Big Data

Predicting The Outcome of MMA Fights

Research 1

Overview 1.1

The data set which I got from, https://www.kaggle.com/datasets/rajeevw/ufcdata (11/10/2023, 10:45) includes every main card UFC bout from 1993 to 2021. Some of the data includes the fighters weight, height, reach, age, fighters stance, who won and various other statistics within the fight.

I chose this dataset as I have an interest in the sport of mixed martial arts, and I wanted to know how accurately I could predict the outcome of the fights by using supervised machine learning.

Objectives 1.2

With this data set I want to identify patterns within the fights that i could use to predict which fighter would win. This will be based off the fighter's stance and further variables within the fight, for example; height, reach etc.

Peer Reviewed Paper 1.3

The article starts by providing a comprehensive overview of mixed martial arts (MMA), tracing its historical evolution and its current landscape. It details the structure of MMA contests, covering regulations, weight classes, and judging criteria while drawing comparisons and contrasts with boxing.

Addressing the complexity of predicting MMA fights, the article acknowledges the unpredictability of combat sports, citing the ubiquitous nature of unforeseen events often encapsulated by the phrase 'A Puncher's Chance.' It meticulously outlines the data acquisition process and subsequent modifications, emphasizing the intricate estimation of individual fighter skills. This estimation encompasses various aspects, striking abilities, takedown proficiency, submission accuracy, knockout probabilities, and control times per takedown.

Central to the article is the introduction of the Markov chain model, a sophisticated analytical approach integrating diverse fighter skill data to refine the accuracy of predicting fight outcomes. This model incorporates four key dynamics: Standing States, Ground States, Model Complexity, and Simulation, meticulously simulating potential fight scenarios based on fighter strategies, position effectiveness, action rates, and fight durations.

Furthermore, the article details the modelling of judges' decisions using logistic regression and ordered probit regression models. It compares these models with benchmark approaches like Bradley–Terry and logistic regression, emphasizing their respective strengths and limitations in predicting fight outcomes.

Leveraging MMA fight data spanning from 2001 to 2018, the study achieves a commendable 61.77% accuracy in forecasting fight results, showcasing the efficacy of the proposed predictive model. However, the article underscores the continuous need for improvement, advocating for more granular data, diverse methodological approaches, and adaptability to dynamic trends for heightened predictive precision.

In essence, the article presents an advanced framework for anticipating MMA fight outcomes by amalgamating fighter skill estimations with empirical fight data. Despite notable success, it underscores the persistent pursuit of refinement and advancement for enhanced predictive accuracy and applicability.

Link: https://www.sciencedirect.com/science/article/pii/S0169207022000073?via%3Dihub#section-cited-by

Code 2

Imports 2.1

In [1]:

```
#import necessary libraries
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import plotly.io as pio
pio.renderers.default = 'notebook'
```

Here we are simply connecting to the .csv file giving it the variable name; df, for ease of calling.

In [2]:

```
df = pd.read_csv('data.csv')
```

Preparing the Data 2.2

Before analyzing, it's crucial to review and comprehend the .csv file's data by assessing columns and data types.

In [3]:

```
#Preparing/Checking Data
#===================================

# data size
print(f'The dataset contains {df.shape[0]} rows, and {df.shape[1]} columns')

# Always good to check the names of the columns
df.columns

# and check the data types
df.info()

# for every column
for i in df.columns:
    # print how many features it has
    print(i,len(df[i].unique()))

df.describe()
```

```
The dataset contains 1203 rows, and 144 columns
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1203 entries, 0 to 1202
```

```
Columns: 144 entries, R_fighter to R_age
dtypes: bool(1), float64(102), int64(33), object(8)
memory usage: 1.3+ MB
R_fighter 536
B_fighter 632
Referee 114
date 477
location 150
Winner 2
title_bout 2
weight_class 13
B_avg_KD 244
B_avg_opp_KD 158
B_avg_SIG_STR_pct 735
B_avg_opp_SIG_STR_pct 736
B_avg_TD_pct 616
B_avg_opp_TD_pct 614
B_avg_SUB_ATT 305
B_avg_opp_SUB_ATT 275
B_avg_REV 163
B_avg_opp_REV 159
B_avg_SIG_STR_att 917
B_avg_SIG_STR_landed 787
B_avg_opp_SIG_STR_att 934
B_avg_opp_SIG_STR_landed 771
B_avg_TOTAL_STR_att 944
B_avg_TOTAL_STR_landed 849
B_avg_opp_TOTAL_STR_att 959
B_avg_opp_TOTAL_STR_landed 828
B_avg_TD_att 490
B_avg_TD_landed 400
B_avg_opp_TD_att 478
B_avg_opp_TD_landed 390
B_avg_HEAD_att 908
B_avg_HEAD_landed 712
B_avg_opp_HEAD_att 894
B_avg_opp_HEAD_landed 729
B_avg_BODY_att 623
B_avg_BODY_landed 588
B_avg_opp_BODY_att 627
B_avg_opp_BODY_landed 573
B_avg_LEG_att 574
B_avg_LEG_landed 555
B_avg_opp_LEG_att 588
B_avg_opp_LEG_landed 552
B_avg_DISTANCE_att 905
B_avg_DISTANCE_landed 746
B_avg_opp_DISTANCE_att 895
B_avg_opp_DISTANCE_landed 753
B_avg_CLINCH_att 615
B_avg_CLINCH_landed 558
B_avg_opp_CLINCH_att 595
B_avg_opp_CLINCH_landed 558
B_avg_GROUND_att 636
B_avg_GROUND_landed 569
B_avg_opp_GROUND_att 592
B_avg_opp_GROUND_landed 520
B_avg_CTRL_time(seconds) 1013
B_avg_opp_CTRL_time(seconds) 1022
B_total_time_fought(seconds) 991
B_total_rounds_fought 67
B_total_title_bouts 12
B_current_win_streak 11
B_current_lose_streak 6
B_longest_win_streak 11
B_wins 21
```

```
B_losses 14
B_draw 1
B_win_by_Decision_Majority 3
B_win_by_Decision_Split 6
B_win_by_Decision_Unanimous 11
B_win_by_KO/TKO 12
B_win_by_Submission 11
B_win_by_TKO_Doctor_Stoppage 3
B_Stance 2
B_Height_cms 21
B_Reach_cms 24
B_Weight_lbs 23
R_avg_KD 374
R_avg_opp_KD 227
R_avg_SIG_STR_pct 898
R_avg_opp_SIG_STR_pct 879
R_avg_TD_pct 787
R_avg_opp_TD_pct 776
R_avg_SUB_ATT 444
R_avg_opp_SUB_ATT 390
R_avg_REV 215
R_avg_opp_REV 229
R_avg_SIG_STR_att 1036
R_avg_SIG_STR_landed 955
R_avg_opp_SIG_STR_att 1045
R_avg_opp_SIG_STR_landed 919
R_avg_TOTAL_STR_att 1041
R_avg_TOTAL_STR_landed 976
R_avg_opp_TOTAL_STR_att 1069
R_avg_opp_TOTAL_STR_landed 977
R_avg_TD_att 669
R_avg_TD_landed 580
R_avg_opp_TD_att 672
R_avg_opp_TD_landed 549
R_avg_HEAD_att 1021
R_avg_HEAD_landed 877
R_avg_opp_HEAD_att 1013
R_avg_opp_HEAD_landed 885
R_avg_BODY_att 805
R_avg_BODY_landed 771
R_avg_opp_BODY_att 799
R_avg_opp_BODY_landed 745
R_avg_LEG_att 778
R_avg_LEG_landed 740
R_avg_opp_LEG_att 768
R_avg_opp_LEG_landed 753
R_avg_DISTANCE_att 1026
R_avg_DISTANCE_landed 916
R_avg_opp_DISTANCE_att 1024
R_avg_opp_DISTANCE_landed 897
R_avg_CLINCH_att 796
R_avg_CLINCH_landed 750
R_avg_opp_CLINCH_att 773
R_avg_opp_CLINCH_landed 737
R_avg_GROUND_att 828
R_avg_GROUND_landed 750
R_avg_opp_GROUND_att 743
R_avg_opp_GROUND_landed 682
R_avg_CTRL_time(seconds) 1094
R_avg_opp_CTRL_time(seconds) 1105
R_total_time_fought(seconds) 1082
R_total_rounds_fought 75
R_total_title_bouts 15
R_current_win_streak 14
R_current_lose_streak 5
R_longest_win_streak 16
```

```
R_wins 23
R_losses 15
R_draw 1
R_win_by_Decision_Majority 3
R_win_by_Decision_Split 6
R_win_by_Decision_Unanimous 11
R_win_by_KO/TKO 12
R_win_by_Submission 13
R_win_by_TKO_Doctor_Stoppage 3
R_Stance 2
R_Height_cms 21
R_Reach_cms 24
R_Weight_lbs 25
B_age 27
R_age 26
```

Out[3]:

	Winner	B_avg_KD	B_avg_opp_KD	B_avg_SIG_STR_pct	B_avg_opp_SIG_STR_pct	B_avg_TD_pct	B_avg_
count	1203.000000	1203.000000	1203.000000	1203.000000	1203.000000	1203.000000	
mean	0.640898	0.263192	0.173225	0.454541	0.425203	0.297864	
std	0.479937	0.385142	0.312796	0.123221	0.125891	0.262875	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.379388	0.345781	0.062500	
50%	1.000000	0.062500	0.000000	0.451875	0.417109	0.250000	
75%	1.000000	0.500000	0.250000	0.525000	0.497935	0.500000	
max	1.000000	5.000000	2.000000	1.000000	1.000000	1.000000	

8 rows × 135 columns

A new column, "A_Winner," is added to the DataFrame to clarify fight outcomes. This enhances comparison ease. Saving the DataFrame to a new CSV file boosts model usability/reusability. This separation enables additional data incorporation without altering result files, ensuring a more organized and presentable dataset

In [4]:

```python
df['A_Winner'] = ''  # Initialize the column with empty strings

#Makes the actual results clearer/easier to understand
for index, row in df.iterrows():

    if row['Winner'] == 1:
        df.at[index, 'A_Winner'] = row['R_Stance']

    elif row['Winner'] == 0:
        df.at[index, 'A_Winner'] = row['B_Stance']

    else:
        #Making sure that the data is correct and the previous
        #function worked
        df.at[index, 'A_Winner'] = "Error"

# Save the DataFrame to a new CSV file
df.to_csv('output.csv', index=False)
```

The current dataset will undergo deeper cleaning later despite minimal changes. To enhance model reusability, a separate .csv file for results will be created. This allows for adding new fights and introducing additional data into the model for future expansion.

Exploring the Data 2.3

Age Graph

This graph compares how both stances fair before and after they turn 30

Columns with numerical data are isolated. Percentiles [0.1, 0.25, 0.5, 0.75] are computed. Winners in the dataset are counted to explore age-related correlations for visualization prep.

In [5]:
```
#Graph for age
numeric_columns = df.select_dtypes(include=[np.number])
# Select only numeric columns
quantiles = numeric_columns.quantile([0.1, 0.25, 0.5, 0.75], axis=0)
df['Winner'].value_counts()
```

Out[5]:
```
Winner
1    771
0    432
Name: count, dtype: int64
```

Code styles graph with dark grid theme using Set2 colors, duplicates dataset for better visualization, enhancing appeal and interpretation..

In [6]:
```
sns.set_style('darkgrid')
sns.set_palette('Set2')
# first we make a copy of the dataset to decode the variables
#(for visualisation purposes)
dfC = df.copy()
```

The 'stances' function categorizes fighter stances. If Red (R_Stance) or Blue (B_Stance) is Orthodox and wins, it's labeled Orthodox; otherwise, it's Southpaw.

In [7]:
```
# Function to determine stances
def stances(row):
    if (row["R_Stance"] == 'Orthodox' and row['Winner'] == 1 or
        row["B_Stance"] == 'Orthodox' and row['Winner'] == 0):
        return 'Orthodox'
    else:
        return 'Southpaw'
```

The function 'changeW' categorizes fighters in the Red and Blue corners as 'Over 30' if either R_age or B_age is 30 or older; otherwise, it labels them 'Under 30'.

In [8]:
```
# Function to determine the age condition
def changeW(row):
    if (row['R_age'] >= 30) or (row['B_age'] >= 30):
        return 'Over 30'
    else:
        return 'Under 30'
```

This updates the Winner column in the dataframe. It applies a function to each value in the Winner column. If the value is Orthodox, it remains unchanged. Otherwise, it's set to Southpaw.

In [9]:
```
# Update the 'Winner' column based on the 'Orthodox' condition
df['Winner'] = df['Winner'].apply(lambda x: 'Orthodox' if x ==
                                  "Orthodox" else 'Southpaw')
```

The code updates dataframe columns B_Stance and R_Stance based on Winner. If Winner is Orthodox, it applies stances function; otherwise, keeps original stance.

In [10]:
```
# Update the 'B_Stance' and 'R_Stance' columns based on the
#'Winner' condition
df['B_Stance'] = df.apply(lambda row: stances(row['B_Stance']) if
                          row['Winner'] == 'Orthodox' else
                          row['B_Stance'], axis=1)
df['R_Stance'] = df.apply(lambda row: stances(row['R_Stance']) if
                          row['Winner'] == 'Orthodox' else
                          row['R_Stance'], axis=1)
```

The code uses changeW to update 'Winner' based on fighters' ages (>30 as 'Over 30', <=30 as 'Under 30') row by row in the dataframe.

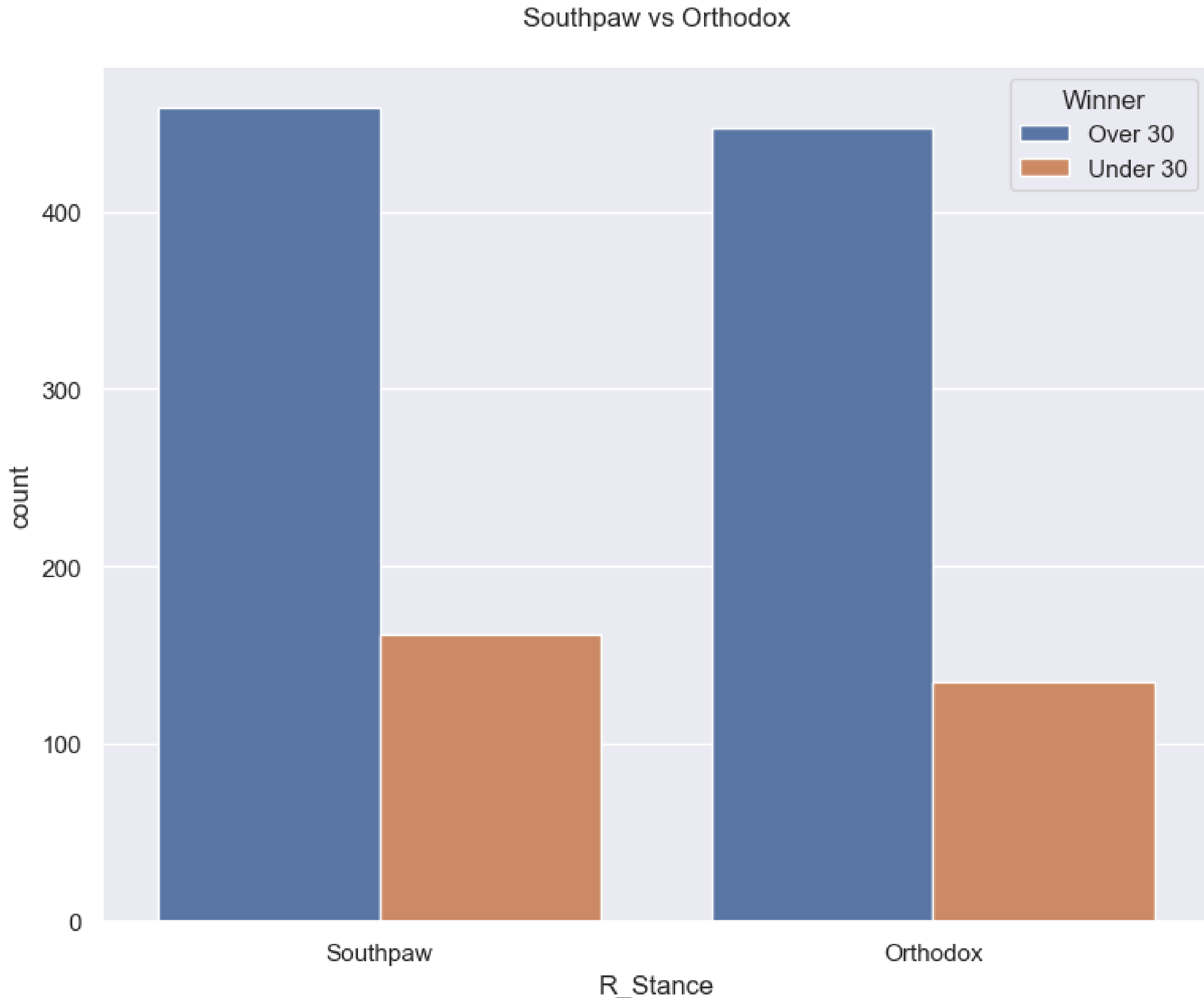
In [11]:
```
# Apply the 'changeW' function to update the 'Winner'
#column based on age condition
df['Winner'] = df.apply(changeW, axis=1)
```

This code creates a copy of the existing DataFrame df and stores it as dfC. This copy enables independent manipulation and analysis without altering the original dataset.

In [12]:
```
# Create a copy of the DataFrame
dfC = df.copy()
```

The code plots count data from 'dfC' with specified figure size, showing 'R_Stance' counts colored by 'Winner', titled 'Southpaw vs Orthodox'.

In [13]:
```
# Plot the data
sns.set(rc={'figure.figsize':(9,7)})
sns.countplot(data=dfC, x='R_Stance', hue='Winner')
plt.title('Southpaw vs Orthodox\n')
plt.show()
```



This graph tell us a few things:

1. That when both the southpaw and orthodox fighters are over the age of 30, the southpaw fighter seems to just take the edge
2. when both the southpaw and orthodox fighters are under the age of 30, the southpaw fighter seem to have a quite sizable advantage

Reach Graph

This graph compares how both stances fair when they have the dis/advantage in Reach

Numeric columns are isolated from a dataframe, their quantiles are computed, and Winner counts are gathered from the dataframe.

In [14]:
```
#graph for reach
# Select only numeric columns
numeric_columns = df.select_dtypes(include=[np.number])
quantiles = numeric_columns.quantile([0.1, 0.25, 0.5, 0.75], axis=0)
df['Winner'].value_counts()
```

Out[14]:
```
Winner
Over 30     906
Under 30    297
Name: count, dtype: int64
```

Matplotlib's style is set to darkgrid while the colour palette is adjusted to Set2. The dataset is copied for variable decoding.

In [15]:
```
sns.set_style('darkgrid')
sns.set_palette('Set2')
# first we make a copy of the dataset to decode the variables
dfC = df.copy()
```

The function stances categorizes based on fighter stances, returning Orthodox if certain conditions are met, else Southpaw.

In [16]:
```
# Function to determine stances
def stances(row):
    if (row["R_Stance"] == 'Orthodox' and row['Winner'] == 1 or
        row["B_Stance"] == 'Orthodox' and row['Winner'] == 0):
        return 'Orthodox'
    else:
        return 'Southpaw'
```

The changeW function assesses reach comparisons between fighters, labelling either Reach Advantage or Reach Disadvantage based on the condition.

In [17]:
```
# Function to determine the age condition
def changeW(row):
    if (row['R_Reach_cms'] >= row['B_Reach_cms']):
        return 'Reach Advantage'
    else:
        return 'Reach Disadvantage'
```

In [18]:
```
# Update the 'Winner' column based on the 'Orthodox' condition
df['Winner'] = df['Winner'].apply(lambda x: 'Orthodox' if x ==
                                  "Orthodox" else 'Southpaw')
```

The Winner column is updated using a lambda function; if Orthodox, it remains unchanged, otherwise, it's set to Southpaw.

In [19]:
```
# Update the 'B_Stance' and 'R_Stance' columns based on the
#'Winner' condition
df['B_Stance'] = df.apply(lambda row: stances(row['B_Stance']) if
                          row['Winner'] == 'Orthodox' else
                          row['B_Stance'], axis=1)
df['R_Stance'] = df.apply(lambda row: stances(row['R_Stance']) if
                          row['Winner'] == 'Orthodox' else
                          row['R_Stance'], axis=1)
```

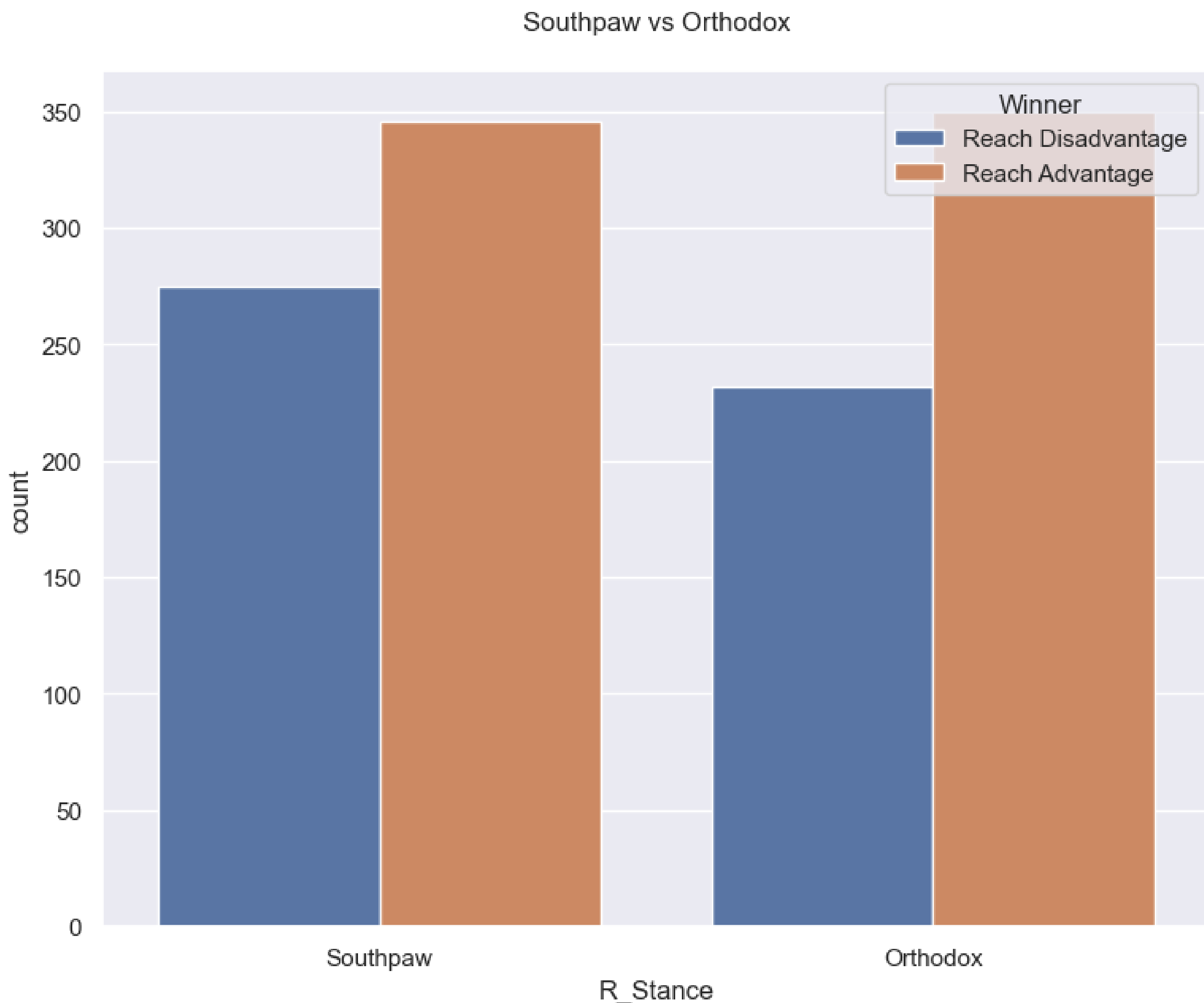
Update Winner using changeW by age conditions; duplicate DataFrame df as dfC for storage and manipulation.

In [20]:
```
# Apply the 'changeW' function to update the 'Winner' column
#based on age condition
df['Winner'] = df.apply(changeW, axis=1)

# Create a copy of the DataFrame
dfC = df.copy()
```

Matplotlib plot with (9,7) size shows count of R_Stance in dfC data, comparing with 'Winner'. Title: Southpaw vs Orthodox. Displayed plot.

In [21]:
```
# Plot the data
sns.set(rc={'figure.figsize':(9,7)})
sns.countplot(data=dfC, x='R_Stance', hue='Winner')
plt.title('Southpaw vs Orthodox\n')
plt.show()
```



This graph tell us a few things:

1. That when both the fighters have the reach advantage, although close the orthodox fighter just edges it
2. When both the fighters have the reach disadvantage, the southpaw has a significant advantage

Height Graph

This graph compares how both stances fair when they have the dis/advantage in Height

First isolates numeric columns, compute their quantiles, and counts Winner values for height related graphing purposes.

In [22]:
```
#graph for height
numeric_columns = df.select_dtypes(include=[np.number])
# Select only numeric columns
quantiles = numeric_columns.quantile([0.1, 0.25, 0.5, 0.75], axis=0)
df['Winner'].value_counts()
```

Out[22]:
```
Winner
```

```
Reach Advantage        696
Reach Disadvantage     507
Name: count, dtype: int64
```

Seaborn configures darkgrid style, Set2 palette for plots; 'df' copied for variable decoding in visualizations.

In [23]:
```
sns.set_style('darkgrid')
sns.set_palette('Set2')
# first we make a copy of the dataset to decode the variables
#(for visualisation_purposes)
dfC = df.copy()
```

Function categorizes stances, Orthodox if conditions (R_Stance, B_Stance, Winner) met, else labels as Southpaw based on outcomes.

In [24]:
```
# Function to determine stances
def stances(row):
    if (row["R_Stance"] == 'Orthodox' and row['Winner'] == 1 or
        row["B_Stance"] == 'Orthodox' and row['Winner'] == 0):
        return 'Orthodox'
    else:
        return 'Southpaw'
```

The function changeW assesses height, assigning Height Advantage if Red corner is taller than Blue; else, Height Disadvantage.

In [25]:
```
# Function to determine the age condition
def changeW(row):
    if (row['R_Height_cms'] >= row['B_Height_cms']):
        return 'Height Advantage'
    else:
        return 'Height Disadvantage'
```

Winner column update,:iIf value is Orthodox, unchanged; else set to Southpaw.

In [26]:
```
# Update the 'Winner' column based on the 'Orthodox' condition
df['Winner'] = df['Winner'].apply(lambda x: 'Orthodox' if x ==
                                  "Orthodox" else 'Southpaw')
```

Update dataframe's B_Stance and R_Stance using Winner condition. Apply stances function for Orthodox winner; keep original values otherwise.

In [27]:
```
# Update the 'B_Stance' and 'R_Stance' columns based on the 'Winner'
#condition
df['B_Stance'] = df.apply(lambda row: stances(row['B_Stance']) if
                          row['Winner'] == 'Orthodox' else
                          row['B_Stance'], axis=1)
df['R_Stance'] = df.apply(lambda row: stances(row['R_Stance']) if
                          row['Winner'] == 'Orthodox' else
                          row['R_Stance'], axis=1)
```

Code updates'Winne' column in dataframe using'change' function, comparing Red and Blue corner heights for advantages/disadvantages.

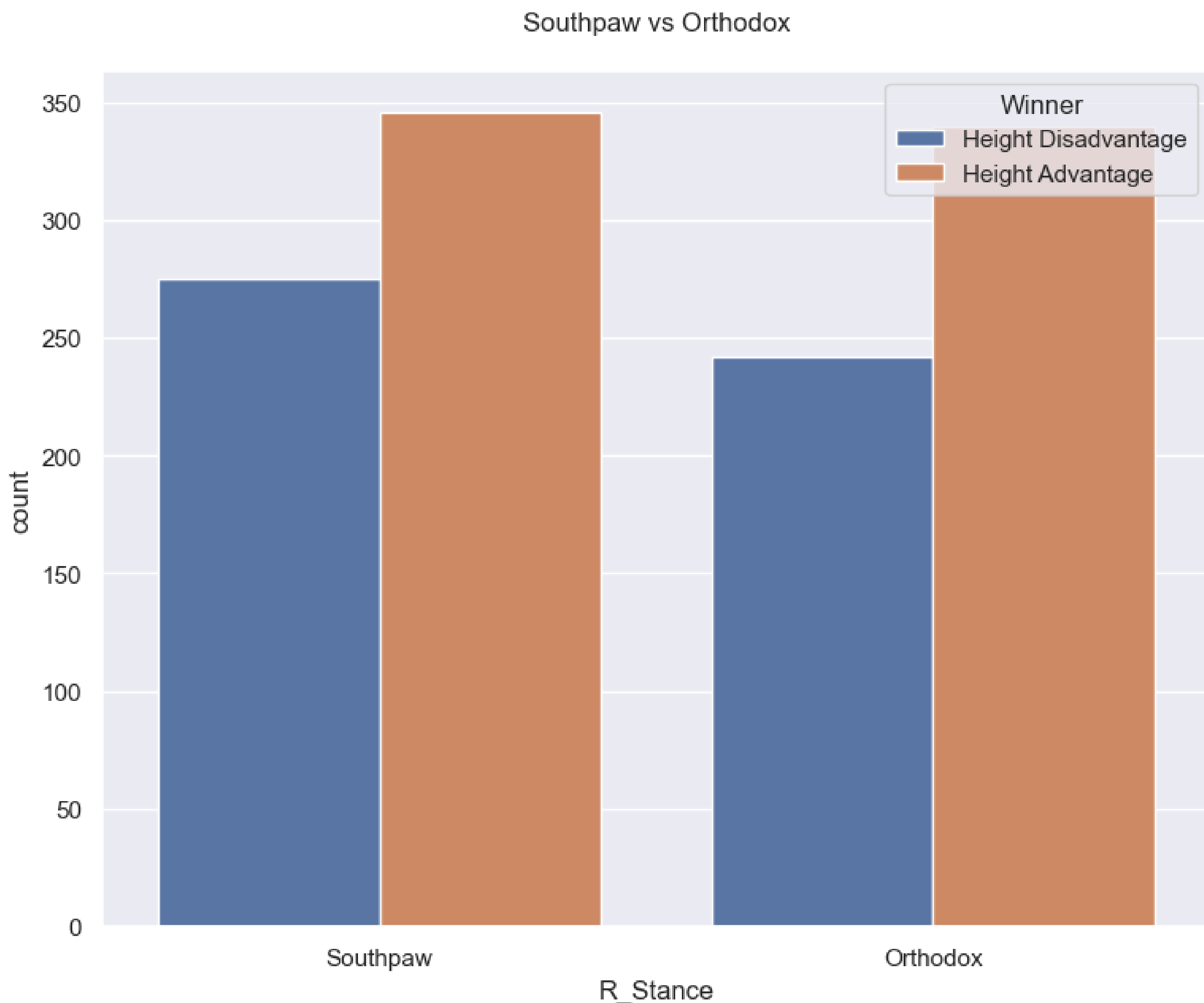
In [28]:
```
# Apply the 'changeW' function to update the 'Winner' column
#based on age condition
df['Winner'] = df.apply(changeW, axis=1)
```

DataFrame dfC created as an independent copy of df for separate analysis and manipulation, preserving original dataset integrity.

In [29]:
```
# Create a copy of the DataFrame
dfC = df.copy()
```

A Seaborn code segment plots a count based on categories in DataFrame dfC (R_Stance), using 'Winner' for color, with 9x7 figure, titled "Southpaw vs Orthodox," displaying the plot.

In [30]:
```
# Plot the data
sns.set(rc={'figure.figsize':(9,7)})
sns.countplot(data=dfC, x='R_Stance', hue='Winner')
plt.title('Southpaw vs Orthodox\n')
plt.show()
```



This graph tell us a few things:

1. That when both the fighters have the Height advantage, although close the southpaw fighter edges it
2. When both the fighters have the reach disadvantage, the southpaw has a significant advantage

Experience Graph

This graph compares how both stances fair when they have the dis/advantage in Ring Experience

Isolate numeric columns, calculate [0.1, 0.25, 0.5, 0.75] percentiles, and count "Winner" values, likely for visualizing experiences.

In [31]:
```
#graph for Experience

numeric_columns = df.select_dtypes(include=[np.number])
# Select only numeric columns
quantiles = numeric_columns.quantile([0.1, 0.25, 0.5, 0.75], axis=0)
df['Winner'].value_counts()
```

Out[31]:
```
Winner
Height Advantage       686
Height Disadvantage    517
Name: count, dtype: int64
```

ConfiguringsSeabor,: plotting style to darkgrid, color palette Set2. Dataset copied as dfC for variable decoding in visualization.

In [32]:
```
sns.set_style('darkgrid')
sns.set_palette('Set2')
# first we make a copy of the dataset to decode the variables
#(for visualisation purposes)
dfC = df.copy()
```

The'stance' function sorts stances by R_Stance, B_Stance, and Winner criteria, yielding Orthodox or Southpaw based on conditions.

In [33]:
```
# Function to determine stances
def stances(row):
    if (row["R_Stance"] == 'Orthodox' and row['Winner'] == 1 or
        row["B_Stance"] == 'Orthodox' and row['Winner'] == 0):
        return 'Orthodox'
    else:
        return 'Southpaw'
```

The"change" function compares Red and Blue fighters' rounds, labeling Red with more or equal rounds as 'More Round Experience.'

In [34]:
```
# Function to determine the age condition
def changeW(row):
    if (row['R_total_rounds_fought'] >= row['B_total_rounds_fought']):
        return 'More Round Experience'
    else:
        return 'Less Round Experience'
```

DataFrame's Winner column updated: If value is Orthodox, unchanged; else, set to Southpaw.

In [35]:
```
# Update the 'Winner' column based on the 'Orthodox' condition
df['Winner'] = df['Winner'].apply(lambda x: 'Orthodox' if x ==
                                  "Orthodox" else 'Southpaw')
```

Winner column updates B_Stance and R_Stance in the DataFrame based on it; if Winner is Orthodox, stances function modifies values..

In [36]:
```
# Update the 'B_Stance' and 'R_Stance' columns based on the 'Winner'
#condition
df['B_Stance'] = df.apply(lambda row: stances(row['B_Stance']) if
                          row['Winner'] == 'Orthodox' else
                          row['B_Stance'], axis=1)
df['R_Stance'] = df.apply(lambda row: stances(row['R_Stance']) if
```

```
                          row['Winner'] == 'Orthodox' else
                          row['R_Stance'], axis=1)
```

The line employs 'changeW' function to update DataFrame's 'Winner' column. It iterates rows, setting 'Winner' based on Red fighter's round experience against Blue.

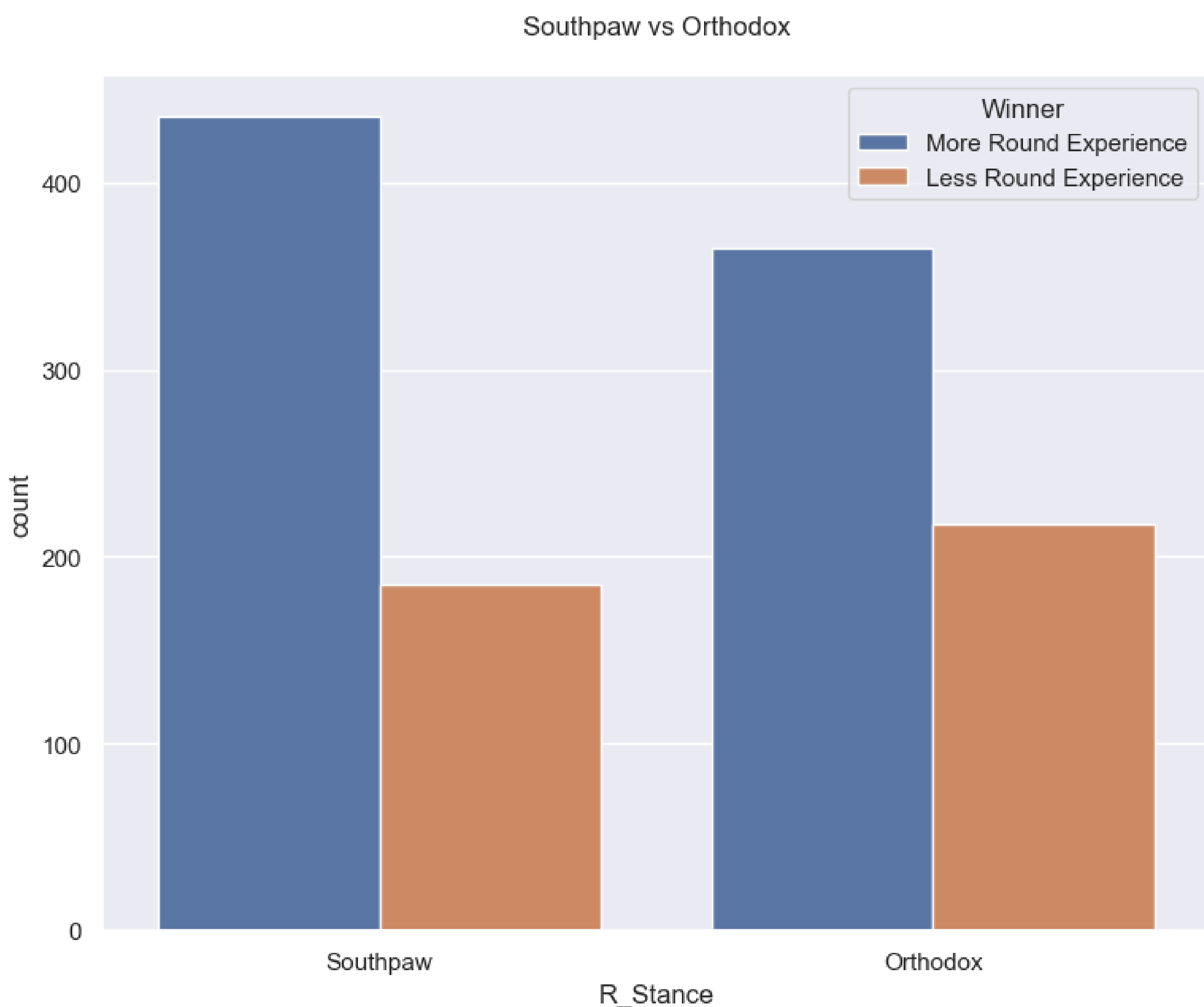
In [37]:
```
# Apply the 'changeW' function to update the 'Winner' column
#based on age condition
df['Winner'] = df.apply(changeW, axis=1)
```

DataFrame df duplicated as dfC retains identical data, enabling separate analysis without altering the initial dataset.

In [38]:
```
# Create a copy of the DataFrame
dfC = df.copy()
```

A count plot from DataFrame dfC: organizes R_Stance counts, colors by Winner, sized (9,7), titled "Southpaw vs Orthodox," displayed.

In [39]:
```
# Plot the data
sns.set(rc={'figure.figsize':(9,7)})
sns.countplot(data=dfC, x='R_Stance', hue='Winner')
plt.title('Southpaw vs Orthodox\n')
plt.show()
```



This graph tell us a few things:

1. That when the fighters have the experience advantage, the southpaw fighter has a serious advantage
2. When the fighters have the experience disadvantage, the orthodox has a significant advantage

This implies perfecting the southpaw stance takes longer than the orthodox one, but over time, it improves performance significantly, indicating its potential superiority.

Decision Tree 2.4

This will take all of the relevant comparisons and disparities between the two fighters that were highlighted with the graphs and uses them to determine the outcome of the fight. The % accuracy is then displayed at the end.

First we must create the P_Winner column in our .csv file, this will for now be populated with empty strings, but will be populated with the decision trees predicted winner though the process.

In [40]:
```
# Initialize the column with empty strings
df['P_Winner'] = ''
```

In [41]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
```

To enhance clarity, dropping numerous irrelevant columns from the messy dataset is crucial despite initial confusion, significantly improving the dataset's relevance.

In [42]:
```
# Cleaning data set
columns_to_drop = [
    'B_avg_KD', 'B_avg_opp_KD', 'B_avg_opp_SIG_STR_pct',
    'B_avg_SIG_STR_pct', 'B_avg_TD_pct', 'B_avg_opp_TD_pct',
    'B_avg_SUB_ATT', 'B_avg_opp_SUB_ATT', 'B_avg_REV',
    'B_avg_opp_REV', 'B_avg_SIG_STR_att', 'B_avg_SIG_STR_landed',
    'B_avg_opp_SIG_STR_att', 'B_avg_opp_SIG_STR_landed',
    'B_avg_TOTAL_STR_att', 'B_avg_TOTAL_STR_landed',
    'B_avg_opp_TOTAL_STR_att', 'B_avg_opp_TOTAL_STR_landed',
    'B_avg_TD_att', 'B_avg_TD_landed', 'B_avg_opp_TD_att',
    'B_avg_opp_TD_landed', 'B_avg_HEAD_att', 'B_avg_HEAD_landed',
    'B_avg_opp_HEAD_att', 'B_avg_opp_HEAD_landed',
    'B_avg_BODY_att', 'B_avg_BODY_landed', 'B_avg_opp_BODY_att',
    'B_avg_opp_BODY_landed', 'B_avg_LEG_att',
    'B_avg_LEG_landed', 'B_avg_opp_LEG_att', 'B_avg_opp_LEG_landed',
    'B_avg_DISTANCE_att', 'B_avg_DISTANCE_landed',
    'B_avg_opp_DISTANCE_att', 'B_avg_opp_DISTANCE_landed',
    'B_avg_CLINCH_att', 'B_avg_CLINCH_landed','B_avg_opp_CLINCH_att',
    'B_avg_opp_CLINCH_landed', 'B_avg_GROUND_att',
    'B_avg_GROUND_landed','B_avg_opp_GROUND_att',
    'B_avg_opp_GROUND_landed', 'B_avg_CTRL_time(seconds)',
    'B_avg_opp_CTRL_time(seconds)','B_total_time_fought(seconds)',
    'B_total_title_bouts', 'B_current_win_streak',
    'B_current_lose_streak','B_longest_win_streak', 'B_wins',
    'B_losses', 'B_draw', 'B_win_by_Decision_Majority',
    'B_win_by_Decision_Split','B_win_by_Decision_Unanimous',
    'B_win_by_KO/TKO', 'B_win_by_Submission',
    'B_win_by_TKO_Doctor_Stoppage','R_avg_KD',
    'R_avg_opp_KD', 'R_avg_SIG_STR_pct', 'R_avg_opp_SIG_STR_pct',
    'R_avg_TD_pct', 'R_avg_opp_TD_pct','R_avg_SUB_ATT',
```

```
    'R_avg_opp_SUB_ATT', 'R_avg_REV', 'R_avg_opp_REV',
    'R_avg_SIG_STR_att', 'R_avg_SIG_STR_landed',
    'R_avg_opp_SIG_STR_att', 'R_avg_opp_SIG_STR_landed',
    'R_avg_TOTAL_STR_att', 'R_avg_TOTAL_STR_landed',
    'R_avg_opp_TOTAL_STR_att', 'R_avg_opp_TOTAL_STR_landed',
    'R_avg_TD_att', 'R_avg_TD_landed', 'R_avg_opp_TD_att',
    'R_avg_opp_TD_landed', 'R_avg_HEAD_att', 'R_avg_HEAD_landed',
    'R_avg_opp_HEAD_att', 'R_avg_opp_HEAD_landed',
    'R_avg_BODY_att', 'R_avg_BODY_landed', 'R_avg_opp_BODY_att',
    'R_avg_opp_BODY_landed', 'R_avg_LEG_att',
    'R_avg_LEG_landed', 'R_avg_opp_LEG_att', 'R_avg_opp_LEG_landed',
    'R_avg_DISTANCE_att', 'R_avg_DISTANCE_landed',
    'R_avg_opp_DISTANCE_att', 'R_avg_opp_DISTANCE_landed',
    'R_avg_CLINCH_att', 'R_avg_CLINCH_landed',
    'R_avg_opp_CLINCH_att', 'R_avg_opp_CLINCH_landed',
    'R_avg_GROUND_att', 'R_avg_GROUND_landed','R_avg_opp_GROUND_att',
    'R_avg_opp_GROUND_landed', 'R_avg_CTRL_time(seconds)',
    'R_avg_opp_CTRL_time(seconds)','R_total_time_fought(seconds)',
    'R_total_title_bouts', 'R_current_win_streak',
    'R_current_lose_streak','R_longest_win_streak', 'R_wins',
    'R_losses', 'R_draw', 'R_win_by_Decision_Majority',
    'R_win_by_Decision_Split','R_win_by_Decision_Unanimous',
    'R_win_by_KO/TKO', 'R_win_by_Submission',
    'R_win_by_TKO_Doctor_Stoppage', 'Referee'
]
```

The code cleans a DataFrame (df) by removing leading/trailing whitespaces from column names, dropping specified columns, and removing rows with missing values.

In [43]:
```
# Remove leading/trailing whitespaces from column names
df.columns = df.columns.str.strip()

# Drop columns
df = df.drop(columns_to_drop, axis=1)

# Drop rows with missing values
df = df.dropna()
df.describe()
```

Out[43]:

	B_total_rounds_fought	B_Height_cms	B_Reach_cms	B_Weight_lbs	R_total_rounds_fought	R_Height_cms	R_
count	1203.000000	1203.000000	1203.000000	1203.000000	1203.000000	1203.000000	
mean	13.638404	179.187182	183.823791	168.347465	18.943475	179.020382	
std	12.795060	7.616544	9.117838	30.300749	15.567206	8.148383	
min	1.000000	152.400000	152.400000	115.000000	1.000000	152.400000	
25%	5.000000	175.260000	177.800000	155.000000	7.000000	172.720000	
50%	9.000000	180.340000	185.420000	170.000000	15.000000	180.340000	
75%	18.000000	185.420000	190.500000	185.000000	27.000000	185.420000	
max	86.000000	203.200000	213.360000	265.000000	85.000000	210.820000	

This code initializes Decision Tree Classifier, defines features/target (df), selects columns, assigns to X/y, performs one-hot encoding for X_encoded.

In [44]:
```
# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

# Define features and target variable
```

```
features = ['B_Stance', 'R_Stance', 'B_age', 'R_age', 'B_Reach_cms',
            'R_Reach_cms', 'B_total_rounds_fought',
            'R_total_rounds_fought']
target = 'A_Winner'

X = df[features]
y = df[target]

# Convert categorical variables to numerical using one-hot encoding
#for the entire dataset
X_encoded = pd.get_dummies(X)
```

DataFrame df split based on 'P_Winner'. X_encoded, y used for train-test split. Model (clf) trained, predictions made on test set

In [45]:
```
# Separate rows where 'P_Winner' is blank and where it's not
blank_winner_rows = df[df['P_Winner'] == '']
non_blank_winner_rows = df[df['P_Winner'] != '']

# Train-test split for computing accuracy
X_train, X_test, y_train, y_test = train_test_split(X_encoded,
                                y, test_size=0.2, random_state=42)
# Train the model on the training set
clf.fit(X_train, y_train)

# Make predictions for the test set
predictions = clf.predict(X_test)
```

The code evaluates Decision Tree accuracy, comparing predicted to actual values using test set, prints accuracy, then trains model on full dataset.

In [46]:
```
# Compute accuracy on the test set
accuracy = accuracy_score(y_test, predictions)
print(f"Decision Tree Accuracy on Test Set: {accuracy * 100:.2f}%")

# Train the model on the entire dataset
clf.fit(X_encoded, y)
```

```
Decision Tree Accuracy on Test Set: 53.53%
```

Out[46]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=42)
```

The code employs a trained model (clf) to predict outcomes (all_predictions) for all data rows (X_encoded), filling the 'P_Winner' column in the original DataFrame (df) with these results.

In [47]:
```
# Make predictions for all rows
if not X_encoded.empty:
    all_predictions = clf.predict(X_encoded)

    # Populate 'P_Winner' column with predictions for all rows
    df.loc[X_encoded.index, 'P_Winner'] = all_predictions
```

The code saves the DataFrame (df) as a new CSV file named 'output.csv', excluding the index.

In [48]:
```
# Save the DataFrame to a new CSV file
df.to_csv('output.csv', index=False)
```

This gives us a 53.5% accuracy.

The percentage achieved, while strong in combat sport aspects, falls slightly short of the paper's 61.77% accuracy with extensive model training. I think i can do better.

Solution Improvement 3

Random Forest 3.1

I decided to use Random Forest as a comparison to my decision tree due to though research I found out that these two models have comparable attributes that tend to complement similar data sets.

In [49]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

This code selects columns for features (X) and target (y) from DataFrame, using one-hot encoding to convert categorical variables in X.

In [50]:
```
# X contains the features, and y contains the target variable
X = df[['B_Stance', 'B_age', 'B_Reach_cms', 'R_Stance', 'R_age',
        'R_Reach_cms', 'B_total_rounds_fought', 'R_total_rounds_fought']]
y = df['A_Winner']

# Convert categorical variables to numerical using one-hot encoding
X = pd.get_dummies(X)
```

This code performs a train-test split on the features (X) and target variable (y). It splits the dataset into training and testing sets with a test size of 20%. Additionally, it initializes a Random Forest Classifier (rf_classifier) with 100 estimators and a random state of 42.

In [51]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                test_size=0.2, random_state=42)

# Initialize the RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators=100,
                                       random_state=42)
```

Train RandomForestClassifier (rf_classifier) on (X_train, y_train), predict (y_pred) using (X_test) for evaluation.

In [52]:
```
# Fit the model on the training data
rf_classifier.fit(X_train, y_train)

# Predict on the test data
y_pred = rf_classifier.predict(X_test)
```

Compare predicted (y_pred) to actual (y_test), calculating accuracy, printing Random Forest Classifier accuracy on test set.

In [53]:
```
# Evaluate the accuracy
accuracy = accuracy_score(y_test, y_pred)
print("")
print("Random Forest Accuracy on Test Set:",
      f"{round(accuracy * 100, 1)}%")
```

```
Random Forest Accuracy on Test Set: 58.9%
```

Conclusion 4

This gives us a 58.9% accuracy.

This is a significant improvement (5.4%) given the circumstances. First of all, on a general note, combat sports are notoriously known for truly anything being possible, David on many of time beats goliath. however, on a more relevant note if you compare my results to the paper, they got 61.77%, which like I mentioned before I'm not too far off the result that they got. which given the data they had available to them and the depth and models that they used; I can firmly say that my model can hold its own.