In [ ]:
```
import pandas as pd
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Below I import the UFC fight data and verify the columns present in the dataset.

In [ ]:
```
dataset = DataFrame(pd.read_csv(
    'C:/Users/Isaia/Documents/Programming/UFC Project/UFC_dataset_2/data.csv'))
dataset.columns
```

Out[ ]:
```
Index(['R_fighter', 'B_fighter', 'Referee', 'date', 'location', 'Winner',
       'title_bout', 'weight_class', 'B_avg_KD', 'B_avg_opp_KD',
       ...
       'R_win_by_Decision_Unanimous', 'R_win_by_KO/TKO', 'R_win_by_Submission',
       'R_win_by_TKO_Doctor_Stoppage', 'R_Stance', 'R_Height_cms',
       'R_Reach_cms', 'R_Weight_lbs', 'B_age', 'R_age'],
      dtype='object', length=144)
```

Below I define a method for displaying the winner of any fight in the dataset. However, it would require knowing the index of the fight (essentially useless unless first used by another function for some reason).

In [ ]:
```
def return_winner(fight):
    if fight["Winner"] == "Red":
        print(fight["R_fighter"] + " defeated " + fight["B_fighter"] + ".")
    elif fight["Winner"] == "Blue":
        print(fight["B_fighter"] + " defeated " + fight["R_fighter"] + ".")
    elif fight["Winner"] == "Draw":
        print ("The fight between " + fight["R_fighter"] + " and " + fight["B_fighter"]

return_winner(dataset.iloc[1004])
```

```
Beneil Dariush defeated Drew Dober.
```

In order to know how many fights there are to iterate through, and how many attributes we may have accesss to, we can use the following functions.

In [ ]:
```
print(dataset.shape) #Prints the number of rows and columns.
print(dataset.shape[0])  # Prints the number of rows.
print(dataset.shape[0])  # Prints the number of columns.
print(len(dataset)) #Another way to get the number of entries (rows).
```

```
(6012, 144)
6012
6012
6012
```

The function below can take two fighters' names as input and return who won. However, a number of fighters have competed against one another more than once, so I decided to show tallies of their wins over each other next to their names.

In [ ]:
```
def winner_of_known_contestants(fighter1, fighter2):
```

```
    x = 0               # A variable for the while loop.
    fight_count = 0     # This tallies the number of times the opponents have fought.
    f1_wins = 0         # The number of wins fighter1 has over fighter2.
    f2_wins = 0         # The number of wins fighter2 has over fighter1.
    draws = 0           # The number of draws they have had.

    while x < len(dataset): # Using this function instead of a constant allows the for
        if (dataset.iloc[x]["R_fighter"].lower() == fighter1.lower() and dataset.iloc[x
            or (dataset.iloc[x]["R_fighter"].lower() == fighter2.lower() and dataset.il

            fight_count += 1
            # If the winner was in the red corner and the first fighter the user named,
            if dataset.iloc[x]['Winner'] == "Red" and dataset.iloc[x]["R_fighter"].lowe
                    f1_wins += 1
            # If the winner was in the blue corner and the first fighter the user named
            elif dataset.iloc[x]['Winner'] == "Blue" and dataset.iloc[x]["B_fighter"].l
                    f1_wins += 1
                # If the winner was in the red corner and the second fighter the user n
            elif dataset.iloc[x]['Winner'] == "Red" and dataset.iloc[x]["R_fighter"].lo
                    f2_wins += 1
            # If the winner was in the blue corner and the second fighter the user name
            elif dataset.iloc[x]['Winner'] == "Blue" and dataset.iloc[x]["B_fighter"].l
                    f2_wins += 1
            # If the fight was a draw, add one to the tally of draws between them.
            else:
                    draws += 1
                    # This dataset is flawed in that "no contests" are displayed as dra
        x += 1

    # Here, I respond if no such fight was found.
    if fight_count == 0:
        print("Either these fighters have not fought, or you should check your spelling
    # Here I just give Uncle Chael his due.
    elif fight_count == 1 and (fighter1.lower() == "chael sonnen" or fighter2.lower() =
        print("Another win for the Bad Guy!")
    elif fight_count > 1 and (fighter1.lower() == "chael sonnen" or fighter2.lower() ==
        print("Classic victories by the Bad Guy!")
    # If a fight did take place, I display each fighter's name, and his/her wins over t
    elif fight_count > 0 and draws == 0:
            print(str(fighter1) + " - " + str(f1_wins) +
                  ", " + str(fighter2) + " - " + str(f2_wins))
    # This code is used to display a draw, if it occurred.
    elif fight_count > 0 and draws > 0:
            print(str(fighter1) + " - " + str(f1_wins) + ". " +
                  str(fighter2) + " - " + str(f2_wins) + ". " + "Draws - " + str(draws)
```

In [ ]:

```
winner_of_known_contestants("Chris Weidman", "Anderson Silva")
winner_of_known_contestants("Luke Rockhold", "Lyoto Machida")
winner_of_known_contestants("Jon Jones", "Chael Sonnen")
```

```
Chris Weidman - 2, Anderson Silva - 0
Luke Rockhold - 1, Lyoto Machida - 0
Another win for the Bad Guy!
```

In [ ]:

```
# This function returns the total number of fights fighters of a given stance have had.
# I decided to make this function redundant.
```

```
""" def stance_total(stance):
    x = 0
    tally = 0

    while x < len(dataset):
        if dataset.iloc[x]['B_Stance'] == stance:
            tally += 1
        elif dataset.iloc[x]['R_Stance'] == stance:
            tally += 1
        x += 1
    return tally """
```

Out[ ]:
```
" def stance_total(stance):\n    x = 0\n    tally = 0\n\n    while x < len(dataset):\n
if dataset.iloc[x]['B_Stance'] == stance:\n            tally += 1\n        elif dataset.
iloc[x]['R_Stance'] == stance:\n            tally += 1\n        x += 1\n    return tally
"
```

How does each stance perform against the others? Let's find out.

In [ ]:
```
def find_best_stance():
    O_wins = 0  # This tallies the wins for orthodox fighters. No contests EXCLUDED.
    SP_wins = 0  # Southpaw fighters.
    Switch_wins = 0 # Switch fighters.
    OS_wins = 0 # Open stance fighters.

    x = 0
    O_fights = 0 # This tallies the number of fights against people of different stance
    SP_fights = 0 # Same for southpaw fighters.
    Switch_fights = 0 # Same for switch fighters.
    OS_fights = 0 # Same for open stance fighters.

    while x < len(dataset):
        bluestance = dataset.iloc[x]['B_Stance']
        red_stance = dataset.iloc[x]['R_Stance']
        winner = dataset.iloc[x]['Winner']
        #SP vs. orthodox group.
        if bluestance == 'Southpaw' and red_stance == 'Orthodox':
            O_fights += 1
            SP_fights += 1
            if winner == "Blue":
                SP_wins += 1
            elif winner == "Red":
                O_wins += 1
        elif red_stance == 'Southpaw' and bluestance == 'Orthodox':
            O_fights += 1
            SP_fights += 1
            if winner == "Blue":
                O_wins += 1
            elif winner == "Red":
                SP_wins += 1
        #SP vs. switch group.
        elif red_stance == 'Southpaw' and bluestance == 'Switch':
            SP_fights += 1
            Switch_fights += 1
            if winner == "Blue":
                Switch_wins += 1
            elif winner == "Red":
                SP_wins += 1
        elif bluestance == 'Southpaw' and red_stance == 'Switch':
```

```
        SP_fights += 1
        Switch_fights += 1
        if winner == "Blue":
            SP_wins += 1
        elif winner == "Red":
            Switch_wins += 1
    #SP vs. open stance group.
    elif red_stance == 'Southpaw' and bluestance == 'Open Stance':
        SP_fights += 1
        OS_fights += 1
        if winner == "Blue":
            OS_wins += 1
        elif winner == "Red":
            SP_wins += 1
    elif bluestance == 'Southpaw' and red_stance == 'Open Stance':
        SP_fights += 1
        OS_fights += 1
        if winner == "Blue":
            SP_wins += 1
        elif winner == "Red":
            OS_wins += 1
    #Open stance vs. orthodox group.
    elif red_stance == 'Orthodox' and bluestance == 'Open Stance':
        O_fights += 1
        OS_fights += 1
        if winner == "Blue":
            OS_wins += 1
        elif winner == "Red":
            O_wins += 1
    elif bluestance == 'Orthodox' and red_stance == 'Open Stance':
        O_fights += 1
        OS_fights += 1
        if winner == "Blue":
            O_wins += 1
        elif winner == "Red":
            OS_wins += 1
    #Open stance vs. switch stance group.
    elif red_stance == 'Switch' and bluestance == 'Open Stance':
        OS_fights += 1
        Switch_fights += 1
        if winner == "Blue":
            OS_wins += 1
        elif winner == "Red":
            Switch_wins += 1
    elif bluestance == 'Switch' and red_stance == 'Open Stance':
        OS_fights += 1
        Switch_fights += 1
        if winner == "Blue":
            Switch_wins += 1
        elif winner == "Red":
            OS_wins += 1
    # Switch vs. Orthodox stance group.
    elif red_stance == 'Switch' and bluestance == 'Orthodox':
        O_fights += 1
        Switch_fights += 1
        if winner == "Blue":
            O_wins += 1
        elif winner == "Red":
            Switch_wins += 1
    elif bluestance == 'Switch' and red_stance == 'Orthodox':
```

```
            O_fights += 1
            Switch_fights += 1
            if winner == "Blue":
                Switch_wins += 1
            elif winner == "Red":
                O_wins += 1
        x += 1

    O_win_rate = str(round((O_wins/O_fights)*100)) + '%'
    Sp_win_rate = str(round((SP_wins/SP_fights)*100)) + '%'
    OS_win_rate = str(round((OS_wins/OS_fights)*100)) + '%'
    Switch_win_rate = str(round((Switch_wins/Switch_fights)*100)) + '%'

    print("\nOrthodox win rate: " + O_win_rate + "\nSouthpaw win rate: " + Sp_win_rate
    + "\nSwitch win rate: " + OS_win_rate + "\nOpen stance win rate: " + Switch_win_rat
```

In [ ]:

```
print("The average win rates of competitors with one stance against competitors of the
output_best_stance = find_best_stance()
print(output_best_stance)
print("\nNote that 'no contests' are wins for neither party, so the rates do not add up
print("\nNote also that there are far more orthodox fighters than fighters of other sta
```

```
The average win rates of competitors with one stance against competitors of the other st
ances

Orthodox win rate: 46%
Southpaw win rate: 52%
Switch win rate: 57%
Open stance win rate: 51%
None

Note that 'no contests' are wins for neither party, so the rates do not add up precisel
y.

Note also that there are far more orthodox fighters than fighters of other stances.
```

The function below can compare any two stances. For some background: typically, right-handed fighters fight in the 'orthodox' stance (90% of the population). This makes southpaws rare, and it is well-known that while southpaws are used to dealing with orthodox fighters, orthodox fighters are often not used to dealing with southpaw fighters.

'Switch' and 'open stance' fighters are very rare in this dataset. There is pressure in many gyms to stick with one stance, rather than 'trying to be fancy' by switching stances at will; but this is changing, and there is definitely also a trend of fighters learning to fight proficiently in both stances. Fighters who do have the confidence to switch stances as a matter of course are often very confident strikers, often for good reason. Finally, I will note that the UFC often does not label fighters who do in fact switch stances very often as switch-stance fighters, so the data are certainly not particularly accurate in this regard.

In [ ]:

```
def compare_stances(stance1, stance2):
    x = 0
    st1_wins = 0
    st2_wins = 0
    st1_w_rate = 0
```

```
    st2_w_rate = 0
    fights = 0 # Tallys the number of fights between the two stances. Note that this IN
    while x < len(dataset):
        bluestance = dataset.iloc[x]['B_Stance']
        red_stance = dataset.iloc[x]['R_Stance']
        winner = dataset.iloc[x]['Winner']
        #.
        if bluestance == stance1 and red_stance == stance2:
            fights += 1
            if winner == "Blue":
                st1_wins += 1
            elif winner == "Red":
                st2_wins += 1
        elif red_stance == stance1 and bluestance == stance2:
            fights += 1
            if winner == "Blue":
                st2_wins += 1
            elif winner == "Red":
                st1_wins += 1
        x += 1
    if fights != 0:
        st1_w_rate = round(st1_wins/fights,5)
        st2_w_rate = round(st2_wins/fights,5) # I return the win rates of both because
    return st1_w_rate,st2_w_rate
```

In [ ]:

```
print(compare_stances('Orthodox', 'Southpaw'))
```

```
(0.45734, 0.52218)
```

Reach

In [ ]:

```
def reach_of_winners(weightclass):
  x = 0
  reach_difference_total = 0.0
  count = 0.0

  while x < len(dataset):
    blue_reach = dataset.iloc[x]['B_Reach_cms']
    red_reach = dataset.iloc[x]['R_Reach_cms']
    if dataset.iloc[x]['weight_class'] != weightclass:
      x += 1
      continue
    # I run this check to ignore rows with missing values.
    if np.isnan(blue_reach) == True or  np.isnan(red_reach) == True:
      x += 1
      continue

    if dataset.iloc[x]['Winner'] == 'Blue':
      reach_difference_total += (blue_reach - red_reach)
      count += 1
      x += 1
    elif dataset.iloc[x]['Winner'] == 'Red':
      reach_difference_total += (red_reach - blue_reach)
      x += 1
      count += 1
    else:
      x += 1
```

```
    # This ensures that if there are no data, there isn't a dividing by 0 error.
    # I know this only applies to catchweight, but I'm pretending I don't know for demo
  if count == 0:
    return 'No relevant data'
  else:
    return round((reach_difference_total/count), 2)
```

The following results show the reach advantage that the winner had over the loser on average, for a given weight class. Note that there are negative values.

In [ ]:
```
for weightclass in dataset['weight_class'].unique():
  print(weightclass + ": " + str(reach_of_winners(weightclass)))
```

```
Bantamweight: -0.58
Middleweight: 0.59
Heavyweight: 1.84
WomenStrawweight: -0.92
WomenBantamweight: 1.04
Lightweight: 0.39
Welterweight: 0.45
Flyweight: -0.11
LightHeavyweight: 1.8
Featherweight: 0.47
WomenFlyweight: 0.53
WomenFeatherweight: 1.11
CatchWeight: -0.07
OpenWeight: No relevant data
```

Although the data are incomplete and have some issues, the results are quite interesting, in that the mean difference is very small. The strength of the correlation and statistical significance are irrelevant, since the differences are so small anyway.

What I thought would be interesting to do next is fine the average difference in reach for opponents of each weight class.

In [ ]:
```
def average_reach_diff(weightclass):
  x = 0
  reach_difference = 0.0
  count = 0.0

  while x < len(dataset):
    blue_reach = dataset.iloc[x]['B_Reach_cms']
    red_reach = dataset.iloc[x]['R_Reach_cms']
    if dataset.iloc[x]['weight_class'] != weightclass:
      x += 1
      continue
    # I run this check to ignore rows with missing values.
    if np.isnan(blue_reach) == True or  np.isnan(red_reach) == True:
      x += 1
      continue

    if blue_reach > red_reach:
      reach_difference += (blue_reach - red_reach)
      count += 1
```

```
      x += 1
    elif red_reach > blue_reach:
      reach_difference += (red_reach - blue_reach)
      count += 1
      x += 1
    else:
      count += 1
      x += 1

    # This ensures that if there are no data, there isn't a dividing by 0 error.
    # I know this only applies to catchweight, but I'm pretending I don't know for demo
  if count == 0:
    return 'No relevant data'
  else:
    return round((reach_difference/count), 2)
```

In [ ]:

```
for weightclass in dataset['weight_class'].unique():
  print(weightclass + ": " + str(average_reach_diff(weightclass)))
```

```
Bantamweight: 7.09
Middleweight: 6.46
Heavyweight: 8.28
WomenStrawweight: 5.6
WomenBantamweight: 5.55
Lightweight: 5.93
Welterweight: 6.42
Flyweight: 5.25
LightHeavyweight: 6.57
Featherweight: 6.31
WomenFlyweight: 6.49
WomenFeatherweight: 6.51
CatchWeight: 6.77
OpenWeight: No relevant data
```

The data show that the average reach difference for every weight class is substantial, which I was already aware of. Next, I want to see whether there is a correlation between winning and having the reach advantage, no matter how small.

In [ ]:

```
def win_rate_reach_adv(weightclass):
  x = 0
  win_count = 0.0
  count = 0.0

  while x < len(dataset):
    blue_reach = dataset.iloc[x]['B_Reach_cms']
    red_reach = dataset.iloc[x]['R_Reach_cms']
    if dataset.iloc[x]['weight_class'] != weightclass:
      x += 1
      continue
    if dataset.iloc[x]['weight_class'] == 'Openweight':
      x += 1
      continue
    # I run this check to ignore rows with missing values.
    if np.isnan(blue_reach) == True or  np.isnan(red_reach) == True:
      x += 1
      continue
```

```
    if blue_reach > red_reach and dataset.iloc[x]['Winner'] == "Blue":
      win_count += 1
      count += 1
      x += 1
    elif blue_reach > red_reach and dataset.iloc[x]['Winner'] == "Red":
      count += 1
      x += 1
    elif red_reach > blue_reach and dataset.iloc[x]['Winner'] == "Red":
      win_count += 1
      count += 1
      x += 1
    elif red_reach > blue_reach and dataset.iloc[x]['Winner'] == "Blue":
      count += 1
      x += 1
    else:
      # This accounts for no contests. To exclude no contests, simple remove the line d
      count += 1
      x += 1
    return round(((win_count/count)*100), 2)
```

In [ ]:

```
for weightclass in dataset['weight_class'].unique():
  print(weightclass + ": " + str(win_rate_reach_adv(weightclass)) + '%')
```

```
Bantamweight: 42.76%
Middleweight: 46.94%
Heavyweight: 51.22%
WomenStrawweight: 35.23%
WomenBantamweight: 46.51%
Lightweight: 42.15%
Welterweight: 44.01%
Flyweight: 42.18%
LightHeavyweight: 48.18%
Featherweight: 44.78%
WomenFlyweight: 42.59%
WomenFeatherweight: 43.75%
CatchWeight: 44.44%
OpenWeight: No relevant data%
```

It is well known that a reach advantage provides some benefits to the striking aspect of MMA, but wrestling is more often than not the determining factor. Therefore, looking at method of victory, takedowns, etc., in relation to the height advantage, could also be a little interesting. I suspect that this slight advantage for the shorter fighter comes from the fact that wrestling often beats striking, and shorter fighters may be or feel obligated to wrestle when up against tall, skilled strikers.