Importa las bibliotecas: Para manejar los datos y hacer solicitudes a la API.

```
# LibrerÃas para interaccionar con la API: requests
import re
import time


import numpy as np
import pandas as pd
import requests
from requests.auth import HTTPBasicAuth
```

Función main().

Se especifica una lista de identificadores de ligas de fútbol.
Se llama a la función get_all_stats_players_season() para obtener datos de jugadores en esas ligas.
Los datos recopilados se almacenan en un DataFrame y se exportan a un archivo de Excel.

```python
def main():
    ligas = ['688', '295', '339', '296', '869', '617', '143', '284', '879', '685']
    # ligas = ['295', '339'] # 339 errores
    start = time.time()
    data_all_players = get_all_stats_players_season(ligas)
    print(f"El total de jugadores es {data_all_players.shape[0]}")
    end = time.time()
    print(f'Demora el proceso {(end - start) / 60} Min')
    data_all_players.to_excel('/home/chidalgo/git/atl_nacional_analytics/data/all_playersAgo.xlsx', index=False)
```

Función “get_total_players”
Obtiene los datos de jugadores de una liga específica.

Los parámetros son:
total: Un número entero que representa la cantidad total de jugadores en la liga y
wyid: el identificador de la liga.
Todo esto va a devolver un dataframe.

Funcionamiento:
La función verifica si el valor de total no es None (es decir, si se conoce la cantidad total de jugadores en la liga).
Calcula la cantidad total de páginas necesarias para obtener todos los datos de los jugadores. Esto se hace dividiendo el total por 100 y redondeando hacia arriba utilizando la función np.ceil() de NumPy.
Inicializa un DataFrame vacío llamado df_tot para almacenar los datos recopilados. Luego, utiliza un bucle for para iterar a través de todas las páginas de datos. En cada iteración, realiza una solicitud a la API de Wyscout para obtener datos de jugadores. La URL de la solicitud incluye el identificador de la liga (wyid), el límite de jugadores por página (100) y el número de página actual.
Los datos obtenidos de cada página se almacenan temporalmente en un DataFrame llamado tmp_df. Luego, se concatenan los datos de tmp_df con los datos previamente recopilados en df_tot. Por ùltimo, se restablece el índice del DataFrame resultante y se elimina la columna de índice anterior.

```python
def get_total_players(total: int, wyid: str) -> DF:
    """
    :param total: Integer with the total players of a league
    :param wyid: wyid of league
    :return: Dataframe
    """
    if total is not None:
        #total_pages = 1
        total_pages = int(np.ceil(total / 100))
        df_tot = pd.DataFrame()
        for page in range(1, total_pages + 1):
            r = (requests.
                 get(f'https://apirest.wyscout.com/v3/competitions/{wyid}/players?limit=100&page={page}',
                     auth=AUTH))
            tmp_df = pd.DataFrame(r.json()['players'])
            df_tot = pd.concat([df_tot, tmp_df])
        return df_tot.reset_index().drop('index', axis=1)
    else:
        pass
```

Función get_teams_info

Obtiene la información sobre los equipos de una liga.
Paràmetros: “wyId” (identificador de la liga).
Devuelve un dataframe.

La función realiza una solicitud a la API de Wyscout.
(La URL de la solicitud se forma utilizando el identificador de la liga (wyd) proporcionado como parámetro).
Y la respuesta se almacena en “r”. Se extraen los datos de los equipos en formato JSON utilizando r.json()['teams'] y los guarda en un DataFrame llamado “equipos”.
Se seleccionan algunas columnas específicas de información sobre los equipos, como su identificador (wyId), nombre, nombre oficial, ciudad y área.
Luego se agrega una columna llamada 'country' al DataFrame. Esta columna se obtiene al normalizar los datos del área y seleccionar el nombre del país.
Y esto devolverá un dataframe con lo solicitado.

```python
def get_teams_info(wyd):
    r = requests.get(f'https://apirest.wyscout.com/v3/competitions/{wyd}/teams', auth=AUTH)
    equipos = pd.DataFrame(r.json()['teams'])
    equipos = equipos[['wyId', 'name', 'officialName', 'city', 'area']]
    equipos['country'] = pd.json_normalize(equipos['area'])['name'].values[0]
    return equipos
```

Función get_advanced_stats_player

Obtiene estadísticas avanzadas de un jugador en una liga especifica.

Tiene varios parámetros:
id_player: El identificador del jugador.
id_liga: El identificador de la liga.
season_id (opcional): El identificador de la temporada de la que se desean obtener las estadísticas.
Si no se proporciona, se obtienen las estadísticas de la temporada actual.
index_ (opcional): Un índice para identificar el jugador en el DataFrame resultante.
force_season_act (opcional): Un indicador booleano que, si es True, fuerza la obtención de las estadísticas de la temporada actual, incluso si se proporciona season_id.
Todo esto devuelve un dataframe.

¿Cómo funciona?
Empieza por una solicitud a la API.
Luego, se determina si se deben obtener estadísticas de la temporada actual o de una temporada en específica.
Esto se hace verificando si force_season_act es True.
Si es True, se obtienen las estadísticas de la temporada actual.
Si season_id se proporciona, se obtienen las estadísticas de esa temporada.
Las estadísticas se van a almacenar en un dataframe “ttl”.
También se obtiene información sobre la posición principal y secundaria del jugador, y se almacena en un DataFrame llamado “position”.
Por ultimo, Las estadísticas, la posición y otros datos relevantes se combinan en un único DataFrame que se retorna como resultado de la función.

Función get_advanced_stats_player

```python
def get_advanced_stats_player(id_player, id_liga, season_id=None, index_=None, force_season_act=False):
    if index_ is None:
        index_ = [0]
    r_nac = requests.get(f'https://apirest.wyscout.com/v3/players/{id_player}', auth=AUTH)
    if force_season_act:
        active_season = requests.get(f'https://apirest.wyscout.com/v3/competitions/{id_liga}/seasons', auth=AUTH)
        asa = pd.json_normalize(pd.DataFrame(active_season.json()['seasons'])['season'])
        asac = int(asa['wyId'][asa['active']])
        r = requests.get(
            f'https://apirest.wyscout.com/v3/players/{id_player}/'
            f'advancedstats?compId={id_liga}&seasonId={asac}',
            auth=AUTH)
    else:
        if season_id:
            r = requests.get(
                f'https://apirest.wyscout.com/v3/players/{id_player}/'
                f'advancedstats?compId={id_liga}&seasonId={season_id}',
                auth=AUTH)
        else:
            r = requests.get(f'https://apirest.wyscout.com/v3/players'
                             f'/{id_player}/advancedstats?compId={id_liga}',
                             auth=AUTH)
```

```python
    if r.status_code == 200:
        try:
            position = pd.DataFrame({'id_player': id_player,
                                     'main_position': r.json()['positions'][0]['position']['name'],
                                     'code_main_positon': r.json()['positions'][0]['position']['code'],
                                     'percent_main_position': r.json()['positions'][0]['percent'],
                                     'nationality': r_nac.json()['birthArea']['name'],
                                     'second_position': r.json()['positions'][1]['position']['name'],
                                     'code_second_position': r.json()['positions'][1]['position']['code'],
                                     'percent_second_position': r.json()['positions'][1]['percent']},
                                    index=[index_])
        except (IndexError, KeyError):
            try:
                position = pd.DataFrame({'id_player': id_player,
                                         'main_position': r.json()['positions'][0]['position']['name'],
                                         'code_main_positon': r.json()['positions'][0]['position']['code'],
                                         'percent_main_position': r.json()['positions'][0]['percent'],
                                         'nationality': r_nac.json()['birthArea']['name'],
                                         'second_position': np.nan,
                                         'code_second_position': np.nan,
                                         'percent_second_position': np.nan}
                                        , index=[index_])
            except (IndexError, KeyError):
                position = pd.DataFrame({'id_player': id_player,
                                         'main_position': np.nan,
                                         'code_main_positon': np.nan,
                                         'percent_main_position': np.nan,
                                         'nationality': np.nan,
                                         'second_position': np.nan,
                                         'code_second_position': np.nan,
                                         'percent_second_position': np.nan}
                                        , index=[index_])
```

```python
    # Total vars
    total_vars = pd.DataFrame(r.json()['total'], index=[index_])
    total_vars.columns = ['total_' + x for x in total_vars.columns]
    total_vars['id_player'] = id_player
    # Average vars
    average_vars = pd.DataFrame(r.json()['average'], index=[index_])
    average_vars.columns = ['average_' + x for x in average_vars.columns]
    average_vars['id_player'] = id_player
    # Percent
    percent_vars = pd.DataFrame(r.json()['percent'], index=[index_])
    percent_vars.columns = ['percent_' + x for x in percent_vars.columns]
    percent_vars['id_player'] = id_player
    # total bd player
    ttl = (position.copy().merge(total_vars, how='left', on='id_player').
           merge(average_vars, how='left', on='id_player').
           merge(percent_vars, how='left', on='id_player'))
    return ttl
```

Función get_all_stats_players_season

Esta función va a ser la parte central del script, recopila información muy detallada sobre jugadores en varias ligas.
Como parámetros tiene “id_ligas” (identificador de la liga).
Funciona así:
La función toma la lista de identificadores de ligas id_ligas como entrada y realiza una serie de operaciones para obtener datos de los jugadores en esas ligas.
Dentro de la función nos vamos a encontrar con que:
Se inicializa una lista vacía llamada total_lista_ligas para almacenar los datos recopilados de todas las ligas. Luego, la función itera a través de cada identificador de liga en la lista id_ligas. Para cada liga, se realiza una solicitud a la API
para obtener datos de todos los jugadores en esa liga.
Esto se hace utilizando la función get_total_players y se almacena en total_jugadores.
Si se obtienen datos de jugadores, se seleccionan las columnas específicas de interés
(definidas en vars_players) y se almacenan en total_jugadores.
Se obtendra una lista de identificadores de jugadores (lista_jugadores) a partir de los datos de total_jugadores. Luego, se itera a través de cada identificador de jugador en lista_jugadores y se llama a la función get_advanced_stats_player para obtener estadísticas avanzadas de cada jugador en la liga. Las estadísticas se guardan en ply_tmp.
Si las estadísticas no son “None” se agrega la columna “compId” con el identificador de la liga y se concatena en “total_players_ligas”.
Con la función “get_teams_info” se obtendrá información sobre los equipos de la liga y luego se fusiona con los datos de los jugadores para incluir detalles del equipo actual. Por ultimo, los datos de jugadores de la liga se concatenan con los datos recopilados anteriormente en total_jugadores y se almacenan en total_lista_ligas.
Se verifica si se encontraron datos para concatenar. Si es así, se concatenan todos los datos de jugadores de diferentes ligas en un solo DataFrame “total_players_tot”, que se retorna como resultado de la función. Si no se encuentran datos para concatenar, se muestra un mensaje de advertencia.

Función “get_all_stats_players_season”

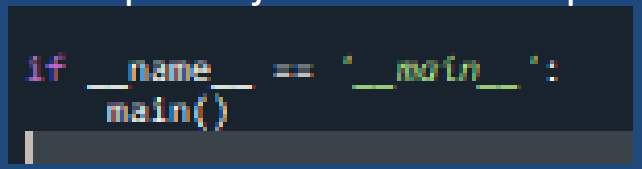
```python
def get_all_stats_players_season(id_ligas: list):
    vars_players = ['wyId', 'shortName', 'firstName', 'middleName', 'lastName', 'height', 'weight'
        , 'birthDate', 'foot', 'currentTeamId', 'status', 'imageDataURL']
    total_lista_ligas = list()
    for liga in id_ligas:
        print(f"Liga {liga}")
        total_players_ligas = pd.DataFrame()
        r1 = requests.get(f'https://apirest.wyscout.com/v3/competitions/{liga}/players', auth=AUTH)
        if r1.status_code==200:
            total_players = r1.json()['meta']['total_items']
            total_jugadores = get_total_players(total_players, f'{liga}')
            if total_players != 0:
                total_jugadores = total_jugadores[vars_players]
            else:
                print(f"La liga {liga} no tiene jugadores en la temporada actual")
                continue
            lista_jugadores = total_jugadores.wyId
            for k, jugador in enumerate(lista_jugadores):

                ply_tmp = get_advanced_stats_player(id_player=jugador, id_liga=liga, index_=[k])
                if ply_tmp is not None:
                    try:
                        ply_tmp['compId'] = liga
                        total_players_ligas = pd.concat([total_players_ligas, ply_tmp])
                    except TypeError:
                        continue
            try:
                equipos = get_teams_info(liga).rename(columns={'wyId': 'currentTeamId'})
                total_jugadores = total_jugadores.merge(equipos, how='left', left_on='currentTeamId',
                                                        right_on='currentTeamId')
                total_jugadores = total_jugadores.merge(total_players_ligas, how='left', left_on='wyId', right_on='id_player')
                print(f"Total jugadores para la liga {liga} es {total_jugadores.shape[0]}")
                total_lista_ligas.append(total_jugadores)
            except KeyError:
                continue

    try:
        total_players_tot = pd.concat(total_lista_ligas).reset_index().drop('index', axis=1)
        return total_players_tot
    except ValueError:
        print('No se encontraron datos para concatenar: verifique los accesos a la API y vuelva a intentarlo.')
```

Por ultimo

El script se ejecuta en el bloque



lo que significa que cuando ejecutas el script, la función main() se llama para comenzar la recopilación de datos.
Los datos se almacenan en un solo archivo de Excel.
Precisamente, en la función main(), se guarda el DataFrame “final total_players_tot.xlsx”