Smart Cities Hackathon
Tier 3 Challenge 12

Tier 3 Challenge 12: 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.

Step 1

For this challenge, we're going to use a project management tool for Python called uv. Install uv on your machine using one of the following commands in your terminal:

MacOS/Linux:
curl -LsSf https://astral.sh/uv/install.sh | sh

windows:
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"

Step 2

Create a new folder on your computer for Tier 3 Challenge 12 by running the command 'uv init challenge_12' this should create a brand new python project for you to build out this challenge in.

Step 3

Navigate into the folder and run the command 'uv add marimo'. This will add Marimo, a modern Python notebook with powerful features, to your project as a dependency.

Step 4

Run the command 'uv run marimo edit' from inside your new folder, and click on the url displayed in the terminal. Your notebook environment should open up in your system browser. From here, make a new notebook.

Step 5

Within your new notebook you should see a cell with the contents '`import marimo as mo`'. Replace the contents of the cell with the following imports and press the play button in the top right corner of the cell to run it.

```
import marimo as mo
import polars as pl
import altair as alt
import requests
import json
from prophet import Prophet
```

You should see a popup warning you that several of these packages were not found, and asking if you would like to install them. Make sure that the ‘Install with’ dropdown in the popup is set to ‘uv’, then click Install.

Step 6

We’re now ready to download some data. You can create a new cell by either clicking ctrl-shift-p, or clicking the circular ‘+’ button to the lower left of the current cell, or clicking the ‘Python’ option at the bottom of the last cell in the notebook. Create a new cell and enter the following code. For now, leave the placeholder ‘YOUR_BEARER_TOKEN’ as-is.

```
url = 'https://api.core.aql.com/v1/sensors/sensor-
data/aggregate/min_ave_max'

headers = {
    'Accept': 'application/json',
    'Authorization': 'Bearer YOUR_BEARER_TOKEN',
    'Content-Type': 'application/json'
}
```

Step 7

Create a new cell, and enter the following into it

```
_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"
}

_response = requests.post(url, headers=headers, json=_data)

if _response.status_code == 200:
    # Save the response to a file
    with open('temp_data.json', 'w') as _f:
        json.dump(_response.json(), _f, indent=4)
    print("Data saved to temp_data.json")
else:
```

```
    print(f"Request failed with status code: {_response.status_code}")
    print(_response.text)
```

Running the cell should give you an error:

```
Request failed with status code: 401
{"message":"Unauthenticated."}
```

Step 8

Now enter your bearer token into the placeholder that we left earlier, and run only that cell. You should see the cell that previously printed an error message automatically rerun, and print out a new message (hopefully ‘Data saved to temp_data.json’). This is a cool feature of Marimo notebooks! They automatically track variable dependencies between cells, meaning any changes that you make earlier on in the notebook will intelligently be propagated through the correct cells. One side effect of this is that we can’t have two cells that assign the same variable. We can get around this by adding an underscore _ to the start of a variable name, which automatically untracks it.

Step 9

Create a new cell and enter the followin into it:

```
_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"
}


# Send the POST request
_response = requests.post(url, headers=headers, json=_data)


# Check if the request was successful
if _response.status_code == 200:
    # Save the response to a file
    with open('humidity_data.json', 'w') as _f:
        json.dump(_response.json(), _f, indent=4)
    print("Data saved to humidity_data.json")
else:
```

```
    print(f"Request failed with status code: {_response.status_code}")
    print(_response.text)
```

Step 10

You should now have two files, temp_data.json and humidity_data.json visible in your file browser. Let's load our data into a polars dataframe. You might have spotted that the two files have some of the same column names. To avoid confusion moving forward, let's also rename the columns we'll be working with to something more distinctive. Create a new cell and enter the following:

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

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

Step 11

Let's have a look at our dataframes. Create two new cells and enter the following:

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

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

Marimo will always output the last expression written in each cell below that cell. In this case, this allows us to have a look at the contents of our dataframes as easy-to-read tables. Try replacing the sample method with 'describe()' to get a nice statistical overview of the contents of your data. See if you can get a feel for the data we're working with. 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

It’s time to visualise our data! Create a new cell, then enter and run the following

```
alt.Chart(temp_data).mark_line().encode(
    x='sensorReadingDate',
    y='averageTemp',
)
```

This should give us a nice overview of all the temperature data we have. Already we should be able to see some patterns, like daily temperature fluctuations and a few cloudy days at the start of the period. This chart has some issues though… there’s no way to distinguish between different sensors, and the chart clearly hasn’t scaled the X axis properly. Let’s fix these.

Step 13

Create a new cell and copy your chart code across. We can add two new encodings, color and tooltip, to make the chart a little more informative. Then we can add a title to ensure that we know what we’re looking at if we ever need to reopen the notebook, and reduce the opacity of the lines a little to make it easier on the eyes. Finally, we can add the ‘:T’ annotation to the x axis to let altair know that this is Time data.

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

This should give us a much easier to read chart. Try hovering your mouse over some of the data points.

Step 14

We can make this chart even nicer using some of the features of Marimo notebooks. Wrap the chart in a ‘mo.ui.altair_chart()’ function like this:

```
```

```
mo.ui.altair_chart(
    alt.Chart(temp_data, title='Temperature
Data').mark_line(opacity=0.6).encode(
    x='sensorReadingDate:T',
    y='averageTemp',
    color='sensor_id',
    tooltip=['sensorReadingDate', 'averageTemp', 'sensor_id']
))
```

This chart looks a lot like the old one, with two handy features. First, it scales to fit the browser window automatically. Second, clicking on an entry in the sensor_id legend highlights the data for the associated sensor, which means we can get a better look at our individual sensor readings.

Step 15
Let's try doing the same for humidity:

```
mo.ui.altair_chart(alt.Chart(humidity_data).mark_line(opacity=0.6).encode(
    x='sensorReadingDate:T',
    y='averageHumidity',
    color='sensor_id',
    tooltip=['sensorReadingDate', 'averageHumidity', 'sensor_id']
))
```

Step 16
Let's try joining our temperature and humidity data together on the basis of device name. Enter the following into a 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 17
We now have a dataframe that contains both temperature and humidity data for each device, paired up by date. We can plot a new chart that lets us see all this information at once:

```
chart = mo.ui.altair_chart(
    alt.Chart(data)
    .mark_circle()
    .properties(width=500, height=500)
    .encode(
```

```
        alt.X("averageTemp", scale=alt.Scale(zero=False)),
        alt.Y("averageHumidity", scale=alt.Scale(zero=False)),
        color="deviceName",
        tooltip=["sensorReadingDate", "deviceName"]
    )
)

chart
```

This should give us a nice overview of the data. By looking at the visualised data, can you make a guess at the relationship between temperature and humidity?

Step 18

You might have noticed that in the previous step, we bound the chart to a variable and placed that variable at the end of the cell, instead of placing the chart there directly. This means we have an altair chart variable which is accessible from other cells, which allows us to do something interesting. Make a new cell with the following in it:

```
chart.value
```

Once you've run this cell, try selecting a section of the temperature-humidity graph with your mouse. The chart.value cell should dynamically populate with a table representing your selection in the chart.

Step 19

It's time to get ready to do some timeseries prediction using Prophet, a tool from Meta. First lets set up our timeseries data with appropriate column names, and plot it to make sure it makes sense. Enter the following into a cell and run it:

```
series = data.select(
    pl.col.sensorReadingDate.alias("ds"),
    pl.col.averageTemp.alias("y"),
    pl.col.deviceName,
)

series_chart = mo.ui.altair_chart(
    alt.Chart(series.with_columns(pl.col.ds.str.to_datetime()))
    .mark_line()
    .encode(x="ds", y="y", color="deviceName")
)

series_chart
```

Step 20

We're now in possession of properly formatted timeseries data, meaning that we can use Prophet to make predictions. Enter the following code into a cell and run it:

```
predicted_data = real_data = (
        alt.Chart(series_chart.value)
        .mark_line(color="orange", opacity=0.6)
        .encode(x="ds", y="y", tooltip=["ds", "y"])
    )

if not series_chart.value.is_empty():
    series_selection = series_chart.value.with_columns(
        pl.col.ds.cast(pl.String)
    )

    m = Prophet().fit(series_selection.to_pandas())

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

    forecast = m.predict(future)

    predicted_data = (
        alt.Chart(forecast)
        .mark_line(opacity=0.6)
        .encode(x="ds", y="yhat", tooltip=["ds", "yhat"])
    )

mo.ui.altair_chart(predicted_data + real_data)
```

Initially, this code should output an empty chart. This is because we've bound the data that we're using to make our prediction to the value of our series chart. Try clicking on an entry in the legend of the last timeseries chart that we just plotted. This should populate the value of our chart selection, and trigger Prophet to create a prediction for the device that we've selected.

Step 21

What if we want to see the prediction for our timeseries further than 72 hours into the future? We could just increase the 'periods' value of our prophet code, but let's do something more interesting. Create a new cell and paste in the following:

```
slider = mo.ui.slider(start=0, stop=1000, step = 20)
slider
```

This should give us a slider ui element that we can use to set values defined in our code.

Step 22

Find the ‘periods’ variable in our Prophet cell and change its value from ‘72’ to ‘slider.value’. Rerun the cell. Now try adjusting the slider. You can use this to increase 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?

At this point, you might try using the same steps to plot and predict the humidity data for one of your devices.

Well done! You have now completed this challenge.

Bonus Steps

The chart we’ve created is pretty small, and checking that the predictions of Prophet make sense alongside our real data is tricky. Can you figure out a way to zoom in on the data we’re interested in?

We’ve made predictions for our temperature data. Can you make the same kind of predictions for our humidity data? Remember that copying code blocks without changing the variable names won’t work, as Marimo doesn’t allow for redefining variables between cells.