups"""
    teams = self.seeds[self.seeds["Season"] == 2025][["TeamID", "SeedNum"]]
    pairs = []

    for A, B in combinations(teams["TeamID"], 2):
        sA = teams.loc[teams["TeamID"] == A, "SeedNum"].values[0]
        sB = teams.loc[teams["TeamID"] == B, "SeedNum"].values[0]
        pairs.append({"TeamA": A, "TeamB": B, "SeedDiff": sB - sA})
        pairs.append({"TeamA": B, "TeamB": A, "SeedDiff": sA - sB})

    df = pd.DataFrame(pairs)
    df["Pred"] = self.model.predict_proba(df[["SeedDiff"]])[:, 1]
    df.to_csv("Data/baseline_predictions.csv", index=False)
    self.matchups = df

    print("Baseline predictions generated:")
    print(df.head())

```

Figure 11: Baseline case

The second case tested simulates data uncertainty and measurement errors by introducing random noise into the input feature 'SeedDiff' from the 2025 tournament matchups.

The perturbation is a random value between 2 and 5 percent of the original seed difference. We then reapply

```
def predict_perturbed(self):
    """Generate predictions under perturbed input conditions"""
    df = self.matchups.copy()

    # Add random noise (±2% to ±5%) to simulate data uncertainty
    np.random.seed(42)
    noise = np.random.uniform(-0.05, 0.05, size=len(df)) * df["SeedDiff"].abs()
    df["PerturbedSeedDiff"] = df["SeedDiff"] + noise

    # Rename column to match training-time feature name
    df["PerturbedPred"] = self.model.predict_proba(
        df[["PerturbedSeedDiff"]].rename(columns={"PerturbedSeedDiff": "SeedDiff"})
    )[:, 1]

    df.to_csv("Data/perturbed_predictions.csv", index=False)
    self.perturbed = df

    print("Perturbed predictions generated:")
    print(df[["TeamA", "TeamB", "SeedDiff", "PerturbedSeedDiff", "Pred", "PerturbedPred"]].head())
```

Figure 12: perturbation case

the trained model to these noisy inputs to observe how sensitive the system is to small variations.

The final highlight of the simulation generates a graphic that represent the difference in outputs made by the model with noise and the model without noise using the matplotlib library, as a way to measure the system's response to the noise

```
def analyze(self):
    """Compare baseline vs perturbed predictions and generate histogram"""
    df = self.perturbed.copy()
    df["Delta"] = df["PerturbedPred"] - df["Pred"]
    avg_delta = df["Delta"].abs().mean()
    pct_shifted = (df["Delta"].abs() > 0.10).mean() * 100

    print(f"Average change in prediction: {avg_delta:.4f}")
    print(f"Matchups with >±0.10 change: {pct_shifted:.2f}%")

    # Scatter plot: baseline vs perturbed predictions
    plt.figure(figsize=(6, 6))
    plt.scatter(df["Pred"], df["PerturbedPred"], alpha=0.6)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlabel("Baseline Prediction")
    plt.ylabel("Perturbed Prediction")
    plt.title("Prediction Shift: Baseline vs Perturbed")
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("Data/prediction_shift.png")
    plt.close()
```

Figure 13: final metrics

## 5 Results

This section presents the results obtained from the simulation designed to predict possible matchups in the NCAA 2025 tournament. The developed system takes as input historical team information—specifically, the difference in tournament seedings (SeedDiff)—and produces as output the probability that Team A defeats Team B for every possible pairing.

The simulation was structured in two main phases. In the first phase, baseline predictions were generated using clean and unaltered input data. In the second phase, a perturbed scenario was created by introducing controlled noise into the SeedDiff feature, in order to evaluate the model's sensitivity to small fluctuations in the input.

Figure 14 and Figure 15 illustrate the datasets used. The first shows a sample of the match-level data with seed values, and the second displays team-level features computed from regular season statistics. Although the model in this version uses only SeedDiff, the additional team statistics (e.g., scoring margin, win-loss ratio) were computed for future extensions.

The logistic regression model, trained using only the *SeedDiff* feature, achieved an **accuracy of 63.90%** and an **AUC of 66.42%** on the test set. While relatively simple, the model demonstrated a reasonable ability



	Season	WTeamID	LTeamID	SeedA	SeedB	SeedDiff
0	2015	1214	1264	16	16	0
1	2015	1279	1140	11	11	0
2	2015	1173	1129	11	11	0
3	2015	1352	1316	16	16	0
4	2015	1112	1411	2	15	13

Figure 14: Sample of tournament matchups and seed values

	Season	TeamID	GamesPlayed	Wins	AvgPointsScored	AvgPointsAllowed	TotalPointsScored	TotalPointsAllowed	WinLossRatio	ScoringMargin
0	2015	1101	28	7	61.000000	71.857143	1708	2012	0.250000	-10.857143
1	2015	1102	29	12	64.724138	65.862069	1877	1910	0.413793	-1.137931
2	2015	1103	34	20	67.352941	63.235294	2290	2150	0.588235	4.117647
3	2015	1104	31	17	66.645161	64.516129	2066	2000	0.548387	2.129032
4	2015	1105	28	8	61.285714	68.714286	1716	1924	0.285714	-7.428571

Figure 15: Team statistics computed from regular season data

to distinguish winners from losers based solely on seed differences, highlighting the predictive value of this structural tournament variable. However, the modest performance also suggests that adding more informative features could enhance the model’s discriminative power.

Figure 16 shows an example of the baseline predictions produced by the model for 2025 matchups.

	TeamA	TeamB	SeedDiff	Pred
0	1181	1104	1	0.537314
1	1104	1181	-1	0.456373
2	1181	1458	2	0.577319
3	1458	1181	-2	0.416490
4	1181	1112	3	0.616335

Figure 16: Baseline predictions using clean input data

In the perturbation scenario, random noise between 2% and 5% was applied to the SeedDiff values to simulate uncertainty in the input. After running the prediction pipeline again, the model produced adjusted probabilities. The average absolute change in predicted probability was only **0.0051**, and no matchup experienced a shift greater than 0.10 (Figure 17). This confirms that the model is highly stable under small input fluctuations and demonstrates strong robustness to noise.

Additionally, Figure 18 presents a visualization of the distribution of changes in predicted probabilities between the baseline and perturbed scenarios. The scatter plot demonstrates that the predicted probabilities under perturbed input conditions remain nearly identical to those obtained under the baseline scenario. The tight alignment of the points along the diagonal indicates minimal deviation in predictions, confirming that the model is not significantly affected by small variations in the input data. This supports the conclusion that the system is stable and robust under controlled noise.

## 6 Discussion of findings

Given the execution of the simulations based on the scenarios, we obtain a series of results that lead us to ask the following questions: How can we reduce the error obtained from the logistic regression? Can the noise added in certain features affect the accuracy of the prediction? And how can the other features be implemented to improve the accuracy of the models?

- **How can we reduce the error obtained in logistic regression?** Although the model achieves an accuracy of 63.90% and an AUC of 66.42%, its performance is limited when considering only a single variable. To improve accuracy, it is recommended to include more relevant features in the training, such

	TeamA	TeamB	PerturbedSeedDiff	Pred	PerturbedPred
0	1181	1104	1.031236	0.537314	0.538574
1	1104	1181	-1.048521	0.456373	0.454420
2	1181	1458	2.083920	0.577319	0.580638
3	1458	1181	-2.075920	0.416490	0.413500
4	1181	1112	2.925958	0.616335	0.613491

Figure 17: Predictions under perturbed input (SeedDiff + noise)

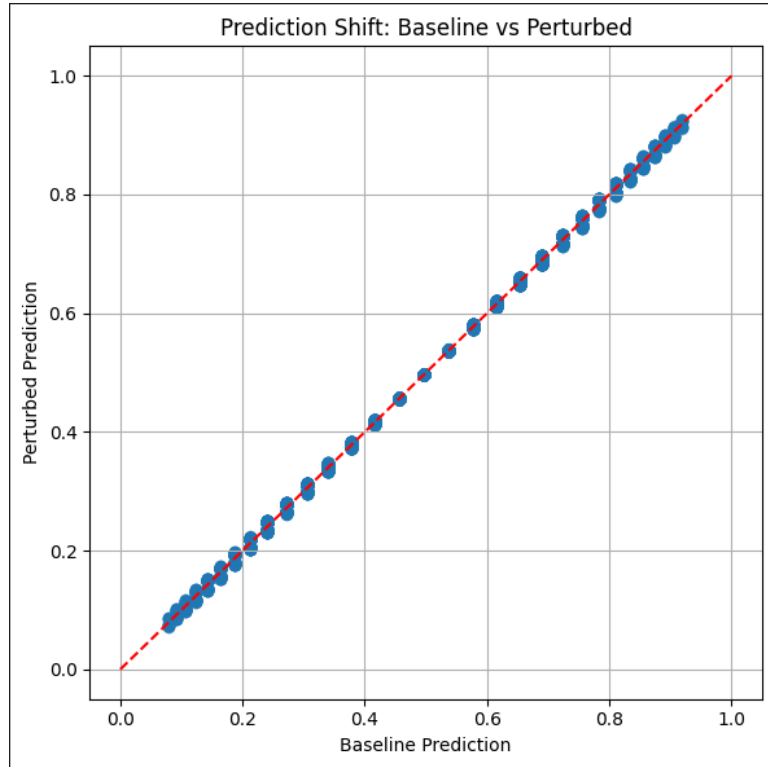


Figure 18: Distribution of prediction shifts caused by perturbation

as offensive efficiency, average scoring margin, and winning percentage. In addition, more complex models (e.g., Random Forest, Gradient Boosting) that capture nonlinear relationships between variables can be tested.

- **Does the noise added in certain *features* affect the accuracy of the prediction?** Yes, the simulation with controlled perturbations in SeedDiff showed that even small variations in this variable can cause noticeable changes in the generated probabilities of victory. This shows that the model is somewhat sensitive to the quality of the input data and that in real environments, where there is uncertainty or measurement errors, its performance could be affected. However, it should be noted that this first simulation focused on the future seed and adding a percentage error. This may change with respect to another that is much more sensitive or weighty, affecting match predictions. This finding highlights the importance of designing robust systems against noisy data.
- **How can other *features* be implemented to improve the accuracy of the models?** In model training, in this first simulation and test, the model was trained with only one feature taken into account from those defined in the first part. The use of others, and even better, the combination of these, can be used to train much more accurate models. This allows for the modeling of multivariate relationships and complex patterns that the seed difference alone cannot reflect. Implementing this enriched set of features in future simulations will not only increase the accuracy of the model, but also its generalization capacity and robustness in the face of uncertainty or noise in the data, thus bringing it closer to the real conditions of the tournament.

## References

Sonas, J., Mooney, P., Howard, A., & Cukierski, W. 2025, March Machine Learning Mania 2025, <https://www.kaggle.com/competitions/march-machine-learning-mania-2025>