Introduction

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.

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

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.4/60.4 kB 2.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.4/149.4 kB 7.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 69.7/69.7 MB 12.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.6/77.6 kB 6.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 777.6/777.6 kB 47.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.2/139.2 kB 12.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.5/26.5 MB 49.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.5/45.5 kB 3.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.3/135.3 kB 11.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 244.6/244.6 kB 19.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 318.4/318.4 kB 25.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 985.8/985.8 kB 35.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 148.6/148.6 kB 11.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.8/139.8 kB 13.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.8/65.8 kB 5.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.7/55.7 kB 5.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.5/118.5 kB 11.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.2/196.2 kB 17.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 434.9/434.9 kB 31.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 43.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 49.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 51.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 83.2/83.2 kB 7.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 50.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 44.4 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the package
bigframes 2.8.0 requires google-cloud-bigquery[bqstorage,pandas]>=3.31.0, but you hav
```

```
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/Modified - modified.csv')
df

# You can find the sample dataset here: https://drive.google.com/file/d/1D479FLjp9aO3Mg8g
```

	Unnamed: 0	SystemCodeNumber	Capacity	Occupancy	LastUpdatedDate	LastUpdatedTi
0	0	BHMBCCMKT01	577	61	04-10-2016	07:59:
1	1	BHMBCCMKT01	577	64	04-10-2016	08:25:
2	2	BHMBCCMKT01	577	80	04-10-2016	08:59:
3	3	BHMBCCMKT01	577	107	04-10-2016	09:32:
4	4	BHMBCCMKT01	577	150	04-10-2016	09:59:
...	...	...	...	...	...	
1307	1307	BHMBCCMKT01	577	309	19-12-2016	14:30:
1308	1308	BHMBCCMKT01	577	300	19-12-2016	15:03:
1309	1309	BHMBCCMKT01	577	274	19-12-2016	15:29:
1310	1310	BHMBCCMKT01	577	230	19-12-2016	16:03:
1311	1311	BHMBCCMKT01	577	193	19-12-2016	16:30:

1312 rows × 12 columns

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
# Combine the 'LastUpdatedDate' and 'LastUpdatedTime' columns into a single datetime colu
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
# Include ALL required columns when creating the streaming CSV
df[["Timestamp", "Occupancy", "Capacity", "QueueLength",
    "TrafficConditionNearby", "IsSpecialDay", "VehicleType"]].to_csv("parking_stream.csv"
```

```
# 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
    QueueLength: int # Length of the queue for the parking spot
    TrafficConditionNearby: str # Traffic condition nearby
    IsSpecialDay: int # Whether it's a special day (1 if yes, 0 if no)
    VehicleType: str # Type of vehicle
```

```
# 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 stre

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 ag
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")
)
```

Step 2: Making a simple pricing function

```
# Define a daily tumbling window over the data stream using Pathway
# This block performs temporal aggregation and computes a dynamic price for each day
import datetime

delta_window = (
    data_with_time.windowby(
        pw.this.t,  # Event time column to use for windowing (parsed datetime)
        instance=pw.this.day,  # Logical partitioning key: one instance per calendar day
        window=pw.temporal.tumbling(datetime.timedelta(days=1)),  # Fixed-size daily wind
        behavior=pw.temporal.exactly_once_behavior(),  # Guarantees exactly-once processin
```

```
        behavior=pw.temporal.exactly_once_behavior()  # Guarantees exactly-once processin
    )
    .reduce(
        t=pw.this._pw_window_end,                       # Assign the end timestamp of ea
        occ_max=pw.reducers.max(pw.this.Occupancy),     # Highest occupancy observed in
        occ_min=pw.reducers.min(pw.this.Occupancy),     # Lowest occupancy observed in t
        cap=pw.reducers.max(pw.this.Capacity),          # Maximum capacity observed (typ
    )
    .with_columns(
        # Compute the price using a simple dynamic pricing formula:
        #
        # Pricing Formula:
        #     price = base_price + demand_fluctuation
        #     where:
        #         base_price = 10 (fixed minimum price)
        #         demand_fluctuation = (occ_max - occ_min) / cap
        #
        # Intuition:
        # - The greater the difference between peak and low occupancy in a day,
        #   the more volatile the demand is, indicating potential scarcity.
        # - Dividing by capacity normalizes the fluctuation (to stay in [0,1] range).
        # - This fluctuation is added to the base price of 10 to set the final price.
        # - Example: If occ_max = 90, occ_min = 30, cap = 100
        #            => price = 10 + (90 - 30)/100 = 10 + 0.6 = 10.6

        price=10 + (pw.this.occ_max - pw.this.occ_min) / pw.this.cap
    )
)
```

Model 2: Demand-Based Pricing

Demand-based pricing dynamically adjusts parking fees in real time according to current demand, occupancy, and other influencing factors. This approach uses data such as occupancy rates, time of day, and special events to set prices that help balance parking supply and demand. By increasing prices during peak demand and lowering them when demand is low, this model aims to:

```
* Improve space utilization and parking availability,

* Reduce congestion and cruising for parking,

* Optimize revenue for operators.
```

Real-world implementations, such as Los Angeles' Express Park, have shown that demand-

Real world implementations, such as Los Angeles' Express Park, have shown that demand based pricing can reduce parking duration by 37%, increase availability by 10%, and boost revenues by 16% . Theoretical models often target an optimal occupancy rate (e.g., 85%) by adjusting prices in response to real-time data, thereby ensuring efficient and fair use of limited urban parking resources.

```
# Debug: Check what columns are actually available
print("Original DataFrame columns:", df.columns.tolist())
print("Pathway table columns:", data_with_time.keys())

# Check CSV content
test_df = pd.read_csv("parking_stream.csv")
print("CSV columns:", test_df.columns.tolist())
```

```
Original DataFrame columns: ['Unnamed: 0', 'SystemCodeNumber', 'Capacity', 'Occupancy
Pathway table columns: dict_keys(['Timestamp', 'Occupancy', 'Capacity', 'QueueLength'
CSV columns: ['Timestamp', 'Occupancy', 'Capacity', 'QueueLength', 'TrafficConditionN
```

```
# Calculate demand-based pricing within the window context
demand_window = (
    data_with_time.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,
        # Calculate demand metrics within the window
        avg_occupancy_rate=pw.reducers.avg(pw.this.Occupancy / pw.this.Capacity),
        avg_queue=pw.reducers.avg(pw.this.QueueLength),
        avg_traffic=pw.reducers.avg(
            pw.if_else(pw.this.TrafficConditionNearby == "High", 1.0,
                      pw.if_else(pw.this.TrafficConditionNearby == "Medium", 0.5, 0.0))
        ),
        avg_special_day=pw.reducers.avg(pw.this.IsSpecialDay)
    )
    .with_columns(
        # Calculate demand using window aggregates
        demand_score = (
            1.0 * pw.this.avg_occupancy_rate +
            0.5 * pw.this.avg_queue +
            0.3 * pw.this.avg_traffic +
            2.0 * pw.this.avg_special_day
        )
    )
    .with_columns(
        # Calculate final price
        price = 10 * (1 + 0.5 * pw.this.demand_score)
```

```
        price = 10 * (1 + 0.5 * pw.this.demand_score)
    )
    .with_columns(
        # Ensure price bounds
        price = pw.if_else(pw.this.price < 5.0, 5.0,
                           pw.if_else(pw.this.price > 20.0, 20.0, pw.this.price))
    )
)
```

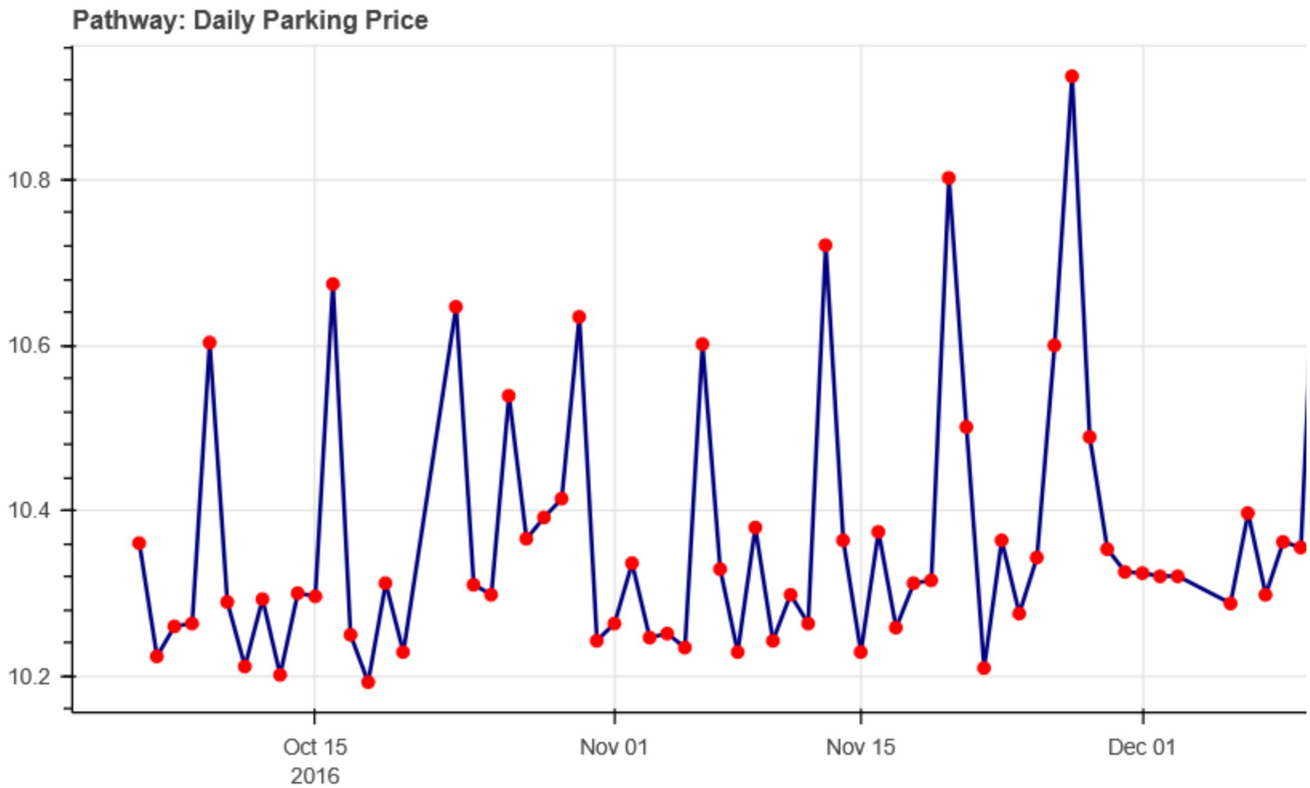
Step 3: Visualizing Daily Price Fluctuations with a Bokeh Plot

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

```
# Activate the Panel extension to enable interactive visualizations
pn.extension()

# Define a custom Bokeh plotting function that takes a data source (from Pathway) and retur
def price_plotter(source):
    # Create a Bokeh figure with datetime x-axis
    fig = bokeh.plotting.figure(
        height=400,
        width=800,
        title="Pathway: Daily Parking Price",
        x_axis_type="datetime",  # Ensure time-based data is properly formatted on the x-ax
    )
    # Plot a line graph showing how the price evolves over time
    fig.line("t", "price", source=source, line_width=2, color="navy")

    # Overlay red circles at each data point for better visibility
    fig.circle("t", "price", source=source, size=6, color="red")

    return fig

# Use Pathway's built-in .plot() method to bind the data stream (delta_window) to the Bokeh
# - 'price_plotter' is the rendering function
# - 'sorting_col="t"' ensures the data is plotted in time order
viz = delta_window.plot(price_plotter, sorting_col="t")

# Create a Panel layout and make it servable as a web app
# This line enables the interactive plot to be displayed when the app is served
pn.Column(viz).servable()
```

```
BokehDeprecationWarning: 'circle() method with size value' was deprecated in Bokeh 3.
```

Streaming mode




```
# 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()
```

```
WARNING:pathway_engine.connectors.monitoring:PythonReader: Closing the data source
```