Prediction of NFL Scores and Spread Optimization

Ashley Lauren. Mersman

Northwest Missouri State University, Maryville MO 64468, USA
S2571660@nwmissouri.edu

Abstract. The abstract should briefly summarize the contents of the paper in 150–250 words.

Keywords: sports · betting · machine learning · NFL

1 Introduction

In 2023, approximately 28% of the American public was expected to bet on the NFL. That's 73 million people for an increase of almost 60% from the previous year. Sports betting is legal in 34 states and counting and is a multi-billion dollar industry. [2]

Being able to predict game outcomes and optimize the spread is highly profitable for the industry. The spreads must be close enough to incentivize bettors but also large enough to maximize profits and cover any losses.

Game information (date, season, week, teams, and scores), betting information (favorite to win, spread, over/under), stadium information (stadium, location, elevation, field type, and stadium type) and weather information (temperature, wind speed, and humidity).

1.1 Goals of this Research

The goal of this project is to create a new predictive models for game outcomes and score differentials.

1.2 Process

Data

1. Data Collection
   (a) A curated dataset from Kaggle.com was used that relied on sources for weather, NFL team information, betting information, and game information from the past 50+ years. [4]
2. Data Cleaning

   (a) The data will be cleaned to filter out unnecessary columns leaving the following, game information (date, season, week, teams, and scores), betting information (favorite to win, spread, over/under), stadium information (stadium, location, elevation, field type, and stadium type) and weather information (temperature, wind speed, and humidity).
   (b) Columns will have their data types checked and corrected.
   (c) Check for and correct any duplicates.
3. Data Exploration
   (a) Check for correlations and distributions.

Machine Learning

1. Pre-processing
2. Model Selection and Parameter Tuning
3. Model Analysis

Conclusions

2 Data

2.1 Data Collection

A curated dataset on Kaggle.com by Spreadspoke, a sports data analysis company, was used as the basis of this project. Spreadspoke used a variety of sources, including but not limited to: ESPN, NFL.com, NOAA, NFLweather.com, Pro-Football.com and multiple others referenced on the original dataset page. The dataset if refreshed on a weekly basis. The data is available as multiple CVS's and the R file used to curate the dataset. [4]

The dataset time frame is 1966 for NFL game results and 1979 for betting odds data.

The kaggle package was used with the kaggle api to download the dataset. [1]

nfl_stadiums.csv contains data about the 120 stadiums each game has been played. This data goes beyond simply NFL stadiums due to internationally played games and other special games. Variables:

1. stadium_name
2. stadium_location: city, state of the stadium
3. stadium_open: the year the stadium opened
4. stadium_close: the year the stadium closed
5. stadium_type: weather type (indoor, outdoor, retractable)
6. stadium_address
7. stadium_weather_station_zip-code: the zip-code used for weather data collection
8. stadium_weather_type: weather category based on average temperature
9. stadium_capacity: stadium seating maximum

10. stadium_surface: field type (Grass, Turf, Field-Turf, Hellas Matrix Turf, Grass, Turf (1969-1970), Grass, Turf (1970-1971), Grass, Turf (1971-1974))
11. stadium_weather_station: weather station ID for NOAA data
12. stadium_weather_station_name
13. stadium_latitude
14. stadium_longitude
15. stadium_azimuthangle: angle of the stadium from North
16. stadium_elevation

nfl_teams.csv is a datafile containing information about the specific NFL teams. There are 44 entries due to teams moving cities, changing names, and other modifications. Variables:

1. team_name
2. team_name_short
3. team_id
4. team_id_pfr: team id on Profootball-reference.com
5. team_conference
6. team_division
7. team_conference_pre2022
8. team_division_pre2022

The game data in spreadspoke_scores.csv contains game data since the 1966 season totaling 13,788 games. It has game information, weather information, and betting information Variables:

1. schedule_date: date game played
2. schedule_season: game season
3. schedule_week: week of season
4. schedule_playoff: boolean value, is this a playoff game
5. team_home: home team
6. score_home: home team score
7. score_away: away team score
8. team_away: away team
9. team_favorite_id: favorite team to win
10. spread_favorite: spread
11. over_under_line: over under
12. stadium: stadium name
13. stadium_neutral: Boolean, is the stadium a neutral site
14. weather_temperature
15. weather_wind_mph
16. weather_humidity
17. weather_detail: precipitation detail

The season time information(schedule_date, schedule_season, schedule_week, and schedule_playoff), scoring and team information (team_home, team_away, score_home, score_away), and conditions (stadium, stadium_neutral, stadium_type, stadium_surface, stadium_elevation, weather_temperature, weather_wind_mph, weather_humidity, weather_detail) will be used to predict the game outcome. The betting information (team_favorite_id, spread_favorite, and over_under_line) will be used to optimize the spread. Winner will be added by comparing the team_home and score_home with the team_away and score_away columns.

2.2 Data Cleaning

The data cleaning process started by inspecting the datasets using Pandas `pd.info()` The commas were removed from the stadium_capacity variable using `pd.replace()` and the datatype for stadium_capacity, stadium_open, and stadium_close were changed to Int64 using `pd.astype()`.

```
stadiums.stadium_capacity = stadiums.stadium_capacity.replace(',','', regex=True)
stadiums.stadium_capacity = stadiums.stadium_capacity.astype("Int64")
stadiums.stadium_open = stadiums.stadium_open.astype("Int64")
stadiums.stadium_close = stadiums.stadium_close.astype("Int64")
```

A column for "winner" was created in the dataset for scores by iterating through the dataframe and comparing the score_home with the score_away. The team with the larger score was selected as winner and appended to a list of winners. In the case the scores are equal Tie is appended. The list is then converted to a column in dataframe.

```
#establish who won each game
winners = []

#iterate through games compariong scores to determine winners, append the winner to the lis
for i,v in games.score_home.items():
    if games.score_home[i] > games.score_away[i]:
        winners.append(games.team_home[i])

    elif games.score_away[i]>games.score_home[i]:
        winners.append(games.team_away[i])

    else:
        winners.append("Tie")

#convert list to column
games["winner"] = winners
```

The spread is a point differential that quantifies the margin by which a team is expected to win or lose. A spread of +7 for a home team means that the the home team is expected to lose by 7 points. The favored team must win by more than 7 points in order for the bettor to win a bet for the favored team. [3] The spread_favorite was set up conversely to what is expected by betting lines and had missing values. Therefore a new spread (score_diff) was created to act as the spread. A score_total variable was also created.

```
games['score_difference'] = games['score_away'] - games['score_home']
games['score_total'] = games['score_away'] + games['score_home']
```

The datasets for the score information and stadium information were then merged into one dataframe using `pd.merge()`.

```
merged_df = games.merge(stadiums, left_on="stadium",
                        right_on="stadium_name", suffixes=['_x', '_y'], how="left")
```

The dataframe columns were selected and reordered. Null values were then examined. All rows with missing scores for either score_home or score_away were dropped. The null values under weather_detail were replaced with "No Precip" since the column is used to indicate additional weather information, it is assumed then that the lack of data means lack of precipitation. Lastly, due to the large number of missing values for stadium_surface this column was dropped. The mean value for weather_humidity, weather_wind_mph, and stadium_elevation were used in place of the null values.

```
merged_df = merged_df[['schedule_date', 'schedule_season', 'schedule_week', 'schedule_playo
                        'team_home', 'team_away', 'score_home', 'score_away', 'winner', 'sco
                        ,'stadium', 'stadium_neutral', 'stadium_type', 'stadium_elevation',
                       'weather_temperature', 'weather_wind_mph', 'weather_detail', 'weather_

merged_df = merged_df.dropna(subset=['score_home'])
merged_df = merged_df.dropna(subset=['stadium_type'])

mean_humidity = merged_df['weather_humidity'].mean()
merged_df["weather_humidity"].fillna(mean_humidity, inplace = True)

mean_temp = merged_df['weather_temperature'].mean()
merged_df["weather_temperature"].fillna(mean_temp, inplace = True)

mean_wind = merged_df['weather_wind_mph'].mean()
merged_df["weather_wind_mph"].fillna(mean_wind, inplace = True)



mean_elevation = merged_df['stadium_elevation'].mean()
merged_df["stadium_elevation"].fillna(mean_elevation, inplace = True)


merged_df['weather_detail'].fillna('No Precip', inplace = True)
```

The data was checked for duplicates using `pd.duplicated().sum()`. No duplicated values were detected.

The cleaned dataset has 13,558 rows and 20 columns.

Table 1. Data Variables and Definitions

Variable	Description	Data Type
schedule_date	Date of the game	Object
schedule_season	Year of Football Season	Integer
schedule_week	Week of season	Object
schedule_playoff	True for Playoff, False for Regular Season	Boolean
team_home	Home Team	Object
team_away	Away Team	Object
score_home	Home Team Score	Float
score_away	Away Team Score	Float
winner	Winning Team	Object
score_difference	Difference Between Away Team Score and Home Team Score	Float
score_total	Total Points Scored	Float
stadium	Stadium Name	Object
stadium_neutral	False for Bias, True for Neutral Site	Boolean
stadium_type	Outdoor, Indoor, Or Covered Stadium	Object
stadium_elevation	Elevation of Stadium	Float
weather_temperature	Temperature at Game, F	Float
weather_wind_mph	Wind Speed at Game, mph	Float
weather_detail	Precipitation at Game	Object
weather_humidity	Humidity at Game, %	Float

Finally, the distribution of the columns was reviewed as a final check for outliers, null_values, or strange data that would require further examination. `pd.describe(include='all'`

The dependent variable will be the score difference

The raw files for this project can be found at The Python notebook and other files used for this project can be found at `https://github.com/AMersman/capstone_NFL`.

2.3 Data Exploration

Statistics and Verification Data exploration started by confirming the data was cleaned and missing values were imputed correctly. The first rows of the dataframe were inspected. `df.head()` The datatypes were verified once more. `df.info()` and the summary statistics were viewed using `df.summary(include='all')`.

The Patriots are the winning-est team. Outdoor stadiums are most common. Most games do not have precipitation.

The mean Score for hometeam is 22.76 and for the awayteam is 20.09; there is a home team advantage but it's not large (2.67 points and the mean spread_favorite is 2.66) The mean number of points scored overall is 42.85. There is also not much difference between the mean and the median of these values.

The distribution of the variables was visualized to check for spread and outliers using `df.hist(figsize=(10,10))`

	schedule_date	schedule_season	schedule_week	schedule_playoff	team_home	team_away	score_home	score_away	winner	spread_favorite
count	11098	11098.000000	11098.000000	11098.000000	11098	11098	11098.000000	11098.000000	11098	11098.000000
unique	2117	NaN	NaN	NaN	43	43	NaN	NaN	44	NaN
top	1/3/2010	NaN	NaN	NaN	San Francisco 49ers	Pittsburgh Steelers	NaN	NaN	New England Patriots	NaN
freq	16	NaN	NaN	NaN	383	374	NaN	NaN	451	NaN
mean	NaN	2001.592269	9.505316	0.043792	NaN	NaN	22.760768	20.093711	NaN	-2.656109
std	NaN	12.691542	5.358300	0.204641	NaN	NaN	10.354768	10.070002	NaN	5.803760
min	NaN	1966.000000	1.000000	0.000000	NaN	NaN	0.000000	0.000000	NaN	-26.500000
25%	NaN	1991.000000	5.000000	0.000000	NaN	NaN	16.000000	13.000000	NaN	-6.500000
50%	NaN	2002.000000	10.000000	0.000000	NaN	NaN	23.000000	20.000000	NaN	-3.000000
75%	NaN	2013.000000	14.000000	0.000000	NaN	NaN	30.000000	27.000000	NaN	2.000000
max	NaN	2023.000000	22.000000	1.000000	NaN	NaN	70.000000	59.000000	NaN	18.500000

Fig. 1. Summary Statistics for Cleaned Dataset

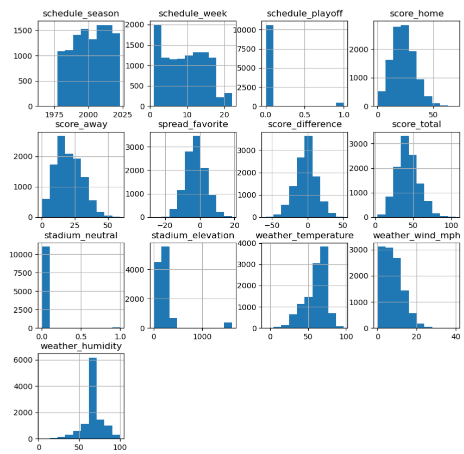


Fig. 2. Distribution of Quantitative Variables

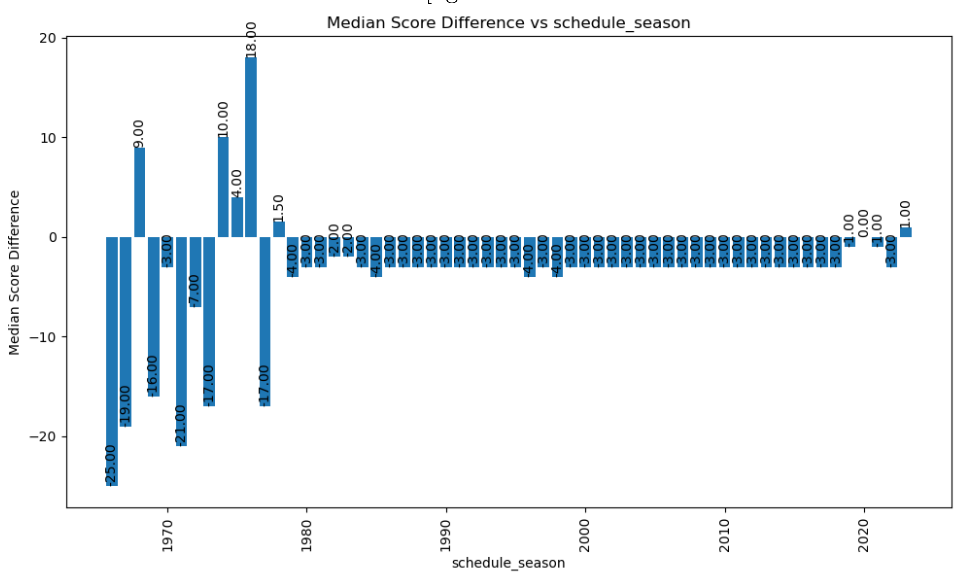
While not completely symmetrical there is little skew to most variables. It should be noted that two of the variables are actually Boolean representations and not continuous variables, hence the distributions. schedule_season and schedule_week are relatively uniform which makes sense for the variable. The increase in number of games played per season makes up the jump in schedule_season distribution.

Visualizations The differences in score_difference across multiple variables was visualized by iterating through the columns to create bar charts of the variables.

```
'''Check median score difference across selected variables'''
def plot_median_diff(column_name):
    items = df[column_name].unique()
    med_total_list = []

    for item in items:
        med_total = df[df[column]==item]['score_difference'].median()
        med_total_list.append(med_total)
```

```
    plt.figure(figsize=(10, 6))
    bars = plt.bar(items, med_total_list, align='center')
    plt.xlabel(column_name)
    plt.ylabel('Median Score Difference')
    plt.title(f'Median Score Difference vs {column_name}')
    plt.xticks(rotation=90)
    plt.tight_layout()

    for bar, label in zip(bars, med_total_list):
            plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), f'{label:.2f}', ha

    plt.show()

columns_of_interest = ['schedule_season', 'team_home', 'team_away' ,'weather_temperature',

for column in columns_of_interest:
    plot_median_diff(column)
```

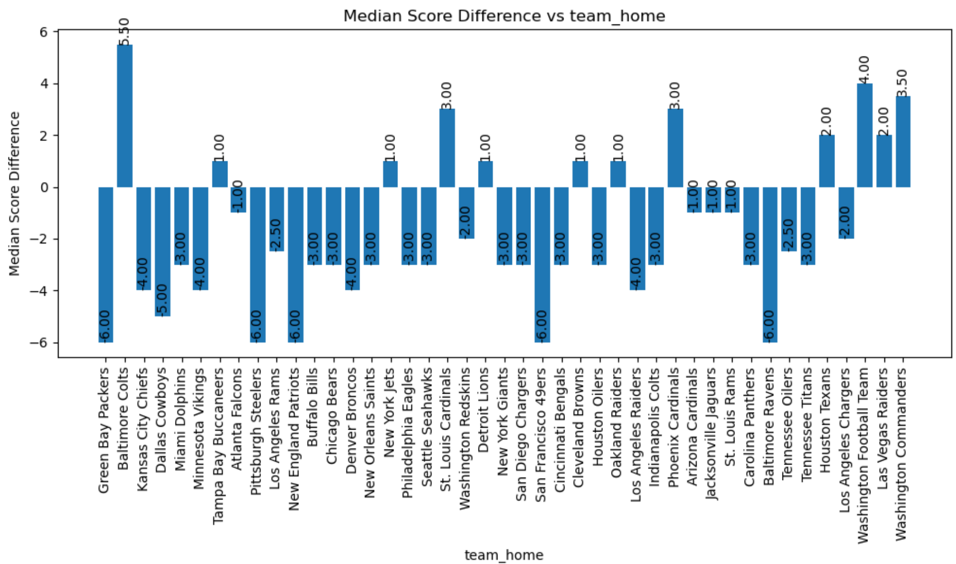
[fig



1]

Fig. 3. Score Difference by Season

The score differences have stabilized and are very small (-3 favoring the home team since the 1980's. This is by design with rule changes favoring close games



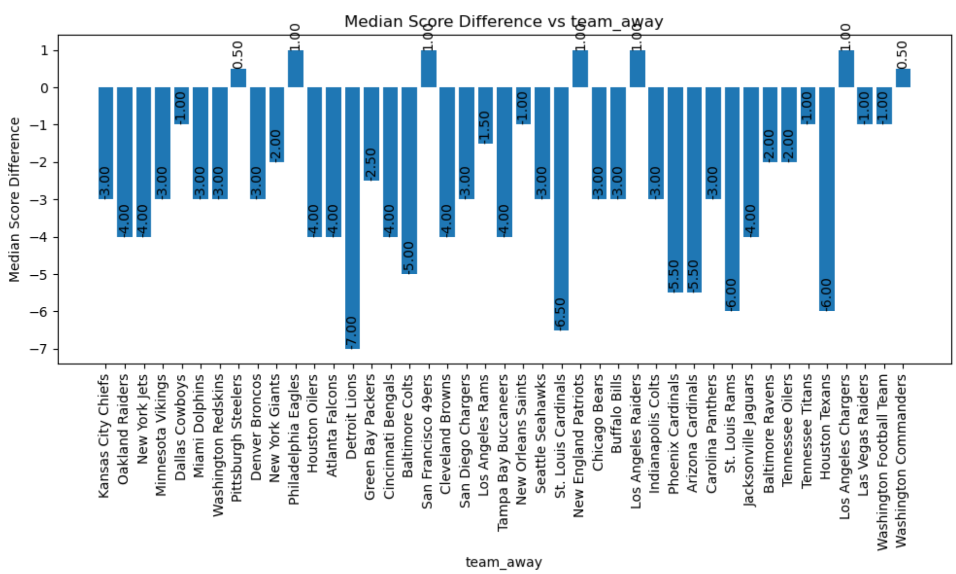


Fig. 4. Score Difference by Home and Away Team

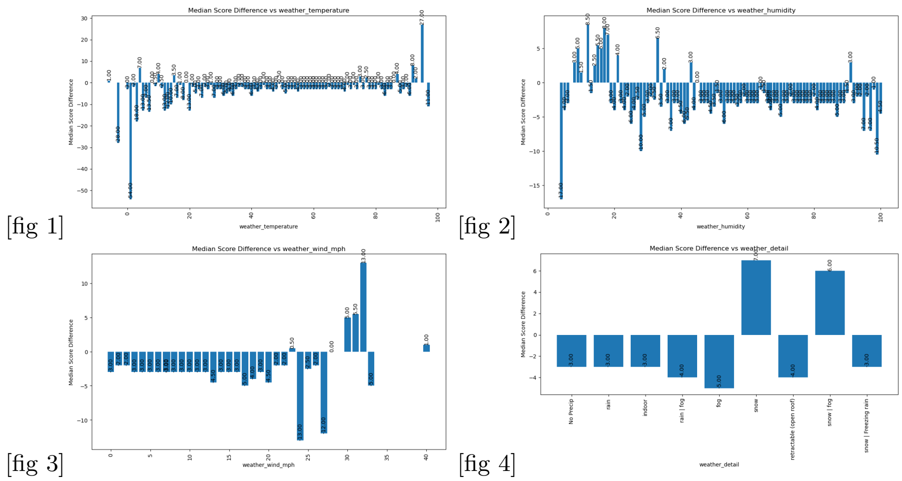


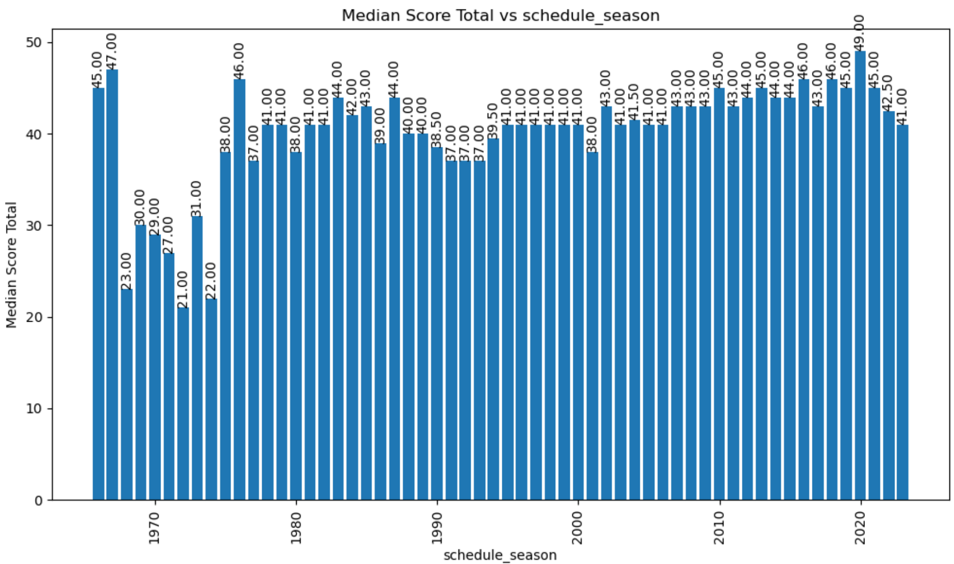
Fig. 5. Score Difference by Weather Conditions

to increase interest and viewership. Weather extremes have an effect on score difference and oddly snow/fog seem to favor the away team which is unexpected.

The same process was used to visualize the score_total across the same variables.

```
'''Check median score total across selected variables'''
def plot_median_total(column_name):
    items = df[column_name].unique()
    med_total_list = []

    for item in items:
        med_total = df[df[column]==item]['score_total'].median()
        med_total_list.append(med_total)

    plt.figure(figsize=(10, 6))
    bars = plt.bar(items, med_total_list, align='center')
    plt.xlabel(column_name)
    plt.ylabel('Median Score Total')
    plt.title(f'Median Score Total vs {column_name}')
    plt.xticks(rotation=90)
    plt.tight_layout()

    for bar, label in zip(bars, med_total_list):
            plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height(), f'{label:.2f}',
            ha='center',
            va='bottom', rotation = 90)

    plt.show()

columns_of_interest = ['schedule_season', 'team_home', 'team_away', 'weather_humidity',
'weather_temperature', 'weather_detail', 'weather_wind_mph' ]

for column in columns_of_interest:
    plot_median_total(column)
```

[fig



1]

Fig. 6. Score Total by Season

Some interesting patterns emerged while viewing the data. Contrary to first assumptions, lower temperatures/lower humidity typically have higher score totals as do snow/freezing rain games. As previously mentioned these games favor the away team but they are also higher scoring overall.

As expected though, score totals go down as wind speed increases. This makes sense when you consider the ways in which points are scored in football. The higher wind speeds make scoring points more difficult because it reduces the accuracy of the field-goal kicker to score. It also reduces the accuracy of the passing game and punting.

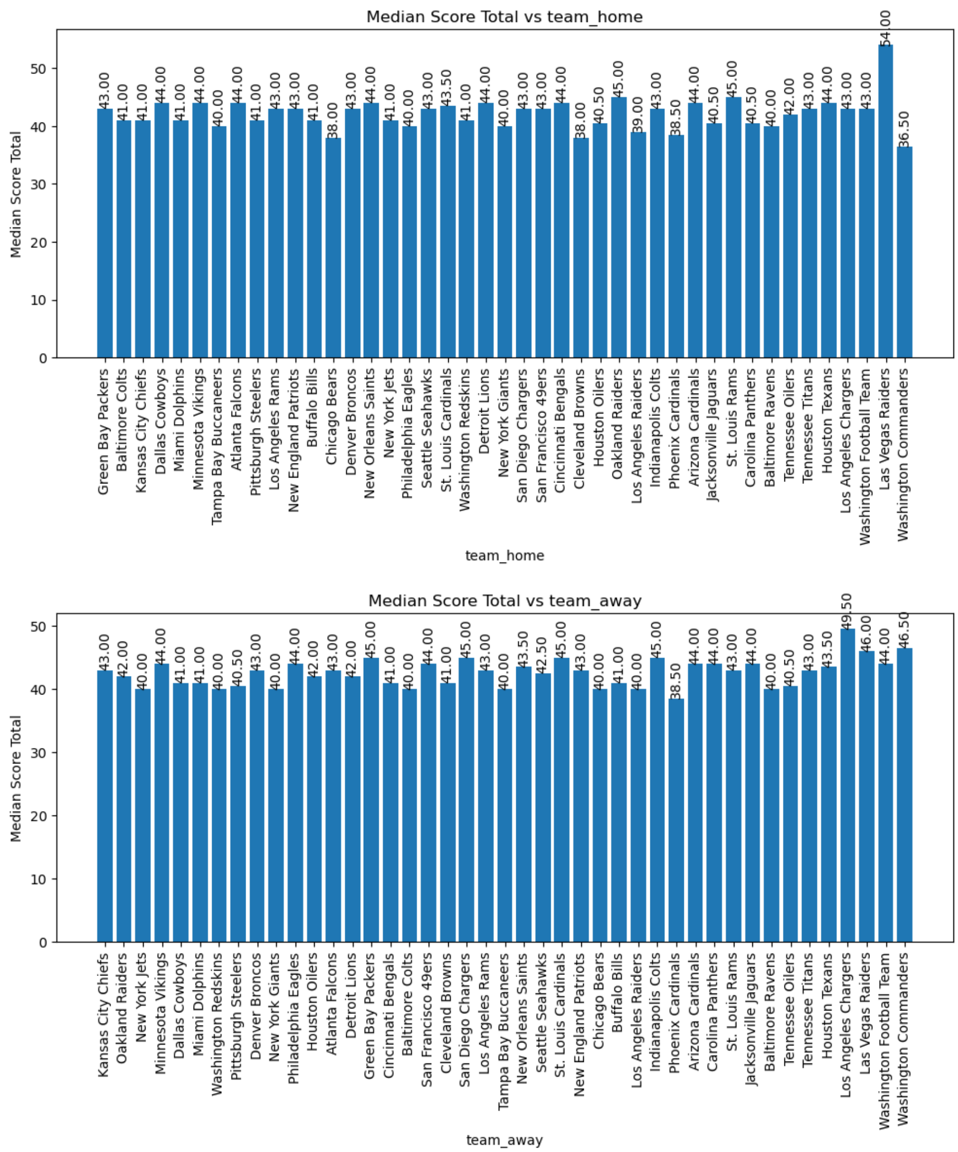
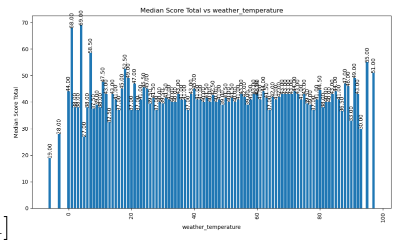
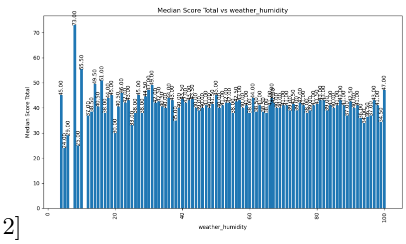


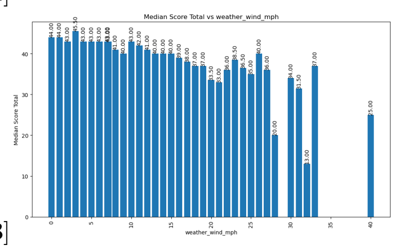
Fig. 7. Score Total by Home and Away Team



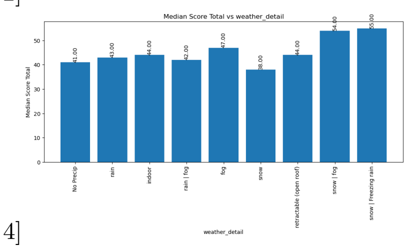
[fig 1]



[fig 2]



[fig 3]



[fig 4]

Fig. 8. Score Total by Weather Conditions

Finally, a correlation matrix was used to view the correlations between the numeric variables.

There is a lot of correlation around the score_home, score_away, and score_total variables. This makes sense but you can also see that there is a relationship between the home team and total score shows a bias to the home team.

There is also a correlation in the weather variables that is obvious as well, colder temperatures have higher wind speeds and lower humidity. Higher stadium elevations also typically see lower humidity and temperatures. Teams also seem to get better at scoring in away games as the game as evolved leading to slightly larger score totals.

Data Exploration Summary In summary, the verification shows that the data was cleaned efficiently. There are no missing values remaining and the data types are correct. The data distributions show that the score_difference is normally distributed and has no outliers. It is also relatively stable on average over the seasons.

The data exploration was used to determine the variables for both a regression and classification model based on the best correlation and data type. The exploration phase is important to make these decisions without relying on any preexisting bias or false understanding of the data.

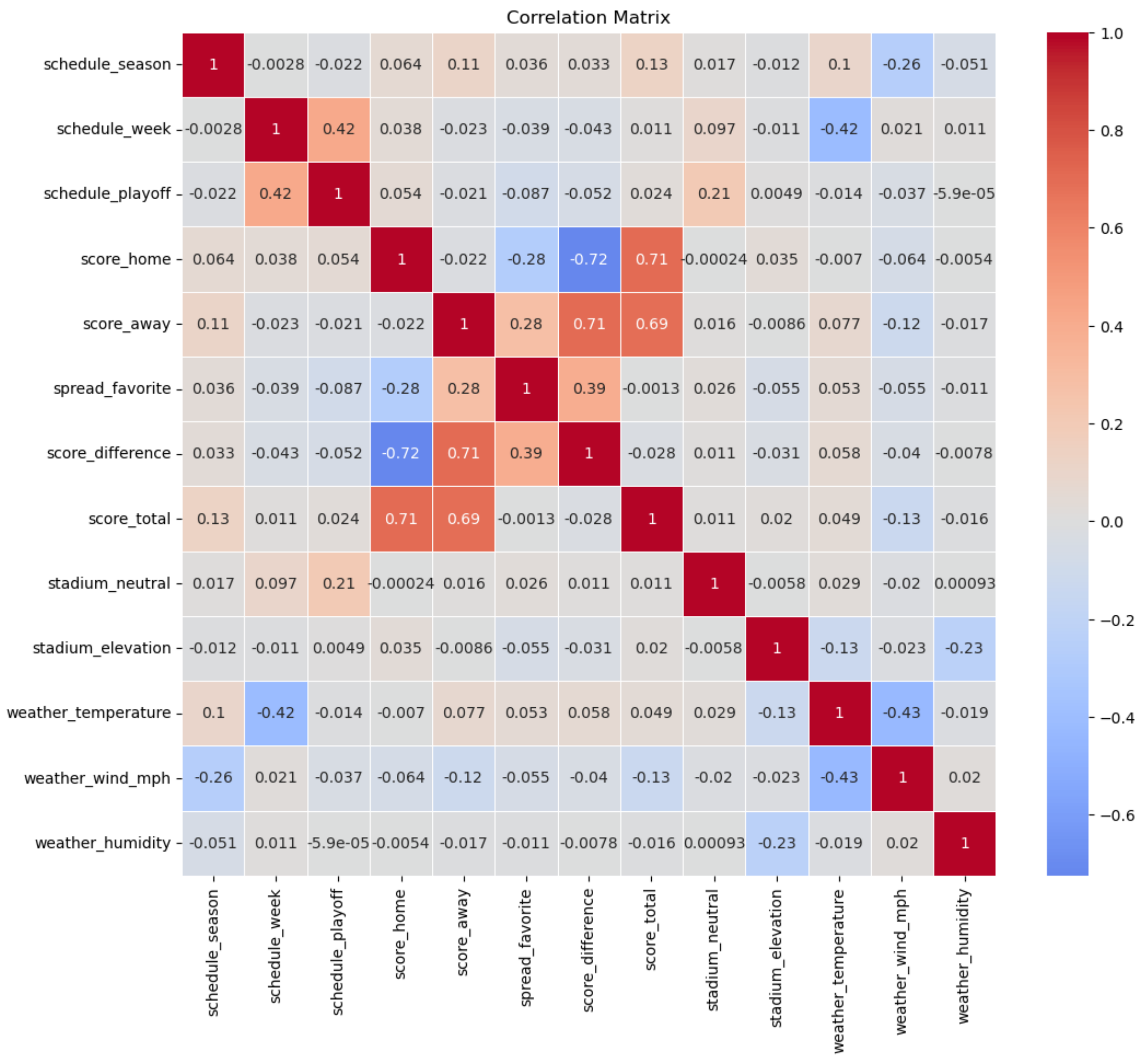


Fig. 9. Correlation of Variables

3 Machine Learning

3.1 Pre-Processing

Before predictive modeling could being the variables needed to be split into 4 sets. X,y for classification of winner and X,y for regression of the score difference.

Regression for Score Difference The regression model was done first. The dataframe was copied and then the appropriate features for regression were selected.

```
df2 = df.copy()
df2 = df2[['schedule_season', 'schedule_week', 'schedule_playoff', 'team_home', 'team_away'
        'score_home', 'score_away', 'stadium_neutral', 'stadium_type', 'weather_temperature
       'weather_detail', 'weather_wind_mph', 'score_difference']]
```

Numeric and categorical variable were defined to standardize the numeric variables using `scaler = StandardScaler(), scaler.fit_transform()` and encode the categorical variables using `encoded_df2 = pd.get_dummies(standardized_df2, columns=categorical_features2)`.

The X and y were then selected. The y being 'score_difference' and the X variables dropping the variables about score for the home and away team.

```
X_reg = encoded_df2.drop(['score_difference', 'score_home', 'score_away'], axis=1)
y_reg = encoded_df2['score_difference']
```

The variables were then split into training and test sets for model selection and trials.

Classification of Winning Team The same steps were followed for the classification with a few changes.

The standardization of numerical features was the same.

```
standardized_df = df.copy()
standardized_df = df[['schedule_season', 'schedule_week', 'schedule_playoff', 'team_home',
         'winner', 'stadium_neutral', 'stadium_type', 'weather_temperature',
       'weather_detail', 'weather_wind_mph']]
numeric_features = df.select_dtypes(include=['int64','float64']).columns
categorical_features = df.select_dtypes(exclude=['int64','float64']).columns

# Standardize the numeric columns
scaler = StandardScaler()
standardized_df[numeric_features] = scaler.fit_transform(standardized_df[numeric_features])

# Now your numeric columns are standardized, excluding the specified columns
standardized_df.head()
```

The categorical variables were preprocessed using labelencoder() instead of the one$_{h}ot_{e}ndcodingsincethewinnerwasabinary1/0optionforHome/Awayteam$.

```
# Creating a instance of label Encoder.
le = LabelEncoder()

for col in standardized_df:
    label = le.fit_transform(standardized_df[col])
    standardized_df[col] = label

standardized_df.head()
```

The same process for splitting the X,y into train/test splits was used for classification as it was for the regression models.

```
X_cat = standardized_df.drop(columns=['winner'])
y_cat = standardized_df['winner']
X_cat_train, X_cat_test, y_cat_train, y_cat_test = train_test_split(X_cat, y_cat, test_size=
```

The appropriate X, y are then used in the nest stage to train and test the appropriate models for regression and classification and analyze the performance of those methods.

3.2 Model Selection and Analysis

The general steps for model select, fit, prediction and analysis were performed for all models. The first model for each was either the DummyRegressor or DummyClassifier. If the subsequent models cannot out perform the dummy models than no further work was performed.

```
#Establish the DummyRegressor model
dr = DummyRegressor(strategy="mean")
#fit the model using the training sets
dr.fit(X_reg_train, y_reg_train)
#Predict
y_dr_pred = dr.predict(X_reg_test)

# Evaluate model performance
print('Dummy Regression Model Performance')
print('MAE:', mean_absolute_error(y_cat_test, y_dr_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_cat_test, y_dr_pred)))
print('MSE:', mean_squared_error(y_cat_test, y_dr_pred))
```

These steps were also performed for the RandomForest `rf = RandomForestRegressor(random_state = 42)`, LinearRegression `lr = LinearRegression()`, and DecisionTreeRegression `dt = DecisionTreeRegressor()`

Their performance was poor. Overall the DummyRegressor was performing the same as the other regression models.

Table 2. Regression Models and Performance

Model	MAE	RMSE	MSE
DummyRegressor	0.7693	0.9897	0.9794
RandomForestRegressor	0.7739	0.9862	0.9726
LinearRegression	0.7699	0.9781	0.9568
DecisionTreeRegressor	1.0766	1.3687	1.8601

The classification models had a slight change in the model metrics used from the regression models.

```
#Establish Classifier Model
dr = DummyClassifier(strategy="stratified", random_state=42)
#Fit model using training sets
dr.fit(X_cat_train, y_cat_train)
#Predict
y_cat_pred = dr.predict(X_cat_test)

# Evaluate model performance
print('DUMMY Classifier Model Performance')
print('MAE:', mean_absolute_error(y_cat_test, y_cat_pred))
print('RMSE:', np.sqrt(mean_squared_error(y_cat_test, y_cat_pred)))
print('MSE:', mean_squared_error(y_cat_test, y_cat_pred))
print('Accuracy:', accuracy_score(y_cat_test, y_cat_pred))
```

The same steps were repeated for the RandomForestClassier `rf_clf = RandomForestClassifier(random_state=42)`, the DecisionTreeClassifer dt$_{c}lf = DecisionTreeClassifier(max_{d}epth = 2, random_{s}tate = 42), andtheMLPClassifier.$

```
NN_clf = MLPClassifier(solver='adam', alpha=1e-5,
hidden_layer_sizes=(6, 3), random_state=1)
```

Just like the regression models the performance compared to the dummy model was poor.

4 Conclusions and Further Work

References

1. https://www.kaggle.com/docs/api

Table 3. Classification Models and Performance

Model Accuracy	**MAE**	**RMSE**	**MSE**
DummyClassifier 0.5132	0.4886	0.7016	0.4922
RandomForestClassifier 0.5676	0.4324	0.6576	0.4324
DecisionTreeClassifier .5637	0.4363	0.6601	0.4363
NeuralNetClassifier .5688	0.4312	.6567	.4312

2. Contessa Brewer, J.G.: A record 73 million americans plan to bet on the nfl this season, survey says (September 2003), `https://www.cnbc.com/2023/09/06/nfl-week-1-record-number-of-americans-to-bet-on-nfl-this-season-.html`
3. Preciado, D.: What is a spread in sports betting?, `https://www.forbes.com/betting/sports-betting/what-is-a-spread/#:~:text=If%20the%20spread%20is%20set,by%20more%20than%20seven%20points.`
4. Spreadspoke: Nfl scores and betting data, `https://www.kaggle.com/datasets/tobycrabtree/nfl-scores-and-betting-data/data`