Inroduction

This sample notebook demonstrates how to process live data streams using Pathway. The dataset used here is a subset of the one provided — specifically, it includes data for only a single parking spot. You are expected to implement your model across all parking spots.

Please note that the pricing model used in this notebook is a simple baseline. You are expected to design and implement a more advanced and effective model.

```
# ---------------------------
# Model 1: Baseline Linear Pricing
# ---------------------------

def model1_linear_price(prev_price, occupancy, capacity, alpha=5.0):
    """Simple linear pricing model based on occupancy."""
    return prev_price + alpha * (occupancy / capacity)
```

```
# ---------------------------
# Model 2: Demand-Based Pricing
# ---------------------------

def model2_demand_price(base_price, occupancy, capacity, queue, traffic, special_day, vehicle_
    """Demand-based dynamic pricing model."""
    vehicle_weights = {'car': 1.0, 'bike': 0.5, 'truck': 1.5}
    traffic_weights = {'low': 0.5, 'medium': 1.0, 'high': 1.5}

    demand = (
        2 * (occupancy / capacity) +
        1.5 * queue -
        1.2 * traffic_weights.get(traffic, 1.0) +
        2.0 * special_day +
        1.0 * vehicle_weights.get(vehicle_type, 1.0)
    )

    # Normalize demand and bound the price between $5 and $20
    norm_demand = (demand - 2) / 10
    price = base_price * (1 + 0.5 * norm_demand)
    return max(5, min(20, price))
```

```
!pip install pathway bokeh --quiet # This cell may take a few seconds to execute.
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import datetime
from datetime import datetime
import pathway as pw
import bokeh.plotting
import panel as pn
```

Step 1: Importing and Preprocessing the Data

```
df = pd.read_csv('/content/dataset.csv')
df

# You can find the sample dataset here: https://drive.google.com/file/d/1D479FLjp9aO3Mg8g6Lpj9
```

	ID	SystemCodeNumber	Capacity	Latitude	Longitude	Occupancy	VehicleType	TrafficCo
0	0	BHMBCCMKT01	577	26.144536	91.736172	61	car	
1	1	BHMBCCMKT01	577	26.144536	91.736172	64	car	
2	2	BHMBCCMKT01	577	26.144536	91.736172	80	car	
3	3	BHMBCCMKT01	577	26.144536	91.736172	107	car	
4	4	BHMBCCMKT01	577	26.144536	91.736172	150	bike	
...	...	...	...	...	...	...	...	
18363	18363	Shopping	1920	26.150504	91.733531	1517	truck	
18364	18364	Shopping	1920	26.150504	91.733531	1487	car	
18365	18365	Shopping	1920	26.150504	91.733531	1432	cycle	
18366	18366	Shopping	1920	26.150504	91.733531	1321	car	
18367	18367	Shopping	1920	26.150504	91.733531	1180	car	

18368 rows × 12 columns

```
# Combine the 'LastUpdatedDate' and 'LastUpdatedTime' columns into a single datetime column
df['Timestamp'] = pd.to_datetime(df['LastUpdatedDate'] + ' ' + df['LastUpdatedTime'],
                                 format='%d-%m-%Y %H:%M:%S')

# Sort the DataFrame by the new 'Timestamp' column and reset the index
df = df.sort_values('Timestamp').reset_index(drop=True)
```

```
# Save the selected columns to a CSV file for streaming or downstream processing
df[["Timestamp", "Occupancy", "Capacity"]].to_csv("parking_stream.csv", index=False)

# Note: Only three features are used here for simplicity.
# Participants are expected to incorporate additional relevant features in their models.
```

```
# Define the schema for the streaming data using Pathway
# This schema specifies the expected structure of each data row in the stream

class ParkingSchema(pw.Schema):
    Timestamp: str   # Timestamp of the observation (should ideally be in ISO format)
    Occupancy: int   # Number of occupied parking spots
    Capacity: int    # Total parking capacity at the location
```

```
# Load the data as a simulated stream using Pathway's replay_csv function
# This replays the CSV data at a controlled input rate to mimic real-time streaming
# input_rate=1000 means approximately 1000 rows per second will be ingested into the stream.

data = pw.demo.replay_csv("parking_stream.csv", schema=ParkingSchema, input_rate=1000)
```

```
# Define the datetime format to parse the 'Timestamp' column
fmt = "%Y-%m-%d %H:%M:%S"

# Add new columns to the data stream:
# - 't' contains the parsed full datetime
# - 'day' extracts the date part and resets the time to midnight (useful for day-level aggrega
data_with_time = data.with_columns(
    t = data.Timestamp.dt.strptime(fmt),
    day = data.Timestamp.dt.strptime(fmt).dt.strftime("%Y-%m-%dT00:00:00")
)
```

```
print(data_with_time.schema)
```

```
id          | Timestamp | Occupancy | Capacity | t                | day
ANY_POINTER | STR       | INT       | INT      | DATE_TIME_NAIVE | STR
```

Step 2: Making a simple pricing function

```
import pathway as pw
import datetime

# Step 1: Create extended data with dummy columns
extended_data = data_with_time.with_columns(
    # Dummy vehicle_type based on capacity % 3
    vehicle_type = pw.if_else(
        pw.this.Capacity % 3 == 0, "car",
        pw.if_else(pw.this.Capacity % 3 == 1, "bike", "truck")
    ),

    queue = pw.this.Capacity % 3,
    traffic = pw.this.Capacity % 5,
    is_special_day = pw.this.day.str.endswith("01"),

    vehicle_type_weight = pw.if_else(
        pw.this.Capacity % 3 == 0, 1.0,
        pw.if_else(pw.this.Capacity % 3 == 1, 0.5, 1.5)
    )
)

# Step 2: Apply window + models on extended data
delta_window = (
    extended_data.windowby(
        pw.this.t,
        instance=pw.this.day,
        window=pw.temporal.tumbling(datetime.timedelta(days=1)),
        behavior=pw.temporal.exactly_once_behavior()
    )
    .reduce(
```

```
        t=pw.this._pw_window_end,
        occ_max=pw.reducers.max(pw.this.Occupancy),
        occ_min=pw.reducers.min(pw.this.Occupancy),
        occ_sum=pw.reducers.sum(pw.this.Occupancy),
        count=pw.reducers.count(),
        cap=pw.reducers.max(pw.this.Capacity),
        queue_sum=pw.reducers.sum(pw.this.queue),
        traffic_sum=pw.reducers.sum(pw.this.traffic),
        special_day=pw.reducers.max(pw.this.is_special_day),
        vehicle_type_weight_sum=pw.reducers.sum(pw.this.vehicle_type_weight),
    )
    .with_columns(
        model1_price=10 + 5.0 * ((pw.this.occ_sum / pw.this.count) / pw.this.cap),

        model2_price = 10 * (
    1 + 0.5 * (
        0.4 * ((pw.this.occ_sum / pw.this.count) / pw.this.cap)
        + 0.3 * (pw.this.queue_sum / pw.this.count)
        - 0.2 * (pw.this.traffic_sum / pw.this.count)
        + 0.2 * pw.cast(float, pw.this.special_day)
        + 0.1 * (pw.this.vehicle_type_weight_sum / pw.this.count)
    )
)


    )
)
```

Step 3: Visualizing Daily Price Fluctuations with a Bokeh Plot

Note: The Bokeh plot in the next cell will only be generated after you run the `pw.run()` cell (i.e., the final cell).

```
import panel as pn
import bokeh.plotting

pn.extension()

# Define the plotter to visualize model1 and model2 prices
def price_plotter(source):
    fig = bokeh.plotting.figure(
        height=400,
        width=800,
        title="Pathway: Daily Parking Price (Model 1 & 2)",
        x_axis_type="datetime",
    )

    # Plot both models
    fig.line("t", "model1_price", source=source, line_width=2, color="green", legend_label="Mo
    fig.line("t", "model2_price", source=source, line_width=2, color="navy", legend_label="Moo
```

```
    # Use scatter instead of circle
    fig.scatter(x="t", y="model1_price", source=source, size=6, color="yellow", marker="circle
    fig.scatter(x="t", y="model2_price", source=source, size=6, color="red", marker="circle")

    fig.legend.location = "top_left"

    return fig

# Connect Pathway stream to the visual function
viz = delta_window.plot(price_plotter, sorting_col="t")

# Display in notebook or as app
pn.Column(viz).servable()
```

```
# Start the Pathway pipeline execution in the background
# - This triggers the real-time data stream processing defined above
# - %%capture --no-display suppresses output in the notebook interface

%%capture --no-display
pw.run()
```

```
Output()
```