import pandas as pd

# Load the dataset

file\_path = 'C:/Users/batuh/Desktop/ufc-fighters-statistics.csv'

ufc\_data = pd.read\_csv(file\_path)

# Selecting relevant columns

relevant\_columns = ['height\_cm', 'weight\_in\_kg', 'reach\_in\_cm', 'wins', 'losses',

'significant\_strikes\_landed\_per\_minute', 'significant\_striking\_accuracy',

'average\_takedowns\_landed\_per\_15\_minutes', 'takedown\_accuracy']

# Creating a new DataFrame with only relevant columns

ufc\_analysis = ufc\_data[relevant\_columns]

# Dropping rows with missing values in the relevant columns

ufc\_analysis\_clean = ufc\_analysis.dropna()

import seaborn as sns

import matplotlib.pyplot as plt

# Calculating the correlation matrix

correlation\_matrix = ufc\_analysis\_clean.corr()

# Plotting the correlation matrix

plt.figure(figsize=(12, 8))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Correlation Matrix of UFC Fighters' Attributes and Performance Metrics")

plt.show()

stances = ufc\_data['stance'].dropna().unique()

# Initializing an empty list to store performance data for each stance

performance\_data = []

for stance in stances:

stance\_data = ufc\_data[ufc\_data['stance'] == stance]

performance\_mean = stance\_data[performance\_metrics].mean()

performance\_mean.name = stance

performance\_data.append(performance\_mean)

# Converting the list of Series into a DataFrame

stance\_performance = pd.concat(performance\_data, axis=1).transpose()

# Plotting the performance metrics for different stances

stance\_performance.plot(kind='bar', figsize=(15, 8))

plt.title('Comparison of Performance Metrics Across Different Stances')

plt.ylabel('Average Metrics')

plt.xlabel('Stance')

plt.xticks(rotation=45)

plt.show()

# Analyzing the impact of striking accuracy on fight outcomes (wins)

plt.figure(figsize=(8, 5))

sns.scatterplot(x='significant\_striking\_accuracy', y='wins', data=ufc\_data)

plt.title('Impact of Striking Accuracy on Fight Outcomes (Wins)')

plt.xlabel('Significant Striking Accuracy')

plt.ylabel('Wins')

plt.show()

# Visualizing Takedown Defense and Takedown Accuracy

# Filtering out irrelevant or extreme values (e.g., takedown defense above 100%)

filtered\_ufc\_data = ufc\_data[(ufc\_data['takedown\_defense'] <= 100) & (ufc\_data['takedown\_accuracy'] <= 100)]

# Scatter plot for Takedown Defense vs. Takedown Accuracy

plt.figure(figsize=(10, 6))

sns.scatterplot(x='takedown\_accuracy', y='takedown\_defense', data=filtered\_ufc\_data)

plt.title('Takedown Defense vs. Takedown Accuracy')

plt.xlabel('Takedown Accuracy (%)')

plt.ylabel('Takedown Defense (%)')

plt.show()

# Analyzing Striking Accuracy and Defense

# Scatter plot for Significant Striking Accuracy vs. Significant Strike Defense

plt.figure(figsize=(10, 6))

sns.scatterplot(x='significant\_striking\_accuracy', y='significant\_strike\_defence', data=ufc\_data)

plt.title('Significant Striking Accuracy vs. Strike Defense')

plt.xlabel('Significant Striking Accuracy (%)')

plt.ylabel('Significant Strike Defence (%)')

plt.show()

# Correlation between Takedown Accuracy and Takedown Defense

takedown\_corr = filtered\_ufc\_data[['takedown\_accuracy', 'takedown\_defense']].corr()

# Displaying the correlation

takedown\_corr

import pandas as pd

# Load the dataset

ufc\_data\_path = 'C:/Users/batuh/Desktop/ufc.csv'

ufc\_data = pd.read\_csv(ufc\_data\_path)

# Unifying fighter statistics into individual records

fighter\_1\_data = ufc\_data[['Fighter 1', 'Fighter\_1\_KD', 'Fighter\_1\_STR', 'Weight\_Class', 'Method', 'Date']].copy()

fighter\_2\_data = ufc\_data[['Fighter 2', 'Fighter\_2\_KD', 'Fighter\_2\_STR', 'Weight\_Class', 'Method', 'Date']].copy()

# Renaming columns to unify

fighter\_1\_data.columns = ['Fighter', 'KD', 'STR', 'Weight\_Class', 'Method', 'Date']

fighter\_2\_data.columns = ['Fighter', 'KD', 'STR', 'Weight\_Class', 'Method', 'Date']

# Create a new column 'Fighter\_Name' to store fighter names

ufc\_data['Fighter\_Name\_1'] = ufc\_data['Fighter 1']

ufc\_data['Fighter\_Name\_2'] = ufc\_data['Fighter 2']

# Combining the datasets back into one

combined\_data = pd.concat([fighter\_1\_data, fighter\_2\_data], axis=0)

# Filtering relevant weight classes

relevant\_weight\_classes = ['Lightweight', 'Welterweight', 'Middleweight', 'Light Heavyweight', 'Heavyweight']

combined\_data = combined\_data[combined\_data['Weight\_Class'].isin(relevant\_weight\_classes)]

# Calculating KO/TKO wins for each fighter

combined\_data['KO/TKO\_Win'] = combined\_data['Method'].apply(lambda x: 1 if 'KO/TKO' in x else 0)

fighter\_stats = combined\_data.groupby('Fighter').agg(

Total\_KD=('KD', 'sum'),

Total\_STR=('STR', 'sum'),

KO\_TKO\_Wins=('KO/TKO\_Win', 'sum'),

Fights=('Date', 'count')

)

# Filtering based on criteria: more than 100 significant strikes and at least 5 KO/TKO wins

filtered\_fighter\_stats = fighter\_stats[(fighter\_stats['Total\_STR'] > 140) & (fighter\_stats['KO\_TKO\_Wins'] >= 7)]

# Create a new column 'Fighter\_Name' to store fighter names

filtered\_fighter\_stats['Fighter\_Name'] = filtered\_fighter\_stats.index

# Create Punch\_Power\_Score and Adjusted\_Score columns

filtered\_fighter\_stats['Punch\_Power\_Score'] = filtered\_fighter\_stats['Total\_KD'] / filtered\_fighter\_stats['Total\_STR']

filtered\_fighter\_stats['Adjusted\_Score'] = filtered\_fighter\_stats['Punch\_Power\_Score'] \* (1 + 0.01 \* (filtered\_fighter\_stats['Fights'] - 1))

# Determine the most fought weight class for each fighter

most\_frequent\_weight\_class = combined\_data.groupby('Fighter')['Weight\_Class'].agg(lambda x: x.mode().iloc[0])

# Joining the most frequent weight class with the filtered fighter stats

filtered\_fighter\_stats = filtered\_fighter\_stats.join(most\_frequent\_weight\_class, on='Fighter', how='left').rename(columns={'Weight\_Class': 'Most\_Frequent\_Weight\_Class'})

# Ranking fighters within each weight class based on their adjusted punch power score

ranked\_fighters\_per\_class = filtered\_fighter\_stats.groupby('Most\_Frequent\_Weight\_Class').apply(

lambda x: x.nlargest(5, 'Adjusted\_Score')

).reset\_index(drop=True)

# Display the top 5 power punchers in each specified division with fighter names

ranked\_fighters\_per\_class[ranked\_fighters\_per\_class['Most\_Frequent\_Weight\_Class'].isin(relevant\_weight\_classes)]

# Ensure all 'Time' values are strings to handle them uniformly

data['Time'] = data['Time'].astype(str)

# Calculate total fight duration in minutes

def calculate\_total\_minutes(row):

try:

rounds = (row['Round'] - 1) \* 5

time\_parts = row['Time'].split(':')

minutes = rounds + int(time\_parts[0]) + int(time\_parts[1]) / 60 if len(time\_parts) == 2 else rounds

except:

minutes = 0 # Default to 0 in case of any issue

return minutes

data['Total\_Minutes'] = data.apply(calculate\_total\_minutes, axis=1)

# Filtering data for the specified weight classes

weight\_classes\_of\_interest = ['Lightweight', 'Welterweight', 'Middleweight', 'Light Heavyweight', 'Heavyweight']

data\_filtered = data[data['Weight\_Class'].isin(weight\_classes\_of\_interest)]

# Initialize DataFrame to store durability scores

durability\_scores = pd.DataFrame()

# Calculate durability scores with adjusted criteria

for index, row in data\_filtered.iterrows():

for fighter\_num in [1, 2]:

opponent\_num = 1 if fighter\_num == 2 else 2

fighter = row[f'Fighter {fighter\_num}']

weight\_class = row['Weight\_Class']

won = row['Winner'] == fighter

# Adjust criteria: Focus on knockdowns and takedowns absorbed, significant strikes absorbed

knockdowns\_absorbed = row[f'Fighter\_{opponent\_num}\_KD']

takedowns\_absorbed = row[f'Fighter\_{opponent\_num}\_TD']

sig\_strikes\_absorbed = row[f'Fighter\_{opponent\_num}\_STR']

# Adjust fight duration score based on fight outcome

fight\_duration = row['Total\_Minutes'] \* (0.5 if not won else 1)

# Simplified calculation for durability score

durability\_score = fight\_duration + (knockdowns\_absorbed + takedowns\_absorbed) \* 2 - sig\_strikes\_absorbed \* 0.5

# Append to DataFrame

new\_row = {

'Fighter': fighter,

'Weight\_Class': weight\_class,

'Durability\_Score': durability\_score

}

durability\_scores = pd.concat([durability\_scores, pd.DataFrame([new\_row])], ignore\_index=True)

# Aggregate scores by fighter and weight class

durability\_scores\_aggregated = durability\_scores.groupby(['Fighter', 'Weight\_Class']).agg({'Durability\_Score': 'sum'}).reset\_index()

# Find top 5 durable fighters in each weight class

top\_durable\_fighters\_per\_class = durability\_scores\_aggregated.groupby('Weight\_Class').apply(lambda x: x.nlargest(5, 'Durability\_Score')).reset\_index(drop=True)

top\_durable\_fighters\_per\_class

- import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load data

ufc\_data = pd.read\_csv("C:/Users/batuh/Desktop/ufc-fighters-statistics.csv")

# Handle missing reach data

average\_reach\_height\_difference = ufc\_data['reach\_in\_cm'].mean() - ufc\_data['height\_cm'].mean()

ufc\_data['reach\_in\_cm'].fillna(ufc\_data['height\_cm'] + average\_reach\_height\_difference, inplace=True)

# Function to analyze and plot for each weight class

def analyze\_plot\_class(data, lower\_bound, upper\_bound, class\_name):

class\_data = data[(data['weight\_in\_kg'] > lower\_bound) & (data['weight\_in\_kg'] <= upper\_bound)]

correlation = class\_data['reach\_in\_cm'].corr(class\_data['wins'])

# Plot

sns.set(style="whitegrid")

plt.figure(figsize=(8, 6))

sns.scatterplot(data=class\_data, x='reach\_in\_cm', y='wins')

sns.regplot(data=class\_data, x='reach\_in\_cm', y='wins', ci=None, color='blue', line\_kws={'color':'red'})

plt.title(f"Correlation between Reach and Wins in {class\_name}: {round(correlation, 2)}")

plt.xlabel("Reach (cm)")

plt.ylabel("Wins")

plt.show()

# Plot for each weight class

analyze\_plot\_class(ufc\_data, 93.5, float('inf'), "Heavyweight")

analyze\_plot\_class(ufc\_data, 92, 93.4, "Light Heavyweight")

analyze\_plot\_class(ufc\_data, 83, 84, "Middleweight")

analyze\_plot\_class(ufc\_data, 77, 78, "Welterweight")

analyze\_plot\_class(ufc\_data, 70, 71, "Lightweight")

analyze\_plot\_class(ufc\_data, 65, 67, "Featherweight")

analyze\_plot\_class(ufc\_data, 60, 62, "Bantamweight")

analyze\_plot\_class(ufc\_data, 55, 57, "Flyweight")

import pandas as pd

# Load the data

file\_path = 'C:/Users/batuh/Desktop/ufc-fighters-statistics.csv'

data = pd.read\_csv(file\_path)

# Display the first few rows of the dataframe to understand its structure

data.head()

# Filter the dataset for relevant columns and remove rows with missing stance or outcome data

filtered\_data = data[['name', 'stance', 'wins', 'losses', 'draws', 'significant\_striking\_accuracy']].dropna(subset=['stance'])

# Calculate total fights and win rate for each fighter

filtered\_data['total\_fights'] = filtered\_data['wins'] + filtered\_data['losses'] + filtered\_data['draws']

filtered\_data['win\_rate'] = filtered\_data['wins'] / filtered\_data['total\_fights']

# Group by stance to calculate average win rate and average striking accuracy

stance\_stats = filtered\_data.groupby('stance').agg(

average\_win\_rate=('win\_rate', 'mean'),

average\_striking\_accuracy=('significant\_striking\_accuracy', 'mean'),

total\_fighters=('name', 'count')

).reset\_index()

stance\_stats.sort\_values(by='average\_win\_rate', ascending=False)

import seaborn as sns

import matplotlib.pyplot as plt

# Plotting the relationship between significant striking accuracy and win rate

plt.figure(figsize=(10, 6))

sns.scatterplot(data=filtered\_data, x='significant\_striking\_accuracy', y='win\_rate', hue='stance', alpha=0.6)

plt.title('Relationship between Significant Striking Accuracy and Win Rate by Stance')

plt.xlabel('Significant Striking Accuracy (%)')

plt.ylabel('Win Rate')

plt.legend(title='Stance')

plt.grid(True)

plt.show()

import pandas as pd

from scipy.stats import pearsonr, linregress

import matplotlib.pyplot as plt

import numpy as np

# Load the dataset

data = pd.read\_csv(r"C:\Users\batuh\Desktop\ufc-fighters-statistics.csv")

# Unique stances

stances = data['stance'].unique()

# Calculating the correlation coefficient and linear regression for each stance

for stance in stances:

stance\_data = data[data['stance'] == stance].copy() # Create a copy to avoid SettingWithCopyWarning

stance\_data['win\_rate'] = stance\_data['wins'] / (stance\_data['wins'] + stance\_data['losses'])

# Check if there are at least two data points

if len(stance\_data) > 1:

correlation, \_ = pearsonr(stance\_data['significant\_striking\_accuracy'], stance\_data['win\_rate'])

print(f'Correlation coefficient for {stance} stance: {correlation:.3f}')

# ... (rest of your linear regression code)

else:

print(f'Not enough data to calculate correlation or perform regression for {stance} stance.')

# Linear regression and plotting

for stance in stances:

stance\_data = data[data['stance'] == stance]

stance\_data['win\_rate'] = stance\_data['wins'] / (stance\_data['wins'] + stance\_data['losses'])

if len(stance\_data) > 1:

slope, intercept, r\_value, p\_value, std\_err = linregress(stance\_data['significant\_striking\_accuracy'], stance\_data['win\_rate'])

# Plotting the regression line

plt.scatter(stance\_data['significant\_striking\_accuracy'], stance\_data['win\_rate'], label=f'{stance} stance')

line = slope \* stance\_data['significant\_striking\_accuracy'] + intercept

plt.plot(stance\_data['significant\_striking\_accuracy'], line, label=f'{stance} Regression Line')

print(f'Linear regression for {stance} stance: Slope = {slope:.3f}, Intercept = {intercept:.3f}, '

f'R-squared = {r\_value\*\*2:.3f}, p-value = {p\_value:.3g}')

else:

print(f'Not enough data to perform linear regression for {stance} stance.')

plt.xlabel('Significant Striking Accuracy')

plt.ylabel('Win Rate')

plt.title('Linear Regression Analysis by Stance')

plt.legend()

plt.show()

# Group statistics for each stance

for stance in stances:

stance\_data = data[data['stance'] == stance]

stance\_data['win\_rate'] = stance\_data['wins'] / (stance\_data['wins'] + stance\_data['losses'])

if len(stance\_data) > 0:

mean\_accuracy = stance\_data['significant\_striking\_accuracy'].mean()

median\_accuracy = stance\_data['significant\_striking\_accuracy'].median()

std\_accuracy = stance\_data['significant\_striking\_accuracy'].std()

mean\_win\_rate = stance\_data['win\_rate'].mean()

median\_win\_rate = stance\_data['win\_rate'].median()

std\_win\_rate = stance\_data['win\_rate'].std()

print(f'{stance} stance: Mean Accuracy = {mean\_accuracy:.2f}%, Median Accuracy = {median\_accuracy:.2f}%, '

f'Std Accuracy = {std\_accuracy:.2f}%, Mean Win Rate = {mean\_win\_rate:.2f}%, '

f'Median Win Rate = {median\_win\_rate:.2f}%, Std Win Rate = {std\_win\_rate:.2f}%')

else:

print(f'Not enough data for {stance} stance statistics.')