Trends in the Olympic Decathlon: Data Collection

Aleksandar Bahat

Introduction

The decathlon is track and field's ten-part "combined" event. It consists of:

- the 100m, long jump, shot put, high jump, and 400m on the first day of competition, and
- the 110m hurdles, discus throw, pole vault, javelin throw, and 1500m on the second day.

Each event is scored according to a standardized formula, and ranks are determined by total points across all ten events.

The decathlon has been a part of every modern Summer Olympics since they began in 1912. Because there's so much quantitative data associated with it — that's 25 Olympics, every one with dozens of decathletes, each of whom generates 10 times, distances, and heights, *and* 10 scores to match — it's a data analyst's dream! Its complexity invites many questions, such as:

- Do Olympic decathletes tend to favor certain events? Put another way, where should decathletes focus their training? Does the current scoring system fairly balance running, jumping, and throwing?
- Do Olympic decathletes tend towards a certain body type? A certain age?
- How do the winners and medalists differ from the rest of the competitors?
- How have the answers to those questions changed over time? Have Olympic decathletes gotten better at hurdling? Worse at shot putting? Taller? Slimmer? And so on.

This notebook contains the first half of this project, the data collection. The data we need is extensive and spread across many different webpages, so better to automate the scraping rather than copying and pasting tables manually. In the next notebook we'll do the analysis.

Our primary source is the decathlon page on Olympedia.org, a thorough archive of Olympic results and athlete biographies. However, it's not quite complete: it includes the overall results from the Tokyo 2020 decathlon, but not the results for each of its constituent events. So we get that data from the relevant Wikipedia page instead.

For each Olympic year except 2020, we'll create an "overall" DataFrame, a "biographical" DataFrame, and an "events" DataFrame. Then we'll combine those three DataFrames into one. All of those DataFrames will be indexed by athlete. To finish, we'll combine the DataFrames from every year (including one for 2020, created separately) into one multi-indexed DataFrame, with year as the primary index and athlete as the secondary index. This master DataFrame gets exported to a CSV file. Numerous local corrections are required along the way, since the data is often formatted inconsistently or contains errors.

Libraries

[1]:
```
import requests
from bs4 import BeautifulSoup
import functools
from decimal import *
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
```

Navigating within and between the relevant webpages

There are three HTML elements on the Olympedia webpages we need to find:

- "striped" tables,
- tables of class "biodata",
- drop-down menus with the name "Parts" and ID "runs".

The three functions below take a URL (as a string) and return the first occurrence of each such element at that URL (as a BeautifulSoup Tag object).

[2]:
```
def striped_table(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    table = soup.find('table', {'class':'table table-striped'})

    return table
```

[3]:
```
def biodata_table(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    table = soup.find('table', {'class':'biodata'})

    return table
```

[4]:
```
def parts_menu(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    menus = soup.find_all('select')

    for menu in menus:
        if 'runs' in menu['id']:
            return menu
```

We also need to find the URLs of relevant subpages, starting from the top-level decathlon page cited in the Introduction. Note that we memoize these functions to reduce runtime; this becomes essential later when we need to find and read from very many webpages to create the DataFrames.

[5]:
```
URL_PREFIX = 'https://www.olympedia.org'
URL_TOP = 'https://www.olympedia.org/event_names/93'
```

[6]:
```
@functools.cache
def overall_url(year):
    table = striped_table(URL_TOP)
    rows = table.find_all('tr')

    for row in rows[1:]:
        row_data = row.find_all('td')
        if row_data[0].text == year:
            rel_url = row_data[1].find('a')['href']
            return URL_PREFIX + rel_url
```

[7]:
```
@functools.cache
def bio_url(year, athlete):
    table = striped_table(overall_url(year))
    rows = table.find_all('tr')

    for row in rows[1:]:
        row_data = row.find_all('td')
        if row_data[2].text.strip() == athlete:
            rel_url = row_data[2].find('a')['href']
            return URL_PREFIX + rel_url
```

[8]:
```
@functools.cache
def event_url(year, event):
    menu = parts_menu(overall_url(year))
    options = menu.find_all('option')

    for option in options[1:]:
        if option.text == event:
            rel_url = '/results/' + option['value']
            return URL_PREFIX + rel_url
```

The overall data

Here we collect data from the subpages given by the `overall_url` function. For an example, see the overall page for the 2016 decathlon. The desired attributes are the athletes' overall ranks, overall points, and nationalities. Sometimes the results for the constituent events are also included on these subpages, but not always; because of this lack of consistency, it's easier to collect that data separately.

The decathlon's scoring system changed a few times during the 20th century. Since we want to compare performances across time, we'll use the modern scoring system (first implemented in 1985) for everything. Some earlier Olympics have both hand-timed and automatically-timed results for the running events; we always choose the automatic times.

All the numerical data we collect will be represented with the Decimal type, as the inexactness of floating point

arithmetic causes problems with the score calculations later on. For our purposes, the performance disadvantage compared to using the float type is not significant.

[9]:
```
def d(x):
    try:
        return Decimal(x)
    except:
        return Decimal('NaN')
```

Some of the country codes in the data no longer exist. Since we'll want to visualize the geographic distribution of participants, medalists, etc. later on, we replace those country codes with their modern equivalents. The mapping from old to new codes was created manually by looking up the nationalities of the 61 competitors whose original states no longer exist, and is stored in the `old_to_new_IOC.csv` file.

We also need to convert IOC country codes to the more common 3-letter ISO codes. The mapping in the file `IOC_to_ISO.csv` comes from the table at WorldData.info here.

[10]:
```
@functools.cache
def old_to_new_IOC(old_ioc, athlete):
    D = pd.read_csv('old_to_new_IOC.csv', index_col=[0,1])
    try:
        return D.at[(old_ioc, athlete), 'New IOC']
    except:
        return old_ioc
```

[11]:
```
@functools.cache
def IOC_to_ISO(ioc):
    D = pd.read_csv('IOC_to_ISO.csv', index_col=0)
    return D.at[ioc, 'ISO']
```

Now we can assemble the overall data for a given year.

[12]:
```
@functools.cache
def overall_data(year):
    table = striped_table(overall_url(year))
    unfiltered_data = pd.read_html(str(table), index_col='Athlete')[0]

    for points_type in (' (1985 Auto Tables)', ' (1985 Hand Tables)', ''):
        if 'Points' + points_type in unfiltered_data.columns:
            points = 'Points' + points_type
            break
    if year == '1972':    # 1972 auto-time scores on Olympedia are mislabeled
        points = 'Points (1985 Hand Tables)'
    cols = ['Pos', points, 'NOC']
    data = unfiltered_data.reindex(columns=cols)

    data = data.replace({'=':''}, regex=True)    # Removes '=' from tied ranks
    data = data.rename(columns={'Pos':'Overall Rank',
```

```
                                points:'Overall Points',
                                'NOC':'Country'})

    for col in ('Overall Rank', 'Overall Points'):
        data[col] = data[col].apply(d)
        data[col] = data[col].apply(round, args=(0,))

    if year == '1920':    # One row of the 1920 data is entirely empty
        data = data.drop(index=np.nan)
    if year == '1968':    # Some 1968 scores on Olympedia are 1 point too low
        data.at['Wu Ah-Min', 'Overall Points'] = d('7155')
        data.at['Franz Biedermann', 'Overall Points'] = d('6221')

    data = data.reset_index()   # 'Athlete' column to be an argument of new_NOC
    data['Country'] = data.apply(lambda x: old_to_new_IOC(x.Country, x.Athlete),
                                 axis=1)
    data['Country'] = data['Country'].apply(IOC_to_ISO)
    data = data.set_index('Athlete')

    return data
```

Let's test the `overall_data` function for the year 2016.

[13]:
```
overall_data('2016').head(3)
```

[13]:

Athlete	Overall Rank	Overall Points	Country
Ashton Eaton	1	8893	USA
Kévin Mayer	2	8834	FRA
Damian Warner	3	8666	CAN

The biographical data

Here we collect data from the subpages given by the `bio_url` function. For an example, see the biography page for Ashton Eaton. The desired attributes are the athletes' heights, weights, and ages at the given Olympics. Because of how the data is formatted, we need a few auxiliary functions to obtain this information.

First, we need a way of extracting an athlete's birth date from the webpage. The birth date is embedded in a table entry whose form is usually, but not always, "[day] [month] [year] in [birth place]". Sometimes the day is missing, sometimes the day and the month are both missing, in one instance the phrase includes the word "circa" because the precise year is unknown, and sometimes no birth date is listed at all.

- If the birth day is missing but the month and year are known, we set the day to the 16th, halfway through the month.
- If the birth month is missing but the year is known, we set the day and month to the 2nd of July, halfway through the year.

- If an approximate year is known (the "circa" case), we treat it as the exact year and then do the same thing as above.

There are few enough exceptions of this kind that no general age trends will be obscured by these minor approximations.

The `extract_date` function takes a string as input and returns a datetime object as output.

[14]:
```
def extract_date(date_in_place):
    date = date_in_place.partition(' in ')[0]
    date_format = '%d %B %Y'

    if 'circa' in date:
        date = ''.join([d for d in date if d.isdigit()])
    if date.isnumeric():
        date = '2 July ' + date

    if date.isalpha() or not date:
        return pd.NaT
    try:
        return datetime.strptime(date, date_format)
    except:
        return datetime.strptime('16 ' + date, date_format)
```

We also need to know the start date of the Olympic decathlon in each year.

[15]:
```
@functools.cache
def oly_date(year):
    table = biodata_table(overall_url(year))
    rows = table.find_all('tr')
    for row in rows:
        row_header = row.find('th')
        if row_header.text == 'Date':
            dates = row.find('td').text

    start_day = next(x for x in dates.split() if x.isdigit())
    start_month = next(x for x in dates.split() if x.isalpha())
    year = next(x for x in reversed(dates.split()) if x.isdigit())
    date = ' '.join([start_day, start_month, year])
    date_format = '%d %B %Y'

    return datetime.strptime(date, date_format)
```

Calculating age (in years, as a Decimal rounded to the nearest hundredth) from two datetime objects is a simple subtraction.

[16]:
```
def age(birth_date, current_date):
    age_in_days = d((current_date - birth_date).days)
    age = round(age_in_days / d(365.25), 2)     # Ignores leap years
```

```
    return age
```

We also need to be able to extract heights and weights from table entries of the form "[height] / [weight]". Sometimes one of the measurements is missing. For at least one athlete, the weight is given as a range, in which case we return the middle of the range. All heights and weights in the archive are in cm and kg, respectively, so again we omit units. These functions both return Decimals rounded to the nearest integer.

[17]:
```
def extract_height(height_and_weight):
    height = height_and_weight.partition(' cm')[0]
    height = round(d(height), 0)

    return height
```

[18]:
```
def extract_weight(height_and_weight):
    weight = height_and_weight.partition(' / ')[2].partition(' kg')[0]

    if '-' in weight:
        low_weight = d(weight.partition('-')[0])
        high_weight = d(weight.partition('-')[2])
        weight = round((low_weight + high_weight) / d(2), 0)
    else:
        weight = round(d(weight), 0)

    return weight
```

Finally, we need a list of all the athletes competing in a given year's decathlon.

[19]:
```
@functools.cache
def athletes(year):
    return [x for x in list(overall_data(year).index) if isinstance(x, str)]
```

Now we can assemble the biographical data for a given year, first for one athlete, then for all the year's athletes.

[20]:
```
@functools.cache
def bio_data(year, athlete):
    table = biodata_table(bio_url(year, athlete))
    unfiltered_data = pd.read_html(str(table), index_col=0)[0]

    unfiltered_data = unfiltered_data.T
    cols = ['Born','Measurements']
    data = unfiltered_data.reindex(columns=cols, fill_value='')

    birth_date = data['Born'].apply(extract_date)
    data['Age'] = birth_date.apply(age, current_date=oly_date(year))
    data['Height'] = data['Measurements'].apply(extract_height)
    data['Weight'] = data['Measurements'].apply(extract_weight)

    data = data.drop(columns=['Born', 'Measurements'])
```

```
    data = data.rename_axis('Athlete')
    data = data.rename(index={1:athlete})

    return data
```

[21]:
```
@functools.cache
def bios_data(year):
    frames = [bio_data(year, athlete) for athlete in athletes(year)]
    return pd.concat(frames)
```

Testing the `bio_data` and `bios_data` functions for the year 2016:

[22]:
```
bio_data('2016', 'Ashton Eaton')
```

[22]:
```
                Age Height Weight
Athlete
Ashton Eaton  28.57    186     81
```

[23]:
```
bios_data('2016').head(3)
```

[23]:
```
                 Age Height Weight
Athlete
Ashton Eaton   28.57    186     81
Kévin Mayer    24.52    186     77
Damian Warner  26.78    185     83
```

The events data

Here we collect data from the subpages given by the `event_url` function. For an example, see the event page for the 2016 decathlon 100m. The desired attributes are the athletes' results (times, distances, or heights), ranks, and scores in the given event. As with the biographical data, we need a couple of auxiliary functions.

Times for the 1500m are given in the usual "[minutes]:[seconds]" format. For ease of calculation and consistency with the other running events, we'll use the following function to convert these times to seconds, taking a string as input and returning a Decimal as output. (Times from the 400m occur both in seconds and in the "[minutes]:[seconds]" format, hence the "if" statement.)

[24]:
```
def extract_seconds(time):
    if type(time) == float or time.replace('.', '', 1).isdigit():
        sec = time
    else:
        sec = pd.to_timedelta('00:0' + time, errors='coerce').total_seconds()

    return round(d(sec), 2)
```

A few of the long jump distances have a "w" suffix to indicate that the jump was wind-aided: jumps with a >2.0 m/s tailwind can't be counted as records, though they are still valid within a given decathlon competition. Wind-aided jumps are rare enough that it will not cause confusion to simply drop the "w" when it occurs.

[25]:
```
def remove_w(distance):
    try:
        return distance.replace('w', '')
    except:
        return distance
```

In order to make historical comparisons, we need to use a single scoring system for all Olympic years. Naturally, we choose the modern scoring system. Although event scores *are* included on these subpages, these are the scores according to the scoring system *during that year*, not necessarily the modern one. So we calculate the scores ourselves, using the raw results (times, distances, heights) along with the modern scoring formulas implemented below. The formula parameters a, b, and c are set by the governing body World Athletics; see here for details.

[26]:
```
events = ('100 metres', 'Long Jump',
          'Shot Put', 'High Jump',
          '400 metres', '110 metres Hurdles',
          'Discus Throw', 'Pole Vault',
          'Javelin Throw', '1,500 metres')

runs = ('100 metres', '400 metres', '110 metres Hurdles', '1,500 metres')
jumps = ('Long Jump', 'High Jump', 'Pole Vault')
throws = ('Shot Put', 'Discus Throw', 'Javelin Throw')
```

[27]:
```
def points(x, event):
    SP = pd.read_csv('scoring_parameters.csv',
                     index_col=0,
                     converters={'a':d, 'b':d, 'c':d})
    a = SP['a'].to_dict()
    b = SP['b'].to_dict()
    c = SP['c'].to_dict()

    x = d(str(x)).quantize(d('.01'), rounding=ROUND_DOWN)

    if x.is_nan():
        return d('NaN')

    if event in runs:
        if x.compare(b[event]) == d('-1'):
            p = a[event] * (b[event] - x)**c[event]
        else:
            p = d('0')
    if event in jumps:
        if (x*d('100')).compare(b[event]) == d('1'):
            p = a[event] * (x*d('100') - b[event])**c[event]
        else:
            p = d('0')
    if event in throws:
        if x.compare(b[event]) == d('1'):
```

```
            p = a[event] * (x - b[event])**c[event]
        else:
            p = d('0')
    p = p.quantize(d('1'), rounding=ROUND_DOWN)

    return p
```

To distinguish between hand times and automatic times, we define a "precision" function to keep track of the number of decimal places in the reported results for each event and year. Hand times are always reported to the nearest 0.1 seconds, while automatic times are always reported to the nearest 0.01 seconds. While distances and heights are almost always reported to the nearest centimeter, distances were reported to the nearest millimeter in some early Olympics.

[28]:
```
def precision(year, event):
    year = int(year)

    if event in ('100 metres', '100 metres Time'):
        return 1 if (year <= 1948 or year in (1964,)) else 2
    if event in ('Long Jump', 'Long Jump Distance'):
        return 3 if (year in (1920, 1924, 1948)) else 2
    if event in ('Shot Put', 'Shot Put Distance'):
        return 3 if (year in (1924,)) else 2
    if event in ('High Jump', 'High Jump Height'):
        return 2
    if event in ('400 metres', '400 metres Time'):
        return 1 if (year <= 1948 or year in (1960, 1964)) else 2
    if event in ('110 metres Hurdles', '110 metres Hurdles Time'):
        return 1 if (year <= 1948 or year in (1964,)) else 2
    if event in ('Discus Throw', 'Discus Throw Distance'):
        return 3 if (year in (1920, 1924)) else 2
    if event in ('Pole Vault', 'Pole Vault Height'):
        return 2
    if event in ('Javelin Throw', 'Javelin Throw Distance'):
        return 3 if (year in (1920, 1924)) else 2
    if event in ('1,500 metres', '1,500 metres Time'):
        return 1 if (year <= 1948 or year in (1960, 1964, 1972, 1980)) else 2
```

Now we can assemble the event data for a given year, first for one event, then for all the year's events. We always use automatically-timed results for the running events when available; up to 1948, the running results were only hand-timed. In order to calculate the scores correctly according to the modern system, we must add 0.24 seconds to hand-timed 100m and 110m results, and 0.14 seconds to hand-timed 400m results.

[29]:
```
@functools.cache
def event_data(year, event):
    table = striped_table(event_url(year, event))
    unfiltered_data = pd.read_html(str(table), index_col='Athlete')[0]
```

```
    for results_type in ('Time', 'T(A)', 'T(H)', 'Distance', 'BHC'):
        if results_type in unfiltered_data.columns:
            results = results_type
            break
    cols = ['Pos', results]
    data = unfiltered_data.reindex(columns=cols)

    if event in ('Long Jump'):
        data[results] = data[results].apply(remove_w)
    if event in ('400 metres', '1,500 metres'):
        data[results] = data[results].apply(extract_seconds)

    data[results] = data[results].apply(str).apply(d)
    data[results] = data[results].apply(round, args=(precision(year, event),))

    auto_results = data[results]
    if event in ('100 metres', '110 metres Hurdles'):
        if int(year) <= 1948 or int(year) in (1964,):
            auto_results = auto_results + d('0.24')
    if event in ('400 metres',):
        if int(year) <= 1948 or int(year) in (1960, 1964):
            auto_results = auto_results + d('0.14')
    data[event + ' Points'] = auto_results.apply(points, args=(event,))

    data = data.replace({'=':''}, regex=True)
    data = data.replace({0:'', '0':''}, regex=False)    # Removes null results
    renamed_results = results
    if results in ('T(A)', 'T(H)'):
        renamed_results = 'Time'
    if results in ('BHC'):    # 'BHC' means 'best height cleared'
        renamed_results = 'Height'
    data = data.rename(columns={'Pos':event + ' Rank',
                                results:event + ' ' + renamed_results})

    data[event + ' Rank'] = data[event + ' Rank'].apply(d)
    data[event + ' Rank'] = data[event + ' Rank'].apply(round, args=(0,))

    # Some 1952 hurdles results have no automatic times, only hand times
    if year == '1952' and event == '110 metres Hurdles':
        data.at['Ignace Heinrich', event + ' Time'] = d('16.0')
        data.at['Brígido Iriarte', event + ' Time'] = d('16.6')
        data.at['Reynaldo Oliver', event + ' Time'] = d('16.7')
        data.at['Eeles Landström', event + ' Time'] = d('17.1')
        data.at['Ignace Heinrich', event + ' Points'] = points('16.24', event)
        data.at['Brígido Iriarte', event + ' Points'] = points('16.84', event)
        data.at['Reynaldo Oliver', event + ' Points'] = points('16.94', event)
        data.at['Eeles Landström', event + ' Points'] = points('17.34', event)
```

```
    return data
```

[30]:
```
@functools.cache
def events_data(year):
    frames = [event_data(year, event) for event in events]
    data = pd.concat(frames, axis=1)
    data = data.fillna(d('NaN'))

    return data
```

Testing the `event_data` and `events_data` functions for the year 2016:

[31]:
```
event_data('2016', '100 metres').head(3)
```

[31]:
```
             100 metres Rank 100 metres Time 100 metres Points
Athlete
Damian Warner               1           10.30              1023
Ashton Eaton                2           10.46               985
Zach Ziemek                 3           10.71               926
```

[32]:
```
events_data('2016').head(3)
```

[32]:
```
             100 metres Rank 100 metres Time 100 metres Points  \
Athlete
Damian Warner               1           10.30              1023
Ashton Eaton                2           10.46               985
Zach Ziemek                 3           10.71               926

             Long Jump Rank Long Jump Distance Long Jump Points  \
Athlete
Damian Warner              3               7.67              977
Ashton Eaton               1               7.94             1045
Zach Ziemek                9               7.49              932

             Shot Put Rank Shot Put Distance Shot Put Points High Jump Rank  \
Athlete
Damian Warner           23             13.66             708             10
Ashton Eaton            10             14.73             773             14
Zach Ziemek             25             13.44             694              5

              ... Discus Throw Points Pole Vault Rank Pole Vault Height  \
Athlete       ...
Damian Warner ...                765              15              4.70
Ashton Eaton  ...                777               3              5.20
Zach Ziemek   ...                858               3              5.20
```

```
              Pole Vault Points Javelin Throw Rank Javelin Throw Distance  \
Athlete
Damian Warner               819                 11                   63.19
Ashton Eaton                972                 18                   59.77
Zach Ziemek                 972                 15                   60.92

              Javelin Throw Points 1,500 metres Rank 1,500 metres Time  \
Athlete
Damian Warner                  786                5           264.90
Ashton Eaton                   734                4           263.33
Zach Ziemek                    752               18           282.97

              1,500 metres Points
Athlete
Damian Warner                 778
Ashton Eaton                  789
Zach Ziemek                   662

[3 rows x 30 columns]
```

Putting it all together

Now that the component DataFrames are complete, we can combine all the data for a given year.

[33]:
```
@functools.cache
def all_types_data(year):
    frames = [overall_data(year), bios_data(year), events_data(year)]
    data = pd.concat(frames, axis=1)

    sum_cols = [x for x in data.columns
                if 'Points' in x and 'Overall' not in x]
    if year == '1952':    # 1952 scores are miscalculated on Olympedia
        col = 'Overall Points'
        data[col] = data[col].where(~(data['Overall Rank'] > 0),
                                    data[sum_cols].sum(axis=1).apply(d))

    return data
```

Testing the `all_types_data` function for the year 2016:

[34]:
```
all_types_data('2016').head(3)
```

[34]:
```
              Overall Rank Overall Points Country    Age Height Weight  \
Athlete
Ashton Eaton             1           8893     USA  28.57    186     81
Kévin Mayer              2           8834     FRA  24.52    186     77
Damian Warner            3           8666     CAN  26.78    185     83
```

```
             100 metres Rank 100 metres Time 100 metres Points  \
Athlete
Ashton Eaton               2           10.46               985
Kévin Mayer                9           10.81               903
Damian Warner              1           10.30              1023

             Long Jump Rank  ... Discus Throw Points Pole Vault Rank  \
Athlete                      ...
Ashton Eaton              1  ...                 777               3
Kévin Mayer               5  ...                 804               1
Damian Warner             3  ...                 765              15

             Pole Vault Height Pole Vault Points Javelin Throw Rank  \
Athlete
Ashton Eaton              5.20              972                18
Kévin Mayer               5.40             1035                 6
Damian Warner             4.70              819                11

             Javelin Throw Distance Javelin Throw Points 1,500 metres Rank  \
Athlete
Ashton Eaton                  59.77                  734                 4
Kévin Mayer                   65.04                  814                 6
Damian Warner                 63.19                  786                 5

             1,500 metres Time 1,500 metres Points
Athlete
Ashton Eaton            263.33                 789
Kévin Mayer             265.49                 774
Damian Warner           264.90                 778

[3 rows x 36 columns]
```

Finally, we put the `all_types_data` DataFrames from every Olympic year together. Since the scraping functions used here are not applicable for the year 2020, we import the 2020 data from a CSV file (created manually from the results on Wikipedia).

[35]:
```
years = [str(x) for x in range(1912, 2020, 4) if x not in (1916, 1940, 1944)]
```

[36]:
```
@functools.cache
def all_types_data_2020():
    data = pd.read_csv('decathlon_data_2020.csv',
                       index_col='Athlete')

    rank_cols = [x for x in data.columns if ' Rank' in x]
    points_cols = [x for x in data.columns if ' Points' in x]
    htwt_cols = ['Height', 'Weight']
```

```
    int_cols = rank_cols + points_cols + htwt_cols

    age_col = ['Age']

    time_cols = [x for x in data.columns if ' Time' in x]
    distance_cols = [x for x in data.columns if ' Distance' in x]
    height_cols = [x for x in data.columns if ' Height' in x]
    results_cols = time_cols + distance_cols + height_cols

    num_cols = int_cols + age_col + results_cols

    for col in num_cols:
        data[col] = data[col].apply(d)
        if col in int_cols:
            data[col] = data[col].apply(round, args=(0,))
        if col in age_col:
            data[col] = data[col].apply(round, args=(2,))
        if col in results_cols:
            data[col] = data[col].apply(round, args=(precision('2020', col),))

    data = data.reset_index()    # 'Athlete' column to be an argument of new_NOC
    data['Country'] = data.apply(lambda x: old_to_new_IOC(x.Country, x.Athlete),
                                 axis=1)
    data['Country'] = data['Country'].apply(IOC_to_ISO)
    data = data.set_index('Athlete')

    return data
```

[37]:
```
@functools.cache
def decathlon_data():
    frames = [all_types_data(year) for year in years] + [all_types_data_2020()]
    data = pd.concat(frames, keys=years + ['2020'], names=['Year'])

    return data
```

[38]:
```
def export_decathlon_data():
    decathlon_data().to_csv('decathlon_data.csv')
    print('Exported!')
```

We conclude by executing the export, and displaying the head and tail of the final DataFrame for good measure:

[39]:
```
export_decathlon_data()
```

```
Exported!
```

[40]:
```
display(decathlon_data().head(3))
display(decathlon_data().tail(3))
```

```
                    Overall Rank Overall Points Country    Age Height Weight  \
Year Athlete
1912 Jim Thorpe                1           6564     USA  25.13    183     86
     Hugo Wieslander           2           5966     SWE  23.09    182     81
     Charles Lomberg           3           5722     SWE  25.60    182     75

                    100 metres Rank 100 metres Time 100 metres Points  \
Year Athlete
1912 Jim Thorpe                  3            11.2               765
     Hugo Wieslander            13            11.8               643
     Charles Lomberg            13            11.8               643

                    Long Jump Rank  ... Discus Throw Points Pole Vault Rank  \
Year Athlete                        ...
1912 Jim Thorpe                 3   ...                 603               3
     Hugo Wieslander            8   ...                 590               8
     Charles Lomberg            1   ...                 571               3

                    Pole Vault Height Pole Vault Points Javelin Throw Rank  \
Year Athlete
1912 Jim Thorpe                  3.25              418                  4
     Hugo Wieslander             3.10              381                  1
     Charles Lomberg             3.25              418                  7

                    Javelin Throw Distance Javelin Throw Points  \
Year Athlete
1912 Jim Thorpe                      45.70                 525
     Hugo Wieslander                 50.40                 595
     Charles Lomberg                 41.83                 469

                    1,500 metres Rank 1,500 metres Time 1,500 metres Points
Year Athlete
1912 Jim Thorpe                    1             280.1                 680
     Hugo Wieslander               6             285.0                 649
     Charles Lomberg              12             312.2                 492

[3 rows x 36 columns]
                              Overall Rank Overall Points Country    Age  \
Year Athlete
2020 Cedric Dubler                      21           7008     AUS  26.56
     Niklas Kaul                       NaN            NaN     DEU  23.48
     Thomas Van der Plaetsen           NaN            NaN     BEL  30.61
```

```
                           Height Weight 100 metres Rank 100 metres Time  \
Year Athlete
2020 Cedric Dubler            191    NaN             13           10.89
     Niklas Kaul              190     90             21           11.22
     Thomas Van der Plaetsen  186     81             17           11.05

                           100 metres Points Long Jump Rank  ...  \
Year Athlete                                                 ...
2020 Cedric Dubler                       885            10  ...
     Niklas Kaul                         812            10  ...
     Thomas Van der Plaetsen             850           NaN  ...

                           Discus Throw Points Pole Vault Rank  \
Year Athlete
2020 Cedric Dubler                         732            NaN
     Niklas Kaul                           NaN            NaN
     Thomas Van der Plaetsen               NaN            NaN

                           Pole Vault Height Pole Vault Points  \
Year Athlete
2020 Cedric Dubler                       NaN              NaN
     Niklas Kaul                         NaN              NaN
     Thomas Van der Plaetsen             NaN              NaN

                           Javelin Throw Rank Javelin Throw Distance  \
Year Athlete
2020 Cedric Dubler                        13                  58.52
     Niklas Kaul                         NaN                    NaN
     Thomas Van der Plaetsen             NaN                    NaN

                           Javelin Throw Points 1,500 metres Rank  \
Year Athlete
2020 Cedric Dubler                          716               21
     Niklas Kaul                            NaN              NaN
     Thomas Van der Plaetsen                NaN              NaN

                           1,500 metres Time 1,500 metres Points
Year Athlete
2020 Cedric Dubler                   303.69                539
     Niklas Kaul                        NaN                NaN
     Thomas Van der Plaetsen            NaN                NaN

[3 rows x 36 columns]
```

That's it for the data collection! Now the investigation can begin…