
BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



BÀI BÁO CÁO

MÔN: TOÁN RỜI RẠC 2

Giảng viên: Kim Ngọc Bách
Tên sinh viên: Nguyễn Minh Tuấn Kiệt
Mã sinh viên: B23DCCE060
Lớp: D23CQCE06 - B
Môn: Lập trình Python

Hà Nội, Tháng 5/2025

Bài 1:

Mã Nguồn Đầy Đủ

```
1 import time
2 import pandas as pd
3 from collections import Counter
4 from selenium import webdriver
5 from selenium.webdriver.chrome.service import Service
6 from selenium.webdriver.chrome.options import Options
7 from webdriver_manager.chrome import ChromeDriverManager
8 from bs4 import BeautifulSoup, Comment
9 import sys
10 import traceback
11 from selenium.webdriver.common.by import By
12 from selenium.webdriver.support.ui import WebDriverWait
13 from selenium.webdriver.support import expected_conditions as EC
14 from selenium.common.exceptions import TimeoutException
15
16 # Safe text retrieval function, returns 'N/a' on error
17 def safe_get_text(element, default='N/a'):
18     if not element: return default
19     text = element.get_text(strip=True)
20     return text or default
21
22 # Function to process nationality (for 3-letter codes)
23 def get_nationality(td_element):
24     if td_element is None: return 'N/a'
25     try:
26         strings = list(td_element.stripped_strings)
27         full_text = ' '.join(strings)
28         if not full_text: return 'N/a'
29         parts = full_text.split()
30         if not parts: return 'N/a'
31         # Prioritize standalone 3-letter uppercase codes
32         for i in range(len(parts) - 1, -1, -1):
33             part = parts[i]
34             if len(part) == 3 and part.isupper() and part.isalpha():
35                 return part
36         # If not found, check text within link (if any)
37         link = td_element.find('a')
38         if link:
39             link_text = safe_get_text(link)
40             if link_text != 'N/a' and len(link_text) >= 2 and len(link_text) <=
41                 4 and link_text.isupper() and link_text.isalpha():
42                 return link_text
43         # If still not found, take the last element if it's a country code (can
44             be 2-4 letters)
45         last_part = parts[-1]
46         if len(last_part) <= 4 and last_part.isupper() and last_part.isalpha():
```

```

45         return last_part
46     # Last case, if only one element and it's a country code
47     if len(parts) == 1 and len(parts[0]) <= 4 and parts[0].isupper() and
48         parts[0].isalpha():
49         return parts[0]
50     return 'N/a'
51 except Exception as e:
52     # print(f"Error processing nationality: {e} for element: {td_element}")
53     return 'N/a'
54
55 # Function to calculate age from birth year string or age string
56 def calculate_age(age_or_birth_str, current_year=None):
57     if current_year is None:
58         try: current_year = pd.Timestamp.now().year
59         except: current_year = 2025 # Fallback year
60
61     if not isinstance(age_or_birth_str, str) or age_or_birth_str == 'N/a':
62         return 'N/a'
63     age_or_birth_str = age_or_birth_str.strip()
64
65     # Case 1: String already contains age
66     try:
67         if '-' in age_or_birth_str:
68             age_part = age_or_birth_str.split('-')[0]
69             if age_part.isdigit() and 14 < int(age_part) < 50: return age_part
70             elif age_or_birth_str.isdigit() and 14 < int(age_or_birth_str) < 50:
71                 return age_or_birth_str
72     except (ValueError, TypeError): pass
73
74     # Case 2: String contains birth year
75     try:
76         if '-' in age_or_birth_str and len(age_or_birth_str.split('-')) == 3: #
77             YYYY-MM-DD format
78             parts = age_or_birth_str.split('-')
79             if len(parts[0]) == 4 and parts[0].isdigit():
80                 birth_year = int(parts[0])
81                 if 1900 < birth_year <= current_year: return str(current_year -
82                     birth_year)
83
84     year_part = ''.join(filter(str.isdigit, age_or_birth_str)) # Find any
85     4-digit year
86     if len(year_part) >= 4:
87         potential_years = [year_part[i:i+4] for i in range(len(year_part) -
88             3)]
89         for year_str in potential_years:
90             try:
91                 birth_year = int(year_str)
92                 if 1900 < birth_year <= current_year: return
93                     str(current_year - birth_year)
94             except ValueError: continue

```

```

88     if ',' in age_or_birth_str: # "Month Day, YYYY" format
89         year_str = age_or_birth_str.split(',')[1].strip()
90         if len(year_str) == 4 and year_str.isdigit():
91             birth_year = int(year_str)
92             if 1900 < birth_year <= current_year: return str(current_year
93                 - birth_year)
94
95     except (ValueError, TypeError, IndexError): pass
96
97     # Case 3: Only a 4-digit birth year
98     try:
99         if len(age_or_birth_str) == 4 and age_or_birth_str.isdigit():
100             birth_year = int(age_or_birth_str)
101             if 1900 < birth_year <= current_year: return str(current_year -
102                 birth_year)
103     except (ValueError, TypeError): pass
104     return 'N/a'
105
106 # Function to scrape a table from a given URL
107 def scrape_fbref_table(driver, url, table_id=None, required_stats=None,
108     min_minutes=90):
109     print(f"Attempting to scrape data from: {url}")
110     try:
111         driver.get(url)
112         print(" Page requested. Waiting for table...")
113
114         wait_time = 25
115         locator = (By.ID, table_id) if table_id else (By.CSS_SELECTOR,
116             "table.stats_table")
117         print(f" Waiting for table with {'ID: ' + table_id if table_id else
118             'class stats_table'}")
119
120         try:
121             WebDriverWait(driver,
122                 wait_time).until(EC.visibility_of_element_located(locator))
123             print(f" Table {locator} visible.")
124         except TimeoutException:
125             print(f" Warning: Table {locator} not visible within {wait_time}s
126                 on {url}. Checking HTML comments...")
127
128         time.sleep(1) # Allow JS to fully render
129         html = driver.page_source
130         soup = BeautifulSoup(html, 'html.parser')
131         print(" Parsed page source.")
132
133         data_table = soup.find('table', {'id': table_id}) if table_id else None
134         if not data_table: data_table = soup.find('table', {'class': lambda x:
135             x and 'stats_table' in x.split()})
136
137         if not data_table: # Check HTML comments
138             comments = soup.find_all(string=lambda text: isinstance(text,
139                 Comment))

```

```

130     for comment in comments:
131         comment_soup = BeautifulSoup(comment, 'html.parser')
132         potential_table = comment_soup.find('table', {'id': table_id})
133         if table_id else None
134         if not potential_table: potential_table =
135             comment_soup.find('table', {'class': lambda x: x and
136                 'stats_table' in x.split()})
137         if potential_table:
138             print(f" Found table {with ID {table_id} if table_id else
139                 ''} in HTML comment.")
140             data_table = potential_table
141             break
142         if not data_table:
143             print(f"Error: Table not found on {url}.")
144             return pd.DataFrame()
145
146     tbody = data_table.find('tbody')
147     rows = tbody.find_all('tr') if tbody else [r for r in
148         data_table.find_all('tr') if r.find(['th', 'td'], {'data-stat':
149             True}) and not r.find('th', {'scope': 'col'})]
150     if not tbody and not rows:
151         print(f"Error: No data rows found in table on {url}")
152         return pd.DataFrame()
153     elif not tbody:
154         print(f" Found {len(rows)} potential data rows directly in table.")
155
156     print(f" Found {len(rows)} rows for {url}. Processing...")
157
158     base_stats_needed = {'player', 'team', 'nationality', 'position',
159         'age', 'birth_year', 'minutes', 'minutes_90s'}
160     stats_to_extract = set(base_stats_needed)
161     if required_stats:
162         stats_to_extract.update(required_stats)
163     else: # If no required_stats, get from header
164         print(" Warning: No specific list of required stats, will fetch
165             from table header.")
166         thead = data_table.find('thead')
167         if thead and (header_rows := thead.find_all('tr')):
168             last_header_row = header_rows[-1]
169             header_stats = {th.get('data-stat', '').strip() for th in
170                 last_header_row.find_all('th')}
171             stats_to_extract.update(stat for stat in header_stats if stat
172                 and stat not in ['ranker', 'matches', 'match_report'])
173             print(f" Dynamically fetching stats from header:
174                 {sorted(list(stats_to_extract - base_stats_needed))}")
175
176     players_data = []
177     collected_count, skipped_header, skipped_minutes, skipped_no_player =
178         0, 0, 0, 0
179
180     for i, row in enumerate(rows):

```

```

169     if row.has_attr('class') and any(c in row['class'] for c in
170         ['thead', 'partial_table', 'spacer']):
171         skipped_header += 1; continue
172     if not row.find(['th', 'td'], {'data-stat' : True}): continue
173
174     player_cell = row.find(['th', 'td'], {'data-stat': 'player'})
175     player_name = safe_get_text(player_cell)
176     if player_name == 'N/a' or player_name == '' or player_name ==
177         'Player':
178         skipped_no_player += 1; continue
179
180     minutes_played_num = -1
181     minutes_td = row.find('td', {'data-stat': 'minutes'})
182     minutes_90s_td = row.find('td', {'data-stat': 'minutes_90s'})
183     minutes_str = safe_get_text(minutes_td, '').replace(',', '')
184     minutes_90s_str = safe_get_text(minutes_90s_td, '').replace(',', '')
185
186     try:
187         if minutes_str.isdigit():
188             minutes_played_num = int(minutes_str)
189         elif minutes_90s_str:
190             try: minutes_played_num = float(minutes_90s_str) * 90
191             except ValueError:
192                 if minutes_90s_str.isdigit(): minutes_played_num =
193                     int(minutes_90s_str) * 90
194                 else: minutes_played_num = -1
195         if not (minutes_played_num < 0 and minutes_td is None and
196             minutes_90s_td is None) and minutes_played_num <
197             min_minutes: # Allow players with no minutes data if fields
198                 are missing, otherwise filter
199                 skipped_minutes += 1; continue
200     except (ValueError, TypeError, AttributeError):
201         skipped_minutes += 1; continue
202
203     player_stats = {}
204     all_cells = row.find_all(['th', 'td'])
205     processed_stats_in_row = set()
206
207     for cell in all_cells:
208         stat = cell.get('data-stat', '').strip()
209         if stat and stat in stats_to_extract and stat not in
210             processed_stats_in_row:
211             processed_stats_in_row.add(stat)
212             if stat == 'nationality': player_stats['nationality'] =
213                 get_nationality(cell)
214             elif stat == 'birth_year' or stat == 'age': # 'age' column
215                 often contains birth year or age
216                 age_birth_text = safe_get_text(cell)
217                 if 'original_age_value' not in player_stats:
218                     player_stats['original_age_value'] = age_birth_text
219                 calculated_age = calculate_age(age_birth_text)

```

```

210         if calculated_age != 'N/a': player_stats['Age'] =
            calculated_age
211         elif player_stats.get('Age', 'N/a') == 'N/a':
            player_stats['Age'] = age_birth_text if
            age_birth_text != 'N/a' else 'N/a'
212         elif stat == 'player': player_stats['Player'] = player_name
213         elif stat == 'team':
214             team_name = safe_get_text(cell.find('a'),
                default=safe_get_text(cell))
215             player_stats['Team'] = team_name
216         elif stat == 'position':
217             position_text = safe_get_text(cell)
218             player_stats['Position'] =
                position_text.split(',')[0].strip() if ',' in
                position_text and
                position_text.split(',')[0].strip() else
                position_text
219         elif stat == 'minutes': player_stats['minutes'] =
            minutes_str or '0'
220         elif stat == 'minutes_90s': player_stats['minutes_90s'] =
            minutes_90s_str or '0.0'
221         else: player_stats[stat] = safe_get_text(cell)
222
223     # Fallbacks for essential columns if not picked up by general loop
224     player_stats.setdefault('Player', player_name)
225     if 'Team' not in player_stats:
226         team_td_fallback = row.find('td', {'data-stat': 'team'})
227         player_stats['Team'] =
            safe_get_text(team_td_fallback.find('a'),
                default=safe_get_text(team_td_fallback)) if team_td_fallback
            else 'N/a'
228     if 'Position' not in player_stats:
229         pos_td_fallback = row.find('td', {'data-stat': 'position'})
230         pos_text_fb = safe_get_text(pos_td_fallback)
231         player_stats['Position'] = (pos_text_fb.split(',')[0].strip()
            if ',' in pos_text_fb and pos_text_fb.split(',')[0].strip()
            else pos_text_fb) if pos_td_fallback else 'N/a'
232     if player_stats.get('Age', 'N/a') == 'N/a':
233         age_td_fallback = row.find('td', {'data-stat': 'age'})
234         age_text_fallback = safe_get_text(age_td_fallback)
235         player_stats['Age'] = calculate_age(age_text_fallback)
236         if 'original_age_value' not in player_stats:
            player_stats['original_age_value'] = age_text_fallback
237     if 'nationality' not in player_stats:
238         nat_td_fallback = row.find('td', {'data-stat': 'nationality'})
239         player_stats['nationality'] = get_nationality(nat_td_fallback)
240     player_stats.setdefault('minutes', minutes_str or '0')
241     player_stats.setdefault('minutes_90s', minutes_90s_str or '0.0')
242
243     players_data.append(player_stats)
244     collected_count += 1

```

```

245
246 print(f" Finished processing rows for {url}. Summary - Found:
      {len(rows)}, Skipped header: {skipped_header}, No player name:
      {skipped_no_player}, Low minutes ({min_minutes}):
      {skipped_minutes}, Collected: {collected_count}")
247 if not players_data:
248     print(f"Warning: No player data met criteria from {url}.")
249     return pd.DataFrame()
250
251 df = pd.DataFrame(players_data)
252 if 'Player' in df.columns and 'Team' in df.columns: # Deduplication
253     if 'minutes' in df.columns:
254         df['minutes_numeric'] =
            pd.to_numeric(df['minutes'].astype(str).str.replace(',',
            ''), errors='coerce').fillna(0)
255         df = df.sort_values(by=['Player', 'Team', 'minutes_numeric'],
            ascending=[True, True, False])
256         df = df.drop_duplicates(subset=['Player', 'Team'],
            keep='first').drop(columns=['minutes_numeric'])
257     else: df = df.drop_duplicates(subset=['Player', 'Team'],
            keep='first')
258
259     try: # Set index
260         df['Player'] = df['Player'].astype(str)
261         df['Team'] = df['Team'].astype(str)
262         if 'position' in df.columns and 'Position' not in df.columns:
263             df.rename(columns={'position': 'Position'}, inplace=True)
264         elif 'position' in df.columns and 'Position' in df.columns:
265             df.drop(columns=['position'], inplace=True)
266         df = df.set_index(['Player', 'Team'])
267         print(f" Created and indexed DataFrame for {url}. Shape:
            {df.shape}")
268     except KeyError as e:
269         print(f"Error setting index for {url}: {e}. Columns:
            {df.columns.tolist()}")
270         return df if not df.empty else pd.DataFrame()
271 else:
272     print(f"Error: Missing 'Player' or 'Team' in {url}. Columns:
        {df.columns.tolist()}")
273     return df if not df.empty else pd.DataFrame()
274 time.sleep(1.5) # Anti-blocking delay
275 return df
276 except TimeoutException as e:
277     print(f"Scraping error for {url}: Page element timed out. {e}")
278 except Exception as e:
279     print(f"Unknown error scraping {url}: {e}\nTraceback:
        {traceback.format_exc()}")
280 return pd.DataFrame()
# User-requested stats and FBRef mapping (Category, Sub-Category, Statistic
Name) -> FBRef Key

```



```

281 USER_REQUESTED_STAT_MAPPING = {
282     ('', '', 'Nation'): 'nationality', ('', '', 'Position'): 'Position', ('',
    '', 'Age'): 'Age',
283     ('Playing Time', '', 'MP'): 'games', ('Playing Time', '', 'Starts'):
    'games_starts', ('Playing Time', '', 'Min'): 'minutes',
284     ('Performance', '', 'Gls'): 'goals', ('Performance', '', 'Ast'):
    'assists', ('Performance', '', 'CrdY'): 'cards_yellow', ('Performance',
    '', 'CrdR'): 'cards_red',
285     ('Expected', '', 'xG'): 'xg', ('Expected', '', 'xAG'): 'xg_assist', #
    FBRef uses xAG
286     ('Progression', '', 'PrgC'): 'progressive_carries', ('Progression', '',
    'PrgP'): 'progressive_passes', ('Progression', '', 'PrgR'):
    'progressive_passes_received',
287     ('Per 90 Minutes', '', 'Gls'): 'goals_per90', ('Per 90 Minutes', '',
    'Ast'): 'assists_per90', ('Per 90 Minutes', '', 'xG'): 'xg_per90',
    ('Per 90 Minutes', '', 'xGA'): 'xg_assist_per90',
288     ('Goalkeeping', 'Performance', 'GA90'): 'gk_goals_against_per90',
    ('Goalkeeping', 'Performance', 'Save%'): 'gk_save_pct', ('Goalkeeping',
    'Performance', 'CS%'): 'gk_clean_sheets_pct', ('Goalkeeping', 'Penalty
    Kicks', 'Save%'): 'gk_pens_save_pct',
289     ('Shooting', 'Standard', 'SoT%'): 'shots_on_target_pct', ('Shooting',
    'Standard', 'SoT/90'): 'shots_on_target_per90', ('Shooting',
    'Standard', 'G/Sh'): 'goals_per_shot', ('Shooting', 'Standard',
    'Dist'): 'average_shot_distance',
290     ('Passing', 'Total', 'Cmp'): 'passes_completed', ('Passing', 'Total',
    'Cmp%'): 'passes_pct', ('Passing', 'Total', 'TotDist'):
    'passes_total_distance',
291     ('Passing', 'Short', 'Cmp%'): 'passes_pct_short', ('Passing', 'Medium',
    'Cmp%'): 'passes_pct_medium', ('Passing', 'Long', 'Cmp%'):
    'passes_pct_long',
292     ('Passing', 'Expected', 'KP'): 'assisted_shots', ('Passing', 'Expected',
    '1/3'): 'passes_into_final_third', ('Passing', 'Expected', 'PPA'):
    'passes_into_penalty_area', ('Passing', 'Expected', 'CrsPA'):
    'crosses_into_penalty_area', ('Passing', 'Expected', 'PrgP'):
    'progressive_passes',
293     ('Goal and Shot Creation', 'SCA', 'SCA'): 'sca', ('Goal and Shot
    Creation', 'SCA', 'SCA90'): 'sca_per90', ('Goal and Shot Creation',
    'GCA', 'GCA'): 'gca', ('Goal and Shot Creation', 'GCA', 'GCA90'):
    'gca_per90',
294     ('Defensive Actions', 'Tackles', 'Tkl'): 'tackles', ('Defensive Actions',
    'Tackles', 'TklW'): 'tackles_won', ('Defensive Actions', 'Challenges',
    'Att'): 'challenges', ('Defensive Actions', 'Challenges', 'Lost'):
    'challenges_lost',
295     ('Defensive Actions', 'Blocks', 'Blocks'): 'blocks', ('Defensive Actions',
    'Blocks', 'Sh'): 'blocked_shots', ('Defensive Actions', 'Blocks',
    'Pass'): 'blocked_passes', ('Defensive Actions', 'Blocks', 'Int'):
    'interceptions',
296     ('Possession', 'Touches', 'Touches'): 'touches', ('Possession', 'Touches',
    'Def Pen'): 'touches_def_pen_area', ('Possession', 'Touches', 'Def
    3rd'): 'touches_def_3rd', ('Possession', 'Touches', 'Mid 3rd'):
    'touches_mid_3rd', ('Possession', 'Touches', 'Att 3rd'):

```

```

    'touches_att_3rd', ('Possession', 'Touches', 'Att Pen')):
    'touches_att_pen_area',
297 ('Possession', 'Take-Ons', 'Att'): 'take_ons', ('Possession', 'Take-Ons',
    'Succ%'): 'take_ons_won_pct', ('Possession', 'Take-Ons', 'Tkld%'):
    'take_ons_tackled_pct',
298 ('Possession', 'Carries', 'Carries'): 'carries', ('Possession', 'Carries',
    'PrgDist'): 'carries_progressive_distance', ('Possession', 'Carries',
    'ProgC'): 'progressive_carries', ('Possession', 'Carries', '1/3'):
    'carries_into_final_third', ('Possession', 'Carries', 'CPA'):
    'carries_into_penalty_area', ('Possession', 'Carries', 'Mis'):
    'miscontrols', ('Possession', 'Carries', 'Dis'): 'dispossessed',
299 ('Possession', 'Receiving', 'Rec'): 'passes_received', ('Possession',
    'Receiving', 'PrgR'): 'progressive_passes_received',
300 ('Miscellaneous Stats', 'Performance', 'Fls'): 'fouls', ('Miscellaneous
    Stats', 'Performance', 'Fld'): 'fouled', ('Miscellaneous Stats',
    'Performance', 'Off'): 'offsides', ('Miscellaneous Stats',
    'Performance', 'Crs'): 'crosses', ('Miscellaneous Stats',
    'Performance', 'Recov'): 'ball_recoveries',
301 ('Miscellaneous Stats', 'Aerial Duels', 'Won'): 'aerials_won',
    ('Miscellaneous Stats', 'Aerial Duels', 'Lost'): 'aerials_lost',
    ('Miscellaneous Stats', 'Aerial Duels', 'Won%'): 'aerials_won_pct',
302 }
303 required_fbref_keys = set(USER_REQUESTED_STAT_MAPPING.values()) | {'player',
    'team', 'birth_year', 'minutes_90s'} # Add basic keys
304 print(f"\nTargeting {len(required_fbref_keys)} FBRef keys for scraping
    (user-requested + basic).")
305
306 urls = {
307     'standard': 'https://fbref.com/en/comps/9/stats/Premier-League-Stats',
    'shooting':
    'https://fbref.com/en/comps/9/shooting/Premier-League-Stats',
308 'passing': 'https://fbref.com/en/comps/9/passing/Premier-League-Stats',
    'gca': 'https://fbref.com/en/comps/9/gca/Premier-League-Stats',
309 'defense': 'https://fbref.com/en/comps/9/defense/Premier-League-Stats',
    'possession':
    'https://fbref.com/en/comps/9/possession/Premier-League-Stats',
310 'misc': 'https://fbref.com/en/comps/9/misc/Premier-League-Stats',
    'keepers': 'https://fbref.com/en/comps/9/keepers/Premier-League-Stats',
311 }
312 table_ids = {
313     'standard': 'stats_standard', 'shooting': 'stats_shooting', 'passing':
    'stats_passing', 'gca': 'stats_gca',
314 'defense': 'stats_defense', 'possession': 'stats_possession',
    'playingtime': 'stats_playing_time', # Keep ID if URL re-enabled
315 'misc': 'stats_misc', 'keepers': 'stats_keeper',
316 }
317
318 print("\nSetting up Selenium WebDriver...")
319 try:
320     options = Options()
321     # options.add_argument("--headless")

```

```

322 options.add_argument("--no-sandbox")
323 options.add_argument("--disable-dev-shm-usage")
324 options.add_argument("--log-level=3") # Reduce browser logs
325 options.add_argument("user-agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36")
326 options.add_argument("--disable-blink-features=AutomationControlled")
327 options.add_experimental_option("excludeSwitches", ["enable-automation"])
328 options.add_experimental_option('useAutomationExtension', False)
329 try:
330     service = Service(ChromeDriverManager().install())
331     driver = webdriver.Chrome(service=service, options=options)
332     print("WebDriver using ChromeDriverManager.")
333 except Exception as driver_manager_err:
334     print(f"Warning: ChromeDriverManager failed ({driver_manager_err}).
        Trying default ChromeDriver from PATH...")
335     driver = webdriver.Chrome(options=options) # Fallback to system PATH
336     print("WebDriver using ChromeDriver from system PATH.")
337 driver.execute_script("Object.defineProperty(navigator, 'webdriver', {get:
    () => undefined})") # Hide webdriver property
338 print("WebDriver setup complete.")
339 except Exception as e:
340     print(f"Critical error during WebDriver setup: {e}")
341     sys.exit(1)
342
343 all_dfs = {}
344 MIN_MINUTES_PLAYED = 90
345 scraping_successful = False
346
347 print("\n--- Starting to scrape data from URLs ---")
348 for category, url in urls.items():
349     table_id = table_ids.get(category)
350     df_cat = scrape_fbref_table(driver, url, table_id=table_id,
        min_minutes=MIN_MINUTES_PLAYED, required_stats=required_fbref_keys)
351     if df_cat is not None and not df_cat.empty:
352         cols_to_keep = [col for col in df_cat.columns if col in
            required_fbref_keys]
353         if cols_to_keep:
354             all_dfs[category] = df_cat[cols_to_keep]
355             print(f"--> Success: Fetched data for {category}
                ({all_dfs[category].shape[0]} players, {len(cols_to_keep)}
                stats)")
356             scraping_successful = True
357             else: print(f"--> Warning: {category} contained no required stats.")
358         else: print(f"--> Warning: Fetching failed or no data for {category} from
            {url}")
359     print("-" * 30)
360 driver.quit()
361
362 if not scraping_successful or not all_dfs:
363     print("ERROR: No data successfully fetched. Cannot continue.")
364     sys.exit(1)

```

```

365
366 print("\n--- Merging scraped DataFrames ---")
367 merged_df = None
368 df_keys_priority = ['standard', 'keepers'] + [k for k in all_dfs.keys() if k
369 not in ['standard', 'keepers']]
369 for category in df_keys_priority:
370     if category not in all_dfs or all_dfs[category].empty: continue
371     df_cat = all_dfs[category]
372     if merged_df is None: merged_df = df_cat
373     else:
374         try:
375             merged_df = merged_df.merge(df_cat, left_index=True,
376                                         right_index=True, how='outer', suffixes=(None, f'__{category}'))
377         except Exception as merge_error:
378             print(f"CRITICAL ERROR merging '{category}': {merge_error}") #
379             Potentially log more details or stop
380     print(f" {'Started with' if merged_df is df_cat else 'Merged'}
381           '{category}'. Current shape: {merged_df.shape if merged_df is not None
382           else 'N/A'}")
383 if merged_df is None:
384     print("ERROR: No DataFrames merged. Cannot create result file.")
385     sys.exit(1)
386 print(f"\nInitial merge complete. Total unique Player/Team pairs:
387       {len(merged_df)}")
388 merged_df = merged_df.reset_index().fillna('N/a')
389
390 print("\n--- Building final DataFrame based on user request ---")
391 final_df = pd.DataFrame({'Player': merged_df['Player'], 'Team':
392 merged_df['Team']})
393 final_columns_structure = []
394 missing_stats_log = []
395 processed_fbref_keys_final = {'Player', 'Team'}
396 merged_df_columns_list = merged_df.columns.tolist()
397
398 def find_column_match(df_columns, base_key, suffix_marker='_'):
399     if base_key in df_columns: return base_key
400     suffixed_cols = [c for c in df_columns if isinstance(c, str) and
401 c.startswith(base_key + suffix_marker)]
402     return suffixed_cols[0] if suffixed_cols else None
403
404 print(f"Processing {len(USER_REQUESTED_STAT_MAPPING)} requested stats...")
405 for col_tuple, base_key in USER_REQUESTED_STAT_MAPPING.items():
406     final_columns_structure.append(col_tuple)
407     matched_col = find_column_match(merged_df_columns_list, base_key)
408     if matched_col:
409         final_df[col_tuple] = merged_df[matched_col]
410         processed_fbref_keys_final.add(matched_col)
411         if matched_col != base_key: missing_stats_log.append(f"Used suffixed
412 '{matched_col}' for {col_tuple} (orig: {base_key})")
413     else:
414         final_df[col_tuple] = 'N/a'
415

```

```

407         missing_stats_log.append(f"Missing {col_tuple} (orig: {base_key}).")
408
409     print("\n--- Checks and Reports ---")
410     unused_original_columns = [col for col in merged_df_columns_list if col not in
411                               processed_fbref_keys_final]
412     if unused_original_columns: print(f"Info: {len(unused_original_columns)}
413                                     unrequested columns dropped. (e.g., {'',
414                                     '.join(sorted(unused_original_columns)[:5])){'...' if
415                                     len(unused_original_columns) > 5 else ''})")
416     if missing_stats_log:
417         print("Warning - Mapping issues:")
418         for warning in sorted(list(set(missing_stats_log))): print(f" -
419                             {warning}")
420     else: print("All requested stats mapped successfully.")
421     print("-----\n")
422
423     print("Creating column index for final DataFrame...")
424     try:
425         multiindex_tuples = [('', '', 'Player'), ('', '', 'Team')] +
426                             final_columns_structure
427         if len(multiindex_tuples) == final_df.shape[1]:
428             final_df.columns = pd.MultiIndex.from_tuples(multiindex_tuples,
429                                                         names=['Category', 'Sub-Category', 'Statistic'])
430             print("MultiIndex created.")
431         else: raise ValueError(f"Column count mismatch for MultiIndex: DF has
432                               {final_df.shape[1]}, tuples {len(multiindex_tuples)}.")
433     except Exception as multiindex_error:
434         print(f"Error creating MultiIndex: {multiindex_error}. Using flat column
435               names as fallback.")
436         flat_fallback_cols = ['Player', 'Team'] + ['_'.join(filter(None, map(str,
437                                     tpl))) for tpl in final_columns_structure]
438         final_df.columns = [f"{col}_{i}" if flat_fallback_cols.count(col) > 1
439                             else col for i, col in enumerate(flat_fallback_cols)]
440
441     is_multiindex = isinstance(final_df.columns, pd.MultiIndex)
442     player_col_id = ('', '', 'Player') if is_multiindex else 'Player'
443     if player_col_id in final_df.columns:
444         try:
445             final_df = final_df.sort_values(by=player_col_id, ascending=True,
446                                             key=lambda col: col.astype(str).str.lower(), na_position='last')
447             print("Sorted DataFrame by Player name.")
448         except Exception as e: print(f"Warning: Could not sort by Player
449                                     ({player_col_id}): {e}.")
450     else: print(f"Warning: Player column '{player_col_id}' not found for sorting.")
451
452     print("\nReordering final columns...")
453     PRIORITY_COLS_TUPLE = [('', '', 'Player'), ('', '', 'Team'), ('', '',
454                             'Nation'), ('', '', 'Position'), ('', '', 'Age')]
455     PRIORITY_COLS_FLAT = ['Player', 'Team', 'Nation', 'Position', 'Age']
456     priority_cols_definition = PRIORITY_COLS_TUPLE if is_multiindex else
457                               PRIORITY_COLS_FLAT

```

```

443 all_current_cols = final_df.columns.tolist()
444 priority_cols_present = [col for col in priority_cols_definition if col in
    all_current_cols]
445 other_cols = sorted([col for col in all_current_cols if col not in
    priority_cols_present])
446 final_column_order = priority_cols_present + other_cols
447 try:
448     final_df = final_df[final_column_order]
449     print("Column reordering successful.")
450 except Exception as e: print(f"Error reordering columns: {e}.")
451
452 print("\nPreparing to export final CSV file...")
453 final_df_export = final_df.copy()
454 if isinstance(final_df_export.columns, pd.MultiIndex):
455     print("Flattening MultiIndex columns for CSV...")
456     flat_columns = []
457     processed_flat_names = set()
458     for col_tuple in final_df_export.columns:
459         parts = [str(c).strip().replace(' ', '_').replace('/',
            '_').replace('%',
            'Pct').replace('+/-', '_Net').replace('#', 'Num').replace('(', '').replace(')', '')
            for c in col_tuple if str(c).strip()]
460         base_flat_col = '_'.join(parts) if parts else f"col_{len(flat_columns)}"
461         original_base = base_flat_col
462         current_count = 1
463         while base_flat_col in processed_flat_names:
464             base_flat_col = f"{original_base}_{current_count}"; current_count
                += 1
465         flat_columns.append(base_flat_col)
466         processed_flat_names.add(base_flat_col)
467     if len(flat_columns) == final_df_export.shape[1]: final_df_export.columns
        = flat_columns
468     else:
469         print(f"CRITICAL ERROR: Column count mismatch after flattening
            ({len(flat_columns)} vs {final_df_export.shape[1]}). Aborting
            save.")
470         sys.exit(1)
471
472 print("Reordering flattened columns for export...")
473 PRIORITY_COLS_FLAT_FINAL = ['Player', 'Team', 'Nation', 'Position', 'Age']
474 id_cols_flat_final = [c for c in PRIORITY_COLS_FLAT_FINAL if c in
    final_df_export.columns]
475 other_cols_flat_final = sorted([c for c in final_df_export.columns if c not in
    id_cols_flat_final])
476 final_export_order_flat = id_cols_flat_final + other_cols_flat_final
477 try:
478     final_df_export = final_df_export[final_export_order_flat]
479     print("Flat column reordering for export successful.")
480 except Exception as e: print(f"Error reordering flat columns for export: {e}")
481
482 output_filename = 'results.csv'

```

```

483 print(f"\nSaving final results to {output_filename}...")
484 try:
485     if final_df_export.empty or final_df_export.shape[1] == 0: print("Warning:
        Final DataFrame is empty or has no columns. Saving empty CSV.")
486     missing_protected = [col for col in id_cols_flat_final if col not in
        final_df_export.columns]
487     if missing_protected: print(f"CRITICAL WARNING: Basic ID columns lost
        before saving: {missing_protected}.")
488     final_df_export.to_csv(output_filename, index=False, encoding='utf-8-sig')
489     print(f"Successfully saved results to {output_filename}. Shape:
        {final_df_export.shape}")
490     print(f"Final columns (first 25):
        {final_df_export.columns.tolist()[:25]}{'...' if
        len(final_df_export.columns) > 25 else ''}")
491 except Exception as e:
492     print(f"ERROR saving CSV '{output_filename}': {e}\nTraceback:
        {traceback.format_exc()}")
493 print("\n--- Script complete ---")

```

Giải thích đoạn code

Thư viện cần cài đặt

Các thư viện Python cần thiết cho đoạn code bao gồm:

- **time**: Thư viện chuẩn để làm việc với thời gian.
- **pandas**: Thư viện để xử lý và phân tích dữ liệu.
- **collections**: Thư viện chuẩn, sử dụng **Counter**.
- **selenium**: Thư viện để tự động hóa trình duyệt web.
- **webdriver_manager**: Thư viện để quản lý driver cho Selenium.
- **bs4** (BeautifulSoup): Thư viện để phân tích cú pháp HTML và XML.
- **sys**: Thư viện chuẩn để làm việc với hệ thống.
- **traceback**: Thư viện chuẩn để in ra thông tin lỗi.

Để cài đặt các thư viện không chuẩn, sử dụng lệnh:

```
1 pip install pandas selenium webdriver-manager beautifulsoup4
```

Cách code vận hành chung

Đoạn code được thiết kế để thu thập dữ liệu bóng đá từ trang web **FBRef.com**, cụ thể là từ các bảng thống kê của Premier League. Nó sử dụng **Selenium** để tự động hóa trình duyệt web, **BeautifulSoup** để phân tích HTML, và **pandas** để xử lý dữ liệu.

Quy trình chung của code như sau:

-
1. **Thiết lập Selenium WebDriver:** Khởi tạo trình duyệt Chrome với các tùy chọn để tránh bị phát hiện là bot.
 2. **Thu thập dữ liệu từ các URL:** Sử dụng hàm `scrape_fbref_table` để lấy dữ liệu từ các bảng thống kê khác nhau.
 3. **Xử lý và làm sạch dữ liệu:** Sử dụng các hàm như `safe_get_text`, `get_nationality`, và `calculate_age`.
 4. **Hợp nhất dữ liệu:** Hợp nhất các DataFrame từ các bảng thành một DataFrame duy nhất.
 5. **Tạo DataFrame cuối cùng:** Tạo DataFrame theo yêu cầu người dùng dựa trên `USER_REQUESTED_STAT_MAPPING`.
 6. **Xuất dữ liệu ra file CSV:** Lưu DataFrame vào file `results.csv`.

Giải thích từng hàm chính

`safe_get_text(element, default='N/a')`

- **Mục đích:** Trích xuất văn bản từ phần tử HTML một cách an toàn.
- **Cách hoạt động:**
 - Nếu `element` là `None`, trả về `default`.
 - Lấy văn bản từ `element` và loại bỏ khoảng trắng thừa.
 - Nếu văn bản rỗng, trả về `default`.

`get_nationality(td_element)`

- **Mục đích:** Trích xuất mã quốc tịch từ ô HTML.
- **Cách hoạt động:**
 - Nếu `td_element` là `None`, trả về `'N/a'`.
 - Lấy tất cả chuỗi văn bản từ `td_element`.
 - Tìm mã quốc tịch 3 chữ cái in hoa.
 - Nếu không tìm thấy, kiểm tra văn bản trong liên kết (nếu có).
 - Nếu vẫn không tìm thấy, lấy phần cuối nếu là mã quốc tịch (2-4 chữ cái).

`calculate_age(age_or_birth_str, current_year=None)`

- **Mục đích:** Tính tuổi từ chuỗi chứa tuổi hoặc năm sinh.
- **Cách hoạt động:**
 - Nếu chuỗi là số từ 14 đến 50, trả về tuổi đó.
 - Nếu chuỗi chứa năm sinh (ví dụ: "1990-01-01"), tính tuổi bằng cách trừ năm sinh từ năm hiện tại.
 - Xử lý các định dạng như "YYYY-MM-DD" hoặc "Month Day, YYYY".
 - Nếu không xử lý được, trả về `'N/a'`.

```
scrape_fbref_table(driver, url, table_id=None, required_stats=None, min_minutes=90)
```

- **Mục đích:** Thu thập dữ liệu từ bảng trên FBRef.com.
- **Cách hoạt động:**
 - Mở URL bằng Selenium.
 - Đợi bảng xuất hiện với `table_id` hoặc class `stats_table`.
 - Phân tích HTML bằng BeautifulSoup.
 - Trích xuất dữ liệu từ các hàng, lọc người chơi có số phút dưới `min_minutes`.
 - Tạo DataFrame từ dữ liệu thu thập.

```
find_column_match(df_columns, base_key, suffix_marker='__')
```

- **Mục đích:** Tìm cột trong DataFrame khớp với `base_key`.
- **Cách hoạt động:**
 - Nếu `base_key` tồn tại trong danh sách cột, trả về nó.
 - Nếu không, tìm cột bắt đầu bằng `base_key` + `suffix_marker`.

Kết quả đầu ra của file CSV

Kết quả của đoạn code là một file CSV tên `results.csv`, chứa dữ liệu thống kê bóng đá từ các bảng trên trang FBRef.com (Premier League). Dưới đây là mô tả chi tiết về cấu trúc và nội dung của file:

- **Cấu trúc cột:**
 - * File chứa các cột được định nghĩa trong `USER_REQUESTED_STAT_MAPPING`, bao gồm:
 - *Thông tin cơ bản:* **Player** (tên cầu thủ), **Team** (đội bóng), **Nation** (quốc tịch, mã 2-4 chữ cái in hoa), **Position** (vị trí thi đấu), **Age** (tuổi của cầu thủ).
 - *Thống kê hiệu suất:* Số trận (**MP**), số lần ra sân từ đầu (**Starts**), số phút thi đấu (**Min**), số bàn thắng (**Goals**), số kiến tạo (**Ast**), xG (**xG**), xAG (**xAG**), số thẻ vàng (**CrdY**), số thẻ đỏ (**CrdR**), v.v.
 - *Thống kê chi tiết:* Tỷ lệ sút trúng đích (**SoT%**), số đường chuyền hoàn thành (**Cmp**), tỷ lệ chuyền bóng chính xác (**Cmp%**), số lần tắc bóng (**Tkl**), số lần đánh chặn (**Int**), số lần chạm bóng (**Touches**), số lần rê bóng thành công (**Succ%**), v.v.
 - *Thống kê thủ môn (nếu có):* Tỷ lệ cứu thua (**Save%**), tỷ lệ giữ sạch lưới (**CS%**), tỷ lệ cứu penalty (**Save%**), v.v.
 - * Các cột được sắp xếp với thứ tự ưu tiên: **Player**, **Team**, **Nation**, **Position**, **Age**, sau đó là các cột thống kê khác theo thứ tự alphabet.
- **Định dạng dữ liệu:**

- * Mỗi hàng đại diện cho một cầu thủ duy nhất trong một đội bóng, với các giá trị thống kê tương ứng.
- * Giá trị thiếu được điền bằng 'N/a'.
- * Dữ liệu được sắp xếp theo cột **Player** (theo thứ tự bảng chữ cái, không phân biệt hoa thường).
- * File sử dụng mã hóa **utf-8-sig** để hỗ trợ các ký tự đặc biệt (như tên cầu thủ có dấu).

– **Ví dụ nội dung** (minh họa, dữ liệu thực tế phụ thuộc vào `FBRef.com`):

```
1 Player,Team,Nation,Position,Age,MP,Starts,Min,Gls,Ast,xG,xAG,...
2 Mohamed Salah,Liverpool,EGY,FW,32,30,28,2500,15,10,12.5,8.0,...
3 Erling Haaland,Manchester City,NOR,FW,24,32,30,2700,25,5,20.0,3.5,...
```

– **Đặc điểm bổ sung:**

- * Các cột được đặt tên phẳng (flat) để phù hợp với định dạng CSV, với các ký tự đặc biệt (như %, /, -) được thay thế bằng các chuỗi hợp lệ (ví dụ: % thành Pct, - thành _).
- * File không chứa cột chỉ mục (`index=False`), đảm bảo đầu ra là bảng dữ liệu thuần túy.
- * Nếu không thu thập được dữ liệu (do lỗi mạng, trang web thay đổi cấu trúc, v.v.), file có thể rỗng hoặc chỉ chứa tiêu đề cột.

Bài 2:

Mã Nguồn Đầy Đủ

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5 import sys
6 import traceback
7 import re
8
9 # --- Configuration ---
10 INPUT_CSV = 'results.csv'
11 OUTPUT_TOP_BOTTOM = 'top_3.txt'
12 OUTPUT_STATS_SUMMARY = 'results2.csv'
13 OUTPUT_HISTOGRAM_DIR = 'histograms'
14 HIST_SUBDIR_ALL = 'all_players'
15 HIST_SUBDIR_TEAMS = 'by_team'
16 OUTPUT_HIGHEST_SCORING_TEAMS = 'highest_scoring_teams.txt' # For highest
    scoring teams output
17
18 ID_COLS = ['Player', 'Team', 'Nation', 'Position', 'Age']
19
20 # Patterns for selecting specific stats for histograms
21 HISTOGRAM_OFFENSIVE_PATTERNS = [
```

```

22     'gl', 'goal', 'sh', 'shot', 'sot', 'xg', 'npxg', 'xa', 'assist',
        'keypass', 'kp',
23     'sca', 'gca', 'att_pen', 'crspa', 'succ_dribbles', 'prog_passes_rec',
        'touches_att_pen',
24     'progcarry', 'progpas' # Added for progressive carries/passes if named as
        such
25 ]
26 HISTOGRAM_DEFENSIVE_PATTERNS = [
27     'tkl', 'tackle', 'tklw', 'int', 'interception', 'block', 'clr',
        'clearance',
28     'sav', 'save', 'cs', 'clean_sheet', 'ga', 'goals_against', 'err', # 'ga'
        and 'goals_against' for goals against
29     'crdy', 'crdr', 'card', 'foul', 'aerialswon', 'pkcon', 'pressure',
        'recover'
30 ]
31
32
33 # --- Helper Functions ---
34 def clean_numeric_column(series):
35     series_str = series.astype(str)
36     series_cleaned = series_str.str.replace('%', '', regex=False)
37     series_cleaned = series_cleaned.str.replace(',', '', regex=False)
38     series_numeric = pd.to_numeric(series_cleaned, errors='coerce')
39     return series_numeric
40
41 def get_numeric_columns(df, exclude_cols):
42     numeric_cols = []
43     potential_cols = [col for col in df.columns if col not in exclude_cols]
44     print(f" Potentially analyzing {len(potential_cols)} columns (excluding:
        {', '.join(exclude_cols)})")
45     original_dtypes = df[potential_cols].dtypes
46
47     for col in potential_cols:
48         if pd.api.types.is_numeric_dtype(original_dtypes[col]):
49             numeric_cols.append(col)
50             continue
51         try:
52             # Attempt to coerce to numeric, then check ratio of valid numbers
53             coerced = pd.to_numeric(df[col].astype(str).str.replace(',', ''),
                errors='coerce')
54             valid_ratio = coerced.notna().sum() / len(coerced) if len(coerced)
                > 0 else 0
55             if valid_ratio > 0.1: # Consider a column numeric if more than 10%
                can be converted
56                 numeric_cols.append(col)
57         except Exception as e:
58             print(f" Skipping column '{col}' due to error during numeric check:
                {e}", file=sys.stderr)
59     return sorted(list(set(numeric_cols)))
60
61 def format_player_list(series):

```

```

62     return [f"{player} ({score})" for player, score in series.items()]
63
64 # --- Main Analysis Logic ---
65 if __name__ == "__main__":
66     print(f"Loading data from {INPUT_CSV}...")
67     try:
68         df = pd.read_csv(INPUT_CSV)
69         print(f"Data loaded successfully. Shape: {df.shape}")
70         if df.empty:
71             print(f"Error: {INPUT_CSV} is empty. Cannot proceed.",
72                   file=sys.stderr)
73             sys.exit(1)
74         print(f"Original columns ({len(df.columns)}): {'',
75               '.join(df.columns[:min(10, len(df.columns))])}'...)") # Show first 10
76         if 'Player' in df.columns: df['Player'] = df['Player'].astype(str)
77         if 'Team' in df.columns: df['Team'] = df['Team'].astype(str)
78     except FileNotFoundError:
79         print(f"Error: {INPUT_CSV} not found. Please ensure the file exists.",
80               file=sys.stderr)
81         sys.exit(1)
82     except Exception as e:
83         print(f"Error loading {INPUT_CSV}: {e}", file=sys.stderr)
84         print(traceback.format_exc(), file=sys.stderr)
85         sys.exit(1)
86
87     print("\nApplying cleaning to potential numeric columns...")
88     df_cleaned = df.copy()
89     potential_numeric_cols_for_cleaning = [col for col in df.columns if col
90                                           not in ID_COLS]
91     cleaned_count = 0
92     for col in potential_numeric_cols_for_cleaning:
93         if col in df_cleaned.columns and not
94             pd.api.types.is_numeric_dtype(df_cleaned[col]):
95             df_cleaned[col] = clean_numeric_column(df_cleaned[col])
96             cleaned_count += 1
97     print(f"Attempted cleaning on {cleaned_count} non-numeric columns
98           (excluding ID cols).")
99
100    print("\nIdentifying numeric columns for analysis after cleaning...")
101    NON_STAT_COLS = list(ID_COLS)
102    # Add specific playing time columns from your CSV structure to
103    NON_STAT_COLS
104    # Common playing time columns that might appear from Problem1.py
105    # (e.g., Playing_Time_Min, Playing_Time_MP, Playing_Time_Starts)
106    # Adjust these based on the actual output of Problem1.py if needed
107    generated_playing_time_cols = ['Playing_Time_Min', 'Playing_Time_MP',
108                                  'Playing_Time_Starts', 'Min', 'MP', 'Starts']
109    for pt_col in generated_playing_time_cols:
110        if pt_col in df_cleaned.columns:
111            NON_STAT_COLS.append(pt_col)
112    NON_STAT_COLS = sorted(list(set(NON_STAT_COLS))) # Ensure unique

```

```

105 stat_cols = get_numeric_columns(df_cleaned, NON_STAT_COLS)
106
107
108 if not stat_cols:
109     print("\nError: No numeric statistic columns identified after
110         cleaning.", file=sys.stderr)
111     print("Please check the input CSV structure and the
112         cleaning/identification logic.")
113     print(f" Columns excluded as non-stats: {NON_STAT_COLS}")
114     sys.exit(1)
115
116 print(f"\nIdentified {len(stat_cols)} numeric statistic columns for
117     analysis.")
118 # Example: print first 5 stat_cols
119 print(f" Sample stats: {'', '.join(stat_cols[:min(5, len(stat_cols))])}'...")
120 # GK stats identification (remains useful)
121 potential_gk_cols_in_stats = [c for c in stat_cols if 'gk' in c.lower() or
122     'goal' in c.lower() or 'sav' in c.lower() or 'pk' in c.lower() or 'ga'
123     in c.lower() or 'cs' in c.lower()]
124 if potential_gk_cols_in_stats:
125     print(f" Potential GK stats identified: {'',
126         '.join(sorted(potential_gk_cols_in_stats))}'")
127 else:
128     print(" No columns matching typical Goalkeeping patterns found in
129         identified stats.")
130 df_numeric = df_cleaned
131
132 print(f"\nCalculating Top/Bottom 3 players per statistic ->
133     {OUTPUT_TOP_BOTTOM}")
134 try:
135     with open(OUTPUT_TOP_BOTTOM, 'w', encoding='utf-8') as f:
136         f.write("Top and Bottom 3 Players per Statistic\n")
137         f.write("=====\n\n")
138         for col in stat_cols:
139             if 'Player' not in df_numeric.columns:
140                 f.write(f"--- {col} ---\n")
141                 f.write("Error: 'Player' column not found. Cannot determine
142                     top/bottom players.\n\n")
143                 continue
144
145             # Ensure the column actually exists in df_numeric before
146             proceeding
147             if col not in df_numeric.columns:
148                 f.write(f"--- {col} ---\n")
149                 f.write(f"Error: Column '{col}' not found in DataFrame for
150                     Top/Bottom analysis.\n\n")
151                 continue
152
153             stat_df = df_numeric[['Player', col]].copy()
154             stat_df[col] = pd.to_numeric(stat_df[col], errors='coerce') #
155                 Coerce to numeric

```

```

144         stat_df.dropna(subset=[col], inplace=True)
145
146     if stat_df.empty:
147         f.write(f"--- {col} ---\n")
148         f.write(f"No valid numeric data for this statistic\n\n")
149         continue
150     try:
151         highest = stat_df.sort_values(by=col,
152                                     ascending=False).set_index('Player')[col]
153         lowest = stat_df.sort_values(by=col,
154                                     ascending=True).set_index('Player')[col]
155         top_3 = highest.head(3)
156         bottom_3 = lowest.head(3)
157         f.write(f"--- {col} ---\n")
158         f.write("Top 3:\n")
159         if not top_3.empty:
160             for player, score in top_3.items():
161                 score_str = f"{score:.2f}" if pd.notna(score) else "N/A"
162                 f.write(f" - {player}: {score_str}\n")
163         else:
164             f.write(" (No players found for Top 3)\n")
165         f.write("\nBottom 3:\n")
166         if not bottom_3.empty:
167             for player, score in bottom_3.items():
168                 score_str = f"{score:.2f}" if pd.notna(score) else "N/A"
169                 f.write(f" - {player}: {score_str}\n")
170         else:
171             f.write(" (No players found for Bottom 3)\n")
172         f.write("\n-----\n\n")
173     except Exception as sort_err:
174         f.write(f"--- {col} ---\n")
175         f.write(f"Error sorting data for statistic '{col}':\n\n")
176         f.write(f"-----\n\n")
177     print("Top/Bottom 3 players saved.")
178 except Exception as e:
179     print(f"Error during Task 1 (Top/Bottom 3): {e}", file=sys.stderr)
180     print(traceback.format_exc(), file=sys.stderr)
181
182 print(f"\nCalculating Median, Mean, Std Dev per statistic ->
183       {OUTPUT_STATS_SUMMARY}")
184 results_data = []
185 try:
186     if not stat_cols:
187         print("Warning: No numeric stats columns identified for Task 2.",
188               file=sys.stderr)
189     else:
190         # Ensure only existing stat_cols are used for aggregation

```

```

187     valid_stat_cols_for_agg = [sc for sc in stat_cols if sc in
188                               df_numeric.columns]
189     if not valid_stat_cols_for_agg:
190         print("Warning: None of the identified stat_cols exist in the
191               DataFrame for aggregation.", file=sys.stderr)
192     else:
193         global_agg = df_numeric[valid_stat_cols_for_agg].agg(['median',
194                     'mean', 'std'])
195         for stat in valid_stat_cols_for_agg:
196             if stat in global_agg.columns:
197                 results_data.append({
198                     'Team': 'all',
199                     'Statistic': stat,
200                     'Median': global_agg.loc['median', stat],
201                     'Mean': global_agg.loc['mean', stat],
202                     'Std': global_agg.loc['std', stat]
203                 })
204             else:
205                 print(f"Warning: Statistic '{stat}' not found in global
206                       aggregation results.", file=sys.stderr)
207
208 if 'Team' in df_numeric.columns:
209     valid_teams_df =
210         df_numeric[df_numeric['Team'].astype(str).str.lower() != 'all']
211     if not valid_teams_df.empty:
212         teams_for_grouping = valid_teams_df['Team'].unique()
213         if len(teams_for_grouping) > 0 and valid_stat_cols_for_agg: #
214             Also check if there are stats to group by
215             grouped =
216                 valid_teams_df.groupby('Team')[valid_stat_cols_for_agg]
217             if not grouped.groups:
218                 print("Warning: Grouping by 'Team' resulted in empty
219                       groups.", file=sys.stderr)
220             else:
221                 try:
222                     team_agg = grouped.agg(['median', 'mean', 'std'])
223                     if team_agg.empty:
224                         print("Warning: Aggregation per team produced
225                               empty results.", file=sys.stderr)
226                     else:
227                         for team_name_idx in team_agg.index:
228                             for stat_col_name_agg in
229                                 valid_stat_cols_for_agg:
230                                 median_key = (stat_col_name_agg, 'median')
231                                 mean_key = (stat_col_name_agg, 'mean')
232                                 std_key = (stat_col_name_agg, 'std')
233                                 if median_key in team_agg.columns and
234                                    mean_key in team_agg.columns and
235                                    std_key in team_agg.columns:
236                                     results_data.append({
237                                         'Team': team_name_idx,

```

```

226         'Statistic': stat_col_name_agg,
227         'Median':
228             team_agg.loc[team_name_idx,
229                 median_key],
230         'Mean': team_agg.loc[team_name_idx,
231             mean_key],
232         'Std': team_agg.loc[team_name_idx,
233             std_key]
234     })
235 except Exception as group_agg_e:
236     print(f"Error during per-team aggregation:
237         {group_agg_e}", file=sys.stderr)
238     print(traceback.format_exc(), file=sys.stderr)
239 elif not valid_stat_cols_for_agg:
240     print("Warning: No valid stat columns to perform per-team
241         aggregation.", file=sys.stderr)
242 else:
243     print("Warning: No unique teams found for per-team
244         statistics (excluding 'all').", file=sys.stderr)
245 else:
246     print("Warning: DataFrame became empty after filtering out
247         'all' team. No per-team stats.", file=sys.stderr)
248 else:
249     print("Warning: 'Team' column not found. Cannot calculate per-team
250         statistics.", file=sys.stderr)
251
252 if not results_data:
253     print("Error: No statistics could be calculated for Task 2.",
254         file=sys.stderr)
255 else:
256     summary_long_df = pd.DataFrame(results_data)
257     summary_long_df.fillna(value=np.nan, inplace=True)
258     if summary_long_df.empty or not {'Team', 'Statistic', 'Median',
259         'Mean', 'Std'}.issubset(summary_long_df.columns):
260         print("Error: Cannot create pivot table due to missing columns
261             or empty data frame after aggregation.", file=sys.stderr)
262     else:
263         try:
264             summary_pivot = summary_long_df.pivot_table(
265                 index='Team', columns='Statistic', values=['Median',
266                     'Mean', 'Std']
267             )
268             if isinstance(summary_pivot.columns, pd.MultiIndex):
269                 summary_pivot.columns =
270                     summary_pivot.columns.swaplevel(0, 1)
271                 metric_order = pd.CategoricalDtype(['Median', 'Mean',
272                     'Std'], ordered=True)
273                 summary_pivot.sort_index(axis=1, level=0, inplace=True)
274                 summary_pivot.sort_index(axis=1, level=1, key=lambda x:
275                     x.astype(metric_order), inplace=True)

```



```

260         summary_pivot.columns = [f"{metric} of {stat}" for stat,
261                                   metric in summary_pivot.columns]
262     summary_pivot = summary_pivot.reset_index()
263     if 'all' in summary_pivot['Team'].values:
264         all_row = summary_pivot[summary_pivot['Team'] == 'all']
265         other_rows = summary_pivot[summary_pivot['Team'] !=
266                                     'all'].sort_values(by='Team')
267         summary_pivot = pd.concat([all_row, other_rows],
268                                   ignore_index=True)
269     summary_pivot.to_csv(OUTPUT_STATS_SUMMARY, index=False,
270                          encoding='utf-8-sig', float_format='%.3f')
271     print(f"Median/Mean/Std Dev summary saved to
272           {OUTPUT_STATS_SUMMARY}")
273 except Exception as pivot_e:
274     print(f"Error during pivoting or formatting results for Task
275           2: {pivot_e}", file=sys.stderr)
276     print(traceback.format_exc(), file=sys.stderr)
277 except Exception as e:
278     print(f"Error during Task 2 (Median/Mean/Std Dev): {e}",
279           file=sys.stderr)
280     print(traceback.format_exc(), file=sys.stderr)
281
282 # --- Task: Identify teams with the highest average score per statistic
283 # (Requirement 1) ---
284 print(f"\nIdentifying teams with the highest average score per statistic
285       -> {OUTPUT_HIGHEST_SCORING_TEAMS}")
286 highest_scoring_teams_dict = {} # Renamed to avoid conflict
287 if 'Team' in df_numeric.columns and stat_cols: # Check if stat_cols is not
288     empty
289     try:
290         valid_teams_df_for_means =
291             df_numeric[df_numeric['Team'].astype(str).str.lower() !=
292                         'all'].copy()
293         # Ensure only existing stat_cols are used
294         existing_stat_cols_for_means = [sc for sc in stat_cols if sc in
295                                         valid_teams_df_for_means.columns]
296
297         if not valid_teams_df_for_means.empty and
298             existing_stat_cols_for_means:
299             for col in existing_stat_cols_for_means: # Ensure numeric type
300                 for mean calculation
301                 valid_teams_df_for_means[col] =
302                     pd.to_numeric(valid_teams_df_for_means[col],
303                                   errors='coerce')
304
305         team_means =
306             valid_teams_df_for_means.groupby('Team')[existing_stat_cols_for_means].r
307
308         if not team_means.empty:
309             with open(OUTPUT_HIGHEST_SCORING_TEAMS, 'w',
310                     encoding='utf-8') as f_highest:

```

```

292         f_highest.write("Team with Highest Average Score per
                Statistic\n")
293         f_highest.write("=====\n\n")
294         for col in existing_stat_cols_for_means:
295             if col in team_means.columns and
                team_means[col].notna().any():
296                 try:
297                     best_team_idx = team_means[col].idxmax()
298                     highest_score_val = team_means[col].max()
299                     highest_scoring_teams_dict[col] =
                        (best_team_idx, highest_score_val)
300                     f_highest.write(f"- Highest Avg {col}:
                        {best_team_idx}
                        ({highest_score_val:.2f})\n")
301                     # print(f"- Highest Avg {col}:
                        {best_team_idx}
                        ({highest_score_val:.2f})" # Optional
                        console print
302                 except ValueError:
303                     f_highest.write(f"- Highest Avg {col}: N/A
                        (all values NaN or empty after
                        grouping)\n")
304                 except Exception as idxmax_e:
305                     f_highest.write(f"- Highest Avg {col}: Error
                        ({idxmax_e})\n")
306             else:
307                 f_highest.write(f"- Highest Avg {col}: N/A
                        (column data insufficient or all NaN)\n")
308             print(f"Highest scoring team data saved to
                {OUTPUT_HIGHEST_SCORING_TEAMS}")
309         else:
310             print("Warning: Calculating team means resulted in an empty
                DataFrame.", file=sys.stderr)
311     elif not existing_stat_cols_for_means:
312         print("Warning: No valid statistic columns found in DataFrame
                to calculate team means.", file=sys.stderr)
313     else: # valid_teams_df_for_means is empty
314         print("Warning: No valid team data (excluding 'all') to
                calculate highest scores.", file=sys.stderr)
315     except Exception as e:
316         print(f"Error during Highest Team Scores task: {e}",
                file=sys.stderr)
317         print(traceback.format_exc(), file=sys.stderr)
318     elif not 'Team' in df_numeric.columns:
319         print("Warning: 'Team' column not found. Cannot perform Highest Team
                Scores task.", file=sys.stderr)
320     else: # stat_cols is empty
321         print("Warning: No numeric statistics identified. Cannot perform
                Highest Team Scores task.", file=sys.stderr)
322
323

```

```

324 # --- Filter stats for histograms (Requirement 2) ---
325 print("\nSelecting Offensive and Defensive statistics for histogram
      generation...")
326 stats_for_histograms = []
327 if stat_cols: # Ensure stat_cols is not empty
328     all_hist_patterns = HISTOGRAM_OFFENSIVE_PATTERNS +
      HISTOGRAM_DEFENSIVE_PATTERNS
329     for col in stat_cols:
330         if col not in df_numeric.columns: continue # Skip if col somehow
      isn't in df
331         col_lower = col.lower()
332         if any(pattern in col_lower for pattern in all_hist_patterns):
333             stats_for_histograms.append(col)
334
335 if not stats_for_histograms:
336     print("Warning: No offensive or defensive statistics identified for
      histogram plotting based on current patterns.", file=sys.stderr)
337 else:
338     print(f"Identified {len(stats_for_histograms)} offensive/defensive
      stats for histograms: {'', '.join(stats_for_histograms[:min(5,
      len(stats_for_histograms))])}...")
339
340
341 print(f"\nGenerating histograms for Offensive/Defensive Stats ->
      {OUTPUT_HISTOGRAM_DIR}/")
342 hist_path_all = os.path.join(OUTPUT_HISTOGRAM_DIR, HIST_SUBDIR_ALL)
343 hist_path_teams = os.path.join(OUTPUT_HISTOGRAM_DIR, HIST_SUBDIR_TEAMS)
344 try:
345     os.makedirs(hist_path_all, exist_ok=True)
346     os.makedirs(hist_path_teams, exist_ok=True)
347 except OSError as e:
348     print(f"Error creating histogram directories: {e}", file=sys.stderr)
349     # Decide if to exit or continue: sys.exit(1) or pass
350
351 plot_errors_all = 0
352 plots_generated_all = 0
353 plot_errors_teams = 0
354 plots_generated_teams = 0
355
356 for col in stats_for_histograms: # Use the filtered list
357     try:
358         data_to_plot_all = df_numeric[col].dropna() # Ensure col exists
359         if data_to_plot_all.empty or not
360             pd.api.types.is_numeric_dtype(data_to_plot_all):
361             pass
362         else:
363             plt.figure(figsize=(10, 6))
364             plt.hist(data_to_plot_all, bins=20, edgecolor='black',
365                     color='skyblue')
366             plt.title(f'Distribution of {col} (All Players)')
367             plt.xlabel(col)

```

```

366         plt.ylabel('Frequency (Number of Players)')
367         plt.grid(axis='y', alpha=0.75)
368         safe_col_name = "".join(c if c.isalnum() else "_" for c in col)
369         plot_filename_all = os.path.join(hist_path_all,
370                                         f'hist_all_{safe_col_name}.png')
371         plt.savefig(plot_filename_all)
372         plt.close()
373         plots_generated_all += 1
374     except Exception as e:
375         plot_errors_all += 1
376         print(f"Error generating histogram for {col} (All Players): {e}",
377               file=sys.stderr)
378         plt.close()
379
380     if 'Team' in df_numeric.columns:
381         # Ensure team names are strings for filtering and file naming
382         teams_list = df_numeric[df_numeric['Team'].astype(str).str.lower()
383                                != 'all']['Team'].astype(str).dropna().unique()
384         if len(teams_list) == 0:
385             continue
386         for team_name_str in teams_list:
387             try:
388                 team_data = df_numeric[df_numeric['Team'] ==
389                                       team_name_str][col].dropna()
390                 if team_data.empty or not
391                     pd.api.types.is_numeric_dtype(team_data):
392                     continue
393                 plt.figure(figsize=(8, 5))
394                 plt.hist(team_data, bins=15, edgecolor='black',
395                         color='lightcoral')
396                 plt.title(f'Distribution of {col} for {team_name_str}',
397                          fontsize=10)
398                 plt.xlabel(col, fontsize=9)
399                 plt.ylabel('Frequency', fontsize=9)
400                 plt.xticks(fontsize=8); plt.yticks(fontsize=8)
401                 plt.grid(axis='y', alpha=0.6)
402                 safe_col_name = "".join(c if c.isalnum() else "_" for c in
403                                         col)
404                 safe_team_name = "".join(c if c.isalnum() else "_" for c in
405                                         team_name_str) # team_name_str is already string
406                 plot_filename_team = os.path.join(hist_path_teams,
407                                                    f'hist_{safe_team_name}_{safe_col_name}.png')
408                 plt.savefig(plot_filename_team)
409                 plt.close()
410                 plots_generated_teams += 1
411             except Exception as e:
412                 plot_errors_teams += 1
413                 print(f"Error generating histogram for {col} -
414                       {team_name_str}: {e}", file=sys.stderr)
415                 plt.close()

```

```

406 print(f"\nHistograms generation summary (Offensive/Defensive Stats):")
407 print(f" - All Players: {plots_generated_all} successful,
      {plot_errors_all} errors.")
408 if 'Team' in df_numeric.columns:
409     print(f" - Per Team: {plots_generated_teams} successful,
      {plot_errors_teams} errors.")
410
411 # --- Best Performing Team Analysis (using highest_scoring_teams_dict) ---
412 print("\n--- Best Performing Team Analysis (Based on Average Stats) ---")
413 analysis_text = "Based on the average statistics per team:\n"
414 team_mentions_high = {}
415 team_mentions_low = {}
416
417 # Define patterns for interpreting stats (can be reused or adjusted)
418 # These are used for the textual analysis part.
419 LOWER_IS_BETTER_PATTERNS = ['ga', 'goals_against', 'offside', 'fls',
      'foul', 'lost', 'crd', 'card', 'pkcon', 'err_leading_to_shot'] # Added
      more specific
420 KEY_OFFENSIVE_PATTERNS_TEXT = ['gls', 'goal', 'xg', 'sot', 'sca', 'gca',
      'att_pen', 'shot', 'assist', 'key_pass', 'prog_pass_rec']
421 KEY_DEFENSIVE_PATTERNS_TEXT = ['tklw', 'tackles_won', 'int',
      'interception', 'block', 'clr', 'clearance', 'sav', 'save', 'cs',
      'clean_sheet', 'aerial_won']
422 KEY_POSSESSION_PATTERNS_TEXT = ['cmp_pct', 'pass_accuracy', 'prgp',
      'progressive_pass', 'prgc', 'progressive_carr', 'touch', 'progression',
      'prog']
423
424
425 if highest_scoring_teams_dict:
426     current_team_means_for_analysis = pd.DataFrame() # Initialize
427     if 'Team' in df_numeric.columns and stat_cols:
428         temp_df = df_numeric[df_numeric['Team'].astype(str).str.lower() !=
      'all'].copy()
429         valid_cols = [c for c in stat_cols if c in temp_df.columns]
430         if valid_cols:
431             for c in valid_cols: temp_df[c] = pd.to_numeric(temp_df[c],
      errors='coerce')
432             current_team_means_for_analysis =
      temp_df.groupby('Team')[valid_cols].mean()
433
434     for stat, (team, score) in highest_scoring_teams_dict.items():
435         stat_lower = stat.lower()
436         is_lower_better_stat = any(pattern in stat_lower for pattern in
      LOWER_IS_BETTER_PATTERNS)
437
438         if is_lower_better_stat:
439             if not current_team_means_for_analysis.empty and stat in
      current_team_means_for_analysis.columns and
      current_team_means_for_analysis[stat].notna().any():
440                 try:

```

```

441         min_team_idx =
442             current_team_means_for_analysis[stat].idxmin()
443             team_mentions_low[min_team_idx] =
444                 team_mentions_low.get(min_team_idx, 0) + 1
445         except ValueError: pass # All NaN for this stat
446         except Exception as min_err: print(f" (Error determining min
447             for {stat}: {min_err})")
448     else:
449         team_mentions_high[team] = team_mentions_high.get(team, 0) + 1
450
451 most_mentioned_high = sorted(team_mentions_high.items(), key=lambda
452     item: item[1], reverse=True)
453 most_mentioned_low = sorted(team_mentions_low.items(), key=lambda item:
454     item[1], reverse=True)
455
456 if most_mentioned_high:
457     analysis_text += f"- '{most_mentioned_high[0][0]}' leads
458         {most_mentioned_high[0][1]} 'higher-is-better' stats.\n"
459 if most_mentioned_low:
460     analysis_text += f"- '{most_mentioned_low[0][0]}' leads
461         {most_mentioned_low[0][1]} 'lower-is-better' stats.\n"
462
463 def get_leaders_text_analysis(patterns, means_df, lower_better_def):
464     leaders = {} # Store as {team: count_of_leading_stats}
465     if means_df.empty: return {"N/A"}
466     for stat_col in means_df.columns:
467         stat_col_lower = stat_col.lower()
468         if any(p in stat_col_lower for p in patterns):
469             if means_df[stat_col].notna().any():
470                 try:
471                     best_team = means_df[stat_col].idxmin() if any(lb in
472                         stat_col_lower for lb in lower_better_def) else
473                         means_df[stat_col].idxmax()
474                     leaders[best_team] = leaders.get(best_team, 0) + 1
475                 except ValueError: pass # All NaN
476     # Return teams sorted by how many relevant stats they lead
477     sorted_leaders = sorted(leaders.items(), key=lambda x: x[1],
478         reverse=True)
479     return {team for team, count in sorted_leaders[:3]} if
480         sorted_leaders else {"N/A"} # Top 3 or N/A
481
482 if not current_team_means_for_analysis.empty:
483     off_leaders =
484         get_leaders_text_analysis(KEY_OFFENSIVE_PATTERNS_TEXT,
485             current_team_means_for_analysis, LOWER_IS_BETTER_PATTERNS)
486     def_leaders = get_leaders_text_analysis(KEY_DEFENSIVE_PATTERNS_TEXT
487         + LOWER_IS_BETTER_PATTERNS, current_team_means_for_analysis,
488             LOWER_IS_BETTER_PATTERNS) # include lower_is_better in def
489     patterns

```

```

475     poss_leaders =
        get_leaders_text_analysis(KEY_POSSESSION_PATTERNS_TEXT,
        current_team_means_for_analysis, LOWER_IS_BETTER_PATTERNS)
476     analysis_text += f"- Offensive Leaders (top teams by # of led
        stats): {'', '.join(sorted(list(off_leaders)))]}\n"
477     analysis_text += f"- Defensive Leaders: {'',
        '.join(sorted(list(def_leaders)))]}\n"
478     analysis_text += f"- Possession Leaders: {'',
        '.join(sorted(list(poss_leaders)))]}\n"
479     else:
480         analysis_text += "Could not generate detailed categorical leaders
            as team means were not available for analysis.\n"
481
482     else:
483         analysis_text += "Could not determine highest scoring teams, so further
            detailed analysis is limited.\n"
484
485     analysis_text += "\nDisclaimer: This analysis is based solely on average
        player statistics per team derived from the input data..."
486     print(analysis_text)
487     try:
488         with open("team_performance_analysis_summary.txt", "w",
            encoding="utf-8") as f_analysis:
489             f_analysis.write(analysis_text)
490             print("\nTeam performance analysis summary saved to
                team_performance_analysis_summary.txt")
491     except Exception as e_write_analysis:
492         print(f"Error writing team performance analysis summary:
            {e_write_analysis}")
493
494     print("\n--- Analysis Finished ---")

```

Giải thích chi tiết về mã code

Tổng quan về mã code

Mã code này được viết bằng Python và sử dụng các thư viện như `pandas`, `numpy`, và `matplotlib` để phân tích dữ liệu bóng đá từ file `results.csv`. Nó thực hiện các tác vụ sau:

- **Đọc dữ liệu:** Tải dữ liệu từ file CSV.
- **Làm sạch dữ liệu:** Chuyển đổi các cột không phải số thành dạng số.
- **Phân tích thống kê:** Tính toán top/bottom 3 cầu thủ, median, mean, và độ lệch chuẩn (std) cho từng chỉ số và từng đội.
- **Xác định đội xuất sắc:** Tìm đội có điểm trung bình cao nhất cho mỗi chỉ số.
- **Tạo biểu đồ:** Vẽ histogram cho các chỉ số tấn công và phòng thủ.
- **Tóm tắt hiệu suất:** Phân tích và ghi kết quả hiệu suất đội vào file.

Các hàm chính và cách hoạt động

Hàm `clean_numeric_column(series)`

- **Mục đích:** Chuyển đổi một cột dữ liệu (series) chứa ký tự không phải số (như %, ,) thành dạng số.
- **Cách hoạt động:**
 1. Chuyển series thành chuỗi bằng `astype(str)`.
 2. Loại bỏ ký tự % và , bằng `str.replace`.
 3. Chuyển đổi thành số bằng `pd.to_numeric`, nếu lỗi thì trả về NaN.
- **Ví dụ:** "15%" → 15.0, "1,234" → 1234.0.

Hàm `get_numeric_columns(df, exclude_cols)`

- **Mục đích:** Xác định các cột trong DataFrame có thể là số, ngoại trừ các cột trong `exclude_cols`.
- **Cách hoạt động:**
 1. Lọc các cột không nằm trong `exclude_cols`.
 2. Kiểm tra kiểu dữ liệu gốc; nếu là số thì thêm vào danh sách.
 3. Nếu không phải số, thử chuyển đổi và tính tỷ lệ giá trị hợp lệ (>10% thì coi là cột số).
- **Ví dụ:** Với DataFrame có cột ['Player', 'Goals', 'Assists'] và `exclude_cols=['Player']` hàm trả về ['Goals', 'Assists'].

Hàm `format_player_list(series)`

- **Mục đích:** Định dạng danh sách cầu thủ và điểm số thành chuỗi để ghi file.
- **Cách hoạt động:** Tạo danh sách chuỗi "player (score)" từ series.
- **Ví dụ:** Series {"Salah": 28, "Haaland": 21} → ["Salah (28)", "Haaland (21)"].

Phần chính (if `__name__ == "__main__":`)

- **Đọc dữ liệu:** Tải `results.csv` bằng `pd.read_csv`. Nếu lỗi (file không tồn tại, trống), chương trình dừng.
- **Làm sạch dữ liệu:** Áp dụng `clean_numeric_column` cho các cột không phải số (ngoại trừ `ID_COLS`).
- **Xác định cột số:** Gọi `get_numeric_columns` để lấy danh sách `stat_cols`.
- **Top/Bottom 3:**
 - Sắp xếp từng cột số, lấy 3 giá trị cao nhất/thấp nhất cùng tên cầu thủ.
 - Ghi vào `top_3.txt`.

- **Median, Mean, Std:**

- Tính toán cho toàn bộ dữ liệu và từng đội bằng `agg(['median', 'mean', 'std'])`.
- Ghi vào `results2.csv`.

- **Đội cao nhất:**

- Tính trung bình từng chỉ số theo đội, tìm đội có giá trị cao nhất.
- Ghi vào `highest_scoring_teams.txt`.

- **Histogram:**

- Lọc chỉ số tấn công/phòng thủ dựa trên `HISTOGRAM_OFFENSIVE_PATTERNS` và `HISTOGRAM_DEFENSIVE_PATTERNS`.
- Vẽ histogram cho tất cả cầu thủ (`all_players`) và từng đội (`by_team`), lưu vào `histograms`.

- **Phân tích đội:**

- Đếm số chỉ số mà mỗi đội dẫn đầu (cao hơn tốt hoặc thấp hơn tốt).
- Xác định đội dẫn đầu về tấn công, phòng thủ, kiểm soát bóng.
- Ghi vào `team_performance_analysis_summary.txt`.

Kết quả đầu ra

File `top_3.txt`

- **Nội dung:** Top 3 và bottom 3 cầu thủ cho mỗi chỉ số.
- **Ví dụ:**

```
--- Defensive_Actions_Blocks_Blocks ---
Top 3:
- Nathan Collins: 69.00
- Murillo: 66.00
- Tyrick Mitchell: 62.00
Bottom 3:
- Aaron Ramsdale: 0.00
- Ali Al Hamadi: 0.00
- Alisson: 0.00
```

File `results2.csv`

- **Nội dung:** Median, mean, std của từng chỉ số, cho tất cả cầu thủ (`all`) và từng đội.
- **Ví dụ:**

```
Team,Median of Defensive_Actions_Blocks_Blocks,Mean of Defensive_Actions_Blocks
all,11.000,15.062,13.699,...
Arsenal,10.000,13.682,11.167,...
```

File `highest_scoring_teams.txt`

- **Nội dung:** Đội có trung bình cao nhất cho mỗi chỉ số.
- **Ví dụ:**
 - Highest Avg Defensive_Actions_Blocks_Blocks: Brentford (20.38)
 - Highest Avg Defensive_Actions_Blocks_Int: Bournemouth (13.83)

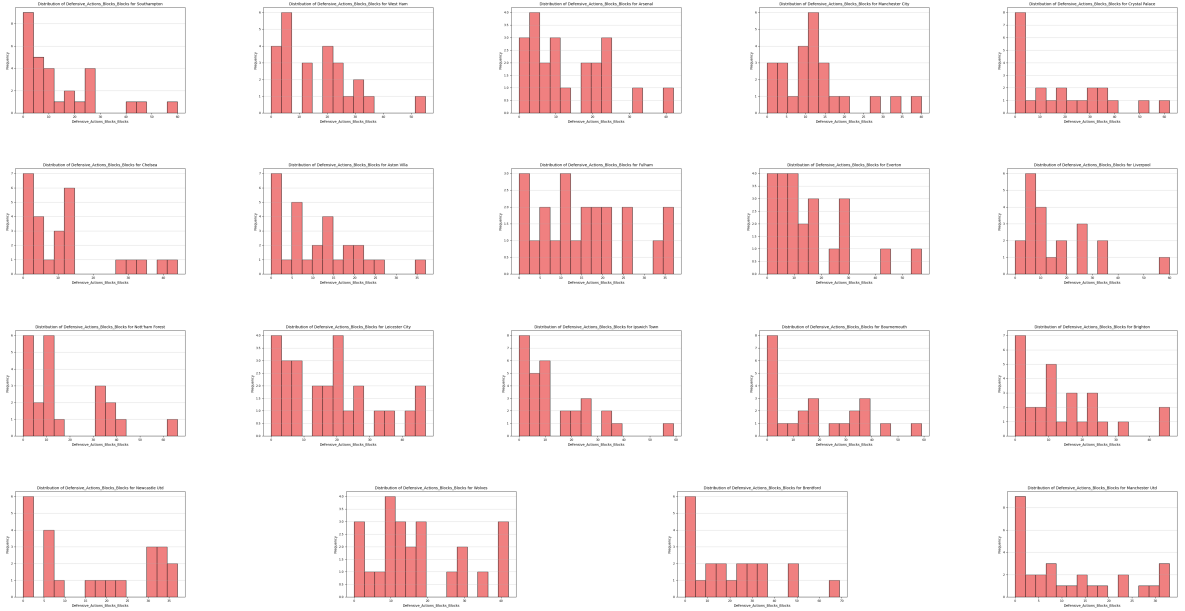
Thư mục `histograms`

- **Nội dung:** Biểu đồ histogram cho các chỉ số tấn công/phòng thủ.
- **Cấu trúc:**
 - `all_players`: Histogram cho tất cả cầu thủ (ví dụ: `hist_all_Defensive_Actions_Blocks`)
 - `by_team`: Histogram cho từng đội (ví dụ: `hist_Arsenal_Defensive_Actions_Blocks_Blo`)

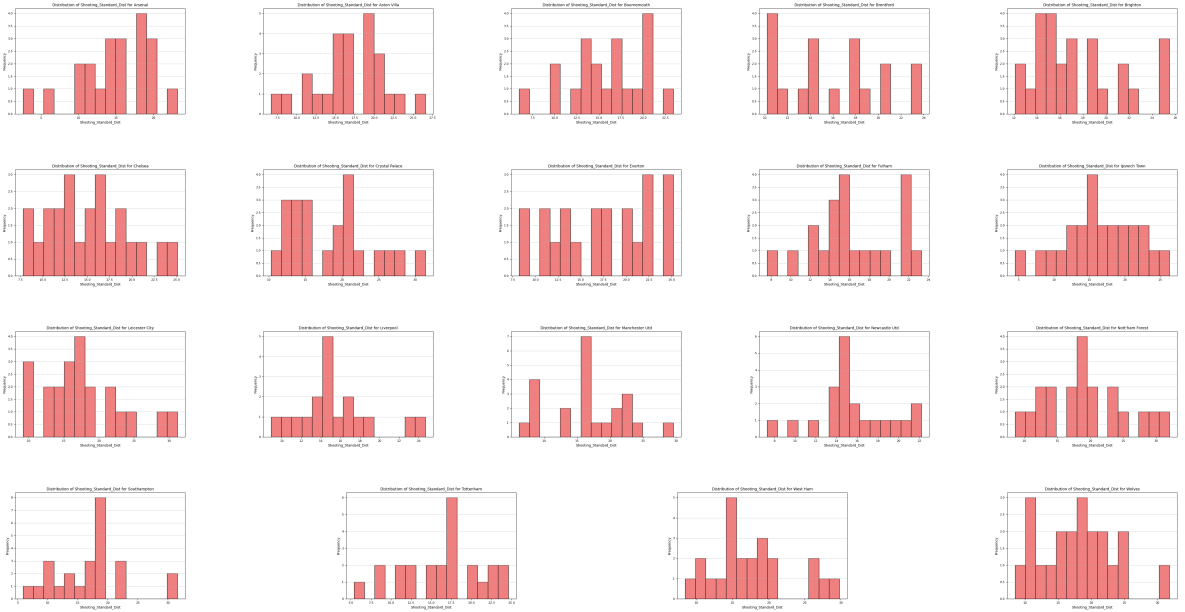
File `team_performance_analysis_summary.txt`

- **Nội dung:** Tóm tắt hiệu suất đội dựa trên số chỉ số dẫn đầu.
- **Ví dụ:**

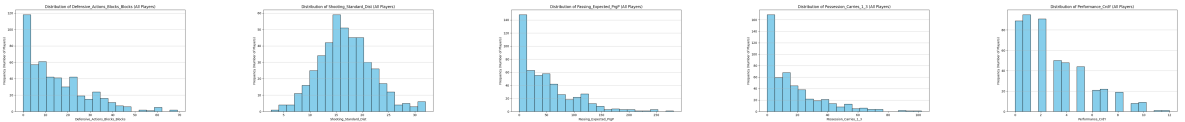
```
Based on the average statistics per team:
- 'Liverpool' leads 23 'higher-is-better' stats.
- 'Manchester City' leads 2 'lower-is-better' stats.
- Offensive Leaders: Brentford, Everton, Liverpool
- Defensive Leaders: Bournemouth, Brentford, Manchester City
- Possession Leaders: Brentford, Liverpool, Manchester City
```



Hình 1: Chỉ số phòng thủ của các đội



Hình 2: Chỉ số tấn công của các đội



Hình 3: Chỉ số của tất cả các cầu thủ

Bài 3:

Mã nguồn đầy đủ

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.cluster import KMeans
4 from sklearn.preprocessing import StandardScaler, OneHotEncoder
5 from sklearn.impute import SimpleImputer
6 from sklearn.compose import ColumnTransformer
7 from sklearn.pipeline import Pipeline
8 from sklearn.decomposition import PCA
9 import matplotlib.pyplot as plt
10 import seaborn as sns
11
12 # Load the dataset
13 try:
14     df = pd.read_csv('result.csv')
15     print(f"Successfully loaded result.csv. Dataset size: {df.shape}")
16 except FileNotFoundError:
17     print("Error: File 'result.csv' not found.")
18     print("Please ensure you have run the BTL-BAI1.py script first and the CSV
19         file is created in the same directory.")
20     exit()
21 except Exception as e:
22     print(f"Error while reading CSV file: {e}")
23     exit()
24
25 # Extract player information
26 player_info = df[['Player', 'Team', 'Position', 'Age']].copy()
27
28 # Identify numeric and categorical features
29 potential_numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
30 cols_to_exclude = ['Age']
31 numeric_features = [col for col in potential_numeric_cols if col not in
32                     cols_to_exclude]
33 categorical_features = ['Position']
34
35 # Create features dataframe
36 features_df = df[numeric_features + categorical_features].copy()
37
38 # Define preprocessing pipelines
39 numeric_transformer = Pipeline(steps=[
40     ('imputer', SimpleImputer(strategy='mean')),
41     ('scaler', StandardScaler())
42 ])
43
44 categorical_transformer = Pipeline(steps=[
45     ('imputer', SimpleImputer(strategy='most_frequent')),
46     ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
47 ])
```

```

46
47 # Combine preprocessors
48 preprocessor = ColumnTransformer(
49     transformers=[
50         ('num', numeric_transformer, numeric_features),
51         ('cat', categorical_transformer, categorical_features)
52     ],
53     remainder='drop'
54 )
55
56 # Preprocess the data
57 try:
58     X_processed = preprocessor.fit_transform(features_df)
59     print(f"Data preprocessing completed. Feature matrix size:
60           {X_processed.shape}")
61     try:
62         feature_names_out = preprocessor.get_feature_names_out()
63     except AttributeError:
64         feature_names_out = numeric_features + \
65                             list(preprocessor.transformers_[1][1].named_steps['onehot']
66                                 \
67                                 .get_feature_names_out(categorical_features))
68 except Exception as e:
69     print(f"Error during data preprocessing: {e}")
70     print("Selected numeric columns:", numeric_features)
71     print("Selected categorical columns:", categorical_features)
72     print("Data types of numeric columns:")
73     print(df[numeric_features].dtypes)
74     print("Data types of categorical columns:")
75     print(df[categorical_features].dtypes)
76     print("Number of NA values in numeric columns:")
77     print(df[numeric_features].isna().sum())
78     print("Number of NA values in categorical columns:")
79     print(df[categorical_features].isna().sum())
80     exit()
81
82 # Calculate inertia for different k values (Elbow Method)
83 inertia = []
84 possible_k = range(2, 11)
85
86 print("\nCalculating Inertia for different k values (Elbow Method)...")
87 for k in possible_k:
88     kmeans = KMeans(n_clusters=k, init='k-means++', random_state=42, n_init=10)
89     kmeans.fit(X_processed)
90     inertia.append(kmeans.inertia_)
91
92 # Plot the Elbow curve
93 plt.figure(figsize=(10, 6))
94 plt.plot(possible_k, inertia, marker='o')
95 plt.title('Elbow Method for Determining Optimal Number of Clusters (k)')
96 plt.xlabel('Number of Clusters (k)')

```

```

95 plt.ylabel('Inertia (Within-cluster Sum of Squares)')
96 plt.xticks(possible_k)
97 plt.grid(True)
98 plt.show()
99
100 # Select optimal number of clusters
101 optimal_k = 4
102 print(f"\n=> Based on the Elbow plot, selected k = {optimal_k}")
103
104 # Perform final clustering
105 kmeans_final = KMeans(n_clusters=optimal_k, init='k-means++', random_state=42,
106                       n_init=10)
106 clusters = kmeans_final.fit_predict(X_processed)
107
108 # Add cluster labels to dataframes
109 player_info['Cluster'] = clusters
110 df['Cluster'] = clusters
111
112 print(f"\nAssigned {len(df)} players to {optimal_k} clusters.")
113 print("Number of players in each cluster:")
114 print(player_info['Cluster'].value_counts().sort_index())
115
116 # Perform PCA for dimensionality reduction
117 print("\nPerforming PCA to reduce data to 2 dimensions...")
118 pca = PCA(n_components=2, random_state=42)
119 X_pca = pca.fit_transform(X_processed)
120
121 # Create PCA dataframe
122 pca_df = pd.DataFrame(data=X_pca, columns=['Principal Component 1', 'Principal
123                                     Component 2'])
123 pca_df['Cluster'] = clusters
124 pca_df['Player'] = player_info['Player'].values
125 pca_df['Position'] = player_info['Position'].values
126
127 # Plot 2D cluster visualization
128 print("Plotting 2D cluster visualization...")
129 plt.figure(figsize=(12, 8))
130 sns.scatterplot(
131     x="Principal Component 1", y="Principal Component 2",
132     hue="Cluster",
133     palette=sns.color_palette("hsv", optimal_k),
134     data=pca_df,
135     legend="full",
136     alpha=0.8
137 )
138
139 plt.title(f'Player Clustering ({optimal_k} Clusters) After PCA Reduction')
140 plt.xlabel('Principal Component 1')
141 plt.ylabel('Principal Component 2')
142 plt.grid(True)
143 plt.show()

```

```

144
145 # Analyze cluster characteristics
146 print(f"\nAnalyzing basic characteristics of {optimal_k} clusters:")
147 cluster_summary = player_info.groupby('Cluster').agg(
148     count=('Player', 'size'),
149     common_position=('Position', lambda x: x.mode()[0] if not x.mode().empty
150                     else 'N/A'),
150     avg_age=('Age', lambda x: pd.to_numeric(x, errors='coerce').mean())
151 ).reset_index()
152
153 print("\nOverview of cluster characteristics (Count, Most Common Position,
154       Average Age):")
155 print(cluster_summary)
156
157 # Calculate mean statistics for each cluster
158 print("\nMean values of original statistics for each cluster:")
159 numeric_original_df = df[numeric_features + ['Cluster']].copy()
160 for col in numeric_features:
161     numeric_original_df[col] = pd.to_numeric(numeric_original_df[col],
162                                             errors='coerce')
163
164 cluster_means = numeric_original_df.groupby('Cluster').mean()
165 print(cluster_means.round(2))
166
167 # PCA information
168 print("\nPCA Information:")
169 explained_variance = pca.explained_variance_ratio_
170 print(f"Variance explained by PC1: {explained_variance[0]:.2%}")
171 print(f"Variance explained by PC2: {explained_variance[1]:.2%}")
172 print(f"Total variance explained by 2 PCs: {explained_variance.sum():.2%}")
173
174 print("\n--- End ---")

```

Giải thích chi tiết từng bước của code

Bước 1: Tải và kiểm tra dữ liệu

Code bắt đầu bằng việc tải file CSV `result.csv` sử dụng hàm `pd.read_csv` từ thư viện `pandas`. Nếu file không tồn tại hoặc có lỗi, chương trình sẽ in thông báo lỗi và dừng thực thi.

Bước 2: Chuẩn bị dữ liệu

- Tách thông tin cầu thủ: Các cột `Player`, `Team`, `Position`, `Age` được sao chép vào DataFrame `player_info`.
- Xác định đặc trưng:
 - Đặc trưng số: Các cột chứa giá trị số, ngoại trừ `Age`.
 - Đặc trưng phân loại: Cột `Position`.

-
- Tạo DataFrame `features_df` chứa các đặc trưng đã chọn.

Bước 3: Tiền xử lý dữ liệu

Sử dụng Pipeline và ColumnTransformer để tiền xử lý:

- Đối với đặc trưng số:
 - Điền giá trị thiếu bằng giá trị trung bình (`SimpleImputer`).
 - Chuẩn hóa dữ liệu bằng `StandardScaler`.
- Đối với đặc trưng phân loại:
 - Điền giá trị thiếu bằng giá trị phổ biến nhất (`SimpleImputer`).
 - Mã hóa one-hot bằng `OneHotEncoder`.

Kết quả là ma trận `X_processed` đã được tiền xử lý.

Bước 4: Xác định số cụm tối ưu

Sử dụng phương pháp Elbow:

- Tính inertia cho k từ 2 đến 10.
- Vẽ biểu đồ inertia theo k để tìm điểm "elbow".
- Chọn $k = 4$ làm số cụm tối ưu.

Bước 5: Phân cụm

- Áp dụng thuật toán KMeans với $k = 4$ trên `X_processed`.
- Gán nhãn cụm (0, 1, 2, 3) cho từng cầu thủ, lưu vào `player_info` và `df`.

Bước 6: Giảm chiều dữ liệu

Sử dụng PCA để giảm chiều dữ liệu xuống 2 chiều:

- Tạo ma trận `X_pca` từ `X_processed`.
- Tạo DataFrame `pca_df` chứa hai thành phần chính và nhãn cụm.

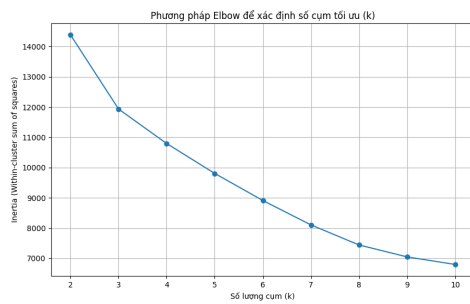
Bước 7: Trực quan hóa

Sử dụng `sns.scatterplot` từ seaborn để vẽ biểu đồ phân tán 2D:

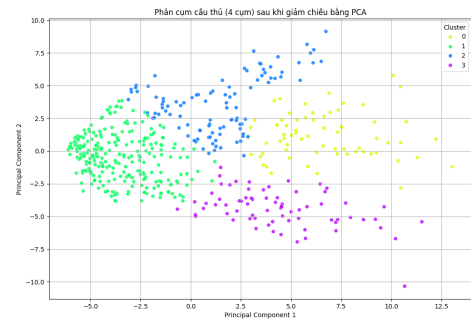
- Trục x: Thành phần chính 1 (PC1).
- Trục y: Thành phần chính 2 (PC2).
- Màu sắc: Phân biệt các cụm bằng nhãn cụm.

Bước 8: Phân tích cụm

- Tính toán:
 - Số lượng cầu thủ trong mỗi cụm.
 - Vị trí thi đấu phổ biến nhất và tuổi trung bình của mỗi cụm.
 - Giá trị trung bình của các chỉ số thống kê gốc cho mỗi cụm.
- Hiển thị tỷ lệ phương sai được giải thích bởi PCA.



(a) Biểu đồ Elbow



(b) Phân cụm PCA

Hình 4: Elbow and PCA

Bài 4:

Mã nguồn lấy data cho chuyển nhượng cầu thủ

```
1 # --- Import necessary libraries ---
2 import time
3 import pandas as pd
4 from bs4 import BeautifulSoup
5 from selenium import webdriver
6 from selenium.webdriver.chrome.service import Service as ChromeService
7 from webdriver_manager.chrome import ChromeDriverManager
8 from selenium.common.exceptions import WebDriverException
9
10 # --- Function to set up Selenium WebDriver ---
11 def setup_driver():
12     """Initialize and return an instance of Chrome WebDriver."""
13     try:
14         service = ChromeService(ChromeDriverManager().install())
15         driver = webdriver.Chrome(service=service)
16         print("WebDriver initialized successfully.")
17         return driver
18     except WebDriverException as e:
19         print(f"Error initializing WebDriver: {e}")
20         print("Ensure Google Chrome is installed.")
21         print("Or try updating webdriver-manager: pip install --upgrade\n        webdriver-manager")
22         return None
23     except Exception as e:
24         print(f"Unknown error initializing driver: {e}")
25         return None
26
27 # --- Function to scrape data from a specific URL ---
28 def scrape_page(driver, url):
29     """Scrape player data from a URL using Selenium driver."""
30     if driver is None:
31         print("Error: Invalid driver.")
32         return []
33     try:
34         driver.get(url)
35         print(f"Accessing: {url}")
36         time.sleep(3)
37         soup = BeautifulSoup(driver.page_source, 'html.parser')
38
39         table = soup.find('table', class_='table table-hover no-cursor\n        table-striped leaguetable mvp-table similar-players-table mb-0')
40         if not table:
41             print(f"Warning: No data table found on page {url}")
42             return []
43
44         tbody = table.find('tbody')
45         if not tbody:
```

```

46     print(f"Warning: No tbody tag found in table on page {url}")
47     return []
48
49     data = []
50     rows = tbody.find_all('tr')
51     print(f"Found {len(rows)} rows on page {url}")
52
53     for row in rows:
54         try:
55             skill_div = row.find('div', class_='table-skill__skill')
56             pot_div = row.find('div', class_='table-skill__pot')
57             skill_text = skill_div.text.strip() if skill_div else None
58             pot_text = pot_div.text.strip() if pot_div else None
59             skill = float(skill_text) if skill_text else None
60             pot = float(pot_text) if pot_text else None
61             skill_pot = f"{skill}/{pot}" if skill is not None and pot is
                not None else None
62
63             player_link = row.select_one('td.td-player div.text a')
64             player_name = player_link.text.strip() if player_link else None
65
66             team_span = row.find('span', class_='td-team__teamname')
67             team = team_span.text.strip() if team_span else None
68
69             etv_span = row.find('span', class_='player-tag')
70             etv = etv_span.text.strip() if etv_span else None
71
72             if player_name and team and etv and skill_pot:
73                 data.append({
74                     'player_name': player_name,
75                     'team': team,
76                     'price': etv,
77                     'skill/pot': skill_pot
78                 })
79
80         except Exception as e:
81             print(f"Error processing a row: {e}. Skipping this row.")
82             continue
83
84     return data
85
86 except WebDriverException as e:
87     print(f"WebDriver error accessing {url}: {e}")
88     return []
89 except Exception as e:
90     print(f"Unknown error scraping page {url}: {e}")
91     return []
92
93 # --- Main section to perform scraping ---
94 base_url = "https://www.footballtransfers.com/en/players/uk-premier-league"
95 total_pages = 22

```

```

96 all_data = []
97
98 print("Initializing WebDriver...")
99 driver = setup_driver()
100
101 if driver:
102     try:
103         print(f"Starting to scrape data from {total_pages} pages...")
104         for page in range(1, total_pages + 1):
105             print(f"\n--- Processing page {page}/{total_pages} ---")
106             if page == 1:
107                 url = base_url
108             else:
109                 url = f"{base_url}/{page}"
110
111             page_data = scrape_page(driver, url)
112
113             if page_data:
114                 all_data.extend(page_data)
115                 print(f"Added {len(page_data)} records from page {page}.")
116             else:
117                 print(f"No valid data returned from page {page}.")
118
119         except Exception as e:
120             print(f"An error occurred during scraping: {e}")
121         finally:
122             print("\nClosing WebDriver...")
123             driver.quit()
124
125         if all_data:
126             print(f"\nTotal of {len(all_data)} records scraped.")
127             df_final = pd.DataFrame(all_data)
128             try:
129                 df_final.to_csv('football_transfers_players.csv', index=False,
130                                encoding='utf-8-sig')
131                 print("Data successfully saved to 'football_transfers_players.csv'")
132                 print("\nPreview of the first 5 rows of data:")
133                 print(df_final.head())
134             except Exception as e:
135                 print(f"Error saving CSV file: {e}")
136             else:
137                 print("\nNo data collected. CSV file will not be created.")
138         else:
139             print("Failed to initialize WebDriver. Unable to proceed with scraping.")
140     print("\nCompleted.")

```

Mã nguồn chính

```
1 import pandas as pd
2
3 def combine_and_filter_player_data():
4     """
5     Combine data from football_transfers_players.csv and results.csv,
6     then filter players with playing time > 900 minutes and display that time.
7     """
8     try:
9         df_transfers = pd.read_csv('football_transfers_players.csv')
10        df_fbref = pd.read_csv('results.csv')
11        print("Successfully read 'football_transfers_players.csv' and
12              'results.csv'.")
13    except FileNotFoundError as e:
14        print(f"Error: One of the required CSV files not found: {e}")
15        print("Ensure 'football_transfers_players.csv' and 'results.csv' are
16              created and in the same directory.")
17        return
18    except pd.errors.EmptyDataError as e:
19        print(f"Error: One of the CSV files is empty: {e}")
20        return
21    except Exception as e:
22        print(f"Unknown error reading CSV files: {e}")
23        return
24
25    if 'player_name' in df_transfers.columns:
26        df_transfers.rename(columns={'player_name': 'Player'}, inplace=True)
27    elif 'Player' not in df_transfers.columns:
28        print("Error: No player name column ('player_name' or 'Player') found
29              in 'football_transfers_players.csv'.")
30        return
31
32    if 'Player' not in df_fbref.columns:
33        print("Error: No 'Player' column found in 'results.csv'.")
34        return
35
36    minutes_col_fbref = None
37    candidate_minute_cols = ['Playing_Time_Min', 'Min', 'minutes']
38
39    for col_name in candidate_minute_cols:
40        if col_name in df_fbref.columns:
41            minutes_col_fbref = col_name
42            break
43
44    if minutes_col_fbref is None:
45        possible_min_cols = [col for col in df_fbref.columns if 'min' in
46                             col.lower() and ('time' in col.lower() or 'play' in col.lower() or
47                             col.lower() == 'min')]
48        if possible_min_cols:
49            minutes_col_fbref = possible_min_cols[0]
```

```

45         print(f"Warning: No standard minutes column found. Using heuristic
           column: '{minutes_col_fbref}'")
46     else:
47         print(f"Error: Could not identify minutes played column in
           'results.csv'. Available columns: {df_fbref.columns.tolist()}")
48     return
49
50     print(f"Using column '{minutes_col_fbref}' from 'results.csv' for
           filtering minutes played.")
51
52     df_fbref[minutes_col_fbref] = pd.to_numeric(df_fbref[minutes_col_fbref],
           errors='coerce')
53     df_fbref.dropna(subset=[minutes_col_fbref], inplace=True)
54
55     df_fbref_filtered = df_fbref[df_fbref[minutes_col_fbref] > 900][['Player',
           minutes_col_fbref]].copy()
56
57     if df_fbref_filtered.empty:
58         print(f"\nNo players in 'results.csv' with playing time
           ({minutes_col_fbref}) > 900 minutes.")
59         return
60
61     df_fbref_filtered.rename(columns={minutes_col_fbref:
           'Total_Minutes_Played'}, inplace=True)
62
63     players_with_high_minutes_count = df_fbref_filtered['Player'].nunique()
64     print(f"\nFound {players_with_high_minutes_count} players in 'results.csv'
           with > 900 minutes played.")
65
66     players_to_keep = df_fbref_filtered['Player'].unique()
67     df_transfers_filtered_by_name =
           df_transfers[df_transfers['Player'].isin(players_to_keep)].copy()
68
69     df_fbref_for_merge = df_fbref_filtered.drop_duplicates(subset=['Player'],
           keep='first')
70     df_final_output = pd.merge(df_transfers_filtered_by_name,
           df_fbref_for_merge, on='Player', how='left')
71
72     print(f"\nInitial number of players in 'football_transfers_players.csv':
           {len(df_transfers)}")
73     print(f"Final number of players (matching > 900 minutes criteria and in
           transfers): {len(df_final_output)}")
74
75     if not df_final_output.empty:
76         print("\n--- Preview of first 5 rows of filtered player data (including
           total minutes played): ---")
77         print(df_final_output.head())
78
79     try:
80         output_filename =
           'filtered_football_transfers_players_gt900min_with_total_time.csv'

```

```

81     cols = ['Player'] + [col for col in df_final_output.columns if col
82                          != 'Player' and col != 'Total_Minutes_Played'] +
83     ['Total_Minutes_Played']
84     cols_exist = [col for col in cols if col in df_final_output.columns]
85     df_final_output_ordered = df_final_output[cols_exist]
86     df_final_output_ordered.to_csv(output_filename, index=False,
87                                   encoding='utf-8-sig')
88     print(f"\nSaved filtered player data to '{output_filename}'")
89 except Exception as e:
90     print(f"Error saving output CSV file: {e}")
91 else:
92     print("\nNo players from 'football_transfers_players.csv' match the >
93         900 minutes criteria or could not be merged.")
94
95 if __name__ == '__main__':
96     combine_and_filter_player_data()

```

Phân tích code trong BTL-BAI4.py

File này chịu trách nhiệm thu thập dữ liệu cầu thủ từ trang web <https://www.footballtransfers.com/en/players/uk-premier-league> và lưu vào file CSV có tên `football_transfers_players.csv`

Import các thư viện cần thiết

```

import time
import pandas as pd
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.chrome.service import Service as ChromeService
from webdriver_manager.chrome import ChromeDriverManager
from selenium.common.exceptions import WebDriverException

```

Danh sách các thư viện:

- **time**: Dùng để tạm dừng chương trình nhằm tránh tải quá nhanh.
- **pandas**: Thư viện xử lý và phân tích dữ liệu.
- **BeautifulSoup**: Phân tích cú pháp HTML từ trang web.
- **selenium**: Tự động hóa trình duyệt để tải trang động.
- **ChromeDriverManager**: Quản lý và cài đặt ChromeDriver tự động.
- **WebDriverException**: Xử lý các ngoại lệ từ Selenium.

Hàm setup_driver()

```
def setup_driver():
    try:
        service = ChromeService(ChromeDriverManager().install())
        driver = webdriver.Chrome(service=service)
        print("WebDriver initialized successfully.")
        return driver
    except WebDriverException as e:
        print(f"Error initializing WebDriver: {e}")
        print("Ensure Google Chrome is installed.")
        print("Or try updating webdriver-manager: pip install --upgrade webdriver-m")
        return None
    except Exception as e:
        print(f"Unknown error initializing driver: {e}")
        return None
```

Chức năng:

- Khởi tạo trình duyệt Chrome thông qua Selenium WebDriver.
- Sử dụng `ChromeDriverManager().install()` để cài đặt ChromeDriver tự động.
- Trả về đối tượng driver nếu thành công, hoặc `None` nếu có lỗi.

Hàm scrape_page(driver, url)

```
def scrape_page(driver, url):
    if driver is None:
        print("Error: Invalid driver.")
        return []
    try:
        driver.get(url)
        print(f"Accessing: {url}")
        time.sleep(3)
        soup = BeautifulSoup(driver.page_source, 'html.parser')
```

Chức năng:

- Thu thập dữ liệu từ một URL cụ thể.
- Kiểm tra driver hợp lệ, tải trang web, chờ 3 giây, sau đó phân tích HTML bằng BeautifulSoup.

```
        table = soup.find('table', class_='table table-hover no-cursor table-striped')
        if not table:
            print(f"Warning: No data table found on page {url}")
            return []

        tbody = table.find('tbody')
        if not tbody:
            print(f"Warning: No tbody tag found in table on page {url}")
            return []
```


- Tìm bảng dữ liệu với class cụ thể trên trang web.
- Nếu không tìm thấy bảng hoặc thẻ <tbody>, trả về danh sách rỗng.

```
data = []
rows = tbody.find_all('tr')
print(f"Found {len(rows)} rows on page {url}")

for row in rows:
    try:
        skill_div = row.find('div', class_='table-skill__skill')
        pot_div = row.find('div', class_='table-skill__pot')
        skill_text = skill_div.text.strip() if skill_div else None
        pot_text = pot_div.text.strip() if pot_div else None
        skill = float(skill_text) if skill_text else None
        pot = float(pot_text) if pot_text else None
        skill_pot = f"{skill}/{pot}" if skill and pot else None
```

- Xử lý từng hàng trong bảng dữ liệu.
- Trích xuất và chuyển đổi thông tin kỹ năng (skill) và tiềm năng (pot).

```
player_link = row.select_one('td.td-player div.text a')
player_name = player_link.text.strip() if player_link else None

team_span = row.find('span', class_='td-team__teamname')
team = team_span.text.strip() if team_span else None

etv_span = row.find('span', class_='player-tag')
etv = etv_span.text.strip() if etv_span else None
```

- Trích xuất tên cầu thủ, tên đội bóng, và giá trị chuyển nhượng (ETV).

```
if player_name and team and etv and skill_pot:
    data.append({
        'player_name': player_name,
        'team': team,
        'price': etv,
        'skill/pot': skill_pot
    })
```

- Nếu tất cả thông tin đầy đủ, thêm dữ liệu vào danh sách data.

```
except WebDriverException as e:
    print(f"WebDriver error accessing {url}: {e}")
    return []
except Exception as e:
    print(f"Unknown error scraping page {url}: {e}")
    return []
```

- Xử lý lỗi từ WebDriver hoặc lỗi khác, trả về danh sách rỗng nếu có vấn đề.

Phần chính (Main Section)

```
base_url = "https://www.footballtransfers.com/en/players/uk-premier-league"
total_pages = 22
all_data = []
```

```
driver = setup_driver()
```

```
if driver:
    try:
        for page in range(1, total_pages + 1):
            if page == 1:
                url = base_url
            else:
                url = f"{base_url}/{page}"
            page_data = scrape_page(driver, url)
            if page_data:
                all_data.extend(page_data)
```

- Thiết lập URL cơ bản và tổng số trang (22).
- Khởi tạo danh sách `all_data` để lưu trữ toàn bộ dữ liệu.
- Dùng vòng lặp để thu thập dữ liệu từ 22 trang.

```
finally:
    driver.quit()
```

```
if all_data:
    df_final = pd.DataFrame(all_data)
    df_final.to_csv('football_transfers_players.csv', index=False, encoding='utf-8')
    print("\nPreview of the first 5 rows of data:")
    print(df_final.head())
```

- Đóng trình duyệt sau khi hoàn tất.
- Nếu có dữ liệu, chuyển thành DataFrame và lưu vào file CSV, sau đó in 5 hàng đầu tiên để kiểm tra.

Phân tích code trong "FINAL RESULT.py"

File này xử lý dữ liệu từ `football_transfers_players.csv`, kết hợp với `results.csv`, lọc các cầu thủ chơi hơn 900 phút và lưu kết quả.

Import thư viện

```
import pandas as pd
```

- Sử dụng **pandas** để đọc và xử lý dữ liệu từ các file CSV.

Hàm `combine_and_filter_player_data()`

```
def combine_and_filter_player_data():
    try:
        df_transfers = pd.read_csv('football_transfers_players.csv')
        df_fbref = pd.read_csv('results.csv')
        print("Successfully read 'football_transfers_players.csv' and 'results.csv'")
    except FileNotFoundError as e:
        print(f"Error: One of the required CSV files not found: {e}")
        return
```

- Đọc dữ liệu từ hai file CSV, xử lý lỗi nếu file không tồn tại.

```
    if 'player_name' in df_transfers.columns:
        df_transfers.rename(columns={'player_name': 'Player'}, inplace=True)
    elif 'Player' not in df_transfers.columns:
        print("Error: No player name column ('player_name' or 'Player') found in 'f")
        return

    if 'Player' not in df_fbref.columns:
        print("Error: No 'Player' column found in 'results.csv'.")
        return
```

- Chuẩn hóa tên cột thành `Player` để đồng nhất giữa hai file.

```
    minutes_col_fbref = None
    candidate_minute_cols = ['Playing_Time_Min', 'Min', 'minutes']
    for col_name in candidate_minute_cols:
        if col_name in df_fbref.columns:
            minutes_col_fbref = col_name
            break

    if minutes_col_fbref is None:
        possible_min_cols = [col for col in df_fbref.columns if 'min' in col.lower()]
        if possible_min_cols:
            minutes_col_fbref = possible_min_cols[0]
```

- Tìm cột chứa thông tin phút thi đấu trong `results.csv`.

```
    df_fbref[minutes_col_fbref] = pd.to_numeric(df_fbref[minutes_col_fbref], errors='coerce')
    df_fbref.dropna(subset=[minutes_col_fbref], inplace=True)

    df_fbref_filtered = df_fbref[df_fbref[minutes_col_fbref] > 900][['Player', minutes_col_fbref]]
```

- Chuyển đổi dữ liệu phút thi đấu sang số, loại bỏ giá trị NaN, và lọc các cầu thủ chơi hơn 900 phút.

```
    df_fbref_filtered.rename(columns={minutes_col_fbref: 'Total_Minutes_Played'}, inplace=True)
    players_to_keep = df_fbref_filtered['Player'].unique()
    df_transfers_filtered_by_name = df_transfers[df_transfers['Player'].isin(players_to_keep)]
```

-
- Đổi tên cột phút thi đấu thành `Total_Minutes_Played`.
 - Lọc dữ liệu chuyển nhượng dựa trên danh sách cầu thủ đã chọn.

```
df_fbref_for_merge = df_fbref_filtered.drop_duplicates(subset=['Player'], keep='first')
df_final_output = pd.merge(df_transfers_filtered_by_name, df_fbref_for_merge, on='Player')
```

- Xóa các hàng trùng lặp trong dữ liệu phút thi đấu.
- Gộp dữ liệu chuyển nhượng và phút thi đấu dựa trên cột `Player`.

```
if not df_final_output.empty:
    cols = ['Player'] + [col for col in df_final_output.columns if col != 'Player']
    df_final_output_ordered = df_final_output[cols]
    df_final_output_ordered.to_csv('filtered_football_transfers_players_gt900mi.csv')
```

- Nếu có dữ liệu, sắp xếp lại thứ tự cột và lưu vào file CSV mới.

Phần chính

```
if __name__ == '__main__':
    combine_and_filter_player_data()
```

- Chạy hàm chính khi file được thực thi.