Smart Cities Hackathon
Tier 3 Challenge 13

Tier 3: Challenge 13

Tier 3 Challenge 13: Environmental Data Analysis

Objective:
The objective of this challenge is to analyse IoT sensor data and create a simple predictive model to forecast future sensor readings.

Problem Description:
You are provided with some data containing IoT sensor readings collected over a period of time. Each record in the dataset represents a snapshot of sensor data, including the timestamp of the reading and various sensor values (e.g., temperature, humidity), and is roughly equivalent to a row in an Excel spreadsheet.

Tasks:

Data pre-processing:

- Load the dataset into a Polars DataFrame using the read_json command.
- Explore the dataset to understand its structure and characteristics.

Data Visualisation:

- Visualise the distribution of sensor readings over time.
- Explore relationships between different sensor variables using scatter plots, histograms, or heatmaps.

Data Forecasting:

- Fit a Prophet model to a time series from your data.
- Explore the limitations of the model when making predictions.

Tier 3: Challenge 13

Step 1
If you have not already done so:

Go to the official Docker website: https://www.docker.com/products/docker-desktop, Download Docker Desktop for your operating system (Windows, macOS, or Linux). Follow the installation instructions provided for your operating system.

Verify Docker installation:
After installing Docker Desktop, open a terminal or command prompt.

Install Docker desktop onto your machine.

Step 2

Create a new folder on your computer for Tier 3 challenge 13.

Step 3

Inside this folder, create a **Dockerfile** and add the following elements.

```
# We'll be using jupyter to do our data exploration
FROM jupyter/base-notebook

# Mount the requirements file into the container
ADD ./requirements.txt ./requirements.txt

# Install any needed dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt
```

Step 4

Next, create a file called **requirements.txt** and add the following contents.

```
vegafusion[embed]>=1.5.0
polars
altair
prophet
```

Step 5

Create a file called **compose.yml** and add the following contents.

```
services:
  jupyter:
    build: .
    ports:
      - "8888:8888"
    volumes:
      - ./hackathon:/home/jovyan/work
    environment:
      - JUPYTER_ENABLE_LAB=yes
    command: start-notebook.sh --ServerApp.token='' --ServerApp.password=''
```

Step 6

In your terminal, run the following command

```
docker compose up -d
```

Step 7

Open your browser and navigate to http://127.0.0.1:8888/lab. It may take a few minutes for the page to load, try periodically refreshing if it doesn't. Once the page has loaded, open the file browser on the left hand side of the page, navigate through to the work folder, and create a new Python3 notebook.

Step 8

Within the notebook, you should see a small box that you can enter text into. This is a Jupyter cell, you can create new cells using the plus button at the top of the page, and run the code in a cell either by selecting it then pressing the play button at the top of the screen, or selecting it then pressing ctrl-enter. Enter the following into a cell.

```
!(curl --location
'https://api.core.aql.com/v1/sensors/sensor-data/aggregate/min_ave_max'  \
--header 'Accept: application/json'  \
--header 'Authorization: Bearer
11473|VNgWEU6wC1Xs5kdQ20SXUTNzCk3kyF6T6mccZZmhb36d4e42'  \
--header 'Content-Type: application/json'  \
--data '{ "sensor_ids": [ "DEg9FPJkwqj", "yk3Qtn2RKZ6", "KEnkUJMGPO8",
"8Y5psrPKW1O", "DE4ESPJkwqj", "4KXMso9vEgY", "ykLJTn2RKZ6", "KE3DhJMGPO8",
"pVxnSDOBqmQ", "Q2qkSxlygOB", "8xnNCrPKW1O", "D2DpfPJkwqj", "yDGZCn2RKZ6",
"K2xESJMGPO8" ], "startDate":  "2024-03-01 00:00:00", "endDate":  "2024-04-24
00:00:00", "sampleInterval": "Day"}') \
> temp_data.json
```

Beginning with a ‘!’ allows you to run commands like this, which would ordinarily require a terminal.

Step 9

Once the previous command has finished running, create a new cell and enter the following into it. Run this cell.

```
!(curl --location
'https://api.core.aql.com/v1/sensors/sensor-data/aggregate/min_ave_max'  \
--header 'Accept: application/json'  \
--header 'Authorization: Bearer YOUR_TOKEN_ID'  \
--header 'Content-Type: application/json'  \
--data '{ "sensor_ids": [ "QLyjtxlygOB", "wgBEUpVWkK0", "XVEOCJDzj2B",
"pVMZSDOBqmQ", "QL4rHxlygOB", "l6rYi7PvmxO", "wgJlIpVWkK0", "XV83tJDzj2B",
"DEmDfPJkwqj", "4qDKCo9vEgY", "pqp0IDOBqmQ", "Q2pNSxlygOB", "wA40TpVWkK0",
"X0OBHJDzj2B"], "startDate":  "2024-03-01 00:00:00", "endDate":  "2024-04-24
00:00:00", "sampleInterval": "Day"}') \
> humidity_data.json
```

Step 10

We now need some python libraries suitable for data analytics. We'll be using a modern dataframe library called Polars, a declarative graphing library called Altair, and a simple forecasting library produced by Meta called Prophet. We'll also be using a plugin for Altair called Vegafusion to allow us to plot massive quantities of data. Enter the following into a new cell and run it.

```
import polars as pl
import altair as alt
from prophet import Prophet

alt.data_transformers.enable("vegafusion")
```

Step 10

You should have two files, temp_data.json and humidity_data.json visible in your file browser as a result of the curl commands you ran earlier. To read the contents of these files into memory, enter the following into a cell and run it.

```
temp_data = pl.read_json('temp_data.json')

humidity_data = pl.read_json('humidity_data.json')
```

Step 11

Let's have a look at our new data! Create four new cells, enter and run the following:

```
temp_data.sample(5)
```

```
temp_data.describe()
```

```
humidity_data.sample(5)
```

```
humidity_data.describe()
```

Have a look at the output of these commands and see if you can get a feel for the data we're working with. The table output from the describe command should have a 'count' statistic that tells us how many data points we're dealing with for each of these dataframes. As we go on, try to compare what you're doing with the work you would expect to do if you were processing this much data using a program like Excel.

Step 12

You might have spotted that the two dataframes have the same column names. To avoid confusion moving forward, let's rename the columns we'll be working with to something more distinctive. Enter the following command into a new cell and run it.

```
temp_data = temp_data.rename({
    "Average":"averageTemp",
    "Maximum":"maximumTemp",
    "Minimum":"minimumTemp"
})

humidity_data = humidity_data.rename({
    "Average":"averageHumidity",
    "Maximum":"maximumHumidity",
    "Minimum":"minimumHumidity"
})
```

Step 13

Now it's time to plot some of our data. You might have noticed that any python data that we leave on the final line of a cell is shown as output below that cell after it runs. This lets us display graphs by leaving a reference to the graph on the final line of the cell.

The following code creates a chart object, specifies that it should use line based marks, and that the x/y positions of these marks should represent the sensorReadingDate/averageTemp column. Enter the following command into a new cell and run it.

```
temp_chart = alt.Chart(temp_data).mark_line().encode(
    x='sensorReadingDate:T',
    y='averageTemp:Q',
).properties(
    title='Temperature Data',
    width=1300
)

temp_chart
```

We've annotated the column names 'sensorReadingDate' and 'averageTemp' with a 'T' and a 'Q' to indicate that this is temporal (time) and quantitative (numerical) data. The other valid annotations are 'O' for ordinal (numerical sequence) and 'N' for nominal (label) data types. Try different combinations of these annotations for your graph and see how they impact the output.

Step 13

The chart we've just created could be a lot better. Let's differentiate between different readings based on the contents of the sensor_id column, and add a tooltip so that we can hover over points that we're interested in and see exactly what they represent. We can also make the chart zoomable and pannable using the 'interactive' method. Modify the previously run cell with these changes.

```
temp_chart = alt.Chart(temp_data).mark_line(opacity=0.6).encode(
    x='sensorReadingDate:T',
    y='averageTemp:Q',
    color='sensor_id:N',
    tooltip=['sensorReadingDate:T', 'averageTemp:Q', 'sensor_id:N']
).properties(
    title='Temperature Data',
    width=1300
).interactive()

temp_chart
```

Step 14

Let's do the same thing for our humidity data. Enter the following command into a new cell and run it.

```
humidity_chart = alt.Chart(humidity_data).mark_line(opacity=0.6).encode(
    x='sensorReadingDate:T',
    y='averageHumidity:Q',
    color='sensor_id:N',
    tooltip=['sensorReadingDate:T', 'averageHumidity:Q', 'sensor_id:N']
).properties(
    width=1300
).interactive()

humidity_chart
```

Step 15

Since our humidity and temperature sensors live in pairs on a given device, we should be able to pair up our humidity and temperature reading data based on the timestamp and device name. First let's select just the rows that we're interested in from the two tables. Then we can create a new table that joins the data from the previous cells based on timestamp and device name. Enter the following command into a new cell and run it.

```
t_data = temp_data.select(
 'deviceName',
 'sensorReadingDate',
 'averageTemp'
                )

h_data = humidity_data.select(
 'deviceName',
 'sensorReadingDate',
 'averageHumidity')

data = t_data.join(h_data, on=['sensorReadingDate', 'deviceName'])
```

Step 16

Now we can plot the contents of our joined table. This time we're encoding the temperature/humidity in the x/y position of circular marks on our graph, and the device name in the colour of the graph. Enter the following command into a new cell and run it.

```
alt.Chart(data).mark_circle().properties(
    width=800,
    height=800
).encode(
    x='averageTemp:Q',
    y='averageHumidity:Q',
    color='deviceName:N',
).interactive()
```

Step 17

Once again the graph we've made could be better. There's too much data for us to differentiate between the different devices. To make things clearer, let's add a 'selection'. This will allow us to select a device name from the legend, and filter out the data for that device in real time on the graph. We'll also use the 'opacity' encoding channel to make the filter visible, and use the 'add_param' method to bind the selection to our graph. Modify the cell you just ran with the changes shown here.

```
selection = alt.selection_point(fields=['deviceName'], bind='legend')

alt.Chart(
    data
).mark_circle(
).properties(
    width=800,
    height=800
).encode(
    alt.X('averageTemp:Q', scale=alt.Scale(zero=False)),
    alt.Y('averageHumidity:Q', scale=alt.Scale(zero=False)),
    color='deviceName:N',
    opacity=alt.condition(selection, alt.value(1), alt.value(0.1))
).add_params(
    selection
).interactive()
```

Try selecting the various different device names. By looking at the visualised data, can you make a guess at the relationship between temperature and humidity?

Step 18

Using what you've learned about so far, find the device name for one of the devices in our dataframe. The following code applies a *filter* to the data, removing any rows that don't belong to the device you've chosen. It then renames some of the columns present in the dataframe for later on, and converts it to 'pandas' format. These changes allow us to apply Prophet to the data, and make predictions about future behaviour. Enter the following command into a new cell and run it.

```
series = data.filter(
    pl.col.deviceName == YOUR_DEVICE_NAME
).select(
    pl.col.sensorReadingDate.alias('ds'),
    pl.col.averageTemp.alias('y')
).to_pandas()
```

Step 19

Now that we have a properly formatted time series, we can use Prophet to make predictions. Here we're creating an instance of Prophet 'm', then fitting it to the past contents of our dataframe. We can then create a new dataframe to contain our predictions up to 72 hours in the future, and then fill it with those predictions. Enter the following command into a new cell and run it.

```
m = Prophet().fit(series)

future = m.make_future_dataframe(periods=72, freq='h')

forecast = m.predict(future)
```

Step 20

Once that's run, we're in a position to plot and evaluate our predictions. Let's create two chart instances, one for our model predictions, and one for our real data. We can then overlay these two plots on top of each other using the + operator. Enter the following command into a new cell.

```
predicted_data =
alt.Chart(forecast).properties(width=1300).mark_line().encode(x='ds',
y='yhat')

real_data =
alt.Chart(series).properties(width=1300).mark_line(color='red').encode(x='ds'
, y='y')

predicted_data + real_data
```

Run the code, and you should see a new chart containing both the past behaviour of your sensor, and three days worth of predictions.

At this point, you might try using the same steps to plot and predict the humidity data for one of your devices. You could also try increasing the range of the prediction to several weeks, months or even years into the future. Can you see anything strange about the predictions that the model starts to make? Why might this be happening? Can you think of a way to fix this?

Well done! You have now completed this challenge.