

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.

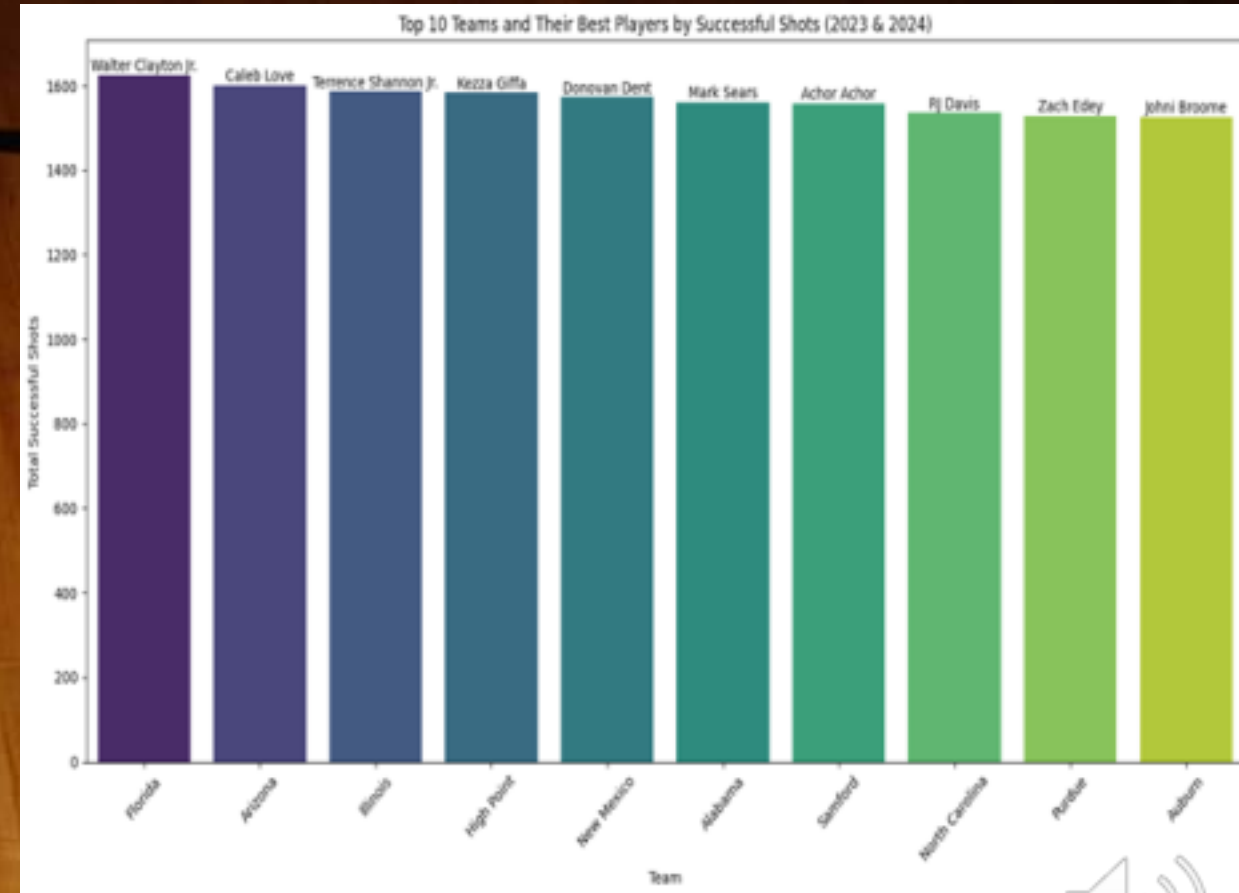
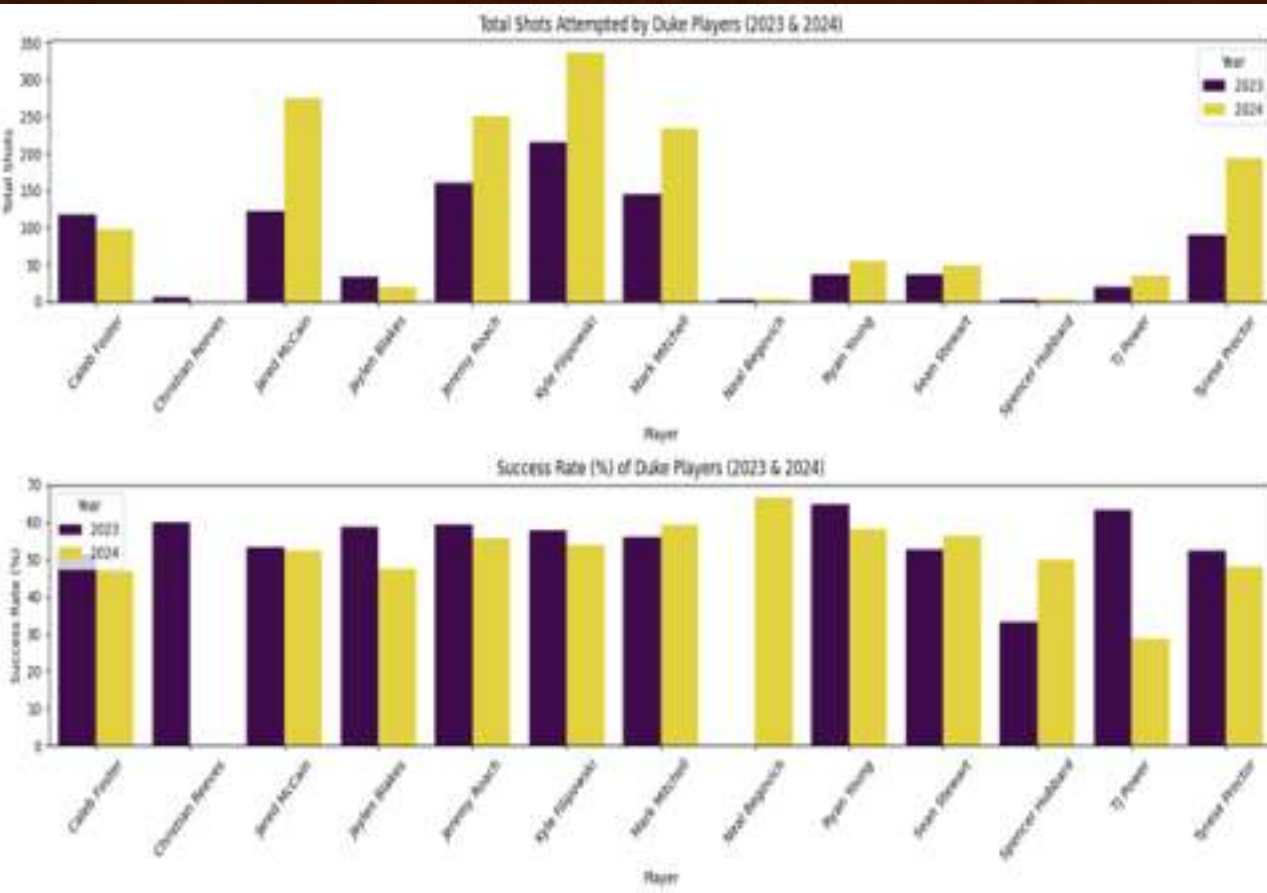
Challenges:

- **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.

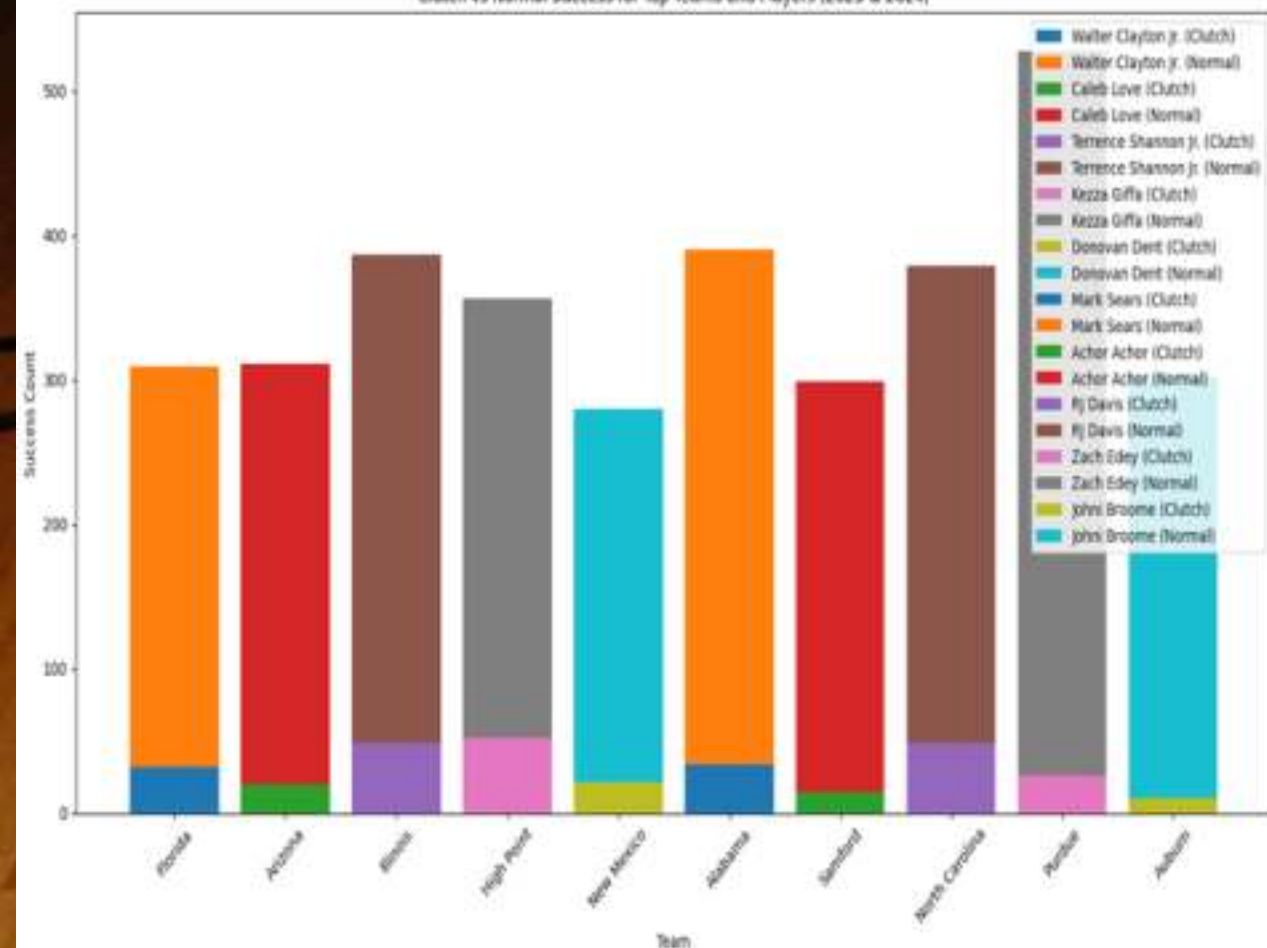


Exploratory Data Analysis (EDA)

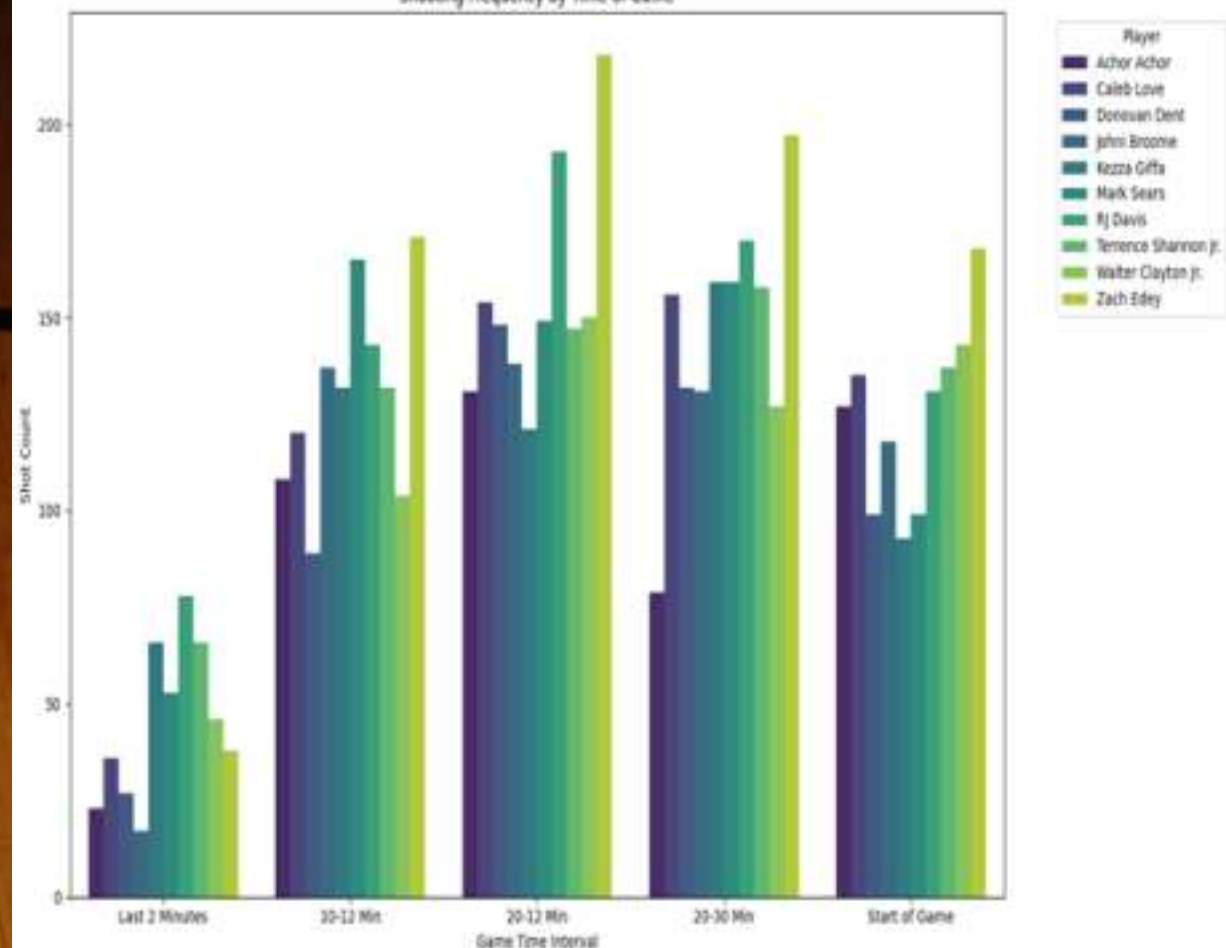
Identified key features impacting game outcomes like scoring trends and clutch performance under pressure.



Clutch vs Normal Success for Top Teams and Players (2023 & 2024)



Shooting Frequency by Time of Game



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      12641
missed    9265
Name: count, dtype: int64
```

	PER	FG_pct	3P_pct	FT_pct	clutch_pct
shooter					
	0.5	0.250000	0.250000	0.312500	0.000000
A'lahn Sumler	1.042129	0.456763	0.394578	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'Nyla White	1.1	0.500000	0.200000	0.400000	0.000000
Zyler Lawrence	0.5	0.250000	0.000000	0.375000	0.000000
Zyon Pullin	1.253968	0.593254	0.150794	0.341270	0.782609
Zyriarous Montle	0.906103	0.417840	0.272300	0.225352	0.545455

9214 rows x 5 columns

```
temp_F1010r_columns = ['game_id', 'shot_outcome', 'seconds', 'PER', 'FG_pct', '3P_pct', 'FT_pct', 'clutch_pct']
temp_F1 = df.melt(id_vars=['game_id', 'seconds', 'PER', 'FG_pct', '3P_pct', 'FT_pct', 'clutch_pct'], var_name='shooter', value_name='value')
game_shot_stats = temp_F1[temp_F1.columns[0:6]].groupby(['game_id', 'shot_outcome']).agg(
    # 'seconds' ['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'])
game_shot_stats
```

```

# 3.5

```

	PER	FG_pct	3P_pct	FT_pct	clutch_pct
	mean	max	median	mean	max
game_id	mean	max	median	mean	max
40117333	1.19521	1.21071	1.2073	1.00000	1.04738
40173254	1.1396	1.09727	1.20412	1.00000	1.00000
40173333	1.09346	1.14171	1.09667	1.0	1.08714
40173334	1.10284	1.17308	1.00246	1.00000	1.00000
40173335	1.11899	1.11185	1.00752	1.00000	1.00000
40193007	1.13457	1.12072	1.04037	1.00000	1.00000
40193008	1.09032	1.10121	1.00000	1.00000	1.00000
40193009	1.10000	1.10000	1.00000	1.00000	1.00000
40193010	1.10000	1.10000	1.00000	1.00000	1.00000
40193011	1.10000	1.10000	1.00000	1.00000	1.00000
40193012	1.10000	1.10000	1.00000	1.00000	1.00000
40193013	1.10000	1.10000	1.00000	1.00000	1.00000
40193014	1.10000	1.10000	1.00000	1.00000	1.00000
40193015	1.10000	1.10000	1.00000	1.00000	1.00000
40193016	1.10000	1.10000	1.00000	1.00000	1.00000
40193017	1.10000	1.10000	1.00000	1.00000	1.00000
40193018	1.10000	1.10000	1.00000	1.00000	1.00000
40193019	1.10000	1.10000	1.00000	1.00000	1.00000
40193020	1.10000	1.10000	1.00000	1.00000	1.00000
40193021	1.10000	1.10000	1.00000	1.00000	1.00000
40193022	1.10000	1.10000	1.00000	1.00000	1.00000
40193023	1.10000	1.10000	1.00000	1.00000	1.00000
40193024	1.10000	1.10000	1.00000	1.00000	1.00000
40193025	1.10000	1.10000	1.00000	1.00000	1.00000
40193026	1.10000	1.10000	1.00000	1.00000	1.00000
40193027	1.10000	1.10000	1.00000	1.00000	1.00000
40193028	1.10000	1.10000	1.00000	1.00000	1.00000
40193029	1.10000	1.10000	1.00000	1.00000	1.00000
40193030	1.10000	1.10000	1.00000	1.00000	1.00000
40193031	1.10000	1.10000	1.00000	1.00000	1.00000
40193032	1.10000	1.10000	1.00000	1.00000	1.00000
40193033	1.10000	1.10000	1.00000	1.00000	1.00000
40193034	1.10000	1.10000	1.00000	1.00000	1.00000
40193035	1.10000	1.10000	1.00000	1.00000	1.00000
40193036	1.10000	1.10000	1.00000	1.00000	1.00000
40193037	1.10000	1.10000	1.00000	1.00000	1.00000
40193038	1.10000	1.10000	1.00000	1.00000	1.00000
40193039	1.10000	1.10000	1.00000	1.00000	1.00000
40193040	1.10000	1.10000	1.00000	1.00000	1.00000
40193041	1.10000	1.10000	1.00000	1.00000	1.00000
40193042	1.10000	1.10000	1.00000	1.00000	1.00000
40193043	1.10000	1.10000	1.00000	1.00000	1.00000
40193044	1.10000	1.10000	1.00000	1.00000	1.00000
40193045	1.10000	1.10000	1.00000	1.00000	1.00000
40193046	1.10000	1.10000	1.00000	1.00000	1.00000
40193047	1.10000	1.10000	1.00000	1.00000	1.00000
40193048	1.10000	1.10000	1.00000	1.00000	1.00000
40193049	1.10000	1.10000	1.00000	1.00000	1.00000
40193050	1.10000	1.10000	1.00000	1.00000	1.00000
40193051	1.10000	1.10000	1.00000	1.00000	1.00000
40193052	1.10000	1.10000	1.00000	1.00000	1.00000
40193053	1.10000	1.10000	1.00000	1.00000	1.00000
40193054	1.10000	1.10000	1.00000	1.00000	1.00000
40193055	1.10000	1.10000	1.00000	1.00000	1.00000
40193056	1.10000	1.10000	1.00000	1.00000	1.00000
40193057	1.10000	1.10000	1.00000	1.00000	1.00000
40193058	1.10000	1.10000	1.00000	1.00000	1.00000
40193059	1.10000	1.10000	1.00000	1.00000	1.00000
40193060	1.10000	1.10000	1.00000	1.00000	1.00000
40193061	1.10000	1.10000	1.00000	1.00000	1.00000
40193062	1.10000	1.10000	1.00000	1.00000	1.00000
40193063	1.10000	1.10000	1.00000	1.00000	1.00000
40193064	1.10000	1.10000	1.00000	1.00000	1.00000
40193065	1.10000	1.10000	1.00000	1.00000	1.00000
40193066	1.10000	1.10000	1.00000	1.00000	1.00000
40193067	1.10000	1.10000	1.00000	1.00000	1.00000
40193068	1.10000	1.10000	1.00000	1.00000	1.00000
40193069	1.10000	1.10000	1.00000	1.00000	1.00000
40193070	1.10000	1.10000	1.00000	1.00000	1.00000
40193071	1.10000	1.10000	1.00000	1.00000	1.00000
40193072	1.10000	1.10000	1.00000	1.00000	1.00000
40193073	1.10000	1.10000	1.00000	1.00000	1.00000
40193074	1.10000	1.10000	1.00000	1.00000	1.00000
40193075	1.10000	1.10000	1.00000	1.00000	1.00000
40193076	1.10000	1.10000	1.00000	1.00000	1.00000
40193077	1.10000	1.10000	1.00000	1.00000	1.00000
40193078	1.10000	1.10000	1.00000	1.00000	1.00000
40193079	1.10000	1.10000	1.00000	1.00000	1.00000
40193080	1.10000	1.10000	1.00000	1.00000	1.00000
40193081	1.10000	1.10000	1.00000	1.00000	1.00000
40193082	1.10000	1.10000	1.00000	1.00000	1.00000
40193083	1.10000	1.10000	1.00000	1.00000	1.00000
40193084	1.10000	1.10000	1.00000	1.00000	1.00000
40193085	1.10000	1.10000	1.00000	1.00000	1.00000
40193086	1.10000	1.10000	1.00000	1.00000	1.00000
40193087	1.10000	1.10000	1.00000	1.00000	1.00000
40193088	1.10000	1.10000	1.00000	1.00000	1.00000
40193089	1.10000	1.10000	1.00000	1.00000	1.00000
40193090	1.10000	1.10000	1.00000	1.00000	1.00000
40193091	1.10000	1.10000	1.00000	1.00000	1.00000
40193092	1.10000	1.10000	1.00000	1.00000	1.00000
40193093	1.10000	1.10000	1.00000	1.00000	1.00000
40193094	1.10000	1.10000	1.00000	1.00000	1.00000
40193095	1.10000	1.10000	1.00000	1.00000	1.00000
40193096	1.10000	1.10000	1.00000	1.00000	1.00000
40193097	1.10000	1.10000	1.00000	1.00000	1.00000
40193098	1.10000	1.10000	1.00000	1.00000	1.00000
40193099	1.10000	1.10000	1.00000	1.00000	1.00000
40193100	1.10000	1.10000	1.00000	1.00000	1.00000
40193101	1.10000	1.10000	1.00000	1.00000	1.00000
40193102	1.10000	1.10000	1.00000	1.00000	1.00000
40193103	1.10000	1.10000	1.00000	1.00000	1.00000
40193104	1.10000	1.10000	1.00000	1.00000	1.00000
40193105	1.10000	1.10000	1.00000	1.00000	1.00000
40193106	1.10000	1.10000	1.00000	1.00000	1.00000
40193107	1.10000	1.10000	1.00000	1.00000	1.00000
40193108	1.10000	1.10000	1.00000	1.00000	1.00000
40193109	1.10000	1.10000	1.00000	1.00000	1.00000
40193110	1.10000	1.10000	1.00000	1.00000	1.00000
40193111	1.10000	1.10000	1.00000	1.00000	1.00000
40193112	1.10000	1.10000	1.00000	1.00000	1.00000
40193113	1.10000	1.10000	1.00000	1.00000	1.00000
40193114	1.10000	1.10000	1.00000	1.00000	1.00000
40193115	1.10000	1.10000	1.00000	1.00000	1.00000
40193116	1.10000	1.10000	1.00000	1.00000	1.00000
40193117	1.10000	1.10000	1.00000	1.00000	1.00000
40193118	1.10000	1.10000	1.00000	1.00000	1.00000
40193119	1.10000	1.10000	1.00000	1.00000	1.00000
40193120	1.10000	1.10000	1.00000	1.00000	1.00000
40193121	1.10000	1.10000	1.00000	1.00000	1.00000
40193122	1.10000	1.10000	1.00000	1.00000	1.00000
40193123	1.10000	1.10000	1.00000	1.00000	1.00000
40193124	1.10000	1.10000	1.00000	1.00000	1.00000
40193125	1.10000	1.10000	1.00000	1.00000	1.00000
40193126	1.10000	1.10000	1.00000	1.00000	1.00000
40193127	1.10000	1.10000	1.00000	1.00000	1.00000
40193128	1.10000	1.10000	1.00000	1.00000	1.00000
40193129	1.10000	1.10000	1.00000	1.00000	1.00000
40193130	1.10000	1.10000	1.00000	1.00000	1.00000
40193131	1.10000	1.10000	1.00000	1.00000	1.00000
40193132	1.10000	1.10000	1.00000	1.00000	1.00000
40193133	1.10000	1.10000	1.00000	1.00000	1.00000
40193134	1.10000	1.10000	1.00000	1.00000	1.00000
40193135	1.10000	1.10000	1.00000	1.00000	1.00000
40193136	1.10000	1.10000	1.00000	1.00000	1.00000
40193137	1.10000	1.10000	1.00000	1.00000	1.00000
40193138	1.10000	1.10000	1.00000	1.00000	1.00000
40193139	1.10000	1.10000	1.00000	1.00000	1.00000
40193140	1.10000	1.10000	1.00000	1.00000	1.00000
40193141	1.10000	1.10000	1.00000	1.00000	1.00000
40193142	1.10000	1.10000	1.00000	1.00000	1.00000
40193143	1.10000	1.10000	1.00000	1.00000	1.00000
401931					

827 rows × 5 columns

```

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=['home', 'away'],
                           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

```

# 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

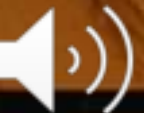
```

✓ 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

```

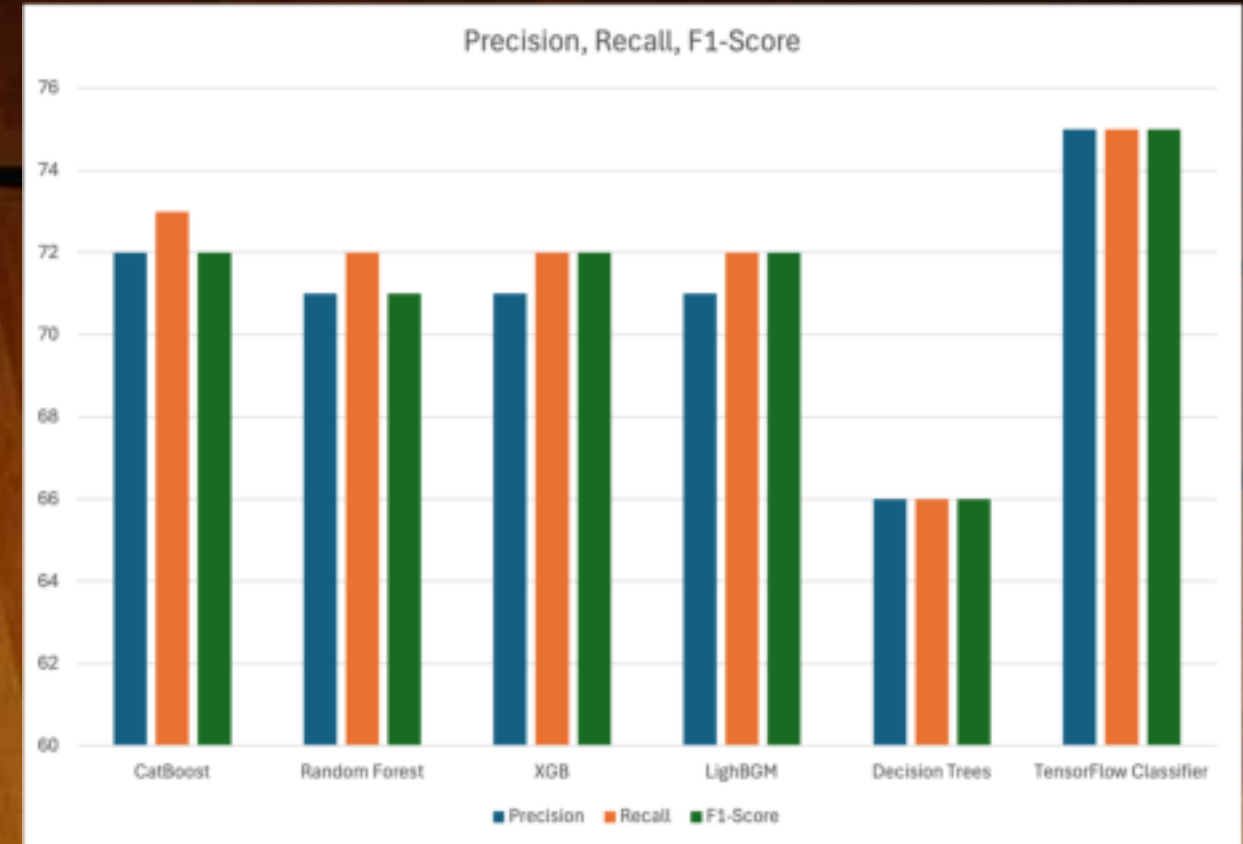
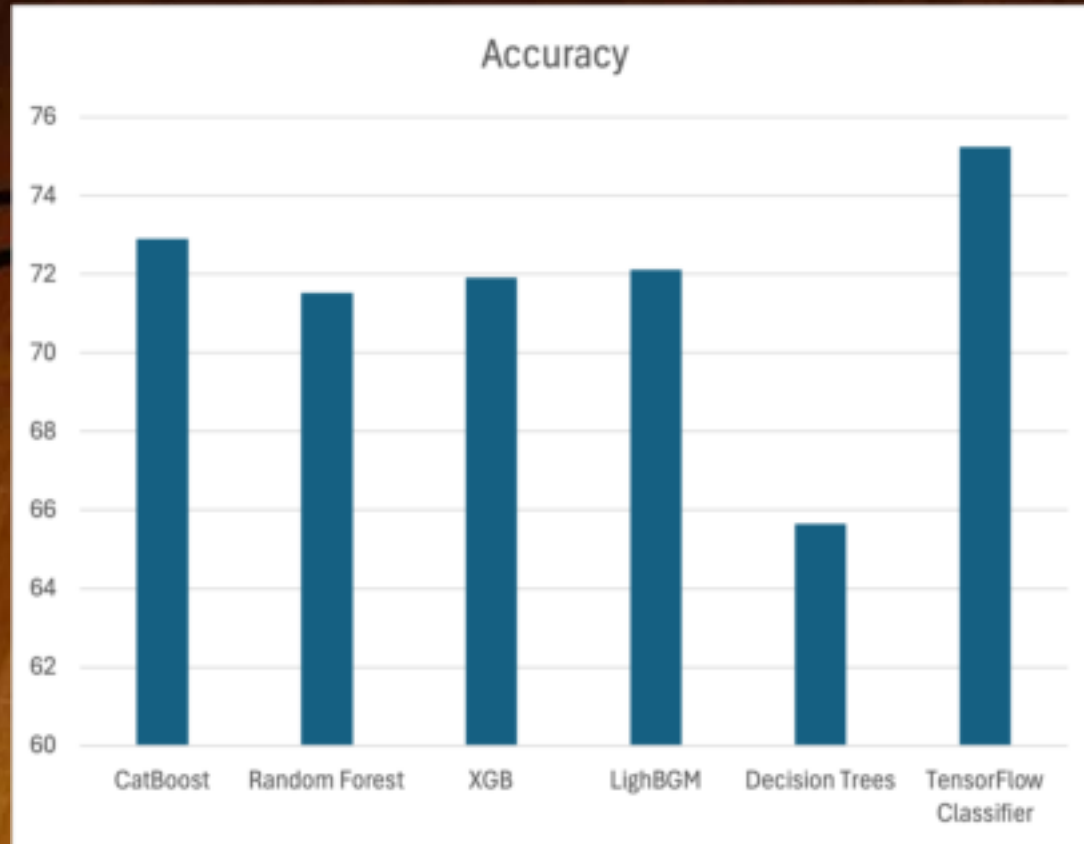


Modelling

- **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]]
```

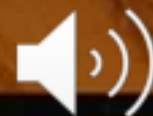
Accuracy: 0.7152641878669276

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]]
```

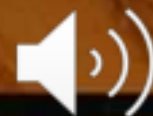
```
Accuracy: 0.7211350293542075
```

```
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]]
```

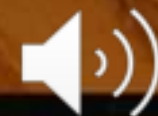
Accuracy: 0.7524461839530333

Tensorflow Classification Report:

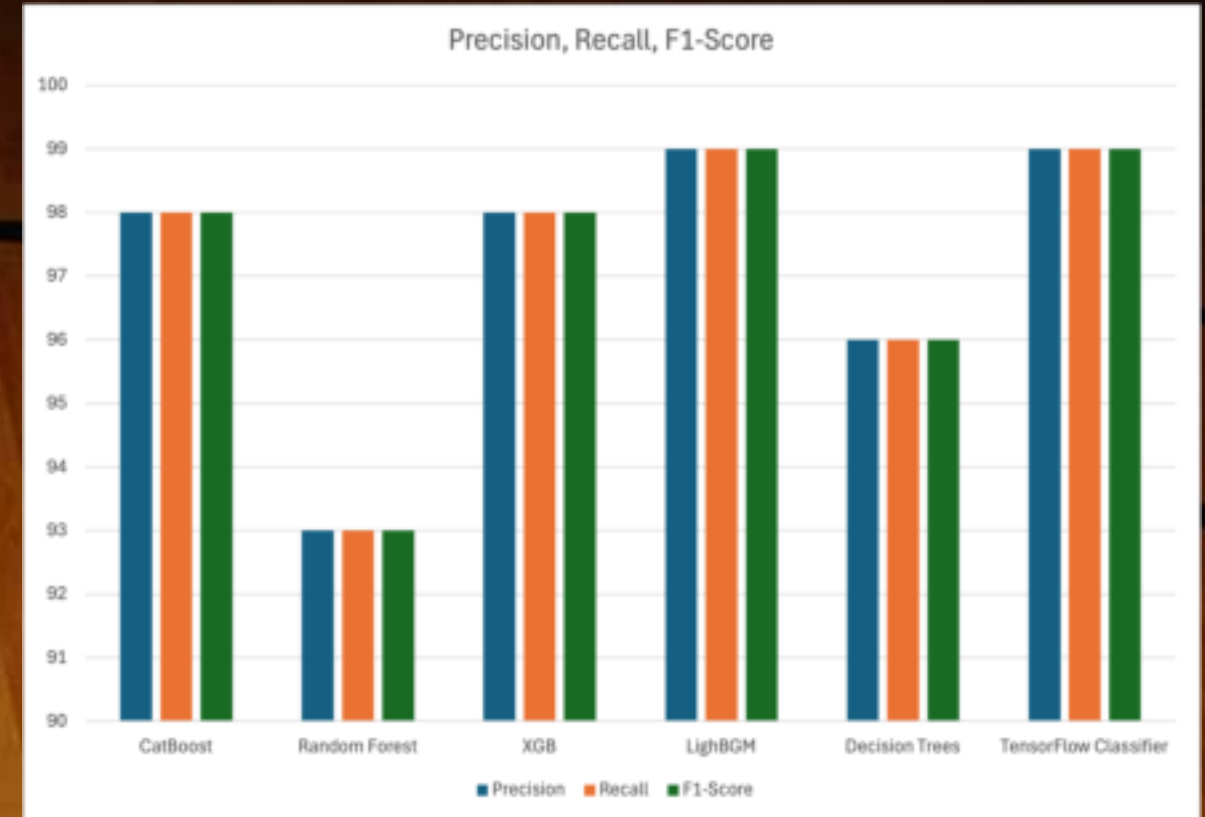
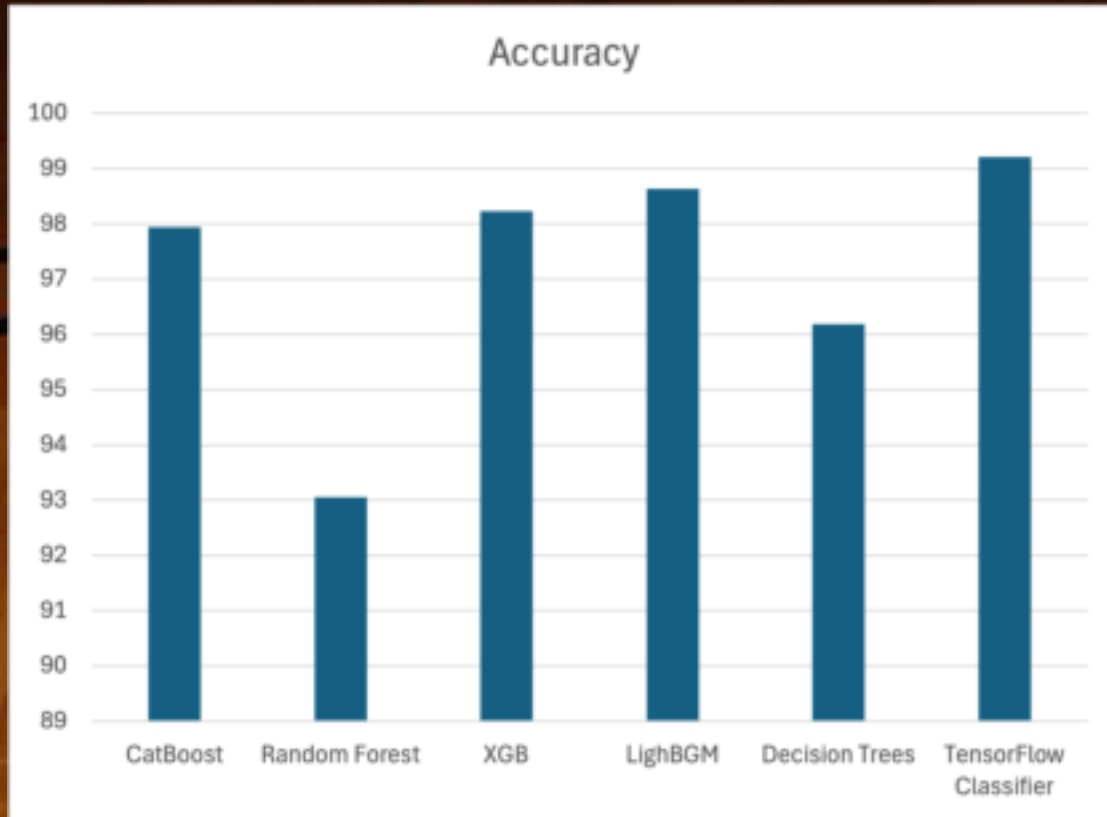
	precision	recall	f1-score	support
0.0	0.69	0.57	0.63	372
1.0	0.78	0.86	0.81	650
accuracy			0.75	1022
macro avg	0.74	0.71	0.72	1022
weighted avg	0.75	0.75	0.75	1022

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:

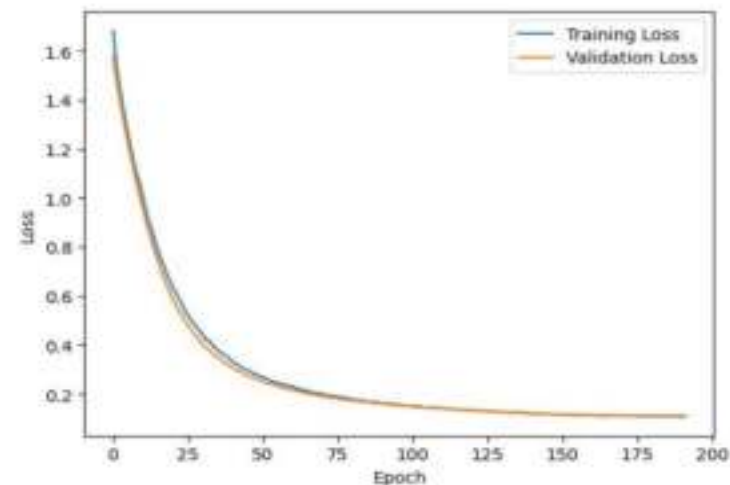
```
[[359 13]
 [ 8 642]]
```

```
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         0.99         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]]
```

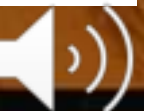
Accuracy: 0.961839530332681

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         1022
weighted avg         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         1022
weighted avg         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.

```
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_mv(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()
    data_dict["FT_pct_max_away"] = player_stats.loc[away_players, "FT_pct"].max()
    data_dict["FT_pct_max_home"] = player_stats.loc[home_players, "FT_pct"].max()
    data_dict["FT_pct_median_away"] = player_stats.loc[away_players, "FT_pct"].median()
    data_dict["FT_pct_median_home"] = player_stats.loc[home_players, "FT_pct"].median()
```

```
# 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(classifier, data_df, home_team, away_team, home_players, away_players)

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

return output[0]
```

[298] ✓ 0.0s

```
predict_winner_and_mv(unique_home_teams[186], unique_away_teams[25],
                       unique_players[1866:1866],
                       unique_players[7679], 0, 0)
```

[251] ✓ 0.0s

--- 'Maine wins with a probability of 0.99 and the MVP is Brandon Ruest'

```
unique_home_teams[186], unique_away_teams[25]
```

[252] ✓ 0.0s

--- ('Maine', 'N Colorado')

[+ Code](#) [+ Markdown](#)

```
# Launch the web app
import os
os.environ["GRADIO_TEMP_PATH"] = "/tmp/localhost/12345"
iface = gr.Interface(
    predict_winner_and_mv,
    [home_team, away_team, home_players, away_players, home_rest_days, away_rest_days],
    output=output_text
)
iface.launch(debug=True, inline=True, height=1000)
```

[253] 0.0m 40.0s

--- 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.

Running on local URL: <http://127.0.0.1:1780>

To create a public link, set 'shareTrue' in 'launch()'.

Home Team

•

Away Team

•

Home Players

•

Away Players

•

Home Team Rest Days

0

Away Team Rest Days

0

Prediction

Flag

Clear

Submit



Interactive Interface

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

Running on local URL: <http://127.0.0.1:1780>

To create a public link, set 'shareTrue' in 'launch()'.

Home Team

•

Away Team

•

Home Players

•

Away Players

•

Home Team Rest Days

0

Away Team Rest Days

0

Prediction

Flag

Clear

Submit



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: $Score = (clutch_pct \times 0.4) + (PER \times 0.6)$
- The player with the highest score from the predicted winning team is identified as the **MVP**.



Demonstration

Home Team

Away Team

Home Players

Away Players

Home Team Rest Days


Away Team Rest Days

Prediction

Flag

Clear

Submit



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! 