FlyingKoala

Release 0.0.5b0

Mar 21, 2020

Contents

CHAPTER 1

Install the library

The easiest way to install FlyingKoala is via pip:

```
pip install FlyingKoala
```

1.1 Dependencies

- `xlwings`, `koala2==0.0.31`, `pandas`, `numpy`

1.2 Optional Dependencies

The FlyingKoala supplied modules which have no extra dependencies.

- Horticulture
- Time Series

These packages are required for using some of the FlyingKoala modules.

- Accounting
  - pyopenxl
  - python-harvest-redux==5.0.0b0
- Energy
  - datetime
  - timezonefinder
  - pytz
  - pvlib

These packages are not required but highly recommended as they play very nicely with xlwings.

- Matplotlib
- Pillow/PIL

CHAPTER 2

Install the add-in

The trouble with installing this one is the element of "it depends". The punchline is the add-in needs to be placed where your add-ins go.

Until we can arrange a script that figures it out for us we will need to do it by hand.

Copy the addin\flyingkoala.xlam to;

Here: C:\Users\username\AppData\Roaming\Microsoft\AddIns

Or here: C:\Users\username\AppData\Roaming\Microsoft\Excel\XLSTART

Sometimes there's an XLSTART in your home directory.

It could well be somewhere else... (especially if you're on a Mac)

If you got through the add-in install and are following the example bouncing ball, next up is *Using FlyingKoala*.

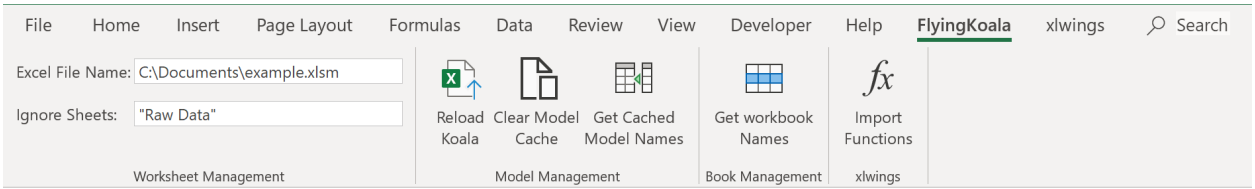
CHAPTER 3

Using FlyingKoala

This guide assumes you have FlyingKoala, the FlyingKoala add-in and xlwings already installed. If that's not the case, head over to *Install the library*.

3.1 1. The FlyingKoala Add-In

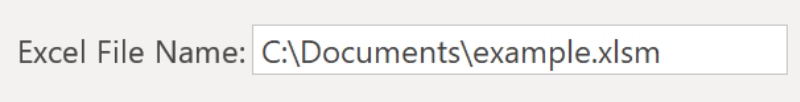
The FlyingKoala Add-In assists users to manage the cache which holds "models" (/equation systems). This is what it looks like.



3.1.1 Excel file name

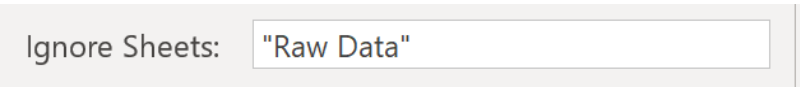
This is the name of the workbook file. This setting is expected to auto-fill.

In the future we expect to be able to load models from other Excel files which is why this configuration is here.



3.1.2 Ignore Sheets

These are worksheets you would like to not load into the koala cache. Most likely used for worksheets which have your raw data, or worksheets which are not currently being used in the modelling you're working on.



3.1.3 Reload Koala

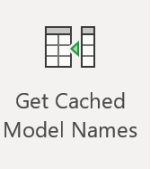
FlyingKoala caches parts of a specified spreadsheet (eg; the one named in "Excel config name"). To extract the part you are interested in, first Koala needs to have a look at the spreadsheet you are interested in. This button does that initial loading. If the spreadsheet has been loaded, this button will load it again. Great for when a formula in the spreadsheet has changed.

3.1.4 Clear Model Cache

This button clears the FlyingKoala cache of all the loaded models. Handy if you want to re-load some of the equations.

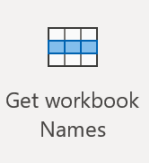
3.1.5 Get Cached Model Names

A message box will appear naming all the models which have been loaded into the FlyingKoala model cache.



3.1.6 Get workbook Names

A message box will appear naming all the named ranges xlwings has access to in the loaded workbook. This helps you check spelling on some of the named ranges. This list can also be seen, and managed, in the Formulas ==> Name Manager menu of Excel.

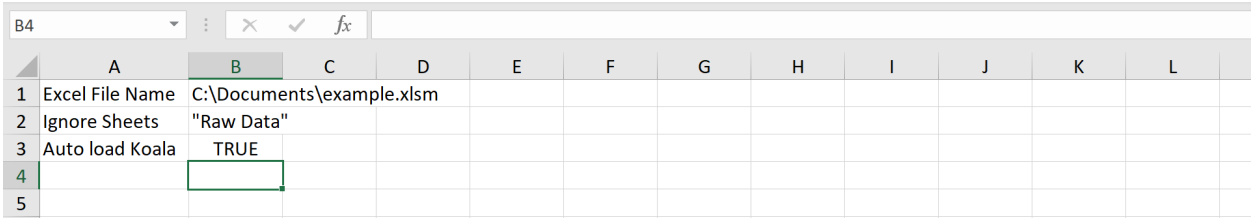


3.1.7 Import Functions

This is an convenience button. It calls the xlwings UDF Import Functions method. It saves having to move between Add-Ins while developing UDFs.

3.2 2. The FlyingKoala Configuration

The FlyingKoala configuration worksheet assists users to manage the relationship between koala and xlwings. It must be named FlyingKoala.conf and this is what it looks like.



Currently the config management uses fixed cell references, so don't move anything. (**TODO: open for contribution. There's an example in xlwings for inspiration.**)

This is not kept in an external file as the FlyingKoala operations are generally workbook specific. You are likely to want to have this workbook behave in a particular way – especially when someone opens the file and re-calc while they are using it.

3.2.1 Excel file name

This is the name of the workbook file. This setting is expected to auto-fill, but will also be over-written by whatever gets put in the corresponding field in the Add-In.

3.2.2 Ignore Sheets

These are a comma delimited list of worksheets you want to have Koala ignore when it loads your spreadsheet into cache. If there is a space in the worksheet name you'll need to use double quotes.

3.2.3 Auto load Koala

This will allow or deny xlwings the ability to load the workbook when you load UDFs. Basically, it's telling xlwings that when the Python interpreter service starts it's allowed to initialise Koala or not.

- TRUE: the spreadsheet will be loaded into Koala when xlwings interpreter service starts.
- FALSE: The spreadsheet will _not_ be loaded into Koala when xlwings interpreter service starts.

3.3 3. Using the FlyingKoala User Defined Functions

Providing the User Defined Functions (UDFs) you are expecting to use already exist in the FlyingKoala library you can simply import them. They won't be loaded at this point, but will become available for use in Excel like any other formula. An equation will become loaded into Koala cache as a model when you use it.

Make sure you have the dependencies installed for the FlyingKoala module you want to use. Notes on this can be found in *Install the library*.

For calculating Growing Degree-Days you would have a module that would look like this:

```
import xlwings as xw
from flyingkoala import *
from flyingkoala.horticulture import *
```

For doing some time series transformation you would have a module that would look like this:

```
import xlwings as xw
from flyingkoala import *
from flyingkoala.timeseries import *
```

For doing some time series transformation while calculating Growing Degrees-Day you would have a module that would look like this:

```
import xlwings as xw
from flyingkoala import *
from flyingkoala.horticulture import *
from flyingkoala.timeseries import *
```

3.4 4. Using the FlyingKoala VBA macros

Providing the macro functions you are expecting to use already exist in the FlyingKoala library they will be installed with FlyingKoala.

You'll want to be familiar with writing VBA and reading API style documentation for this one.

Once everything is installed correctly you can call the FlyingKoala VBA functions as you normally do with VBA.

3.5 5. Freestyling with FlyingKoala

If the functionality you want isn't yet supported in a FlyingKoala module, you'll need to write your own (and maybe put it forward to be included in FlyingKoala :D).

This is the advanced approach. It has a rather steep curve. Once you 'get it' things aren't terrible, but I do admit there are a lot of moving parts. The worked example in the example document Introduction_Article.PDF is a great resource.

From here on you'll want to be an particularly familiar with;

- writing Python
- writing xlwings UDFs
- using Excel named ranges, named cells and Excel's Manage Ranges feature

Strap in... Here we go!

At the very core FlyingKoala offers an xlwings friendly interface to Koala2.

Koala2 is a project which can read an MS Excel equation, convert it into Python and then evaluate (run) the Python to produce a result for the equation. FlyingKoala caches the Python code generated for each MS Excel equation and so we can change out the definition of an equation on each call to a function.

To take advantage of the FlyingKoala interface to Koala2 we need to write a function which takes at least two types of argument:

- named range name
- terms for an equation

The named range name is simply a string which identifies the name of a named range containing an equation you want interpreted into Python. This is what the equation will have been keyed on in the cache.

Terms for an equation are variables which will be used while evaluating an equation.

Let us work through writing a Python function taking advantage of FlyingKoala's interface with Koala2 using Growing Degrees Day as the example.

Growing Degrees Day calculations are often specific to a particular application. Potentially differing on region, variety of plants or a number of other influences. When we look at Wikipedia we can find no less than two different equations.

$$GDD = \frac{T_{\text{max}} + T_{\text{min}}}{2} - T_{\text{base}}$$

and

$$GDD = \max\left(\frac{T_{\text{max}} + T_{\text{min}}}{2} - T_{\text{base}}\,,\, 0\right)$$

Consider a situation where we want to do some modelling of those two documented examples and, maybe, develop another method to optimize for our specific situation. It would be great to simply write the equations in Excel while we are running scenarios and developing the new one.

We can see in the above that even though the relationship between the terms is quite different, calculating a Growing Degree Day appears to require the same number of inputs, namely T_min, T_max and T_base. The rest of the relationship is in operators and hard constants. This is an observation which often holds up while developing equations... You only have, or need, particular inputs to find the answer to something and if there is another pathway to the answer it is likely to still only need the same inputs (often enough they are the only inputs you *can get*).

The truly variable terms in the Growing Degree Day equations are T_min and T_max. To obviate some Wikipedia reading, T_base is usually set to a value of 10. This makes T_base a soft constant. One you *might* change but are likely not to. We want to support an ability to change it. Remember, we are trying to invent new mathematics so if there's a knob - we need to be able to turn it.

With that understanding, we can now write a Python function which takes a key for the cache (a named range name) and two parameters being T_min and T_max:

```
import numpy as np
import pandas as pd

from flyingkoala import flyingkoala as fk

def DegreeDay(model, T_min, T_max):
    """Function for calculating Degree Day"""

    inputs_for_DegreeDay = pd.DataFrame({'T_min': np.array([T_min]), 'T_max': np.
↪array([T_max])})
    return fk.EvaluateKoalaModel(model.name.name, inputs_for_DegreeDay)
```

This function is not finished. It still needs xlwings mark-up to become a User Defined Function. But we can see here that there is nothing genuinely complex about taking three arguments, packaging two into a Pandas Dataframe and then calling EvaluateKoalaModel.

If we put all the xlwings markup on the above Python function, we can import it as a User Defined Function:

```
@xw.func
@xw.arg('model', xw.Range, doc='Name, as a string, of the model which will be␣
↪evaluated. The Excel cell name / named range')
@xw.arg('T_min', np.array, doc='Daily minimum temperature')
@xw.arg('T_max', np.array, doc='Daily maximum temperature')
@xw.ret(index=False, header=False)
def DegreeDay(model, T_min, T_max):
    """Function to assemble a dataframe for calculating Degree Day"""

    if not fk.isKoalaModelCached(model.name.name):
        fk.generateModelGraph(model)

    inputs_for_DegreeDay = pd.DataFrame({'T_min': np.array([T_min]), 'T_max': np.
↪array([T_max])})
    return fk.EvaluateKoalaModel(model.name.name, inputs_for_DegreeDay)
```

This is essentially all a developer or appropriately skilled data analyst needs to do. The rest is up to the domain expert as it is a case of setting Excel up correctly and then defining the mathematical relationship.

NOTE: The mathematical relationship for the equation is not expressed in Python yet. The definition is the responsibility of the domain expert to define in an Excel formula and Koala2 to manage running that definition in Python.

Although setting Excel up is demonstrated in the worked example in the example document Introduction_Article.PDF I'll run through it briefly here.

To set Excel up...

Define named cells for each of the terms. The keys in the anonymous dict which is used to create inputs_for_DegreeDay need to be the same as the names of the named ranges (cell names) that will be used in the Excel formula.

To be specific there needs to be an Excel cell named 'T_min', another called 'T_max'. These names need to be global (can be identified in the Manage Names menu of Excel). These cells need to be referenced in the Excel equation - **NOT** the cell address.

Now we can write an MS Excel formula which will define the relationship between T_min, T_max and T_base. This is the Excel formula for the first GDD equation and is in a cell called Equation_1:

=((T_max+T_min)/2)-T_base

NOTE: we have used the cell names, not the cell address eg; T_max **not** GDD_formula!B5

Providing the DegreeDay UDF definition is defined, we can use it in Excel:

=DegreeDay(Equation_1, 3, 25)

That Excel equation will grab the equation from the call called Equation_1 which is =((T_max+T_min)/2)-T_base, convert the equation to Python, set T_min to 3, T_max to 25, T_base to 10, evaluate the result and return it as a value to the cell.

CHAPTER 4

API

4.1 User Defined Functions

These are the User Defined Functions UDFs that come with FlyingKoala.

4.2 VBA Macros

These are the RunPython macros that come with FlyingKoala.

They don't work "out of the box" yet – I've not written the plug-in but would work if you wrote your own RunPython calls.

CHAPTER 5

Worked Example - Horticulture

The worked example for the horticulture module demonstrates the integration of xlwings and koala libraries through User Defined Functions (UDFs) designed for a given calculation which may have multiple definitions.

The horticulture module supports the calculation of Growing Degree Days. For those who have not come across this kind of calculation before;

> Growing degree days (GDD) is a weather-based indicator for assessing crop development. It is a calculation used by crop producers that is a measure of heat accumulation used to predict plant and pest development rates such as the date that a crop reaches maturity.
>
> The link

The purpose of a growing degree day is essentially irrelevant for the illustration. What is important is there are many ways of calculating a growing degree day, and that the math is usually quite simple. We can see the growing degree days article on Wikipedia defines two different methods of calculating a growing degree day.

$$GDD = \max\left(\frac{T_{\max} + T_{\min}}{2} - T_{\text{base}}\ ,\ 0\right)$$

and

$$GDD = \frac{T_{\max} + T_{\min}}{2} - T_{\text{base}}$$

It is possible to have more than two definitions to calculate a growing degree day, but for this example two is plenty.

In the case of growing degree days, the integration at the center of FlyingKoala provides the ability to use any definition with two variable terms as the formula for a growing degree day.

The User Defined Function signature for a growing degree day:

```
=DegreeDay(model_name, T_min, T_max)
```

We can see the signature requires the name of a model and two terms. We will get to the concept of a model in a moment but the two terms are easy to understand as they are expressed in the above formulae. Irrespective of why, the example growing degree days formulae both use two.

The actual Python code behind that UDF;

```
import numpy as np
import pandas as pd

from flyingkoala import flyingkoala as fk

def DegreeDay(model, T_min, T_max):
    """Function for calculating Degree Day"""

    inputs_for_DegreeDay = pd.DataFrame({'T_min': np.array([T_min]), 'T_max': np.
↪array([T_max])})
    return fk.EvaluateKoalaModel(model.name.name, inputs_for_DegreeDay)
```

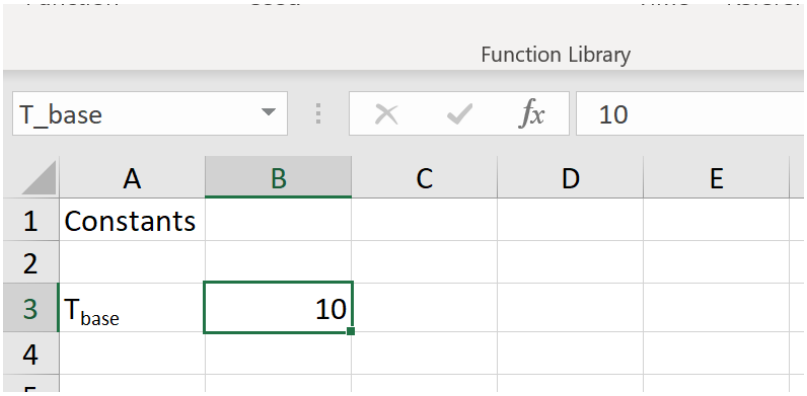
In the above code there is no mention of any mathematical relationship between T_min, T_max or the un-mentioned T_base. So how does the call to the UDF do *any* calculation? This is where the FlyingKoala 'magic' comes in. We need to define the math in an Excel formula.

It's best if the definitions of terms and equations (/models) are named ranges, sometimes called named cells.

And although you can put these elements anywhere in a workbook, I have found it useful to organize worksheets for things like constants, formulas, raw data and workings.

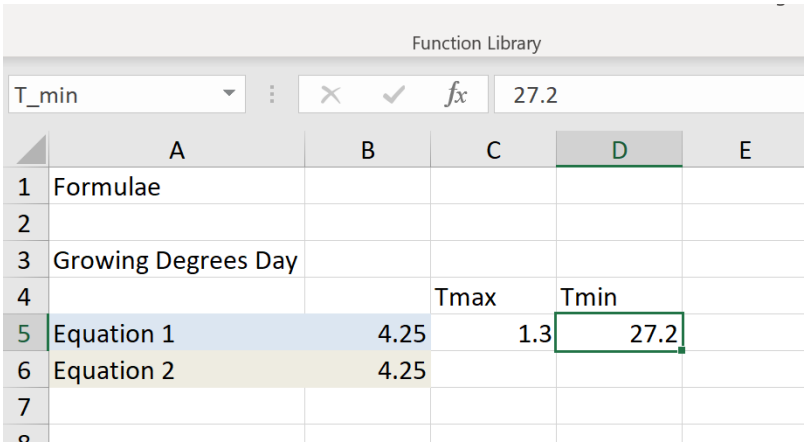
First up define the 'constant' terms. Constant is in quotes as these are values which have a typical value, but *may* change. As the adage goes; 'this value cannot change because {of a good reason}' and the moment it's fixed it has the need to change.

In our case this is the base temperature, T_{base}. Usually 10 Degrees Celsius but doesn't necessarily need to be.

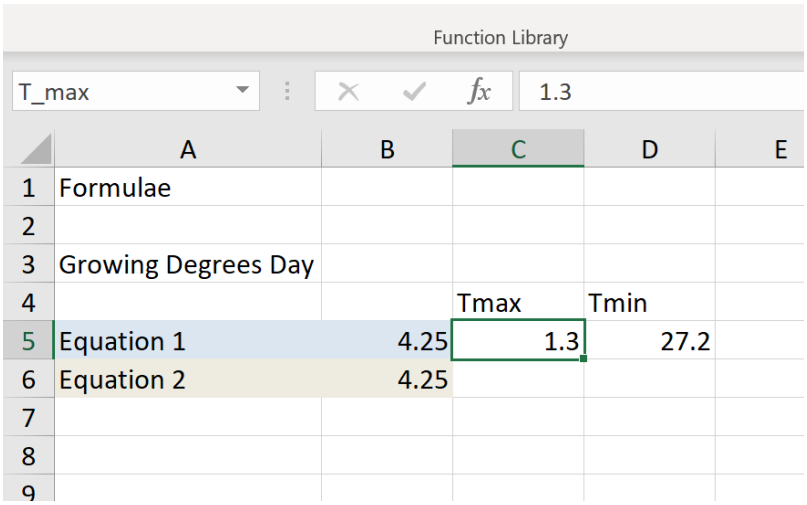


Next up are the equation's variable terms. These are assured to change.

Minimum temperature for a day, T_{min}



Maximum temperature for a day, T_{max}

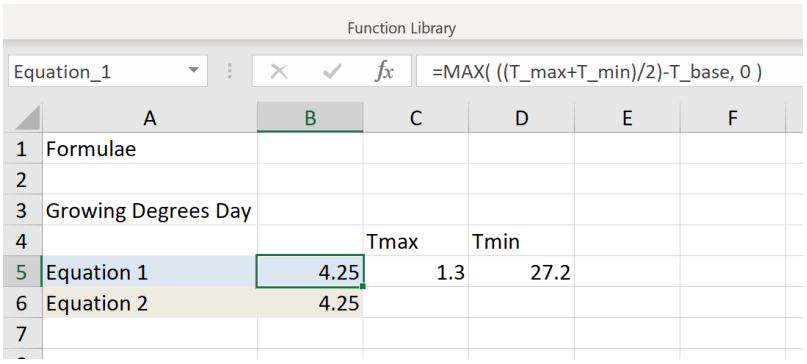


It can be noted that both growing degree days formulae have the same number of terms. This may not be a coincidence. Much of the time a calculation to a particular end only needs a certain number of terms. In other cases the terms used are the only ones available and so, by implication, the number of them is unlikely to change.

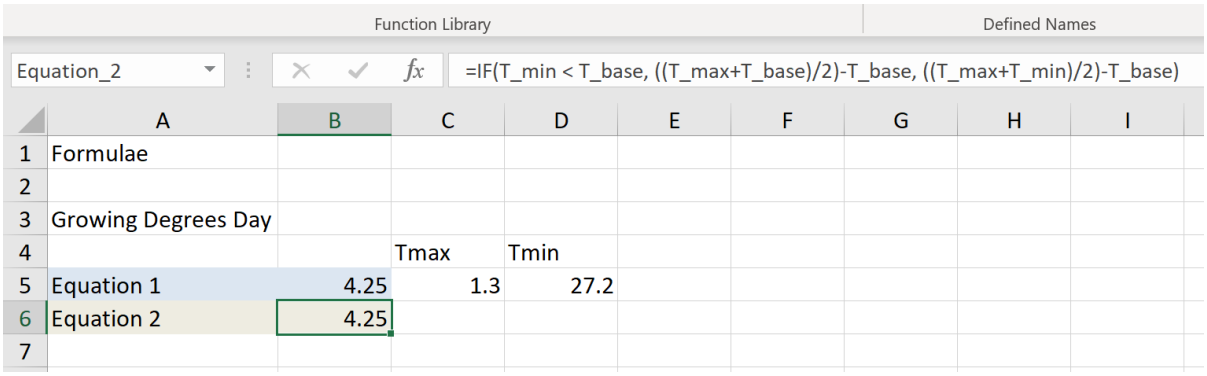
Now that we have named range definitions for each of the terms in the equation, it's time to define the equations.

I had no good way of labelling these equations so have simply called them Equation_1 and Equation_2.

This is Equation_1.



And this is Equation_2.



We can see the use of the named ranges for both the constant and variable terms enhances the expression of the formula. Another advantage of setting the formula up this way is that you can put values in to test the formula. It is a

little labour intensive, but you can use this to calculate values which check your formula expression.

We are now at the point of using these formulas.

A growing degree day value is not much good on its own as it is the sum of them which becomes useful.

For obvious reasons daily temperature data is most often expressed as a time series. **But**, there is more than one way to tackle this;

- Fill-down on a formula
- Use a Dynamic Array

For the above reasons,

- DegreeDay() UDFs T_{min} and T_{max} require values or cells (returns a single value) and
- DegreeDayDynamicArray() UDFs T_{min} and T_{max} are each cell ranges (returns a Dynamic Array).

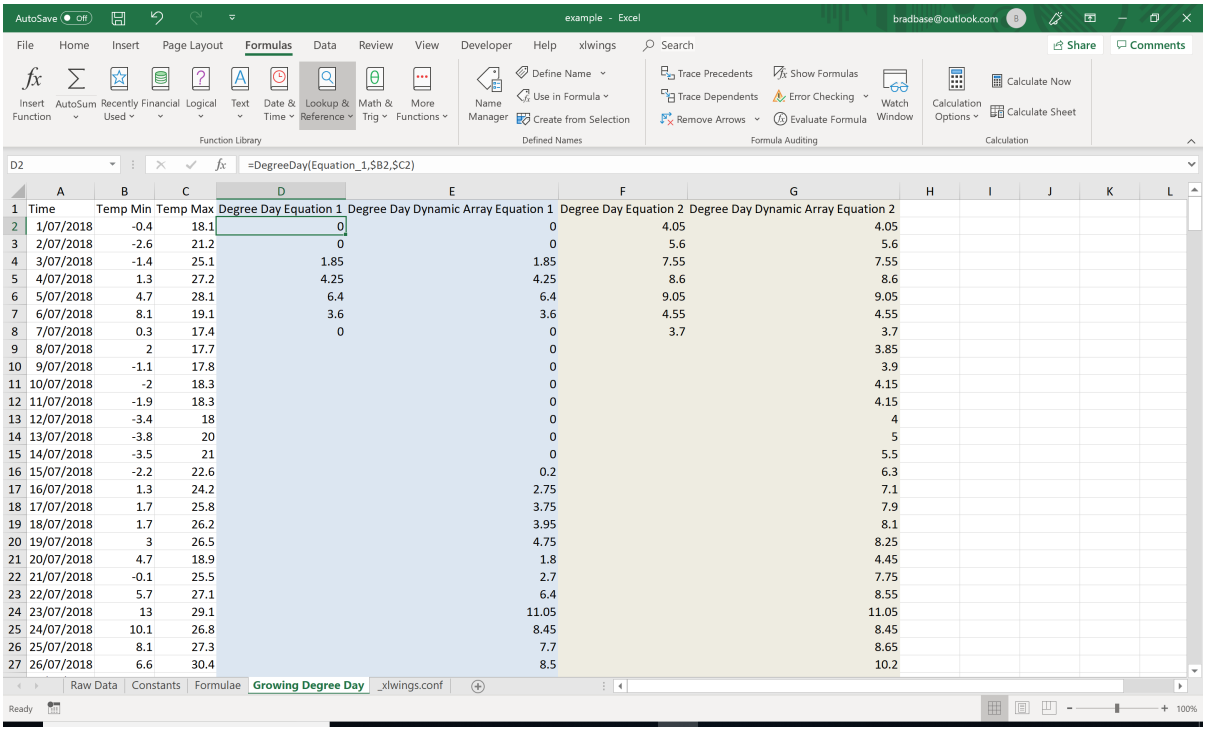
For the Fill-down approach

```
=DegreeDay(model_name, T_min, T_max)
```

can now become something like

```
=DegreeDay(Equation_1, $B2, $C2)
```

And so we can see the workings;



	A	B	C	D	E	F	G
1	Time	Temp Min	Temp Max	Degree Day Equation 1	Degree Day Dynamic Array Equation 1	Degree Day Equation 2	Degree Day Dynamic Array Equation 2
2	1/07/2018	-0.4	18.1	0	0	4.05	4.05
3	2/07/2018	-2.6	21.2	0	0	5.6	5.6
4	3/07/2018	-1.4	25.1	1.85	1.85	7.55	7.55
5	4/07/2018	1.3	27.2	4.25	4.25	8.6	8.6
6	5/07/2018	4.7	28.1	6.4	6.4	9.05	9.05
7	6/07/2018	8.1	19.1	3.6	3.6	4.55	4.55
8	7/07/2018	0.3	17.4	0	0	3.7	3.7
9	8/07/2018	2	17.7		0		3.85
10	9/07/2018	-1.1	17.8		0		3.9
11	10/07/2018	-2	18.3		0		4.15
12	11/07/2018	-1.9	18.3		0		4.15
13	12/07/2018	-3.4	18		0		4
14	13/07/2018	-3.8	20		0		5
15	14/07/2018	-3.5	21		0		5.5
16	15/07/2018	-2.2	22.6		0.2		6.3
17	16/07/2018	1.3	24.2		2.75		7.1
18	17/07/2018	1.7	25.8		3.75		7.9
19	18/07/2018	1.7	26.2		3.95		8.1
20	19/07/2018	3	26.5		4.75		8.25
21	20/07/2018	4.7	18.9		1.8		4.45
22	21/07/2018	-0.1	25.5		2.7		7.75
23	22/07/2018	5.7	27.1		6.4		8.55
24	23/07/2018	13	29.1		11.05		11.05
25	24/07/2018	10.1	26.8		8.45		8.45
26	25/07/2018	8.1	27.3		7.7		8.65
27	26/07/2018	6.6	30.4		8.5		10.2

In the above example when DegreeDay first gets called it will;

- start a UDF server which has a Python interpreter
- load Equation_1 as a model in the FlyingKoala cache while specifying Equation_1 as the output cell and T_min and T_max cells as input
- apply values in $B2 and $C2 to T_min and T_max respectively
- run the Python code which actually calculates the 'answer'/result

- return the result

For subsequent calls, which includes a workbook or worksheet re-calc, DegreeDay will;

- get Equation_1 from the FlyingKoala cache
- apply values in $B2 and $C2 to T_min and T_max respectively
- run the Python code which actually calculates the 'answer'/result
- return the result

Obviously, a fill-down will change the row index like it would in any Excel formula. . .

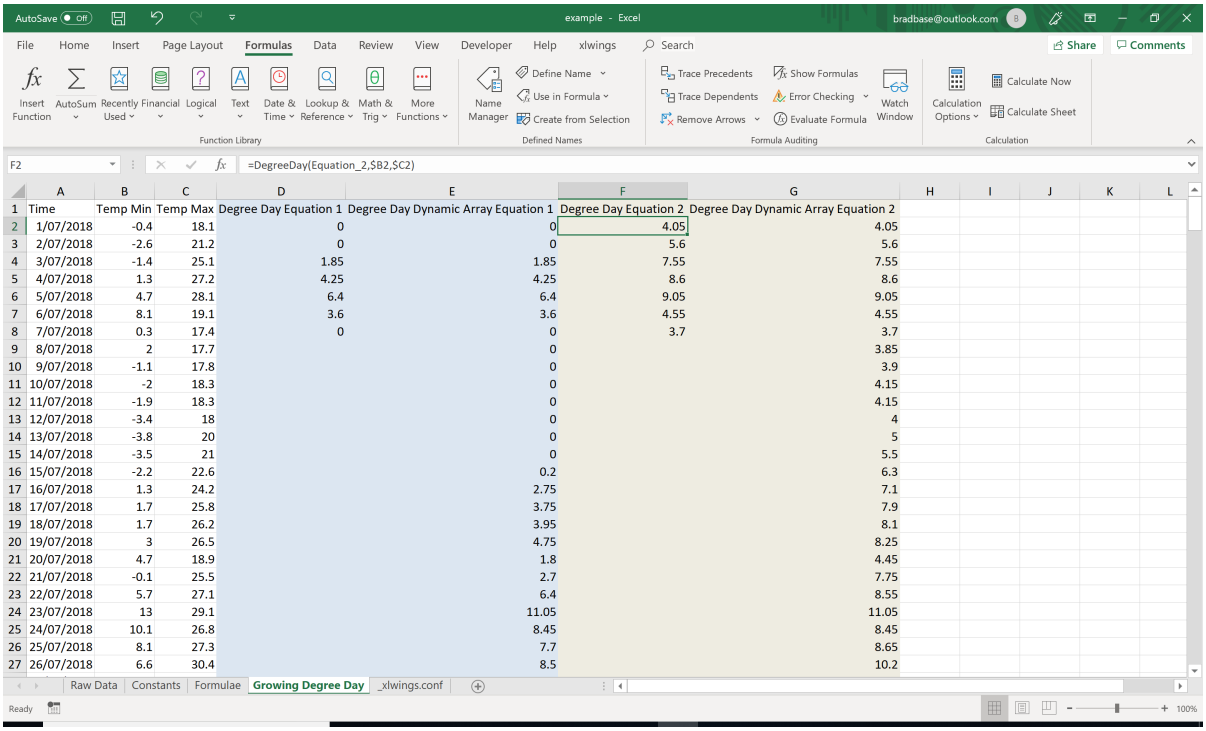
=DegreeDay(Equation_1, $B2, $C2)
=DegreeDay(Equation_1, $B3, $C3)
=DegreeDay(Equation_1, $B4, $C4)
. . .

That's great for Equation 1. But what about Equation_2..? That's easy – Still with the Fill-down approach;

=DegreeDay(model_name, T_min, T_max)

can now become something like

=DegreeDay(Equation_2, $B2, $C2)



	A	B	C	D	E	F	G
1	Time	Temp Min	Temp Max	Degree Day Equation 1	Degree Day Dynamic Array Equation 1	Degree Day Equation 2	Degree Day Dynamic Array Equation 2
2	1/07/2018	-0.4	18.1	0	0	4.05	4.05
3	2/07/2018	-2.6	21.2	0	0	5.6	5.6
4	3/07/2018	-1.4	25.1	1.85	1.85	7.55	7.55
5	4/07/2018	1.3	27.2	4.25	4.25	8.6	8.6
6	5/07/2018	4.7	28.1	6.4	6.4	9.05	9.05
7	6/07/2018	8.1	19.1	3.6	3.6	4.55	4.55
8	7/07/2018	0.3	17.4	0	0	3.7	3.7
9	8/07/2018	2	17.7		0		3.85
10	9/07/2018	-1.1	17.8		0		3.9
11	10/07/2018	-2	18.3		0		4.15
12	11/07/2018	-1.9	18.3		0		4.15
13	12/07/2018	-3.4	18		0		4
14	13/07/2018	-3.8	20		0		5
15	14/07/2018	-3.5	21		0		5.5
16	15/07/2018	-2.2	22.6		0.2		6.3
17	16/07/2018	1.3	24.2		2.75		7.1
18	17/07/2018	1.7	25.8		3.75		7.9
19	18/07/2018	1.7	26.2		3.95		8.1
20	19/07/2018	3	26.5		4.75		8.25
21	20/07/2018	4.7	18.9		1.8		4.45
22	21/07/2018	-0.1	25.5		2.7		7.75
23	22/07/2018	5.7	27.1		6.4		8.55
24	23/07/2018	13	29.1		11.05		11.05
25	24/07/2018	10.1	26.8		8.45		8.45
26	25/07/2018	8.1	27.3		7.7		8.65
27	26/07/2018	6.6	30.4		8.5		10.2

The fill-down approach is awesome. But as you try and do larger and larger time series it becomes quite cumbersome. Each filled cell calculation needs to do a full round-trip from Excel to Python, get evaluated, and return a result from Python to Excel. All the data conversion in that takes time.

The solution to this round-trip per time period in the time series is to do the calculation 'in bulk'. Send a range of cells for T_min and T_max, run the calculations on the array(/s) and send a resulting range back. Enter the Dynamic Array.

Dynamic Arrays are part of Excel but are going to help us a great deal when it comes to optimizing series calculation.

For the Dynamic Array approach

=DegreeDayDynamicArray(model_name, T_min, T_max)

can now become something like

=DegreeDayDynamicArray(Equation_1, $B2:$B366, $C2:$C366)

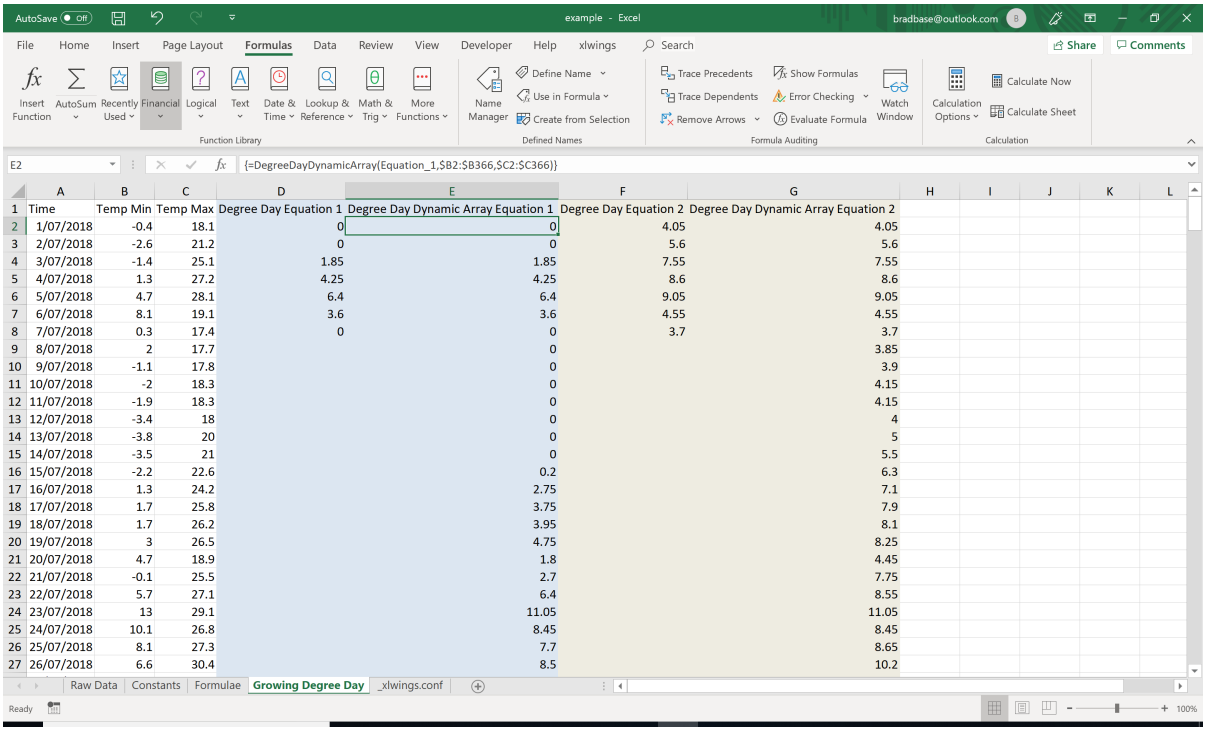
The ranges for T_min and T_max **must** be the same shape. eg; they need to have the same number of elements else you'll get an error. They don't need to be ranges next to each other. Knowing this, the below can be understood as a valid expression;

=DegreeDayDynamicArray(Equation_1, $B2:$B366, $E2:$E366)

Although I have no idea why this would be wanted, it is valid;

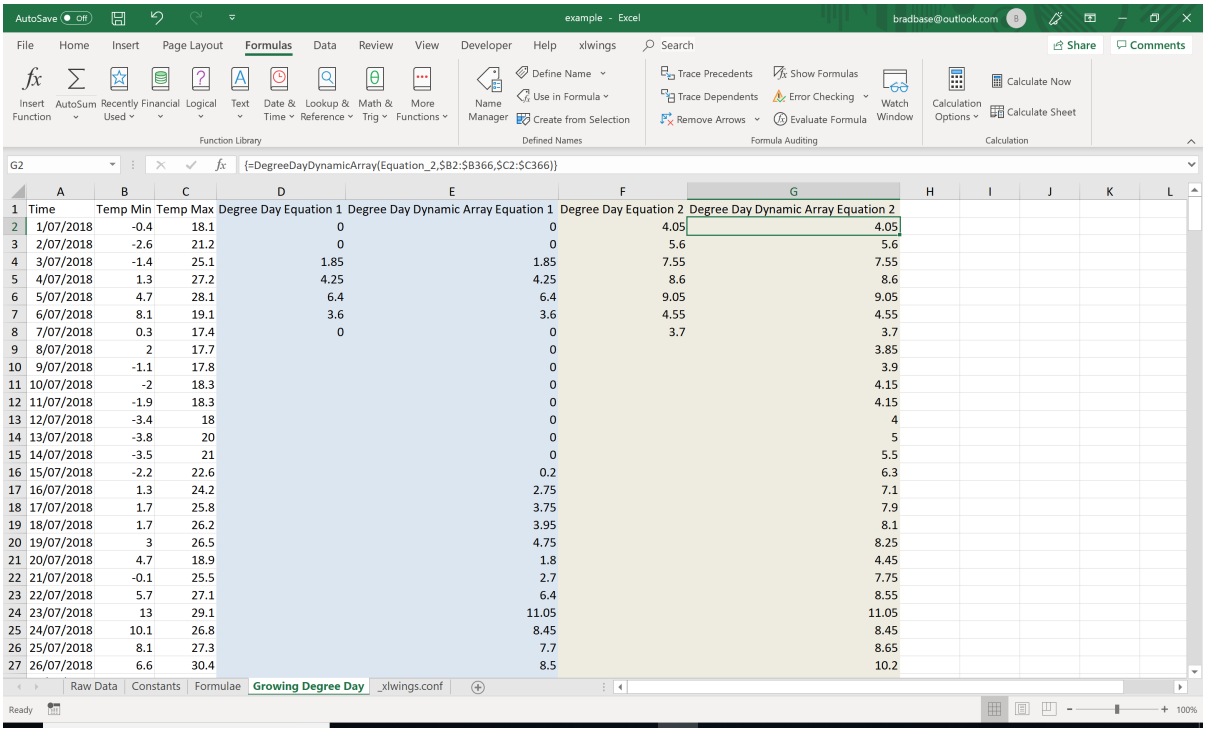
=DegreeDayDynamicArray(Equation_1, $B2:$B366, $G5:$G369)

And so we can see the workings for the vanilla case Equation_1;



	A	B	C	D	E	F	G
1	Time	Temp Min	Temp Max	Degree Day Equation 1	Degree Day Dynamic Array Equation 1	Degree Day Equation 2	Degree Day Dynamic Array Equation 2
2	1/07/2018	-0.4	18.1	0	0	4.05	4.05
3	2/07/2018	-2.6	21.2	0	0	5.6	5.6
4	3/07/2018	-1.4	25.1	1.85	1.85	7.55	7.55
5	4/07/2018	1.3	27.2	4.25	4.25	8.6	8.6
6	5/07/2018	4.7	28.1	6.4	6.4	9.05	9.05
7	6/07/2018	8.1	19.1	3.6	3.6	4.55	4.55
8	7/07/2018	0.3	17.4	0	0	3.7	3.7
9	8/07/2018	2	17.7		0		3.85
10	9/07/2018	-1.1	17.8		0		3.9
11	10/07/2018	-2	18.3		0		4.15
12	11/07/2018	-1.9	18.3		0		4.15
13	12/07/2018	-3.4	18		0		4
14	13/07/2018	-3.8	20		0		5
15	14/07/2018	-3.5	21		0		5.5
16	15/07/2018	-2.2	22.6		0.2		6.3
17	16/07/2018	1.3	24.2		2.75		7.1
18	17/07/2018	1.7	25.8		3.75		7.9
19	18/07/2018	1.7	26.2		3.95		8.1
20	19/07/2018	3	26.5		4.75		8.25
21	20/07/2018	4.7	18.9		1.8		4.45
22	21/07/2018	-0.1	25.5		2.7		7.75
23	22/07/2018	5.7	27.1		6.4		8.55
24	23/07/2018	13	29.1		11.05		11.05
25	24/07/2018	10.1	26.8		8.45		8.45
26	25/07/2018	8.1	27.3		7.7		8.65
27	26/07/2018	6.6	30.4		8.5		10.2

And for Equation_2



	A	B	C	D	E	F	G
1	Time	Temp Min	Temp Max	Degree Day Equation 1	Degree Day Dynamic Array Equation 1	Degree Day Equation 2	Degree Day Dynamic Array Equation 2
2	1/07/2018	-0.4	18.1	0	0	4.05	4.05
3	2/07/2018	-2.6	21.2	0	0	5.6	5.6
4	3/07/2018	-1.4	25.1	1.85	1.85	7.55	7.55
5	4/07/2018	1.3	27.2	4.25	4.25	8.6	8.6
6	5/07/2018	4.7	28.1	6.4	6.4	9.05	9.05
7	6/07/2018	8.1	19.1	3.6	3.6	4.55	4.55
8	7/07/2018	0.3	17.4	0	0	3.7	3.7
9	8/07/2018	2	17.7		0		3.85
10	9/07/2018	-1.1	17.8		0		3.9
11	10/07/2018	-2	18.3		0		4.15
12	11/07/2018	-1.9	18.3		0		4.15
13	12/07/2018	-3.4	18		0		4
14	13/07/2018	-3.8	20		0		5
15	14/07/2018	-3.5	21		0		5.5
16	15/07/2018	-2.2	22.6		0.2		6.3
17	16/07/2018	1.3	24.2		2.75		7.1
18	17/07/2018	1.7	25.8		3.75		7.9
19	18/07/2018	1.7	26.2		3.95		8.1
20	19/07/2018	3	26.5		4.75		8.25
21	20/07/2018	4.7	18.9		1.8		4.45
22	21/07/2018	-0.1	25.5		2.7		7.75
23	22/07/2018	5.7	27.1		6.4		8.55
24	23/07/2018	13	29.1		11.05		11.05
25	24/07/2018	10.1	26.8		8.45		8.45
26	25/07/2018	8.1	27.3		7.7		8.65
27	26/07/2018	6.6	30.4		8.5		10.2

There is one more feature of these UDFs which is quite valuable and it stems from the fact that the model_name argument in either DegreeDay or DegreeDayDynamicArray is a range.

It can be defined as a variable.

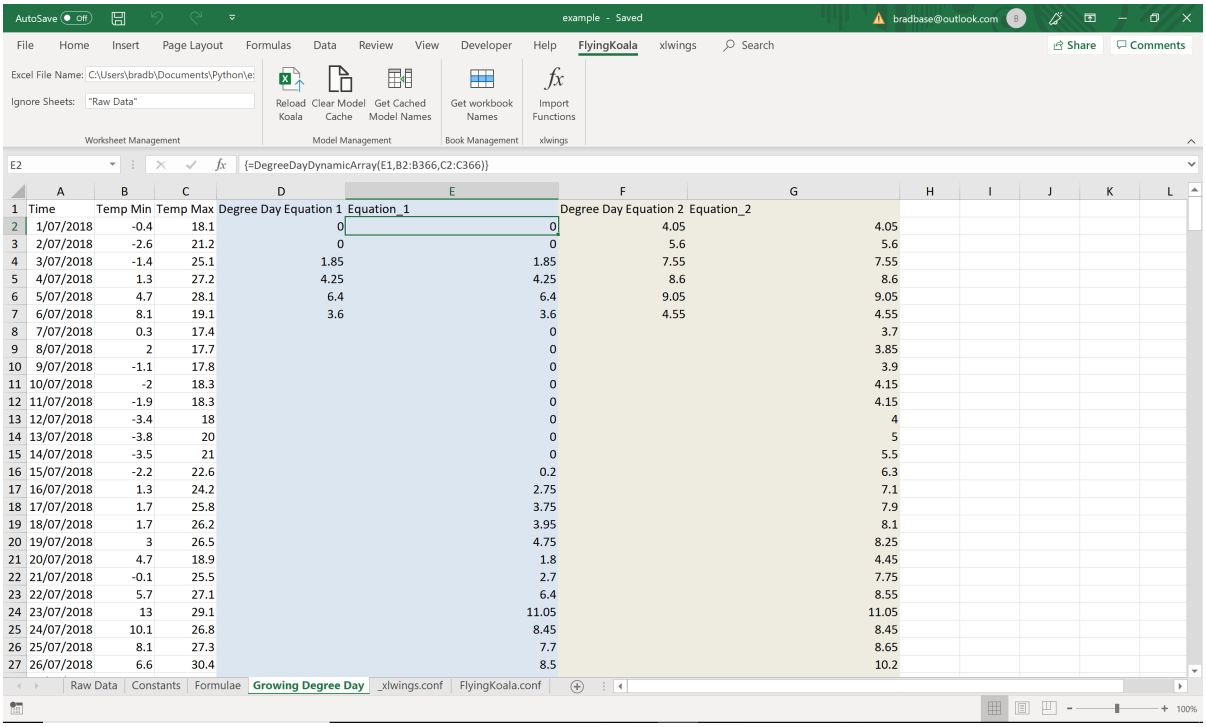
To take the vanilla example for Equation_1 above;

=DegreeDayDynamicArray(Equation_1, $B2:$B366, $C2:$C366)

may become

=DegreeDayDynamicArray(E1, $B2:$B366, $C2:$C366)

which defines the name of the column as the model name which the column is using.



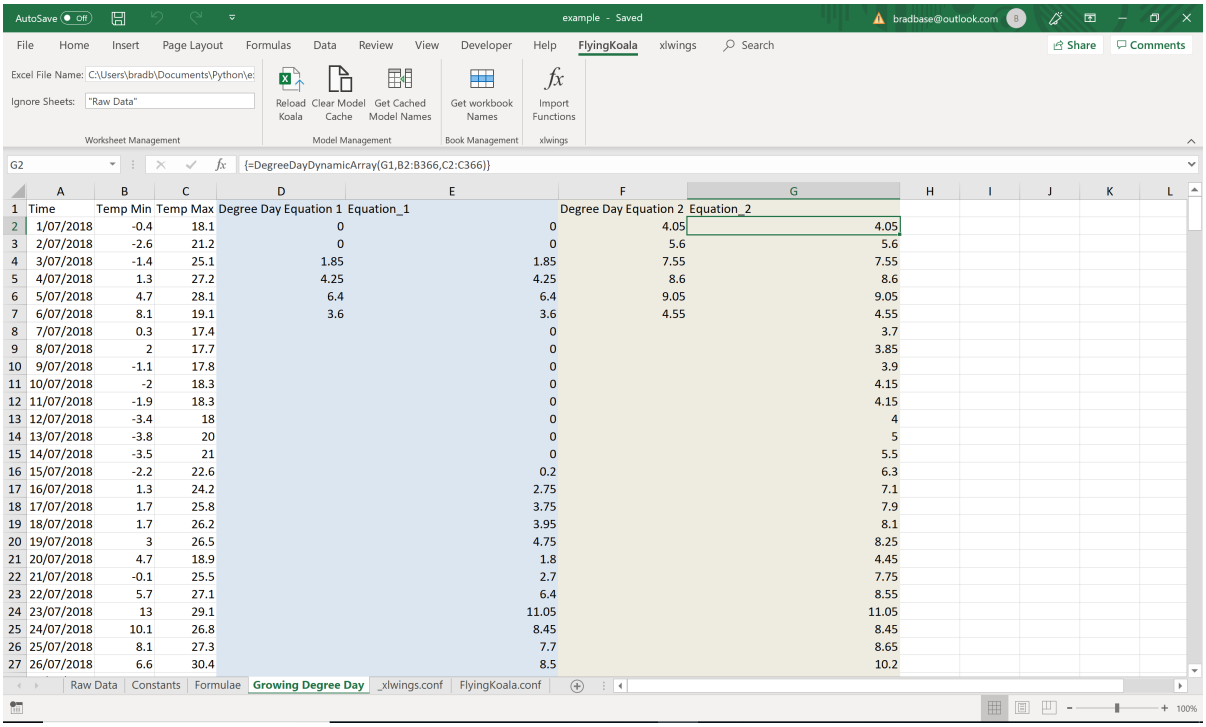
	A	B	C	D	E	F	G
1	Time	Temp Min	Temp Max	Degree Day Equation 1	Equation_1	Degree Day Equation 2	Equation_2
2	1/07/2018	-0.4	18.1	0	0	4.05	4.05
3	2/07/2018	-2.6	21.2	0	0	5.6	5.6
4	3/07/2018	-1.4	25.1	1.85	1.85	7.55	7.55
5	4/07/2018	1.3	27.2	4.25	4.25	8.6	8.6
6	5/07/2018	4.7	28.1	6.4	6.4	9.05	9.05
7	6/07/2018	8.1	19.1	3.6	3.6	4.55	4.55
8	7/07/2018	0.3	17.4		0		3.7
9	8/07/2018	2	17.7		0		3.85
10	9/07/2018	-1.1	17.8		0		3.9
11	10/07/2018	-2	18.3		0		4.15
12	11/07/2018	-1.9	18.3		0		4.15
13	12/07/2018	-3.4	18		0		4
14	13/07/2018	-3.8	20		0		5
15	14/07/2018	-3.5	21		0		5.5
16	15/07/2018	-2.2	22.6		0.2		6.3
17	16/07/2018	1.3	24.2		2.75		7.1
18	17/07/2018	1.7	25.8		3.75		7.9
19	18/07/2018	1.7	26.2		3.95		8.1
20	19/07/2018	3	26.5		4.75		8.25
21	20/07/2018	4.7	18.9		1.8		4.45
22	21/07/2018	-0.1	25.5		2.7		7.75
23	22/07/2018	5.7	27.1		6.4		8.55
24	23/07/2018	13	29.1		11.05		11.05
25	24/07/2018	10.1	26.8		8.45		8.45
26	25/07/2018	8.1	27.3		7.7		8.65
27	26/07/2018	6.6	30.4		8.5		10.2

And, of course, same for Equation_2

```
=DegreeDayDynamicArray(Equation_2, $B2:$B366, $C2:$C366)
```

may become

```
=DegreeDayDynamicArray($G$1, $B2:$B366, $C2:$C366)
```



	A	B	C	D	E	F	G
1	Time	Temp Min	Temp Max	Degree Day Equation 1	Equation_1	Degree Day Equation 2	Equation_2
2	1/07/2018	-0.4	18.1	0	0	4.05	4.05
3	2/07/2018	-2.6	21.2	0	0	5.6	5.6
4	3/07/2018	-1.4	25.1	1.85	1.85	7.55	7.55
5	4/07/2018	1.3	27.2	4.25	4.25	8.6	8.6
6	5/07/2018	4.7	28.1	6.4	6.4	9.05	9.05
7	6/07/2018	8.1	19.1	3.6	3.6	4.55	4.55
8	7/07/2018	0.3	17.4		0		3.7
9	8/07/2018	2	17.7		0		3.85
10	9/07/2018	-1.1	17.8		0		3.9
11	10/07/2018	-2	18.3		0		4.15
12	11/07/2018	-1.9	18.3		0		4.15
13	12/07/2018	-3.4	18		0		4
14	13/07/2018	-3.8	20		0		5
15	14/07/2018	-3.5	21		0		5.5
16	15/07/2018	-2.2	22.6		0.2		6.3
17	16/07/2018	1.3	24.2		2.75		7.1
18	17/07/2018	1.7	25.8		3.75		7.9
19	18/07/2018	1.7	26.2		3.95		8.1
20	19/07/2018	3	26.5		4.75		8.25
21	20/07/2018	4.7	18.9		1.8		4.45
22	21/07/2018	-0.1	25.5		2.7		7.75
23	22/07/2018	5.7	27.1		6.4		8.55
24	23/07/2018	13	29.1		11.05		11.05
25	24/07/2018	10.1	26.8		8.45		8.45
26	25/07/2018	8.1	27.3		7.7		8.65
27	26/07/2018	6.6	30.4		8.5		10.2

It is also valid to define yet another named range

=DegreeDayDynamicArray(active_degree_day_model, $B2:$B366, $C2:$C366)

Where active_degree_day_model is a named range somewhere in the workbook, maybe on a user editable worksheet formulating your assumptions.

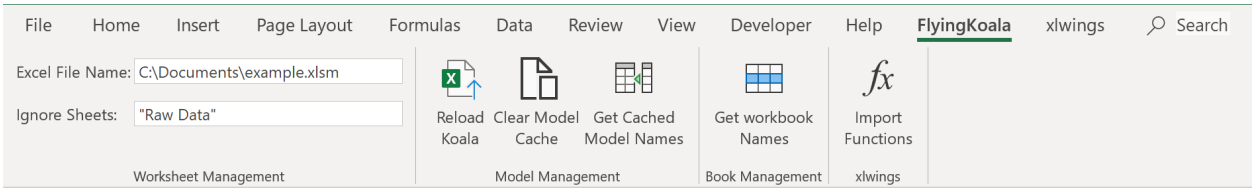
CHAPTER 6

Worked Example - Time Series

The worked example for the timeseries module demonstrates the use of FlyingKoala User Defined Functions for time series transformation.

6.1 1. The FlyingKoala Add-In

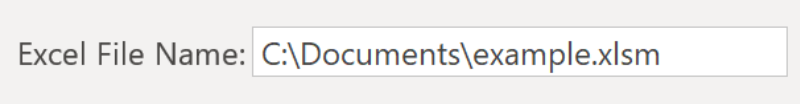
The FlyingKoala Add-In assists users to manage the cache which holds "models" (/equation systems). This is what it looks like.



6.1.1 Excel file name

This is the name of the workbook file. This setting is expected to auto-fill.

In the future we expect to be able to load models from other Excel files which is why this configuration is here.



For doing some time series transformation you would have a module that would look like this:

```
import xlwings as xw
from flyingkoala import *
from flyingkoala.timeseries import *
```

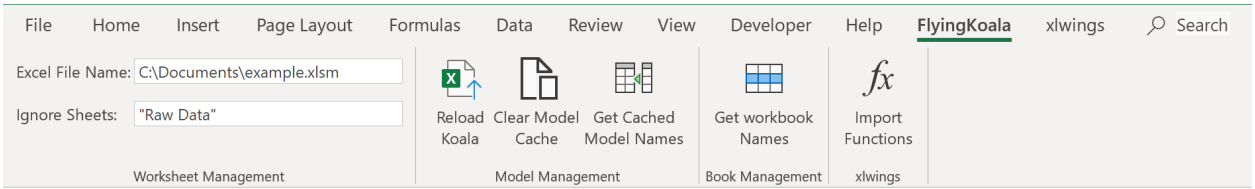
CHAPTER 7

Worked Example - BoM (Bureau of Meteorology)

The worked example for the bom module demonstrates the use of FlyingKoala User Defined Functions for importing data from files.

7.1 1. The FlyingKoala Add-In

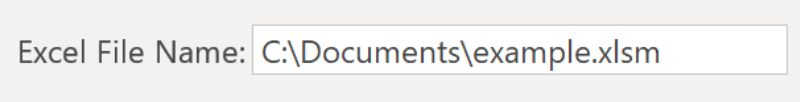
The FlyingKoala Add-In assists users to manage the cache which holds "models" (/equation systems). This is what it looks like.



7.1.1 Excel file name

This is the name of the workbook file. This setting is expected to auto-fill.

In the future we expect to be able to load models from other Excel files which is why this configuration is here.



CHAPTER 8

Glossary

model

A subset of a workbook ("spreadsheet") comprising a collection of cells and the relationship between said cells. Usually identified by input cells and output cells.

Example in Sheet1!B2;

= Sheet1!C2 + Sheet1!D2

In the above we can see the input cells are C2 and D2 (for they define inputs) and output cell B2 (as it defines the output). If this example were loaded as a model only the cells B2, C2 and D2 would be considered - the rest of the workbook would be ignored.

It is obvious that as equations get more complex there can be more input cells. Counterintuitively there can also be more output cells. The reason for this is when you have equations referencing other equations there might be use in having the intermediate cells evaluate.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search