```
!pip install pathway bokeh --quiet
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.4/60.4 kB 1.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 149.4/149.4 kB 5.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 69.7/69.7 MB 11.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.6/77.6 kB 5.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 777.6/777.6 kB 36.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.2/139.2 kB 9.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.5/26.5 MB 57.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.5/45.5 kB 3.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.3/135.3 kB 10.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 244.6/244.6 kB 16.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 319.1/319.1 kB 18.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 985.8/985.8 kB 41.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 148.6/148.6 kB 11.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 139.8/139.8 kB 9.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.8/65.8 kB 4.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.7/55.7 kB 3.8 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.5/118.5 kB 8.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.2/196.2 kB 11.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 434.9/434.9 kB 24.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 53.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.7/2.7 MB 55.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 63.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 83.2/83.2 kB 1.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 60.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 52.1 MB/s eta 0:00:00
ERROR: pip's dependency resolver does not currently take into account all the packages t
bigframes 2.8.0 requires google-cloud-bigquery[bqstorage,pandas]>=3.31.0, but you have g
```

```
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
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files dataset.csv

- **dataset.csv**(text/csv) - 1595541 bytes, last modified: 7/1/2025 - 100% done

```
Saving dataset.csv to dataset.csv
```

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

	ID	SystemCodeNumber	Capacity	Latitude	Longitude	Occupancy	VehicleType
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

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

```
lotss=df['SystemCodeNumber'].unique()
lots=lotss[0:14]    #making a list of parking lots to get plots one by one
lotss
```

```
array(['BHMBCCMKT01', 'BHMBCCTHL01', 'BHMEURBRD01', 'BHMMBMMBX01',
       'BHMNCPHST01', 'BHMNCPNST01', 'Broad Street', 'Others-CCCPS105a',
       'Others-CCCPS119a', 'Others-CCCPS135a', 'Others-CCCPS202',
       'Others-CCCPS8', 'Others-CCCPS98', 'Shopping'], dtype=object)
```

```
df["TrafficCondition_Code"] = df["TrafficConditionNearby"].astype("category").cat.codes  # ]
df["vehicle_Code"] = df["VehicleType"].astype("category").cat.codes  #car:1,bike:0,truck:3:c
df
```

	ID	SystemCodeNumber	Capacity	Latitude	Longitude	Occupancy	VehicleType
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 × 14 columns

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

```
plots=[]

for lot in lots:
  import datetime
  df['Timestamp'] = pd.to_datetime(df['LastUpdatedDate'] + ' ' + df['LastUpdatedTime'],   #c
                                   format='%d-%m-%Y %H:%M:%S')
  df_lot=df[df['SystemCodeNumber']==lot]   #making dataframe for one lot at a time
  df_lot = df_lot.sort_values('Timestamp').reset_index(drop=True) #sorting time
  class ParkingSchema(pw.Schema):
    Capacity:int
    Occupancy:int
    Timestamp:str
    SystemCodeNumber:str
    VehicleType:str
    TrafficConditionNearby:str
    QueueLength:int
    IsSpecialDay:int
    TrafficCondition_Code:int
    vehicle_Code:int
  filename = f"parking_stream_{lot}.csv"
  df_lot[["Timestamp", "Occupancy", "Capacity", "SystemCodeNumber","VehicleType","TrafficCor
  data = pw.demo.replay_csv(filename, schema=ParkingSchema, input_rate=1000)   # Load the da
  fmt = "%Y-%m-%d %H:%M:%S"
  data_with_time = data.with_columns(
```

```
    t = data.Timestamp.dt.strptime(fmt),   #containes full datetime
    day = data.Timestamp.dt.strptime(fmt).dt.strftime("%Y-%m-%dT00:00:00"),  #contains onl
    hour = data.Timestamp.dt.strptime(fmt).dt.hour(),   #contains hour
    day_of_week = data.Timestamp.dt.strptime(fmt).dt.weekday(),   #assigns monday: 0,...,s
    occupancy_rate = data.Occupancy / data.Capacity
)
def time_of_day_weight(hour):   #the reason for choosing these categories is explained in

  if 11 <= hour < 14:
      return 1.0    # Midday
  elif 14 <= hour < 17:
      return 0.7    # Evening
  else:
      return 0.4    # Morning

def weekday_weight(day_of_week):

  if day_of_week < 5:
      return 1.0    # Weekday
  else:
      return 0.7    # Weekend
def queue_time_of_day_weight(hour):

  if 12<= hour < 14:
      return 1.0
  else:
      return 0.7
def traffic_time_of_day_weight(hour):

  if 12<= hour < 14:
      return 1.0

  else:
      return 0.5

def hour_effect(hour):
  a = 1.0
  mu = 12
  sigma = 3
  return a * np.exp(-((hour - mu) ** 2) / (2 * sigma ** 2))
beta=1.0
gamma=1.0
delta=1.0
epsilon=1.0
zeta=1.0
vehicle_weights = {'car': 1.0, 'bike': 0.8, 'truck': 1.3, 'cycle': 0.7}
def demand_fn(occ_rate,tod_weight,wd_weight,adj_queue,adj_traffic,hour_term,veh_term,spec_
  return (
     2.0 * occ_rate * tod_weight * wd_weight+beta * np.log1p(adj_queue)+gamma * np.log1p(a
  )
import datetime
```

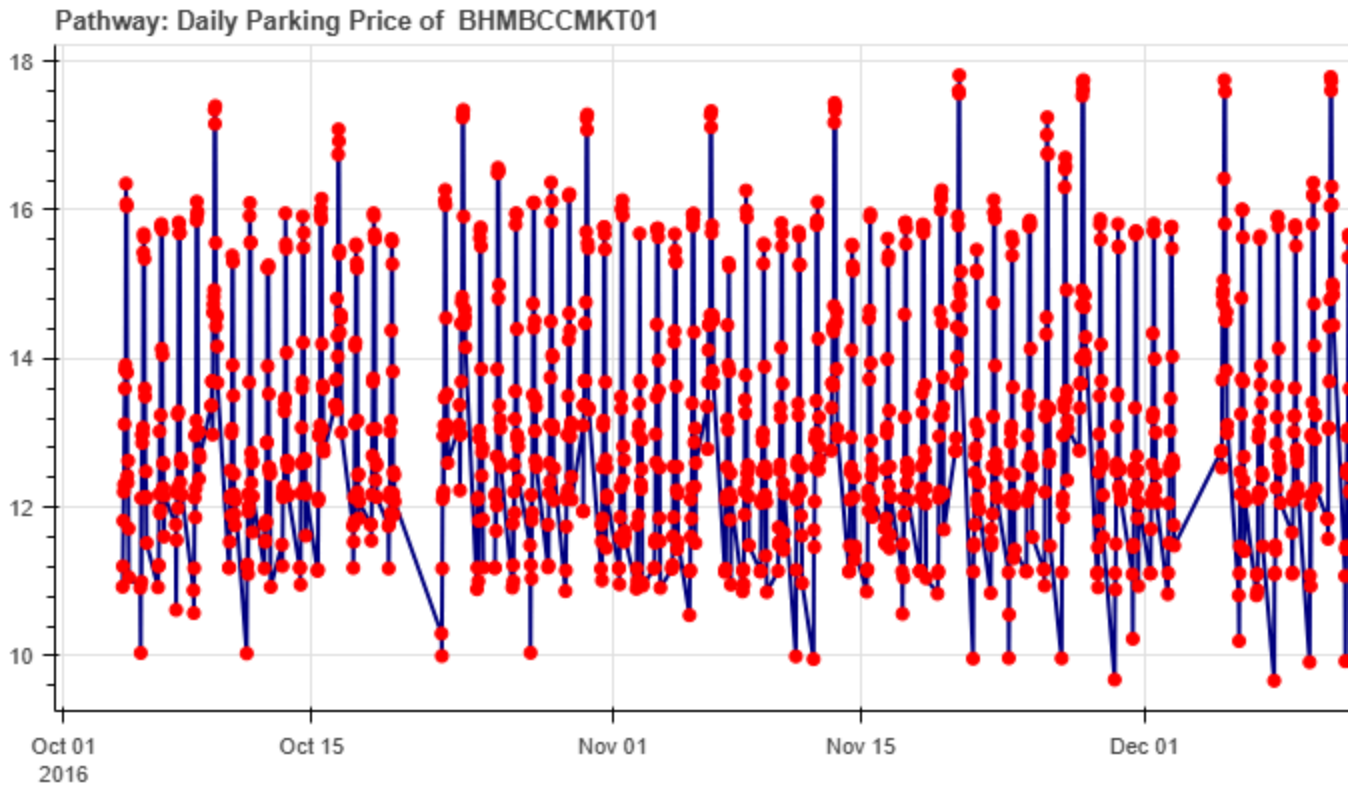
```
data_with_price=(
   data_with_time.with_columns(

       tod_weight = pw.apply(time_of_day_weight, data_with_time.hour),
       wd_weight = pw.apply(weekday_weight, data_with_time.day_of_week),
       queue_weight = pw.apply(queue_time_of_day_weight,data_with_time.hour),
       traffic_weight = pw.apply(traffic_time_of_day_weight,data_with_time.hour),
   )
   .with_columns(

       adj_queue = pw.apply(lambda q, w: q * w, data_with_time.QueueLength, pw.this.queue_
       adj_traffic = pw.apply(lambda t, w: t * w, data_with_time.TrafficCondition_Code, pw

       hour_term = pw.apply(hour_effect,data_with_time.hour),
       veh_term = vehicle_weights.get(data_with_time.VehicleType, 1.0),
       spec_term = data_with_time.IsSpecialDay
   )
   .with_columns(
    demand=pw.apply(demand_fn,
                    data_with_time.occupancy_rate,
                    pw.this.tod_weight,
                     pw.this.wd_weight,
                    pw.this.adj_queue,
                    pw.this.adj_traffic,
                    pw.this.hour_term,
                    pw.this.veh_term,
                    pw.this.spec_term
    )
   )
)
LAMBDA=1.0
```

```
def normalize(d):
  return (d - 2) / (8 - 2)
def price_fn(demand_norm, LAMBDA=LAMBDA):
  return 10 * (1 + LAMBDA * demand_norm)
# aligned_data = data_with_price.demand.with_universe_of(data_with_price)
plot_table=(data_with_price.with_columns(
   # with_price=data_with_price
    demand_norm = pw.apply(normalize,data_with_price.demand)
)
.with_columns(
    price = pw.apply(price_fn, pw.this.demand_norm)
)
)
def price_plotter(source):  #writing bokeh plot fn
```

```
        # Create a Bokeh figure with datetime x-axis
        fig = bokeh.plotting.figure(
            height=400,
            width=800,
            title=f"Pathway: Daily Parking Price of  {lot}",
            x_axis_type="datetime",
        )
        # 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.scatter("t", "price", source=source, size=6, color="red")

        return fig


    viz = plot_table.plot(price_plotter, sorting_col="t")
    plots.append(pn.Column(f"Lot: {lot}", viz.servable()))
dashboard = pn.Column(*plots)
dashboard.servable()
```

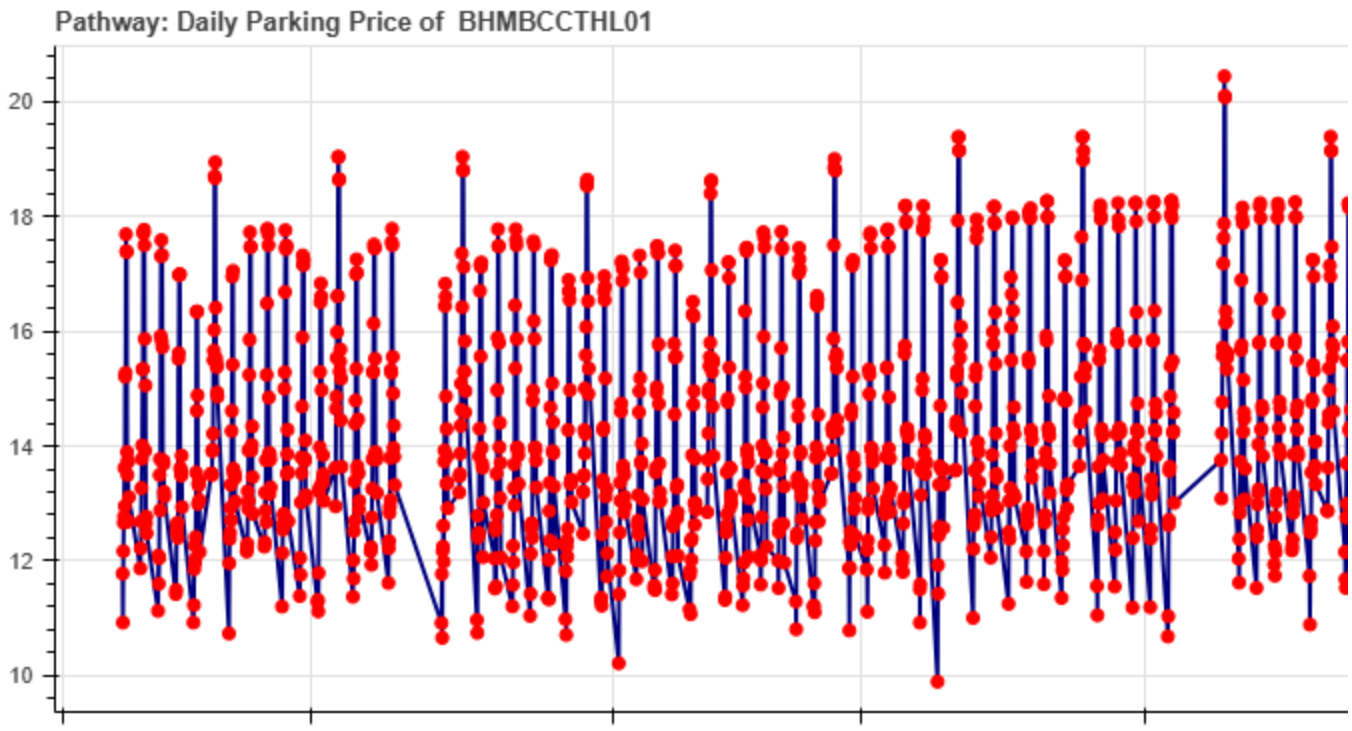
Lot: BHMBCCMKT01

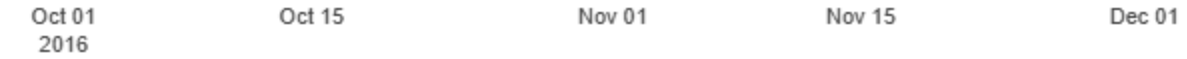
Streaming mode



Lot: BHMBCCTHL01

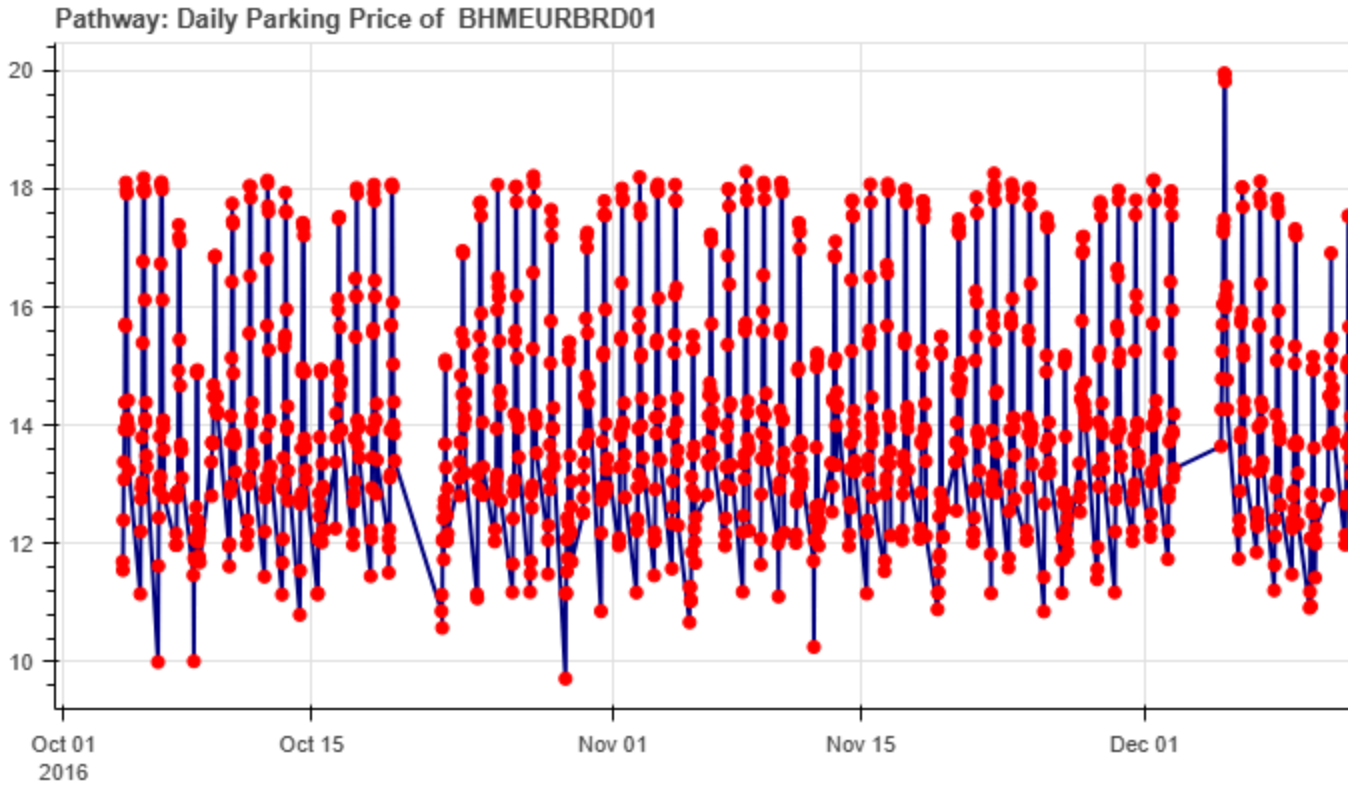
Streaming mode





Lot: BHMEURBRD01

Streaming mode

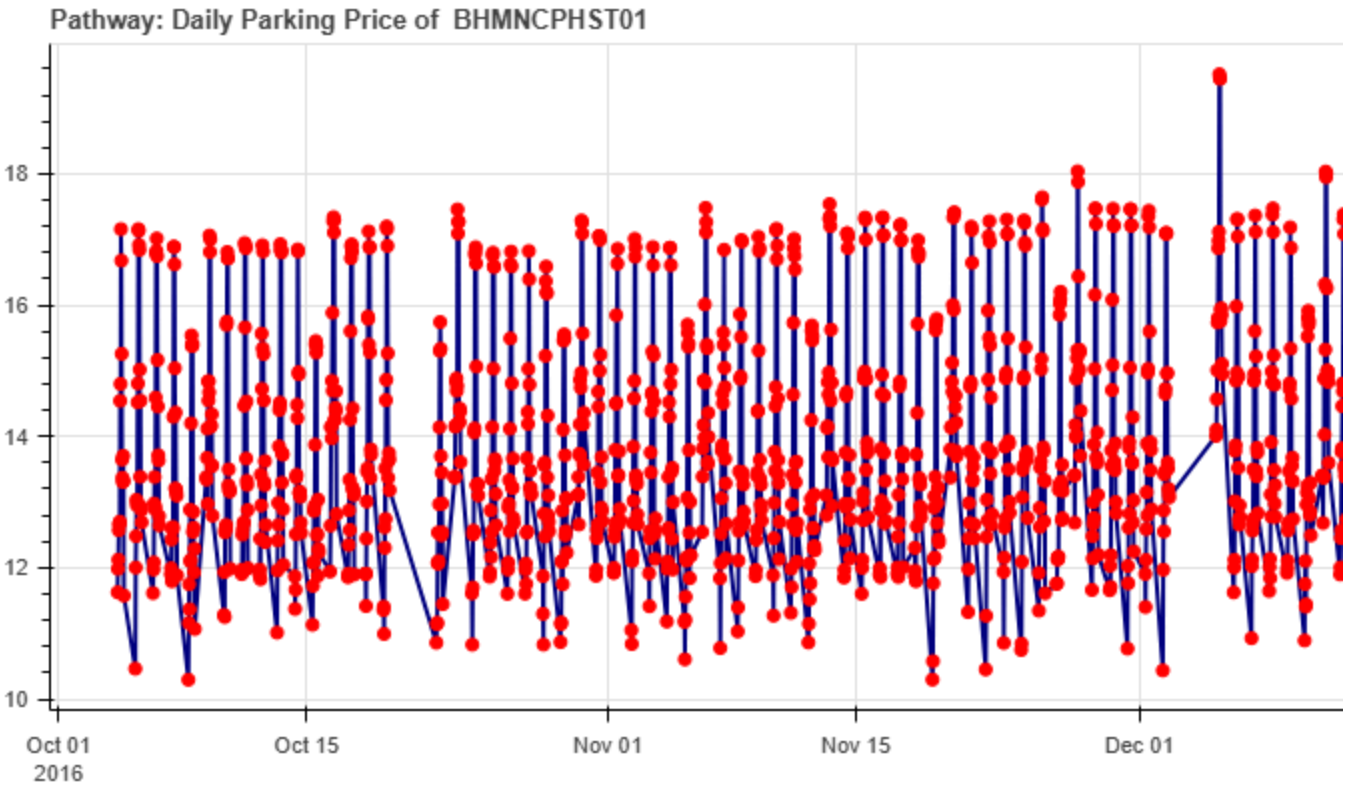


Lot: BHMMBMMBX01

Streaming mode

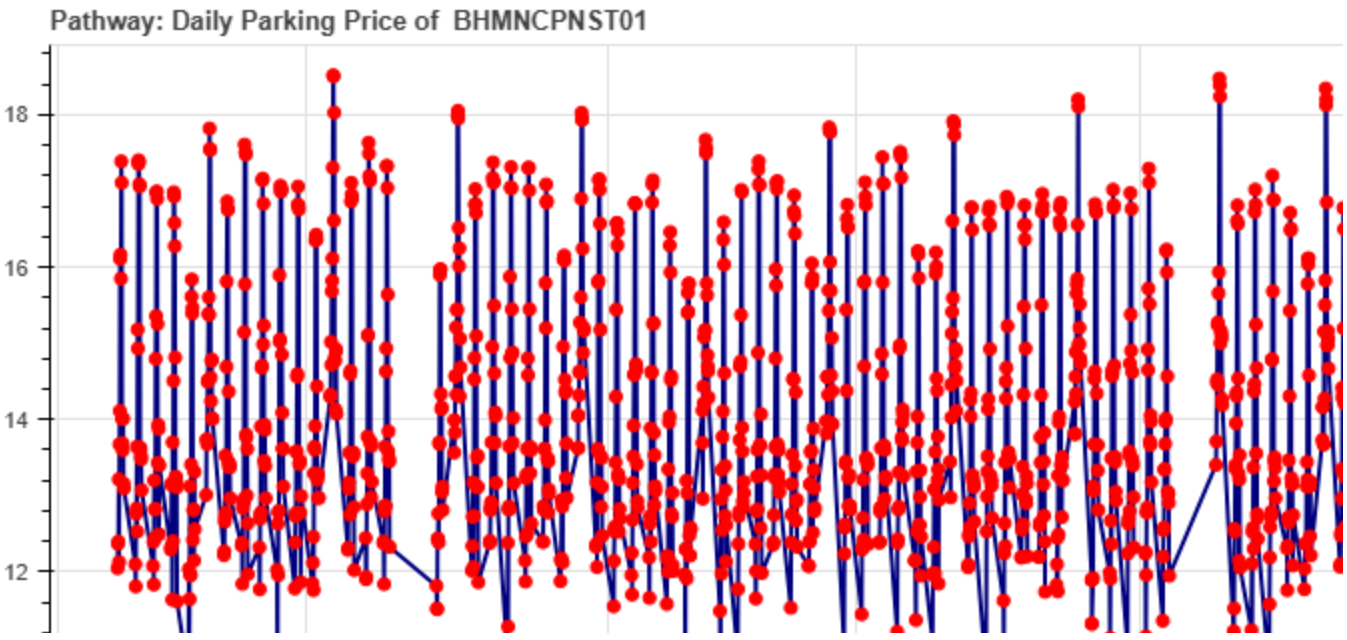
Lot: BHMNCPHST01

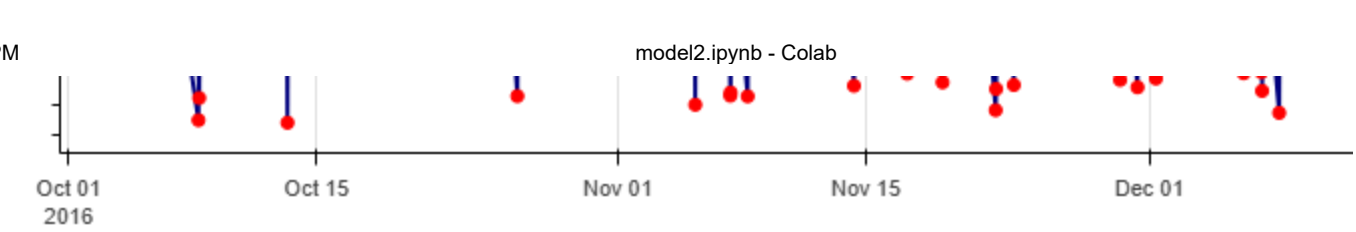
Streaming mode



Lot: BHMNCPNST01

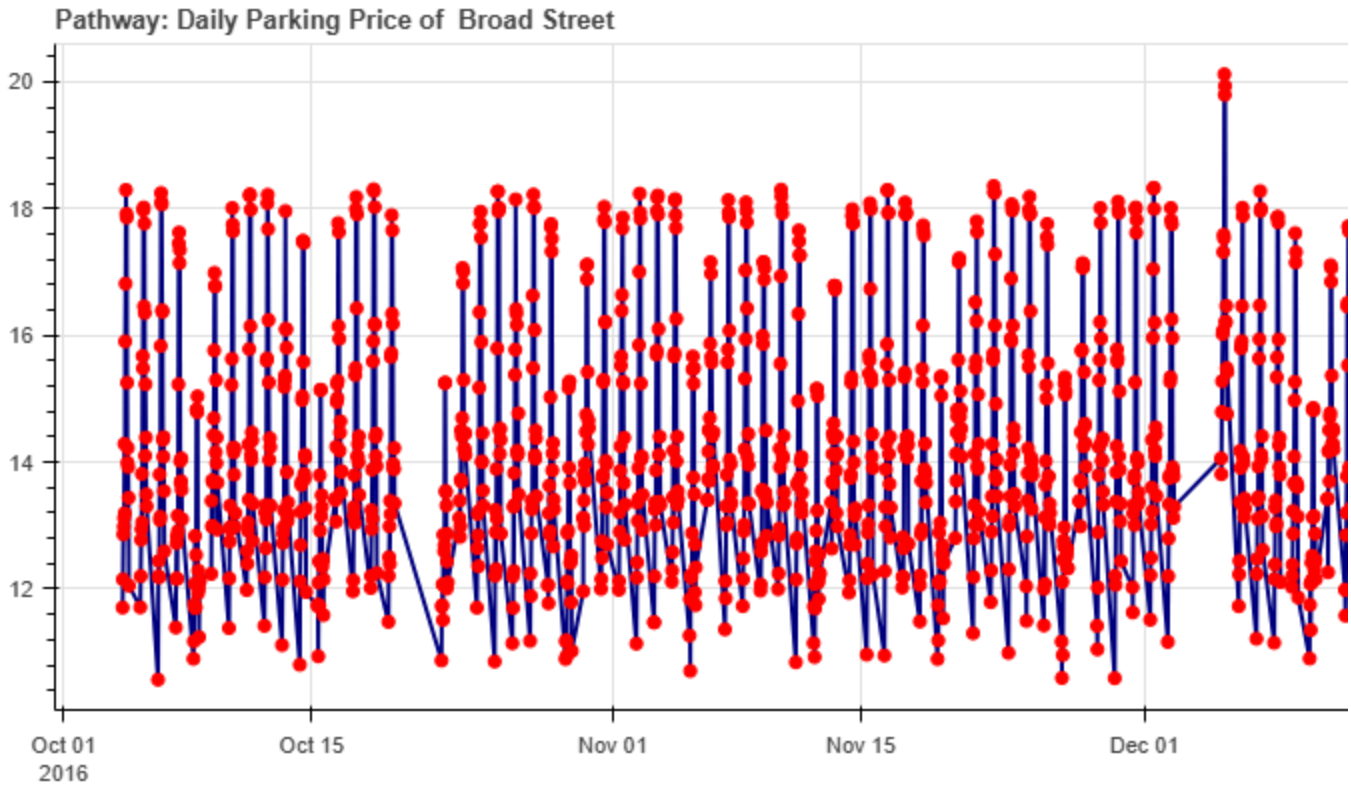
Streaming mode





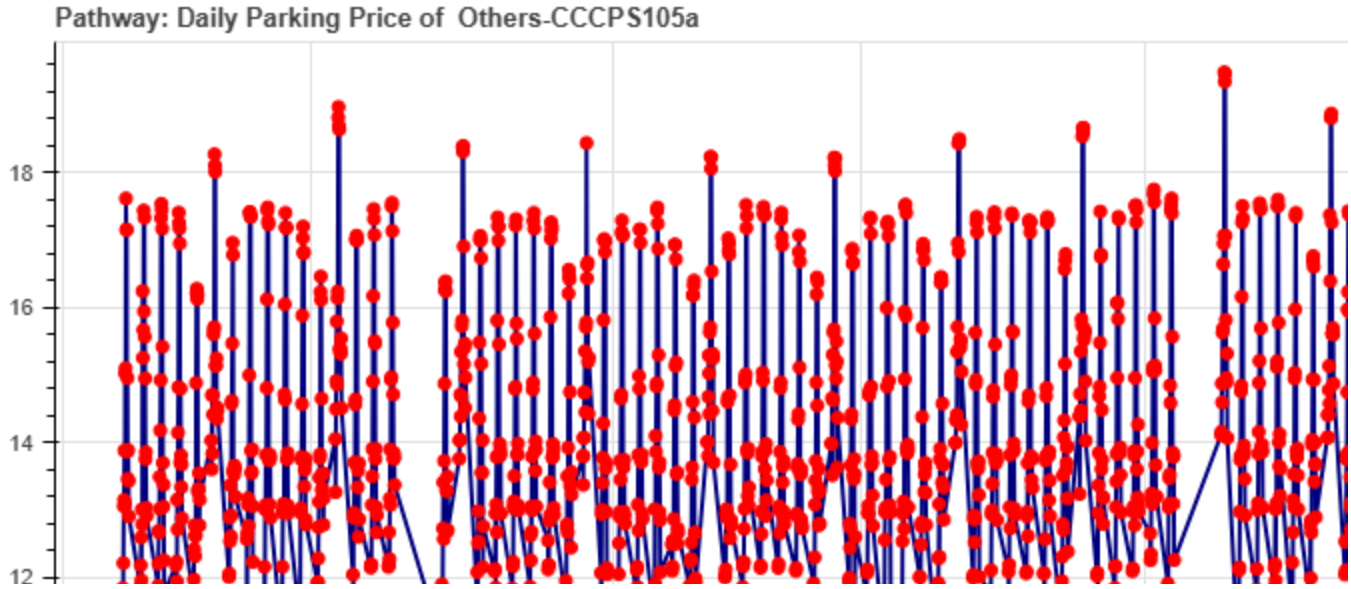
Lot: Broad Street

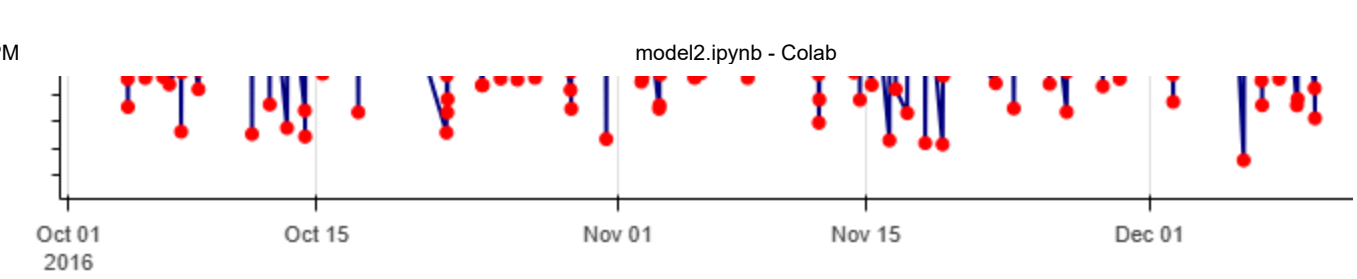
Streaming mode



Lot: Others-CCCPS105a

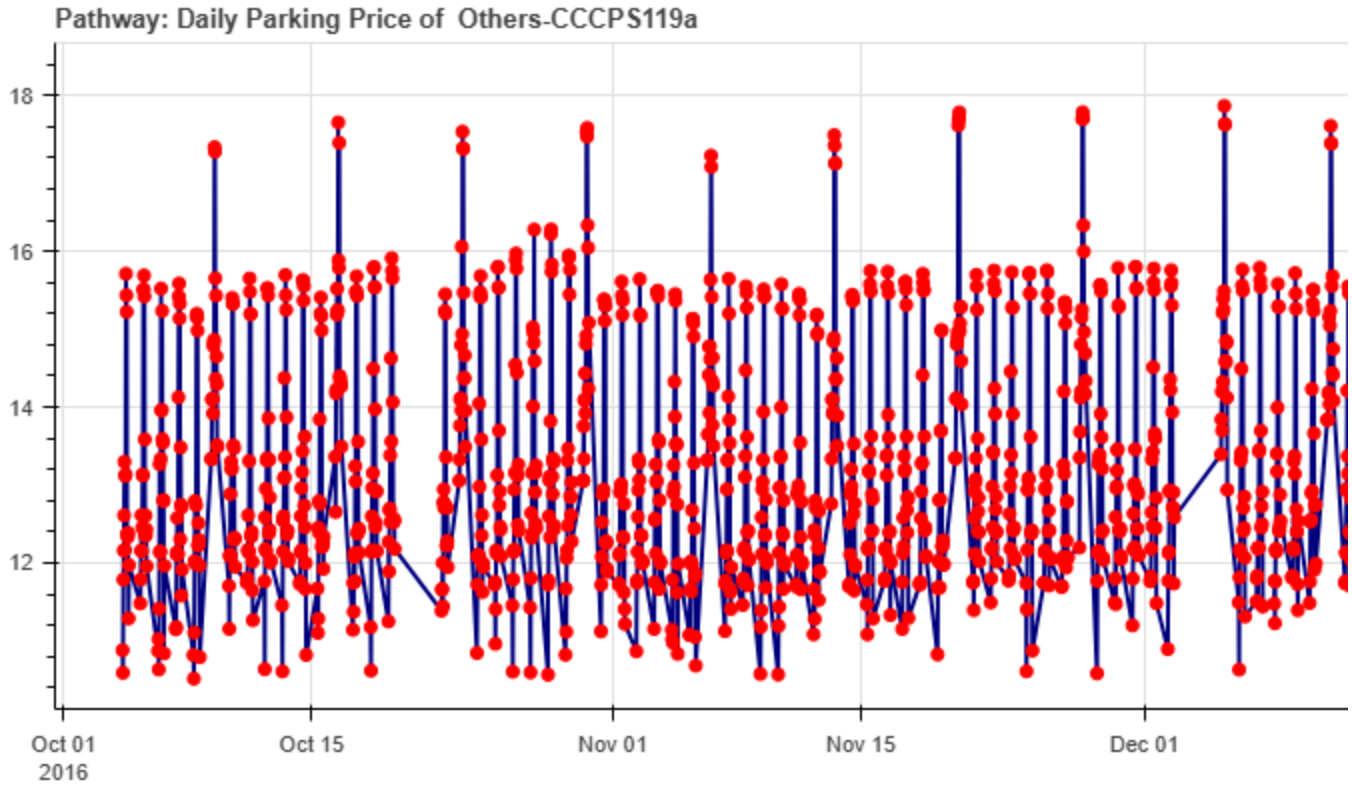
Streaming mode





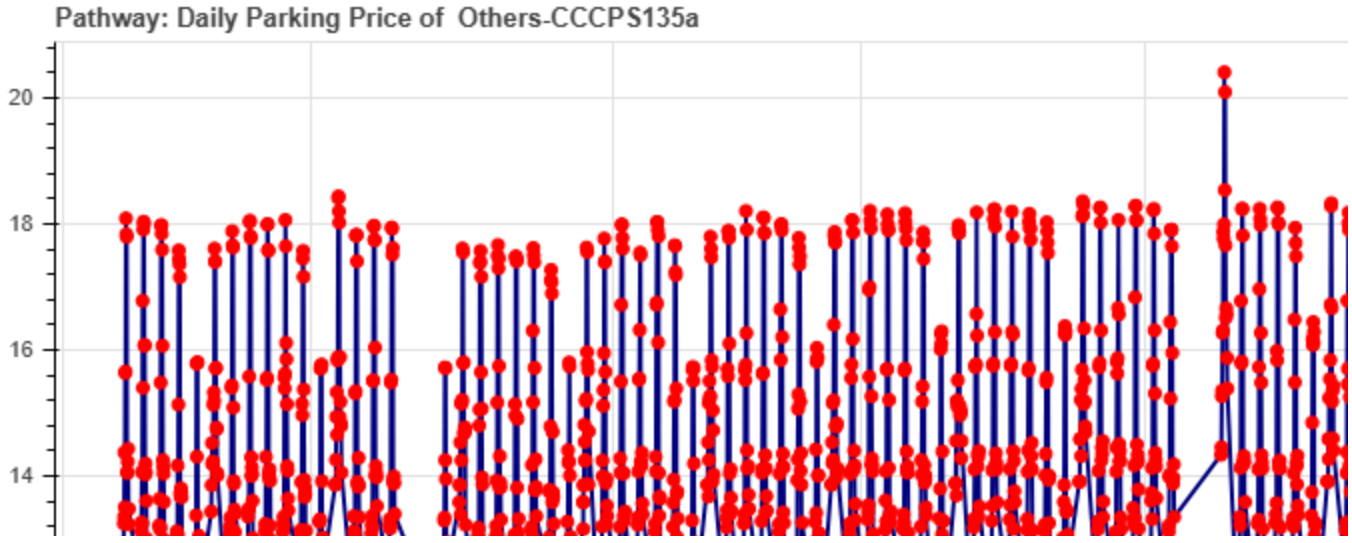
Lot: Others-CCCPS119a

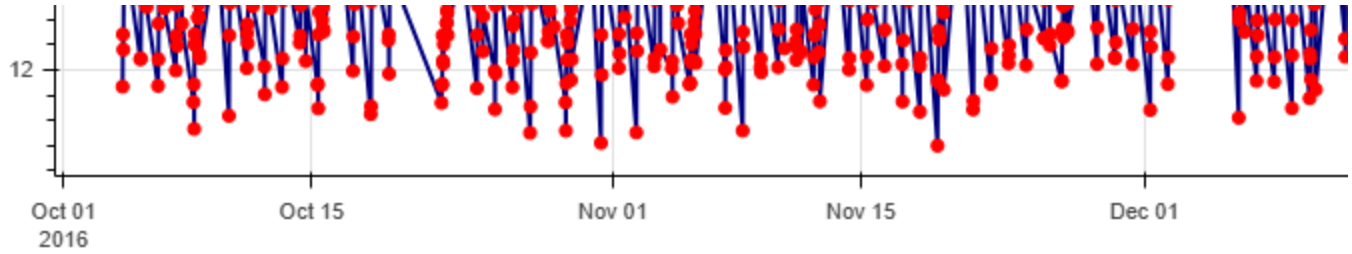
Streaming mode



Lot: Others-CCCPS135a

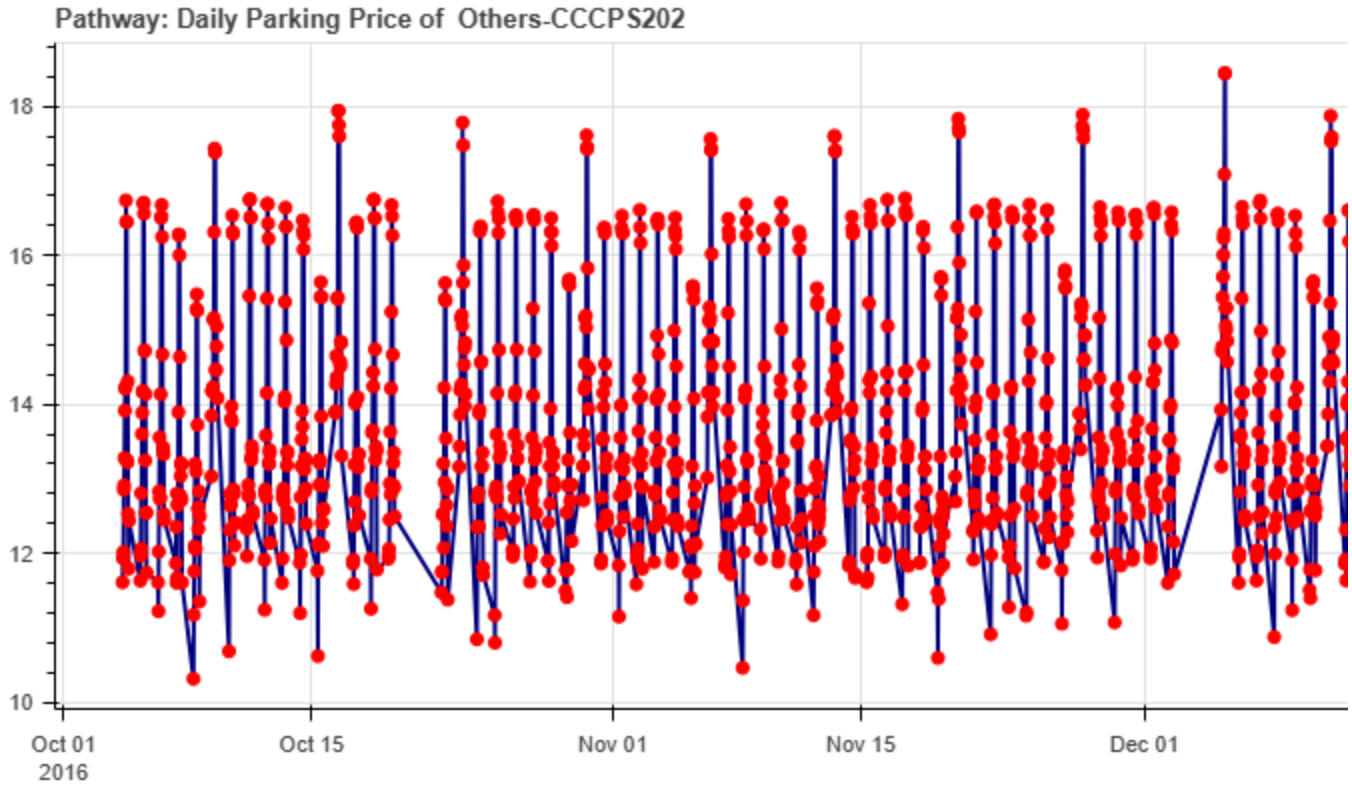
Streaming mode





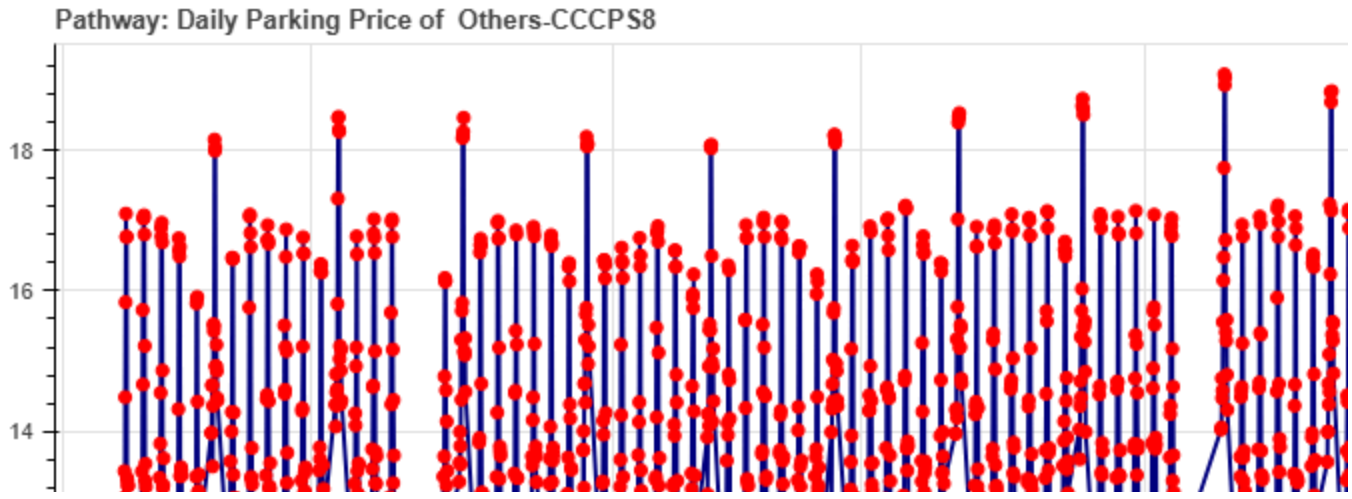
Lot: Others-CCCPS202

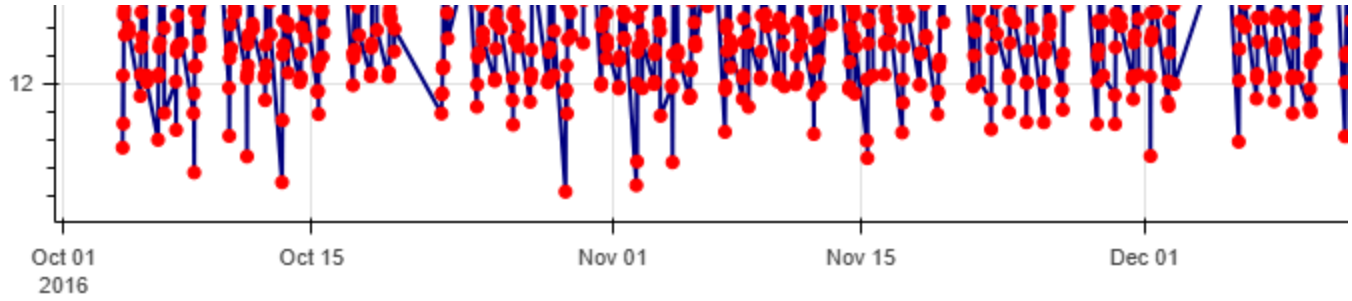
Streaming mode



Lot: Others-CCCPS8

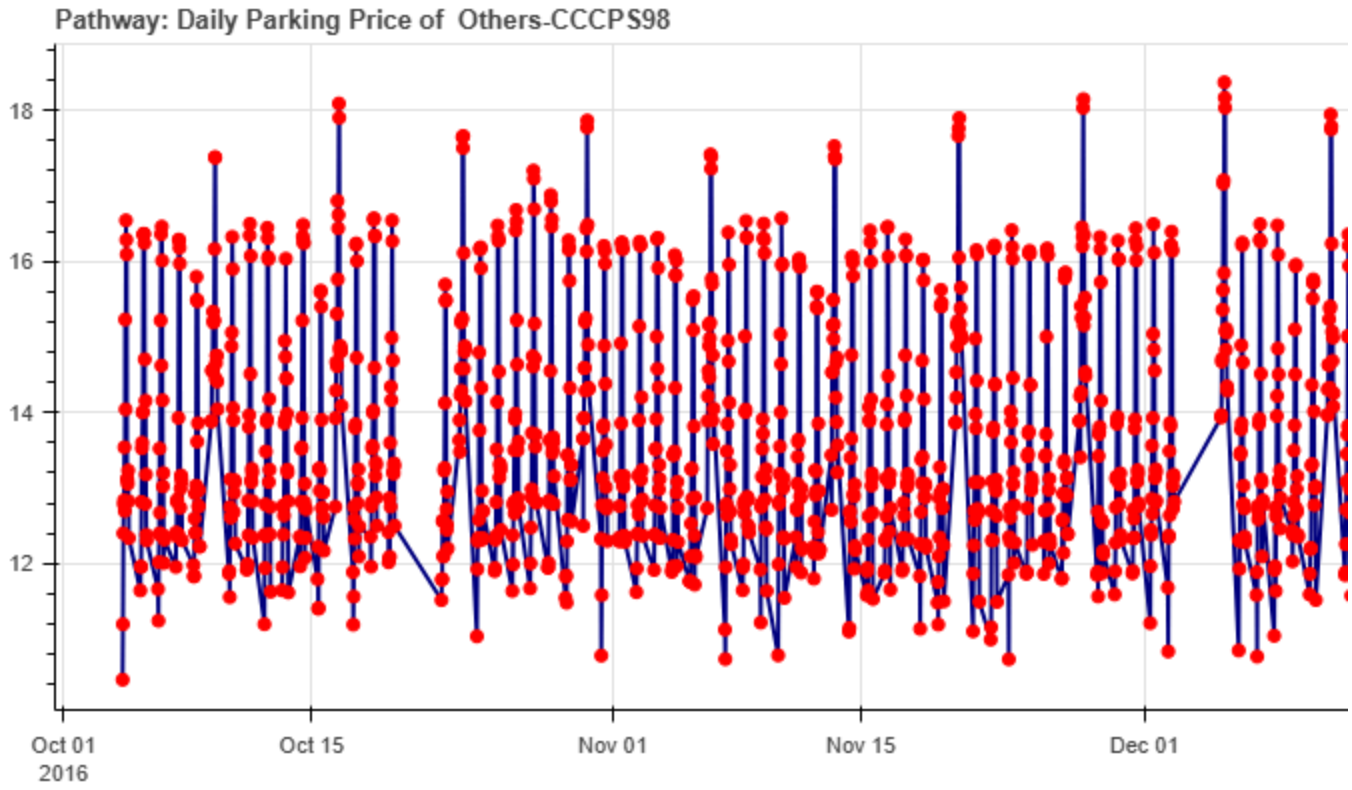
Streaming mode





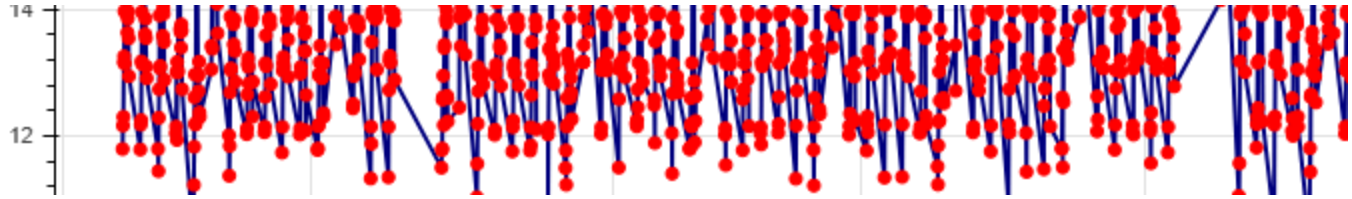
Lot: Others-CCCPS98

Streaming mode



Lot: Shopping

Streaming mode


14
12