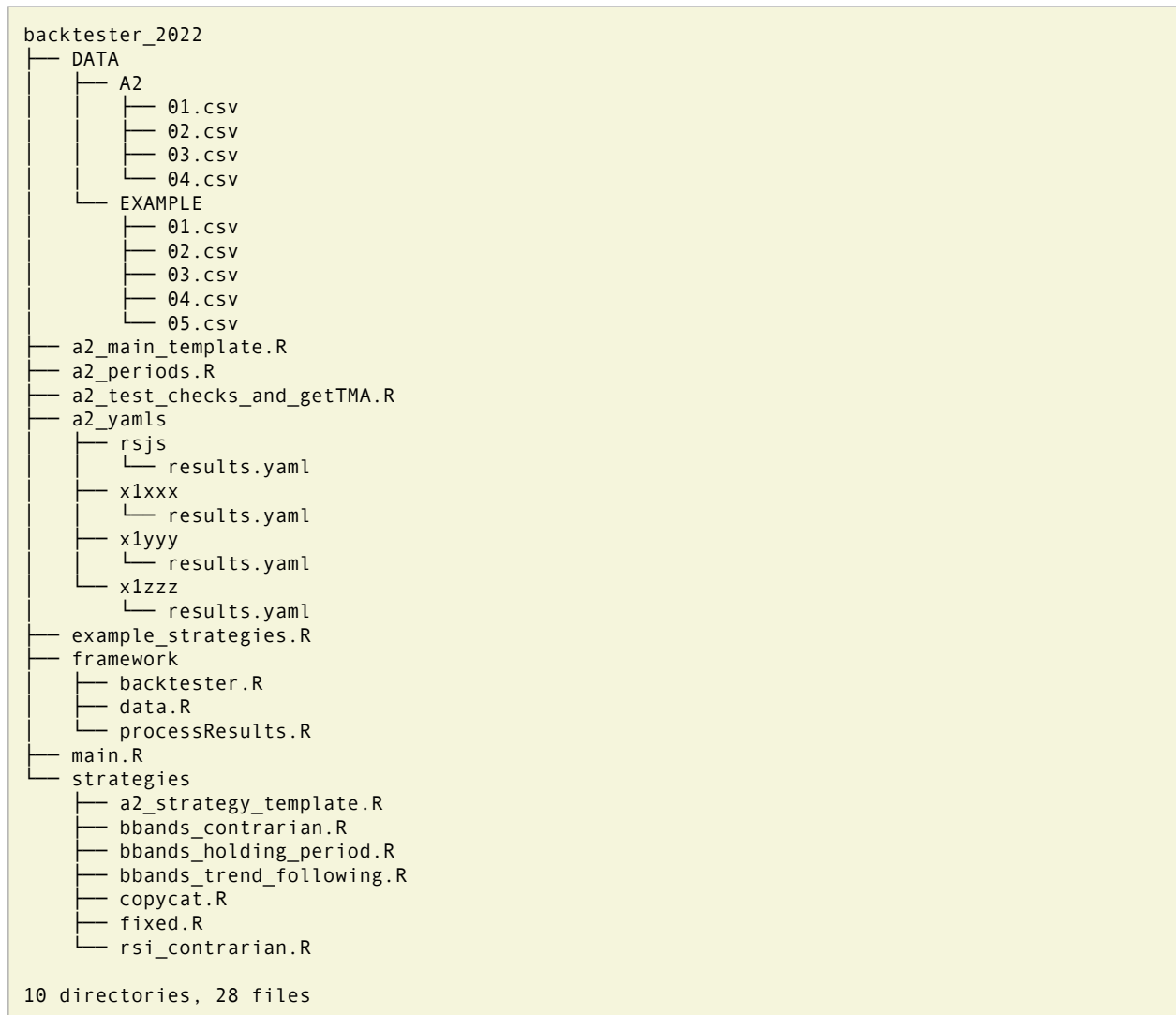


COMP226 Assignment 2: Strategy Development

Continuous Assessment Number	2 (of 2)
Weighting	15%
Assignment Circulated	Thursday 31 March 2022 (week 9)
Deadline	Friday 13 May 2022 (week 12)
Submission Mode	Submit up to two files to the CodeGrade assignment on Canvas: <code>strategy.R</code> (required to get marks) and <code>results.yaml</code> (optional).
Learning Outcomes Assessed	<p>This assignment addresses the following learning outcomes:</p> <ul style="list-style-type: none"> • Understand the spectrum of computer-based trading applications and techniques, from profit-seeking trading strategies to execution algorithms. • Be able to design trading strategies and evaluate critically their historical performance and robustness. • Understand the common pitfalls in developing trading strategies with historical data.
Summary of Assessment	<ul style="list-style-type: none"> • The goal is to implement and optimize a well-defined trading strategy within the <code>backtester_2022</code> framework. • Marks are available for the correct implementation of 10 functions in <code>strategy.R</code> (70%). • Further marks (that require a correct implementation in <code>strategy.R</code>) are available for the results of a cross-validated optimisation that you can include in <code>results.yaml</code> (30%). • CodeGrade pre-deadline tests and offline example outputs are available to help you check the correctness of your work.
Submission necessary to pass module	No
Late Submission Penalty	Standard UoL policy; resubmissions after the deadline may not be considered.
Expected time taken	Roughly 8-12 hours

Before you move on and read more about the assignment and start working on it, please make sure you have worked through "`backtester.pdf`" (and the corresponding slides and video if you want), which is an intro to the `backtester_2022` framework. Only return to this document when you already have the framework up and running.

First, let's recall the contents of the backtester_2022.zip:



In the above listing, the following files/directories are specifically there for assignment 2:

- a2_main_template.R
- a2_periods.R
- a2_test_checks_and_getTMA.R
- strategies/a2_strategy_template.R
- a2_example_yamls
- DATA/A2

The relevance of these files and directories will be explained below. The rest of the document is split into three parts:

- Part 1 describes the 10 functions that should be implemented to fully complete strategy.R; you should start from a2_strategy_template.R;
- Part 2 describes how to create (the optional) results.yaml;
- Part 3 describes submission via CodeGrade and the available pre-deadline tests.

In addition to pre-deadline tests on CodeGrade, **example outputs are provided** (in this document and as files) so that you can test whether you have implemented things correctly.

As for assignment 1, the pre-deadline tests will determine your mark for the first part, corresponding to 70% of the overall marks that are available. Assuming that you have achieved full marks on the first part, the pre-deadline tests will check that the form of `results.yaml` is correct, and that it uses the expected student username (i.e., your one) and corresponding time periods; the pre-deadline tests do **not** check the correctness of the other fields in `results.yaml`, which will be checked post deadline only if you pass the pre-deadline test for `results.yaml`. For those other fields, you should use the examples provided (which are in the subdirectory `a2_example_yamls`).

Part 1: strategy implementation (70%)

The trading strategy that you should implement is a triple moving average (TMA) momentum strategy, which is described in slides 4.7. The specification of the strategy and the functions that it should comprise are given **in full detail**, so the correctness of your code can and will be checked automatically.

Two template files are provided to get you started:

- `strategies/a2_strategy_template.R`, which should become the file `strategy.R` that you eventually submit;
- `a2_main_template.R`, which uses `DATA/A2` and `strategies/a2_strategy_template.R`.

If you source `a2_main_template.R` with no edits to these two files you will get an error:

```
Error in if (store$iter > params$lookbacks$long) { :  
  argument is of length zero
```

This is because the strategy requires a parameter called `lookbacks` that you will need to pass in from `a2_main_template.R`. Read on to see what form this parameter should take, and, more generally, how you should be editing these two files.

`a2_strategy_template.R` contains 10 incomplete functions that you need to complete. The first 6 functions (`checkE01`, ..., `checkE06`) are error checks for the inputs to `getTMA`. These error checks are all one-liners, worth 3% each. They are intentionally meant to be straightforward to implement. The next three functions compute the moving averages (`getTMA`), use them to compute the position sign (`getPosSignFromTMA`), and compute the position size (`getPosSize`). The final, tenth function, `getOrders` combines the last three to implement that actual trading strategy. Recall that every strategy in the backtester framework has a `getOrders` function.

The TMA momentum strategy that you should implement uses three moving averages with different lookbacks (window lengths). The short lookback should be smaller than the medium one, which in turn should be smaller than the long lookback. In every trading period, the strategy will compute the value of these three moving averages (for the series that it trades on, which will be determined by `params$series`). You will achieve this by completing the implementation of the function `getTMA`.

The following table indicates the position that the strategy will take depending on the relative values of the three moving averages (MAs). You will compute this position (sign, but not size) by completing the function `getPosSignFromTMA`. The system is out of the market (i.e., flat) when the relationship between the short MA and the medium MA does not match the relationship between the medium MA and the long MA.

MA		MA		MA	Position
short MA	<	medium MA	<	long MA	short
short MA	>	medium MA	>	long MA	long

The function `getPosSignFromTMA` takes the output of `getTMA` as input. The position size, i.e., the number of units to be long or short, is determined by `getPosSize`. As for all strategies in the backtester framework, the positions are given to the backtester by `getOrders`. Here are the detailed specification and marks available for these 10 functions.

Function name	Input parameters	Expected behaviour	Marks available for a correct implementation
checkE01 ... checkE06	prices; lookbacks.	The behaviour of these checks are specified as comments in the template. Hints are given below.	3% for each of the 6 checks; 18% in total.
getTMA	prices; lookbacks. The specific form that these arguments should take is specified in the template code via the 6 checks that you need to implement.	The function should return a list with three named elements, short, medium, and long. Each element should be equal to the value of a simple moving average with the respective window size as defined by lookbacks. The windows should all end in the same period, the final row of prices.	12%
getPosSignFromTMA	tma_list is a list with three named elements, short, medium, and long. These correspond to the simple moving averages as returned by getTMA.	This function should return either 0, 1, or -1. If the short value of tma_list is less than the medium value, and the medium value is less than the long value, it should return -1 (indicating short). If the short value of tma_list is greater than the medium value, and the medium value is greater than the long value, it should return 1 (indicating long). Otherwise, the return value should be 0 (indicating flat).	10%
getPosSize	current_close: this is the current close for one of the series. constant: this argument should have a default value of 5000.	The function should return (constant divided by current_close) rounded down to the nearest integer.	5%

getOrders	The arguments to this function are always the same for all strategies used in the backtester framework.	This function should implement the strategy outlined below in "Strategy specification".	25%
-----------	---	---	-----

All-or-nothing tests

Since the check functions and getPosSignFromTMA function will only return a small number of possible correct values (2 for the check functions, and 3 for getPosSign), these are implemented as "all-or-nothing" tests where you either get full marks for passing all tests or no marks if you fail at least one test. As a very simple function, getPosSign is also marked with all-or-nothing tests, so from the first 10 functions, partial marks are only available for getTMA and getOrders.

Strategy specification

The strategy should apply the following logic independently to only the series in params\$series (e.g., params\$series could be c(1,3), which would mean trade only on series 1 and 3).

It does nothing until there have been params\$lookbacks\$long-many periods.

In the (params\$lookbacks\$long+1)-th period, and in every period after, the strategy computes three simple moving averages with window lengths equal to:

- params\$lookbacks\$short
- params\$lookbacks\$medium
- params\$lookbacks\$long

The corresponding windows always end in the current period. The strategy should in this period send **market orders** to assume a position (make sure you take into account positions from earlier) according to getPosSignFromTMA and getPosSize. (Limit orders are not required at all, and can be left as all zero.)

Hints

You can develop the first 9 functions without running the backtester.

For the checks you may find the following functions useful:

- The operator ! means not, and can be used to negate a boolean.

- `sapply` allows one to apply a function element-wise to a vector or list (e.g., to `c("short", "medium", "long")`).
- `all` is a function that checks if all elements of a vector are true (for example, it can be used on the result of `sapply`).
- `%in%` can be used to check if an element exists inside a vector.

To compute the moving average in `getTMA` you can use `SMA` from the `TTR` package.

For `getPosSize`, you can use the function `floor`.

For `getOrders` some instructions are given as comments in `a2_strategy_template.R`.

If an error occurs within a function and you would like to inspect the contents of a variable that is local to the function, in addition to printing, you can also use *global assignment* (`<<-`) for debugging.

Example output for `checkE01 ... checkE06` and `getTMA`

The file `a2_test_checks_and_getTMA.R` is provided to give you guidance on how you can test the six functions, `checkE01 ... checkE06`. For each one, two tests are provided: for a correct implementation, one test should produce `TRUE` and the other `FALSE`. (You don't need to use these tests, as you can also just rely on the tests on CodeGrade.)

To use these tests, first source `a2_test_checks_and_getTMA.R` and also source the implementations that you would like to test. The tests that should return `TRUE` are `test_checkE01() ... test_checkE06()`; for tests that should return `FALSE`, there is single function, `test_pass_all_checks`, which takes the function to test as its only argument. Here's an example of both types of test for `E01` (where a correct implementation of `checkE01` has been sourced):

```
> test_checkE01()
[1] TRUE
> test_pass_all_checks(checkE01)
[1] FALSE
```

The way these tests work is clear from the source code in `a2_test_checks_and_getTMA.R`:

```
#####
# Source the functions that you would like to test, e.g., with
# source('strategies/a2_strategy_template.R') or source('strategies/strategy.R')
#####
source('framework/data.R'); dataList <- getData(directory="A2")
prices <- dataList[[1]]
prices_19_rows <- dataList[[1]]$Close[1:19]
prices_20_rows <- dataList[[1]]$Close[1:20]
prices_20_rows_renamed <- prices_20_rows
colnames(prices_20_rows_renamed) <- 'Closed'
bad_prices <- c(1,2,3)
lookbacks_no_names <- list(5,10,25) # list elements not named
lookbacks_not_integer <- list(short=5,medium=as.integer(10),long=as.integer(20))
lookbacks_wrong_order <- list(short=as.integer(15),medium=as.integer(10),long=as.integer(20))
lookbacks <- list(short=as.integer(5),medium=as.integer(10),long=as.integer(20))

test_checkE01 <- function()
```

```

    checkE01(prices,lookbacks_no_names)
test_checkE02 <- function()
  checkE02(prices,lookbacks_not_integer)
test_checkE03 <- function()
  checkE03(prices,lookbacks_wrong_order)
test_checkE04 <- function()
  checkE04(bad_prices,lookbacks)
test_checkE05 <- function()
  checkE05(prices_19_rows,lookbacks)
test_checkE06 <- function()
  checkE06(prices_20_rows_renamed,lookbacks)
test_pass_all_checks <- function(check_func)
  check_func(prices_20_rows,lookbacks)
test_getTMA <- function() # same inputs as test_pass_all_checks()
  getTMA(prices_20_rows,lookbacks)

```

The final test function in this file is for getTMA, where you should get the following return values for a correct implementation:

```

> test_getTMA()
$short
[1] 3081.5

$medium
[1] 3122.5

$long
[1] 3128.875

```

If you want to do further testing, you can use the pre-deadline tests on CodeGrade, which applies to all 10 functions, or you can extend a2_test_checks_and_getTMA.R by adding alternative examples yourself.

Example output for getPosSize

Here is one example input for each of the three possible outputs:

```

> getPosSignFromTMA(list(short=10,medium=20,long=30))
[1] -1
> getPosSignFromTMA(list(short=10,medium=30,long=20))
[1] 0
> getPosSignFromTMA(list(short=30,medium=20,long=10))
[1] 1

```

Example output for getPosSignFromTMA

Here are two examples of correct outputs:

```

> current_close <- 100.5
> getPosSize(current_close,constant=100.5)
[1] 1
  getPosSize(current_close,constant=100.4)
[1] 0

```

Example output for getOrders

The following table gives the correct value of "profit" across 3 different time periods, using the "EXAMPLE" data, and the following parameters:

```
params$lookbacks <- list(short=as.integer(5),
                          medium=as.integer(50),
                          long=as.integer(100))
params$series <- 1:4
```

start period	end period	profit
1	250	2086.184
1	1000	4103.204
500	1500	-2179.298

The examples of results.yaml (details below) can also be used to further establish the correctness of getOrders, along with all the tests on CodeGrade.

Part 2: cross-validation (30%)

Warning

You can only access the final 30% of marks if you get 70% for the first part; otherwise CodeGrade will not process results.yaml.

In this part of the assignment you are asked to do a cross-validated parameter optimization of profit, where you will use an in-sample and out-of-sample time period. Every student has their own in-sample and out-of-sample periods based on their MWS username (only the part before the @, e.g., for Rahul Savani, this username is rsjs, rather than the full email form rsjs@liverpool.ac.uk). By having different time periods for different students, there is not one single correct results.yaml.

To get your in-sample and out-of-sample periods, use a2_periods.R as follows. Source it and run the function getPeriods with your MWS username as per the following example (where we use the fake username "x1xxx"). Use startIn, endIn, startOut, and endOut as the start and end of the in-sample and out-of-sample periods respectively.

```
> source('a2_periods.R')
> getPeriods('x1xxx')
$startIn
[1] 1

$endIn
[1] 884

$startOut
[1] 885

$endOut
[1] 2000
```

You will do two parameter sweeps. One on your in-sample period, and one on your out-of-sample period (normally one doesn't do a sweep on the out-of-sample period in practice; we do it here to allow detailed cross-period performance analysis). The sweep will

be over the following parameters: the short, medium, and long lookbacks, and the subset of series that are traded on.

Parameter	Values
short lookback	5, 10
medium lookback	50, 100
long lookback	200, 300
series	All subsets of 1:4 that have at least two elements

The correct resulting number of parameter combinations is 88.

Hint

You can use `expand.grid` to create the relevant parameter combinations; alternatively you could use nested for loops.

The following information (full example below) is needed in `results.yaml`:

1. Your username and the corresponding periods. This information is used for a pre-deadline check to give you confidence that you are using the right periods.
2. The parameter combination that gives the best profit on the in-sample period (where the series parameter is encoded in binary, see below); the corresponding profit.
3. The parameter combination that gives the best profit on the out-of-sample period (where the series parameter is encoded in binary, see below); the corresponding profit.
4. `rank_on_out`: The rank (a possibly fractional number between 1 and 88) that describes where the parameter combination from 2. ranks on the out-of-sample period.
5. `rank_on_in`: The rank (a possibly fractional number between 1 and 88) that describes where the parameter combination from 3. ranks on the in-sample period.

How to compute the rank

Use the `rank` (package:base) with the argument `ties.method='average'`.

Interpretation

An ideal scenario is for the best in-sample parameter combination to also be the best out-of-sample parameter combination. In practice, this is often not the case, as we have seen in the slides. Here, as we did in the slides, we are exploring the difference between parameter combination performance on in-sample and out-of-sample periods, where "a good outcome" is for the `rank_on_out` and `rank_on_in` to both be close to 1 (where 1 is ideal).

NOTE: you will **not** submit the code used to do the optimisation, which takes a long time to run; you will only submit the **results** of the optimisation in `results.yaml`.

Example output for `results.yaml`

In the `a2_yamls` subdirectory, three examples of `results.yaml` are provided for the fake usernames "x1xxx", "x1yyy", and "x1zzz", and using the "EXAMPLE" data. For "x1xxx", the yaml file contents are:

```
username: x1xxx
periods:
  startIn: 1
  endIn: 884
  startOut: 885
  endOut: 2000
ins:
  short: 5.0
  medium: 50.0
  long: 300.0
  series1: 1
  series2: 1
  series3: 1
  series4: 1
  profit: 5963.63
  rank_on_out: 2.0
out:
  short: 10.0
  medium: 50.0
  long: 300.0
  series1: 1
  series2: 1
  series3: 1
  series4: 0
  profit: 3072.562
  rank_on_in: 19.0
```

Note how the `params$series` parameter is represented in the yaml, as 4 binary variables (taking values 0 or 1): `series1`, `series2`, `series3`, and `series4`.

Once you have correctly completed part 1, and have also created the code to do the parameter sweep and ranking you can use these three examples to test your output. These examples are done using the "EXAMPLE" data so that they do not leak information about the correct answers on the "A2" data.

Marks breakdown for `results.yaml`

The marks for `results.yaml` are only available if you have achieved 70% on the first part. Moreover, the yaml file must have the right format, and must show the correct username and periods -- there is a pre-deadline test that checks all of this for you.

Here is an example blank `results.yaml`, shown with additional line numbers:

```
1 username:
2 periods:
3   startIn:
4   endIn:
5   startOut:
6   endOut:
7 ins:
8   short:
9   medium:
10  long:
11  series1:
12  series2:
```

```

13  series3:
14  series4:
15  profit:
16  rank_on_out:
17 out:
18  short:
19  medium:
20  long:
21  series1:
22  series2:
23  series3:
24  series4:
25  profit:
26  rank_on_in:

```

Note that the line numbers on the left are not part of the file; they are shown since they are used in the following tables.

Required for passing the pre-deadline check:

Field(s)	Line numbers in example	Marks
username	1	0
periods	3-6	0

Assuming that your submitted yaml passed the pre-deadline check, which checks the username and periods fields and the format of the yaml file, the following marks are available for the remaining fields:

Field(s)	Line numbers in example	Marks
In-sample best params (unique)	8-14	5
In-sample best profit	15	5
rank_on_out	16	5
Out-of-sample best params	18-24	5
Out-of-sample best profit	25	5
rank_on_in	26	5


Part 3: submission and pre-deadline tests

To get marks the submission of `strategy.R` is required; the submission of `results.yaml` is optional:

By default all files are *denied*. Exceptions and requirements:

ALLOWED

»




>

results.yaml

REQUIRED

»



>

strategy.R

There are pre-deadline tests for all 10 functions in `strategy.R` that are needed for the first 70% of marks.

Randomisation

To reduce the incentive for trying to hardcode answers, tests involve randomness in the inputs. This does mean that there can be some (small) variance in the mark for wrong answers, but there is none for correct answers. This is a reasonable price to pay for being able to see all the tests that were run openly.

For the functions `checkE01`,..., `checkE06`, these are "all-or-nothing" tests (to prevent always returning `TRUE` or always returning `FALSE` from getting marks). If you do not pass all tests, you will be shown only the tests that you failed. For example, here's what happens for `checkE01` if it just returns `TRUE`:

1

checkE01 Run the unit tests using python3.7 \$FIXTURES/run_tests.py checkE01.

0 / 3

Results

Output

checkE01 (0 / 4)

<div>checkE01 checkE01(['datapath: ./DATA/A2/O4.csv', 'data[50:150]', 'short=5', 'medium=50', 'long=100'])</div> <div>Expected [1] FALSE to be returned but got [1] TRUE</div> <div>No stack trace.</div>	<div>checkE01 checkE01(['datapath: ./DATA/A2/O4.csv', 'data[200:500]', 'short=5', 'medium=50', 'long=100'])</div> <div>Expected [1] FALSE to be returned but got [1] TRUE</div> <div>No stack trace.</div>
<div>checkE01 checkE01(['datapath: ./DATA/A2/O4.csv', 'data[100:200]', 'short=5', 'medium=50', 'long=100'])</div> <div>Expected [1] FALSE to be returned but got [1] TRUE</div> <div>No stack trace.</div>	<div>checkE01 checkE01(['datapath: ./DATA/A2/O2.csv', 'data[100:200]', 'short=5', 'medium=50', 'long=100'])</div> <div>Expected [1] FALSE to be returned but got [1] TRUE</div> <div>No stack trace.</div>

When the output says "expected [1] FALSE" that means that the input arguments should have passed this check.

One only sees tests where FALSE was expected but TRUE was returned. Here is what happens for `checkE02` if it always returns `FALSE` (one sees only tests where `TRUE` was expected):

checkE01 (0 / 8)

<div>checkE01 checkE01(['5', '50', '100'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>	<div>checkE01 checkE01(['short=5', '50', '100'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>
<div>checkE01 checkE01(['short=5'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>	<div>checkE01 checkE01(['short=5', '50'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>
<div>checkE01 checkE01(['short=5', 'medium=50', '100'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>	<div>checkE01 checkE01(['5', 'medium=50', 'long=100'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>
<div>checkE01 checkE01(['5', '50', 'long=100'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>	<div>checkE01 checkE01(['short=5', '50', 'long=100'])</div> <div>Expected [1] TRUE to be returned but got [1] FALSE</div> <div>No stack trace.</div>

For `getTMA`, partial marks are possible. Here's an example of the tests for a `getTMA` implementation that passes some but not all tests:

getTMA (4 / 8)

<p>getTMA getTMA(['datapath:./DATA/A2/01.csv', 'data[100:250]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> For short expected: 2891.0 received: 2870.0 diff exceeded error tolerance (0.01): 21.0 	✗	<p>getTMA getTMA(['datapath:./DATA/A2/02.csv', 'data[100:250]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> For short expected: 2240.4 received: 2215.2 diff exceeded error tolerance (0.01): 25.2000000000000273 	✗
<p>getTMA getTMA(['datapath:./DATA/A2/03.csv', 'data[100:250]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> For short expected: 313.36 received: 315.81 diff exceeded error tolerance (0.01): 2.4499999999999886 	✗	<p>getTMA getTMA(['datapath:./DATA/A2/04.csv', 'data[100:250]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> For short expected: 35.816 received: 35.791 diff exceeded error tolerance (0.01): 0.025000000000005684 	✗
<p>getTMA getTMA(['datapath:./DATA/A2/01.csv', 'data[200:600]', 'short=10', 'medium=75', 'long=200'])</p>	✓	<p>getTMA getTMA(['datapath:./DATA/A2/02.csv', 'data[200:600]', 'short=10', 'medium=75', 'long=200'])</p>	✓
<p>getTMA getTMA(['datapath:./DATA/A2/03.csv', 'data[200:600]', 'short=10', 'medium=75', 'long=200'])</p>	✓	<p>getTMA getTMA(['datapath:./DATA/A2/04.csv', 'data[200:600]', 'short=10', 'medium=75', 'long=200'])</p>	✓

The way this submission was created was to break a corerct implementation for the short TMA for certain lookback values. Note that the errors on CodeGrade show that the problem is only with the short TMA and only for certain values of the lookback; this type of information may be useful in debugging. Note also that a broken getTMA (or getPosSignFromTMA or getPosSize) should also break getOrders, because it should use them. For example, here's the output for getOrders when the same submission as used for getTMA is used:

getOrders (7 / 13)

<p>getOrders getOrders(['data[1:884]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> expected: 1650.898 received: 2219.918 diff exceeded error tolerance (0.01): 569.02000000000002 	✗	<p>getOrders getOrders(['data[1:819]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> expected: 1342.302 received: 1920.684 diff exceeded error tolerance (0.01): 578.38200000000001 	✗
<p>getOrders getOrders(['data[1:817]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> expected: 1330.294 received: 1908.676 diff exceeded error tolerance (0.01): 578.38199999999998 	✗	<p>getOrders getOrders(['data[1:884]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> expected: 2095.28 received: 2342.66 diff exceeded error tolerance (0.01): 247.37999999999995 	✗
<p>getOrders getOrders(['data[1:819]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> expected: 1427.4 received: 1684.38 diff exceeded error tolerance (0.01): 256.98 	✗	<p>getOrders getOrders(['data[1:817]', 'short=5', 'medium=50', 'long=100'])</p> <p>Output failed to match the expected output.</p> <ol style="list-style-type: none"> expected: 1461.4 received: 1718.38 diff exceeded error tolerance (0.01): 256.98 	✗
<p>getOrders getOrders(['data[284:905]', 'short=15', 'medium=75', 'long=150'])</p>	✓	<p>getOrders getOrders(['data[219:615]', 'short=15', 'medium=75', 'long=150'])</p>	✓
<p>getOrders getOrders(['data[341:926]', 'short=15', 'medium=75', 'long=150'])</p>	✓	<p>getOrders getOrders(['data[284:905]', 'short=15', 'medium=75', 'long=150'])</p>	✓
<p>getOrders getOrders(['data[219:615]', 'short=15', 'medium=75', 'long=150'])</p>	✓	<p>getOrders getOrders(['data[341:926]', 'short=15', 'medium=75', 'long=150'])</p>	✓
<p>getOrders getOrders(['data[1:2000]', 'short=17', 'medium=24', 'long=99'])</p>	✓		

As for getTMA, partial marks are possible for getOrders. The tests for getOrders use the resulting profit for comparison.

Since getPosSignFromTMA should only return 0,1,-1, it is an "all-or-nothing" test. Here's the output when a wrong implementation that always return 1 is submitted (only failed tests where an expected output of 0 or -1 are shown):

getPosSignFromTMA (0 / 12)

<p>getPosSignFromTMA getPosSignFromTMA(['short=10.0', 'medium=10.0', 'long=10.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>	<p>getPosSignFromTMA getPosSignFromTMA(['short=10.0', 'medium=10.0', 'long=20.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>
<p>getPosSignFromTMA getPosSignFromTMA(['short=10.0', 'medium=10.0', 'long=30.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>	<p>getPosSignFromTMA getPosSignFromTMA(['short=10.0', 'medium=20.0', 'long=20.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>
<p>getPosSignFromTMA getPosSignFromTMA(['short=10.0', 'medium=20.0', 'long=30.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: -1 2. received: 1</p>	<p>getPosSignFromTMA getPosSignFromTMA(['short=10.0', 'medium=30.0', 'long=30.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>
<p>getPosSignFromTMA getPosSignFromTMA(['short=20.0', 'medium=20.0', 'long=20.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>	<p>getPosSignFromTMA getPosSignFromTMA(['short=20.0', 'medium=20.0', 'long=30.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>
<p>getPosSignFromTMA getPosSignFromTMA(['short=20.0', 'medium=30.0', 'long=30.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>	<p>getPosSignFromTMA getPosSignFromTMA(['short=30.0', 'medium=30.0', 'long=30.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 1</p>
<p>getPosSignFromTMA getPosSignFromTMA(['short=5.0', 'medium=12.0', 'long=50.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: -1 2. received: 1</p>	<p>getPosSignFromTMA getPosSignFromTMA(['short=60.0', 'medium=70.0', 'long=100.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: -1 2. received: 1</p>

For getPosSize, since it is very simple and there should not be lots of "edge cases" we again implement it as an "all-or-nothing" test. Here's the output when the flooring of the position size has been ommitted:

getPosSize (0 / 4)

<p>getPosSize getPosSize(['100.5', '1000.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 9 2. received: 9.950249</p>	<p>getPosSize getPosSize(['100.5', '100.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 0.9950249</p>
<p>getPosSize getPosSize(['200.0', '100.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 0 2. received: 0.5</p>	<p>getPosSize getPosSize(['100.5', '10000.0'])</p> <p>Output failed to match the expected output.</p> <p>1. expected: 99 2. received: 99.50249</p>

So for getTMA and getOrders partial marks are possible, for the other 8 functions it is all or nothing, and then only failed tests are shown if the test is not passed.

For results.yaml, the pre-deadline test checks that the username and periods are correct and that the format of the yaml is correct. Here is any example of using the wrong username:

You scored 100% of the 100% required to continue.			
YAML pre-deadline checks		Options · 0 %	
No	Summary	Score	Pass
> 1	YAML pre-deadline checks Run python3.7 \$FIXTURES/check_yaml.py \$STUDENT/results.yaml 15913 and check for successful completion.	0 / 1	✖
You scored 98.59% of the 100% required to continue. No further tests will be run.			

Output

1. *****
2. Expected username rsjs but got rsjs1
3. Failed to pass the checks.
4. *****

Errors can also arise for the wrong periods or for a badly formatted yaml or one with the wrong fields.

When the submitted yaml passes all pre-deadline tests, you will see the following:

getOrders				Options	100 %
No	Summary			Score	Pass
> 1	getOrders Run the unit tests using python3.7 \$FIXTURES/run_tests.py getOrders.			25 / 25	✓
You scored 100% of the 100% required to continue.					

YAML pre-deadline checks				Options	100 %
No	Summary			Score	Pass
> 1	YAML pre-deadline checks Run python3.7 \$FIXTURES/check_yaml.py \$STUDENT/results.yaml 15913 and check for successful completion.			1 / 1	✓
You scored 100% of the 100% required to continue.					

Only in this case will your `results.yaml` submission be marked post-deadline.

Warning

Your code will be put through the department's automatic plagiarism and collusion detection system. Student's found to have plagiarized or colluded will likely receive a mark of zero. Do not show your work to other students, and do not search for answers online.

Good luck with the assignment.

THE END