```python
In [1]: import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```python
In [2]: data = pd.read_csv("footballmathces_v2.csv", encoding="ISO-8859-1",delimiter = ";")
df = data.copy()
df.head()
```

Out[2]:

	Result	Above_2.5	Total Goals	away_player_stats_player10_stats_age	away_player_stats_player10_stats_season_appearances	away_player_stats_player10_stats_season_assists	away_player_stats_player10_stats_season_goals	awa
0	0	1	6	30	0	0	0	
1	2	1	7	36	25	8	4	
2	0	1	4	32	0	0	0	
3	0	0	0	28	3	1	0	
4	1	0	1	36	18	3	1	

5 rows × 318 columns

```python
In [3]: df["weather_data_condition"].unique()
```

```
Out[3]: array(['Clear sky', 'Mainly clear partly cloudy and overcast',
       'Drizzle: Light moderate and dense intensity',
       'Snow fall: Slight moderate and heavy intensity',
       'Rain: Slight moderate and heavy intensity'], dtype=object)
```

```python
In [4]: dummy_df = pd.get_dummies(df['weather_data_condition'])
new_df = dummy_df.applymap(lambda x: 1 if x  else 0)

df = pd.concat([df, new_df], axis=1)
```

```
C:\Users\Baki Akgun\AppData\Local\Temp\ipykernel_21764\423584530.py:2: FutureWarning: DataFrame.applymap has been deprecated. Use DataFrame.map instead.
  new_df = dummy_df.applymap(lambda x: 1 if x  else 0)
```

```python
In [5]: df.drop(columns = "weather_data_condition",inplace = True)
```

```python
In [6]: def extract_and_replace(value):
    return value.split('.')[0]

df['weather_data_degree'] = df['weather_data_degree'].apply(lambda x: extract_and_replace(x))

df['weather_data_wspeed'] = df['weather_data_wspeed'].apply(lambda x: extract_and_replace(x))
df["away_team_data_average_age"] = df['away_team_data_average_age'].apply(lambda x: extract_and_replace(x))
df["home_team_data_average_age"] = df['home_team_data_average_age'].apply(lambda x: extract_and_replace(x))
df['weather_data_degree'] = df['weather_data_degree'].apply(lambda x: extract_and_replace(x))

df['weather_data_wspeed'] = df['weather_data_wspeed'].astype(int)
df["away_team_data_average_age"] = df['away_team_data_average_age'].astype(int)
df["home_team_data_average_age"] = df['home_team_data_average_age'].astype(int)
df['weather_data_degree'] = df['weather_data_degree'].astype(int)
```

```python
In [7]: df
```

Out[7]:

	Result	Above_2.5	Total Goals	away_player_stats_player10_stats_age	away_player_stats_player10_stats_season_appearances	away_player_stats_player10_stats_season_assists	away_player_stats_player10_stats_season_goals	a
0	0	1	6	30	0	0	0	
1	2	1	7	36	25	8	4	
2	0	1	4	32	0	0	0	
3	0	0	0	28	3	1	0	
4	1	0	1	36	18	3	1	
...	...	...	...	...	...	...	...	
813	1	1	3	24	0	0	0	
814	1	0	2	28	24	7	9	
815	2	1	3	31	0	0	0	
816	2	1	3	26	4	1	0	
817	2	0	1	30	0	0	0	

818 rows × 322 columns

```python
In [8]: for player_num in range(1, 12):
    gol_sutun = f'away_player_stats_player{player_num}_stats_season_goals'
    appearances_sutun = f'away_player_stats_player{player_num}_stats_season_appearances'
    assists_sutun = f'away_player_stats_player{player_num}_stats_season_assists'

    # Sütunlar varsa
    if gol_sutun in df.columns and appearances_sutun in df.columns:
        df[f'away_player{player_num}_gol_atma_orani'] = (df[gol_sutun] / df[appearances_sutun]) * 100
```

```python
In [9]: for player_num in range(1, 12):
    gol_sutun = f'home_player_stats_player{player_num}_stats_season_goals'
    appearances_sutun = f'home_player_stats_player{player_num}_stats_season_appearances'
    assists_sutun = f'home_player_stats_player{player_num}_stats_season_assists'

    # Sütunlar varsa
    if gol_sutun in df.columns and appearances_sutun in df.columns:
        df[f'home_player{player_num}_gol_atma_orani'] = (df[gol_sutun] / df[appearances_sutun]) * 100
```

```python
In [10]: for player_num in range(1, 12):
    gol_sutun = f'away_player_stats_player{player_num}_stats_overall_goals'
    appearances_sutun = f'away_player_stats_player{player_num}_stats_overall_appearances'
    assists_sutun = f'away_player_stats_player{player_num}_stats_overall_assists'

    # Sütunlar varsa
    if gol_sutun in df.columns and appearances_sutun in df.columns:
        df[f'away_player{player_num}_gol_atma_orani_overall'] = (df[gol_sutun] / df[appearances_sutun]) * 100
```

```python
In [11]: for player_num in range(1, 12):
    gol_sutun = f'home_player_stats_player{player_num}_stats_overall_goals'
    appearances_sutun = f'home_player_stats_player{player_num}_stats_overall_appearances'
    assists_sutun = f'home_player_stats_player{player_num}_stats_overall_assists'

    # Sütunlar varsa
    if gol_sutun in df.columns and appearances_sutun in df.columns:
        df[f'home_player{player_num}_gol_atma_orani_overall'] = (df[gol_sutun] / df[appearances_sutun]) * 100
```

```python
In [12]: player_seasonal_columns = df.columns[df.columns.str.contains('stats_season')]

df.drop(columns=player_seasonal_columns, inplace=True)
```

```python
In [13]: player_seasonal_columns = df.columns[df.columns.str.contains('stats_total')]

df.drop(columns=player_seasonal_columns, inplace=True)
```

```python
In [14]: team_average = df[[f'home_player{player_num}_gol_atma_orani' for player_num in range(1, 12)]].mean(axis=1)
team_averag_away = df[[f'away_player{player_num}_gol_atma_orani' for player_num in range(1, 12)]].mean(axis=1)

df['team_average_home'] = team_average
df['team_average_away'] = team_averag_away
```

```python
In [15]: df.head()
```

Out[15]:

	Result	Above_2.5	Total Goals	away_player_stats_player10_stats_age	away_player_stats_player10_stats_value	away_player_stats_player11_stats_age	away_player_stats_player11_stats_value	away_player_stats_player1_stats_age
0	0	1	6	30	650000	27	2500000	32
1	2	1	7	36	2000000	31	20000000	37
2	0	1	4	32	250000	30	1000000	28
3	0	0	0	28	1600000	31	1300000	34
4	1	0	1	36	1000000	38	75000	33

5 rows × 104 columns

```python
In [16]: for column in df.columns:
    print(f"{column}: {df[column].isnull().sum()} eksik değer")
df = df.fillna(0)
```

```
Result: 0 eksik değer
Above_2.5: 0 eksik değer
Total Goals: 0 eksik değer
away_player_stats_player10_stats_age: 0 eksik değer
away_player_stats_player10_stats_value: 0 eksik değer
away_player_stats_player11_stats_age: 0 eksik değer
away_player_stats_player11_stats_value: 0 eksik değer
away_player_stats_player1_stats_age: 0 eksik değer
away_player_stats_player1_stats_value: 0 eksik değer
away_player_stats_player2_stats_age: 0 eksik değer
away_player_stats_player2_stats_value: 0 eksik değer
away_player_stats_player3_stats_age: 0 eksik değer
away_player_stats_player3_stats_value: 0 eksik değer
away_player_stats_player4_stats_age: 0 eksik değer
away_player_stats_player4_stats_value: 0 eksik değer
away_player_stats_player5_stats_age: 0 eksik değer
away_player_stats_player5_stats_value: 0 eksik değer
away_player_stats_player6_stats_age: 0 eksik değer
away_player_stats_player6_stats_value: 0 eksik değer
away_player_stats_player7_stats_age: 0 eksik değer
away_player_stats_player7_stats_value: 0 eksik değer
away_player_stats_player8_stats_age: 0 eksik değer
away_player_stats_player8_stats_value: 0 eksik değer
away_player_stats_player9_stats_age: 0 eksik değer
away_player_stats_player9_stats_value: 0 eksik değer
away_team_data_average_age: 0 eksik değer
away_team_data_national_players: 0 eksik değer
away_team_data_squad_size: 0 eksik değer
away_team_data_stadium_seat: 0 eksik değer
away_team_data_team_trophies: 0 eksik değer
away_team_data_team_value: 0 eksik değer
home_player_stats_player10_stats_age: 0 eksik değer
home_player_stats_player10_stats_value: 0 eksik değer
home_player_stats_player11_stats_age: 0 eksik değer
home_player_stats_player11_stats_value: 0 eksik değer
home_player_stats_player1_stats_age: 0 eksik değer
home_player_stats_player1_stats_value: 0 eksik değer
home_player_stats_player2_stats_age: 0 eksik değer
home_player_stats_player2_stats_value: 0 eksik değer
home_player_stats_player3_stats_age: 0 eksik değer
home_player_stats_player3_stats_value: 0 eksik değer
home_player_stats_player4_stats_age: 0 eksik değer
home_player_stats_player4_stats_value: 0 eksik değer
home_player_stats_player5_stats_age: 0 eksik değer
home_player_stats_player5_stats_value: 0 eksik değer
home_player_stats_player6_stats_age: 0 eksik değer
home_player_stats_player6_stats_value: 0 eksik değer
home_player_stats_player7_stats_age: 0 eksik değer
home_player_stats_player7_stats_value: 0 eksik değer
home_player_stats_player8_stats_age: 0 eksik değer
home_player_stats_player8_stats_value: 0 eksik değer
home_player_stats_player9_stats_age: 0 eksik değer
home_player_stats_player9_stats_value: 0 eksik değer
home_team_data_average_age: 0 eksik değer
home_team_data_national_players: 0 eksik değer
home_team_data_squad_size: 0 eksik değer
home_team_data_stadium_seat: 0 eksik değer
home_team_data_team_trophies: 0 eksik değer
home_team_data_team_value: 0 eksik değer
match_statistics_away_corners: 0 eksik değer
match_statistics_away_fouls: 0 eksik değer
match_statistics_away_free_kicks: 0 eksik değer
match_statistics_away_offsides: 0 eksik değer
match_statistics_away_position: 0 eksik değer
match_statistics_away_shots_off_target: 0 eksik değer
match_statistics_away_total_shots: 0 eksik değer
match_statistics_home_corners: 0 eksik değer
match_statistics_home_fouls: 0 eksik değer
match_statistics_home_free_kicks: 0 eksik değer
match_statistics_home_offsides: 0 eksik değer
match_statistics_home_position: 0 eksik değer
match_statistics_home_shots_off_target: 0 eksik değer
match_statistics_home_total_shots: 0 eksik değer
weather_data_degree: 0 eksik değer
weather_data_wspeed: 0 eksik değer
Clear sky: 0 eksik değer
Drizzle: Light moderate and dense intensity: 0 eksik değer
Mainly clear partly cloudy and overcast: 0 eksik değer
Rain: Slight moderate and heavy intensity: 0 eksik değer
Snow fall: Slight moderate and heavy intensity: 0 eksik değer
away_player1_gol_atma_orani: 273 eksik değer
away_player2_gol_atma_orani: 319 eksik değer
away_player3_gol_atma_orani: 295 eksik değer
away_player4_gol_atma_orani: 241 eksik değer
away_player5_gol_atma_orani: 230 eksik değer
away_player6_gol_atma_orani: 280 eksik değer
away_player7_gol_atma_orani: 293 eksik değer
away_player8_gol_atma_orani: 282 eksik değer
away_player9_gol_atma_orani: 327 eksik değer
away_player10_gol_atma_orani: 278 eksik değer
away_player11_gol_atma_orani: 362 eksik değer
home_player1_gol_atma_orani: 283 eksik değer
home_player2_gol_atma_orani: 311 eksik değer
home_player3_gol_atma_orani: 288 eksik değer
home_player4_gol_atma_orani: 241 eksik değer
home_player5_gol_atma_orani: 233 eksik değer
home_player6_gol_atma_orani: 304 eksik değer
home_player7_gol_atma_orani: 293 eksik değer
home_player8_gol_atma_orani: 276 eksik değer
home_player9_gol_atma_orani: 311 eksik değer
home_player10_gol_atma_orani: 300 eksik değer
home_player11_gol_atma_orani: 359 eksik değer
team_average_home: 8 eksik değer
team_average_away: 10 eksik değer
```

```python
In [17]: # df.to_csv("denemefootball.csv",index = False)
```

```python
In [18]: # df = df.drop(columns =  df.filter(regex='stats_value'))
```

```python
In [19]: # df = df.drop(columns =  df.filter(regex='stats_age'))
```

```python
In [20]: # df = df.drop(columns =  df.filter(regex='team_value'))
```

```python
In [21]: # df = df.drop(columns =  df.filter(regex='stadium_seat'))
```

```python
In [22]: # df = df.drop(columns =  df.filter(regex='national_players'))
```

```python
In [23]: # df = df.drop(columns =  df.filter(regex='squad_size'))
```

```python
In [24]: # df = df.drop(columns =  "home_player1_gol_atma_orani")
```

```python
In [25]: # df = df.drop(columns =  "away_player1_gol_atma_orani")
```

```python
In [26]: df = pd.read_csv("denemefootball.csv")

df.columns
```

```
Out[26]: Index(['Result', 'Above_2.5', 'Total Goals', 'away_team_data_average_age',
       'away_team_data_team_trophies', 'home_team_data_average_age',
       'home_team_data_team_trophies', 'match_statistics_away_corners',
       'match_statistics_away_fouls', 'match_statistics_away_free_kicks',
       'match_statistics_away_offsides', 'match_statistics_away_position',
       'match_statistics_away_shots_off_target',
       'match_statistics_away_total_shots', 'match_statistics_home_corners',
       'match_statistics_home_fouls', 'match_statistics_home_free_kicks',
       'match_statistics_home_offsides', 'match_statistics_home_position',
       'match_statistics_home_shots_off_target',
       'match_statistics_home_total_shots', 'weather_data_degree',
       'weather_data_wspeed', 'Clear sky',
       'Drizzle: Light moderate and dense intensity',
       'Mainly clear partly cloudy and overcast',
       'Rain: Slight moderate and heavy intensity',
       'Snow fall: Slight moderate and heavy intensity',
       'away_player2_gol_atma_orani', 'away_player3_gol_atma_orani',
       'away_player4_gol_atma_orani', 'away_player5_gol_atma_orani',
       'away_player6_gol_atma_orani', 'away_player7_gol_atma_orani',
       'away_player8_gol_atma_orani', 'away_player9_gol_atma_orani',
       'away_player10_gol_atma_orani', 'away_player11_gol_atma_orani',
       'home_player2_gol_atma_orani', 'home_player3_gol_atma_orani',
       'home_player4_gol_atma_orani', 'home_player5_gol_atma_orani',
       'home_player6_gol_atma_orani', 'home_player7_gol_atma_orani',
       'home_player8_gol_atma_orani', 'home_player9_gol_atma_orani',
       'home_player10_gol_atma_orani', 'home_player11_gol_atma_orani',
       'team_average_home', 'team_average_away'],
      dtype='object')
```

```python
In [27]: from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import StandardScaler

df = df.dropna()

scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=df.columns)

X, y = scaled_df.drop(columns=["Result", "Above_2.5", "Total Goals", "match_statistics_home_total_shots", "match_statistics_away_total_shots"]), scaled_df["Total Goals"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=47)

model_linear = LinearRegression()
model_linear.fit(X_train, y_train)

y_pred_linear = model_linear.predict(X_test)
r2_linear = r2_score(y_test, y_pred_linear)
print("Linear Regression Test R^2 Score:", r2_linear)

y_train_pred = model_linear.predict(X_train)

r2_linear_train = r2_score(y_train, y_train_pred)
print("Linear Regression Train R^2 Score:", r2_linear_train)
print("-----------------------------------------------")

model_random_forest = RandomForestRegressor(n_estimators=250, min_samples_split=2)
model_random_forest.fit(X_train, y_train)

# Eğitim seti üzerinde tahmin yapın
y_pred_rf_train = model_random_forest.predict(X_train)

y_pred_rf = model_random_forest.predict(X_test)
r2_rf = r2_score(y_test, y_pred_rf)
print("Random Forest Test R^2 Score:", r2_rf)

r2_rf_train = r2_score(y_train, y_pred_rf_train)
print("Random Forest Train R^2 Score:", r2_rf_train)
print("-----------------------------------------------")
model_gradient_boosting = GradientBoostingRegressor(n_estimators=80, min_samples_split=2)
model_gradient_boosting.fit(X_train, y_train)

y_pred_gb_train = model_gradient_boosting.predict(X_train)

y_pred_gb = model_gradient_boosting.predict(X_test)
r2_gb = r2_score(y_test, y_pred_gb)
print("Gradient Boosting Test R^2 Score:", r2_gb)

# Eğitim seti üzerinde R-kare (R²) skorunu hesaplayın
r2_gb_train = r2_score(y_train, y_pred_gb_train)
print("Gradient Boosting Train R^2 Score:", r2_gb_train)
```

```
Linear Regression Test R^2 Score: -0.059492873983678196
Linear Regression Train R^2 Score: 0.0724715661291675
-----------------------------------------------
Random Forest Test R^2 Score: -0.010768947715824151
Random Forest Train R^2 Score: 0.8579331915164541
-----------------------------------------------
Gradient Boosting Test R^2 Score: -0.0721901195454322
Gradient Boosting Train R^2 Score: 0.48910982304238804
```

```python
In [28]: from catboost import CatBoostRegressor

X, y = scaled_df.drop(columns=["Result", "Above_2.5", "Total Goals"]), df["Total Goals"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=47)

model = CatBoostRegressor(iterations=2000,
                          learning_rate=0.01,
                          depth=10,
                          loss_function='RMSE',
                          verbose=100)

model.fit(X_train, y_train, eval_set=(X_test, y_test), verbose=False)

y_test_pred = model.predict(X_test)
test_r2 = r2_score(y_test, y_test_pred)

print("CatBoost Test R^2:", test_r2)

y_train_pred = model.predict(X_train)
train_r2 = r2_score(y_train, y_train_pred)
print("CatBoost Train R^2:", train_r2)
```

```
CatBoost Test R^2: 0.021513981781433045
CatBoost Train R^2: 0.4888646633354614
```

```python
In [32]: from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

df = df.dropna()

X, y = df.drop(columns=["Result", "Above_2.5", "Total Goals"]), df["Above_2.5"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)

# Random Forest Classifier modelini eğitin
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, y_train)

y_pred_gb = gb_model.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("Gradient Boosting Test Accuracy:", accuracy_gb)
accuracy_gb_train = accuracy_score(y_train, gb_model.predict(X_train))
print("Gradient Boosting Train Accuracy:", accuracy_gb_train)

print("----------------------")

y_pred_rf = rf_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest Test Accuracy:", accuracy_rf)
accuracy_rf_train = accuracy_score(y_train, rf_model.predict(X_train))
print("Random Forest Train Accuracy:", accuracy_rf_train)
print("----------------------")


y_pred_tree = tree_model.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print("Decision Tree Test Accuracy:", accuracy_tree)


accuracy_tree_train = accuracy_score(y_train, tree_model.predict(X_train))
print("Decision Tree Train Accuracy:", accuracy_tree_train)
```

```
Gradient Boosting Test Accuracy: 0.676829268292683
Gradient Boosting Train Accuracy: 0.9587155963302753
----------------------
Random Forest Test Accuracy: 0.6280487804878049
Random Forest Train Accuracy: 1.0
----------------------
Decision Tree Test Accuracy: 0.5304878048780488
Decision Tree Train Accuracy: 1.0
```

```python
In [34]: from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

df = df.dropna()

X, y = df.drop(columns=["Result", "Above_2.5", "Total Goals"]), df["Result"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=47)

gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)

rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

tree_model = DecisionTreeClassifier()
tree_model.fit(X_train, y_train)

y_pred_gb = gb_model.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("Gradient Boosting Test Accuracy:", accuracy_gb)


accuracy_gb_train = accuracy_score(y_train, gb_model.predict(X_train))
print("Gradient Boosting Train Accuracy:", accuracy_gb_train)
print("----------------------")

y_pred_rf = rf_model.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest Test Accuracy:", accuracy_rf)

accuracy_rf_train = accuracy_score(y_train, rf_model.predict(X_train))
print("Random Forest Train Accuracy:", accuracy_rf_train)

print("----------------------")
y_pred_tree = tree_model.predict(X_test)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
print("Decision Tree Test Accuracy:", accuracy_tree)


accuracy_tree_train = accuracy_score(y_train, tree_model.predict(X_train))
print("Decision Tree Train Accuracy:", accuracy_tree_train)
```

```
Gradient Boosting Test Accuracy: 0.49390243902439024
Gradient Boosting Train Accuracy: 0.9617737003058104
----------------------
Random Forest Test Accuracy: 0.5975609756097561
Random Forest Train Accuracy: 1.0
----------------------
```

```
Decision Tree Test Accuracy: 0.8451219512195122
Decision Tree Train Accuracy: 1.0
```

In [ ]: