Orange Data Hoops Challenge

"The Data-Alley Oops"

Team Members:

Isha Agrawal
Chenna Keshav Chinthakindi
Sree Chandan Kamireddi
Kushwanth Sai Chandu Meesala

Goal: Understanding Player Performance and predicting the winning Shot.

Challenges:

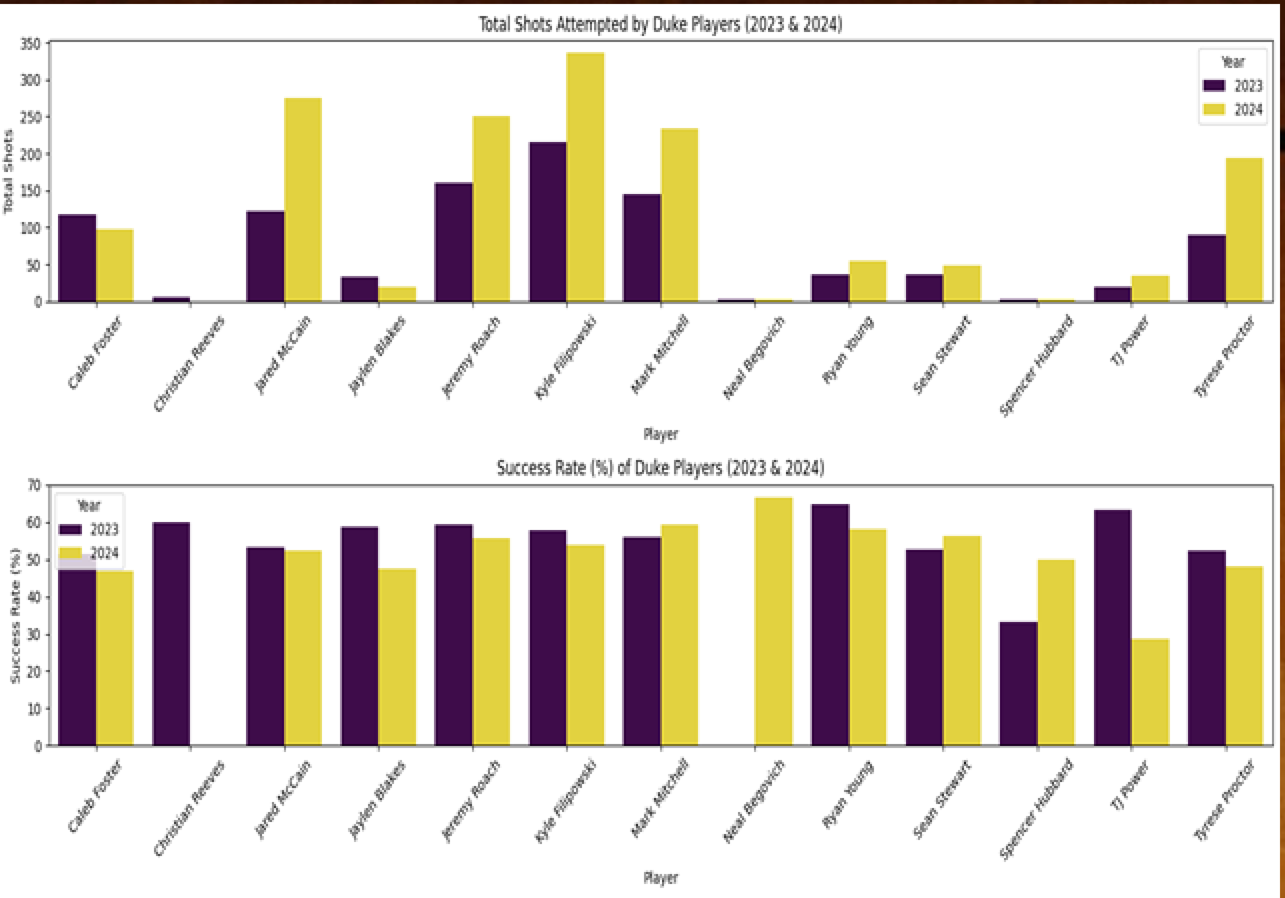
- **Data Inefficiency**: Initial data lacked accuracy for predictions, so we engineered new features focused on clutch and efficiency metrics.

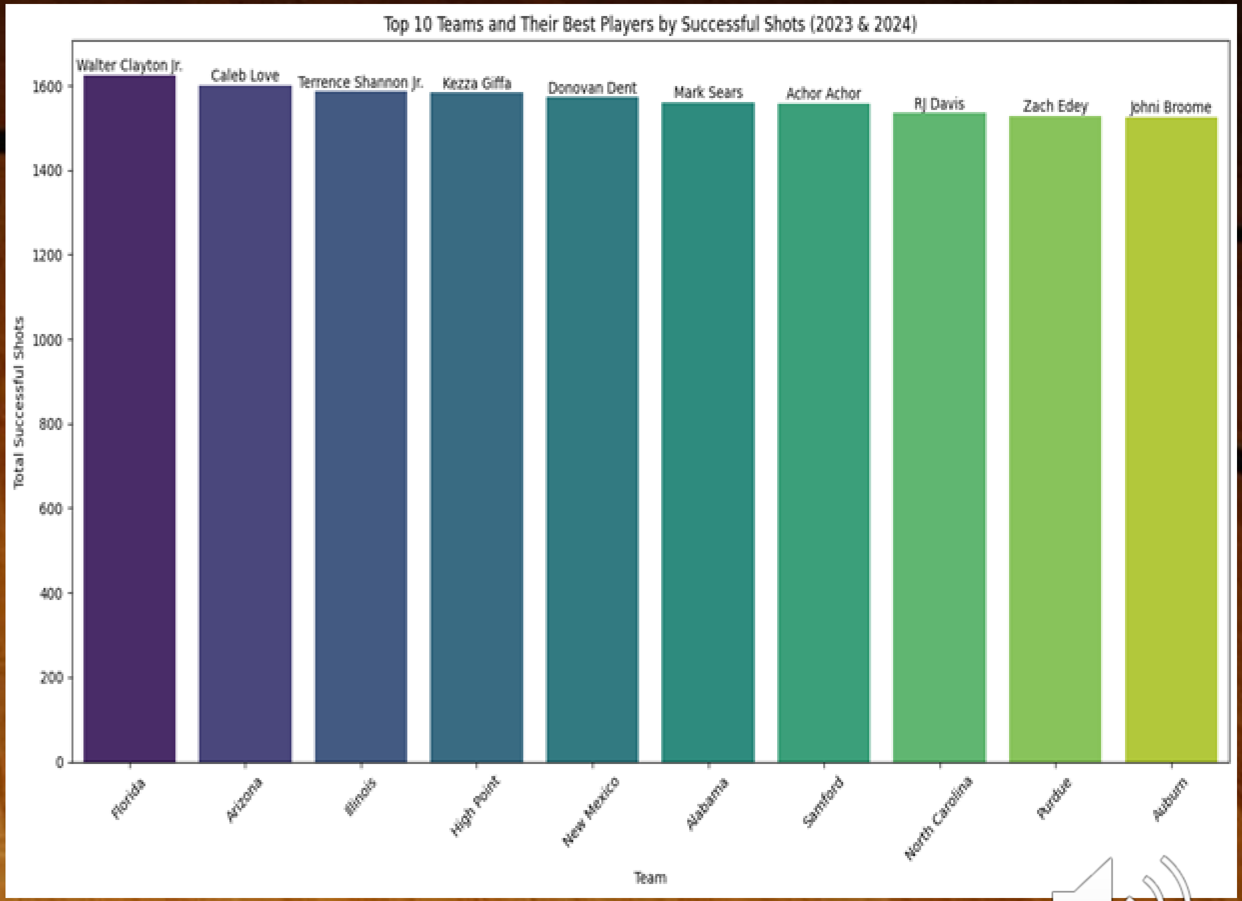
- **Missing Values**: The dataset had numerous nulls, requiring careful imputation to ensure model reliability.

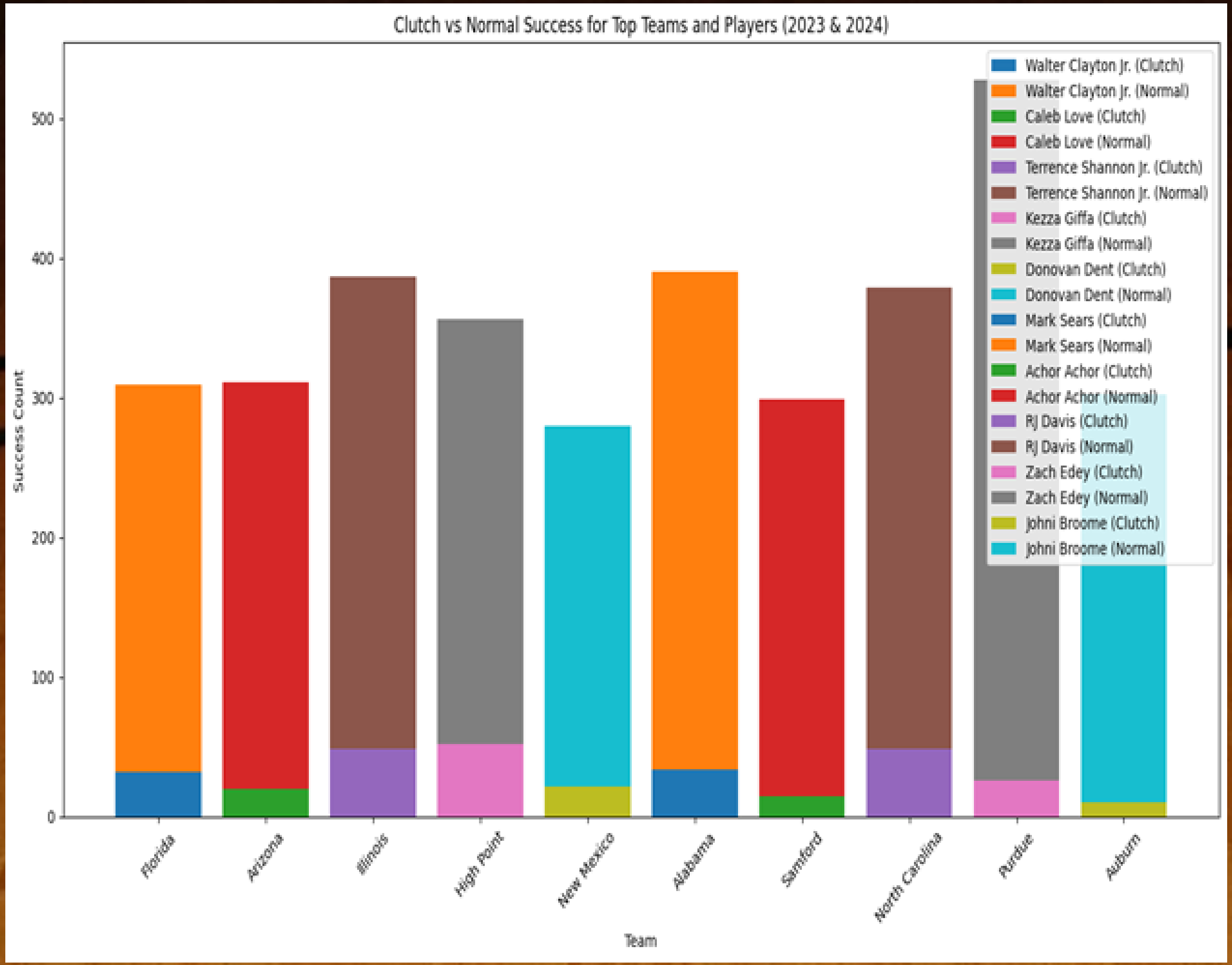
- **Time Constraints**: Limited time meant prioritizing key features and adjustments to quickly boost accuracy.

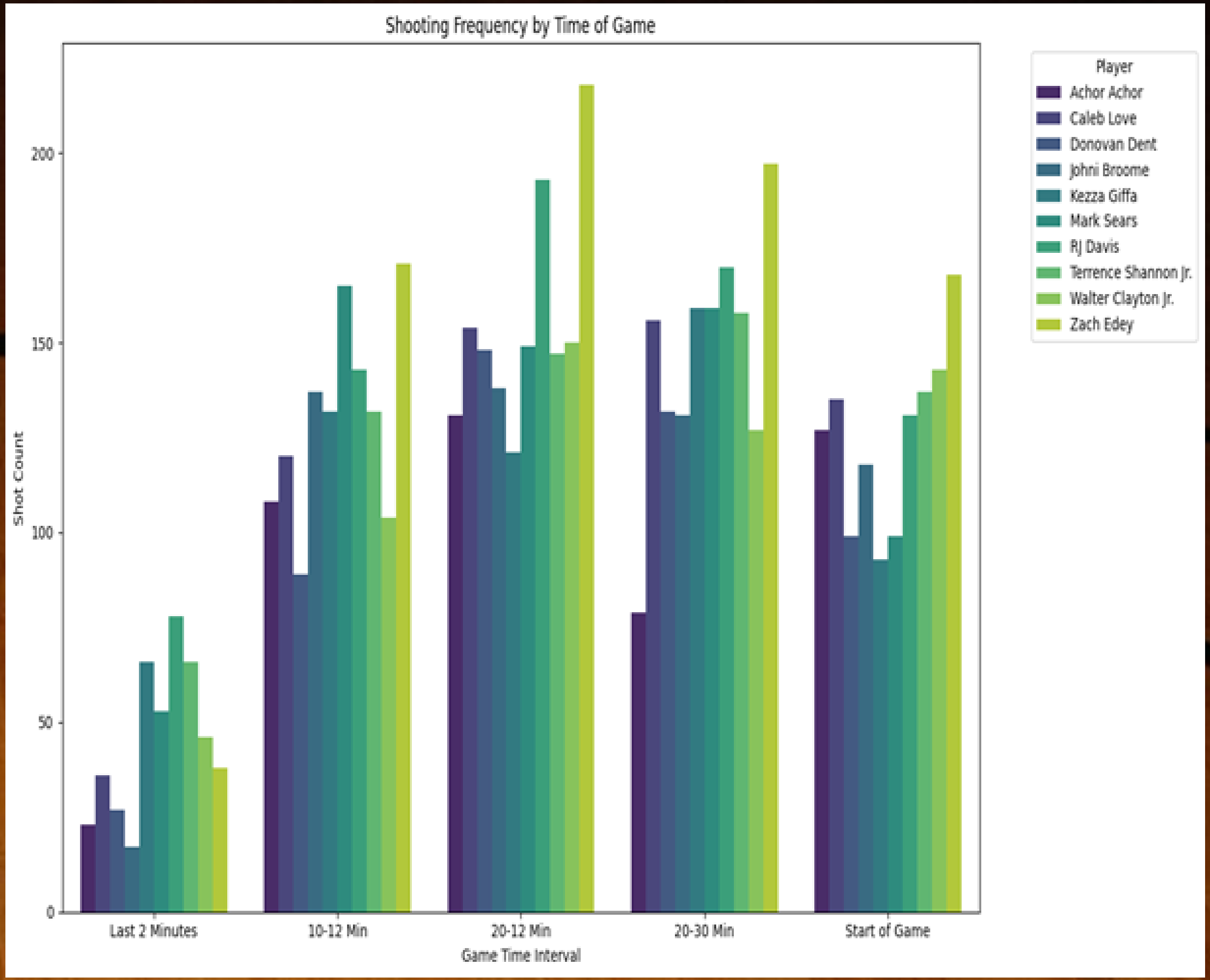
Exploratory Data Analysis (EDA)

Identified key features impacting game outcomes like scoring trends and clutch performance under pressure.






Clutch vs Normal Success for Top Teams and Players (2023 & 2024)
Success Count
0
100
200
300
400
500
Florida
Arizona
Illinois
High Point
New Mexico
Alabama
Samford
North Carolina
Purdue
Auburn
Team
Walter Clayton Jr. (Clutch)
Walter Clayton Jr. (Normal)
Caleb Love (Clutch)
Caleb Love (Normal)
Terrence Shannon Jr. (Clutch)
Terrence Shannon Jr. (Normal)
Kezza Giffa (Clutch)
Kezza Giffa (Normal)
Donovan Dent (Clutch)
Donovan Dent (Normal)
Mark Sears (Clutch)
Mark Sears (Normal)
Achor Achor (Clutch)
Achor Achor (Normal)
RJ Davis (Clutch)
RJ Davis (Normal)
Zach Edey (Clutch)
Zach Edey (Normal)
Johni Broome (Clutch)
Johni Broome (Normal)


Shooting Frequency by Time of Game
Shot Count
0
50
100
150
200
Last 2 Minutes
10-12 Min
20-12 Min
20-30 Min
Start of Game
Game Time Interval
Player
Achor Achor
Caleb Love
Donovan Dent
Johni Broome
Kezza Giffa
Mark Sears
RJ Davis
Terrence Shannon Jr.
Walter Clayton Jr.
Zach Edey

Feature Engineering

- Engineered new features such as rest days, PER(Player Efficiency Rate), Field goal (FG), three-point (3P) percentages and player clutch stats.

```
shot_outcome
made      12841
missed     9265
Name: count, dtype: int64
```

shooter	PER	FG_pct	3P_pct	FT_pct	clutch_pct
	0.5	0.250000	0.250000	0.312500	0.000000
A'lahn Sumler	1.042129	0.456763	0.394678	0.175166	0.692308
A.J. Hoggard	1.07971	0.512077	0.164251	0.256039	0.722222
A.J. Lopez	1.121212	0.506494	0.329004	0.225108	0.500000
A.J. Neal	0.955414	0.401274	0.458599	0.184713	0.571429
...	...	...	...	...	...
Zvonimir Ivisic	1.352113	0.633803	0.225352	0.281690	0.000000
Zy'Nyia White	1.1	0.500000	0.200000	0.400000	0.000000
Zyeir Lawrence	0.5	0.250000	0.000000	0.375000	0.000000
Zyon Pullin	1.253968	0.593254	0.150794	0.341270	0.782609
Zytarious Mortle	0.906103	0.417840	0.272300	0.225352	0.545455

9214 rows × 5 columns

```
temp_filter_columns = ['game_id', 'action_team', 'points', 'PER', 'FG_pct', '3P_pct', 'FT_pct', 'clutch_pct']

temp_df = df.merge(player_stats[['PER', 'FG_pct', '3P_pct', 'FT_pct', 'clutch_pct']], left_on='shooter', right_index=True, how='left')

game_wise_player_stats = temp_df[temp_filter_columns].drop_duplicates().groupby(['game_id', 'action_team']).agg(
            {
                # 'points': ['mean', 'max', 'median'],
                'PER': ['mean', 'max', 'median'],
                'FG_pct': ['mean', 'max', 'median'],
                '3P_pct': ['mean', 'max', 'median'],
                'FT_pct': ['mean', 'max', 'median'],
                'clutch_pct': ['mean', 'max', 'median']}
            ).unstack()
game_wise_player_stats
```

	PER						FG_pct				...	FT_pct				clutch_pct					
	mean		max		median		mean		max		...	max		median		mean		max		median	
action_team	away	home	away	home	away	home	away	home	away	home	...	away	home	away	home	away	home	away	home	away	home
game_id																					
401573353	1.195221	1.278711	1.29703	1.392683	1.247036	1.264706	0.546876	0.586411	0.648515	0.635659	...	0.352273	0.495475	0.230360	0.274011	0.546667	0.647950	1.000000	1.0	0.666667	0.666667
401573354	1.1386	1.099737	1.264113	1.291803	1.163972	1.072727	0.507972	0.501345	0.597403	0.616393	...	0.296000	0.287958	0.258740	0.227273	0.492682	0.450510	0.700000	1.0	0.575758	0.500000
401573355	1.036346	1.14171	1.266667	2.0	1.091514	1.102494	0.473713	0.533018	0.581114	1.000000	...	0.320988	0.426471	0.180157	0.241558	0.481137	0.568885	1.000000	1.0	0.500000	0.571429
401573356	1.192284	1.173806	1.360248	1.266564	1.176082	1.168116	0.544934	0.532174	0.638629	0.607088	...	0.445455	0.322034	0.223602	0.268139	0.607907	0.618266	1.000000	1.0	0.652174	0.666667
401573357	1.118399	1.111051	1.318182	1.188713	1.097902	1.09434	0.520815	0.521215	0.659091	0.584416	...	0.419192	0.367876	0.341121	0.311538	0.448413	0.497199	0.714286	1.0	0.611111	0.428571
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
401634067	1.134574	1.126772	1.246231	1.242105	1.125278	1.123934	0.514723	0.522940	0.621429	0.621053	...	0.385714	0.442105	0.217647	0.249092	0.367965	0.568154	0.666667	1.0	0.500000	0.598039
401634068	1.095032	1.12121	1.365854	1.229665	1.094488	1.135135	0.497969	0.499882	0.682927	0.567568	...	0.328358	0.513514	0.191950	0.193548	0.408403	0.522511	0.666667	1.0	0.500000	0.545455
401634069	1.180533	1.148721	1.326409	1.308642	1.173585	1.127371	0.521537	0.496145	0.554896	0.641638	...	0.303704	0.341297	0.253937	0.193703	0.431046	0.470769	0.666667	1.0	0.358824	0.500000
401634070	1.17603	1.118986	1.473684	1.246612	1.170732	1.100917	0.547030	0.514873	0.625000	0.623306	...	0.318949	0.337302	0.252049	0.244240	0.540882	0.460823	0.818182	1.0	0.565217	0.500000
401636059	1.128188	1.122508	1.290102	1.186813	1.127371	1.135501	0.504514	0.509209	0.641638	0.583127	...	0.352941	0.317618	0.205962	0.199666	0.464719	0.542929	0.857143	1.0	0.500000	0.595960

6001 rows × 30 columns

```python
games_and_dates = df[['game_id', 'date', 'home', 'away']].drop_duplicates()
games_and_dates["date"] = pd.to_datetime(games_and_dates["date"])
# combine the home and away teams into a single column, but two rows per game
games_and_dates = pd.melt(games_and_dates, id_vars=['game_id', 'date'], value_vars=['ho
                          var_name='home_away', value_name='team')
games_and_dates.drop(columns=['home_away'], inplace=True)
games_and_dates.sort_values(by=['team', 'date'], inplace=True)
games_and_dates['rest_days'] = games_and_dates.groupby('team')['date'].diff().dt.days -
games_and_dates
```

✓ 0.7s

	game_id	date	team	rest_days
11465	401600143	2023-12-30	ANTELOPE	NaN
11872	401592097	2023-12-10	AR-Fort Smith	NaN
6377	401594537	2023-11-06	AR-Pine Bluff	NaN
377	401604754	2023-11-09	AR-Pine Bluff	2.0
378	401611838	2023-11-11	AR-Pine Bluff	1.0
...	...	...	...	...
7428	401587744	2024-02-17	Youngstown St	2.0
10008	401587749	2024-02-23	Youngstown St	5.0
8734	401587754	2024-02-25	Youngstown St	1.0
1903	401587756	2024-02-28	Youngstown St	2.0
5980	401625696	2024-03-07	Youngstown St	7.0

```python
# Merge the dataframes
home_wins = game_wins.merge(home_games[["game_id", "home"]].drop_duplicates(), on='game_id', how='left')

# Calculate total games played by each team at home
total_home_games = home_wins.groupby('home')['game_id'].count()

# Calculate total wins by each team at home
home_wins = home_wins.groupby('home')['winner'].sum()

# Calculate win percentage
home_win_percentage = (home_wins / total_home_games) * 100
home_win_percentage
```

✓ 0.1s

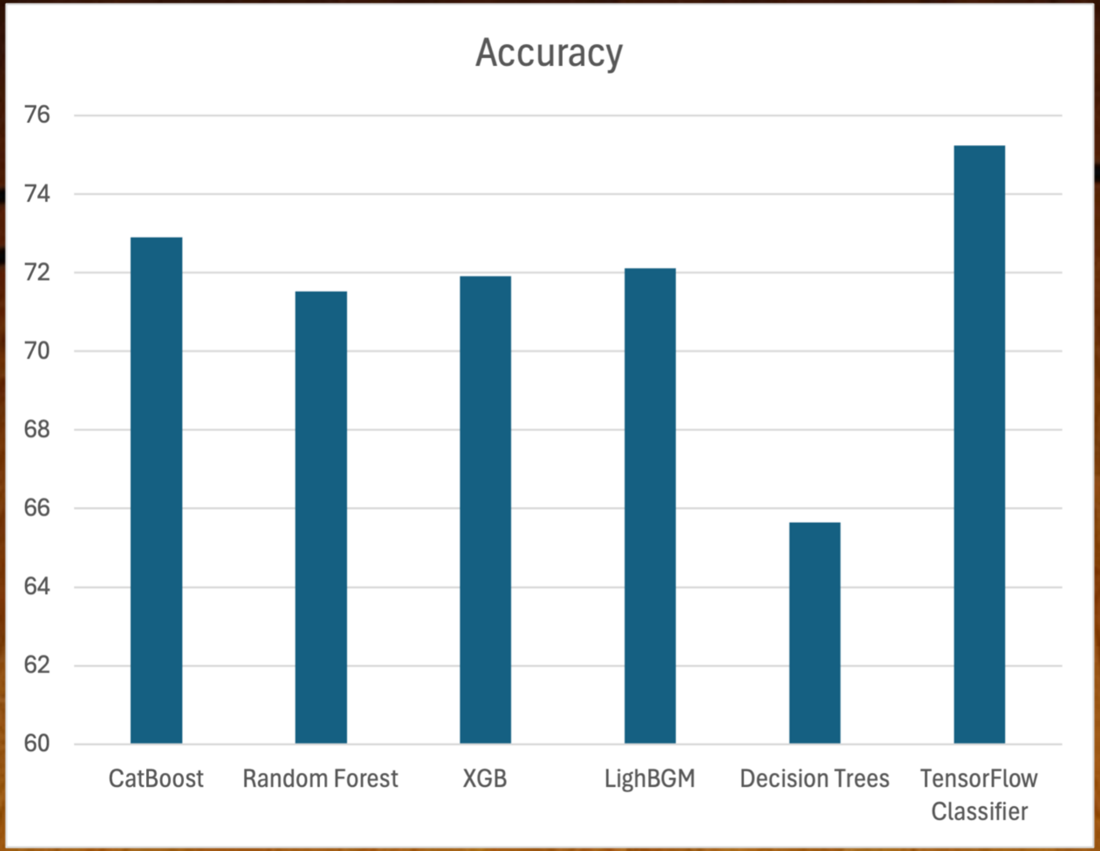
```
home
AR-Pine Bluff     61.538462
Abilene Chrstn    57.142857
Air Force         25.000000
Akron             78.947368
Alabama           83.333333
                    ...
Wright St         52.941176
Wyoming           58.823529
Xavier            57.894737
Yale              81.818182
Youngstown St     80.000000
Length: 364, dtype: float64
```

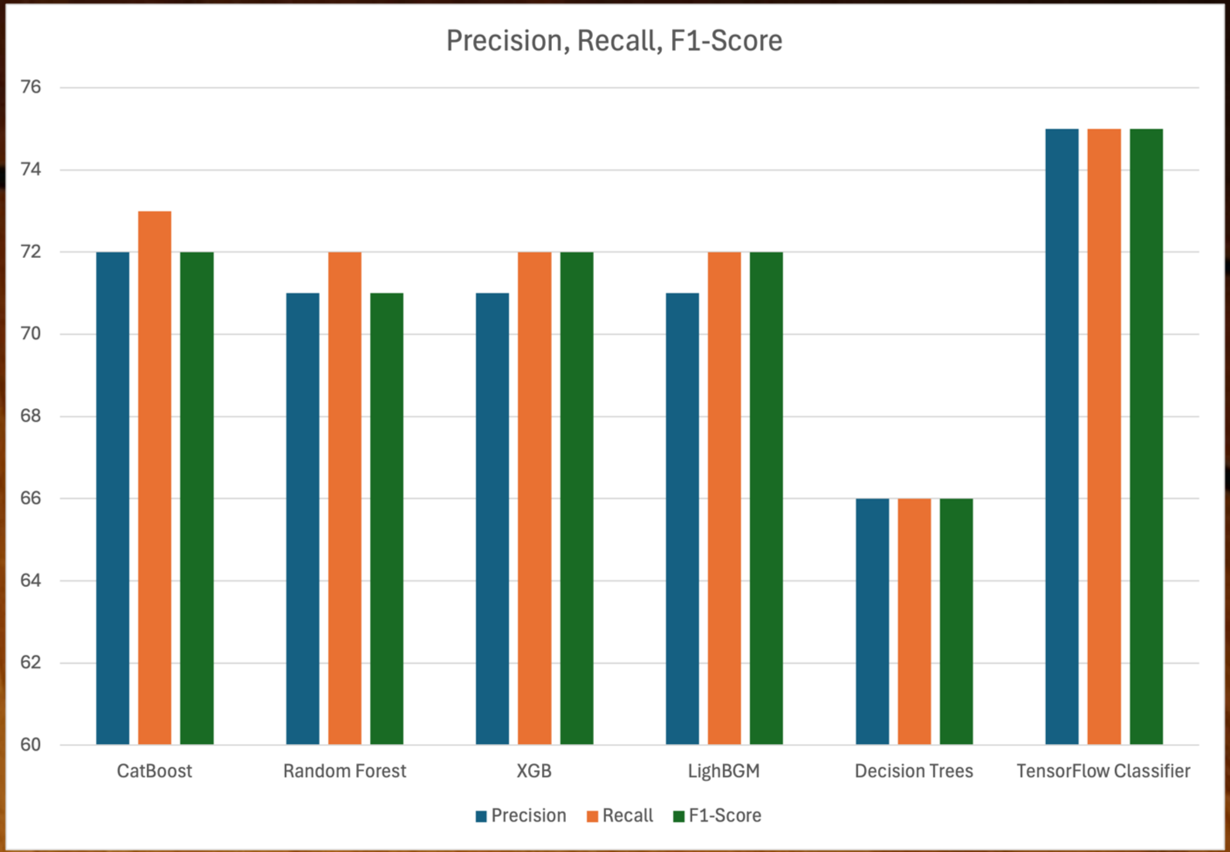
Modelling

- Method-1: The first iteration of the model does not incorporate game situations or dynamic variables such as scores or in-game events. It is designed to make predictions based solely on pre-game factors and static team/player metrics.

- Method-2: The enhanced model includes all relevant details, integrating dynamic game situations such as current scores, time remaining, and possession data. This allows for more contextual and accurate predictions based on real-time game scenarios.

Method -1: Used 6 different models for prediction and chose the one with the best accuracy.





Method-1

```
# Predict on the testing data
y_pred = catboost_classifier.predict(X_test)

# Evaluate the model
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.7289628180039139
Classification Report:
              precision    recall  f1-score   support

         0.0       0.65      0.55      0.60       372
         1.0       0.76      0.83      0.80       650

    accuracy                           0.73      1022
   macro avg       0.71      0.69      0.70      1022
weighted avg       0.72      0.73      0.72      1022

Confusion Matrix:
[[204 168]
 [109 541]]
```

```
Accuracy: 0.7152641878669276
RandomForest Classification Report:
              precision    recall  f1-score   support

         0.0       0.64      0.50      0.56       372
         1.0       0.75      0.84      0.79       650

    accuracy                           0.72      1022
   macro avg       0.69      0.67      0.68      1022
weighted avg       0.71      0.72      0.71      1022

Confusion Matrix:
[[187 185]
 [106 544]]
```

Method-1

Accuracy: 0.7191780821917808

XGB Classification Report:

	precision	recall	f1-score	support
0.0	0.63	0.56	0.59	372
1.0	0.76	0.81	0.79	650
accuracy			0.72	1022
macro avg	0.70	0.69	0.69	1022
weighted avg	0.71	0.72	0.72	1022

Confusion Matrix:

```
[[210 162]
 [125 525]]
```

Accuracy: 0.7211350293542075

lightbgm Classification Report:

	precision	recall	f1-score	support
0.0	0.64	0.55	0.59	372
1.0	0.76	0.82	0.79	650
accuracy			0.72	1022
macro avg	0.70	0.68	0.69	1022
weighted avg	0.71	0.72	0.72	1022

Confusion Matrix:

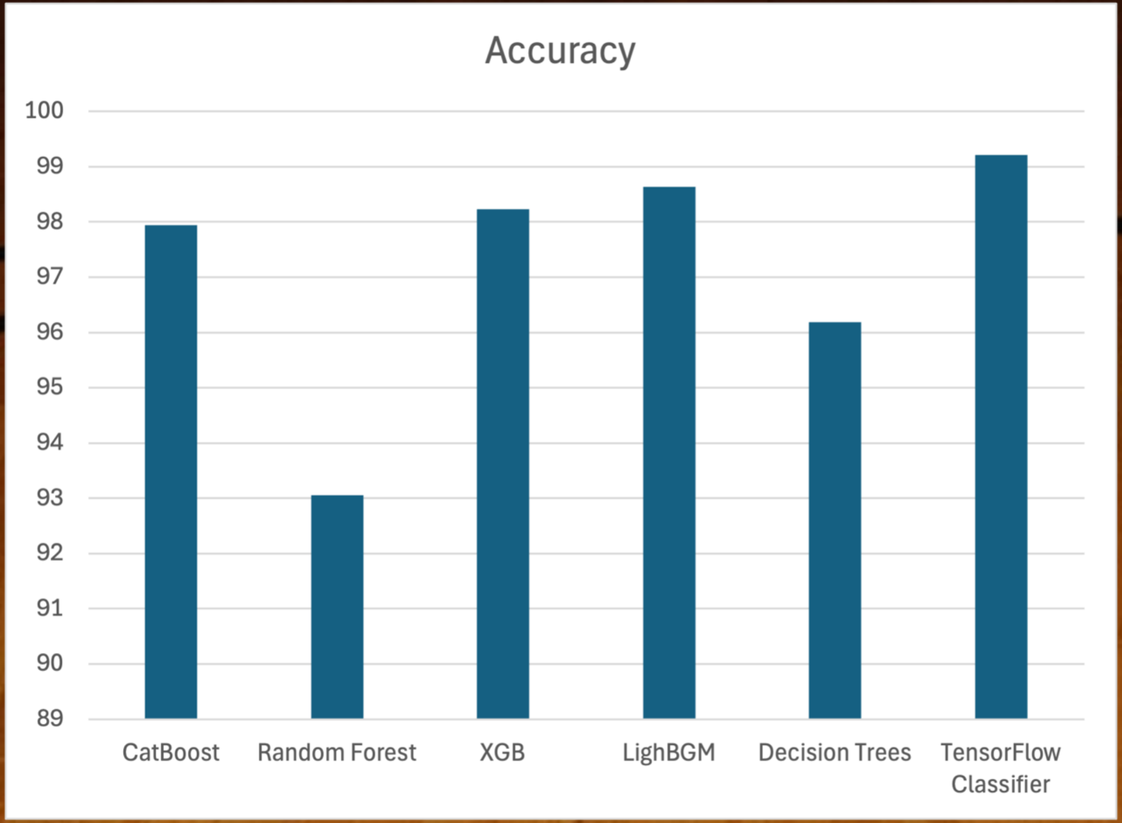
```
[[204 168]
 [117 533]]
```

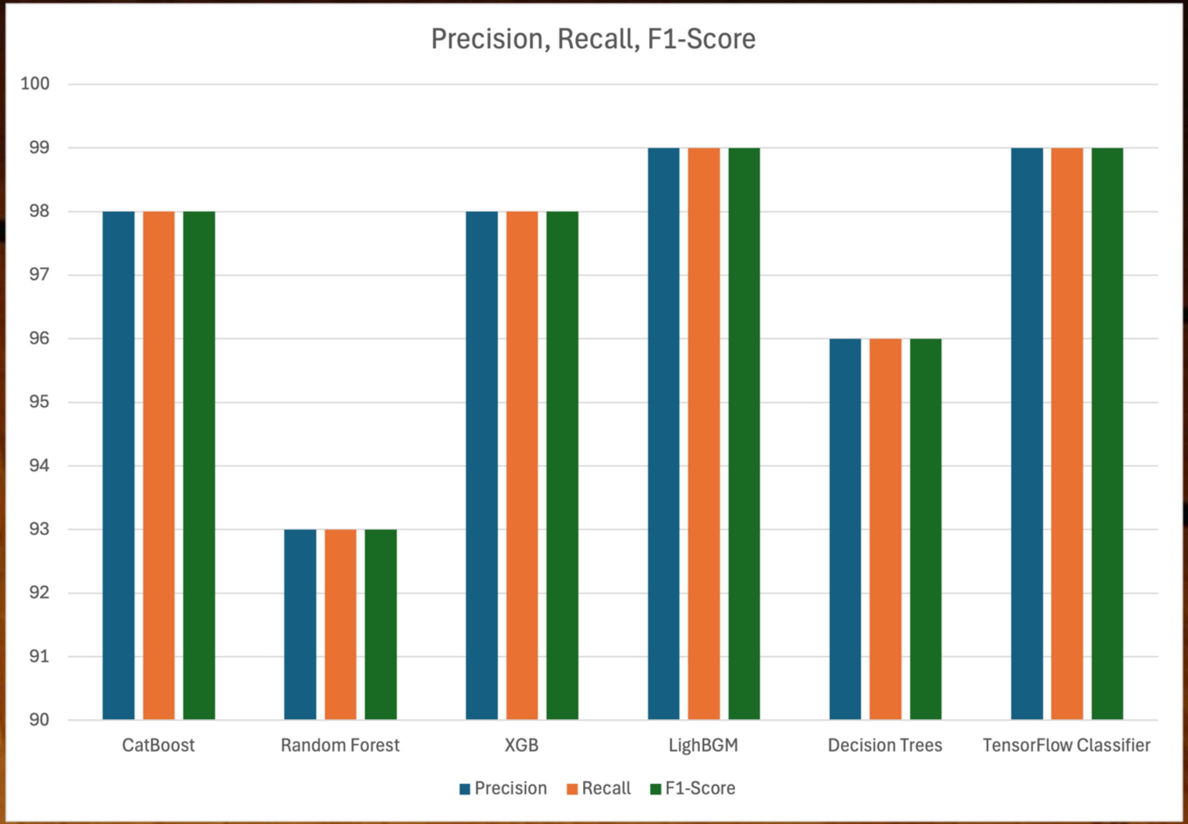
Method-1

```
Accuracy: 0.6565557729941291
DecisionTree Classification Report:
              precision    recall  f1-score   support

         0.0       0.53      0.53      0.53       372
         1.0       0.73      0.73      0.73       650

    accuracy                           0.66      1022
   macro avg       0.63      0.63      0.63      1022
weighted avg       0.66      0.66      0.66      1022

Confusion Matrix:
[[199 173]
 [178 472]]
```

```
Accuracy: 0.7524461839530333
Tensorflow Classification Report:
              precision    recall  f1-score   support

         0.0       0.69      0.57      0.63       372
         1.0       0.78      0.86      0.81       650

    accuracy                           0.75      1022
   macro avg       0.74      0.71      0.72      1022
weighted avg       0.75      0.75      0.75      1022

Confusion Matrix:
[[213 159]
 [ 94 556]]
```

Method -2: Used the same 6 models for prediction and chose the one with the best accuracy.



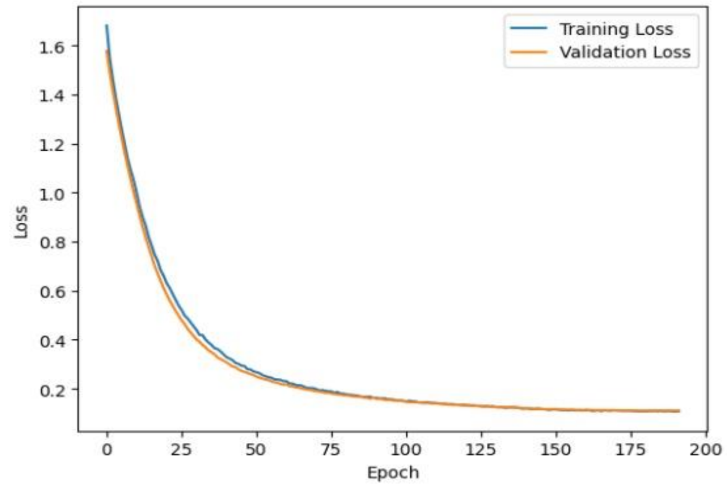


Method-2

```
# Fit the model on the training data
catboost_classifier.fit(X_train, y_train)

# Predict on the testing data
y_pred = catboost_classifier.predict(X_test)

# Evaluate the model
print(f'Accuracy: {accuracy_score(y_test, y_pred)}')
print('Classification Report:')
print(classification_report(y_test, y_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9794520547945206
Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.97      0.97       372
         1.0       0.98      0.99      0.98       650

    accuracy                           0.98      1022
   macro avg       0.98      0.98      0.98      1022
weighted avg       0.98      0.98      0.98      1022

Confusion Matrix:
[[359  13]
 [  8 642]]
```

```
Accuracy: 0.9921722113502935
Tensorflow Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      0.98      0.99       372
         1.0       0.99      1.00      0.99       650

    accuracy                           0.99      1022
   macro avg       0.99      0.99      0.99      1022
weighted avg       0.99      0.99      0.99      1022

Confusion Matrix:
[[366   6]
 [  2 648]]
```



Method-2

```
Accuracy: 0.9863013698630136
lightbgm Classification Report:
              precision    recall  f1-score   support

         0.0       0.99      0.97      0.98       372
         1.0       0.98      0.99      0.99       650

    accuracy                           0.99      1022
   macro avg       0.99      0.98      0.99      1022
weighted avg       0.99      0.99      0.99      1022

Confusion Matrix:
[[362  10]
 [  4 646]]
```

```
Accuracy: 0.961839530332681
DecisionTree Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.95      0.95       372
         1.0       0.97      0.97      0.97       650

    accuracy                           0.96      1022
   macro avg       0.96      0.96      0.96      1022
weighted avg       0.96      0.96      0.96      1022

Confusion Matrix:
[[354  18]
 [ 21 629]]
```

Method-2

```
Accuracy: 0.9305283757338552
RandomForest Classification Report:
              precision    recall  f1-score   support

         0.0       0.95      0.85      0.90       372
         1.0       0.92      0.98      0.95       650

    accuracy                           0.93      1022
   macro avg       0.94      0.91      0.92      1022
weighted avg       0.93      0.93      0.93      1022

Confusion Matrix:
[[317  55]
 [ 16 634]]
```

```
Accuracy: 0.9823874755381604
xgboost Classification Report:
              precision    recall  f1-score   support

         0.0       0.98      0.97      0.98       372
         1.0       0.98      0.99      0.99       650

    accuracy                           0.98      1022
   macro avg       0.98      0.98      0.98      1022
weighted avg       0.98      0.98      0.98      1022

Confusion Matrix:
[[362  10]
 [  8 642]]
```

Interactive Interface

- Used "Gradio UI Design" to create a user-friendly interface that simplifies input and output for the user.

```python
import gradio as gr

home_team = gr.Dropdown(list(unique_home_teams), label="Home Team")
away_team = gr.Dropdown(list(unique_away_teams), label="Away Team")
home_players = gr.Dropdown(list(unique_players), label="Home Players", multiselect=True, max_choices=7)
away_players = gr.Dropdown(list(unique_players), label="Away Players", multiselect=True, max_choices=7)
home_rest_days = gr.Number(label="Home Team Rest Days")
away_rest_days = gr.Number(label="Away Team Rest Days")

output_text = gr.Textbox(label="Prediction")

def predict_winner_and_mvp(home_team, away_team, home_players, away_players, home_rest_days, away_rest_days):
    # create a dict with the features
    data_dict = dict()
    data_dict["home_rest_days"] = home_rest_days
    data_dict["away_rest_days"] = away_rest_days
    data_dict["home_team_home_rating"] = home_team_performance[home_team]
    data_dict["away_team_away_rating"] = away_team_performance[away_team]
    data_dict["home_team_rating"] = team_performance[home_team]
    data_dict["away_team_rating"] = team_performance[away_team]
    data_dict["home_team_home_win_percentage"] = home_win_percentage[home_team]
    data_dict["away_team_away_win_percentage"] = away_win_percentage[away_team]
    data_dict["home_team_win_percentage"] = team_win_percentage[home_team]
    data_dict["away_team_win_percentage"] = team_win_percentage[away_team]
    data_dict["home_team_home_score_mean"] = home_team_scores.loc[home_team, 'home_team_home_score_mean']
    data_dict["home_team_home_score_median"] = home_team_scores.loc[home_team, 'home_team_home_score_median']
    data_dict["away_team_away_score_mean"] = away_team_scores.loc[away_team, 'away_team_away_score_mean']
    data_dict["away_team_away_score_median"] = away_team_scores.loc[away_team, 'away_team_away_score_median']
    data_dict["home_team_score_mean"] = team_scores.loc[home_team, 'mean_points']
    data_dict["home_team_score_median"] = team_scores.loc[home_team, 'median_points']
    data_dict["away_team_score_mean"] = team_scores.loc[away_team, 'mean_points']
    data_dict["away_team_score_median"] = team_scores.loc[away_team, 'median_points']
    data_dict["PER_mean_away"] = player_stats.loc[away_players, 'PER'].mean()
    data_dict["PER_mean_home"] = player_stats.loc[home_players, 'PER'].mean()
    data_dict["PER_max_away"] = player_stats.loc[away_players, 'PER'].max()
    data_dict["PER_max_home"] = player_stats.loc[home_players, 'PER'].max()
    data_dict["PER_median_away"] = player_stats.loc[away_players, 'PER'].median()
    data_dict["PER_median_home"] = player_stats.loc[home_players, 'PER'].median()
    data_dict["FG_pct_mean_away"] = player_stats.loc[away_players, 'FG_pct'].mean()
    data_dict["FG_pct_mean_home"] = player_stats.loc[home_players, 'FG_pct'].mean()
    data_dict["FG_pct_max_away"] = player_stats.loc[away_players, 'FG_pct'].max()
    data_dict["FG_pct_max_home"] = player_stats.loc[home_players, 'FG_pct'].max()
    data_dict["FG_pct_median_away"] = player_stats.loc[away_players, 'FG_pct'].median()
    data_dict["FG_pct_median_home"] = player_stats.loc[home_players, 'FG_pct'].median()
    data_dict["3P_pct_mean_away"] = player_stats.loc[away_players, '3P_pct'].mean()
    data_dict["3P_pct_mean_home"] = player_stats.loc[home_players, '3P_pct'].mean()
    data_dict["3P_pct_max_away"] = player_stats.loc[away_players, '3P_pct'].max()
    data_dict["3P_pct_max_home"] = player_stats.loc[home_players, '3P_pct'].max()
    data_dict["3P_pct_median_away"] = player_stats.loc[away_players, '3P_pct'].median()
    data_dict["3P_pct_median_home"] = player_stats.loc[home_players, '3P_pct'].median()
    data_dict["FT_pct_mean_away"] = player_stats.loc[away_players, 'FT_pct'].mean()
    data_dict["FT_pct_mean_home"] = player_stats.loc[home_players, 'FT_pct'].mean()
```

```python
    # Create a DataFrame from the data
    data_df = pd.DataFrame(data_dict, index=[0])

    # Rearrange the columns to match the model's input order
    data_df = data_df[X.columns]

    # Predict the winning team and MVP
    results = predict_winning_team_and_player_sklearn(catboost_classifier, data_df, home_team, away_team, home_players, away_players)

    # Convert the results to a string
    output = ['\n'.join([f'{result[0]} wins with a probability of {result[1]:.2f} and the MVP is {result[2]}' for result in results])]

    return output[0]
```
```
[250] ✓ 0.0s
```

```python
predict_winner_and_mvp(unique_home_teams[100], unique_away_teams[35],
                       unique_players[1000:1006],
                       unique_players[70:79], 0, 0)
```
```
[251] ✓ 0.0s
... 'Maine wins with a probability of 0.99 and the MVP is Brandon Rush'
```

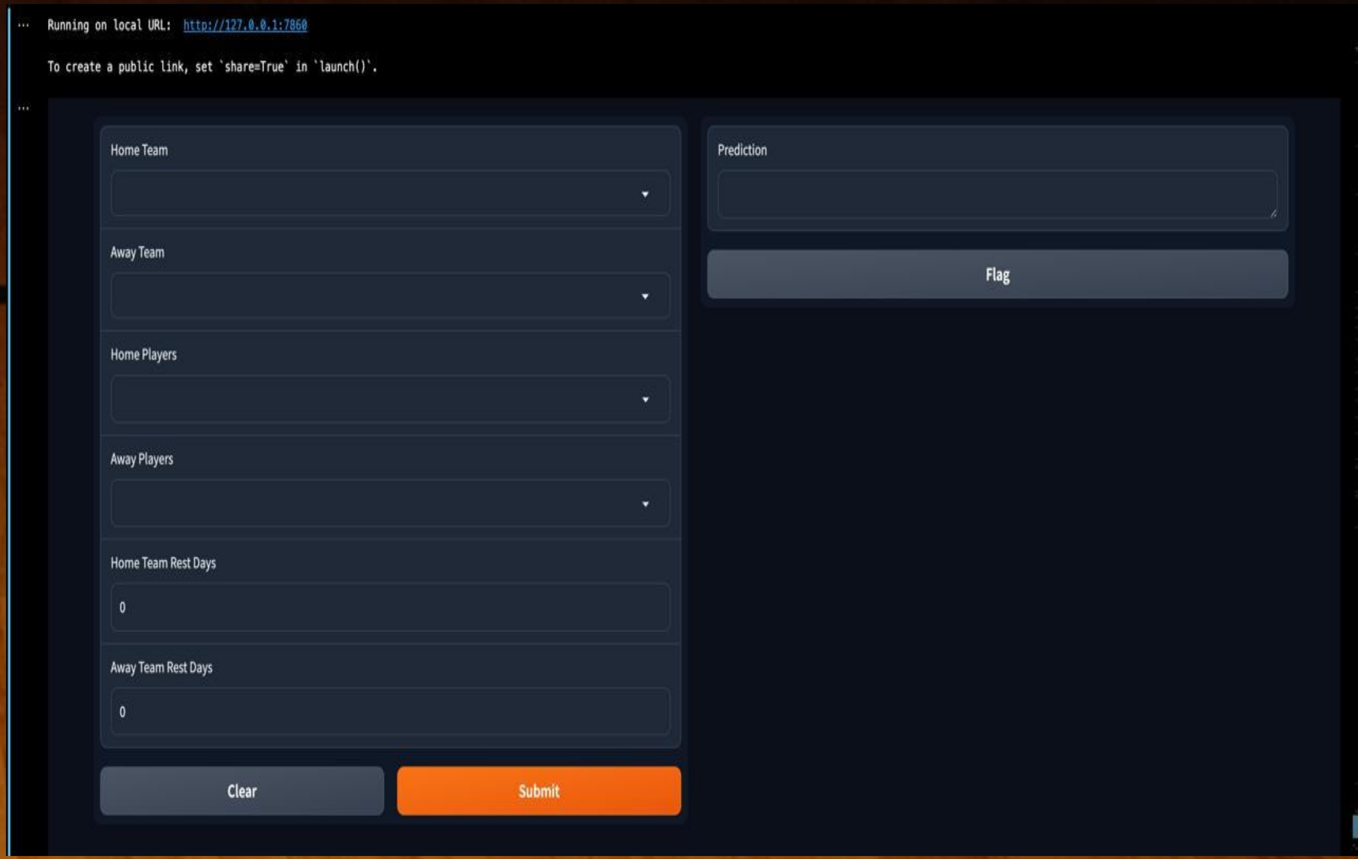
```python
unique_home_teams[100], unique_away_teams[35]
```
```
[252] ✓ 0.0s
... ('Maine', 'N Colorado')
```

+ Code + Markdown

```python
# Launch the web app
import os
os.environ['GRADIO_NODE_PATH'] = '/usr/local/bin/node'
iface = gr.Interface(
            predict_winner_and_mvp,
            [home_team, away_team, home_players, away_players, home_rest_days, away_rest_days],
            outputs=output_text)
iface.launch(debug=True, inline=True, height=1000)
```
```
[253] 66m 45.0s
... Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

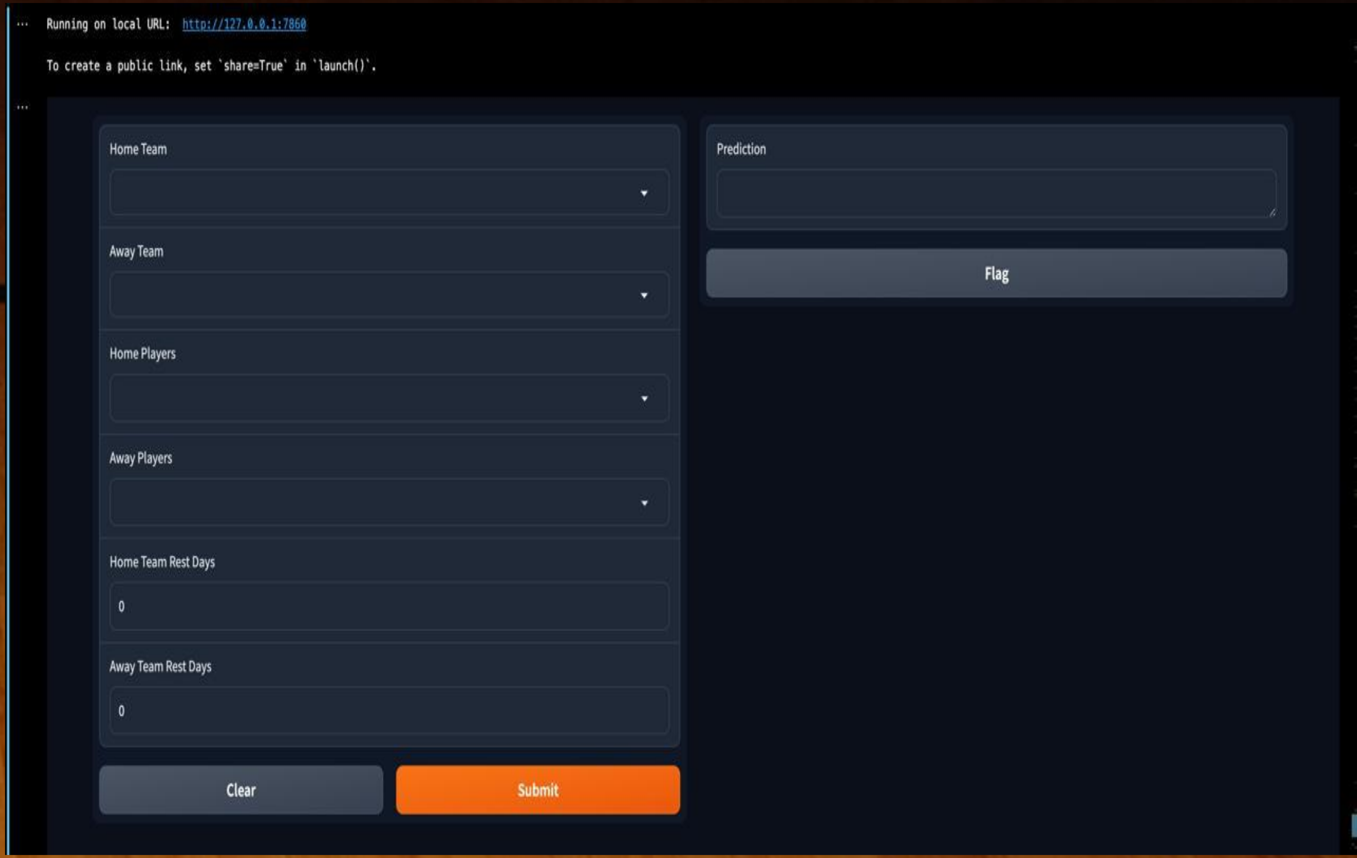
Interactive Interface

- **User Inputs:**
- Select home and away teams.
- Choose 7 players per team from dropdowns and also include Home & Away Rest Days for better prediction.



Interactive Interface

- **Prediction Outputs:**
- Winning team.
- Probability of winning.
- MVP responsible for the winning shot.



Prediction Logic

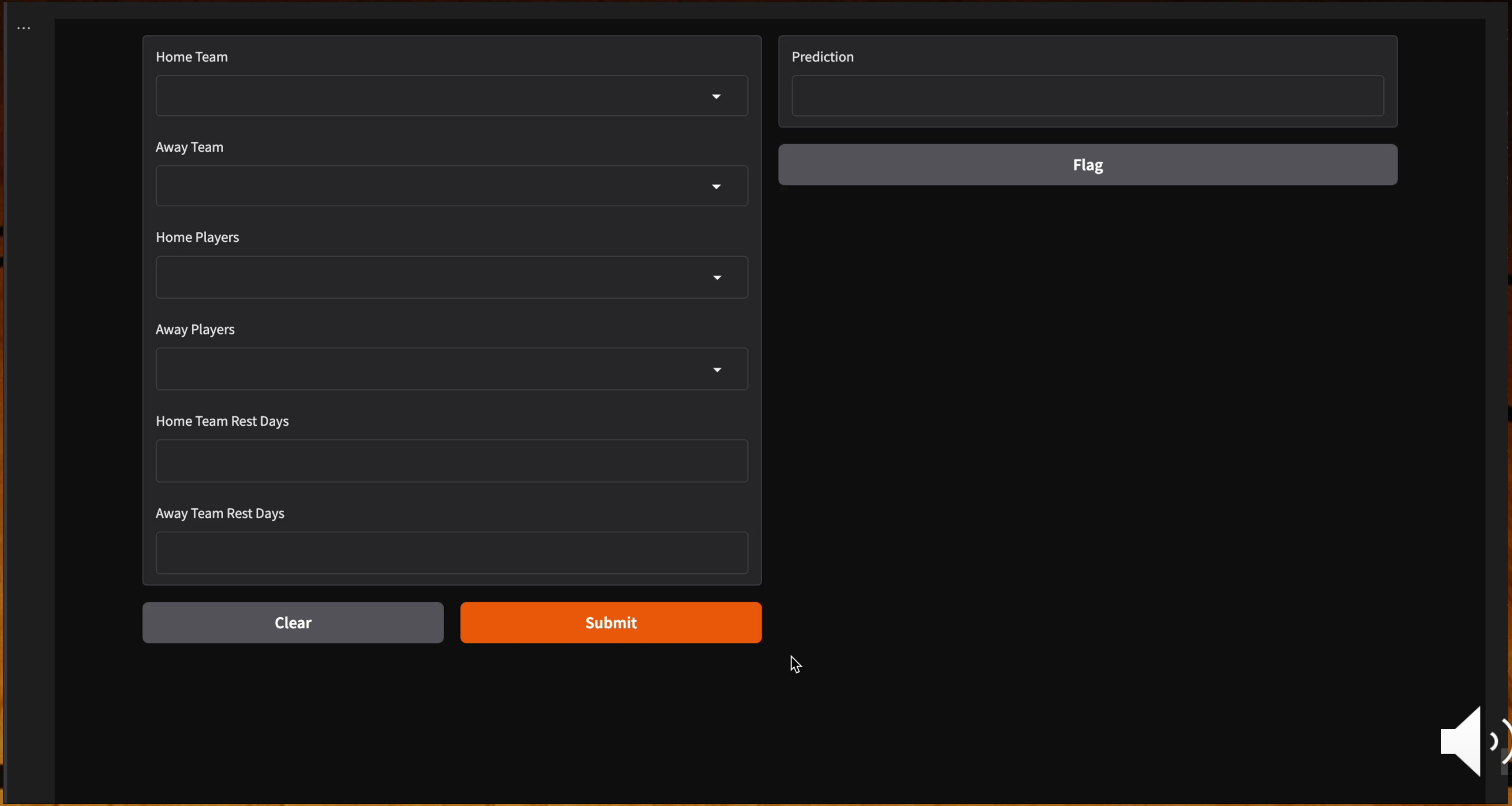
Winning Team Prediction:

- The neural network determines the likelihood of home or away victory based on input features.

Player Selection:

- Weighted scoring formula: ***Score = (clutch_pct × 0.4) + (PER × 0.6)***
- The player with the highest score from the predicted winning team is identified as the ***MVP***.

Demonstration



Results & Insights

Use Cases:

- **For Coaches:** Strategic decision-making for critical game moments.
- **For Analysts:** Advanced player performance analytics.
- **For Fans:** Enhanced engagement through predictive insights.

Future Scope

Enhancements

- Improve model accuracy by incorporating real-time data streams.
- Extend predictions to other aspects like defensive plays.
- Add deeper player-level metrics (e.g., fatigue, injury probability).

Future Scope

Conclusion

- Successfully integrated data science and sports analytics.
- Intuitive tool for predicting game-changing moments.
- Open for questions and feedback.

For Questions & Feedback :

skamired@syr.edu

cchintha@syr.edu

kmeesala@syr.edu

iagrawal@syr.edu

Thank you!