HeatHack Test Book

Contents

This is HeatHack's test of Jupyter Book as a way of producing reading materials for the programme and graphs of temperature and relative humidity data for each of our participating venues.

The book builds automatically when changes are pushed to the main branch of the repository (when enabled, at least).

This is public, but of no use to anyone but the developers (sorry).

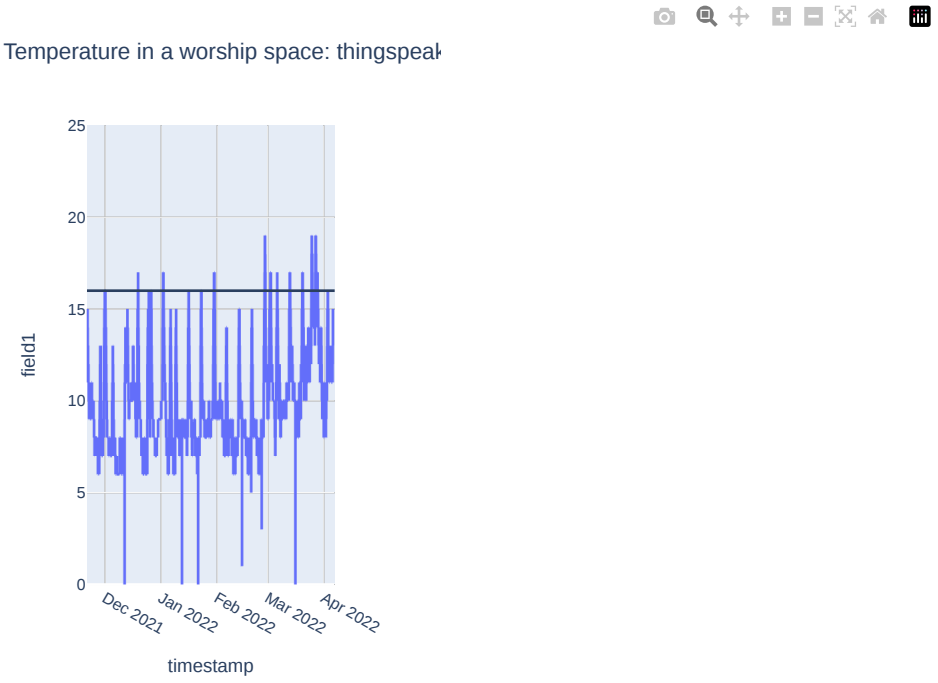
See the Jupyter Book documentation for documentation.

Plots

:TODO: hide code by default. This takes editing some json outside visual studio code, if I remember correctly.

very basic plot with a line at a chosen temperature.

```
import plotly.express as px
import plotly.graph_objects as go

import pandas as pd

filename = "thingspeak-feed"
dfthingspeak = pd.read_csv(filename + ".csv")
dfthingspeak["timestamp"] = pd.to_datetime(dfthingspeak['created_at'])

fig = px.line(dfthingspeak, x='timestamp', y='field1', range_x=['2021-11-
21','2022-04-07'],range_y=[0,25], title="Temperature in a worship space: " +
filename)

fig.add_hline(y=16)

fig.show()
```



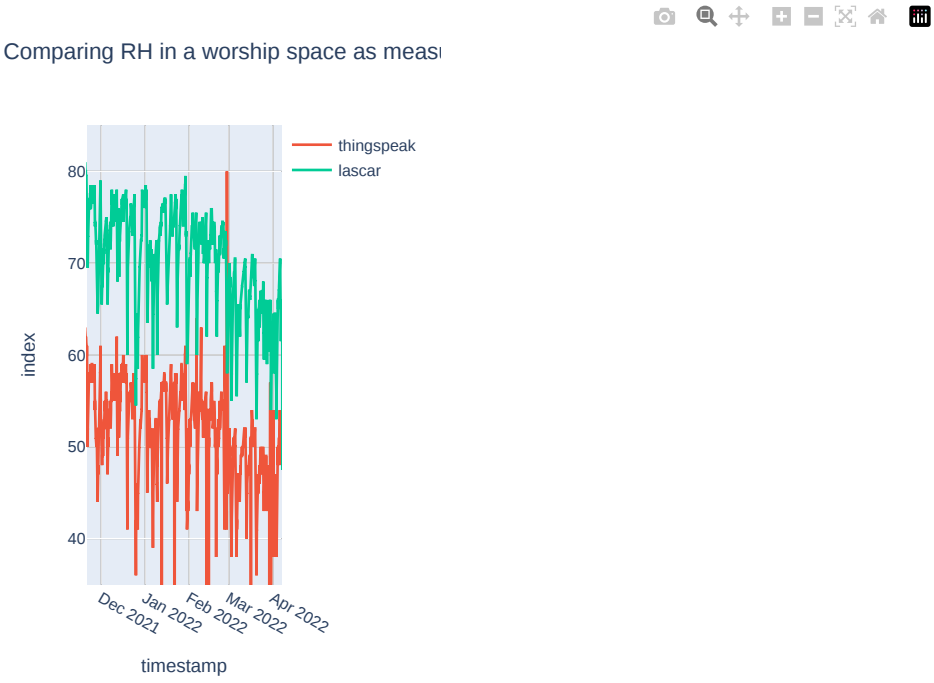
Towards a calibration plot

Simple demonstration of data from two data frames on the same plot - with the wrinkle that one frame is from a lascar logger. We will be roughly exploring the calibration of the RH sensors by running batches of 10 DHT22s alongside a few Lascars over an RH range and showing groups the results, so they can judge how much to trust the data.

Lascars aren't configurable for what they export. I've removed a Unicode character this couldn't deal with (degree symbol) and used Excel to change the data format. These things should be fixable in code, but we won't use Lascars enough for that to be a priority task. Any processing we need to do on Thingspeak feeds is a priority, though.

I don't really understand the interaction between px.line and add_scatter - the difference can get in the way. This way of using plotly and dropping down to graph_objects might be misguided.

```
# Using plotly.express
import plotly.express as px
import plotly.graph_objects as go

import pandas as pd
dfthingspeak = pd.read_csv("thingspeak-feed.csv")
dfthingspeak["timestamp"] = pd.to_datetime(dfthingspeak['created_at'])

dflascar = pd.read_csv("lascar-data.csv")
dflascar['timestamp'] = pd.to_datetime(dflascar['Time'])

# fig = px.line(dfthingspeak, x='timestamp', y='field2', range_x=['2021-11-
21','2022-04-07'],range_y=[35,85], title="Comparing RH as measured by different
devices")
fig = px.line(dfthingspeak, x='timestamp', range_x=['2021-11-21','2022-04-
07'],range_y=[35,85], title="Comparing RH in a worship space as measured by
different devices side by side")
fig.add_scatter(x = dfthingspeak['timestamp'], y = dfthingspeak['field2'], name =
'thingspeak')
fig.add_scatter(x = dflascar['timestamp'], y = dflascar['RH'], name = 'lascar')


fig.show()
```



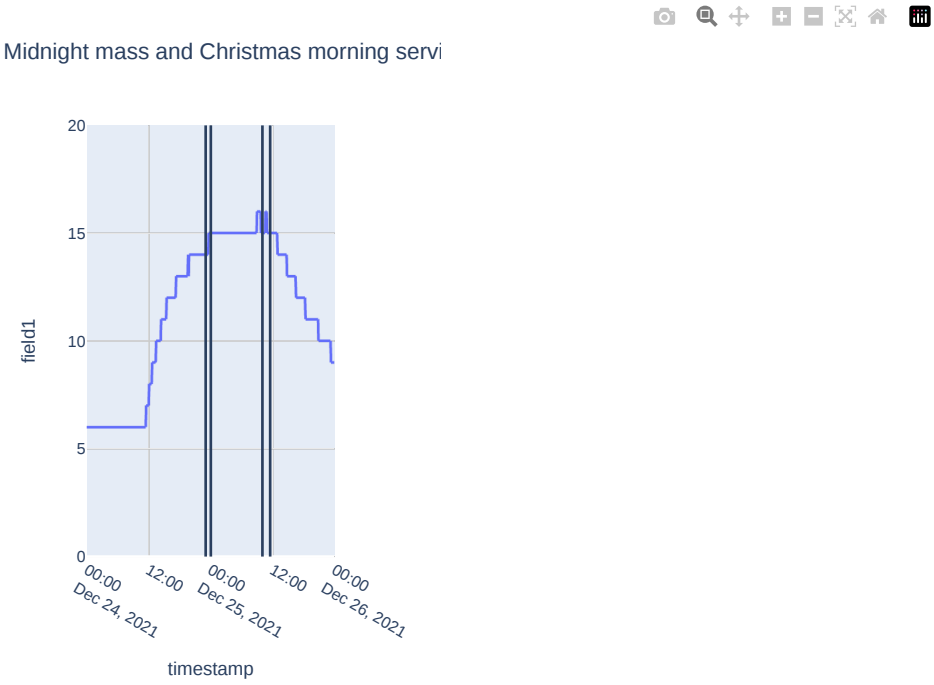
Showing when space is in use.

Vertical lines are useful for the start and end time of events. It would be better rendered as a separate background shading when the space is occupied.

Perhaps we can set up a worksheet where they put in their usual weekly schedule with a descriptive short string to render these. We could use diary export, but if their diary doesn't have a busy/free option, there's too much risk of personal data being in there, and there could be too many diary systems to deal with.

:TODO: It would be helpful if there were a dropdown control for choosing to view a day or a week, and then which specific day or week. That sort of control could be used to choose the group and venue, as well, so we're only producing one master book for everyone.

```
# Using plotly.express
import plotly.express as px

import pandas as pd
df = pd.read_csv("thingspeak-feed.csv")
df["timestamp"] = pd.to_datetime(df['created_at'])


#animation_frame and animation_group should make it possible to add a range
slider??

fig = px.line(df, x='timestamp', y='field1',  range_x=['2021-12-24','2021-12-
26'],range_y=[0,20], title="Midnight mass and Christmas morning services in a
worship space.")
fig.add_vline(x='2021-12-24 23:00')
fig.add_vline(x='2021-12-25 00:00')
fig.add_vline(x='2021-12-25 10:00')
fig.add_vline(x='2021-12-25 11:30')

fig.show()
```



Admonitions formatting

Note

Notes have a nice format with a relevant icon and a specific colour.

Warning

See, this one is a bit different because it's a warning. We want a bunch of possibilities like this that aren't types they have natively: Key Concept (which is a some science), Fun Fact!, and Further Readings.

Admonitions are the general case where you can say gets in the "header", but how do you set the icon and colour - where does the "class" go and what are the ones that are already defined?

Key Concepts

Here's some text about some science.

Further Readings

How do we make this one different?

Fun Fact!

And this one - can it be fun?

Test of panels

We want to know what panels look like on GitHub Pages. Here's one.

At some point we want to know, how hard is it to put a javascript simulation in (for now, any simple one will do)?

Test of sidebars

This is a place to look at the side bar rendering. Maybe it can have classes like admonitions do. We might want to have some of our admonition types be side bars instead.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Here's a sidebar!

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Markdown Files

Whether you write your book's content in Jupyter Notebooks (`.ipynb`) or in regular markdown files (`.md`), you'll write in the same flavor of markdown called **MyST Markdown**. This is a simple file to help you get started and show off some syntax.

What is MyST?

MyST stands for "Markedly Structured Text". It is a slight variation on a flavor of markdown called "CommonMark" markdown, with small syntax extensions to allow you to write **roles** and **directives** in the Sphinx ecosystem.

For more about MyST, see [the MyST Markdown Overview](#).

Sample Roles and Directivs

Roles and directives are two of the most powerful tools in Jupyter Book. They are kind of like functions, but written in a markup language. They both serve a similar purpose, but **roles are written in one line**, whereas **directives span many lines**. They both accept different kinds of inputs, and what they do with those inputs depends on the specific role or directive that is being called.

Here is a “note” directive:

> **Note**
>
> Here is a note

It will be rendered in a special box when you build your book.

Here is an inline directive to refer to a document: Notebooks with MyST Markdown.

Citations

You can also cite references that are stored in a `bibtex` file. For example, the following syntax: `{cite}`holdgraf_evidence_2014`` will render like this: [HdHPK14].

Moreover, you can insert a bibliography into your page with this syntax: The `{bibliography}` directive must be used for all the `{cite}` roles to render properly. For example, if the references for your book are stored in `references.bib`, then the bibliography is inserted with:

[HdHPK14] Christopher Ramsay Holdgraf, Wendy de Heer, Brian N. Pasley, and Robert T. Knight. Evidence for Predictive Coding in Human Auditory Cortex. In *International Conference on Cognitive Neuroscience*. Brisbane, Australia, Australia, 2014. Frontiers in Neuroscience.

Learn more

This is just a simple starter to get you started. You can learn a lot more at jupyterbook.org.

Notebooks with MyST Markdown

Jupyter Book also lets you write text-based notebooks using MyST Markdown. See the Notebooks with MyST Markdown documentation for more detailed instructions. This page shows off a notebook written in MyST Markdown.

An example cell

With MyST Markdown, you can define code cells with a directive like so:

```
print(2 + 2)
```

```
4
```

When your book is built, the contents of any `{code-cell}` blocks will be executed with your default Jupyter kernel, and their outputs will be displayed in-line with the rest of your content.

> **See also**
>
> Jupyter Book uses Jupytext to convert text-based files to notebooks, and can support many other text-based notebook files.

Create a notebook with MyST Markdown

MyST Markdown notebooks are defined by two things:

1. YAML metadata that is needed to understand if / how it should convert text files to notebooks (including information about the kernel needed). See the YAML at the top of this page for example.
2. The presence of `{code-cell}` directives, which will be executed with your book.

That's all that is needed to get started!

Quickly add YAML metadata for MyST Notebooks

If you have a markdown file and you'd like to quickly add YAML metadata to it, so that Jupyter Book will treat it as a MyST Markdown Notebook, run the following command:

```
jupyter-book myst init path/to/markdownfile.md
```

ThingSpeak Graphs Plots

:TODO: hide code by default. This takes editing some json outside visual studio code, if I remember correctly.

First, select your data by venue

```
# Imports
import ipywidgets as widgets
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from IPython.display import display

# Get the possible data venues
venuekeysfile = "venue-keys.csv"
dfVenueKeys = pd.read_csv(venuekeysfile)
dfVenueKeys = dfVenueKeys.dropna(subset=['channel_id'])



#give user option to select their venue
venueDropdown = widgets.Dropdown(
    options=dfVenueKeys['venue_id'],
    value=dfVenueKeys['venue_id'][0],
    description='Venue ID:',
    disabled=False,
)


container = widgets.HBox(children=[venueDropdown])

print(venueDropdown.value)

#Retrieve the venue and begin graphing
dfCollatedDataSet = pd.DataFrame(columns=['timestamp', 'entry_id', 'temperature',
'rh', 'voltage', 'venue_id'])
for index, venueSensorDetails in dfVenueKeys.iterrows():

    sensorMacOfSelection = venueSensorDetails['sensor_MAC']
    dfTempDataSet = pd.read_csv('deviceData/'+ sensorMacOfSelection + '.csv' )
    dfTempDataSet['timestamp'] = pd.to_datetime(dfTempDataSet['timestamp'])
    dfTempDataSet['venue_id'] = venueSensorDetails['venue_id']

    dfCollatedDataSet = dfCollatedDataSet.append(dfTempDataSet, ignore_index=True)
    dfCollatedDataSet['timestamp'] =
pd.to_datetime(dfCollatedDataSet['timestamp'])
    print('Loading data for venue: ', venueSensorDetails['venue_id'])


print('Check')
dfCollatedDataSet.sample(6)

#dfCollatedDataSet.dtypes
```

```
3
Loading data for venue:  3
Loading data for venue:  7
Loading data for venue:  11
Check
```

```
/tmp/ipykernel_1835/3984753893.py:38: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use pandas.concat
instead.
  dfCollatedDataSet = dfCollatedDataSet.append(dfTempDataSet, ignore_index=True)
/tmp/ipykernel_1835/3984753893.py:38: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use pandas.concat
instead.
  dfCollatedDataSet = dfCollatedDataSet.append(dfTempDataSet, ignore_index=True)
/tmp/ipykernel_1835/3984753893.py:38: FutureWarning: The frame.append method is
deprecated and will be removed from pandas in a future version. Use pandas.concat
instead.
  dfCollatedDataSet = dfCollatedDataSet.append(dfTempDataSet, ignore_index=True)
```

	timestamp	entry_id	temperature	rh	voltage	venue_id
630	2022-08-22 09:04:47+00:00	3397	21.0	60.0	40.0	3
7658	2022-09-11 09:53:53+00:00	10425	21.0	56.0	NaN	3
2598	2022-08-27 16:17:21+00:00	5365	22.0	54.0	NaN	3
4559	2022-08-31 17:34:45+00:00	7326	21.0	51.0	NaN	3
8246	2022-09-13 12:14:52+00:00	11013	26.0	57.0	NaN	3
8397	2022-09-13 22:24:22+00:00	11164	24.0	45.0	NaN	3

```
# Assign an empty figure widget with two traces
trace0 = go.Scatter(x=dfCollatedDataSet['temperature'], y =
dfCollatedDataSet['timestamp'], mode='lines', hoverinfo='all', name='temperature')
#trace1 = go.Scatter(x=dfCollatedDataSet['rh'], mode='lines', hoverinfo='all',
name='relative humidity')


importantValues = []
importantValues.append({'type': 'line',
                   'xref': 'x',
                   'yref': 'y',
                   'x0': dfCollatedDataSet['timestamp'].min(),
                   'y0': 16,
                   'x1': dfCollatedDataSet['timestamp'].max(),
                   'y1': 16,
                   'line_color': 'red',
                   'line_dash': 'dash',
                   'layer':'below'})


annotationsToDraw = []
annotationsToDraw.append({'text': 'Legal indoor temperature for children',
                          'x': dfCollatedDataSet['timestamp'].max(),
                          'y': 16})

g = go.FigureWidget(data=[trace0],
                    layout = go.Layout(
                        title=dict(
                            text='Temperature for Venue: ' +
str(venueDropdown.value)
                        ),
                        shapes=importantValues,
                        annotations=annotationsToDraw

                       #barmode='lines'
                    ))


print("Job Done")
```

```
Job Done
```

```
filter_list = [i for i in dfCollatedDataSet['venue_id'] == venueDropdown.value]
temp_df = dfCollatedDataSet[filter_list]
y1 = temp_df['temperature']
x1 = temp_df['timestamp']
#y2 = temp_df['rh']
#x2 = temp_df['dep_delay']
with g.batch_update():
    g.data[0].y = y1
    g.data[0].x = x1
    #g.data[1].y = y2
    #g.layout.barmode = 'overlay'
    g.layout.xaxis.title = 'Time'
    g.layout.yaxis.title = 'Temperature'
    g.layout.title = "Temperature for Venue = " + str(venueDropdown.value)
```

```
def response(change):
        filter_list = [i for i in dfCollatedDataSet['venue_id'] ==
venueDropdown.value]
        temp_df = dfCollatedDataSet[filter_list]
        y1 = temp_df['temperature']
        x1 = temp_df['timestamp']
        #y2 = temp_df['rh']
        #x2 = temp_df['dep_delay']
        with g.batch_update():
            g.data[0].y = y1
            g.data[0].x = x1
            #g.data[1].y = y2
            #g.layout.barmode = 'overlay'
            g.layout.xaxis.title = 'Time'
            g.layout.yaxis.title = 'Value'
            g.layout.title = dict(text= 'Temperature for Venue = ' +
str(venueDropdown.value))
            # g.add_trace(go.Scatter(x=temp_df['timestamp'], y = 16))


# g.add_hline(y=16,
#             x0=dfCollatedDataSet['timestamp'].min(),
#             x1=dfCollatedDataSet['timestamp'].max(),
#             line_color="red")
venueDropdown.observe(response, names="value")
```

```
graphA = widgets.VBox([container, g])
display(graphA)
```

Collate

```
# Imports
import ipywidgets as widgets
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from IPython.display import display

# Get the possible data venues
venuekeysfile = "venue-keys.csv"
dfVenueKeys = pd.read_csv(venuekeysfile)
dfVenueKeys = dfVenueKeys.dropna(subset=['channel_id'])



#give user option to select their venue
venueDropdown = widgets.Dropdown(
    options=dfVenueKeys['venue_id'],
    value=dfVenueKeys['venue_id'][0],
    description='Venue ID:',
    disabled=False,
)


container = widgets.HBox(children=[venueDropdown])

print(venueDropdown.value)

#Retrieve the venue and begin graphing
dfCollatedDataSet = pd.DataFrame(columns=['timestamp', 'entry_id', 'temperature',
'rh', 'voltage', 'venue_id'])
for index, venueSensorDetails in dfVenueKeys.iterrows():

    sensorMacOfSelection = venueSensorDetails['sensor_MAC']
    dfTempDataSet = pd.read_csv('deviceData/'+ sensorMacOfSelection + '.csv' )
    dfTempDataSet['timestamp'] = pd.to_datetime(dfTempDataSet['timestamp'])
    dfTempDataSet['venue_id'] = venueSensorDetails['venue_id']

    dfCollatedDataSet = dfCollatedDataSet.append(dfTempDataSet, ignore_index=True)
    dfCollatedDataSet['timestamp'] =
pd.to_datetime(dfCollatedDataSet['timestamp'])
    print('Loading data for venue: ', venueSensorDetails['venue_id'])


print('Check')
dfCollatedDataSet.sample(6)

#dfCollatedDataSet.dtypes



# Assign an empty figure widget with two traces
trace0 = go.Scatter(x=dfCollatedDataSet['temperature'], y =
dfCollatedDataSet['timestamp'], mode='lines', hoverinfo='all', name='temperature')
#trace1 = go.Scatter(x=dfCollatedDataSet['rh'], mode='lines', hoverinfo='all',
name='relative humidity')


importantValues = []
importantValues.append({'type': 'line',
                    'xref': 'x',
                    'yref': 'y',
                    'x0': dfCollatedDataSet['timestamp'].min(),
                    'y0': 16,
                    'x1': dfCollatedDataSet['timestamp'].max(),
                    'y1': 16,
                    'line_color': 'red',
                    'line_dash': 'dash',
                    'layer':'below'})


annotationsToDraw = []
annotationsToDraw.append({'text': 'Legal indoor temperature for children',
                          'x': dfCollatedDataSet['timestamp'].max(),
                          'y': 16})

g = go.FigureWidget(data=[trace0],
                    layout = go.Layout(
                        title=dict(
                            text='Temperature for Venue: ' +
str(venueDropdown.value)
                        ),
                        shapes=importantValues,
                        annotations=annotationsToDraw

                       #barmode='lines'
                    ))
```

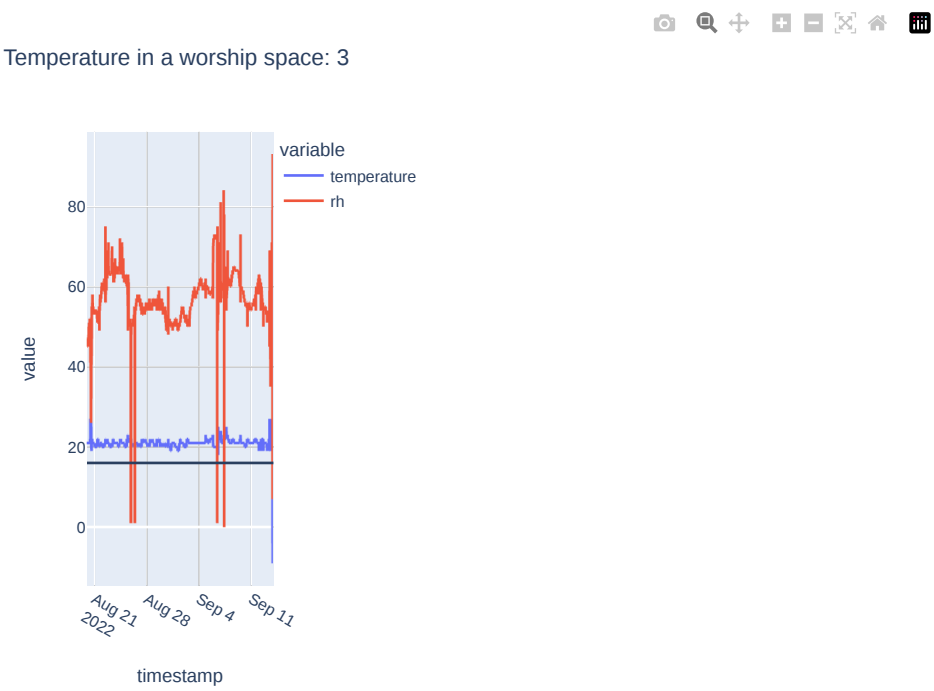
```
print("Job Done")


filter_list = [i for i in dfCollatedDataSet['venue_id'] == venueDropdown.value]
temp_df = dfCollatedDataSet[filter_list]
y1 = temp_df['temperature']
x1 = temp_df['timestamp']
#y2 = temp_df['rh']
#x2 = temp_df['dep_delay']
with g.batch_update():
    g.data[0].y = y1
    g.data[0].x = x1
    #g.data[1].y = y2
    #g.layout.barmode = 'overlay'
    g.layout.xaxis.title = 'Time'
    g.layout.yaxis.title = 'Temperature'
    g.layout.title = "Temperature for Venue = " + str(venueDropdown.value)


def response(change):
        filter_list = [i for i in dfCollatedDataSet['venue_id'] ==
venueDropdown.value]
        temp_df = dfCollatedDataSet[filter_list]
        y1 = temp_df['temperature']
        x1 = temp_df['timestamp']
        #y2 = temp_df['rh']
        #x2 = temp_df['dep_delay']
        with g.batch_update():
            g.data[0].y = y1
            g.data[0].x = x1
            #g.data[1].y = y2
            #g.layout.barmode = 'overlay'
            g.layout.xaxis.title = 'Time'
            g.layout.yaxis.title = 'Value'
            g.layout.title = dict(text= 'Temperature for Venue = ' +
str(venueDropdown.value))
            # g.add_trace(go.Scatter(x=temp_df['timestamp'], y = 16))


# g.add_hline(y=16,
#             x0=dfCollatedDataSet['timestamp'].min(),
#             x1=dfCollatedDataSet['timestamp'].max(),
#             line_color="red")
venueDropdown.observe(response, names="value")



graphA = widgets.VBox([container, g])
display(graphA)
```

```
3
Loading data for venue:  3
Loading data for venue:  7
Loading data for venue:  11
Check
```

```
/tmp/ipykernel_1835/1996588037.py:38: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.

/tmp/ipykernel_1835/1996588037.py:38: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.

/tmp/ipykernel_1835/1996588037.py:38: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
```

```
Job Done
```

```
import plotly.express as px
import plotly.graph_objects as go


# filename = "thingspeak-feed"
# dfthingspeak = pd.read_csv(filename + ".csv")
# dfthingspeak["timestamp"] = pd.to_datetime(dfthingspeak['created_at'])

fig = px.line(dfCollatedDataSet, x='timestamp', y='temperature', range_x=
['','2022-09-09'], title="Temperature in a worship space: " +
str(venueDropdown.value))
fig = px.line(dfCollatedDataSet, x= 'timestamp', y=dfCollatedDataSet.columns[2:4],
title="Temperature in a worship space: " + str(venueDropdown.value))
fig.add_hline(y=16)

fig.show()
```



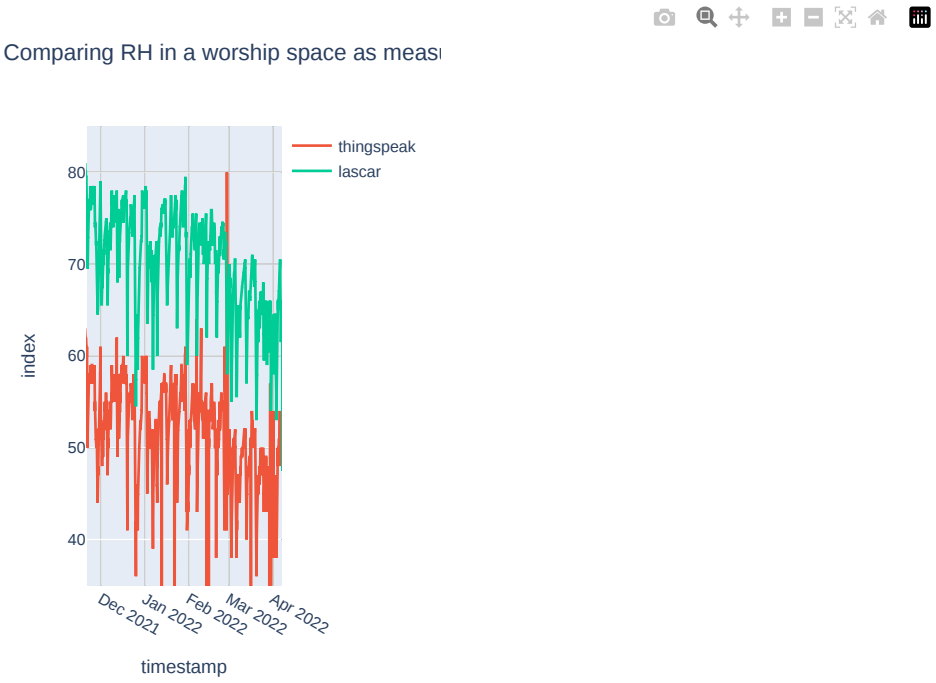
Towards a calibration plot

Simple demonstration of data from two data frames on the same plot - with the wrinkle that one frame is from a lascar logger. We will be roughly exploring the calibration of the RH sensors by running batches of 10 DHT22s alongside a few Lascars over an RH range and showing groups the results, so they can judge how much to trust the data.

Lascars aren't configurable for what they export. I've removed a Unicode character this couldn't deal with (degree symbol) and used Excel to change the data format. These things should be fixable in code, but we won't use Lascars enough for that to be a priority task. Any processing we need to do on Thingspeak feeds is a priority, though.

I don't really understand the interaction between px.line and add_scatter - the difference can get in the way. This way of using plotly and dropping down to graph_objects might be misguided.

```
# Using plotly.express
import plotly.express as px
import plotly.graph_objects as go

import pandas as pd
dfthingspeak = pd.read_csv("thingspeak-feed.csv")
dfthingspeak["timestamp"] = pd.to_datetime(dfthingspeak['created_at'])

dflascar = pd.read_csv("lascar-data.csv")
dflascar['timestamp'] = pd.to_datetime(dflascar['Time'])

# fig = px.line(dfthingspeak, x='timestamp', y='field2', range_x=['2021-11-
21','2022-04-07'],range_y=[35,85], title="Comparing RH as measured by different
devices")
fig = px.line(dfthingspeak, x='timestamp', range_x=['2021-11-21','2022-04-
07'],range_y=[35,85], title="Comparing RH in a worship space as measured by
different devices side by side")
fig.add_scatter(x = dfthingspeak['timestamp'], y = dfthingspeak['field2'], name =
'thingspeak')
fig.add_scatter(x = dflascar['timestamp'], y = dflascar['RH'], name = 'lascar')


fig.show()
```



Showing when space is in use.

Vertical lines are useful for the start and end time of events. It would be better rendered as a separate background shading when the space is occupied.

Perhaps we can set up a worksheet where they put in their usual weekly schedule with a descriptive short string to render these. We could use diary export, but if their diary doesn't have a busy/free option, there's too much risk of personal data being in there, and there could be too many diary systems to deal with.

:TODO: It would be helpful if there were a dropdown control for choosing to view a day or a week, and then which specific day or week. That sort of control could be used to choose the group and venue, as well, so we're only producing one master book for everyone.

```
# Using plotly.express
import plotly.express as px

import pandas as pd
df = pd.read_csv("thingspeak-feed.csv")
df["timestamp"] = pd.to_datetime(df['created_at'])


#animation_frame and animation_group should make it possible to add a range
slider??

fig = px.line(df, x='timestamp', y='field1',  range_x=['2021-12-24','2021-12-
26'],range_y=[0,20], title="Midnight mass and Christmas morning services in a
worship space.")
fig.add_vline(x='2021-12-24 23:00')
fig.add_vline(x='2021-12-25 00:00')
fig.add_vline(x='2021-12-25 10:00')
fig.add_vline(x='2021-12-25 11:30')

fig.show()
```