NBA MVP Prediction

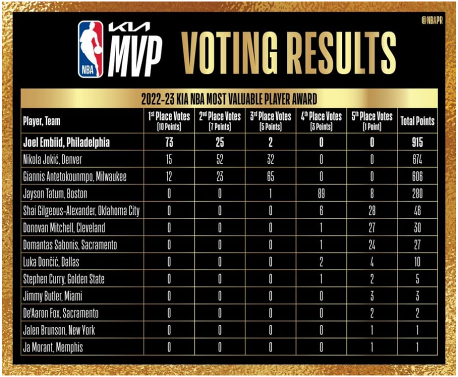
HOML Project

Eloi Beurtheret, Raphael Leonardi, Baptiste Pras

Université Paris-Saclay

Jeudi 11 Décembre 2025

Introduction

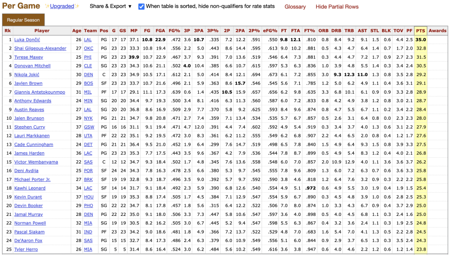


@NBAPR

MVP VOTING RESULTS

2022-23 KIA NBA MOST VALUABLE PLAYER AWARD

Player, Team	1st Place Votes (10 Points)	2nd Place Votes (7 Points)	3rd Place Votes (5 Points)	4th Place Votes (3 Points)	5th Place Votes (1 Point)	Total Points
Joel Embiid, Philadelphia	**73**	**25**	**2**	**0**	**0**	**915**
Nikola Jokić, Denver	15	52	32	0	0	674
Giannis Antetokounmpo, Milwaukee	12	23	65	0	0	606
Jayson Tatum, Boston	0	0	1	89	8	280
Shai Gilgeous-Alexander, Oklahoma City	0	0	0	6	28	46
Donovan Mitchell, Cleveland	0	0	0	1	27	30
Domantas Sabonis, Sacramento	0	0	0	1	24	27
Luka Dončić, Dallas	0	0	0	2	4	10
Stephen Curry, Golden State	0	0	0	1	2	5
Jimmy Butler, Miami	0	0	0	0	3	3
De'Aaron Fox, Sacramento	0	0	0	0	2	2
Jalen Brunson, New York	0	0	0	0	1	1
Ja Morant, Memphis	0	0	0	0	1	1



Per Game ✨Upgraded✨ Share & Export ▾ When table is sorted, hide non-qualifiers for rate stats Glossary Hide Partial Rows

Regular Season

Rk	Player	Age	Team	Pos	G	GS	MP	FG	FGA	FG%	3P	3PA	3P%	2P	2PA	2P%	eFG%	FT	FTA	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS	Awards
1	Luka Dončić	26	LAL	PG	17	17	37.1	**10.8**	**22.9**	.472	3.6	**10.7**	.335	7.2	12.2	.591	.550	**9.8**	**12.1**	.810	0.8	8.4	9.2	9.1	1.5	0.6	4.4	2.5	**35.0**	
2	Shai Gilgeous-Alexander	27	OKC	PG	23	23	33.3	10.8	19.4	.556	2.2	5.0	.443	8.6	14.4	.595	.613	9.0	10.2	.881	0.5	4.3	4.7	6.4	1.4	0.7	1.7	2.0	32.8	
3	Tyrese Maxey	25	PHI	PG	23	23	**39.9**	10.7	22.9	.467	3.7	9.3	.391	7.0	13.6	.519	.546	6.4	7.3	.881	0.3	4.4	4.7	7.2	1.7	0.9	2.7	2.3	31.5	
4	Donovan Mitchell	29	CLE	SG	23	23	34.3	10.6	21.1	.502	**4.0**	10.4	.385	6.6	10.7	.615	.597	5.3	6.3	.836	1.0	3.9	4.8	5.5	1.4	0.3	3.4	2.4	30.5	
5	Nikola Jokić	30	DEN	C	23	23	34.9	10.5	17.1	.612	2.1	5.0	.414	8.4	12.1	.694	.673	6.1	7.2	.855	3.0	**9.3**	**12.3**	**11.0**	1.3	0.8	3.5	2.8	29.2	
6	Jaylen Brown	29	BOS	SF	23	23	33.7	10.7	21.6	.496	2.1	5.9	.363	8.6	**15.7**	.546	.545	5.6	7.1	.785	1.2	5.0	6.2	4.9	1.1	0.4	3.6	3.1	29.1	
7	Giannis Antetokounmpo	31	MIL	PF	17	17	29.1	11.1	17.3	.639	0.6	1.4	.435	**10.5**	15.9	.657	.656	6.2	9.8	.635	3.3	6.8	10.1	6.1	0.9	0.9	3.3	2.8	28.9	
8	Anthony Edwards	24	MIN	SG	20	20	34.4	9.7	19.3	.500	3.4	8.1	.416	6.3	11.3	.560	.587	6.0	7.2	.833	0.8	4.2	4.9	3.8	1.2	0.8	3.0	2.1	28.7	
9	Austin Reaves	27	LAL	SG	20	20	36.8	8.6	16.9	.509	2.9	7.7	.370	5.8	9.2	.625	.593	8.4	9.6	.874	0.8	4.7	5.5	6.7	1.1	0.2	3.4	2.2	28.4	
10	Jalen Brunson	29	NYK	PG	21	21	34.7	9.8	20.8	.471	2.7	7.4	.359	7.1	13.4	.534	.535	5.7	6.7	.857	0.5	2.6	3.1	6.4	0.8	0.0	2.3	2.3	28.0	
11	Stephen Curry	37	GSW	PG	16	16	31.1	9.1	19.4	.471	4.7	12.0	.391	4.4	7.4	.602	.592	4.9	5.4	.919	0.3	3.4	3.7	4.0	1.3	0.6	3.1	2.2	27.9	
12	Lauri Markkanen	28	UTA	PF	22	22	35.1	9.2	19.5	.472	3.0	8.3	.361	6.2	11.2	.555	.549	6.2	6.8	.907	2.2	4.4	6.5	2.0	0.8	0.4	1.2	1.7	27.6	
13	Cade Cunningham	24	DET	PG	21	21	36.4	9.5	21.0	.452	1.9	6.4	.299	7.6	14.7	.519	.498	6.5	7.8	.840	1.5	4.9	6.4	9.3	1.5	0.8	3.9	3.3	27.5	
14	James Harden	36	LAC	PG	23	23	35.3	7.7	17.5	.443	3.5	9.6	.367	4.2	7.9	.536	.544	7.8	8.7	.899	0.5	4.9	5.4	8.3	1.2	0.4	4.0	2.1	26.8	
15	Victor Wembanyama	22	SAS	C	12	12	34.7	9.3	18.4	.502	1.7	4.8	.345	7.6	13.6	.558	.548	6.0	7.0	.857	2.0	10.9	12.9	4.0	1.1	3.6	3.6	3.7	26.2	
16	Deni Avdija	25	POR	SF	24	24	34.3	7.8	16.3	.478	2.5	6.6	.380	5.3	9.7	.545	.555	7.8	9.6	.809	1.3	6.0	7.2	6.3	0.7	0.6	3.6	3.3	25.8	
17	Michael Porter Jr.	27	BRK	SF	19	19	32.8	9.3	18.7	.496	3.5	9.0	.392	5.7	9.7	.592	.590	3.8	4.6	.818	1.2	6.4	7.6	3.2	0.9	0.3	2.2	2.2	25.8	
18	Kawhi Leonard	34	LAC	SF	14	14	31.7	9.1	18.4	.492	2.3	5.9	.390	6.8	12.6	.540	.554	4.9	5.1	**.972**	0.6	4.9	5.5	3.0	1.9	0.4	1.9	1.5	25.4	
19	Kevin Durant	37	HOU	SF	19	19	35.3	8.8	17.4	.505	1.7	4.5	.384	7.1	12.9	.547	.554	5.9	6.7	.890	0.3	4.5	4.8	3.9	1.0	0.6	2.8	2.5	25.3	
20	Devin Booker	29	PHO	SG	22	22	34.7	8.1	17.8	.457	1.8	5.6	.315	6.4	12.2	.522	.506	7.0	8.0	.874	1.0	3.3	4.3	6.7	0.9	0.4	3.7	2.9	25.0	
21	Jamal Murray	28	DEN	PG	22	22	35.0	9.1	18.0	.506	3.3	7.3	.447	5.8	10.6	.547	.597	3.6	4.0	.898	0.5	4.0	4.5	6.8	1.1	0.3	2.4	1.6	25.0	
22	Norman Powell	32	MIA	SG	19	19	30.5	8.2	16.2	.505	3.0	6.7	.445	5.2	9.4	.547	.598	5.5	6.3	.867	0.4	3.2	3.6	2.4	1.1	0.3	1.9	2.5	24.8	
23	Pascal Siakam	31	IND	PF	23	23	34.2	9.0	18.6	.481	1.8	4.9	.366	7.2	13.7	.522	.529	4.8	7.0	.683	1.6	5.4	7.0	4.1	1.3	0.5	2.2	2.8	24.5	
24	De'Aaron Fox	28	SAS	PG	15	15	32.7	8.4	17.3	.486	2.4	6.3	.379	6.0	10.9	.549	.556	5.1	6.0	.844	0.9	2.9	3.7	6.5	1.3	0.3	3.5	2.4	24.3	
25	Tyler Herro	26	MIA	SG	5	5	31.4	8.6	16.4	.524	3.0	6.2	.484	5.6	10.2	.549	.616	3.6	3.8	.947	0.6	4.0	4.6	2.2	1.2	0.6	1.2	1.4	23.8	

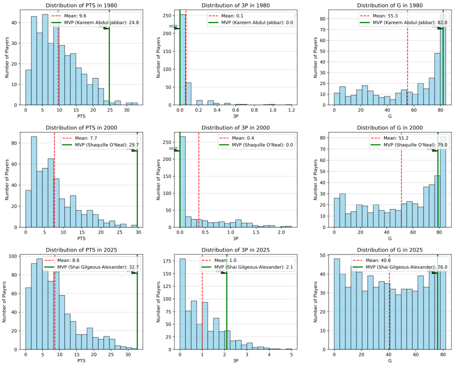
Webscrapping

Scrapped player statistics and standings from https ://www.basketball-reference.com/

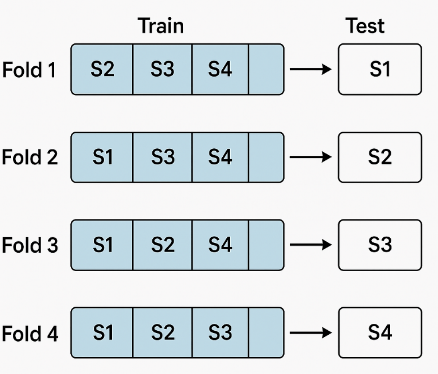
```python
print(f"[INFO] Downloading per-season raw data")
save_dir = os.path.join(raw_data_dir, save_dir)
os.makedirs(save_dir, exist_ok=True)

for year in range(start_year, end_year + 1):
    url = f"https://www.basketball-reference.com/leagues/NBA_{year}_per_game.html"
    print(f"[INFO] Downloading season {year} from {url}...")

    output_path = os.path.join(save_dir, f"NBA_{year}_per_game.csv")
    if os.path.exists(output_path):
        print(f"[SKIP] Skipping season {year}, file already exists.")
        continue


    try:
        response = requests.get(url, impersonate="chrome110")
        response.raise_for_status()

        # Parse the HTML table using pandas
        dfs = pd.read_html(response.text)
        df = dfs[0]

        # Remove header rows duplicated in the table
        df = df[df["Player"] != "Player"]
        df["Season"] = year

        # Save to CSV
        df.to_csv(output_path, index=False)
        print(f"[OK] Saved to {output_path}")

    except Exception as e:
        print(f"[WARN] Failed for season {year}: {e}, skipping")

print(f"[DONE] Saved raw data to {save_dir}")
```

Preprocessing

- Encoded player's position into a one-hot-vector
- Transformed Win-Lose ratio into an integer
- Incorporated team standing
- Normalized statistics per season



Leave-One-Season-Out Strategy



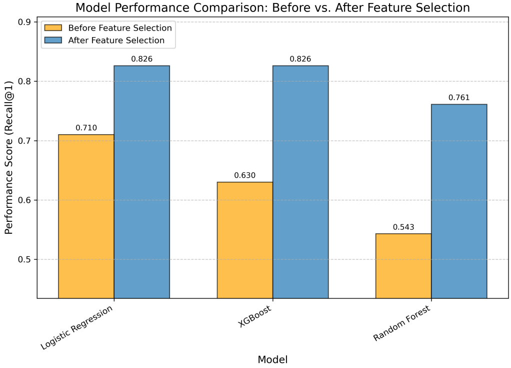

```python
# Process all years for this fold
for other_year in range(year_start, year_end + 1):
    other_year_str = str(other_year)
    year_folder = os.path.join(data_dir, other_year_str)

    # Check files exist
    file_paths = {file: os.path.join(year_folder, file) for file in files_to_process}
    if not all(os.path.isfile(path) for path in file_paths.values()):
        print(f"[WARN] Missing files for year {other_year}, skipping")
        continue

    # Load data
    df_X = pd.read_csv(file_paths["Data.csv"]).astype(np.float32)
    X = df_X.values
    y_top1 = pd.read_csv(file_paths["Y_top1.csv"]).values.squeeze().astype(np.int64)
    y_top10 = pd.read_csv(file_paths["Y_top10.csv"]).values.squeeze().astype(np.int64)
    df_names = pd.read_csv(file_paths["Name.csv"])

    if other_year == year:
        # Save test split
        np.savez_compressed(os.path.join(test_dir, "test.npz"),
                            X=X, y_top1=y_top1, y_top10=y_top10)
        df_names.to_csv(os.path.join(test_dir, "Name.csv"), index=False)
    else:
        # Accumulate for train split
        X_train_list.append(X)
        y_top1_train_list.append(y_top1)
        y_top10_train_list.append(y_top10)
        names_train.append(df_names)

# Save train split
if X_train_list:
    X_train = np.concatenate(X_train_list, axis=0)
    y_top1_train = np.concatenate(y_top1_train_list, axis=0)
    y_top10_train = np.concatenate(y_top10_train_list, axis=0)
    df_names_train = pd.concat(names_train, axis=0, ignore_index=True)

    np.savez_compressed(os.path.join(train_dir, "train.npz"),
                        X=X_train, y_top1=y_top1_train, y_top10=y_top10_train)
    df_names_train.to_csv(os.path.join(train_dir, "Name.csv"), index=False)
```

Feature Selection (1)

- Greedy Forward Selection
- Greedy Backward Selection
- Exhaustive Selection (unused because it's over 1 billion possibilities)

```python
def greedy_forward_selection(model_class, dataset_dir, pipeline_name, fixed_params,
                             output_dir, year_start, year_end, patience=3):
    """
    Greedy forward selection using Recall@1 as the scoring metric with early stopping.
    """
    print(f"[INFO] Running greedy forward selection (Recall@1) on dataset: {dataset_dir}")

    data = load_dataset(dataset_dir, year_start)
    num_features = data["X_train"].shape[1]
    available_features = list(range(num_features))
    selected_features = []
    best_score = -1
    no_improve_count = 0
    recall_scores = []

    for i in range(1, num_features + 1):
        current_best_score = -1
        feature_to_add = None

        for feat in tqdm(available_features, desc=f"Testing additions (current={len(selected_features)})", leave=False):
            candidate = selected_features + [feat]
            score = compute_recall_at_1_avg(model_class, fixed_params, dataset_dir, year_start, year_end, candidate)
            if score > current_best_score or (score == current_best_score and feat < (feature_to_add or float('inf'))):
                current_best_score = score
                feature_to_add = feat

        if current_best_score > best_score:
            best_score = current_best_score
            best_combination = selected_features + [feature_to_add]
            no_improve_count = 0
        else:
            no_improve_count += 1

        recall_scores.append(current_best_score)
        selected_features.append(feature_to_add)
        available_features.remove(feature_to_add)

        print(f"[INFO] Added feature {feature_to_add} | Total: {len(selected_features)} | Recall@1: {current_best_score:.3f}")

        if no_improve_count >= patience:
            print(f"[EARLY STOP] No improvement for {patience} consecutive steps. Stopping.")
            break
```

Feature Selection (2)



- Logistic Regression: ["Team Overall", "G", "MP", "FGA", "FG%", "3P", "FT", "FT%", "ORB", "DRB", "BLK", "PF", "AST"]
- XGBoost: ["Team Overall", "PTS", "TOV", "GS", "MP", "2PA", "3PA", "FT"]
- Random Forest: ["2P", "Team Overall", "PTS", "G", "TOV", "FT%", "PF"]

Models

We focused on three models :

- Logistic Regression ; simple model, originally intended as a baseline
- eXtreme Gradient Boosting ; chosen as our main gradient boosting model because it outperformed the other models in preliminary testing
- Random Forest ; as trees handle tabular data very well

For each player, we predict a probability of being the MVP. The one having the highest probability is then predicted as MVP (allows for a ranking)

Hyperparameters Tuning (1)

```
# Loop over seasons
global_results = []
for year in tqdm(range(year_start, year_end + 1), desc=f"{param_name}={param_value}", file=sys.stdout):
    # Load dataset
    data = load_dataset(datasets_dir, year)
    X_train = data["X_train"]
    y_train = data["y_top1_train"]
    X_test = data["X_test"]
    y_test = data["y_top1_test"]
    y10_test = data["y_top10_test"]
    player_names_test = data["Name_test"]

    if selected_feature_names is not None:
        all_feature_names = get_feature_names(dataset_name, year)
        selected_indices = [i for i, name in enumerate(all_feature_names) if name in selected_feature_names]
        X_train = X_train[:, selected_indices]
        X_test = X_test[:, selected_indices]

    # Train model
    model = model_class(**model_init_params)
    model.fit(X_train, y_train)

    # Evaluate
    results = evaluate_model(model, X_test, y_test, y10_test, player_names_test, top_ks=top_ks)
    global_results.append({
        "year": year,
        **results
    })

# Aggregate
df_results = pd.DataFrame(global_results)

# Summarize → same logic as summarize_global_results
summary_metrics = {}
```

Hyperparameters Tuning (2)

Pipeline: allStats_from1980

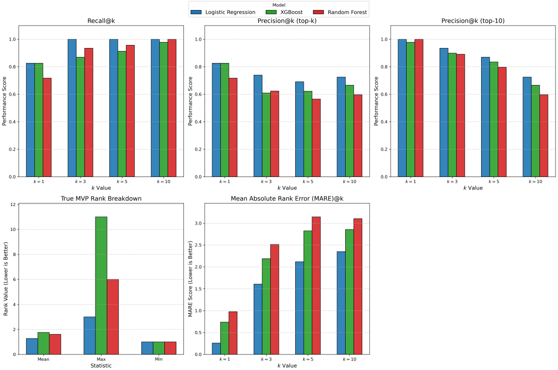
min_samples_leaf	Recall@1	Recall@3	Recall@5	Recall@10	Precision@1 (top-k)	Precision@3 (top-k)	Precision@5 (top-k)	Precision@10 (top-k)	Precision@1 (top-10)	Precision@3 (top-10)	Precision@5 (top-10)	Precision@10 (top-10)	Mean Abs Rank Error@1	Mean Abs Rank Error@3	Mean Abs Rank Error@5	Mean Abs Rank Error@10	True MVP avg rank	True MVP min rank	True MVP max rank	True MVP correct (rank=1)
1	0.500	0.870	0.913	0.978	0.500	0.652	0.604	0.611	1.000	0.928	0.835	0.611	1.630	2.217	2.822	2.891	2.261	1.000	12.000	0.500
2	0.543	0.826	0.935	1.000	0.543	0.630	0.622	0.644	1.000	0.928	0.830	0.644	1.065	2.246	2.900	2.980	2.109	1.000	8.000	0.543
5	0.478	0.848	0.913	1.000	0.478	0.652	0.626	0.663	1.000	0.942	0.839	0.663	1.326	2.058	2.817	3.024	2.217	1.000	8.000	0.478
10	0.500	0.848	0.957	1.000	0.500	0.630	0.630	0.677	1.000	0.928	0.852	0.677	1.457	2.283	2.848	2.941	2.174	1.000	8.000	0.500
20	0.478	0.761	0.913	1.000	0.478	0.616	0.639	0.690	1.000	0.928	0.870	0.690	1.413	2.457	2.761	2.893	2.435	1.000	9.000	0.478

- Logistic Regression: {"solver": "saga", "penalty": "l2", "class_weight": None, "max_iter": 50000, "C": 5.0}
- XGBoost: {"n_estimators": 100, "learning_rate": 0.1, "max_depth": 4, "scale_pos_weight": 5, "eval_metric": "logloss", "random_state": 42}
- Random Forest: {"n_estimators": 750, "max_depth": 15, "class_weight": None, "min_samples_leaf": 2, "random_state": 42}

Evaluation (1)

```python
# Top-k
for k in top_ks:
    # Recall@k: whether the true MVP is present in the top-k predictions (1 = yes, 0 = no)
    top_k_true = sorted_true[:k]
    hit = 1 in top_k_true
    results[f"top_{k}_hit"] = int(hit)
    vprint(f"[INFO] Top-{k} hit: {hit}")

    pred_top_k_names = sorted_names[:k]
    pred_top_k_real_ranks = sorted_y10[:k]

    # Precision@k vs real top-k: proportion of top-k predicted players who are ranked in the real top-k (according to available top-k annotations)
    pred_top_k_real_ranks = sorted_y10[:k]
    real_topk_mask = (y_10 >= 1) & (y_10 <= k)
    nb_real_topk = real_topk_mask.sum()
    n_in_real_topk = sum(1 for r in pred_top_k_real_ranks if 1 <= r <= k)
    denom_topk = min(k, nb_real_topk) if nb_real_topk > 0 else 1
    precision_at_k_exact = n_in_real_topk / denom_topk
    results[f"precision_at_{k}_topk"] = precision_at_k_exact
    vprint(f"[METRIC] Precision@{k} vs real top-{k}: {precision_at_k_exact:.3f}")

    # Precision@k vs real top-10: proportion of top-k predicted players who are ranked in the real top-10 (according to available top-10 annotations)
    real_top10_mask = (y_10 >= 1) & (y_10 <= 10)
    nb_real_top10 = real_top10_mask.sum()
    n_in_real_top10 = sum(1 for r in pred_top_k_real_ranks if 1 <= r <= 10)
    denom_top10 = min(k, nb_real_top10) if nb_real_top10 > 0 else 1
    precision_at_k_top10 = n_in_real_top10 / denom_top10
    results[f"precision_at_{k}_top10"] = precision_at_k_top10
    vprint(f"[METRIC] Precision@{k} vs real top-10: {precision_at_k_top10:.3f}")

    # Mean absolute rank error@k: distance between predicted rank table and real rank table
    abs_errors = []
    for pred_rank_idx, real_rank in enumerate(pred_top_k_real_ranks, 1):
        if real_rank == -1:
            assumed_rank = 11  # If not in real top-10, treat as 11th
        else:
            assumed_rank = real_rank
        abs_error = abs(pred_rank_idx - assumed_rank)
        abs_errors.append(abs_error)
    mean_abs_error = np.mean(abs_errors)
```

Evaluation (2)



Conclusion

Predicted MVP for year 2026 : Shai Gilgeous-Alexander (prob=0.8047)
Predicted top-10 for year 2026 :

- 1 Shai Gilgeous-Alexander (prob=0.8047)
- 2 Nikola Jokić (prob=0.7473)
- 3 Luka Dončić (prob=0.0355)
- 4 Cade Cunningham (prob=0.0344)
- 5 Jalen Duren (prob=0.0037)
- 6 Isaiah Hartenstein (prob=0.0031)
- 7 Austin Reaves (prob=0.0029)
- 8 Jamal Murray (prob=0.0028)
- 9 Ajay Mitchell (prob=0.0021)
- 10 Alperen Şengün (prob=0.0018)