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.

ThingSpeak Graphs: Choose your venue from the dropdown and explore your data with the slider!

```
# 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)




# Assign an empty figure widget with two traces
trace0 = go.Scatter(customdata=dfCollatedDataSet[dfCollatedDataSet['venue_id'] ==
0],
                    y=dfCollatedDataSet['temperature'],
                    x = dfCollatedDataSet['timestamp'],
                    mode='lines',
                    hoverinfo='all',
                    name='Temperature',
                    )

trace1 = go.Scatter(customdata=dfCollatedDataSet[dfCollatedDataSet['venue_id'] ==
0],
                    y=dfCollatedDataSet['rh'],
                    x = dfCollatedDataSet['timestamp'],
                    mode='lines',
                    hoverinfo='all',
                    name='Relative Humidity',

)


g = go.FigureWidget(data=[trace0, trace1],
                    layout = go.Layout(
                        yaxis=dict(range=[0,0])
                    ))


print("Job Done")
```

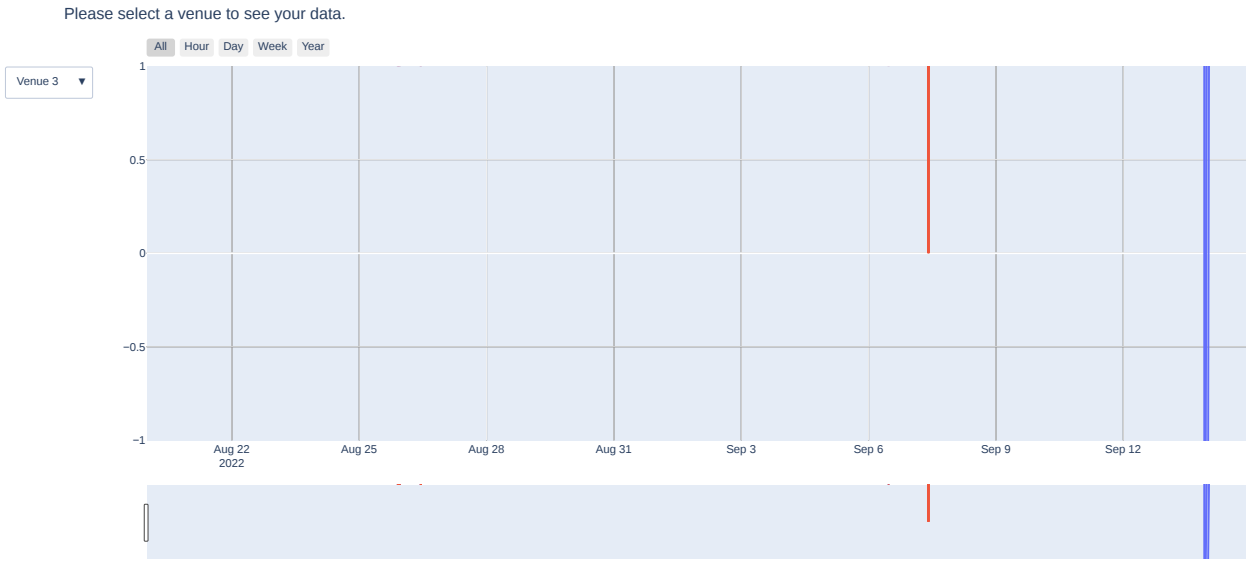
```
3
Loading data for venue:  3
Loading data for venue:  7
Loading data for venue:  11
Check
```

```
/tmp/ipykernel_1822/4225850045.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_1822/4225850045.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_1822/4225850045.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)
```

```
Job Done
```

Graphs!

```
updatemenu = []
buttons = []

# button with one option for each dataframe
for index, venue in dfVenueKeys.iterrows():
    buttons.append(dict(method='update',
                        label='Venue ' + str(venue['venue_id']),
                        visible=True,
                        args=[{'y':
[dfCollatedDataSet[dfCollatedDataSet['venue_id']==venue['venue_id']]
['temperature'].values,

dfCollatedDataSet[dfCollatedDataSet['venue_id']==venue['venue_id']]['rh'].values],
                               'x':
[dfCollatedDataSet[dfCollatedDataSet['venue_id']==venue['venue_id']]
['timestamp'].values],
                               'type':'scatter',
                            #  'name': 'Temperature',

                               },
                               {
                                   'title.text': 'Temperature for Venue = ' +
str(venue['venue_id']),
                                   'title.font.color': 'green',
                                   'yaxis.range':
[-5,dfCollatedDataSet[dfCollatedDataSet['venue_id']==venue['venue_id']]
[['temperature', 'rh']].max().max()+5],
                                   'yaxis.title.text': 'Temperature',
                                   'xaxis.title.text': 'Timestamp'

                               },
                               ],
                        )
                )

# some adjustments to the updatemenus
updatemenu = []
your_menu = dict()
updatemenu.append(your_menu)

updatemenu[0]['buttons'] = buttons
updatemenu[0]['direction'] = 'down'
updatemenu[0]['showactive'] = True
# updatemenu[0]['active'] = 0



fig = go.Figure(g)
# add dropdown menus to the figure
fig.update_layout(showlegend=True,
              updatemenus=updatemenu,
              autosize = True,
              title= "Please select a venue to see your data.",
              width=1500,
              height=700,
)

fig.update_layout(
    hovermode='x unified',
    hoverlabel=dict(
        bgcolor="white",
        # font_size=16,
        font_family="Rockwell"
    )
)

# Add range slider
fig.update_layout(
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(
                     label="All",
                     step="all"
                     ),
                                dict(count=1,
                     label="Hour",
                     step="hour",
                     stepmode="todate"),
                dict(count=1,
                     label="Day",
                     step="day",
                     stepmode="backward"),
                dict(count=7,
                     label="Week",
                     step="day",
                     stepmode="backward"),
                dict(count=1,
                     label="Year",
                     step="year",
                     stepmode="backward")
            ])
        ),
        rangeslider=dict(
            visible=True
        ),
        type="date"
    )
)


#fig.update_yaxes(range=[50, 60])


fig.add_hline(y=16, annotation_text='Legal Lower Temp for Children',
annotation_font_color="blue", line_color='red', layer='above', line_dash='dash')
# fig.update_yaxes(range = [-5, dfCollatedDataSet['temperature'].max()+5])
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.

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.

```
fig1 = fig

fig1.add_vrect(x0='2022-09-14 06:00', x1='2022-09-14 07:30',
               annotation_text="Morning Prayer",
               annotation_position="top left",
               annotation=dict(font_size=14,
               font_family="Times New Roman"),
               fillcolor="green",
               opacity=0.25,
               line_width=0)

fig1.add_vrect(x0='2022-09-14 09:00', x1='2022-09-14 12:00',
               annotation_text="Clinic",
               annotation_position="top left",
               annotation=dict(font_size=14,
               font_family="Times New Roman"),
               fillcolor="green",
               opacity=0.25,
               line_width=0)

fig1.add_vrect(x0='2022-09-14 14:00', x1='2022-09-14 18:00',
               annotation_text="Indoor Football",
               annotation_position="top left",
               annotation=dict(font_size=14,
               font_family="Times New Roman"),
               fillcolor="green",
               opacity=0.25,
               line_width=0)


fig1.show()
```

Please select a venue to see your data.