Part 1:

1. Idea:

This Python program automates the collection of detailed statistical data for English Premier League football players from FBref.com, focusing on those who have played over 90 minutes. The goal is to extract a diverse set of performance metrics and professional parameters, then compile, process, and save them into a results.csv file. This file is structured with each row representing a player, each column corresponding to a statistic, players sorted alphabetically by their first name, and missing values marked as "N/a", facilitating subsequent in-depth data analysis.

2. Code Analysis

The program is written in Python and utilizes several powerful libraries to perform tasks ranging from web browsing to data processing:

- **Selenium (webdriver):** Used to automate the Chrome browser, allowing the program to access and interact with the dynamic content of the FBref website.
- **BeautifulSoup4 (bs4):** This library parses the HTML code of web pages, helping to extract the necessary data tables.
- **Pandas (pd):** The central library for data manipulation. Pandas is used to create and manage DataFrames (data tables), and to perform operations such as merging, filtering, and cleaning data.
- **StringIO (io.StringIO):** Supports reading strings as file objects, which is necessary when converting HTML tables (as strings) into Pandas DataFrames.
- **Time (time):** The time module is used to introduce delays (time.sleep(5)) during web browsing, ensuring pages have sufficient time to load completely before data extraction, and also to avoid sending too many requests to the server 연속적으로.

Main Operational Flow of the Code:

1. **Initialize Browser:** An instance of webdriver.Chrome() is created.

```
driver=webdriver.Chrome()
```

2. **Define Links (links):** A dictionary stores target URLs on FBref and the IDs of the HTML tables containing the corresponding statistical data (e.g., Standard, Goalkeeping, Shooting, etc.).

```python
# các liên kết đến các bảng dữ liệu
links = {
    "Standard": ("https://fbref.com/en/comps/9/stats/Premier-League-Stats", "stats_standard"),
    "Goalkeeping": ("https://fbref.com/en/comps/9/keepers/Premier-League-Stats", "stats_keeper"),
    "Shooting": ("https://fbref.com/en/comps/9/shooting/Premier-League-Stats", "stats_shooting"),
    "Passing": ("https://fbref.com/en/comps/9/passing/Premier-League-Stats", "stats_passing"),
    "GnS Creation": ("https://fbref.com/en/comps/9/gca/Premier-League-Stats", "stats_gca"),
    "Defensive act": ("https://fbref.com/en/comps/9/defense/Premier-League-Stats", "stats_defense"),
    "Possession": ("https://fbref.com/en/comps/9/possession/Premier-League-Stats", "stats_possession"),
    "Misc": ("https://fbref.com/en/comps/9/misc/Premier-League-Stats", "stats_misc")
}
```

3. **scraping(url, table_id) Function:**
   - This function is the core of data collection from each page.
   - It navigates to the url using driver.get().
   - Pauses for 5 seconds for the page to load.
   - Retrieves the page source (driver.page_source) and uses BeautifulSoup to parse it.
   - Finds the specific HTML table based on table_id.
   - Uses pd.read_html() (in conjunction with StringIO) to convert the HTML table into a Pandas DataFrame. The header row is specified by header=1.
   - Performs basic cleaning steps to remove any redundant or unnecessary header rows within the table.

```python
def  (variable) driver: WebDriver
    driver.get(url)
    time.sleep(5)
    soup = BeautifulSoup(driver.page_source, "html.parser")
    table = soup.find("table", {"id": table_id})
    df = pd.read_html(StringIO(str(table)), header=1)[0]
    df=df[~df["Rk"].str.contains("Rk", na=False)]
    df = df[df.apply(lambda row: not any(row.astype(str) == row.name), axis=1)]
    return df
```

4. **Data Collection and Merging (all_df):**
   - The program iterates through each item in the links dictionary.
   - Calls the scraping function to get a DataFrame for each type of statistic.
   - The first DataFrame is assigned as all_df. Subsequent DataFrames are merged (pd.merge) into all_df based on the common columns "Player" and "Squad". An outer join is used to ensure no player data is lost. Suffixes (suffixes) are added to duplicate column names to distinguish their origin.

```python
all_df = None
# Duyệt qua từng liên kết và lấy dữ liệu, sau đó gộp lại
for name, (url, table_id) in links.items():
    df = scraping(url, table_id)
    if all_df is None:
        all_df = df
    else:
        all_df = pd.merge(all_df, df, on=["Player", "Squad"], how="outer", suffixes=("", f"_{name}"))
        print(f"Gộp với {name} xong, kích thước all_df: {all_df.shape}")
```

5. **Post-Merge Data Processing:**
   - **Filter by playing time:** The "Min" column is converted to a numeric type. Only players with playing time (Min) greater than 90 minutes are retained.

```python
if "Min" in all_df.columns:
    all_df["Min"] = pd.to_numeric(all_df["Min"], errors="coerce")
    all_df = all_df[all_df["Min"] > 90]
```

   - **Handle missing values (NaN):** All NaN values in the DataFrame are replaced with the string "N/a".

```python
# Thay thế giá trị NaN bằng "N/a"
all_df.fillna("N/a", inplace=True)
```

   - **Create "First_name" column:** A new column containing the player's first name is created from the "Player" column to facilitate sorting.

```python
all_df["First_name"] = all_df["Player"].apply(lambda x: x.split()[0] if isinstance(x, str) else "N/a")
```

   - **Sort data:** The all_df DataFrame is sorted alphabetically by "First_name".

```python
all_df = all_df.sort_values(by="First_name")
```

   - **Select columns (columns_to_keep):** Only columns whose names are in the predefined columns_to_keep list (containing all desired statistical indicators) are retained in the final DataFrame.

```
# Các cột cần giữ lại trong DataFrame
columns_to_keep = [
    "Player", "Squad", "Nation", "Pos", "Age",
    "MP", "Starts", "Min",
    "Gls", "Ast", "CrdY", "CrdR",
    "xG", "xAG",
    "PrgC", "PrgP", "PrgR",
    "Gls.1", "Ast.1", "xG.1", "xAG.1",
    "GA90", "Save%", "CS%",
    "PKsv",
    "SoT%", "SoT/90", "G/Sh", "Dist",
    "Cmp", "Cmp%", "TotDist",
    "Cmp%.1", "Cmp%.2", "Cmp%.3",
    "KP", "1/3", "PPA", "CrsPA", "PrgP_Passing",
    "SCA", "SCA90", "GCA", "GCA90",
    "Tkl", "TklW", "Att_Defensive act", "Lost",
    "Blocks", "Sh_Defensive act", "Pass", "Int",
    "Touches", "Def Pen", "Def 3rd_Possession",
    "Mid 3rd_Possession", "Att 3rd_Possession", "Att Pen",
    "Att_Possession", "Succ%", "Tkld%",
    "Carries", "PrgDist_Possession", "PrgC_Possession",
    "1/3_Possession", "CPA", "Mis", "Dis",
    "Rec", "PrgR_Possession",
    "Fls", "Fld_Misc", "Off", "Crs", "Recov",
    "Won", "Lost_Misc", "Won%"
]
```

6. **Save Results:**
   - The complete all_df DataFrame is saved to a results.csv file.
   - The index=False parameter is used to prevent writing the DataFrame index to the file.
   - encoding="utf-8-sig" ensures correct handling of special characters.
   - A confirmation message indicating the file has been saved is printed to the console.

```
86    # Lưu DataFrame vào file CSV
87    all_df.to_csv("results.csv", index=False, encoding="utf-8-sig")
88    print("Dữ liệu đã được lưu vào file results.csv.")
```

Part 2:

1. Concept

This Python program is designed to perform statistical analysis on a pre-existing dataset of footballer statistics, presumably stored in a file named results.csv. The core idea is to process this data to extract meaningful insights, including identifying top and bottom performers, calculating descriptive statistics for various metrics both league-wide and فريق-wise, and visualizing the distribution of key statistics. The program aims to generate summary files (top_3.txt, results2.csv) and display histograms for selected attacking and defensive metrics.

2. Code Analysis

The program is written in Python and utilizes several common libraries for data manipulation and visualization:

- **Pandas (pd):** Used extensively for reading the input CSV file, data manipulation (handling missing values, selecting columns, grouping), and creating summary DataFrames.
- **NumPy (np):** While not directly used in many operations, Pandas relies on NumPy for its numerical operations and data structures. It's explicitly imported and used for np.nan when handling potentially missing aggregated statistics.
- **Matplotlib.pyplot (plt):** Employed for generating and displaying histograms to visualize the distribution of player statistics.
- **OS (os):** Imported but not actively used in the provided snippet (the comment indicates it's for working with directories, which might be relevant in a fuller version of the script, e.g., for creating output directories if they don't exist).

Main Operational Flow of the Code:

1. **plot_histograms(df, numeric_cols, grouped_by_squad) Function:**
    - This function is responsible for generating and displaying histograms.
    - It iterates through the provided numeric_cols.
    - For each statistic:
        - It prepares the data by dropping any missing (NaN) values.

- **League-wide Histogram:** A histogram is plotted showing the distribution of the statistic for all players in the league.
- **Team-specific Histograms:** If grouped_by_squad (a Pandas GroupBy object) is provided, the function then iterates through each team. For each team, it filters the data for that team's players and plots a separate histogram for the current statistic.

- plt.show() is called after each histogram is configured to display it.

```python
def plot_histograms(df: pd.DataFrame, numeric_cols: list, grouped_by_squad=None): # Bỏ output_dir
    for col_stat in numeric_cols:
        data_for_hist = df[col_stat].dropna()
        # 1. Histogram cho toàn giải
        plt.figure(figsize=(10, 6))
        plt.hist(data_for_hist, bins='auto', color='skyblue', edgecolor='black')
        plt.title(f"Phân phối của {col_stat} - Toàn giải")
        plt.xlabel(col_stat)
        plt.ylabel("Số lượng cầu thủ")
        plt.grid(axis='y', alpha=0.75)
        plt.show() # Hiển thị biểu đồ

        # 2. Histogram cho từng đội (nếu có grouped_by_squad)
        if grouped_by_squad is not None:
            print(f"    Đang xử lý histogram theo đội cho cột: {col_stat}")
            for team_name, team_data_group in grouped_by_squad:
                team_stat_data = team_data_group[col_stat].dropna()

                if team_stat_data.empty:
                    continue

                plt.figure(figsize=(10, 6)) # Tạo figure mới cho mỗi biểu đồ của đội
                plt.hist(team_stat_data, bins='auto', color='lightcoral', edgecolor='black')
                plt.title(f"Phân phối của {col_stat} - Đội: {team_name}")
                plt.xlabel(col_stat)
                plt.ylabel("Số lượng cầu thủ")
                plt.grid(axis='y', alpha=0.75)
                plt.show() # Hiển thị biểu đồ của đội
            print(f"    Đã xử lý xong histogram theo đội cho cột: {col_stat}")

    print("Hoàn tất việc chuẩn bị hiển thị histogram. Các cửa sổ biểu đồ sẽ lần lượt xuất hiện.")
```

2. **main() Function:** This is the primary execution block.
   - **Read Data:** Reads data from results.csv into a Pandas DataFrame df. na_values='N/a' ensures that "N/a" strings are treated as missing values.

```python
df = pd.read_csv("results.csv", na_values='N/a')
```

   - **Handle Missing Values:**
     - Identifies all numeric columns in the DataFrame.
     - Fills any NaN values in these numeric columns with the mean of their respective columns.

- Checks if essential text columns ('Player', 'Squad') are present.

```
    numeric_cols = df.select_dtypes(include='number').columns.tolist()
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
    print(f"Đã xử lý giá trị bị thiếu cho các cột số: {numeric_cols}.")
    required_text_cols = ['Player', 'Squad']
    missing_text_cols = [col for col in required_text_cols if col not in df.columns]
```

- **Write top_3.txt File:**
  - Opens top_3.txt for writing.
  - For each numeric column:
    - Writes a header for the statistic.
    - Determines which columns to display (Player, Squad, and the current statistic).
    - Uses nlargest(3, ...) and nsmallest(3, ...) to find the top 3 and bottom 3 players for that statistic.
    - Writes this information to the text file.

```
with open("top_3.txt", "w", encoding="utf-8") as f:
    for col_stat in numeric_cols:
        f.write(f"----------- Thống kê: {col_stat} -----------\n")

        display_cols_for_top_bottom = []
        if 'Player' in df.columns:
            display_cols_for_top_bottom.append('Player')
        if 'Squad' in df.columns:
            display_cols_for_top_bottom.append('Squad')
        display_cols_for_top_bottom.append(col_stat)

        if df[col_stat].notna().any():
            top3 = df.nlargest(3, col_stat)[display_cols_for_top_bottom]
            bot3 = df.nsmallest(3, col_stat)[display_cols_for_top_bottom]

            f.write(f"Top 3 CAO NHẤT theo {col_stat}:\n")
            f.write(top3.to_string(index=False) + "\n")
            f.write(f"Top 3 THẤP NHẤT theo {col_stat}:\n")
            f.write(bot3.to_string(index=False) + "\n\n")
        else:
            f.write(f"Cột {col_stat} không có dữ liệu hợp lệ để tìm top/bottom 3.\n\n")
```

- **Group Data by Team:**
  - If the 'Squad' column exists, it groups the DataFrame df by this column, creating a DataFrameGroupBy object named grouped_by_squad.

```
grouped_by_squad = None
if 'Squad' in df.column    (variable) df: DataFrame
    grouped_by_squad = df.groupby('Squad')
else:
    print("CẢNH BÁO: Không tìm thấy cột 'Squad'. Không thể nhóm theo đội.")
```

- **Calculate League-wide Statistics:**

- If numeric columns exist and are not empty, it calculates the median, mean, and standard deviation (std) for each numeric column across all players. The result is stored in all_league_stats after transposing (.T).

```
    all_league_stats = None
    if numeric_cols:
        if not df[numeric_cols].empty:
            all_league_stats = df[numeric_cols].agg(['median', 'mean', 'std']).T
            print("Đã tính toán median, mean, std cho toàn giải.")
        else:
            print("Không có dữ liệu số để tính thống kê toàn giải.")
    else:
        print("Không có cột số nào được xác định để tính thống kê toàn giải.")
```

- **Prepare and Create results2.csv:**
  - This section executes if numeric_cols, all_league_stats, and grouped_by_squad are all available.
  - It prepares a list of rows (rows_for_results2) to build the new CSV.
  - Defines the column structure for results2.csv: 'Squad', followed by 'Median of StatX', 'Mean of StatX', 'Std of StatX' for each numeric statistic.
  - **League Totals Row:** Adds a row to rows_for_results2 for the 'all' league statistics.
  - **Team-specific Rows:** If grouped_by_squad contains numeric data, it aggregates median, mean, and std for each team and each numeric column. Each team's aggregated stats are added as a new row. .get() is used with np.nan as a default to prevent errors if a specific statistic combination is missing.
  - A new DataFrame results2_df is created from rows_for_results2 and saved to results2.csv without the index and with utf-8-sig encoding.

```python
if numeric_cols and all_league_stats is not None and grouped_by_squad is not None:
    print("Đang chuẩn bị dữ liệu cho 'results2.csv'...")
    rows_for_results2 = []
    columns_for_results2 = ['Squad']
    for col_stat in numeric_cols:
        columns_for_results2.extend([f"Median của {col_stat}", f"Mean của {col_stat}", f"Std của {col_stat}"])
    all_league_values = []
    for col_stat in numeric_cols:
        if col_stat in all_league_stats.index:
            all_league_values.extend([
                all_league_stats.loc[col_stat, 'median'],
                all_league_stats.loc[col_stat, 'mean'],
                all_league_stats.loc[col_stat, 'std']
            ])
        else:
            all_league_values.extend([np.nan, np.nan, np.nan])
    rows_for_results2.append(['all'] + all_league_values)
    if not grouped_by_squad[numeric_cols].first().empty: # Kiểm tra nếu có dữ liệu số trong nhóm
        team_aggregated_stats = grouped_by_squad[numeric_cols].agg(['median', 'mean', 'std'])
        for team_name, team_specific_stats in team_aggregated_stats.iterrows():
            team_values = []
            for col_stat in numeric_cols:
                team_values.extend([
                    team_specific_stats.get((col_stat, 'median'), np.nan),
                    team_specific_stats.get((col_stat, 'mean'), np.nan),
                    team_specific_stats.get((col_stat, 'std'), np.nan)
                ])
            rows_for_results2.append([team_name] + team_values)
    else:
        print("Cảnh báo: Không có dữ liệu số trong các nhóm đội để tổng hợp cho results2.csv.")
    results2_df = pd.DataFrame(rows_for_results2, columns=columns_for_results2)
    results2_df.to_csv('results2.csv', index=False, encoding='utf-8-sig')
    print(results2_df.head())
else:
    print("Bỏ qua việc tạo 'results2.csv' do thiếu grouped_by_squad, all_league_stats hoặc numeric_cols.")
```

- **Plot Histograms (Specific Columns):**
  - Defines two lists, attack_cols_defined (["Gls", "Ast", "Dist"]) and defense_cols_defined (["Tkl", "TklW", "Att_Defensive act"]).
  - These are combined into target_hist_cols.
  - cols_to_plot_actual is created by filtering target_hist_cols to include only those columns that actually exist in the DataFrame df and are of a numeric data type.
  - If cols_to_plot_actual is not empty, the plot_histograms function is called with these specific columns.

```
    attack_cols_defined = ["Gls", "Ast", "Dist"]
    defense_cols_defined = ["Tkl", "TklW", "Att_Defensive act"]

    target_hist_cols = attack_cols_defined + defense_cols_defined

    cols_to_plot_actual = []
    for col_name in target_hist_cols:
        if col_name in df.columns:
            if pd.api.types.is_numeric_dtype(df[col_name]):
                cols_to_plot_actual.append(col_name)

    if cols_to_plot_actual:
        print(f"\nSẽ vẽ histogram cho các cột: {', '.join(cols_to_plot_actual)}")
        plot_histograms(df, cols_to_plot_actual, grouped_by_squad)
    else:
        print("\nKhông có cột tấn công/phòng ngự hợp lệ nào được tìm thấy để vẽ histogram.")
```

3. **Execution Block (if __name__ == '__main__':)**
    - This standard Python construct ensures that the main() function is called only when the script is executed directly (not when imported as a module).

Part 3:

1. Concept

This Python program employs K-means clustering to classify football players into distinct groups based on their statistical performance. The objective is to identify inherent player categories—such as different roles, performance tiers, or styles of play—without pre-existing labels.

The process involves several key steps:

- **Data Preprocessing:** Cleaning the statistical data, imputing any missing values, and scaling features to ensure fair comparison.
- **Optimal Cluster (K) Determination:** Utilizing the "Elbow method" by plotting K-means inertia (sum of squared distances to cluster centers) for various K values. The script identifies the optimal K as the point where the most significant drop in inertia occurs, indicating a good balance for cluster formation.
- **K-means Clustering:** Applying the K-means algorithm with the determined optimal K to assign players to clusters.
- **Dimensionality Reduction & Visualization:** Using Principal Component Analysis (PCA) to reduce the data to two dimensions. This allows for a 2D scatter plot where players are colored by their assigned cluster, providing a visual

representation of the groupings and their separation.

2. Code Analysis

The program is written in Python and utilizes several libraries from the scikit-learn ecosystem for machine learning tasks, along with Pandas for data handling and Matplotlib for plotting:

- **Pandas (pd):** Used for reading the input CSV file (results.csv), data manipulation (dropping columns, applying cleaning functions), and storing cluster assignments.
- **NumPy (np):** Implicitly used by Pandas and scikit-learn for numerical operations.
- **Scikit-learn (sklearn):**
  - StandardScaler: For standardizing features by removing the mean and scaling to unit variance.
  - KMeans: The K-means clustering algorithm.
  - SimpleImputer: For handling missing values (NaNs) by replacing them with a specified strategy (e.g., mean).
  - PCA: Principal Component Analysis for dimensionality reduction.
- **Matplotlib.pyplot (plt):** Used for generating plots, specifically the Elbow method plot and the 2D scatter plot of the clusters.
- **OS (os):** Imported in the previous script but not explicitly used in this clustering script snippet.

Main Operational Flow of the Code:

1. **Load Data:** Player statistics are read from results.csv into a Pandas DataFrame.
   # Read data
   df = pd.read_csv("results.csv")

2. **Initial Data Cleaning (Dropping Columns):** Identifier columns that are not suitable for clustering (like Player name, Squad, Nation, Position, Age) are dropped from the DataFrame. errors='ignore' ensures that if any of these columns are already missing, the code doesn't raise an error.

```
11  # Bỏ các cột định danh không dùng cho phân cụm
12  cols_to_drop = ["Player", "Squad", "Nation", "Pos", "Age"]
13  df_clean = df.drop(columns=cols_to_drop, errors='ignore')
```

3. **Numeric Data Cleaning and Conversion:**
   - A function clean_numeric is defined to convert columns to a numeric type. It

first converts the column to string type, then removes any '%' characters (e.g., from percentage values), and finally attempts to convert to numeric. Values that cannot be converted become NaN.
    - This function is applied to all columns in df_clean.

```
def clean_numeric(col):
    return pd.to_numeric(col.astype(str).str.replace('%', '', regex=False), errors='coerce')

df_clean = df_clean.apply(clean_numeric)
```

4. **Impute Missing Values:** SimpleImputer is used to fill any NaN values that resulted from the previous cleaning step or were already present. The strategy is "mean", meaning NaNs in each column will be replaced by the mean of that column.

```
imputer = SimpleImputer(strategy="mean")
df_clean_imputed = imputer.fit_transform(df_clean)
```

5. **Data Standardization:** The features (player statistics) are standardized using StandardScaler. This process transforms the data to have a mean of 0 and a standard deviation of 1, which is important for distance-based algorithms like K-means to prevent features with larger magnitudes from dominating the results.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_clean_imputed)
```

6. **Determine Optimal K (Elbow Method):**
    - The script calculates the K-means inertia (sum of squared distances of samples to their closest cluster center) for a range of cluster numbers K_range (1 to 29).
    - It then identifies the optimal_K by finding where the largest drop in inertia occurs between consecutive K values. This point is considered the "elbow" and suggests a good balance between the number of clusters and the compactness of each cluster.
    - The Elbow curve (Inertia vs. Number of clusters) is plotted to visually aid in selecting K.

```python
inertia = []
K_range = range(1, 30)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Tính toán số K tối ưu từ sự thay đổi lớn nhất trong inertia
# Tìm sự thay đổi lớn nhất giữa các inertia liên tiếp
inertia_diff = [inertia[i] - inertia[i+1] for i in range(len(inertia)-1)]
optimal_K = inertia_diff.index(max(inertia_diff)) + 2
print(f"Số K tối ưu là {optimal_K}, vì từ K={optimal_K-1} tới K={optimal_K} ta nhận th

# Vẽ biểu đồ Elbow
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, 'bo-', markersize=8)
plt.xlabel('Số cụm (k)')
plt.ylabel('Inertia')
plt.title('Phương pháp Elbow để chọn số cụm K')
plt.grid(True)
plt.tight_layout()
plt.show()
```

7. **Apply K-means Clustering:** K-means clustering is performed on the scaled data (X_scaled) using the optimal_K found in the previous step. The cluster labels for each player are then added as a new 'Cluster' column to the df_clean DataFrame (note: it might be more appropriate to add it to the original df or a copy to retain player identifiers alongside cluster labels for easier interpretation).

```python
# Áp dụng K-means phân cụm với số K tối ưu
kmeans_optimal = KMeans(n_clusters=optimal_K, random_state=42, n_init=10)
df_clean['Cluster'] = kmeans_optimal.fit_predict(X_scaled)
```

8. **Dimensionality Reduction with PCA:** PCA is used to reduce the dimensionality of the scaled data (X_scaled) to 2 components (n_components=2). This allows for visualization of the clusters in a 2D scatter plot.

```python
# Giảm chiều dữ liệu xuống 2D với PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

9. **Visualize Clustering Results:**

- A scatter plot is created where the x-axis is the first principal component (PC1) and the y-axis is the second principal component (PC2).
- Each point represents a player, and the color of the point is determined by the cluster label assigned by K-means.
- This plot helps to visually inspect the separation and distribution of the identified player clusters.

```
63  plt.figure(figsize=(8, 5))
64  plt.scatter(X_pca[:, 0], X_pca[:, 1], c=df_clean['Cluster'], cmap='viridis', s=50)
65  plt.title('Phân cụm K-means với số K tối ưu (PCA 2D)')
66  plt.xlabel('PC 1')
67  plt.ylabel('PC 2')
68  plt.colorbar(label='Cluster')
69  plt.show()
```

Part4 :

1. Concept

This Python program automates collecting football player transfer values from footballtransfers.com and integrates them with existing statistics from a local results.csv file. Key steps include:

1. Scraping player names, teams, and transfer values.
2. Reading local player statistics.
3. Filtering players from the stats file based on a minimum playing time (900 minutes).
4. Merging scraped values with filtered stats using fuzzy name matching.
5. Saving the combined dataset (stats + transfer values) to players_900mins_transfer_values.csv.

The aim is to enrich player performance data with market valuations for UK Premier League players.

2. Code Analysis

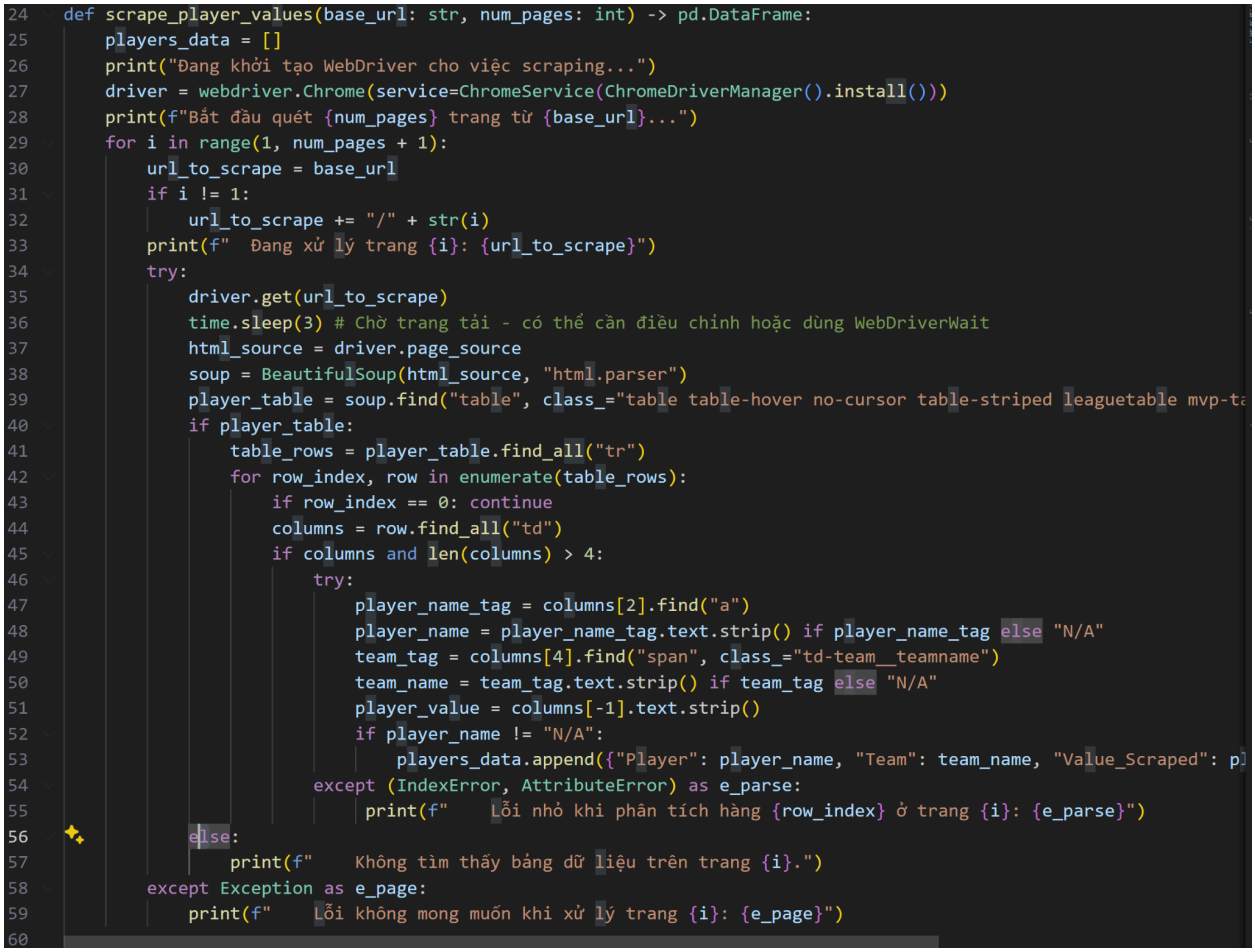
The script uses Python libraries like Pandas, BeautifulSoup, Selenium, WebDriver Manager, Time, and Rapidfuzz for web scraping, data handling, and fuzzy string matching.

Global Configurations:
The script defines constants for URLs, file paths, and processing thresholds (e.g., NUMBER_OF_PAGES_TO_SCRAPE, MINIMUM_PLAYING_TIME_MINS,

FUZZY_MATCH_SIMILARITY_THRESHOLD_PERCENT).

Main Operational Flow of the Code:

1. **scrape_player_values(...) Function:**
   - Scrapes player names, teams, and transfer values from multiple pages of footballtransfers.com using Selenium and BeautifulSoup.
   - Initializes and quits a Chrome WebDriver.
   - Includes error handling and page load delays.
   - Returns a DataFrame of scraped data.



```python
def scrape_player_values(base_url: str, num_pages: int) -> pd.DataFrame:
    players_data = []
    print("Đang khởi tạo WebDriver cho việc scraping...")
    driver = webdriver.Chrome(service=ChromeService(ChromeDriverManager().install()))
    print(f"Bắt đầu quét {num_pages} trang từ {base_url}...")
    for i in range(1, num_pages + 1):
        url_to_scrape = base_url
        if i != 1:
            url_to_scrape += "/" + str(i)
        print(f"  Đang xử lý trang {i}: {url_to_scrape}")
        try:
            driver.get(url_to_scrape)
            time.sleep(3) # Chờ trang tải - có thể cần điều chỉnh hoặc dùng WebDriverWait
            html_source = driver.page_source
            soup = BeautifulSoup(html_source, "html.parser")
            player_table = soup.find("table", class_="table table-hover no-cursor table-striped leaguetable mvp-ta
            if player_table:
                table_rows = player_table.find_all("tr")
                for row_index, row in enumerate(table_rows):
                    if row_index == 0: continue
                    columns = row.find_all("td")
                    if columns and len(columns) > 4:
                        try:
                            player_name_tag = columns[2].find("a")
                            player_name = player_name_tag.text.strip() if player_name_tag else "N/A"
                            team_tag = columns[4].find("span", class_="td-team__teamname")
                            team_name = team_tag.text.strip() if team_tag else "N/A"
                            player_value = columns[-1].text.strip()
                            if player_name != "N/A":
                                players_data.append({"Player": player_name, "Team": team_name, "Value_Scraped": pl
                        except (IndexError, AttributeError) as e_parse:
                            print(f"    Lỗi nhỏ khi phân tích hàng {row_index} ở trang {i}: {e_parse}")
            else:
                print(f"    Không tìm thấy bảng dữ liệu trên trang {i}.")
        except Exception as e_page:
            print(f"    Lỗi không mong muốn khi xử lý trang {i}: {e_page}")
```

2. **read_and_filter_player_stats(...) Function:**
   - Reads player statistics from a local CSV (stats_csv_path).
   - Handles file errors and checks for required columns ('Player', 'Min').
   - Converts 'Min' column to numeric and filters players by min_playing_time.
   - Selects specified columns_to_keep.
   - Returns the filtered DataFrame.

```python
def read_and_filter_player_stats(stats_csv_path: str, min_playing_time: int, columns_to_keep: list) -> pd.DataFra

    df_stats = pd.read_csv(stats_csv_path)
    required_cols_stats = ["Player", "Min"]
    for col in required_cols_stats:
        if col not in df_stats.columns:
            print(f"LỖI: Cột bắt buộc '{col}' không tìm thấy trong '{stats_csv_path}'.")
            return pd.DataFrame()
    if df_stats['Min'].dtype == 'object': # Xử lý cột 'Min' nếu là dạng chuỗi (ví dụ: '1,234')
        df_stats['Min'] = df_stats['Min'].astype(str).str.replace(',', '', regex=False)
        df_stats['Min'] = pd.to_numeric(df_stats['Min'], errors='coerce')
        df_stats.dropna(subset=['Min'], inplace=True)
    filtered_players_df = df_stats[df_stats["Min"] > min_playing_time].copy()
    if filtered_players_df.empty:
        print(f"Không tìm thấy cầu thủ nào có số phút thi đấu > {min_playing_time} phút.")
        return pd.DataFrame()
    print(f"Tìm thấy {len(filtered_players_df)} cầu thủ có số phút thi đấu > {min_playing_time} phút.")
    valid_columns_to_keep = [col for col in columns_to_keep if col in filtered_players_df.columns]
    final_df = filtered_players_df[valid_columns_to_keep].reset_index(drop=True)
    print(f"Đã chọn các cột từ file stats: {valid_columns_to_keep}")
    return final_df

```

3. **combine_data_and_add_values(...) Function:**
    - Merges filtered statistics with scraped transfer values using fuzzy name matching (rapidfuzz).
    - Adds 'Transfer_Value' and 'Match_Score' columns to the statistics DataFrame.
    - **Drops players for whom no transfer value is found.**
    - Returns the combined DataFrame.

```python
def combine_data_and_add_values(
    df_filtered_stats: pd.DataFrame,
    df_scraped_values: pd.DataFrame,
    similarity_threshold: int
) -> pd.DataFrame:
    if df_scraped_values.empty:
        df_filtered_stats['Transfer_Value'] = None
        df_filtered_stats['Match_Score'] = None
        return df_filtered_stats
    df_combined = df_filtered_stats.copy()
    df_combined["Transfer_Value"] = None

    scraped_player_names = df_scraped_values['Player'].dropna().astype(str).tolist()
    if not scraped_player_names:
        print("CẢNH BÁO: Không có tên cầu thủ nào trong DataFrame từ web để so khớp.")
        return df_combined
    print(f"Thực hiện so khớp tên cho {len(df_combined)} cầu thủ đã lọc...")
    matched_count = 0
    for index, row in df_combined.iterrows():
        player_name_to_match = str(row["Player"])
        match_result = process.extractOne(
            player_name_to_match,
            scraped_player_names,
            scorer=fuzz.token_sort_ratio
        )
        if match_result:
            best_match_name, score, _ = match_result
            if score >= similarity_threshold:
                value_series = df_scraped_values.loc[df_scraped_values["Player"] == best_match_name, "Value_Scrap
                if not value_series.empty:
                    df_combined.at[index, "Transfer_Value"] = value_series.iloc[0]
                    df_combined.at[index, "Match_Score"] = score
                    matched_count += 1
    df_combined.dropna(subset=['Transfer_Value'], inplace=True)
    print(f"Số cầu thủ còn lại sau khi loại bỏ những người không có giá trị chuyển nhượng: {len(df_combined)}")
```

4. **main() Function:**
    - Orchestrates the entire workflow: calls scraping, filtering, and combining functions.
    - Includes a fallback to create a sample results.csv if the input stats file is not found, for testing.
    - Saves the final combined DataFrame to OUTPUT_FINAL_CSV_PATH.
    - Prints progress and summary messages.

```python
def main():
    # 1. Thu thập dữ liệu từ web
    df_scraped = scrape_player_values(BASE_URL_SCRAPING, NUMBER_OF_PAGES_TO_SCRAPE)
    try:
        with open(INPUT_STATS_CSV_PATH, 'r') as f:
            pass # Tệp tồn tại
    except FileNotFoundError:
        print(f"\nCẢNH BÁO: Tệp '{INPUT_STATS_CSV_PATH}' không tìm thấy. Tạo tệp mẫu...")
        sample_data_stats = {
            'Player': ['Robert Lewandowski', 'Kylian Mbappe', 'Erling Haaland', 'Mohamed Salah', 'Lionel Messi Jr
            'Nation': ['POL', 'FRA', 'NOR', 'EGY', 'ARG', 'BEL', 'SEN'],
            'Pos': ['FW', 'FW', 'FW', 'FW', 'FW', 'MF', 'FW'],
            'Squad': ['Barcelona', 'PSG', 'Man City', 'Liverpool', 'Inter Miami', 'Man City', 'Al-Nassr'],
            'Age': [35, 25, 23, 31, 36, 32, 31],
            'Min': [3000, 2800, 2500, 3200, 850, 2700, 700], # Một số < 900
        }
        pd.DataFrame(sample_data_stats).to_csv(INPUT_STATS_CSV_PATH, index=False)
        print(f"Tệp mẫu '{INPUT_STATS_CSV_PATH}' đã được tạo. Vui lòng kiểm tra và cập nhật nếu cần.")
    df_stats_filtered = read_and_filter_player_stats(
        INPUT_STATS_CSV_PATH,
        MINIMUM_PLAYING_TIME_MINS,
        COLUMNS_TO_KEEP_FROM_STATS_FILE
    )
    # 3. Kết hợp dữ liệu
    df_final_output = combine_data_and_add_values(
        df_stats_filtered,
        df_scraped,
        FUZZY_MATCH_SIMILARITY_THRESHOLD_PERCENT
    )
    print(f"\nTổng số cầu thủ trong kết quả cuối cùng: {len(df_final_output)}")
    print("\n5 dòng đầu của dữ liệu kết hợp cuối cùng:")
    print(df_final_output.head())
    # 4. Lưu kết quả
    df_final_output.to_csv(OUTPUT_FINAL_CSV_PATH, index=False, encoding='utf-8-sig')
    print("\n--- KẾT THÚC QUY TRÌNH ---")
```

5. **Execution Block (if __name__ == "__main__":)**
    - Standard Python entry point that calls main() when the script is run.

```
if __name__ == "__main__":
    main()
```