get_fight

August 15, 2025

[189]:
```
import requests         # HTTP library - sends requests to websites (like␣
 ↪clicking a link)
from bs4 import BeautifulSoup  # HTML parser - reads and navigates HTML code␣
 ↪from websites
import pandas as pd     # Data analysis library - creates tables/spreadsheets␣
 ↪for our data
import time            # Built-in Python library - lets us add delays between␣
 ↪requests
from typing import List, Dict  # Type hints - helps specify what data types␣
 ↪functions expect
from bs4.element import Tag
```

[190]:
```
fight_url = "http://ufcstats.com/fight-details/6b8be0ee3e569ad2"

# headers mimic real browser visitor (bypass bot detection)
our_headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36'
}

response = requests.get(fight_url,headers=our_headers)
```

[191]:
```
fight_soup = BeautifulSoup(response.content, "html.parser")
```

[192]:
```
fight = {} # dict stores fight data
```

[193]:
```
"""
on the ufc website (eg. http://ufcstats.com/fight-details/6b8be0ee3e569ad2)␣
 ↪fighters are split into red/blue fighter.
when playing around with this data set: https://www.kaggle.com/datasets/rajeevw/
 ↪ufcdata
i noticed the red fighter would tend to win 67% of the time, resulting in a␣
 ↪biased datset
to address this issue the following section of code randomly assignes a fighter␣
 ↪to be A or B (my programs equivelent of red/blue)
"""

import random
```

```
# assigns ids to fighters (1/0) representing an index
fighter_a_id = random.getrandbits(1)
fighter_b_id = 1- fighter_a_id

# test for randomness
# count = 0
# for i in range(1000):
#     fighter_a_id = random.getrandbits(1)
#     fighter_b_id = 1- fighter_a_id
#     count += fighter_a_id
# print(count)


# Find the main container with both fighters
fight_details = fight_soup.find('div', class_='b-fight-details__persons␣
 ↪clearfix')

# Find all individual fighter divs
fighters = fight_details.find_all('div', class_='b-fight-details__person')

# Extract fighter names from the links
fighter_names = []
for fighter in fighters:
    name_link = fighter.find('a', class_='b-fight-details__person-link')
    if name_link:
        # Get the text and strip any extra whitespace
        fighter_name = name_link.get_text().strip()
        fighter_names.append(fighter_name)

fight["fighter_a_name"] = fighter_names[fighter_a_id]
fight["fighter_b_name"] = fighter_names[fighter_b_id]

print(f"Fighter A: {fight['fighter_a_name']}")
print(f"Fighter B: {fight['fighter_b_name']}")
```

```
Fighter A: Robert Whittaker
Fighter B: Reinier de Ridder
```

[194]:
```
weight_class = fight_soup.find("i", class_="b-fight-details__fight-title").text.
 ↪split()[0]

fight["weight_class"] = weight_class

print(weight_class + " fight")
```

```
Middleweight fight
```

[195]:
```
"""
the following code collects basic fight information (victory method, fight␣
 ↪format/#rounds,finish time, final round #)
"""

fight_details = fight_soup.find("div",class_="b-fight-details__content").
 ↪find_all("p","b-fight-details__text")[0]

# finds html elements storing the key fight info
method = fight_details.find("i","b-fight-details__text-item_first")
round = fight_details.find("i","b-fight-details__text-item")
time = round.find_next_sibling("i")
format = time.find_next_sibling("i")

# extracts stats and adds them to fight dictionary
fight["victory_method"] = "".join(method.text.split()[1:])
fight["final_round"] = round.text.split()[1]
fight["finish_time"] = time.text.split()[1]
fight["number_of_rounds"] = format.text.split()[2]

# testing
print(fight["victory_method"])
print(fight["final_round"])
print(fight["finish_time"])
print(fight["number_of_rounds"])


print("\n")
print(fight)
```

```
Decision-Split
5
5:00
5
```

```
{'fighter_a_name': 'Robert Whittaker', 'fighter_b_name': 'Reinier de Ridder',
'weight_class': 'Middleweight', 'victory_method': 'Decision-Split',
'final_round': '5', 'finish_time': '5:00', 'number_of_rounds': '5'}
```

[196]:
```
# find the <i> tag that signifies the winner
winner_status = fight_soup.find("i",␣
 ↪class_="b-fight-details__person-status_style_green")

# get the parent <div class="b-fight-details__person">
winner_div = winner_status.find_parent("div", class_="b-fight-details__person")
```

```
# find the fighter name link inside the winner_div
winner_name_tag = winner_div.find("a", class_="b-link␣
 ↪b-fight-details__person-link")

# get the text and strip it
winner_name = winner_name_tag.text.strip()

if (winner_name==fight["fighter_a_name"]):
    fight["winner"] = "a"
else:
    fight["winner"] = "b"

print(fight)
```

```
{'fighter_a_name': 'Robert Whittaker', 'fighter_b_name': 'Reinier de Ridder',
'weight_class': 'Middleweight', 'victory_method': 'Decision-Split',
'final_round': '5', 'finish_time': '5:00', 'number_of_rounds': '5', 'winner':
'b'}
```

[197]:
```
def parse_fraction(value):
    """
    converts a string fraction in the format 'X of Y' into two integers.

    function is typically used to extract numerical values from strings
    like "12 of 25", which represent stats (e.g., 12 successful strikes
    out of 25 attempted). If the input string is not in the expected format,
    the function returns (0, 0).

    Args:
        value (str): A string representing a fraction in the form 'X of Y'.

    Returns:
        tuple: A tuple of two integers (X, Y). Returns (0, 0) if the input
        does not contain 'of' or is not properly formatted.
    """
    if 'of' in value:
        parts = value.split(' of ')
        return int(parts[0]), int(parts[1])
    return 0, 0

def parse_time_to_seconds(time_str):
    """
    converts a time string in the format 'MM:SS' to total seconds.

    this function is useful for parsing fight time durations that are commonly
    represented as minute-second strings. If the input does not contain a
    colon (':'), the function assumes it's malformed and returns 0.
```

```
    Args:
        time_str (str): A time string in the format 'MM:SS'.

    Returns:
        int: Total time in seconds. Returns 0 if the input is not in the␣
 ↪correct format.
    """
    if ':' in time_str:
        parts = time_str.split(':')
        return int(parts[0]) * 60 + int(parts[1])
    return 0

def parse_totals(totals:Tag,fight:Dict,*,fighter_a_id_:int,fighter_b_id_:
 ↪int,multiple_rows:bool=False):
    """
    Parses a row (or multiple rows) of total fight statistics and updates the␣
 ↪`fight` dictionary with structured data.

    This function extracts various statistical metrics for two fighters (A and␣
 ↪B) from a given HTML table row(s)
    representing total fight stats. These stats can include knockdowns,␣
 ↪significant strikes, takedowns,
    control time, and more.

    The function handles different formats of data:
    - Simple integers (e.g., knockdowns, submission attempts)
    - Percentages (e.g., takedown accuracy, significant strike accuracy)
    - "X of Y" formatted strings (e.g., total strikes, significant strikes)
    - Time strings (e.g., control time in MM:SS)

    Args:
        totals_row (Tag): A BeautifulSoup Tag representing the <tbody> or␣
 ↪parent table row containing fight stats.
        fight (Dict): A dictionary that will be updated with parsed stats for␣
 ↪fighter A and B.
        fighter_a_id_ (int): ID representing Fighter A
        fighter_b_id_ (int): ID representing Fighter B
        multiple_rows (bool, optional): Whether the table contains multiple␣
 ↪rows (e.g., per round).
                                        If True, a suffix (e.g., '_01') is␣
 ↪added to stat keys to differentiate them.

    Returns:
        None: The function updates the `fight` dictionary in-place with new␣
 ↪key-value pairs.
```

```
    """

    stat_names = ['kd', 'sig_str', 'sig_str_pct', 'total_str', 'td', 'td_pct',␣
↪'sub_att', 'rev', 'ctrl']

    rows = totals.find_all("tr")

    # iterates through the rows of the table (representing rounds typically)
    for r,row in enumerate(rows):
        # if there are multiple rows each row represents a round number
        # eg. the 3rd row would represent the 3rd round of the fight and
        # thus have the suffix _03
        suffix = f"_0{r}" if multiple_rows else ""

        # gets all the stat boxes, getting rid of the first column containing␣
↪names
        stat_box = row.find_all("td")[1:]

        for i, stat_name in enumerate(stat_names):
            if i < len(stat_box):
                stat_values = stat_box[i].find_all("p")

                fighter_a_value = stat_values[0].text.strip()
                fighter_b_value = stat_values[1].text.strip()

                # Parse different stat types
                if stat_name in ['kd', 'sub_att', 'rev']:
                    # Simple integers
                    fight[f'fighter_a_{stat_name}{suffix}'] =␣
↪int(fighter_a_value) if fighter_a_value.isdigit() else 0
                    fight[f'fighter_b_{stat_name}{suffix}'] =␣
↪int(fighter_b_value) if fighter_b_value.isdigit() else 0

                elif stat_name in ['sig_str_pct', 'td_pct']:
                    # Percentages - convert to float
                    f1_pct = fighter_a_value.replace('%', '').replace('---',␣
↪'0')
                    f2_pct = fighter_b_value.replace('%', '').replace('---',␣
↪'0')
                    fight[f'fighter_a_{stat_name}{suffix}'] = float(f1_pct) if␣
↪f1_pct.replace('.', '').isdigit() else 0.0
                    fight[f'fighter_b_{stat_name}{suffix}'] = float(f2_pct) if␣
↪f2_pct.replace('.', '').isdigit() else 0.0

                elif stat_name in ['sig_str', 'total_str', 'td']:
                    # "X of Y" format - extract both landed and attempted
```

```
                    fa_landed, fa_attempted = parse_fraction(fighter_a_value)
                    fb_landed, fb_attempted = parse_fraction(fighter_b_value)

                    fight[f'fighter_a_{stat_name}_landed{suffix}'] = fa_landed
                    fight[f'fighter_a_{stat_name}_attempted{suffix}'] =␣
↪fa_attempted
                    fight[f'fighter_b_{stat_name}_landed{suffix}'] = fb_landed
                    fight[f'fighter_b_{stat_name}_attempted{suffix}'] =␣
↪fb_attempted

                elif stat_name == 'ctrl':
                    fight[f'fighter_a_{stat_name}_seconds{suffix}'] =␣
↪parse_time_to_seconds(fighter_a_value)
                    fight[f'fighter_b_{stat_name}_seconds{suffix}'] =␣
↪parse_time_to_seconds(fighter_b_value)
```

[198]:
```
def parse_sig_strikes(sig_strike_element:Tag,fight:Dict,*,fighter_a_id_:
↪int,fighter_b_id_:int,multiple_rows:bool=False):
    """
    Parses a row (or multiple rows) of detailed significant strike distribution␣
↪statistics and updates the fight dictionary.

    This function processes the breakdown of significant strikes by target area␣
↪or position, such as head, body, leg,
    distance, clinch, and ground. It extracts the number of landed and␣
↪attempted strikes for each fighter in each category.

    Note:
        The overall significant strike totals ('sig_str' and 'sig_str_pct') are␣
↪assumed to be handled separately
        in the `parse_totals_row` function and are therefore skipped here.

    Args:
        sig_strike_element (Tag): A BeautifulSoup Tag object representing the␣
↪section of the HTML containing
                                    significant strike breakdown rows.
        fight (Dict): A dictionary to be updated with parsed significant strike␣
↪data for each fighter.
        fighter_a_id_ (int): Unique identifier for Fighter A
        fighter_b_id_ (int): Unique identifier for Fighter B
        multiple_rows (bool, optional): Indicates if multiple rows (e.g., per␣
↪round) are present. When True, a suffix
                                            like '_01', '_02' etc. is appended to␣
↪the stat keys to differentiate rounds.
```

```
    Returns:
        None: The function updates the `fight` dictionary in place with keys of␣
↪the format
              'fighter_a_{stat_name}\_landed',␣
↪'fighter_a_{stat_name}\_attempted', and similarly for fighter B,
              optionally with suffixes if multiple_rows is True.
    """

    # Skip 'sig_str' and 'sig_str_pct' as they were already processed in␣
↪parse_totals_row()
    stat_names = ["head","body","leg","distance","clinch","ground"]


    rows = sig_strike_element.find_all("tr")

    for r,row in enumerate(rows):
        # if there are multiple rows each row represents a round number
        # eg. the 3rd row would represent the 3rd round of the fight and
        # thus have the suffix _03
        suffix = f"_0{r}" if multiple_rows else ""

        # gets all the stat boxes, getting rid of the first column containing␣
↪names
        # and gets rid of 'sig_str' and 'sig_str_pct'
        stat_box = row.find_all("td")[3:]


        for i, stat_name in enumerate(stat_names):
            if i < len(stat_box):
                stat_values = stat_box[i].find_all("p")

                fighter_a_value = stat_values[0].text.strip()
                fighter_b_value = stat_values[1].text.strip()


                # "X of Y" format - extract both landed and attempted

                fa_landed, fa_attempted = parse_fraction(fighter_a_value)
                fb_landed, fb_attempted = parse_fraction(fighter_b_value)

                fight[f'fighter_a_{stat_name}_landed{suffix}'] = fa_landed
                fight[f'fighter_a_{stat_name}_attempted{suffix}'] = fa_attempted
                fight[f'fighter_b_{stat_name}_landed{suffix}'] = fb_landed
                fight[f'fighter_b_{stat_name}_attempted{suffix}'] = fb_attempted
```

[199]:
```
# Extract all fight statistics from UFC stats page into a dictionary

# Find the totals table
totals_table = None
sections = fight_soup.find_all("section", class_="b-fight-details__section␣
 ↪js-fight-section")

totals_table = sections[1]

parse_totals(totals_table,fight,fighter_a_id_=fighter_a_id,fighter_b_id_=fighter_b_id)

test = sections[2]

parse_totals(test,fight,fighter_a_id_=fighter_a_id,fighter_b_id_=fighter_b_id,multiple_rows=Tr



parse_sig_strikes(test.
 ↪find_next_sibling("table"),fight,fighter_a_id_=fighter_a_id,fighter_b_id_=fighter_b_id,␣
 ↪multiple_rows=False)

test2 = sections[4]

parse_sig_strikes(test2,fight,fighter_a_id_=fighter_a_id,fighter_b_id_=fighter_b_id,␣
 ↪multiple_rows=True)
```

[200]:
```
print(fight)
```

```
{'fighter_a_name': 'Robert Whittaker', 'fighter_b_name': 'Reinier de Ridder',
'weight_class': 'Middleweight', 'victory_method': 'Decision-Split',
'final_round': '5', 'finish_time': '5:00', 'number_of_rounds': '5', 'winner':
'b', 'fighter_a_kd': 1, 'fighter_b_kd': 0, 'fighter_a_sig_str_landed': 66,
'fighter_a_sig_str_attempted': 142, 'fighter_b_sig_str_landed': 67,
'fighter_b_sig_str_attempted': 145, 'fighter_a_sig_str_pct': 46.0,
'fighter_b_sig_str_pct': 46.0, 'fighter_a_total_str_landed': 70,
'fighter_a_total_str_attempted': 146, 'fighter_b_total_str_landed': 192,
'fighter_b_total_str_attempted': 282, 'fighter_a_td_landed': 0,
'fighter_a_td_attempted': 0, 'fighter_b_td_landed': 2, 'fighter_b_td_attempted':
15, 'fighter_a_td_pct': 0.0, 'fighter_b_td_pct': 13.0, 'fighter_a_sub_att': 0,
'fighter_b_sub_att': 0, 'fighter_a_rev': 0, 'fighter_b_rev': 0,
'fighter_a_ctrl_seconds': 36, 'fighter_b_ctrl_seconds': 546, 'fighter_a_kd_01':
0, 'fighter_b_kd_01': 0, 'fighter_a_sig_str_landed_01': 20,
'fighter_a_sig_str_attempted_01': 34, 'fighter_b_sig_str_landed_01': 15,
'fighter_b_sig_str_attempted_01': 35, 'fighter_a_sig_str_pct_01': 58.0,
'fighter_b_sig_str_pct_01': 42.0, 'fighter_a_total_str_landed_01': 21,
'fighter_a_total_str_attempted_01': 35, 'fighter_b_total_str_landed_01': 21,
'fighter_b_total_str_attempted_01': 42, 'fighter_a_td_landed_01': 0,
'fighter_a_td_attempted_01': 0, 'fighter_b_td_landed_01': 0,
'fighter_b_td_attempted_01': 3, 'fighter_a_td_pct_01': 0.0,
```

'fighter_b_td_pct_01': 0.0, 'fighter_a_sub_att_01': 0, 'fighter_b_sub_att_01':
0, 'fighter_a_rev_01': 0, 'fighter_b_rev_01': 0, 'fighter_a_ctrl_seconds_01': 0,
'fighter_b_ctrl_seconds_01': 12, 'fighter_a_kd_02': 0, 'fighter_b_kd_02': 0,
'fighter_a_sig_str_landed_02': 9, 'fighter_a_sig_str_attempted_02': 27,
'fighter_b_sig_str_landed_02': 16, 'fighter_b_sig_str_attempted_02': 38,
'fighter_a_sig_str_pct_02': 33.0, 'fighter_b_sig_str_pct_02': 42.0,
'fighter_a_total_str_landed_02': 9, 'fighter_a_total_str_attempted_02': 27,
'fighter_b_total_str_landed_02': 29, 'fighter_b_total_str_attempted_02': 58,
'fighter_a_td_landed_02': 0, 'fighter_a_td_attempted_02': 0,
'fighter_b_td_landed_02': 1, 'fighter_b_td_attempted_02': 3,
'fighter_a_td_pct_02': 0.0, 'fighter_b_td_pct_02': 33.0, 'fighter_a_sub_att_02':
0, 'fighter_b_sub_att_02': 0, 'fighter_a_rev_02': 0, 'fighter_b_rev_02': 0,
'fighter_a_ctrl_seconds_02': 0, 'fighter_b_ctrl_seconds_02': 124,
'fighter_a_kd_03': 1, 'fighter_b_kd_03': 0, 'fighter_a_sig_str_landed_03': 20,
'fighter_a_sig_str_attempted_03': 28, 'fighter_b_sig_str_landed_03': 13,
'fighter_b_sig_str_attempted_03': 30, 'fighter_a_sig_str_pct_03': 71.0,
'fighter_b_sig_str_pct_03': 43.0, 'fighter_a_total_str_landed_03': 21,
'fighter_a_total_str_attempted_03': 29, 'fighter_b_total_str_landed_03': 32,
'fighter_b_total_str_attempted_03': 51, 'fighter_a_td_landed_03': 0,
'fighter_a_td_attempted_03': 0, 'fighter_b_td_landed_03': 1,
'fighter_b_td_attempted_03': 2, 'fighter_a_td_pct_03': 0.0,
'fighter_b_td_pct_03': 50.0, 'fighter_a_sub_att_03': 0, 'fighter_b_sub_att_03':
0, 'fighter_a_rev_03': 0, 'fighter_b_rev_03': 0, 'fighter_a_ctrl_seconds_03':
32, 'fighter_b_ctrl_seconds_03': 159, 'fighter_a_kd_04': 0, 'fighter_b_kd_04':
0, 'fighter_a_sig_str_landed_04': 6, 'fighter_a_sig_str_attempted_04': 22,
'fighter_b_sig_str_landed_04': 14, 'fighter_b_sig_str_attempted_04': 24,
'fighter_a_sig_str_pct_04': 27.0, 'fighter_b_sig_str_pct_04': 58.0,
'fighter_a_total_str_landed_04': 8, 'fighter_a_total_str_attempted_04': 24,
'fighter_b_total_str_landed_04': 37, 'fighter_b_total_str_attempted_04': 48,
'fighter_a_td_landed_04': 0, 'fighter_a_td_attempted_04': 0,
'fighter_b_td_landed_04': 0, 'fighter_b_td_attempted_04': 3,
'fighter_a_td_pct_04': 0.0, 'fighter_b_td_pct_04': 0.0, 'fighter_a_sub_att_04':
0, 'fighter_b_sub_att_04': 0, 'fighter_a_rev_04': 0, 'fighter_b_rev_04': 0,
'fighter_a_ctrl_seconds_04': 4, 'fighter_b_ctrl_seconds_04': 116,
'fighter_a_kd_05': 0, 'fighter_b_kd_05': 0, 'fighter_a_sig_str_landed_05': 11,
'fighter_a_sig_str_attempted_05': 31, 'fighter_b_sig_str_landed_05': 9,
'fighter_b_sig_str_attempted_05': 18, 'fighter_a_sig_str_pct_05': 35.0,
'fighter_b_sig_str_pct_05': 50.0, 'fighter_a_total_str_landed_05': 11,
'fighter_a_total_str_attempted_05': 31, 'fighter_b_total_str_landed_05': 73,
'fighter_b_total_str_attempted_05': 83, 'fighter_a_td_landed_05': 0,
'fighter_a_td_attempted_05': 0, 'fighter_b_td_landed_05': 0,
'fighter_b_td_attempted_05': 4, 'fighter_a_td_pct_05': 0.0,
'fighter_b_td_pct_05': 0.0, 'fighter_a_sub_att_05': 0, 'fighter_b_sub_att_05':
0, 'fighter_a_rev_05': 0, 'fighter_b_rev_05': 0, 'fighter_a_ctrl_seconds_05': 0,
'fighter_b_ctrl_seconds_05': 135, 'fighter_a_head_landed': 62,
'fighter_a_head_attempted': 135, 'fighter_b_head_landed': 41,
'fighter_b_head_attempted': 105, 'fighter_a_body_landed': 4,
'fighter_a_body_attempted': 7, 'fighter_b_body_landed': 26,

'fighter_b_body_attempted': 40, 'fighter_a_leg_landed': 0,
'fighter_a_leg_attempted': 0, 'fighter_b_leg_landed': 0,
'fighter_b_leg_attempted': 0, 'fighter_a_distance_landed': 51,
'fighter_a_distance_attempted': 123, 'fighter_b_distance_landed': 54,
'fighter_b_distance_attempted': 128, 'fighter_a_clinch_landed': 4,
'fighter_a_clinch_attempted': 5, 'fighter_b_clinch_landed': 9,
'fighter_b_clinch_attempted': 11, 'fighter_a_ground_landed': 11,
'fighter_a_ground_attempted': 14, 'fighter_b_ground_landed': 4,
'fighter_b_ground_attempted': 6, 'fighter_a_head_landed_01': 19,
'fighter_a_head_attempted_01': 33, 'fighter_b_head_landed_01': 6,
'fighter_b_head_attempted_01': 19, 'fighter_a_body_landed_01': 1,
'fighter_a_body_attempted_01': 1, 'fighter_b_body_landed_01': 9,
'fighter_b_body_attempted_01': 16, 'fighter_a_leg_landed_01': 0,
'fighter_a_leg_attempted_01': 0, 'fighter_b_leg_landed_01': 0,
'fighter_b_leg_attempted_01': 0, 'fighter_a_distance_landed_01': 19,
'fighter_a_distance_attempted_01': 33, 'fighter_b_distance_landed_01': 10,
'fighter_b_distance_attempted_01': 30, 'fighter_a_clinch_landed_01': 1,
'fighter_a_clinch_attempted_01': 1, 'fighter_b_clinch_landed_01': 5,
'fighter_b_clinch_attempted_01': 5, 'fighter_a_ground_landed_01': 0,
'fighter_a_ground_attempted_01': 0, 'fighter_b_ground_landed_01': 0,
'fighter_b_ground_attempted_01': 0, 'fighter_a_head_landed_02': 8,
'fighter_a_head_attempted_02': 26, 'fighter_b_head_landed_02': 12,
'fighter_b_head_attempted_02': 32, 'fighter_a_body_landed_02': 1,
'fighter_a_body_attempted_02': 1, 'fighter_b_body_landed_02': 4,
'fighter_b_body_attempted_02': 6, 'fighter_a_leg_landed_02': 0,
'fighter_a_leg_attempted_02': 0, 'fighter_b_leg_landed_02': 0,
'fighter_b_leg_attempted_02': 0, 'fighter_a_distance_landed_02': 9,
'fighter_a_distance_attempted_02': 27, 'fighter_b_distance_landed_02': 15,
'fighter_b_distance_attempted_02': 36, 'fighter_a_clinch_landed_02': 0,
'fighter_a_clinch_attempted_02': 0, 'fighter_b_clinch_landed_02': 0,
'fighter_b_clinch_attempted_02': 0, 'fighter_a_ground_landed_02': 0,
'fighter_a_ground_attempted_02': 0, 'fighter_b_ground_landed_02': 1,
'fighter_b_ground_attempted_02': 2, 'fighter_a_head_landed_03': 20,
'fighter_a_head_attempted_03': 28, 'fighter_b_head_landed_03': 10,
'fighter_b_head_attempted_03': 24, 'fighter_a_body_landed_03': 0,
'fighter_a_body_attempted_03': 0, 'fighter_b_body_landed_03': 3,
'fighter_b_body_attempted_03': 6, 'fighter_a_leg_landed_03': 0,
'fighter_a_leg_attempted_03': 0, 'fighter_b_leg_landed_03': 0,
'fighter_b_leg_attempted_03': 0, 'fighter_a_distance_landed_03': 8,
'fighter_a_distance_attempted_03': 13, 'fighter_b_distance_landed_03': 10,
'fighter_b_distance_attempted_03': 26, 'fighter_a_clinch_landed_03': 1,
'fighter_a_clinch_attempted_03': 1, 'fighter_b_clinch_landed_03': 0,
'fighter_b_clinch_attempted_03': 0, 'fighter_a_ground_landed_03': 11,
'fighter_a_ground_attempted_03': 14, 'fighter_b_ground_landed_03': 3,
'fighter_b_ground_attempted_03': 4, 'fighter_a_head_landed_04': 6,
'fighter_a_head_attempted_04': 22, 'fighter_b_head_landed_04': 10,
'fighter_b_head_attempted_04': 18, 'fighter_a_body_landed_04': 0,
'fighter_a_body_attempted_04': 0, 'fighter_b_body_landed_04': 4,

'fighter_b_body_attempted_04': 6, 'fighter_a_leg_landed_04': 0,
'fighter_a_leg_attempted_04': 0, 'fighter_b_leg_landed_04': 0,
'fighter_b_leg_attempted_04': 0, 'fighter_a_distance_landed_04': 5,
'fighter_a_distance_attempted_04': 20, 'fighter_b_distance_landed_04': 12,
'fighter_b_distance_attempted_04': 21, 'fighter_a_clinch_landed_04': 1,
'fighter_a_clinch_attempted_04': 2, 'fighter_b_clinch_landed_04': 2,
'fighter_b_clinch_attempted_04': 3, 'fighter_a_ground_landed_04': 0,
'fighter_a_ground_attempted_04': 0, 'fighter_b_ground_landed_04': 0,
'fighter_b_ground_attempted_04': 0, 'fighter_a_head_landed_05': 9,
'fighter_a_head_attempted_05': 26, 'fighter_b_head_landed_05': 3,
'fighter_b_head_attempted_05': 12, 'fighter_a_body_landed_05': 2,
'fighter_a_body_attempted_05': 5, 'fighter_b_body_landed_05': 6,
'fighter_b_body_attempted_05': 6, 'fighter_a_leg_landed_05': 0,
'fighter_a_leg_attempted_05': 0, 'fighter_b_leg_landed_05': 0,
'fighter_b_leg_attempted_05': 0, 'fighter_a_distance_landed_05': 10,
'fighter_a_distance_attempted_05': 30, 'fighter_b_distance_landed_05': 7,
'fighter_b_distance_attempted_05': 15, 'fighter_a_clinch_landed_05': 1,
'fighter_a_clinch_attempted_05': 1, 'fighter_b_clinch_landed_05': 2,
'fighter_b_clinch_attempted_05': 3, 'fighter_a_ground_landed_05': 0,
'fighter_a_ground_attempted_05': 0, 'fighter_b_ground_landed_05': 0,
'fighter_b_ground_attempted_05': 0}