Instruction Manual

This is an instruction manual on the creation of a CSV containing all race information as provided by the FISA in PDF’s on their site [www.worldrowing.com](http://www.worldrowing.com), using the programs in the ‘src’ folder in ‘celeste’ on git. Each program is described in the sections below. fisa\_scraper.py and sheet\_divider.py should be ran separately, the rest of the programs will be ran automatically when running main.py. The time line of running the programs is as follows:

1. Run fisa\_scraper.py
2. Transform PDF’s to Excels using Adobe Acrobat Pro (further described in section Scraper)
3. Run sheet\_divider.py
4. Run main.py

The eventual product generated by main is a data frame consisting of the information from the FISA files and some additional features that are generated using this information. This data frame is saved as a csv in '../data/Racedata/total\_file\_all\_feat.csv'.

It takes some time to run the programs, I would say at least half an hour, take this into account when using them.

All programs should be situated in ‘src’, which can be found in the directory ‘celeste’ on github. In the same folder there should be a directory called ‘data’ with some content. This will all be present if you clone the ‘celeste’ folder from Git.

The programs use Python version 2.7 and all used libraries and their versions are listed below:

Libraries:

|  |  |
| --- | --- |
| **Library** | **Version** |
| pandas | 0.19.2 |
| xlrd | 1.0.0 |
| re | 2.2.1 |
| numpy | 1.11.3 |
| urllib | 1.17 |

# Scraper

The scraper (fisa\_scraper.py) is the part of the program that retrieves all data about the races from the world rowing site. The data comes in the form of Results files and GPS files, and the program checks the existence of both, and retrieves them if they exist. To be able to do this, it tries out urls consisting of a combination of codes for the contests, years, boat types, and round names. I created lists that contain each of these codes, and other lists that contain the corresponding meaning, which is used to create the file name for the PDF.

An example:

The URL corresponding to the Results file of the Light male Skiff in the A-Finals rowing in the World Cup 3 in 2013:

<http://www.worldrowing.com/assets/pdfs/WCp3F_2013/ROM112101_C73.pdf>

The name of the file that is created in the ‘Results’ folder:

2013\_WCp3\_LM1X\_FA.pdf

Make sure that the following folders are present before you start:

* The ‘src’ folder containing the scraper
* The ‘data’ folder containing the ‘Scrapedata’ folder
* A ‘Results’ folder and a ‘GPS’ folder in the ‘Scrapedata’ folder

This scraper does not generate the Excel files. With Adobe Acrobat Pro they can be converted to excel files. Currently the practice is to create one excel file per year + competition pair per file type (where file type is either Results or GPS) where each sheet represents a race. The file name should be of the form: ‘year competition\_abbriviation filetype’ and all files should be in the folder ‘Excel\_from\_PDF’ without a division in subfolders. An Example is: ‘2013 ECH GPS.xlsx’. The competition abbreviations are: ECH (European Championships), WCH (World Championships), WC1/2/3 (World Cup 1/2/3), OS (Olympic Games). On the next page I will describe the program that is used to generate one separate excel file for each sheet.

Currently files from 2013-2016 are already present in the ‘Excel\_from\_PDF’ folder, these do not have to be generated again.

If a new year needs to be added to the scraper, you can just add it to the ‘years’ list on line 13 in the file fisa\_scraper.py

Some years have exceptions:

* The World Cup 3 code in 2013 and 2014 is WCp3F instead of WCp3
* In 2017 an ‘\_1’ is added behind the year in the URL

For each new contest and year one needs to check the difference between the URL on the world rowing site (when clicking on the Results or GP PDF in the Results section of the website) and the URL that is created by the scraper. If there is a constant difference, this should be added as an exception, like I did for the two exceptions mentioned above.

To be more complete, one can add Paralympic rowing. These files are not considered now.

# Sheet divider

The sheet divider (sheet\_divider.py) splits each excel file containing one sheet for each race in separate excel files for each race. It obtains these files from ‘data/Excel\_from\_PDF’ and saves the resulting files per year in the ‘data/Racedata’. For this to work, there should be a folder for each year in the ‘Racedata’ directory. These are already present when cloning for the years 2013-2016. The filenames are constructed of the year, competition, boat type, race round and file type. An example: 2013 - ECH - M1x - FA – Results.xlsx

It tries every year+competition pair in combination with both file types (Results and GPS) as a filename in the Excel\_from\_PDF directory. If such a combination exists, it transforms the excel file to a dataframe. If it does not exist, it will output the name of the file and the message ‘is no valid filename’.

It checks the location of ‘World Best Time’, to determine whether it is dealing with a normal file layout or a corrupt lay out. If the lay out is corrupt, the information on the file type, boat description, boat type, date, round and race are situated next to each other instead of beneath each other (which is usually the case). This is corrected to the normal lay out, which makes the layout of all output files the same.

To be able to read the files correctly, the Semi-finales are transformed in S1-6. Where S1 is SAB 1 and S6 is SEF 2, and the Quarter Finals are transformed to Q1-4.

# Main

Main is the program that runs all files that form the transition from the separate unstructured excel files to a data frame containing both GPS and Results data. The programs called by main are:

* retrieve\_files.py
* data\_preprocessing\_race.py
* dataframe\_combiner.py
* feature\_creation\_GPS.py
* feature\_creation\_results.py
* feature\_creation\_combined.py

## Functions:

Several functions are created in main, forming the pipeline of the program:

The first is read\_input(argv), which reads the values of -g (gps\_from\_raw) and -r (results\_from\_raw) which define whether the GPS files should be created out of the raw xls files. If the xls files have already been processed before, both should be false. If only one of the two has been processed before, the one that has not been processed should be true and the one that has been processed should be false. This is done because the processing takes a lot of time, therefore it should be skipped when possible.

The second is retrieve\_files(), which creates dictionaries containing the paths of all GPS excel files and of all Results excel files.

The third is make\_data\_GPS(file\_dict\_GPS), which preprocesses all GPS files, using the generated dictionary of the GPS paths, and saves separate speeds and strokes files. The speeds file consists of one column representing the country and the other columns representing the distance and the measured speed in meter per second at that distance. The same goes for strokes, only the measured unit is strokes per minute. Since this process takes a long time, a flag is generated. If the files have already been processed and no new excel files are added, gps\_from\_raw should be False. Example: python main.py -g False.

The fourth is make\_data\_results(file\_dict\_results) ), which preprocesses all Results files, using the generated dictionary of the Results paths. This preprocessing consists of finding the information (for example times, ranks and crew member names) and structuring it in dataframes. The excel files are the csv representation of the dataframes. Since this process takes a long time, a flag is generated. If the files have already been processed and no new excel files are added, results\_from\_raw should be False, which can be done by adding -r False as an argument in the command line. Example: python main.py -r False. Or in combination with the GPS files: python main.py -g False -r False.

create\_all\_df() calls both make\_dfs\_speeds\_strokes(), which creates two big dataframe from all preprocessed GPS files (one for speeds one for strokerate), and make\_df\_results(), which creates one big dataframe from all preprocessed Results files. There is a loop in both these functions looping over the numbers in the range(3,7), which means: 3,4,5,6. If 2017 will be included, this range should be changed to range(3,8). If the dataframe containing all information is already present in the data folder (under the name: total\_file\_all.csv) it will use this csv file to load the data frame, instead of constructing it again. This considering the time it takes to construct it (several minutes with 8GB RAM).

create\_feature\_df(df\_all) generates features out of the information contained by the big data frame and adds this information to the dataframe. Three files with functions creating features have been generated to keep some overview. The first creates features using only speeds and stroke rate data (feature\_creation\_GPS.py), the second creates features using only data originating from the Results files (times/ranks/….) (feature\_creation\_results.py) and the third uses a combination of the data to generate features (feature\_creation\_combined.py).

**The eventual product generated by main is a data frame consisting of the information from the FISA files and some additional features that are generated using this information. This data frame is saved as a csv in '../data/Racedata/total\_file\_all\_feat.csv'.**

## Functionality of imported files:

### Retrieve files (retrieve\_files.py)

This program retrieves all xls filenames generated by sheet\_divider.py and saves the paths to these files in two dicts; one for the GPS files and one for the Results files. The key of the dict is the file name and the content is the path+filename, who will be used later on when all files are processed. This is the function get\_xls\_files()

Later on, when speed and stroke csv’s are generated per race, these filename paths are also saved in a dictionary. This is done in get\_csv\_files()

### Preprocess race (data\_preprocessing\_race.py)

Used in make\_data\_GPS(file\_dict\_GPS), make\_data\_results(file\_dict\_results), make\_dfs\_speeds\_strokes() and make\_df\_results(). And contains the functions: read\_raw\_GPS(), read\_raw\_results(), read\_csv\_GPS(), read\_csv\_results(). It is extended by data\_preparation\_race.py, where the real cleaning up is performed. The dictionary of filenames should be passed to this class.

#### read\_raw\_GPS()

This function transforms the GPS excel files to separate speeds and strokes files. prep\_raw\_GPS(), situated in data\_preperation\_race.py, searches the location of the table in which the recorded speeds and strokerates are situated, extracts the speeds and stroke rates and generates two files per input file, one for the speeds and one for the strokes. Each row is a country racing in the race that is processed and the columns (except for the first two columns which are the index and the country) represent a measure point (for example 50m). These files are saved in ‘../data/Racedata/year/Speeds’ and ‘../data/Racedata/year/Strokes’, where year stands for the year in which the race was performed.

#### read\_raw\_results()

This function transforms the unstructured Results excel files to new results excel files where all information is structured. To make this possible, every piece of information needs to be sought, for their location is not fixed in certain cells. The date needs to be obtained from the rows that are outside the ‘table’, where the ‘table’ is the part of the excel file representing the table that is shown in the PDF, and therefore the format of each row is checked until a date is found and this date is saved. The table is characterized by having headers, of which the first is ‘Rank’. If ‘Rank’ is situated on the first place in the row, the beginning of the table is found. The data frame is cropped to start at the beginning of the table, disregarding the lines above, which do not contain necessary information. The next time a row starts with a string, which is not ‘Rank’, is marked as the end of the table, cropping data frame to exclude the following rows. The table is highly unstructured and values are almost never in the same column as their header. Therefore two functions search for the names, ranks and lanes (create\_names\_ranks\_lanes\_df(not\_nan\_df, col\_names)) the times, interval times and rankings on each of the 500m (create\_time\_betweenranks\_df(not\_nan\_df, col\_names)) and put them in columns with informative headers. These two data frames are concatenated and the result is one structured data frame, which is saved in ‘../data/Racedata/year/Results’ in a csv-format, where the year is the year in which the race took place.

#### read\_csv\_GPS()

This function filters the speeds and strokes files that are created using read\_raw\_GPS() adds the information contained in the title of the file and concatenates all data frames. This leads to one data frame containing either speed or stroke rate measurements, as well as all information on the year, contest type, boat type and round. One row represents one boat in one race. To reach this final data frame, first the year, contest type, boat type and round are obtained from the title of the file, which looks like this: ‘2013 - ECH - LM1x - FA - GPS – speeds’, and are added to an new data frame. Afterwards the speeds or stroke rates are pasted in this data frame. Sometimes a missing dot causes the values of the speeds or the stroke rates to be 10 times higher than expected, and therefore I built in a check to reduce these values to their normal proportions. Some times strange values are obtained for the speed measurements of the first 500m; null for the first 5 measurements, which are not indicatory for the true speed. These races are disregarded. Since there are a large number of irregularities, differences between two 50 meter points of respectively 1 m/s or 4 strokes per meter, these are replaced by an average of the six surrounding points. This gets rid of some of the noise in the data. If a graph has more than 9 of these irregularities, the race is not considered, and is not added to the data frame. The resulting data frame is concatenated to the previously generated speed/stroke rate data frames, which results in one data frame representing all information on speeds and stroke rates. This is saved in either '../data/Racedata/total\_file\_speeds.csv' or '../data/Racedata/total\_file\_strokes.csv'.

#### read\_csv\_results()

This function collects the information in the title of the files that are generated in read\_raw\_results() and creates a data frame containing both the information on the year, contest type, boat type and round and the information that is in the saved data frame. This data frame gets concatenated to the data frames representing the other races and form one data frame with all Results data. This data frame is saved in '../data/Racedata/total\_file\_results.csv'

### Combining data frames (dataframe\_combiner.py)

In this function the stroke rates, speeds and results data frames are combined in one big data frame. This is possible because they have columns with corresponding information; year, contest type, boat type, round and country. This information is unique for each row, and is therefore suitable to merge on. First a suffix is added to the distances of the measurements of the speeds and stroke rates, to be able to distinguish between them. The speeds columns are called ‘50\_speed, 100\_speed, …. , 2000\_speed’ and the stroke rate columns are called ‘50\_stroke, 100\_stroke, …., 2000\_stroke’. Then the stroke rate and speeds data frames are merged, using the ‘inner’ method, which means that only those rows remain that are present in both data frames. If either the speed information or the stroke rate information is not present, the row is disregarded. This is done to be able to generate all features for all rows (later on). The columns of the results data are reordered, to have a resulting data frame where the identifying columns precede the measurements. Then the speeds+strokerates data frame is merged with the results data, again using the ‘inner’ method, and the combined data frame is saved in '../data/Racedata/total\_file\_all.csv'.

### Feature creation GPS (feature\_creation\_GPS.py)

All features that can be created using only the data from the GPS files are generated in this class, and are listed below:

* sex of team
* weight class of team
* strokes smoothing
* feature marking the difference in either speed or strokes per 500m
* the moment at which a rower starts to sprint
* the average stroke rate of a team in the race
* the point at which a team usually starts to sprint

A function is present for each of these features, and each of these functions will be explained below.

#### create\_all\_features()

This function calls all feature creating functions and returns the data frame containing the original data (of both GPS and Results data) and the created features.

#### create\_boatsize\_sex\_weight\_feature()

Creates a feature for the sex of the boat and the weight class of the boat, called resprectively 'sex' and ‘weight\_cat’ . Both are booleans, where 'Men' and 'Heavy' are represented by a 1 and 'Women' and ‘Light' are represented by a 0. It adds these features to the data frame containing all data.

#### smooth\_strokes()

Loops through the measurements of each row and if it finds an resolution irregularity (value before and after current value are the same) it takes the average of the current value and this reoccurring value and replaces the current value by it. This smoothing happens in exp\_smoothing.py, which is called in this function (smooth.smooth\_plot\_strokes\_list(strokes\_list, name, plot\_indicator)) where stroke\_list is a list of the stroke measurements of the considered row (one boat in one race), name is a combination of year, country, contest, round and boattype and is used when save the plot, and plot\_indicator is a Boolean that can be set to true if you want to plot the original stroke rate and the smoothed stroke rate in one plot to see the difference. The function replaces the original measurements by the smoothed measurements. If you don’t want this, the function can be commented out in create\_all\_features().

#### create\_dif\_feature()

This function generates four features marking the difference in each of the four 500m parts of the 2000m race between the first and the last measurement of both the speed or the stroke rate. The difference is calculated and saved in the function calculate\_difference(). These features are stored directly in the data frame containing all data. The column name of each of the features is: dif\_ + section + measurement type. So for example: dif\_50-500m\_stroke.

#### create\_sprint\_feature()

This feature represents the point at which a boat starts sprinting, marked by a positive difference of more than 1.5 in stroke rate between successive measurements. This function loops over all rows and calls the function determine\_sprint\_point(difference\_list\_strokes), where the difference\_list\_strokes is a list of the differences in stroke rate between consecutive measurements. This function determines where the sprint starts and whether this truly is a sprint or if it is an irregularity. An irregularity as defined here is if the stroke rate difference following or preceding the current one is the same as the current one but reversed, marking a little peak instead of a sprint. This check is performed in filter\_irregularities(diffs\_list, index\_list), where diffs\_list is again the list of all differences between successive measurements and index\_list is the list of all differences higher then 1.5 (possibly marking the start of a sprint). If an irregularity is detected, the differences responsible for the irregularity are set to 0 and thereby are no candidate anymore for marking the start of the sprint. If after filtering there are still more than one possible sprint start points left, the highest difference is chosen as the sprint start. When a boat does not sprint, the value of this feature is -1. This feature is added to the data frame with all data under the name sprint\_start.

#### create\_average\_strokepace\_feature()

Creates the average stroke rate in strokes per minute (average\_stroke\_pace), the average stroke rate in strokes per second (average\_stroke\_pace\_per\_second) and the average speed (average\_speed) per boat per race. And next to that the average speed (average\_speed\_team) and variance of the speed (variance\_speed\_team) and average stroke rate (average\_stroke\_team) and variance of the stroke rate (variance\_stroke\_team) per team (meaning the average of the average stroke rates and speed per boat). And finally the 'effect of a stroke' by dividing the speeds by the average stroke per second in the feature called 'speed/stroke'. All these features are added to the data frame.

#### create\_average\_sprint\_feature()

Creates both a Boolean sprint feature stating whether a boat sprints or not (sprint\_bool) and a team based feature where the average sprint point per team and the variance of the sprint point (over all races that team has raced in the current data base) is calculated (average\_sprint\_team, variance\_sprint\_team) and added as a feature to the dataframe.

### Feature creation Results (feature\_creation\_results.py)

All features that can be created using only the data from the Results files are generated in this class, and are listed below:

* A Boolean on when a boat holds the first rank for each 500m point
* The average rank of a team (based on the races in the data)
* The variance in the rank of the team (based on the races in the data)

A function is present for each of these features, and each of these functions will be explained below.

#### create\_all\_features()

This function calls all feature creating functions in the file feature\_creation\_results.py. And returns the input data frame containing all these features

#### first\_position\_feature()

Creates a Boolean that is 1 if the position at the specified measure point is 1 and is 0 otherwise. These features are added directly to the data frame and are named ‘measurepoint\_rank\_first\_bool’, where measurepoint is either 500m, 1000m, 1500m or 2000m.

#### team\_ranking\_feature()

Adds the average rank of a team and the variance of the rank of a team as a feature to each occurrence of the team, based on the races of that team in the data. These are added to the data frame as ‘average\_rank\_team’ and ‘variance\_rank\_team’.

### Feature creation Combined (feature\_creation\_combined.py)

All features that should be created using both the data from the Results files and from the GPS files are generated in this class, and are listed below:

* The times per 50m approximated with the speeds
* The ranks per 50m approximated with the speeds
* The deviation from the median per 50m approximated with the speeds
* How many races there are before and after the current race on the same day
* Features of the opposing teams in the same race
* Total number of strokes performed in current race, based on the average stroke pace and the time

A function is present for each of these features, and each of these functions will be explained below.

#### create\_all\_features(self)

This function calls all feature creating functions in the file feature\_creation\_results.py. And returns the input data frame including all the new features.

#### dif\_median()

This function creates three features: The times per 50m, the ranks per 50m and the deviation from the median crew time all approximated using the speeds, which are measured per 50m. First the median is collected with the function get\_median(race), where race is the data frame containing all information of the currently regarded race. The median time is the time of the middle rank, which is 3 in case of 5 participating crews and the mean of 3 and 4 in the case of 6 participating crews. And the median race is this time divided by the number of measure points (in this case 40 since there is one each 50m and the race is 2000m) and distributed over these 40 measure points, as if the interval time of each 50m is the same. When the median is obtained, the time per 50m is approximated at the function timer\_per\_50\_m(race). First the speeds per 500m are scaled, such that the average speed is 1, for it accounts for exactly 1/(number of measure points) of the time that it took to row that 500m. When a measure point measures a higher speed, that 50m part accounts for a smaller part of the time. Therefore the inverse of the normalized difference from the average speed is added to 1. This leads to a list of scales that can be used to scale the average time per 50m of each 500m with. The function scale\_times(speed\_scale\_per\_500, time\_per\_500) divides the time by the number of measure points, which created a list of 50m\_times all containing 1/(number of measure points) of the time, and scales these times with the previously created scales. These scaled times are the approximation of the time that each 50m took to row. The time\_per\_50\_m(race) function returns a list containing a list of scaled times for each boat. These are transformed to features in a data frame so they can be added to the data frame containing all data. In this process the times per 50m are transformed to the time from the start to that 50m point, summing all previous times. The names of these time approximations are: distancem-time\_approx. An example is: 50m-time\_approx. These approximated times are used to generate ranks per 50m. The lowest time gets the lowest rank, etc. The names are similar to the time approximation names: distancem-rank\_approx. An example: 50m-rank\_approx. The time approximations are also used to calculate the deviation from the time that the median race in compute\_dif\_races(median\_boat\_race, time\_per\_50\_all), where median\_boat\_race is the average time per 50m of the median rank per race and time\_per\_50\_all are the approximated 50m interval times of the boats grouped in lists of races. The difference is calculated by subtracting the median time from the boats time. These are collected in a data frame, where the names of the median deviation features are time\_dif\_median\_distancem. For example: time\_dif\_distance\_50m. Then this data frame is added to the data frame of the race, which means that all new features are now present in this data frame, and this data frame is concatenated to the data frames of races that have already been processed. When all races have been processed, the result is the original data frame plus the new created features.

#### coxswain\_feature()

This feature is a Boolean that states whether a coxswain is present or not. This knowledge is generated from the presence of a plus in the boat type, e.g. M8+. This feature is added directly to the all data data frame

#### same\_day\_feature()

This function creates a feature that states how many races are rowed before and after the current race on the same day. These are generated by grouping the data per team per date. If more than 1 instance is present, it means that the team has rowed more than 1 race on that day. The first has a race after the current race and the second has a race before the current race. In the rare case that three races are rowed on the same day, the first has 0 races before and 2 after, the second has 1 before and 1 after and the third has 2 before and 0 after. A check is performed whether the race after or before the current race is a Repechange, for this is often the race that has to be performed on the same day with another race. The names of the features are: ‘races\_after\_day’, ‘races\_before\_day’, ‘after\_is\_rep’, ‘before\_is\_rep’

#### opposing\_teams()

6x4 + 3 columns are created containing the country ID's of the opponents, the average ranks of the opponents, the average stroke rate of the opponents, the average speed of the opponents and the averages over all opponents of the rank, stroke rate and speed. The data is grouped in races and per race the country, average rank, average stroke rate and average speed of the opposing boats is obtained and saved as features for the currently regarded boat. The opponent features containing the countries are called ‘opponent1-5’. The opponent features containing the average ranks are called ‘opponent1-5\_rank\_list’. The opponent features containing the average stroke rate are called ‘opponent\_avg\_stroke1-5’. The opponent features containing the average speed are called ‘opponent\_avg\_speed1-5’. Per opponent feature type (stroke rate, speed, rank, excluding the country) an average is calculated, to determine the average strength of the teams a boat is opposing. These are saved in the features: ‘average\_stroke\_opponents’, ‘average\_rank\_opponents’, ‘average\_speed\_opponents’.

#### total\_strokes\_feature()

This function adds the total number of strokes performed in the race by the boat to the data frame. For each row the stroke rate in strokes per minute and the time in seconds are obtained. The average stroke rate is calculated and divided by 60 to obtain the stroke rate per second. The time x the average stroke rate per seconds gives the total stroke count, which is saved under the name ‘total\_stroke\_count’.