Task 2. Data Visualization

NOTE:

- please convert streaming vis code cell to markdown or comment it out, prior to running the static visualizations, as it will not run, since streaming vis code cell is still listening to the streams.
- also restart kernell and comment out streaming vis code and after running streaming vis code to run static vis code
- There are some bugs (BUT VERY RARE) regarding matplot lib, sometimes it may require a few runs to get the streaming vis to be perfectly running, in the sense that, the bug was that it does not draw properly for the first 1 minute or so. However, this is out of my control.

imports

In [ ]:
```python
# kafka and mongo
from kafka import KafkaConsumer
from pymongo import MongoClient
# utility libraries
from time import sleep
import datetime as dt
import json
import math
import statistics
# date time library
import datetime
# from datetime import date, datetime
from dateutil.parser import parse
# matplot lib
import matplotlib.pyplot as plt
# folium vis
import folium
```

Constants

In [ ]:
```python
# VIS_TOPIC = 'vis1'  # listening from another topic, so that we dont have to handle "de-embedding of "
TOPIC = "assignment"
PORT_NUMBER = 9092
```

Fieldname Constants

In [ ]:
```python
# FIELDNAMES
PRODUCER = "producer"
# common
DATE = "date"
LAT = "latitude"
LNG = "longitude"
# climate
CLIMATE_ID = "climate_id"
AIR_TEMP = "air_temperature_celcius"
RELATIVE_HUMIDITY = "relative_humidity"
WINDSPEED = "windspeed_knots"
MAX_WINDSPEED = "max_wind_speed"
PRECIPITATION = "precipitation"
GHI = "GHI_w/m2"
HOTSPOT_ENTRIES = "hotspot_entries"
# hotspot
DATETIME = "datetime"
HOTSPOT_ID = "hotspot_id"
SURFACE_TEMP = "surface_temperature_celcius"
CONFIDENCE = "confidence"
FIRE_CAUSE = "fire_cause"
NATURAL_FIRE_CAUSE = "natural"
OTHER_FIRE_CAUSE = "other"
```

Utility Functions

In [ ]:
```python
def date2str(date, need_time=False):
    if need_time:
        date_format = f'%d/%m/%yT%H:%M:%S'
    elif not need_time:
        date_format = f'%d/%m/%y'
    #latest_str = date.strftime(latest, f'%d/%m/%y')
    date_str = date.strftime(date_format)
    return date_str

def generate_tick(start, end, tick_interval):
    ret = []
    val = start
    i = 0
    while True:
        val = start + i * tick_interval
        if val > end:
            break
        ret.append(val)
        i += 1
        # print(val, i)
    return ret

def add_newline_char(str_lst):
    ret = ""
    for text in str_lst:
        ret += f'{text}\n'
    return ret
```

Task 2.1 streaming visualization

- plots a real-time line graph using data stream obtained from the kafka producers

streaming line graph constants

this line is needed for the inline display of graphs in Jupyter Notebook

%matplotlib notebook

LABEL_X = "arrival time (date)" LABEL_Y = "air temperature (celcius)" Y_LIM_PAIR = (0,40) TICK_INTERVAL = 10

def annotate_max(x, y, ax = None): ymax = max(y) xpos = y.index(ymax) xmax = x[xpos] text = f'Max: {DATE}={xmax}, Value={ymax}' if not ax: ax=plt.gca() ax.annotate(text, xy=(xmax, ymax), xytext=(xmax, ymax+5), arrowprops=dict(facecolor='red', shrink=0.05),)

def annotate_min(x, y, ax = None): ymin = min(y) xpos = y.index(ymin) xmin = x[xpos] text = f'Min: {DATE}={xmin}, Value={ymin}' if not ax: ax=plt.gca() ax.annotate(text, xy=(xmin, ymin), xytext=(xmin, ymin+5), arrowprops=dict(facecolor='orange', shrink=0.05),)

def connect_kafka_consumer(): _consumer = None try: """ specifices the auto_offset_reset to reset the consumer offset for topics

```
    can specify more than 1 broker in bootstrap_servers
    """
    _consumer = KafkaConsumer(TOPIC,
                              # stop iteration if no message after 10 sec
                              consumer_timeout_ms=20000,
                              # comment this if you don't want to consume ear
liest available message
                              auto_offset_reset='earliest',
                              bootstrap_servers=[f'localhost:{PORT_NUMBER}'],
                              api_version=(0, 10))
except Exception as ex:
    print('Exception while connecting Kafka')
    print(str(ex))
finally:
    return _consumer
```

def init_plots(): try: width, height = 9.5, 9 fig = plt.figure(figsize=(width,height)) # create new figure ax = fig.add_subplot(111) # adding the subplot axes to the given grid position fig.suptitle('Daily Air Temperature') # giving figure a title

```
    # y-axis stuff
    ax.set_xlabel(LABEL_X)
    ax.set_ylabel(LABEL_Y)
    ax.set_ylim(Y_LIM_PAIR)
    # generate ticks for y axis
    ax.set_yticks(generate_tick(*Y_LIM_PAIR, TICK_INTERVAL))

    fig.show() # displaying the figure
    fig.canvas.draw() # drawing on the canvas
    return fig, ax
except Exception as ex:
    print(str(ex))
```

def consume_messages(consumer, fig, ax): try:

```
    # container for x and y values
    x, y, y_mean = [], [], []

    # print('Waiting for messages')
    for message in consumer:
        # decode the UTF-8 encoded bytes from the PRODUCER
        data = str(message.value.decode('utf-8'))

        # convert string to JSON dict object
        c_dict = json.loads(data)
        # c_dict[DATE] = parse(c_dict[DATE],ignoretz=True)

        print(c_dict)

        x.append(c_dict[DATE])
        y.append(c_dict[AIR_TEMP])

        if len(y) > 5:
            # moving average across latest 5 values from y
            y_mean.append(statistics.mean(y[:5]))
        else:
            y_mean.append(0)

        # print(y)
        # we start plotting only when we have 10 data points
        if len(y) > 10:
            # refresh
            ax.clear()
            # plot the objects, draw later
            ax.plot(x, y)
            ax.plot(x, y_mean)
            # axis stuff
            ax.set_xlabel(LABEL_X)
            ax.set_ylabel(LABEL_Y)
            ax.set_ylim(Y_LIM_PAIR)
            ax.set_yticks(generate_tick(*Y_LIM_PAIR, TICK_INTERVAL))
            # min and max markers
            annotate_max(x, y, ax)
            annotate_min(x, y, ax)
            # draw on canvas
            fig.canvas.draw()

            # remove first data from all data arrays
            _,_,_ = x.pop(0), y.pop(0), y_mean.pop(0)

    plt.close('all')

except Exception as ex:
    print(str(ex))
```

if **name** == '**main**': consumer = connect_kafka_consumer() fig, ax = init_plots() consume_messages(consumer, fig, ax)

Task 2.2 static visualization

In [ ]:
```python
client = MongoClient()
db = client.fit3182_assignment_db
climate_streaming = db.climate_streaming
```

Task 2.2a bar chart

Y_LIM_PAIR = (0,40) TICK_INTERVAL = 10 LABEL_Y = "Count of Fire Events" LABEL_X = "Time (hours)" TITLE = "Hourly Occurences of Fire Events"

def hour_grouping(d_lst): """ Inspired by hash-based serial group by returns a dictionary object, with key-value pairs of: key: hour, value: a lst of records with matching times in terms of hours

```
# NOTE: in time format, HH:MI:SS, 05:00:00 all the way to 05:59:59 is treated
 as the same hour.
"""
groups = {}
hours = generate_tick(0,23,1)
for hour in hours:
    groups[hour] = []
# print(f"hour_grouping, d_lst: {d_lst}")
for rec in d_lst:
    hour = rec["datetime"].hour
    # if not exist yet, create new group
    if hour not in groups:
        groups[hour] = [rec]
    # else append to group
    else:
        groups[hour].append(rec)
# expects groups of 0, 1,..., 23, there are 24 groups in total
return groups
```

def agg_count_and_max_count(hour_groups): agg_count_groups = {} max_count = 0 for hour_key in hour_groups: hour_group = hour_groups[hour_key]

```
    # count the number of entries
    count = len(hour_group)
    if count > max_count:
        max_count = count
    agg_count_groups[hour_key] = count
return agg_count_groups, max_count
```

def fire_counts_from_groups(groups_dict):

```
# GOAL: return counts in order of hour, from hours of 0 to 23
counts = []
sorted_hours = sorted(groups_dict.keys())
for hour_key in sorted_hours:
    count = groups_dict[hour_key]
    counts.append(count)
return counts
```

def plot_bar_chart():

```
# Preparing the data for plotting
fires = []
count = climate_streaming.count_documents({})
climate_data = list(climate_streaming.find({}))
print(f'number of clim stream records: {count}')

# loop through climate recs and compile
compiled_hotspots = []
for c_dic in climate_data:
    hotspots = c_dic[HOTSPOT_ENTRIES] # a lst of dicts
    for fire in hotspots:
        compiled_hotspots.append(fire)
print(f'number of compiled fires: {len(compiled_hotspots)}')

# GOAL: load compiled hotspots into and categorize by hour groups, finally co
unt number of fires
hour_groups = hour_grouping(compiled_hotspots)
fire_counts_groups, max_count = agg_count_and_max_count(hour_groups) # {"0":
 23, ... "23": 53}

width, height = 7.5, 9
fig = plt.figure(figsize=(width, height))
ax = fig.add_axes([0,0,1,1])
hour_x_axis = generate_tick(0,23,1) # 0 to 23 hrs, per hour tick
hour_x_axis = [str(hour) for hour in hour_x_axis]
# print(f"hour_x_axis: {hour_x_axis}")

# title and axis stuff
fig.suptitle(TITLE) # giving figure a title
ax.set_xlabel(LABEL_X)
ax.set_ylabel(LABEL_Y)
ax.set_ylim(Y_LIM_PAIR)

# fire_count_y_axis = generate_tick(0, max_count, (max_count)//30)
fire_counts = fire_counts_from_groups(fire_counts_groups)
print(f"fire_counts: {fire_counts}")

ax.bar(hour_x_axis, fire_counts)
plt.show()
```

plot_bar_chart()

Task 2.2b map visualization

In [ ]:
```python
# Preparing the data for plotting
fires = []
count = climate_streaming.count_documents({})
climate_data = list(climate_streaming.find({}))
print(f'number of clim stream records: {count}')
#print(climate_data)

# Looping through all the data
for c_dic in climate_data:
    hotspots = c_dic[HOTSPOT_ENTRIES] # a lst of dicts
    """
    NOTE:
    modifies the hotspot dictionary objects to
    include the air_temperature and relative humidity of climate-data
    so that we can process it easier during the drawing of markers on the vis map
    """
    for fire_dic in hotspots:
        fire_dic[AIR_TEMP] = c_dic[AIR_TEMP]
        fire_dic[RELATIVE_HUMIDITY] = c_dic[RELATIVE_HUMIDITY]
        fires.append(fire_dic)

# TODO: initialize a map at Victoria state
# https://www.latlong.net/place/victoria-australia-12947.html
fomap = folium.Map(location=[-37.020100, 144.964600], zoom_start=6.5)


COLORMAP = {
    NATURAL_FIRE_CAUSE: "blue",
    OTHER_FIRE_CAUSE: "red" # red  # be4d25
}

# TODO: 2 types of markers, for different fire causes
def compute_color_fire_cause(fire):
    # color map: {fire_cause: color, ...}
    try:
        fire_cause = fire[FIRE_CAUSE]
        color = COLORMAP[fire_cause]
        return color
    except TypeError:
        errmsg = f'fire: {fire}'
        raise TypeError(errmsg)

print(f"fire event count: {len(fires)}")

for fire in fires:
    # we assume that KeyError would not happen, because we are reading from climate_strea
m,
    # without any data from Part A
    color_str = compute_color_fire_cause(fire)
    # print(color_str)

#     popupmsg = add_newline_char([
#             f'date: {date2str(fire[DATE],need_time=False)}',
#             f'lat: {round(fire[LAT],5)}',
#             f'lon: {round(fire[LNG],5)}',
#             f'air_temp(celcius): {fire[AIR_TEMP]}',
#             f'surf_temp(celcius): {fire[SURFACE_TEMP]}',
#             f'relative_humidity: {round(fire[RELATIVE_HUMIDITY], 5)}',
#             f'confidence: {fire[CONFIDENCE]}',
#             f'cause:{fire[FIRE_CAUSE]}'
#         ])

    # NOTE: newline doesnt work in folium tooltips
    popupmsg = (f"""
        date: {date2str(fire[DATE],need_time=False)},
        lat: {round(fire[LAT],5)},
        lng: {round(fire[LNG],5)},
        air_temp(celcius): {fire[AIR_TEMP]},
        surf_temp(celcius): {fire[SURFACE_TEMP]},
        relative_humidity: {round(fire[RELATIVE_HUMIDITY], 5)},
        confidence: {fire[CONFIDENCE]},
        cause:{fire[FIRE_CAUSE]}
    """)

    tooltipmsg = (f"""
        date: {date2str(fire[DATE],need_time=False)},
        air_temp(celcius): {fire[AIR_TEMP]},
        surf_temp(celcius): {fire[SURFACE_TEMP]},
        relative_humidity: {round(fire[RELATIVE_HUMIDITY], 5)}
    """)
    folium.Marker(location=[fire[LAT], fire[LNG]],
                  popup=popupmsg,
                  tooltip=tooltipmsg,
                  # color=color_str
                  icon=folium.Icon(color=color_str, icon="info-sign")
    ).add_to(fomap)

# show the map
fomap

# save the map to a file for later viewing.
# fomap.save('parking-spots.html')
# fomap.save('fire-events.html')
```

In [ ]: