df = pd.read\_csv('players\_22.csv')

df

This creates a Pandas DataFrame object called df that contains the data from the CSV file.df = df.drop('short\_name', axis=1)

df = df.drop('short\_name', axis=1)

This code removes the "short\_name" column from the df DataFrame using the drop() method and specifying the axis parameter as 1 to indicate that we want to drop a column rather than a row. The resulting DataFrame will no longer contain the "short\_name" column.The resulting output is the number of duplicate rows found in the DataFrame based on the "long\_name" column. keeps = ['short\_name','club\_name']

duplicated\_values=df[keeps]

duplicated\_values = duplicated\_values.duplicated().sum()

duplicated\_values

#duplicated\_values.to\_csv('duplicated\_fin.csv')

This code creates a new DataFrame duplicated\_values that contains only the "short\_name" and "club\_name" columns from the df DataFrame and checks for duplicate entries based on these two columns.The duplicated() method is used to identify duplicate entries in the new DataFrame based on the "short\_name" and "club\_name" columns. The resulting Boolean Series contains True for each row that is a duplicate of a previous row and False otherwise. The sum() method is then used to count the number of True values in the Series, which corresponds to the number of duplicate rows in the new DataFrame. The resulting output is the number of duplicate rows found in the new DataFrame based on the "short\_name" and "club\_name" columns.The duplicated() method is used to identify duplicate entries in the DataFrame based on the "short\_name" column. The resulting Boolean Series contains True for each row that is a duplicate of a previous row and False otherwise. The sum() method is then used to count the number of True values in the Series, which corresponds to the number of duplicate rows in the DataFrame. The resulting output is the number of duplicate rows found in the DataFrame based on the "short\_name" column.Overall, the resulting DataFrame df will have the "sofifa\_id" column removed and the rows sorted by their index values. duplicates = df.duplicated(subset='short\_name')

print(duplicates.sum())

This code checks for and prints the number of duplicate entries in the df DataFrame based on the "short\_name" column.df.head(5)

This code displays the first 5 rows of the modified df DataFrame after removing the "short\_name" column and creating the "player\_name" column. The resulting DataFrame will not contain the "short\_name" column and will have a new "player\_name" column that combines the values in the "short\_name" and "club\_name" columns.# Combine columns 'A' and 'B' into a new column 'C' using the .str.cat() method

df['player\_name'] = df['short\_name'].str.cat(df['club\_name'], sep=' ')

This code combines the values in the "short\_name" and "club\_name" columns of the df DataFrame into a new column called "player\_name" using the .str.cat() method. The .str.cat() method is used to concatenate the values in the "short\_name" and "club\_name" columns, with a space separator (' ') between them.The resulting concatenated string is then assigned to the "player\_name" column of the DataFrame. df['player\_name']

Accessing the "player\_name" column of the DataFrame using df['player\_name'] will return a pandas Series object that contains the concatenated values of the "short\_name" and "club\_name" columns for each row of the DataFrame.If the first column of the DataFrame is the "sofifa\_id" column, then this code will sort the DataFrame based on the sofifa\_id values in ascending order. df.drop(df.columns[0], axis=1,inplace=True)

df.sort\_index(inplace=True)

df

This code drops the first column of the df DataFrame (assuming that column represents the "sofifa\_id" column) using the drop() method.duplicates = df.duplicated(subset='long\_name')

print(duplicates.sum())

This code checks for and prints the number of duplicate entries in the df DataFrame based on the "long\_name" column. The duplicated() method is used to identify duplicate entries in the DataFrame based on the "long\_name" column.df.iloc[:, 15:30].isnull().sum()

This code checks for missing values in columns 15 through 30 of the df DataFrame.df.drop(['player\_traits'],axis=1,inplace=True)

This code removes the "player\_traits" column from the df DataFrame using the drop() method and specifying the axis parameter as 1 to indicate that we want to drop a column rather than a row. The inplace parameter is set to True to modify the DataFrame in place instead of returning a new DataFrame. The resulting DataFrame will no longer contain the "player\_traits" column.df.iloc[:, 15:30].isnull().sum()

This code checks for missing values in columns 15 through 30 of the df DataFrame after removing the "player\_traits" column. The .iloc[] method is used to select columns 15 through 30 of the DataFrame by specifying a range of column indices. The resulting DataFrame is then passed to the .isnull() method to generate a Boolean DataFrame that indicates which values are missing. The .sum() method is then used to count the number of missing values in each column of the DataFrame. The resulting output shows the number of missing values in each of the selected columns, with each row representing a different column and the value representing the number of missing values in that column. df['club\_joined'] = pd.to\_datetime(df['club\_joined'])

df['club\_joined'] = df['club\_joined'].dt.year

df['dob'] = pd.to\_datetime(df['dob'])

df['dob'] = df['dob'].dt.year

This code converts the "club\_joined" and "dob" columns of the df DataFrame to datetime format using the pd.to\_datetime() method.# Storing copy of DataFrame without the records that have missing values in value\_eur

import numpy as np

cols\_to\_drop = []

for i in df.columns:

missing = np.abs((df[i].count() - df[i].shape[0])/df[i].shape[0] \* 100)

if missing > 60:

print('{} - {}%'.format(i, round(missing)))

cols\_to\_drop.append(i)

dropped = df.drop(columns=cols\_to\_drop)

This code creates a copy of the df DataFrame called dropped that excludes any rows that have missing values in the "value\_eur" column. Additionally, it identifies and drops any columns that have more than 60% missing values. First, the code calculates the percentage of missing values in each column of the DataFrame using the count() and shape() methods. The percentage of missing values is calculated as the absolute difference between the number of non-missing values and the total number of values in the column, divided by the total number of values in the column, multiplied by 100. Any columns that have more than 60% missing values are identified and stored in the cols\_to\_drop list. Finally, the drop() method is used to create a new DataFrame called dropped that excludes any rows with missing values in the "value\_eur" column and any columns that have more than 60% missing values.The .iloc[] method is used to select columns 15 through 30 of the DataFrame by specifying a range of column indices.

df.to\_csv('test.csv')

This code exports the df DataFrame to a CSV file called "test.csv" using the to\_csv() method. The resulting CSV file will contain the contents of the DataFrame, with each row representing a different player in the dataset and each column representing a different attribute or feature of the player. copy = df.copy()

This code creates a copy of the df DataFrame called copy using the copy() method. The resulting DataFrame will be a new DataFrame that is identical to the original df DataFrame, with the same columns, data types, and values. unique\_positions = df['player\_positions'].unique()

unique\_positions

This code creates a NumPy array called unique\_positions that contains all of the unique values in the "player\_positions" column of the df DataFrame.The .unique() method is used to extract the unique values in the "player\_positions" column. The resulting array will contain all of the unique player positions found in the dataset. sorted\_positions = sorted(unique\_positions)

combined\_positions = ' '.join(sorted\_positions)

print(combined\_positions)

sorted\_positions = sorted(unique\_positions)

combined\_positions = ' '.join(sorted\_positions)

print(combined\_positions)

This code sorts the unique player positions in alphabetical order using the sorted() method and combines them into a single string using the join() method. The resulting string will contain all of the unique player positions found in the dataset, sorted in alphabetical order and separated by spaces. #Create a DataFrame with the sorted positions and length column

positions\_df = pd.DataFrame({'Sorted Positions': sorted\_positions})

positions\_df['Length of Sorted Positions'] = len(sorted\_positions)

#Export to CSV

#positions\_df.to\_csv('positions.csv', index=False)

This code creates a new DataFrame called positions\_df that contains the sorted player positions and the length of the sorted positions. The DataFrame is created using a Python dictionary that contains the sorted positions as the values for the 'Sorted Positions' key. The length of the sorted positions is then calculated using the len() function and added to the DataFrame as a new column called 'Length of Sorted Positions'. Finally, the to\_csv() method is used to export the positions\_df DataFrame to a CSV file called 'positions.csv', with the index parameter set to False to exclude the row index from the exported file. new\_record=''

my\_list=[]

print(len(combined\_positions))

for c in combined\_positions:

if c in [',',' '] :

if new\_record:

my\_list.append(new\_record)

new\_record=""

if c not in [',',' '] :

new\_record+=c

print(my\_list.\_\_len\_\_())

my\_list

This code extracts the unique player positions from the combined\_positions string by iterating over each character in the string and adding each position to a list.one\_df = pd.DataFrame()

one\_df["unique"]=pos\_unique

This code creates a new DataFrame called one\_df with a single column called "unique", which contains the unique player positions from the pos\_unique array. The DataFrame is created by passing the pos\_unique array to the pd.DataFrame() function and specifying the column name as "unique".import numpy as np

my\_set = set(my\_list)

print(len(my\_set))

This code converts the my\_list list of unique player positions to a set called my\_set using the set() function.

1) Does the player's age affect the wage that they take?Additional analysis, such as calculating the correlation coefficient or performing a regression analysis, could be used to further explore the relationship between these two variables and identify any patterns or trends in the data.Additional analysis, such as a scatter plot or regression analysis, could be used to visualize the relationship between these two variables and further explore any patterns or trends in the data.plt.figure(figsize=(8, 6))

sns.barplot(x=country\_players.index, y=country\_players.values, color="blue", capsize=.2)

plt.title('Number Players per country')

plt.xticks(rotation=45)

plt.show()

This code creates a bar plot that shows the number of players from each of the top 10 countries with the most players. The sns.barplot() method from the Seaborn library is used to create the bar plot, with the country names as the x-axis and the number of players as the y-axis. The resulting plot shows the distribution of players across different countries, with each bar representing a different country. The countries are listed in descending order, with the country with the most players listed first. The plot provides a visual summary of the information presented in the country\_players variable, and can be a useful way to quickly identify the countries with the most players. Additionally, the plot can be customized with different colors, labels, and other visual elements to create a more informative and visually appealing plot. # 5) What are the main common characteristics between most expensive players? # Select the attribute columns to analyze (excluding the market value column)

attribute\_columns = ['potential', 'height\_cm', 'skill\_moves', 'shooting', 'passing', 'dribbling', 'defending', 'physic']

# Create subplots for each attribute

fig, axes = plt.subplots(len(attribute\_columns), 1, figsize=(8, 8 \* len(attribute\_columns)))

# Iterate through the attribute columns

for i, attribute in enumerate(attribute\_columns):

# Create a scatter plot with market value on the x-axis and the attribute on the y-axis

axes[i].scatter(df['value\_eur'], df[attribute], alpha=0.5)

axes[i].set\_xlabel('Market Value (EUR)')

axes[i].set\_ylabel(attribute)

axes[i].set\_title(f'{attribute} vs. Market Value')

# Adjust the spacing between subplots

plt.tight\_layout()

# Show the plot

plt.show()

This code creates scatter plots that show the relationship between market value and various player attributes, including potential, height, skill moves, shooting, passing, dribbling, defending, and physic. The attribute\_columns variable selects the columns to analyze, excluding the "value\_eur" column.Additional analysis, such as regression analysis or machine learning models, may be necessary to fully understand the relationship between market value and player attributes.The corr() method is used to calculate the correlation coefficient between the "age" and "wage\_eur" columns of the df DataFrame. The resulting scatter plot shows the distribution of player ages and wages, with each point representing a different player in the dataset. The x-axis represents player age and the y-axis represents player wage in Euros. The plot shows a generally positive trend, indicating that higher wages are generally associated with older players. However, there is also a lot of variability in the data, which suggests that age is not the only factor affecting player wages. The correlation coefficient calculated using the corr() method provides a numerical measure of the strength and direction of the linear relationship between the two variables. A value of 1 indicates a perfect positive correlation, a value of -1 indicates a perfect negative correlation, and a value of 0 indicates no correlation. In this case, the correlation coefficient suggests a positive correlation between player age and wage, but the correlation is not particularly strong (i.e., the coefficient is less than 0.5). Overall, this analysis suggests that player age may be one factor that affects player wages, but it is not the only factor and there is a lot of variability in the data. df.columns

This code displays all of the column names in the df DataFrame. The resulting output will show a list of all the column names, in the order that they appear in the DataFrame. # How does the international reputation of a player impact their wage? # Relationship between International reputation and Wages

reputation\_correlation = df['international\_reputation'].corr(df['wage\_eur'])

print('Correlation : ',reputation\_correlation)

This code calculates the correlation coefficient between a player's international reputation and their wage in Euros (EUR) using the corr() method. The resulting reputation\_correlation variable contains the correlation coefficient between the "international\_reputation" and "wage\_eur" columns of the df DataFrame.# 2) Relationship between International reputation and Wages

fig, ax = plt.subplots(figsize=(10,5))

plt.scatter(data = df, x= 'international\_reputation', y='wage\_eur')

plt.xlabel("International Reputation")

plt.ylabel("Wage in EUR")

plt.title("Reputation & wages in EUR", fontsize = 20)

plt.show()

This code creates a scatter plot that shows the relationship between a player's international reputation and their wage in Euros (EUR). The scatter() method is used to create the scatter plot, with the "international\_reputation" column of the df DataFrame as the x-axis and the "wage\_eur" column as the y-axis.The groupby() method is used to group the df DataFrame by the "international\_reputation" column, and the mean() method is used to calculate the average "overall" rating for each group. The resulting average\_rating\_by\_reputation variable contains a Pandas Series that shows the average overall rating for each level of international reputation. The code then creates a bar plot to visualize the impact of international reputation on overall rating. The plot() method is used to create the bar plot, with the "international\_reputation" column of the df DataFrame as the x-axis and the average overall rating as the y-axis.# 3) Calculate the average overall rating for each level of international reputation

average\_rating\_by\_reputation = df.groupby('international\_reputation')['overall'].mean()

# Create a bar plot to visualize the impact

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

average\_rating\_by\_reputation.plot(kind='bar', color='blue')

plt.xlabel('International Reputation')

plt.ylabel('Average Overall Rating')

plt.title('Impact of International Reputation on Overall Rating')

plt.xticks(rotation=0)

plt.show()

This code calculates the average overall rating for each level of international reputation using the groupby() method.import matplotlib.pyplot as plt

import seaborn as sns

Does the player's age affect the wage that they take?The correlation coefficient provides a numerical measure of the strength and direction of the linear relationship between the two variables.

What is the most age category of the players?The pd.cut() method is used to categorize players into five age categories: "Under 20", "20-24", "25-29", "30-34", and "35+". The value\_counts() method is then used to count the number of players in each age category. The resulting age\_category\_counts variable contains the number of players in each age category based on the player data in the dataset. The code then uses the plot() method to create a bar plot that shows the number of players in each age category. The x-axis represents the age category, and the y-axis represents the number of players in that category. The resulting plot provides a visual summary of the information presented in the age\_category\_counts variable and can be useful for understanding the age distribution of players in the dataset. Additionally, the plot can be customized with different colors, labels, and other visual elements to create a more informative and visually appealing plot. # 8) Is there a correlation between physics and shooting, passing, dribbling, defending and potential of the players? df\_x = df[['shooting', 'defending', 'passing', 'dribbling', 'pace', 'wage\_eur', 'potential', 'overall']]

plt.figure(figsize=(9, 9))

plt.subplots\_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.4, hspace=0.4)

width = 3

height = 3

index = 1

for i in df\_x.columns:

plt.subplot(height, width, index)

plt.scatter(x=df['physic'], y=df\_x[i])

plt.xlabel('physic')

plt.ylabel(i)

index += 1

plt.tight\_layout()

plt.show()

This code creates a scatter plot that shows the relationship between the "physic" attribute and various other attributes of players, including shooting, defending, passing, dribbling, pace, wage\_eur, potential, and overall.The corr() method then calculates the Pearson correlation coefficient between the "defending" and "height\_cm" columns of the filtered dataset, which will give the correlation between the defender players' defending abilities and their height. The resulting correlation value will be a number between -1 and 1, where a value of 1 indicates a perfect positive correlation (i.e., taller defenders tend to have better defending abilities), a value of -1 indicates a perfect negative correlation (i.e., shorter defenders tend to have better defending abilities), and a value of 0 indicates no correlation. # Filter the DataFrame to include only defender players

defender\_players = df[df['player\_positions'].str.contains('CB')]

# Create a scatter plot

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

plt.scatter(defender\_players['height\_cm'], defender\_players['defending'], color='blue')

plt.xlabel('Height (cm)')

plt.ylabel('Overall Rating')

plt.title('Correlation between Defender Players and Height')

plt.show()

This code creates a scatter plot that shows the relationship between defender players' height and their defending abilities. The code first filters the dataset to include only players who play in the center back position, which is done by selecting rows where the "player\_positions" column contains the string "CB".The df['overall'] and df['value\_eur'] code select the "overall" and "value\_eur" columns from the dataset, respectively. The corr() method then calculates the Pearson correlation coefficient between the two columns, which measures the strength of the linear relationship between them. The resulting correlation value will be a number between -1 and 1, where a value of 1 indicates a perfect positive correlation, a value of -1 indicates a perfect negative correlation, and a value of 0 indicates no correlation. The print() function then outputs the calculated correlation value. If the correlation value is positive and close to 1, it suggests that players with higher overall ratings tend to have higher market values. Conversely, if the correlation value is negative and close to -1, it suggests that players with lower overall ratings tend to have higher market values. A correlation value close to 0 indicates that there is no significant relationship between the two variables. # Plotting the scatter plot

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

plt.scatter(df['overall'], df['value\_eur'], alpha=0.5)

plt.xlabel('Overall Rating')

plt.ylabel('Market Value (in Euros)')

plt.title('Correlation between Overall Rating and Market Value')

plt.show()

This code creates a scatter plot that shows the relationship between a player's overall rating and their market value. The df['overall'] and df['value\_eur'] code select the "overall" and "value\_eur" columns from the dataset, respectively.The top\_players['long\_name'] and top\_players['potential'] code extract the "long\_name" and "potential" columns from the filtered dataset, respectively. The for loop then iterates through the top 10 players and prints their names and potential ratings using formatted string literals. The resulting output will show the names and potential ratings of the top 10 players with the highest potential ratings. # Sort the DataFrame by potential rating in descending order

top\_players = df.nlargest(10, 'potential')

# Extract the relevant columns

player\_names = top\_players['long\_name']

potential\_ratings = top\_players['potential']

# Create a bar plot

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

plt.bar(player\_names, potential\_ratings, color='blue')

plt.xlabel('Player')

plt.ylabel('Potential Rating')

plt.title('Top 10 Players with Highest Potential')

plt.xticks(rotation=45)

plt.show()

This code creates a bar plot that shows the top 10 players with the highest potential rating. The df.nlargest(10, 'potential') code sorts the dataset by the "potential" column in descending order and selects the top 10 rows with the highest potential ratings.The code then uses a for loop to iterate through the selected columns and create a scatter plot for each column that shows the relationship between the "physic" attribute and the selected attribute. # 9) Is there a correlation between height and shooting, passing, dribbling, defending and potential of the players? df\_x = df[['shooting', 'defending', 'passing', 'dribbling', 'pace', 'wage\_eur', 'potential', 'overall']]

plt.figure(figsize=(9, 9))

plt.subplots\_adjust(left=0.1,

bottom=0.1,

right=0.9,

top=0.9,

wspace=0.4,

hspace=0.4)

width = 3

height = 3

index = 1

for i in df\_x.columns:

plt.subplot(height, width, index)

plt.scatter(x=df['height\_cm'], y=df\_x[i])

plt.xlabel('height\_cm')

plt.ylabel(i)

index += 1

# Show the plots

plt.show()

This code creates a scatter plot that shows the relationship between the "height\_cm" attribute and various other attributes of players, including shooting, defending, passing, dribbling, pace, wage\_eur, potential, and overall.The code then uses a for loop to iterate through the selected columns and create a scatter plot for each column that shows the relationship between the "height\_cm" attribute and the selected attribute

# 10) Is there a correlation between weight and shooting, passing, dribbling, defending and potential of the players? df\_x = df[['shooting', 'defending', 'passing', 'dribbling', 'pace', 'wage\_eur', 'potential', 'overall']]

plt.figure(figsize=(9, 9))

plt.subplots\_adjust(left=0.1,

bottom=0.1,

right=0.9,

top=0.9,

wspace=0.4,

hspace=0.4)

width = 3

height = 3

index = 1

for i in df\_x.columns:

plt.subplot(height, width, index)

plt.scatter(x=df['weight\_kg'], y=df\_x[i])

plt.xlabel('weight\_kg')

plt.ylabel(i)

index += 1

Show the plots

plt.show()

This code creates a scatter plot that shows the relationship between the "weight\_kg" attribute and various other attributes of players, including shooting, defending, passing, dribbling, pace, wage\_eur, potential, and overall.The code then uses a for loop to iterate through the selected columns and create a scatter plot for each column that shows the relationship between the "weight\_kg" attribute and the selected attribute. The resulting plot can be used to assess whether there is a correlation between the "weight\_kg" attribute and various other attributes of players.# 14) Which most 10 players have the highest potential?

What is the most age category of the players?The pd.cut() method is used to categorize players into five age categories: "Under 20", "20-24", "25-29", "30-34", and "35+". The value\_counts() method is then used to count the number of players in each age category. The resulting age\_category\_counts variable contains the number of players in each age category based on the player data in the dataset. The code then uses the plot() method to create a bar plot that shows the number of players in each age category. The x-axis represents the age category, and the y-axis represents the number of players in that category. The resulting plot provides a visual summary of the information presented in the age\_category\_counts variable and can be useful for understanding the age distribution of players in the dataset. Additionally, the plot can be customized with different colors, labels, and other visual elements to create a more informative and visually appealing plot. # 8) Is there a correlation between physics and shooting, passing, dribbling, defending and potential of the players? df\_x = df[['shooting', 'defending', 'passing', 'dribbling', 'pace', 'wage\_eur', 'potential', 'overall']]

plt.figure(figsize=(9, 9))

plt.subplots\_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.4, hspace=0.4)

width = 3

height = 3

index = 1

for i in df\_x.columns:

plt.subplot(height, width, index)

plt.scatter(x=df['physic'], y=df\_x[i])

plt.xlabel('physic')

plt.ylabel(i)

index += 1

plt.tight\_layout()

plt.show()

This code creates a scatter plot that shows the relationship between the "physic" attribute and various other attributes of players, including shooting, defending, passing, dribbling, pace, wage\_eur, potential, and overall.The corr() method then calculates the Pearson correlation coefficient between the "defending" and "height\_cm" columns of the filtered dataset, which will give the correlation between the defender players' defending abilities and their height.The df.nlargest(10, 'potential') code sorts the dataset by the "potential" column in descending order and selects the top 10 rows with the highest potential ratings.The code then uses a for loop to iterate through the selected columns and create a scatter plot for each column that shows the relationship between the "physic" attribute and the selected attribute. # 9) Is there a correlation between height and shooting, passing, dribbling, defending and potential of the players? df\_x = df[['shooting', 'defending', 'passing', 'dribbling', 'pace', 'wage\_eur', 'potential', 'overall']]

plt.figure(figsize=(9, 9))

plt.subplots\_adjust(left=0.1,

bottom=0.1,

right=0.9,

top=0.9,

wspace=0.4,

hspace=0.4)

width = 3

height = 3

index = 1

for i in df\_x.columns:

plt.subplot(height, width, index)

plt.scatter(x=df['height\_cm'], y=df\_x[i])

plt.xlabel('height\_cm')

plt.ylabel(i)

index += 1

# Show the plots

plt.show()

This code creates a scatter plot that shows the relationship between the "height\_cm" attribute and various other attributes of players, including shooting, defending, passing, dribbling, pace, wage\_eur, potential, and overall.The code then uses a for loop to iterate through the selected columns and create a scatter plot for each column that shows the relationship between the "height\_cm" attribute and the selected attribute

# 10) Is there a correlation between weight and shooting, passing, dribbling, defending and potential of the players?# Filter the DataFrame to include only defender players

defender\_players = df[df['player\_positions'].str.contains('CB')]

# Create a scatter plot

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

plt.scatter(defender\_players['height\_cm'], defender\_players['defending'], color='blue')

plt.xlabel('Height (cm)')

plt.ylabel('Overall Rating')

plt.title('Correlation between Defender Players and Height')

plt.show()

This code creates a scatter plot that shows the relationship between defender players' height and their defending abilities.s