Queen Mary – University of London

ECS640U/ECS765 - Big Data Processing - 2016/17

Coursework 1  
Twitter analysis with Map Reduce

Student name: **Ivaylo Shalev**  
Programme: **MSc Big Data Science – Part Time**  
Student ID: **ec16544**  
ECSS user ID: **iss30**

**Content of the folder**

Files:

* **Ivaylo Shalev - Coursework 1 - Twitter analysis with Map Reduce – Answers.docx** – this document.
* **CourseWork Helper.txt** – helping file containing java compile, Hadoop commands and some results.
* **countries\_lookup.csv** – lookup file containing a list of countries used for the final task.

Folders:

* **logs** – contains the latest Hadoop log files from the execution of each job from the cluster.
* **out** – contains the latest output (result) files of the Hadoop jobs of each job from the cluster and xlsx file containing the result tables with data and graphs.
* **src\_xxx** – folders containing the java source codes (\*.java) for each task.

**Explanation of the approaches taken**

1. **Preparation**

Before starting with part A, I decided to first explore the data format, see how to extract the individual fields and evaluate, test the solution. I used the tail and head commands of Hadoop -fs from command line to see some of the rows, however the only way to go through the whole file and extract statistics was by creating a map-reduce job. The source code for it, is inside the src\_prep folder and the log and output files are with post-fix “prep”.

Based on observation and nature of the data (date, id, text) I found some rules, which can be used to determine if a row from the file is a good example of a row or not. The rules explanation is found in the source code of the mapper.

This job helped me determine these rules and see how many good and bad rows there are. I separated all scenarios into 5 groups:

* **A) Good row** – the row is perfect; all 4 columns exist and are with the expected size and doesn’t contain delimiters in the tweet message.
* **B) Invalid row** – the row has 4 columns, but some of them doesn’t have the expected size – we can’t be 100% sure that the first field contains date information.
* **C) Good row - more than 4** – all fields in the row have the expected format, except that the tweet message column contains delimiter(s). This is OK. The message still can be used, just need to be careful when extracting the whole tweet message.
* **D) More than 4** – the row contains more than 4 fields (more than 3 delimiters) and some of the columns doesn’t obey the rules – we can’t be 100% sure that the first field contains date information.
* **E) Bad row** – all other scenarios. The row contains less than 3 delimiters. These seems to be rows of tweet messages, with EOL (LF) character in the message, which makes it be processed by Hadoop as new line. There is an option in Hadoop do define a custom class for splitting the data by custom rule and not by LF, but I haven’t implemented it as the number of such rows was very small (0.2%) compared with the volume of good ones.

Based on this analysis, I included these rules in all consequent map-reduce jobs and choose to process only the rows from scenarios **A** and **C**, which in total gives **25 697 010** rows.

To make it easier, I kept the names of the source code files the same, but the content is different in each folder.

Results:

bad row 51 093

**good row 25 568 510**

invalid row 39

**good row - more than 4 128 500**

more than 4 2

1. **Part A**

I split this part into 2 sub tasks A1 and A2. A1 is handling the histogram, A2 is handling the average.

**A1**The approach I took was creating a standard map-reduce job with mapper having as key the number of characters (length) of the Tweet message (IntWritable) and as Value the number of encounters (count). The reducer is simply summing the counts per key and outputs the key and count. Because of this format the reducer is used also as combiner to make the whole job faster and more effective.

For the key part, which is aggregating the length of the messages into blocks of 5 I used the round function floor which is rounding to the lower number. In this way by dividing by 5 and rounding and after that multiplying by 5 will round the size of the message to a number exactly dividable by 5.

I decided to display the messages with a bit higher amount of data and they are not so much higher and not so many.

Notice, that the total matches the total from the prep job.

Results:

|  |  |
| --- | --- |
| **length of the message** | **number of messages** |
| 5 | 11,201 |
| 10 | 7,637 |
| 15 | 26,036 |
| 20 | 73,501 |
| 25 | 182,063 |
| 30 | 309,278 |
| 35 | 346,076 |
| 40 | 387,393 |
| 45 | 429,443 |
| 50 | 482,292 |
| 55 | 497,922 |
| 60 | 588,375 |
| 65 | 611,611 |
| 70 | 710,539 |
| 75 | 685,448 |
| 80 | 748,172 |
| 85 | 810,812 |
| 90 | 877,505 |
| 95 | 921,694 |
| 100 | 925,990 |
| 105 | 920,282 |
| 110 | 963,916 |
| 115 | 1,028,821 |
| 120 | 1,081,345 |
| 125 | 1,188,805 |
| 130 | 1,345,724 |
| 135 | 2,039,265 |
| 140 | 7,345,671 |
| 145 | 125,595 |
| 150 | 15,309 |
| 155 | 2,954 |
| 160 | 1,795 |
| 165 | 763 |
| 170 | 511 |
| 175 | 837 |
| 180 | 422 |
| 185 | 212 |
| 190 | 651 |
| 195 | 246 |
| 200 | 94 |
| 205 | 71 |
| 210 | 121 |
| 215 | 74 |
| 220 | 66 |
| 225 | 56 |
| 230 | 37 |
| 235 | 36 |
| 240 | 43 |
| 245 | 65 |
| 250 | 44 |
| 255 | 91 |
| 260 | 67 |
| 265 | 28 |
| 270 | 5 |
| **Total** | **25,697,010** |

**A2**  
The approach I took for this task was using the IntIntPair custom class as value for the mapper and used Text as key. The key contains just a static text and the IntIntPair will contain the actual size of the message with the count (1). The actual sum and calculation is done inside the reducer. The average is calculated by summing the counts of all lengths and dividing it by the summing of the lengths of all messages.

To make the output more descriptive I choose the key to be “Results:” and added some text in the reducer emit procedure.

The reason to separate this task from the first was, that because the first task is aggregating the length of the messages to 5, when I do an average it will not be accurate using this rounded values (it will be smaller than the real one). In order to get an exact average we have to use the exact length of the messages.

As a validation notice that the total amount of messages is the same as total from A1 task.

Results:

|  |  |
| --- | --- |
| Results |  |
| Total Tweet Messages length | 2,817,785,678 |
| Total Number of Tweet Messages | 25,697,010 |
| **Average Length of Tweet Messages** | **109** |

1. **Part B**

The approach I used to this task was similar to the approach of task A1, with the difference that for this task the key is of type Text. In the mapper I extract and convert the epoch\_time value to text with UK date format “dd/MM/yyyy” as this is good enough to be used as key for a day. I use the SimpleDateFormat class to transform the date to string.

The reducer is simply summing the counts and is also used as combiner.

You can see the diagram and the data in the xlsx file.

The diagram correctly shows that the peaks of the tweets were during the opening ceremony (05-06/Aug) and all the way until the closing ceremony on 21/Aug.

And again as validation, the total of messages matches the totals from part A.

Results:

|  |  |
| --- | --- |
| **Tweet Date** | **Number Of Messages** |
| 25/07/2016 | 30,254 |
| 26/07/2016 | 68,687 |
| 27/07/2016 | 65,311 |
| 28/07/2016 | 57,263 |
| 29/07/2016 | 59,966 |
| 30/07/2016 | 54,776 |
| 31/07/2016 | 54,618 |
| 01/08/2016 | 85,209 |
| 02/08/2016 | 140,093 |
| 03/08/2016 | 306,930 |
| 04/08/2016 | 381,688 |
| 05/08/2016 | 706,755 |
| 06/08/2016 | 1,888,120 |
| 07/08/2016 | 1,281,008 |
| 08/08/2016 | 1,475,631 |
| 09/08/2016 | 1,309,407 |
| 10/08/2016 | 1,361,399 |
| 11/08/2016 | 1,231,989 |
| 12/08/2016 | 1,405,747 |
| 13/08/2016 | 1,326,582 |
| 14/08/2016 | 1,309,606 |
| 15/08/2016 | 1,308,216 |
| 16/08/2016 | 1,358,456 |
| 17/08/2016 | 1,292,562 |
| 18/08/2016 | 1,254,470 |
| 19/08/2016 | 1,468,455 |
| 20/08/2016 | 1,212,817 |
| 21/08/2016 | 1,384,620 |
| 22/08/2016 | 869,140 |
| 23/08/2016 | 183,314 |
| 24/08/2016 | 96,168 |
| 25/08/2016 | 62,892 |
| 26/08/2016 | 51,510 |
| 27/08/2016 | 31,312 |
| 28/08/2016 | 33,056 |
| 29/08/2016 | 31,999 |
| 30/08/2016 | 28,373 |
| 31/08/2016 | 28,674 |
| 01/09/2016 | 24,277 |
| 02/09/2016 | 23,411 |
| 03/09/2016 | 19,453 |
| 04/09/2016 | 22,722 |
| 05/09/2016 | 24,109 |
| 06/09/2016 | 28,719 |
| 07/09/2016 | 84,473 |
| 08/09/2016 | 143,419 |
| 09/09/2016 | 29,354 |
| **Total** | **25,697,010** |

1. **Part C**

The output of this task is list of countries and number of encounters: Text and IntWritable.

For this task I used similar IntSumReducer as in the previous jobs, which is just summing-up the number of encounters of a given country, and therefore the it is used as a combiner too.

For this task I also use a replication join using a CSV file as a list of countries with their names in lowercase and in their native language.

The most of the coding and calculation is done inside the mapper. It’s doing the following manipulations:

* Caching the lookup data - countries.
* Extracting hashtags data.
* Filtering and formatting hashtags data.
* Executing Normal, Pre and Post key words matching.

Before coming to conclusion what approach to use, my first version of this job was without a lookup, but rather a job that was just extracting the hashtags and formatting them in order to have some idea of additional pre and post key words. I executed these initial tests on the local Hadoop environment using the example file.

**Caching phase**

This phase is executed once per node in the beginning. To make sure all is good I created also one custom counter to show the total amount of countries read from the CSV file. As the counter is incremented by each mapper this amount becomes high. When executing it on the local, as there is just one mapper it gives the amount of countries: 752. On the cluster as the data is split usually into 44 parts, the counter shows: 33 088.

I decided to use a lookup, as It’s more elegant solution, there are small number of rows and gives an option to update it. The other option is to have this list hardcoded into the code, which is never the best solution. The CSV file is UTF-8 decoded as it contains different languages from all countries. This special property is important as it should be in the code when opening the file (otherwise the text from the file will not match those from the input).

Here is the UTF-8 bit: isr = new InputStreamReader(in, "UTF8");

The file has 5 fields. The idea behind it is to gives us ability to match the hashtag by 4 ways:

* English (latin) name of the country - field 3 first 270 rows.
* Native name of the country - field 3 last 270 rows.
* 2 letters code of the country – field 4.
* 3 letters code of the country – field 5.
* Field 1 is ID of the row.
* Field 2 is the Full name of the Country in English – used as output data.

I use 4 hash table variables to store all these pairs of data:

* countryFullName – contains the ID as key and ID concatenated with Full Name as value (this is to make the value unique because the Full Name is repeated few times in the file – one for English and one for the Native).
* countryName – contains the name of the country (enlglish or native) as key (if there isn’t one puts empty string) and the ID as value.
* countryISO2 – contains the 2 characters country code as key and the ID as value.
* countryISO3 – contains the 3 characters country code as key and the ID as value.

The first hash table is used as an output at the end when the data is emitted to send the full country name. The other 3 are used for matching with the hashtag.

**Extraction phase**

There are few methods of extracting hashtags from a string, from which a tempting one is using regular expressions. However, they can be a bit of a black box for even experienced programmers (at least will take a while for one to understand what a given regular expression is exactly doing). That’s why I decided to create a function which will parse the whole string and extract the hash tags. The name of the function is **GetAllHashTags**. Basically in there I also declare the definition of a hashtag: A hashtag starts with the symbol # and ends with the symbol space (“ “) or beginning of another hashtag (#). This function receives as parameter a string and thus returns a list of hashtags. The list can be of size 0 – if no hashtags are found.

**Filtering and formatting phase**

As we are after hashtags which represent a country, there should not be any punctuation in the beginning or the end of them. For this task, I use a simple regular expression to remove these and format the hashtag and by this also removing the symbol “#” and at the end formatting it to lower case. Now the hashtags are ready to start matching.

**Matching phase**

This is the main part of the solution. Based on my observation of the data I decided to use the following methods of detecting a given hashtag which country is supporting:

* if the hashtag is the name of the country in English: #Bulgaria
* if the hashtag is the name of the country in its native language: #българия
* if the hashtag is the 2 or 3 character code of the country: #bg #brg
* if the hashtag starts with a set of key words and then ends with either: the English name, native name, 2 or 3 character code: #GoBulgaria, #Teamбългария, #WeAreBG, #IloveBGR  
  I found few more key words than the suggested “go” and “team” which are used in the tweets (see the source code).
* if the hashtag starts with either: the English name, native name, 2 or 3 character code and then ends with a set of key words: #BulgariaTeam, #българияTeam, #BGteam, #BGRteam

Benefits:

* it’s doing a controlled matching, with lower chance of mistakenly match of country with another word.
* It’s dynamic as it’s using a lookup file.

Cons:

* As it’s not a fuzzy match, it might miss scenarios where the country word is in the middle of the hashtag.
* About the key words, most of them I found are in English and few in Portuguese and Spanish. By doing this I miss to match a lot other languages, but it’s just a matter of time and effort finding these key words in the other languages. Also this list can be done as a lookup.

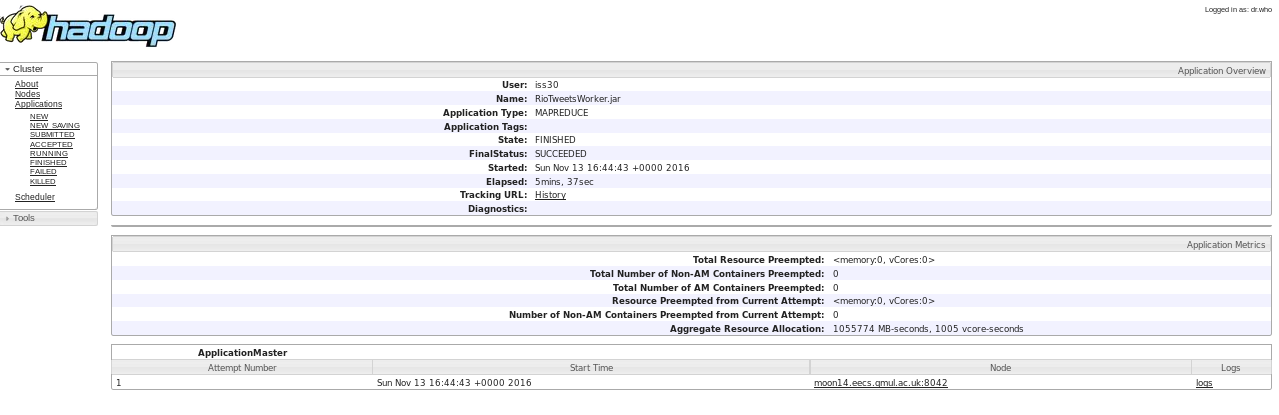
Results:

I had to do few test on the local Hadoop environment. In the source code you will see one block which is commented as one of my tests. This is in order to test my code and see what is missing and what is picked up. In the out folder you’ll find 3 txt files (bdp\_course\_work\_c …) with results.

* The USA one is a test one just printing all hashtags picked up as USA.
* The v1 is the first version of the results – there were some duplicated countries.
* The final one is the one I used for the results.

The total number of hashtags decoded as supporting some country is **5.7** million. This is about **23%** compared with the number of rows.

Execution duration of the job: 5min 37sec. (during rush hour)



The stats:

|  |  |
| --- | --- |
| **Country** | **Number Of Hashtags** |
| United States | 870,261 |
| Brazil | 816,313 |
| Spain | 394,759 |
| Argentina | 363,033 |
| Great Britain | 326,513 |
| Canada | 230,042 |
| Italy | 219,957 |
| Mexico | 190,169 |
| India | 174,467 |
| France | 165,074 |
| China | 147,381 |
| Colombia | 138,564 |
| Japan | 124,851 |
| Serbia | 117,488 |
| Australia | 116,476 |
| Venezuela | 114,036 |
| Jamaica | 111,245 |
| Russian Federation | 88,057 |
| Netherlands | 67,075 |
| Kenya | 54,440 |
| Refugee Olympic Team | 49,266 |
| Poland | 40,863 |
| Sweden | 40,626 |
| Thailand | 40,304 |
| Turkey | 39,072 |
| Republic of Korea | 34,746 |
| Belgium | 32,958 |
| Ireland | 28,174 |
| New Zealand | 28,072 |
| Ecuador | 24,446 |
| Egypt | 24,016 |
| Malaysia | 23,191 |
| Cuba | 21,199 |
| Jordan | 18,054 |
| Nigeria | 17,337 |
| Ethiopia | 13,345 |
| Hungary | 12,961 |
| Honduras | 12,590 |
| Israel | 12,101 |
| Ukraine | 11,912 |
| Lithuania | 11,547 |
| Austria | 11,546 |
| Puerto Rico | 10,545 |
| Côte d'Ivoire | 10,375 |
| Germany | 10,092 |
| Panama | 10,020 |
| Iraq | 9,678 |
| Bahrain | 9,021 |
| Saudi Arabia | 9,016 |
| Tonga | 8,708 |
| Greece | 8,316 |
| Fiji | 8,218 |
| Tunisia | 7,914 |
| Andorra | 7,783 |
| Czech Republic | 7,690 |
| Paraguay | 7,607 |
| Peru | 7,568 |
| Trinidad and Tobago | 7,517 |
| Armenia | 7,486 |
| Dominican Republic | 7,369 |
| Ghana | 7,072 |
| Romania | 6,924 |
| Cameroon | 6,493 |
| Montenegro | 6,014 |
| Uzbekistan | 5,921 |
| Kazakhstan | 5,267 |
| Qatar | 5,241 |
| Iran | 5,129 |
| Indonesia | 4,950 |
| Chile | 4,423 |
| Azerbaijan | 4,083 |
| Portugal | 3,997 |
| Senegal | 3,949 |
| Belarus | 3,885 |
| Hong Kong (China) | 3,870 |
| Norway | 3,766 |
| Uganda | 3,520 |
| Burundi | 3,438 |
| Singapore | 3,259 |
| Algeria | 3,189 |
| Georgia | 2,945 |
| South Africa | 2,877 |
| Bulgaria | 2,550 |
| Croatia | 2,530 |
| Morocco | 2,464 |
| British Indian Ocean Territory | 2,346 |
| Bolivia | 2,125 |
| Finland | 2,073 |
| Philippines | 2,058 |
| Pakistan | 2,045 |
| Estonia | 1,896 |
| Grenada | 1,828 |
| Macao (China) | 1,748 |
| DPR Korea | 1,738 |
| Slovakia | 1,698 |
| Afghanistan | 1,645 |
| Mongolia | 1,579 |
| Tuvalu | 1,469 |
| Botswana | 1,454 |
| Bahamas | 1,443 |
| Zimbabwe | 1,415 |
| Sudan | 1,413 |
| Nepal | 1,366 |
| Kuwait | 1,363 |
| Haiti | 1,344 |
| Denmark | 1,338 |
| Cyprus | 1,247 |
| Switzerland | 1,211 |
| Namibia | 1,101 |
| Somalia | 1,065 |
| Eritrea | 1,060 |
| Lebanon | 1,034 |
| Uruguay | 1,018 |
| South Sudan | 985 |
| Maldives | 965 |
| Brunei | 934 |
| Sri Lanka | 928 |
| Guatemala | 905 |
| Rwanda | 889 |
| Kyrgyzstan | 881 |
| Oman | 880 |
| Libya | 852 |
| Mali | 847 |
| Angola | 846 |
| South Georgia and the South Sandwich Islands | 814 |
| Kiribati | 807 |
| DR Congo | 800 |
| Gabon | 746 |
| Bosnia and Herzegovina | 738 |
| Albania | 720 |
| Niger | 694 |
| Syria | 662 |
| Luxembourg | 647 |
| Costa Rica | 597 |
| Vietnam | 589 |
| Anguilla | 585 |
| Barbados | 572 |
| Myanmar | 545 |
| Moldova | 545 |
| Jersey | 527 |
| Yemen | 526 |
| Djibouti | 488 |
| Netherlands Antilles | 485 |
| Bermuda | 474 |
| Latvia | 446 |
| Benin | 445 |
| Northern Mariana Islands | 425 |
| Suriname | 399 |
| Macedonia | 382 |
| Nicaragua | 379 |
| El Salvador | 371 |
| Guyana | 370 |
| Guadeloupe | 365 |
| Iceland | 362 |
| Monaco | 350 |
| Micronesia (FS) | 350 |
| Saint Barthélemy | 347 |
| Papua New Guinea | 342 |
| Slovenia | 341 |
| Taiwan (ROC) | 330 |
| Tajikistan | 317 |
| Samoa | 315 |
| Sierra Leone | 314 |
| Laos | 306 |
| Cape Verde | 293 |
| Liberia | 289 |
| Vatican City | 265 |
| Nauru | 263 |
| Mozambique | 258 |
| Martinique | 249 |
| Saint Lucia | 249 |
| Belize | 241 |
| Togo | 226 |
| Marshall Islands | 216 |
| Malta | 213 |
| Palau | 211 |
| Guam | 210 |
| Western Sahara | 210 |
| Antigua and Barbuda | 199 |
| Tanzania | 199 |
| Bhutan | 191 |
| Lesotho | 191 |
| Gambia | 180 |
| Bangladesh | 170 |
| Central African Republic | 169 |
| Chad | 169 |
| Guinea | 166 |
| Malawi | 165 |
| San Marino | 163 |
| Dominica | 160 |
| American Samoa | 158 |
| Cook Islands | 154 |
| Comoros | 152 |
| Liechtenstein | 143 |
| Turkmenistan | 142 |
| Zambia | 135 |
| Cambodia | 132 |
| Aruba | 121 |
| Réunion | 119 |
| Congo | 115 |
| Mauritius | 113 |
| Aland Islands | 110 |
| Saint Pierre and Miquelon | 102 |
| Palestinian Territory, Occupied | 98 |
| Madagascar | 95 |
| Timor-Leste | 93 |
| Saint Vincent and Grenadines | 91 |
| Kosovo | 88 |
| The Bahamas | 88 |
| Montserrat | 80 |
| Swaziland | 75 |
| Burkina Faso | 74 |
| Christmas Island | 71 |
| Seychelles | 71 |
| Vanuatu | 66 |
| Guinea-Bissau | 63 |
| Guernsey | 61 |
| Mauritania | 54 |
| Isle of Man | 51 |
| Cayman Islands | 46 |
| French Guiana | 40 |
| US Virgin Islands | 40 |
| Gibraltar | 40 |
| Heard Island and Mcdonald Islands | 39 |
| Equatorial Guinea | 36 |
| The Gambia | 36 |
| British Virgin Islands | 31 |
| New Caledonia | 31 |
| Saint Helena | 31 |
| Cocos Islands | 29 |
| Sao Tome and Principe | 28 |
| Antarctica | 27 |
| United Arab Emirates | 26 |
| Turks and Caicos Islands | 23 |
| Niue | 22 |
| Faroe Islands | 21 |
| Mayotte | 20 |
| Svalbard | 20 |
| Tokelau | 18 |
| Solomon Islands | 17 |
| French Polynesia | 16 |
| US Minor Outlying Islands | 13 |
| East Timor | 12 |
| French Southern Territories | 9 |
| Greenland | 9 |
| Saint Kitts and Nevis | 7 |
| Sahrawi Arab Democratic Republic | 6 |
| Saint Martin | 6 |
| Norfolk Island | 4 |
| Falkland Islands | 3 |
| Pitcairn Islands | 3 |
| Wallis and Futuna Islands | 3 |
| Curaçao | 1 |
| **Total** | **5,723,198** |

**In conclusion**

The country with most support was **USA**, but close to it is **Brasil**. If adding more keywords its possible that Brazil will overcome USA in the results.

One of the obvious questions is why there are 253 countries detected if there were just 208 countries in the Olympics present?

There were hashtags supporting countries, which were not officially on the list of countries (at least on the Olympics website). Like Macao (China). There are hashtags gomacao, goma, etc.

But on the other side there are hashtags of other words which gets decoded as countries. Like the 3 character code of “British Indian Ocean Territory” is “iot”, but IOT also can mean Internet Of Things.

I did a check and there are total of 47 countries from this list which are not present on the official Olympics list. This means 253 – 47 = **205 countries detected**. Just 3 countries short.

Map/Reduce is a powerful technique and solution which makes such analysis very easy, effective and very close to the reality. Most of the time I had to spend with cleaning and formatting the input data, lookup data and keywords. The actual coding and execution of the job were the less time consuming. I think the approach I used is a good balance between time invested in cleaning data against receiving good enough approximation of the result.

**Sources used**

* List of countries and codes – <http://www.nationsonline.org/oneworld/country_code_list.htm>
* List of countries in their native language – <https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_and_their_capitals_in_native_languages>
* List of countries in the Olympics - <https://www.rio2016.com/en/countries#favorite-countries>