



Concept 2 Logbook Scraper

<https://github.com/adhardy/Concept2-Logbook-Scraper>



The Website



LOGBOOK RANKINGS CHALLENGES

2021 One Minute Indoor Rower Rankings

Weight: All | Gender: All | Ages: All | Country: All | All Results | Adaptive: No

Pos.	Name	Age	Location	Country	Club/Affiliation	Distance	Type	Verified
1	Chris Scott	32	Portsmouth	GBR	MAD Team IRC	420	I	Yes
2	Tom Wilson	28	Nottingham	GBR	TEAM OARSOME IRC	412	I	Yes
3	Václav Zitta	38	Prague	CZE		411	I	Yes
4	Scott Keane	47	Beverton, OR	USA		410	S	Yes
5	Jason Marshall	35	Port Coquitlam, BC	CAN	GARAGEATHLETE	408	I	Yes
6	Chris Glasgow	36	Pearl, MS	USA	GARAGEATHLETE	407	I	Yes
6	David Rockstraw	38	Sunderland	GBR		407	I	Yes
8	Simon Handley	46		GBR	MAD Team IRC	406	I	Yes
9	Cameron Whanam	32	Comox Valley, BC	CAN	TEAM OARSOME IRC	405	I	Yes
10	Heikki Vierela	50	Turku	FIN	Finnish Defence Forces	404	S	Yes
11	Anders Jacobsen	40		NOR		403	I	Yes
11	Joel Smith	41	Brisbane, QLD	AUS	Jornsvikings	403	I	Yes
13	Joschim Bratt	25		SWE		402	I	No
13	Matt Snares	36	TN	USA	Gym Mettle	402	I	Yes
13	Keew Mewell	33	Mount Maunganui	NZL	RAW Fitness	402	I	Yes
13	Keaton Rutherford	30	Sydney, NSW	AUS	GARAGEATHLETE	402	I	Yes
17	Tyler Smith	32	Columbus, OH	USA	GARAGEATHLETE	401	I	Yes
18	Derek Baillie	33	Oklahoma City, OK	USA	GARAGEATHLETE	400	I	Yes
19	Chaz Schilems	34	Phoenix, AZ	USA	Phoenix Fire Rowing	399	I	Yes
20	Mark Roberson	53	Cambridge	GBR	MAD Team IRC	398	I	Yes
21	Benjamin Smith	32	Brisbane, QLD	AUS	Fusion Sport	397	I	Yes
22	Chris Barnes	32	Scunthorpe	GBR	Bojangles Gym	396	I	Yes
23	Nathan Cooper	36	Bald Park, QLD	AUS		386	I	Yes

Chris Scott

Age: 32

Gender: Male

Weight Class: Hwt

Country: United Kingdom

Verified: Yes

Type: Indoor Rower

Time: 1:00.0

Distance: 420m

Pace: 1:11.4

Date: November 30, 2020 13:11:00

Entered: ErgData iOS

View Profile



LOGIN

SIGN UP

LOGBOOK RANKINGS CHALLENGES

Search Rankings

Type
☒ Indoor Rower
☐ SkiErg
☐ BikeErg
☐ Ergation

Indoor Rower

Event

Season

Age Range

Weight

Sex

Country

Verified

Filter by
Adaptive

Search

RANKING MENU

RANKING HELP

INDOOR ROWER RECORDS

INDOOR ROWER RECORD REQUIREMENTS

ADAPTIVE ROWING CATEGORIES

ADAPTIVE SKIING CATEGORIES

SKIERS RECORDS

SKIERS RECORD REQUIREMENTS

CONTACT US

Rules and Regulations

- Ranking pieces can be done on any model Concept2 Indoor Rower, Concept2 SkiErg or Concept2 BikeErg.
- All ranking pieces must be single distance or timed pieces (no interval pieces) started from a non-moving flywheel. Accelerated starts from moving flywheels during intervals are not accepted as ranking times.
- The machine should be on a level surface during the piece.
- Modifications to the machine that alter its performance are not allowed. Examples of allowed modifications: seating, pedals, handlebars. Please check with Concept2 if you are unsure.
- There is no required damper setting. You are free to choose the setting you prefer and you may change the setting during the piece as long as it is done by the person using the machine. Learn more about damper settings.
- Indoor rower pieces may be rowed with or without slides. The exception: all 2000 meter record-breaking times must be rowed on Concept2 Indoor Rowers without slides.
- The distance pieces should be set up as a preset workout so the meters count down. To learn how to preset a workout on your monitor, please see the Monitor Support section on our main website. Having done this, once the piece is completed, the terms of a second will appear with the finish time.
- The timer pieces should be set up as a workout so the time counts down while rowing.
- When using the BikeErg, the air flow to and from the flywheel cannot be obstructed in any way with additional objects or covering materials at any point during the piece or the preceding calibration verification.
- The Ranking is an honor based system. Using any methods to gain an unfair advantage or entering false or erroneous information are not allowed.

Chris Scott

Age: 33

Country: United Kingdom

Location: Portsmouth

Affiliation: MAD Team IRC

Team: MAD Men (MAD Team IRC)

Height: 6 ft 6 in

Weight: 260 lb

Wingspan: 6 ft 10 in

Logbook ID: 1169265

Member since: December 07, 2017

3 years

7,500 Ranking Tables

20,000 pages

81,000 athletes

332,000 workouts

430,000 get requests

Multi-Webbing

<https://github.com/adhardy/Multi-Webbing>

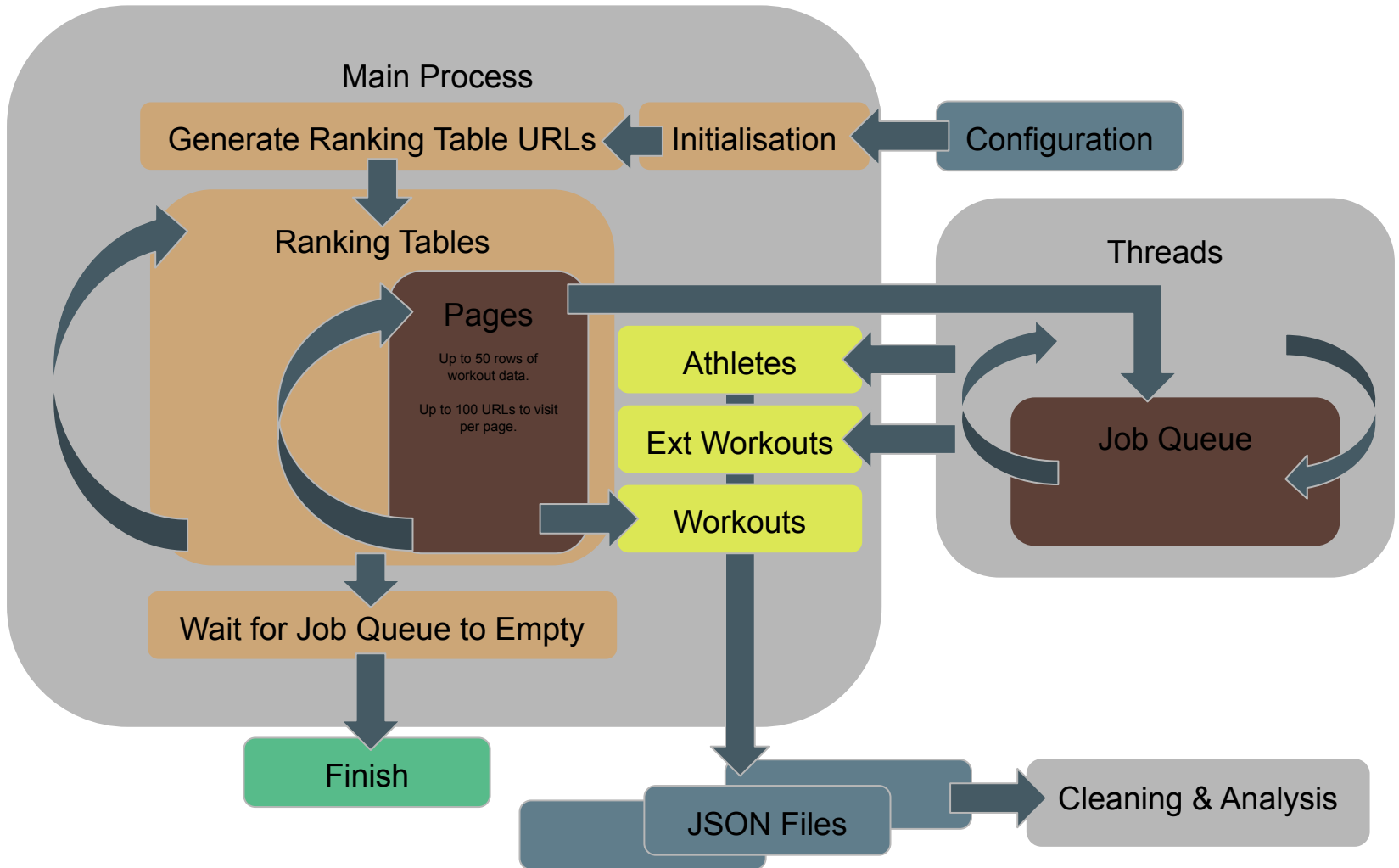
Built on the Threading module.

Allows parallel get requests

Supports “requests” and “selenium”

pip install multi_webbing

```
1 import queue
2 import threading
3 import requests
4 import sys
5 from selenium import webdriver
6 from selenium.webdriver.chrome.options import Options
7
8 class MultiWebbing:
9     """Call this class first to initiate MultiWebbing and the individual threads"""
10     def __init__(self, num_threads, web_module='requests', webdriver=webdriver.Chrome()):
11         #PROM and Harry's tort on our module option
12         #Creates a job queue, lock object, session and creates the number of requested threads"""
13         sys.path
14         self.job_queue = queue.Queue()
15         self.lock = threading.Lock() #Session and lock can be overwritten on a per job basis
16         self.threads = []
17         self.web_module = web_module
18         self.driver = webdriver
19         self.num_threads = num_threads
20         if self.web_module == 'requests':
21             self.session = requests.Session()
22         else:
23             self.session = None
24         for i in range(self.num_threads):
25             self.threads.append(self.Thread(i, self))
26
27     def start(self):
28         """Call after initiating a Threading object to start the threads."""
29         for thread in self.threads:
30             thread.start()
31
32     def finish(self):
33         """When you are ready to finish your threads (e.g. when your work queue is empty and you have visited all pages, call this method to stop and join the threads."""
34         for thread in self.threads:
35             thread.join()
36
37     class Thread(threading.Thread):
38         #Override the Thread function
39         #PROM and Harry's tort on our module option
40         def __init__(self, number, multiwebbing):
41             threading.Thread.__init__(self)
42             self.num_threads = multiwebbing.num_threads
43             self.number = number
44             self.stop_event = threading.Event()
45             self.job_queue = multiwebbing.job_queue
46             self.lock = multiwebbing.lock
47             self.session = multiwebbing.session
48             self.driver = None
49             if self.web_module == 'selenium':
50                 self.options = Options()
51                 self.options.add_argument('--headless')
52                 self.driver = MultiWebbing.driver(options=self.options)
53
54     def run(self):
55         #Override the Thread start()
56         #PROM and Harry's tort on our module option
57         print(f"Starting Thread : {self.number}")
58         while not self.stop_event.isSet():
59             #Thread will continuously check the queue for work until the master process joins the threads, and the stop_event signal is sent
60             try:
61                 job = self.job_queue.get(block=False)
62                 job.set_thread(self) #give job access to thread attributes
63             except queue.Empty:
64                 pass
65             else:
66                 #Print(f"Thread - {self.name} - : Getting profile: {self.job.url}")
67                 #Override the Thread function
68                 self.job.function(job)
69                 #Prints the data structure with the returned data
70                 print(f"Completed thread : {self.number}")
71
72     def join(self, timeout=None):
73         #Used stop event to terminate the work loop before calling join
74         self.stop_event.wait()
75         super().join(timeout)
76         print(f"Joined thread : {self.number}")
77
78 class Job:
79     """Build all the information needed for the worker threads to make a request and execute the job function"""
80     def __init__(self, function, url, custom_data, session = None, lock = None):
81         self.url = url
82         self.function = function
83         self.custom_data = custom_data #your data structure, accessible inside your job function (list, dictionary, list of list, list of dictionaries...)
84         self.request = None
85         self.driver = None
86         self.function = function
87         self.session = None
88         self.lock = None
89
90     def set_thread(self, thread):
91         """Assign access to thread attributes(e.g. session, lock) that may be needed for the job function"""
92         self.thread = thread
93         if self.thread.web_module == "requests":
94             if self.session == None:
95                 self.session = self.thread.session
96             elif self.thread.web_module == "selenium":
97                 self.driver = self.thread.driver
98             if self.lock == None:
99                 self.lock = self.thread.lock
100
101     def get_url_request(self):
102         """Make a get request to job url. Sets job.request to the result, returns 1 upon an error"""
103         try:
104             self.request = self.session.get(self.url)
105         except requests.exceptions.ConnectionError:
106             self.request = None
107             return False
108         return True
109
110     def get_url_selenium(self):
111         """Make a get request to job url. Sets job.request to the result, returns 1 upon an error"""
112         try:
113             self.thread.driver.get(self.url)
114         except #PROM: find the specific connection error/timeout exception name in selenium
115             return False
116         return True
117
118     def get_url(self):
119         if self.thread.web_module == "requests":
120             return self.get_url_request()
121         elif self.thread.web_module == "selenium":
122             return self.get_url_selenium()
123
124     def __str__(self):
125         return self.url
```



Ranking Tables

```
...  
  
tree = html.fromstring(r.text)  
table tree = tree.xpath('//html/body/div[2]/div/main/section[@class="content"]/table')  
  
#get column headings  
columns = table tree[0].xpath('thead/tr/th')  
column headings = [column.text for column in columns]  
  
rows tree = table tree[0].xpath('tbody/tr')  
for row in range(0,num rows):  
    row tree = rows tree[row]  
    self.data.workouts[workout ID] = get workout data(row tree, column  
...  
...
```

2021 One Minute Indoor Rower Rankings

Weight: All | Gender: All | Ages: All | Country: All | All Results | Adaptive: No

Pos.	Name	Age	Location	Country	Club/Affiliation	Distance	Type	Verified
1	Chris Scott	32	Portsmouth	GBR	MAD Team IRC	420	I	Yes
2	Tom Wilson	28	Nottingham	GBR	TEAM OARSOME IRC	412	I	Yes
3	Václav Zita	38	Prague	CZE		411	I	Yes
4	Scott Keane	47	Beaverton, OR	USA		410	S	Yes
5	Jason Marshall	35	Port Coquitlam, BC	CAN	GARAGEATHLETE	408	I	Yes
6	Chris Glasgow	36	Pearl, MS	USA	GARAGEATHLETE	407	I	Yes
6	David Rackstrew	38	Sunderland	GBR		407	I	Yes
8	Simon Handley	46		GBR	MAD Team IRC	406	I	Yes
9	Cameron Wharrem	32	Comox Valley, BC	CAN	TEAM OARSOME IRC	405	I	Yes
10	Heikki Vierela	50	Turku	FIN	Finnish Defence Forces	404	S	Yes
11	Anders Jacobsen	40		NOR		403	I	Yes
11	Joel Smith	41	Brisbane, QLD	AUS	Jomsvikings	403	I	Yes
13	Joachim Bratt	25		SWE		402	I	No
13	Matt Snare	36	TN	USA	Gym Mettle	402	I	Yes
13	Keapa Mewett	33	Mount Maunganui	NZL	RAW Fitness	402	I	Yes
13	Keaton Rutherford	30	Sydney, NSW	AUS	GARAGEATHLETE	402	I	Yes
17	Tyler Smith	32	Columbus, OH	USA	GARAGEATHLETE	401	I	Yes
18	Derek Baillie	33	Oklahoma City, OK	USA	GARAGEATHLETE	400	I	Yes
19	Chaz Schilens	34	Phoenix, AZ	USA	Phoenix Fire Rowing	399	I	Yes

▼<tr>

```
<th>Pos.</th>  
<th>Name</th>  
<th class="hidden-mobile">Age</th>  
<th class="hidden-xs">Location</th>  
<th>Country</th>  
<th class="hidden-xs">Club/Affiliation</th>  
<th>Distance</th>  
<th>Type</th>  
<th>Verified</th>
```

</tr>

</thead>

Ranking Tables

▼ <tbody>

```
▼ <tr id="1">
  <td>1</td>
  ▶ <td>⋮</td>
  <td class="hidden-mobile">32</td>
  <td class="hidden-xs">Portsmouth</td>
  <td>GBR</td>
  <td class="hidden-xs">MAD Team IRC</td>
  <td>420</td>
  <td class="text-center">I</td>
  <td>Yes</td>
</tr>
▶ <tr id="2">⋮</tr>
▶ <tr id="3">⋮</tr>
▶ <tr id="4">⋮</tr>
▶ <tr id="5">⋮</tr>
▶ <tr id="6">⋮</tr>
▶ <tr id="6">⋮</tr>
▶ <tr id="8">⋮</tr>
▶ <tr id="9">⋮</tr>
```

```
def get_workout_data(row_tree, column_headings, ranking_table, profile_ID):
    workout_data = []
    row_data_tree = row_tree.xpath('td | td/a')
    del row_data_tree[1] #remove unwanted element

    #get data elements from each column
    row_list = [x.text for x in row_data_tree]

    #add data from the search parameters
    workout_data = lists2dict(map(str.lower, column_headings), row_list)
    workout_data["year"] = ranking_table.year
    workout_data["machine"] = ranking_table.machine
    workout_data["event"] = ranking_table.event
    workout_data["retrieved"] = strftime("%d-%m-%Y %H:%M:%S", gmtime())
    workout_data["profile_id"] = profile_ID
    for key, val in ranking_table.query_parameters.items():
        Workout_data[key] = val
    return workout_data
```

Athlete Profiles

```
def get_athlete_data(r):
```

```
...
```

```
    #profile labels that are contained in <a> tags
```

```
    a_tag_labels = ["Affiliation:", "Team:"]
```

```
    content = tree.xpath('//section[@class="content"]')
```

```
    #store profile labels as list
```

```
    athlete_profile_labels = content[0].xpath('p/strong')
```

```
    athlete_profile_labels = [label.text for label in athlete_profile_labels]
```

```
    #profile values not contained in tags
```

```
    for profile_label in athlete_profile_labels:
```

```
        #cycle through each profile label and search for the matching value
```

```
        if profile_label in a_tag_labels:
```

```
            profile_value = content[0].xpath('p/strong[contains(text(), "' + profile_label + '")]following-sibling::a/text()')
```

```
        else:
```

```
            profile_value = content[0].xpath('p/strong[contains(text(), "' + profile_label + '")]following-sibling::text()[1]')
```

```
...
```

▼ <p>

33

Age:

Country:

United Kingdom

Location:

Portsmouth

Affiliation:

whitespace

MAD Team IRC

Data Storage - JSON

Pro

Flexible format useful when pages have different/missing data

Highly structured

Easily created from python dictionary

Con

Large - duplication of data labels

Slow to load into pandas!

Ideal → AWS RDS

Easily queried
Compact
Fast
Backed Up

Data Cleaning and Analysis

Merge the three datasets together

```
def merge_frames(self, how="inner"):
    self.merge = pd.merge(
        left=(
            pd.merge(
                left=self.workouts,
                right=self.athletes,
                left_on='profile_id',
                right_on="profile_id",
                right_index=True,
                how=how
            )
        ),
        right = self.extended,
        left_on='workout_id',
        right_on="workout_id",
        right_index=True,
        how=how
    )
```

Assign missing values to NaN

```
df.replace(
    {"": np.nan,
     "None": np.nan,
     None: np.nan},
    inplace=True)
```

Missing Data - Distance

Missing distances <1% of dataset - Drop

```
clean.df.merge = clean.df.merge[clean.df.merge["distance"].notnull()]\nclean.df.merge = clean.df.merge.astype({"distance":int})
```

Height & Weight Conversion

Sample input “5 ft 11 in”, convert to cm

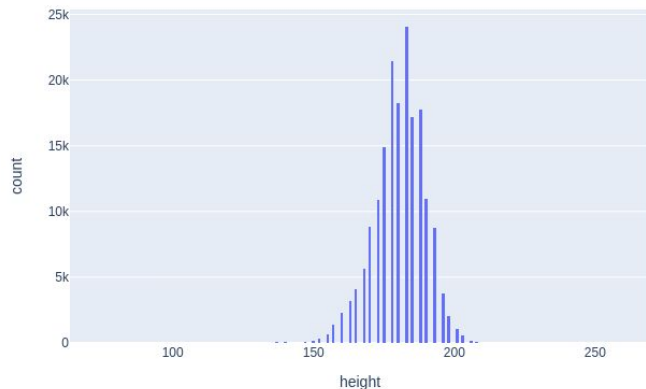
```
df_heights = df_heights.replace({' ft ':' ', ' in':' '}, regex=True)
df_heights = df_heights.str.split(expand=True)
df_heights = df_heights.astype(datatypes)
df_heights["height"] = round(df_heights[0] * ft_to_cm + df_heights[1] * in_to_cm,0)
```

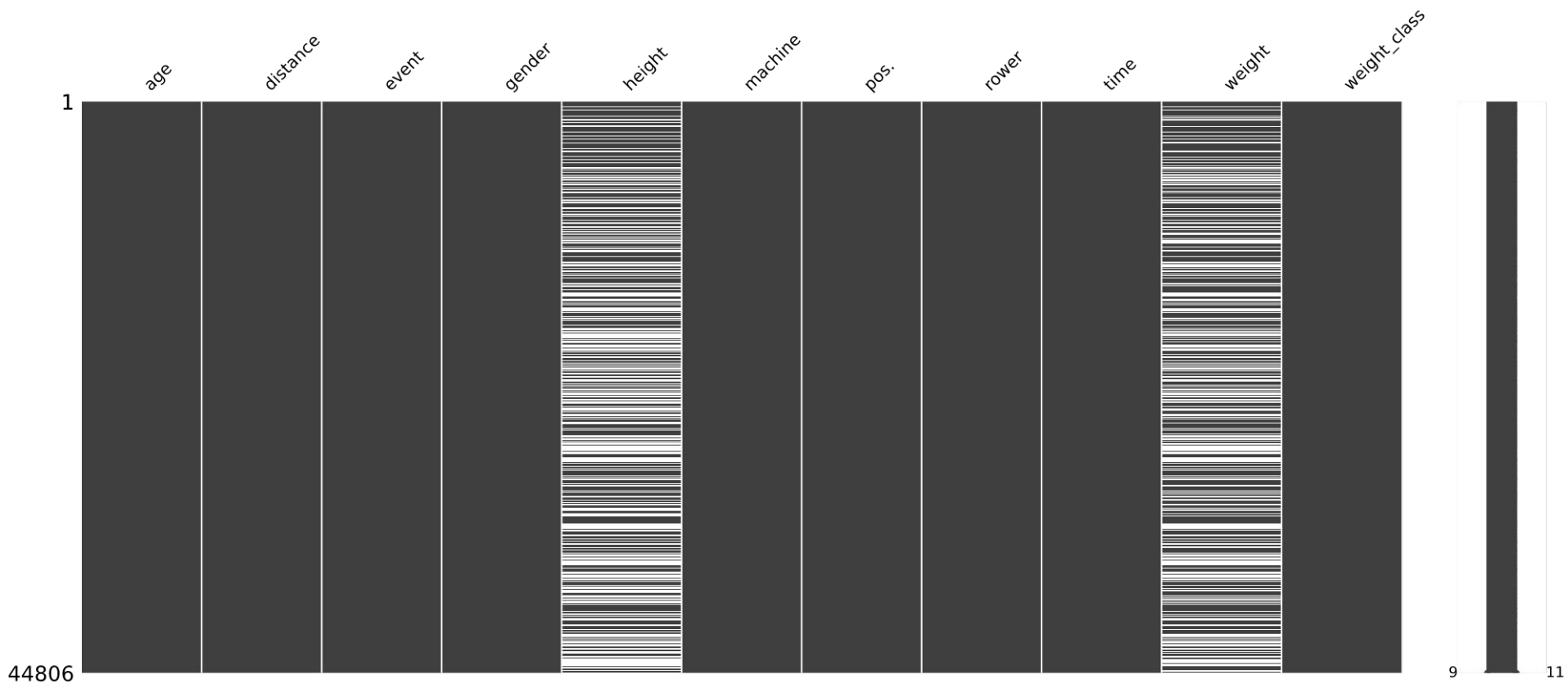
Sample input “160 lbs”, convert to kg

```
df_weights = clean.df.merge[col].replace({' lb':''}, regex=True)
df_weights = df_weights.astype(float)
df_weights = round(df_weights * lbs_kg,1)
```

Form Data (People are Stupid)

```
def clean_heights(height):  
    ft_to_cm = 30.48  
    tallest_human = 272 # weed out impossible heights  
    smallest_human = 60  
  
    # some people entered cm instead of ft and inch, so can recover this by converting back to ft  
    wrong_unit_low = 4300  
    wrong_unit_high = 6096  
    if height > wrong_unit_low and height < wrong_unit_high:  
        return round(height * 1/ft_to_cm,0)  
    if height < smallest_human or height > tallest_human:  
        return np.nan  
    return height
```





Imputing Height and Weight

```
#binarize category columns
```

```
str_cols = df_predict[["event", "gender", "machine", "rower", "weight_class"]].columns
```

```
clfs = {c:LabelEncoder() for c in str_cols}
```

```
for col, clf in clfs.items():
```

```
    df_predict[col] = clfs[col].fit_transform(df_predict[col])
```

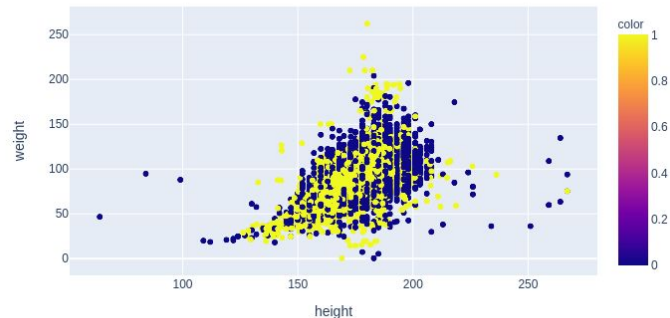
```
#run imputer
```

```
impute_knn = KNNImputer(n_neighbors=3, weights="distance")
```

```
arr_knn = impute_knn.fit_transform(df_predict)
```

```
df_knn = pd.DataFrame(data=arr_knn, columns=df_predict.columns)
```

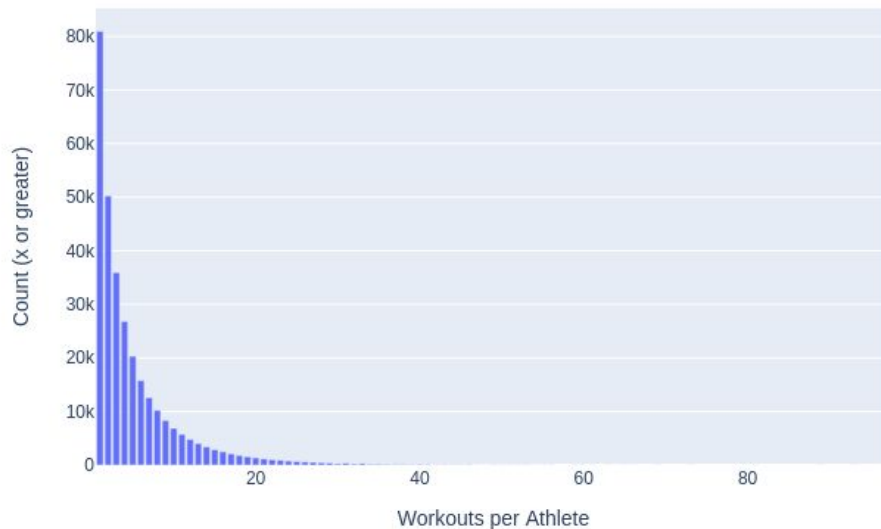
Height vs Weight



Predicting

Given: height, weight, gender, time/distance in other events

Predict: distance/time in new event



Summary

- Multi-thread scraper for simultaneous requests
 - Spun off pip installable module
- Running on EC2
- Site login
- Object Oriented Code Base
- ~300,000 rows
- Cleaned data
 - Strings to integers/floats
 - Bad data dropped/repared
- Missing data
 - Dropped
 - Imputed

Logging In

Lots of profiles hidden to logged out users.

```
def C2_login(session, url_login, username, password, url_login_success):  
    login = session.get(url_login)  
    login_tree = html.fromstring(login.text)  
    hidden_inputs = login_tree.xpath(r'//form//input[@type="hidden"]')  
    form = {x.attrib["name"]: x.attrib["value"] for x in hidden_inputs} #get csrf token  
    form['username'] = username  
    form['password'] = password  
    response = session.post(url_login, data=form)  
    if response.url != url_login_success: #see that we get to the expected page  
        sys.exit("Unable to login to the logbook, quitting.")  
    else:  
        print("Login")  
    return session
```