Readme

Requirements

- Visual Studio 2019 v.16.8.4
- Microsoft .NET framework v.4.8.04161
- Windows 10/11

The software is setup to run on 32bit machines therefore any Windows 10/11 OS should support it.

The project may work with slightly different versions of Visual Studio and .NET framework but above is what it was tested on.

Quickly testing the project using the executable

I've also packaged a binary (.exe) version of the software in this zip called "Executable". If you want to run the software without compiling it, you can simply extract this folder from the zip, and execute "Thomas_Shaer_Dissertation.exe". You can also run the tests by executing "Tests.exe".

I also give a sample code file called "tradingdemo.al" in the Example Code directory. This is the same example seen in the main report under "backtesting case study". I have also included the EURUSD.csv data that is required for this script in the Example Data directory.

Instructions on how to setup the software development environment

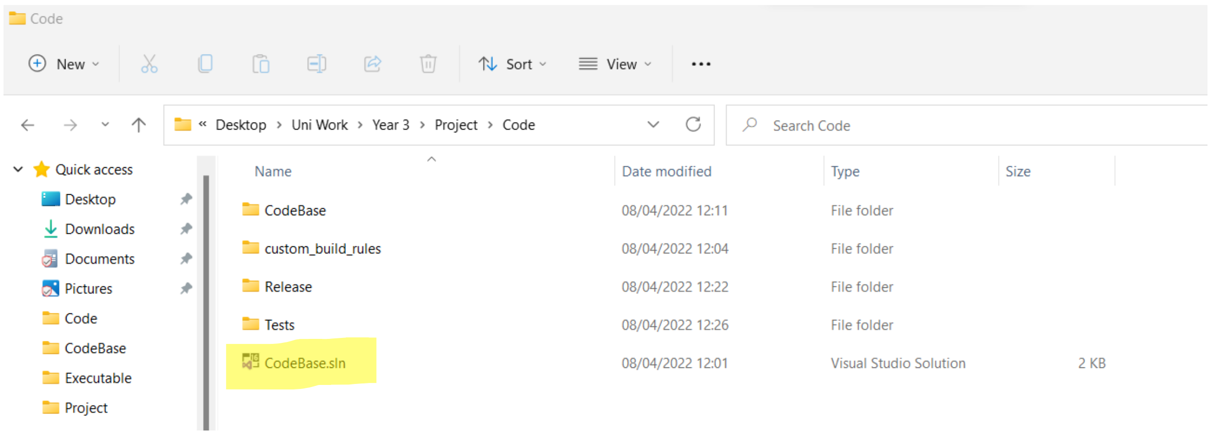
Setting up the development environment for this project can be difficult due to the configuration of the compiler, linking and installing libraries.

To make things easy, I have included the entire development environment for the project contained within a single folder. This includes a Visual Studio solution file, all the source code of the project, all the libraries and dependencies and a preconfigured .vcxproj file which automatically links all these different components together such that no additional configurations must take place for the MSVC (Microsoft Visual C++) compiler. All paths within this folder are relative, so there should be no path linking/dependency errors.

This means that all you must do to setup the development environment, is open the CodeBase.sln solution file which will open Visual Studio 2019 with the project already configured and ready to run.

Steps:

1. Extract the "Code" folder located in the submission zip file.
2. Double click the "CodeBase.sln" file



Name	Date modified	Type	Size
CodeBase	08/04/2022 12:11	File folder	
custom_build_rules	08/04/2022 12:04	File folder	
Release	08/04/2022 12:22	File folder	
Tests	08/04/2022 12:26	File folder	
CodeBase.sln	08/04/2022 12:01	Visual Studio Solution	2 KB

Figure 1 To open the project all that has to be done is double click the CodeBase.sln file

3. This will open the project in Visual Studio 2019. Locate and open the “Solution Explorer” tab if not already present. You should see two projects: **CodeBase** and **Tests.**

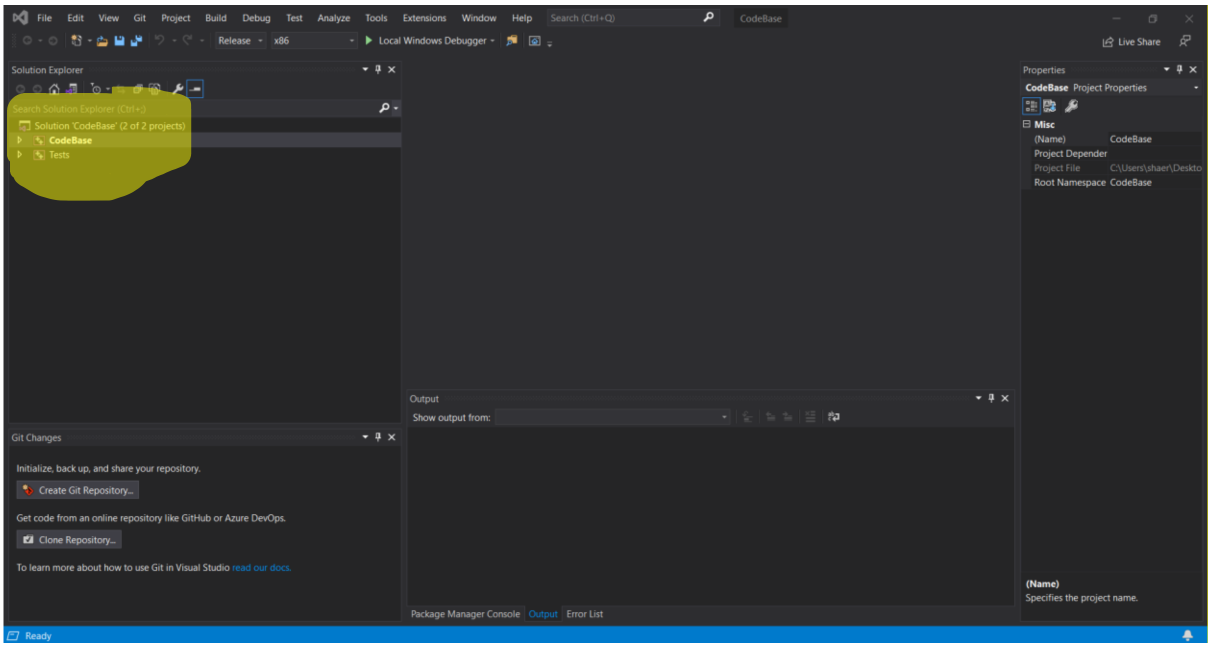
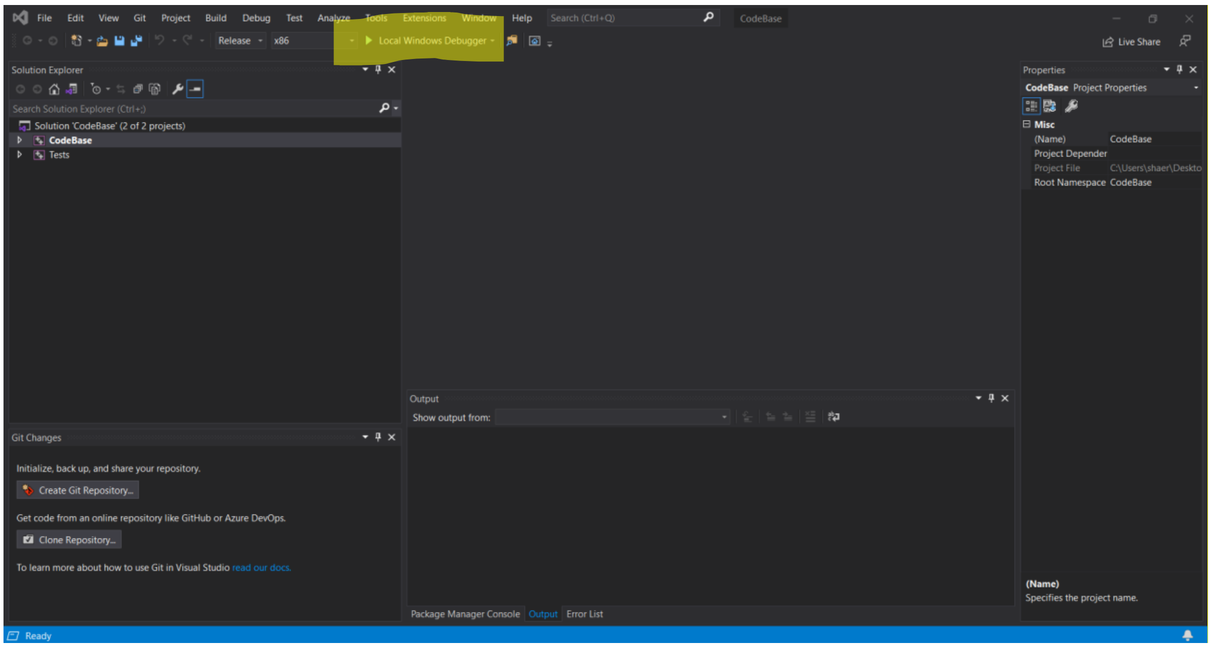

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Release
x86
Local Windows Debugger
Live Share
Solution Explorer
Solution 'CodeBase' (2 of 2 projects)
CodeBase
Tests
Properties
CodeBase Project Properties
Misc
(Name) CodeBase
Project Depender
Project File C:\Users\shaer\Deskto
Root Namespace CodeBase
(Name)
Specifies the project name.
Output
Show output from:
Git Changes
Initialize, back up, and share your repository.
Create Git Repository...
Get code from an online repository like GitHub or Azure DevOps.
Clone Repository...
To learn more about how to use Git in Visual Studio read our docs.
Package Manager Console Output Error List
Ready

Figure 2 This is what Visual Studio 2019 should look like when you open it up. The two projects “CodeBase” and “Tests” should be accessible from the “Solution Explorer” window.

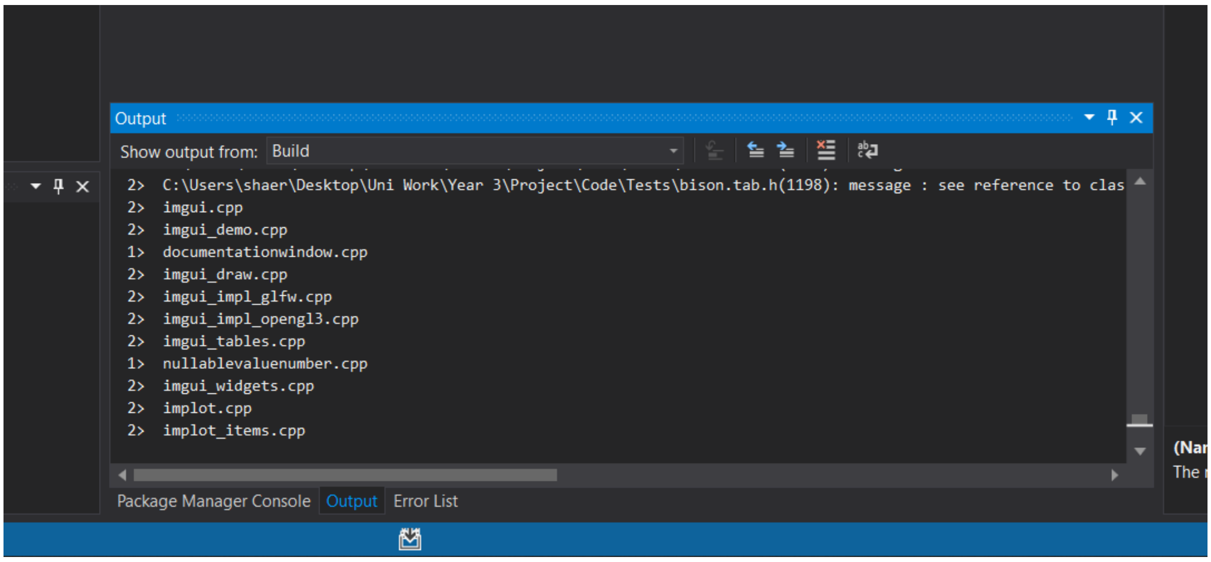
4. The **CodeBase** project contains the actual code for the software, while the **Tests** project contains the code for testing the software.
5. It is recommended that you first run the **CodeBase** project as **Tests** is dependent on it. This should already be the case but if not please follow the steps from this tutorial[i]:

   a. **Right-click on your Solution within the Solution Explorer**
   b. Select the **Set Startup Projects** option
   c. **Select the Projects that you want to run under the Multiple Startup Projects area** or **select a single project from the Single Startup Project sections**
   d. Click **Apply** to apply your changes

To run the project, click the green arrow just below the toolbar called “Local Window debugger”.



The first time it runs, you should see a lot of text in the Output window at the bottom. This is normal and should not be a cause for concern. This is because Visual Studio will compile the entire program as it has not yet been run. The project will be recompiled only if a change is made to the source code/compiler settings. The compilation may take a few minutes depending on your hardware.

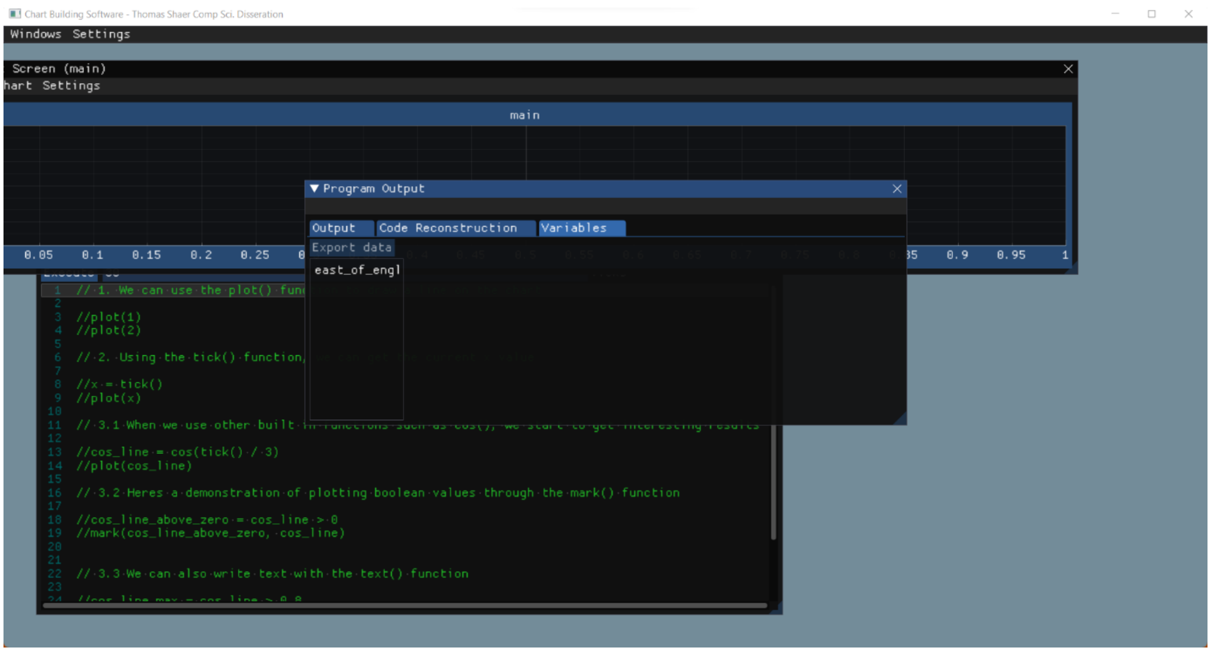


```
2>  C:\Users\shaer\Desktop\Uni Work\Year 3\Project\Code\Tests\bison.tab.h(1198): message : see reference to clas
2>  imgui.cpp
2>  imgui_demo.cpp
1>  documentationwindow.cpp
2>  imgui_draw.cpp
2>  imgui_impl_glfw.cpp
2>  imgui_impl_opengl3.cpp
2>  imgui_tables.cpp
1>  nullablevaluenumber.cpp
2>  imgui_widgets.cpp
2>  implot.cpp
2>  implot_items.cpp
```

Any modification of the .cpp/.h files will result in a partial/full recompilation of the program. If any errors occur they will be displayed in the Error List. Errors might occur if all the requirements aren’t fulfilled in the requirement section. Googling can help solve them.

Figure 3 Text will flow down the output window when compiling the project for the first time. This is normal. If anything goes wrong, you will be informed at the end.

6. The GUI should open automatically. Note, if the windows look funny, they can be resized and opened from the main tool bar.

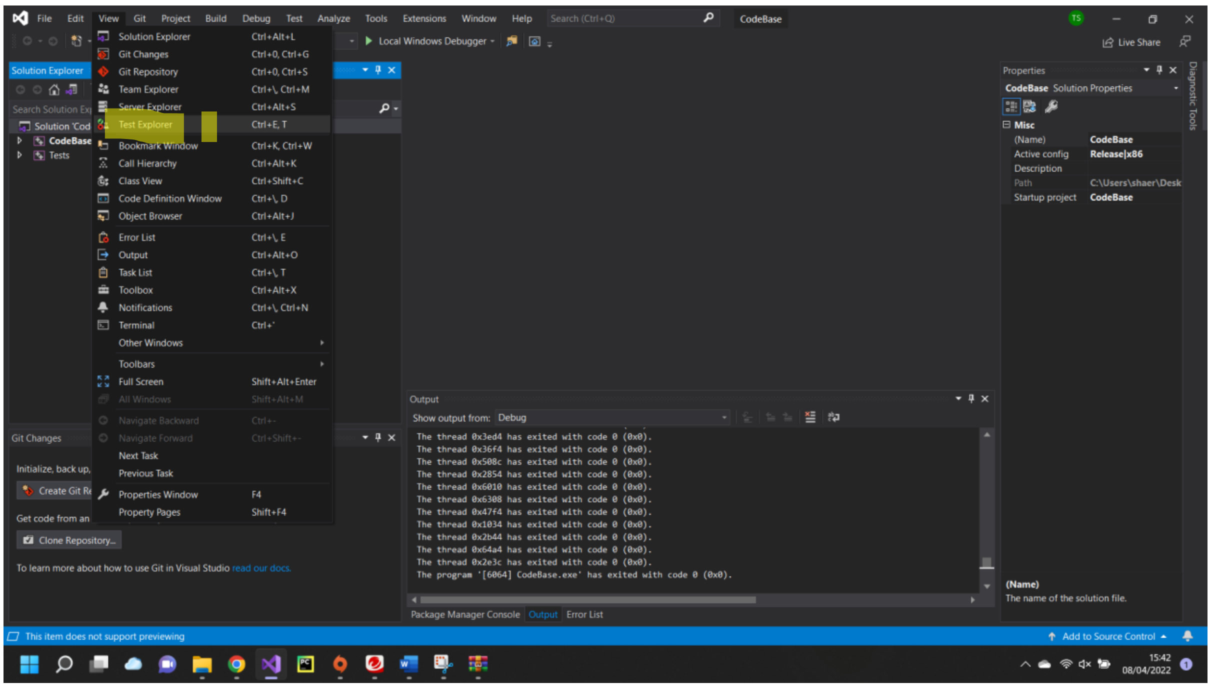


```
// 1. We can use the plot() func

//plot(1)
//plot(2)

// 2. Using the tick() function,

//x = tick()
//plot(x)

// 3.1 When we use other built

//cos_line = cos(tick() / 3)
//plot(cos_line)

// 3.2 Heres a demonstration of plotting boolean values through the mark() function

//cos_line_above_zero = cos_line > 0
//mark(cos_line_above_zero, cos_line)


// 3.3 We can also write text with the text() function
```

Figure 4 The first time the program opens up, the windows may overlap or be too small/large. Windows can be resized and moved with the mouse. You can open up any windows that are closed via the menu bar.

Running the Tests

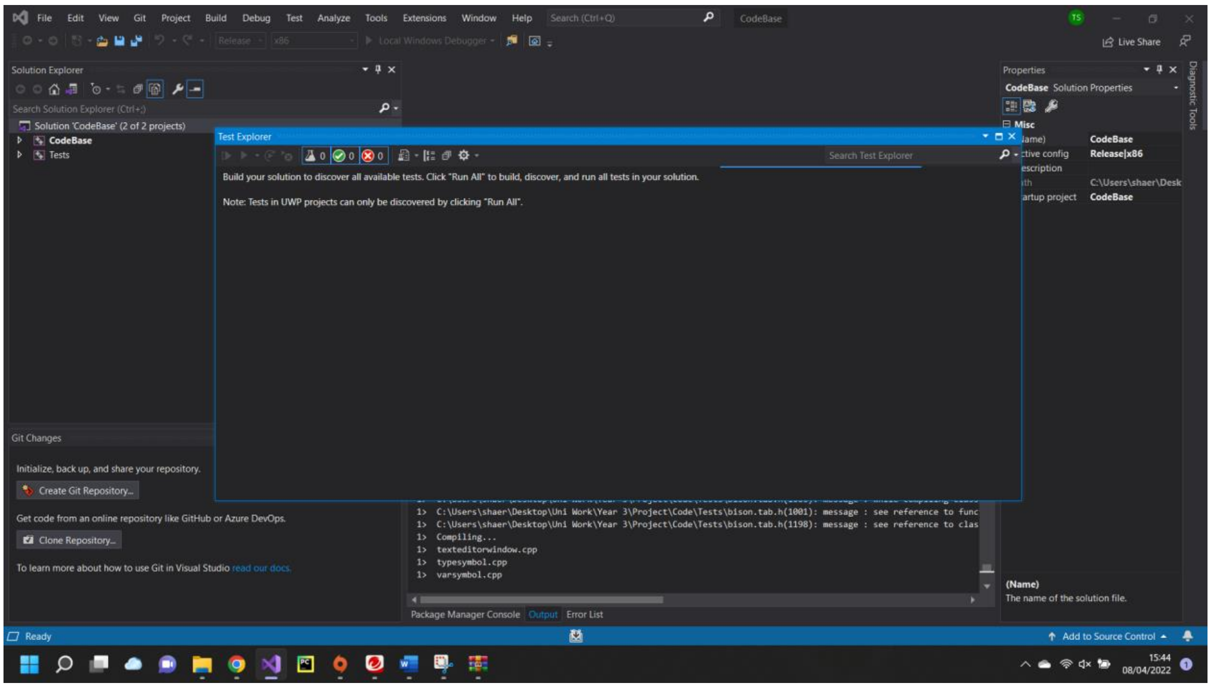
1. To run the tests, you should open the "Test Explorer" window.


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
CodeBase
Local Windows Debugger
Live Share
Solution Explorer Ctrl+Alt+L
Git Changes Ctrl+0, Ctrl+G
Git Repository Ctrl+0, Ctrl+S
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Test Explorer Ctrl+E, T
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X
Notifications Ctrl+\, Ctrl+N
Terminal Ctrl+`
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
All Windows Shift+Alt+M
Navigate Backward Ctrl+-
Navigate Forward Ctrl+Shift+-
Next Task
Previous Task
Properties Window F4
Property Pages Shift+F4
Solution Explorer
Search Solution Ex
Solution 'Cod
CodeBase
Tests
Git Changes
Initialize, back up,
Create Git Re
Get code from an
Clone Repository...
To learn more about how to use Git in Visual Studio read our docs.
Properties
CodeBase Solution Properties
Misc
(Name) CodeBase
Active config Release|x86
Description
Path C:\Users\shaer\Desk
Startup project CodeBase
(Name)
The name of the solution file.
Output
Show output from: Debug
Package Manager Console Output Error List
This item does not support previewing
Add to Source Control
15:42
08/04/2022

```
The thread 0x3ed4 has exited with code 0 (0x0).
The thread 0x36f4 has exited with code 0 (0x0).
The thread 0x508c has exited with code 0 (0x0).
The thread 0x2854 has exited with code 0 (0x0).
The thread 0x6010 has exited with code 0 (0x0).
The thread 0x6308 has exited with code 0 (0x0).
The thread 0x47f4 has exited with code 0 (0x0).
The thread 0x1034 has exited with code 0 (0x0).
The thread 0x2b44 has exited with code 0 (0x0).
The thread 0x64a4 has exited with code 0 (0x0).
The thread 0x2e3c has exited with code 0 (0x0).
The program '[6064] CodeBase.exe' has exited with code 0 (0x0).
```

2. By default, no tests might be detected. Press the green button on the top left of the text explorer window, which will find and run all the tests automatically. The Tests code will be

recompiled the first time you run it (seen in the output window), so it might take a few minutes to run.

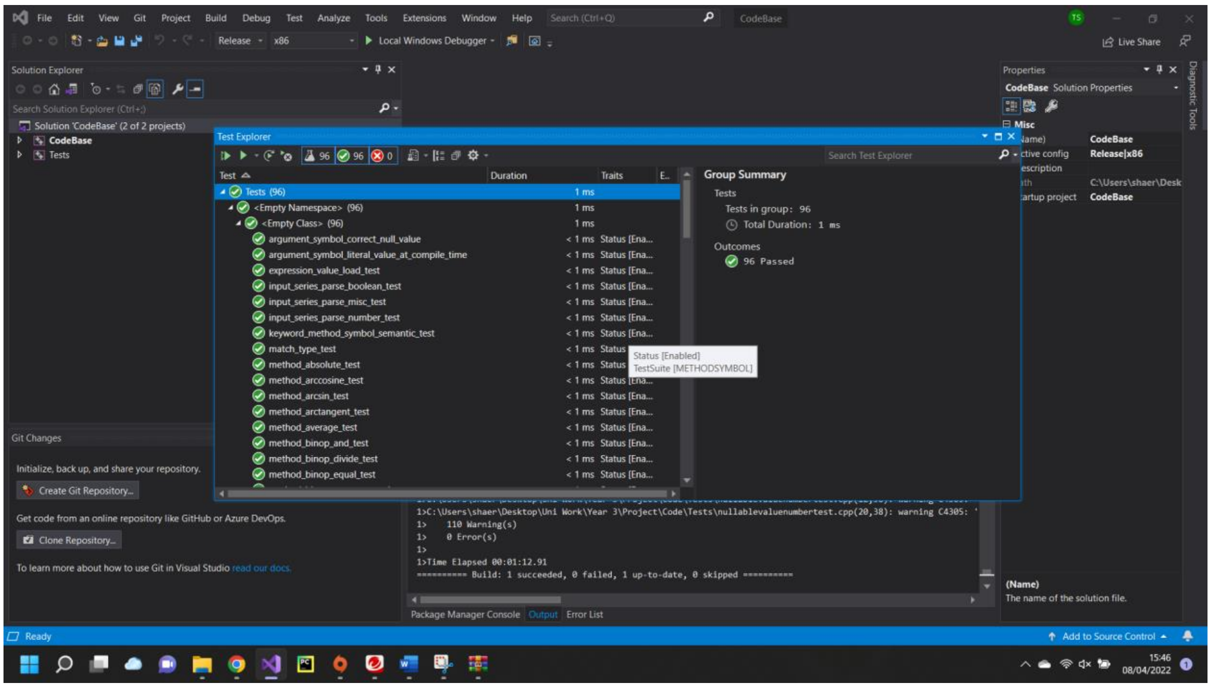


Build your solution to discover all available tests. Click "Run All" to build, discover, and run all tests in your solution.

Note: Tests in UWP projects can only be discovered by clicking "Run All".

```
1>  C:\Users\shaer\Desktop\Uni Work\Year 3\Project\Code\Tests\bison.tab.h(1001): message : see reference to func
1>  C:\Users\shaer\Desktop\Uni Work\Year 3\Project\Code\Tests\bison.tab.h(1198): message : see reference to clas
1>  Compiling...
1>  texteditorwindow.cpp
1>  typesymbol.cpp
1>  varsymbol.cpp
```

Figure 5 The first time you try run the tests it will automatically find them and compile the tests code. This may take a few minutes



Test	Duration	Traits
Tests (96)	1 ms	
<Empty Namespace> (96)	1 ms	
<Empty Class> (96)	1 ms	
argument_symbol_correct_null_value	< 1 ms	Status [Ena...
argument_symbol_literal_value_at_compile_time	< 1 ms	Status [Ena...
expression_value_load_test	< 1 ms	Status [Ena...
input_series_parse_boolean_test	< 1 ms	Status [Ena...
input_series_parse_misc_test	< 1 ms	Status [Ena...
input_series_parse_number_test	< 1 ms	Status [Ena...
keyword_method_symbol_semantic_test	< 1 ms	Status [Ena...
match_type_test	< 1 ms	Status
method_absolute_test	< 1 ms	Status
method_arccosine_test	< 1 ms	Status [Ena...
method_arcsin_test	< 1 ms	Status [Ena...
method_arctangent_test	< 1 ms	Status [Ena...
method_average_test	< 1 ms	Status [Ena...
method_binop_and_test	< 1 ms	Status [Ena...
method_binop_divide_test	< 1 ms	Status [Ena...
method_binop_equal_test	< 1 ms	Status [Ena...

Status [Enabled]
TestSuite [METHODSYMBOL]

Group Summary
Tests
Tests in group: 96
Total Duration: 1 ms
Outcomes
96 Passed

```
1>C:\Users\shaer\Desktop\Uni Work\Year 3\Project\Code\Tests\nullablevaluenumbertest.cpp(20,38): warning C4305: '
1>    110 Warning(s)
1>    0 Error(s)
1>
1>Time Elapsed 00:01:12.91
========== Build: 1 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========
```

Figure 6 Eventually you will see all the tests displayed below in the test explorer window. They should all pass.

Additional Information about the compiler

- Windows SDK version: 10.0
- Platform Toolset: Visual Studio 2019 (v142)
- C++ Language Standard: Previous from the Latest C++ Working Draft (/std:c++latest)

Main Codebase folder breakdown (from the final report)

Folders

- data:
  - contains files related to Bison/Flex GNU library.
- include:
  - contains source files for the libraries: boost, GLFW and nlohmann-json
- lib:
  - contains lib files for static linking for the libraries: boost, GLFW and nlohmann-json
- misc:
  - contains files related to graphics library Dear ImGUI
- Release:
  - contains .obj files for compiled source files
- src:
  - extra source and header files

Files

Executables (.exe)

- win_bison.exe
  - Compiled version of parser generator GNU Bison
- win_flex.exe
  - Compiled version of lexer generator Flex

Program save files

- imgui.ini
  - GUI layout save file for Dear ImGUI library
- settings.json
  - All over saveable settings related to functioning of software

Flex/Bison input files

- bison.y
  - GNU Bison input file. Contains instructions on how to generate the parser
- flex.l
  - Flex input file. Contains instructions on how to generate lexer

Generated Flex/Bison files

- bison.tab.cpp
- bison.tab.h

- flex.flex.cpp
- flex.flex.h

Visual Studio save files

- CodeBase.aps
- CodeBase.rc
- CodeBase.vcxproj
- CodeBase.vcxproj.filters
- CodeBase.vcxproj.user

ImGUI library files

- imconfig.h
- imfilebrowser.h
- imgui.cpp
- imgui.h
- imgui_demo.cpp
- imgui_draw.cpp
- imgui_impl_glfw.cpp
- imgui_impl_glfw.h
- imgui_impl_opengl3.cpp
- imgui_impl_opengl3.h
- imgui_impl_opengl3_loader.h
- imgui_internal.h
- imgui_tables.cpp
- imgui_widgets.cpp
- implot.cpp
- implot.h
- implot_internal.h
- implot_items.cpp
- imstb_rectpack.h
- imstb_textedit.h
- imstb_truetype.h
- stb_image_write.h
- TextEditor.cpp
- TextEditor.h
- resource.h

Project files

- argumentsymbol.cpp
- argumentsymbol.h
- chartplot.cpp
- chartplot.h
- chartwindow.cpp
- chartwindow.h
- datamanagerwindow.cpp
- datamanagerwindow.h
- dataparseexception.h

- documentationwindow.cpp
- documentationwindow.h
- expressionvalue.h
- inputseries.cpp
- inputseries.h
- interpreter.cpp
- interpretercontext.cpp
- interpretercontext.h
- jsonsettings.cpp
- jsonsettings.h
- languageexception.cpp
- langaugeexception.h
- main.cpp
- maingui.cpp
- maingui.h
- menubar.cpp
- menubar.h
- methodbucket.cpp
- methodbucket.h
- methodimplementations.cpp
- methodimplementations.h
- methodimplementationsinterpreter.cpp
- methodimplementationssemantic.cpp
- methodsymbol.cpp
- methodsymbol.h
- node.cpp
- node.h
- nullablevalue.h
- nullablevalueboolean.cpp
- nullablevalueboolean.h
- nullablevaluestring.cpp
- nullablevaluestring.h
- outputwindow.cpp
- outputwindow.h
- parametersymbol.cpp
- parametersymbol.h
- returnsymbol.cpp
- returnsymbol.h
- screenshot.cpp
- screenshot.h
- semanticanalysis.cpp
- sourcelocation.cpp
- sourcelocation.h
- symboltable.cpp
- symboltable.h
- texteditorwindow.cpp

- texteditorwindow.h
- typesymbol.cpp
- typesymbol.h
- varsymbol.cpp
- varsymbol.h
- window.cpp
- window.h

Main Tests folder breakdown (from the final report)

Folders

- include:
  - contains source files for the libraries: boost, GLFW and nlohmann-json
- lib:
  - contains lib files for static linking for the libraries: boost, GLFW and nlohmann-json
- Release & Testing:
  - contains .obj files for compiled source files

Files

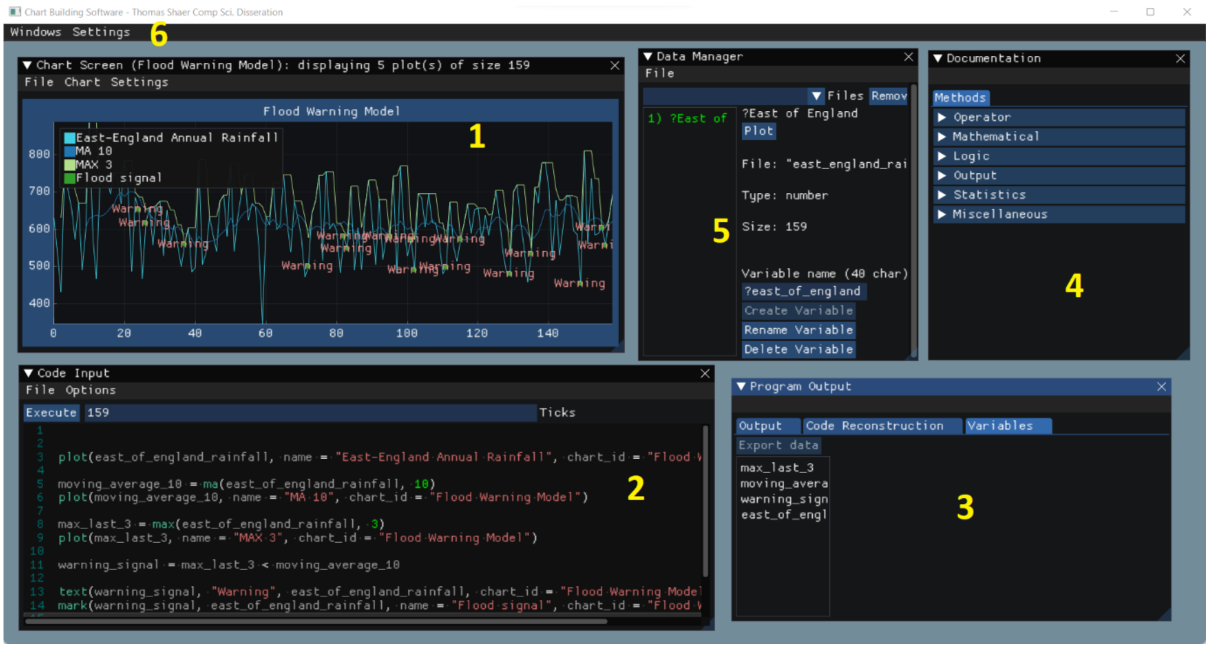
Visual Studio save files

- Tests.vcxproj
- Tests.vcxproj.filters
- Tests.vcxproj.user

Project files

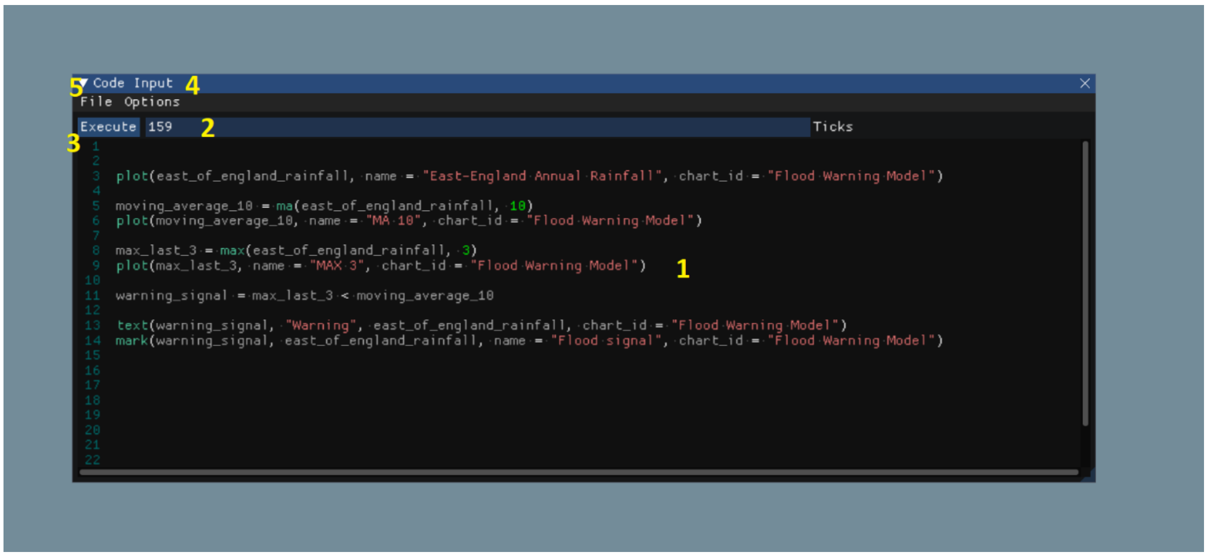
- argumentsymboltest.cpp
- expressionvaluetest.cpp
- inputseriestest.cpp
- main.cpp
- methodbuckettest.cpp
- methodimplementationstest.cpp
- methodsymboltest.cpp
- nullablevaluebooleantest.cpp
- nullablevaluenumbertest.cpp
- nullablevaluestringtest.cpp
- parametersymboltest.cpp
- returnsymboltest.cpp
- sourcelocationtest.cpp
- symboltabletest.cpp
- typesymboltest.cpp
- valuetesthelper.h
- varsymboltest.cpp

Program overview (from the final report)



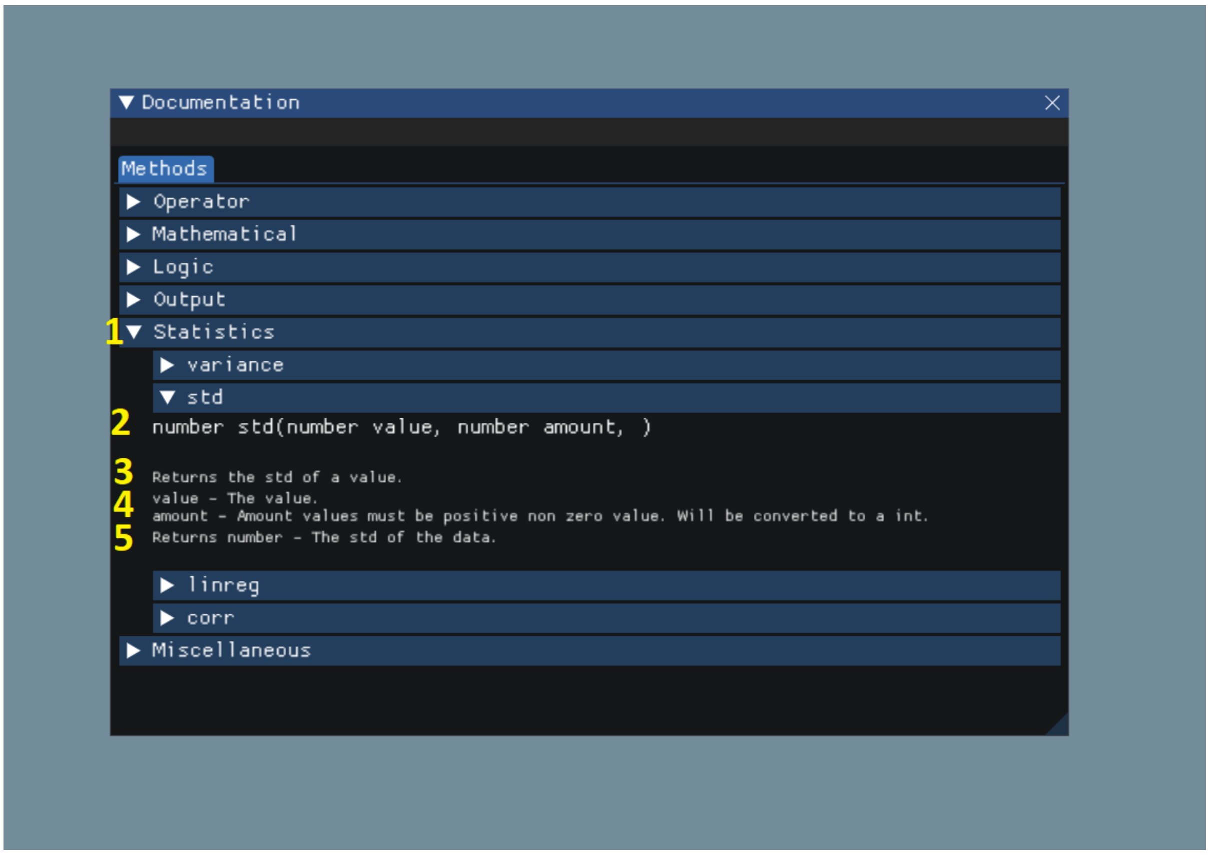
```
plot(east_of_england_rainfall, name = "East-England Annual Rainfall", chart_id = "Flood W

moving_average_10 = ma(east_of_england_rainfall, 10)
plot(moving_average_10, name = "MA 10", chart_id = "Flood Warning Model")

max_last_3 = max(east_of_england_rainfall, 3)
plot(max_last_3, name = "MAX 3", chart_id = "Flood Warning Model")

warning_signal = max_last_3 < moving_average_10

text(warning_signal, "Warning", east_of_england_rainfall, chart_id = "Flood Warning Mode
mark(warning_signal, east_of_england_rainfall, name = "Flood signal", chart_id = "Flood W
```

Figure 7 Overview of all windows
1. Chart window – displays chart content, dynamically created depending on what the script plots to.
2. Code editor window – where the user inputs/manage their code files.
3. Program output window – output from the script is displayed here, including error messages, code reconstruction and user defined variables.
4. Documentation window – details of all registered built-in methods by category, are automatically display here.
5. Data manager window – where data is imported into the program and turned into language variables.
6. Main window toolbar – windows can be opened/closed from here and the global text size can be adjusted



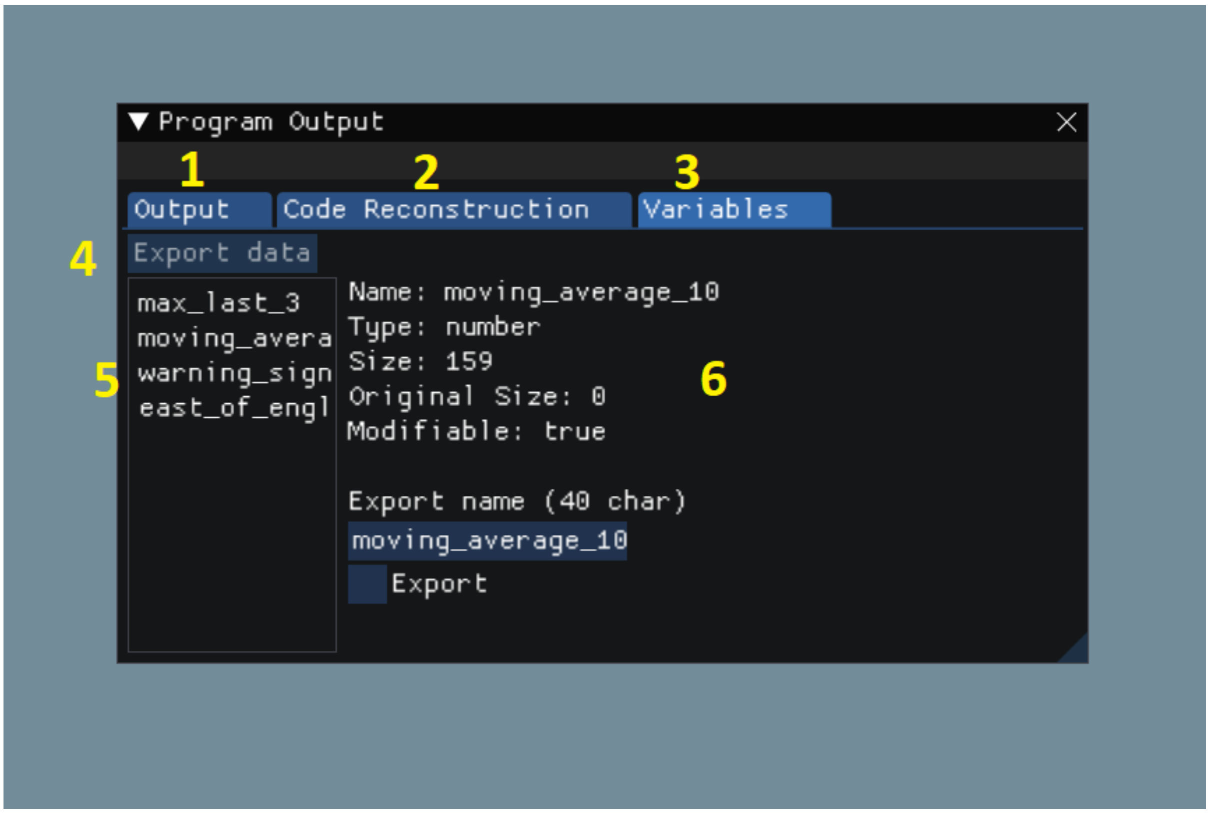
```
plot(east_of_england_rainfall, name = "East-England Annual Rainfall", chart_id = "Flood Warning Model")

moving_average_10 = ma(east_of_england_rainfall, 10)
plot(moving_average_10, name = "MA 10", chart_id = "Flood Warning Model")

max_last_3 = max(east_of_england_rainfall, 3)
plot(max_last_3, name = "MAX 3", chart_id = "Flood Warning Model")

warning_signal = max_last_3 < moving_average_10

text(warning_signal, "Warning", east_of_england_rainfall, chart_id = "Flood Warning Model")
mark(warning_signal, east_of_england_rainfall, name = "Flood signal", chart_id = "Flood Warning Model")
```

Figure 8 Overview of code editor window
1. Text input – this is where the user writes the code.
2. Number of ticks input – the user can select how many times they want the code to execute for (i.e., the size of internal arrays), by default this will revert to the maximum size of the largest inputted variable.
3. Execute button – runs the code.
4. Options menu – allows the user to toggle code highlighting.
5. File menu – allows the user to create/load/save code files to/from disk.



```
number std(number value, number amount, )

Returns the std of a value.
value - The value.
amount - Amount values must be positive non zero value. Will be converted to a int.
Returns number - The std of the data.
```

Figure 9 Overview of documentation window
1. Method category tabs – groups all methods via tabs, currently there is operators, mathematical, logic, output, statistics and miscellaneous.
2. Signature of method – shows signature of method, including return type, name and parameter types.
3. Method description – displays description of method.
4. Method parameters – displays type and description of parameters.
5. Method return type – displays type and description of the return type.



```
max_last_3
moving_avera
warning_sign
east_of_engl

Name: moving_average_10
Type: number
Size: 159
Original Size: 0
Modifiable: true

Export name (40 char)
moving_average_10
```

Figure 10 Overview of output window
1. Text output tab – displays any error messages after running the script.
2. Code reconstruction tab – displays how the code is parsed and interpreted in AST form.
3. Variables tab – displays information about all variables in script including export settings.
4. Export data – takes user to export dialog after selecting some variables to export.
5. Variables – lists all variables in program.
6. Variable information – shows information about selected variable.

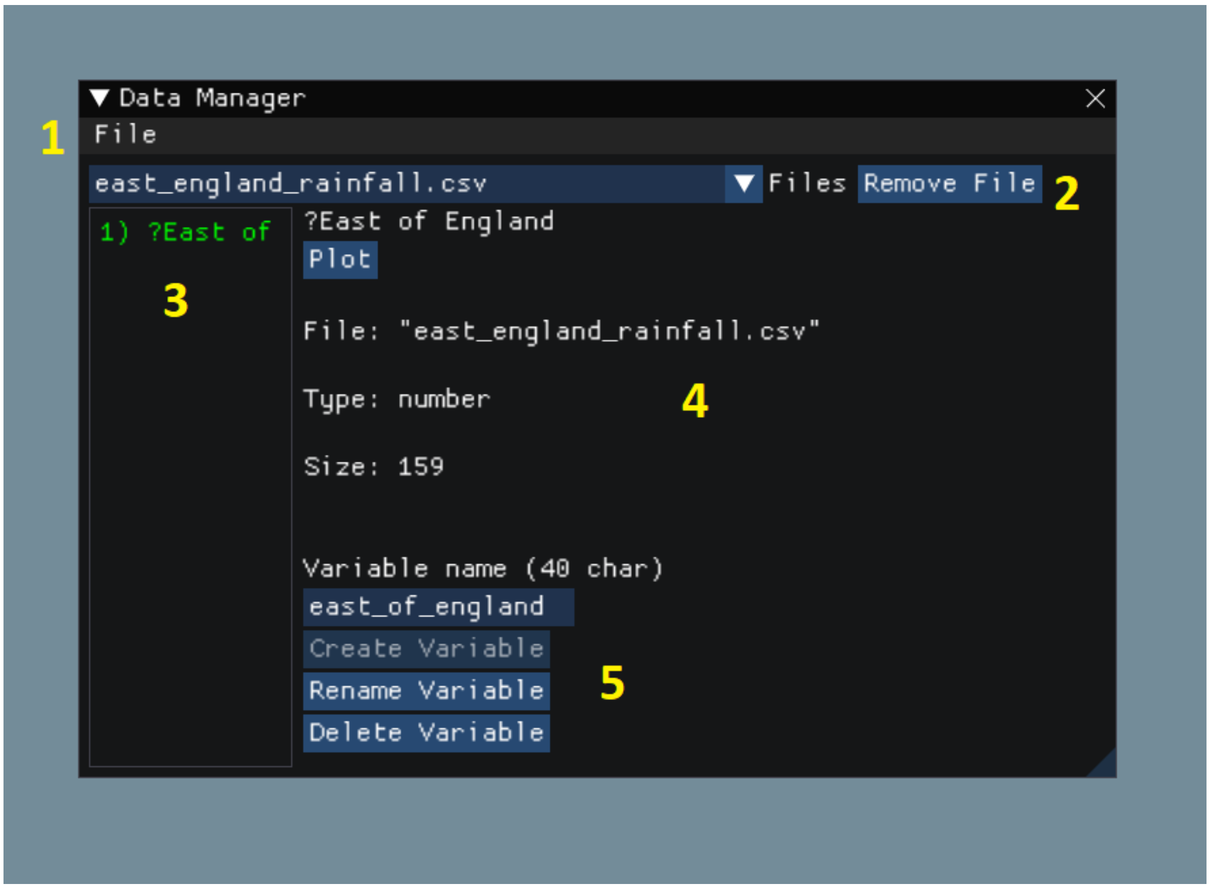


Figure 11 Overview of data manager window
1. File menu – allows users to import/remove all data files.
2. Remove file – allows the user to select a data file to remove. Will remove all associated series/variables with the file.
3. Series list – displays list of all parsed series from imported data files.
4. Series information – displays information about selected series.
5. Variable creation settings – allows the user to convert the series into a variable, including other management settings.

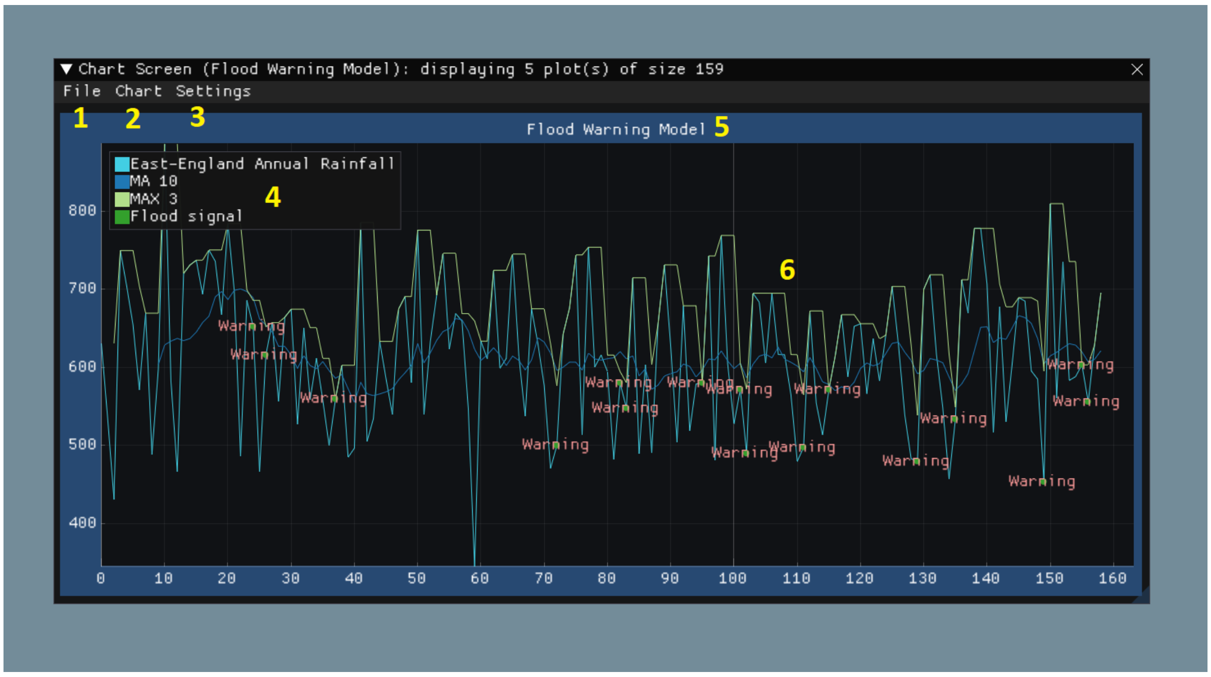


Figure 12 Overview of a chart window
1. File menu – contains settings for exporting chart as a PNG to disk.

2. Chart menu – contains chart control settings, such as clearing the chart, rescaling axis, and ability to disable the key.
3. Settings menu – contains setting for toggling chart anti-aliasing.
4. Chart key – name to colour mapping for each plotted series – name of key is specified by optional "name" parameter of plot() and mark() functions (default is empty)
5. Chart title – title of the chart – specified by optional "chart_id" parameter of plot(), mark() and text() functions (default is "main")
6. Contents of chart – interactable chart explorer.

Language Features (from the final report)

Name	Syntax	Description

Assign/declare	x = 2
	y = tick()
		Declares or assigns a value to a variable.
		If it's the first time a variable is assigned to, it's type will be inferred from the assignable expression.
		Any other assigns after the initial one must have the assignable expression of the same type as the initial one.

If statement	if (3 > 2) {
	…
	}
		An if statement is formed from a Boolean condition and block of code that will be executed if the condition is true.
		Note: else statements are not supported.

Method Call	plot(2)
	y = avg(3, 5)
		A method call is used to invocate a function's logic from the standard library.
		Method calls can occur either as part of an expression or called as a statement on their own.

Ternary Statement	y = x > 3 ? 6 : 2	A ternary statement is an expression made up of a Boolean condition and two expressions. If the Boolean condition is true, the left-hand expression will be executed and returned, and if it is false, the right hand expression will be executed and returned. The two return types of the expressions must be the same.
Comment	// this line will be ignored	A comment is where a user can leave a non-code annotation in the source code. The entire line will be ignored by the parser.

			Note: only single line comments are supported.

Operators	Addition	x = 2 + 2
		y = + 2
			Adds two numbers.
			Can also be used in unary form.
			Implemented as built-in function **operator+**

	Subtract	x = 2 – 2
		y = - 4
			Subtracts two numbers.
			Can also be used in unary form.
			Implemented as built-in function **operator-**

	Multiplication	x = 5 * 2	Multiplies two numbers. Implemented as built-in function **operator***
	Division	x = 6 / 2	Divides two numbers. Implemented as built-in function **operator/**
	Modulus	x = 4 % 2	Performs modulus on two numbers. Implemented as built-in function **operator%**
	Exponent	x = 2 ^ 4	Performs exponent on two numbers. Implemented as built-in function **operator^**

	Logical Not	x = ! true
		y = not false
			Performs not operation on a Boolean.
			Tokens: **!** or **not**
			Implemented as built-in function **operator!**

	Less Than	x = 3 < 2	Performs less than comparison on two numbers. Implemented as built-in function **operator<**
	Less Than or Equal	x = 3 <= 2	Performs less than or equal comparison on two numbers. Implemented as built-in function **operator<=**
	Greater Than	x = 3 > 2	Performs greater than comparison on two numbers. Implemented as built-in function **operator>**
	Greater Than or Equal	x = 3 >= 2	Performs greater than or equal comparison on two numbers. Implemented as built-in function **operator>=**

	Equal	x = 2 == 2
		y = true == true
		z = “Test” == “Test”
			Compares two identical types for equality.
			Implemented as built-in function **operator==**

	Not Equal	x = 2 != 1
		y = true != false
		z = “Test1” != “Test2”
			Compares two identical types for inequality.
			Implemented as built-in function **operator!=**

	Logical And	x = true && true
		y = true and true
			Performs and operation on two Booleans.
			Tokens: **&&** or **and**
			Implemented as built-in function **operator&&**

	Logical Or	x = true || false
		y = true or false
			Performs or operation on two Booleans.
			Tokens: **||** or **or**
			Implemented as built-in function **operator||**

Assign Operators	Plus equals	x += 2	Assign operators are syntactic sugar for: **ID** = **ID OP** EXPR ID is the variable subject OP is the operator EXPR is the right-hand expression
	Minus equals	x -= 3	
	Multiply equals	x *= 2	
	Divide equals	x /= 2	
	Modulus equals	x %= 2	
	Exponent equals	x ^= 2	

[i] https://social.msdn.microsoft.com/Forums/en-US/2fd105f5-6314-4035-8001-50640bbeae30/using-debug-to-run-only-one-project-instead-of-all-in-a-solution?forum=aspvisualstudio