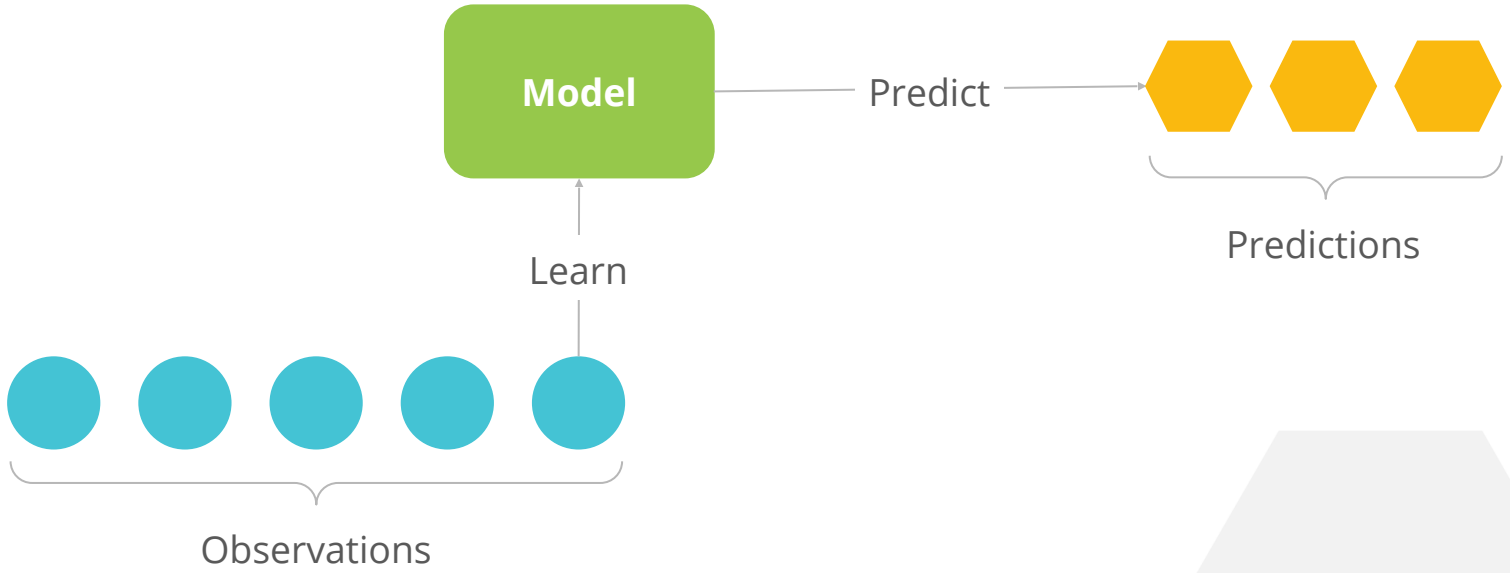
bytewax

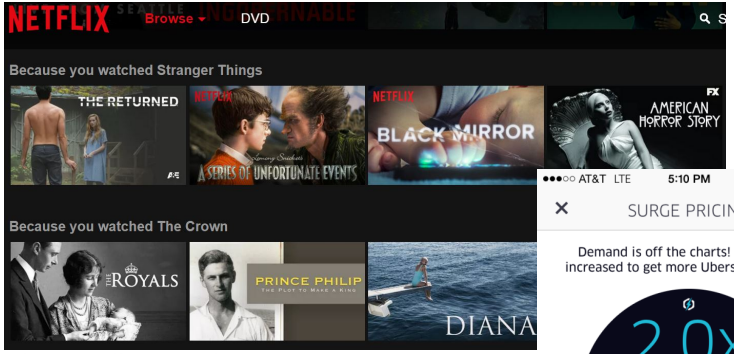
Online Machine Learning

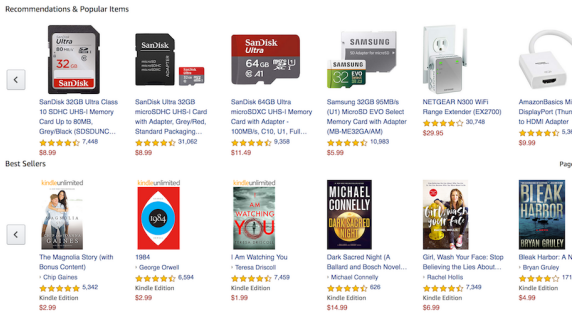
Bytewax + RiverML

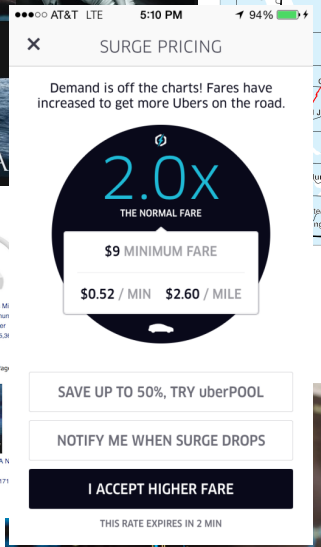
Online Machine Learning

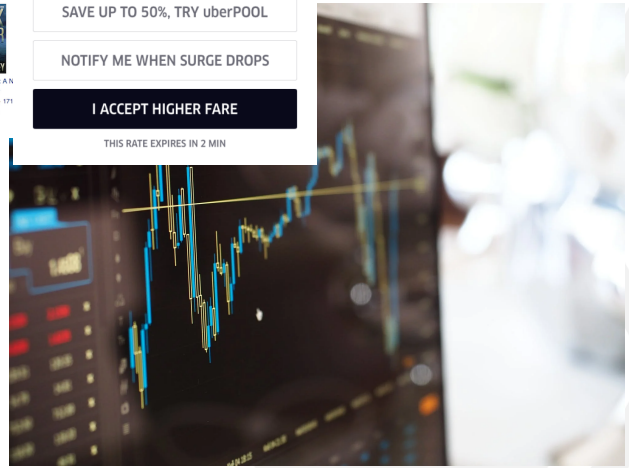


- Content Personalization
- Recommendations
- Nowcasting
- Fraud Detection
- Algorithmic Trading
- Manufacturing Defects
- Dynamic Pricing



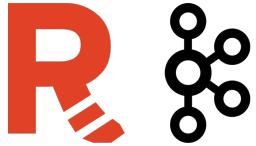






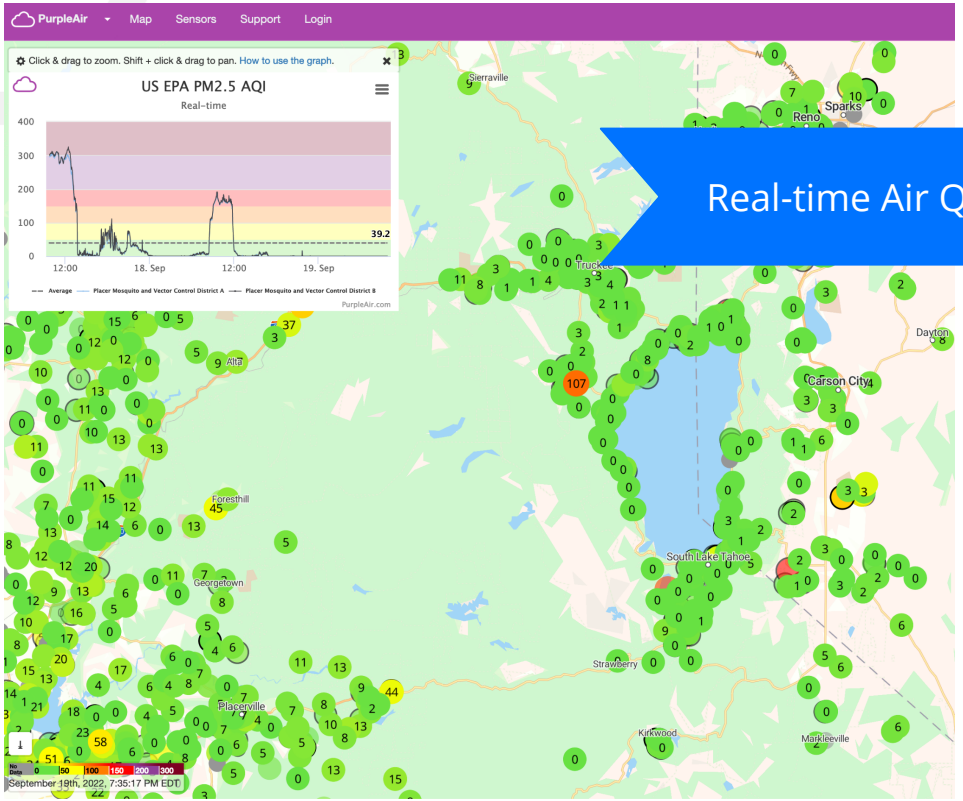
River is a Python library with algorithms and tools for online learning.

Bytewax is a Python library for stateful stream processing in conjunction with streaming platforms



Kafka compatible streaming platform.

Detecting Fires in Real-time with Air Quality Data

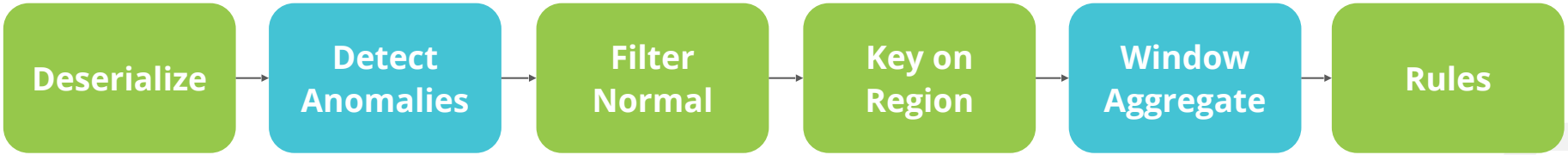


Real-time Air Quality Data

Bytewax Dataflow

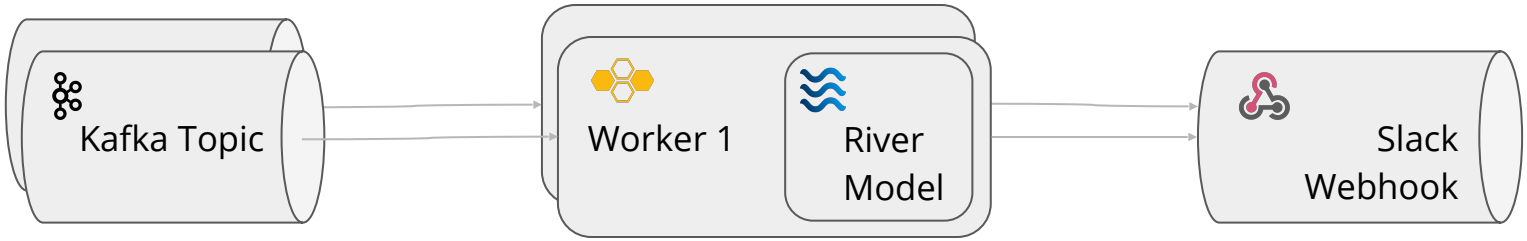
Fires/Anomalies

Dataflow Steps



Stateless Operators

Stateful Operators


Kafka Topic
Worker 1
River
Model
Slack
Webhook

Bytewax Input

```python
flow = Dataflow()
flow.input(
    "aqi_state",
    KafkaInputConfig(
        brokers=["localhost:9092"],
        topic="sensor_data",
        starting_offset = "beginning",
        tail = False
        )
    )

def deserialize(key_bytes__payload_bytes):
    key_bytes, payload_bytes = key_bytes__payload_bytes
    key = json.loads(key_bytes) if key_bytes else None
    sensor_data = json.loads(payload_bytes) if payload_bytes else None
    return key, sensor_data

flow.map(deserialize)
```

Stateful Anomaly Detector w/ River

```
class AnomalyDetector:

    def __init__(self, n_trees=10, height=8, window_size=72, seed=11):
        self.detector = anomaly.HalfSpaceTrees(
                                               n_trees=n_trees,
                                               height=height,
                                               window_size=window_size,
                                               # we are using 1200 as the max for this
                                               # dataset since we know in advanced it
                                               # is the highest
                                               limits={'x': (0.0, 1200)},
                                               seed=seed
                                               )

    def update(self, data):
        data['score'] = self.detector.score_one({'x': float(data['PM2.5_CF1_ug/m3'])})
        self.detector.learn_one({'x': float(data['PM2.5_CF1_ug/m3'])})
        return self, data

flow.stateful_map(
    step_id = "anomaly_detector",
    builder = lambda: AnomalyDetector(n_trees=4, height=3, window_size=50, seed=11),
    mapper = AnomalyDetector.update,
)
flow.filter(lambda x: x[1]['score']>0.6)
flow.filter(lambda x: float(x[1]['PM2.5_CF1_ug/m3'])>50)
```

Windowing w/ Bytewax

```
# We need to specify a wait time that is as long as the difference between
# the oldest (2022-07-01) and the newest (2022-09-18) to ensure out of order
# events are handled correctly
cc = EventClockConfig(get_event_time, wait_for_system_duration=timedelta(days=100))

# Manually set the start time for this dataflow, this is known for this dataset
start_at = datetime.strptime("2022-07-01 00:00:00 UTC", "%Y-%m-%d %H:%M:%S
%Z").replace(tzinfo=timezone.utc)
wc = TumblingWindowConfig(start_at=start_at, length=timedelta(hours=6))

class Anomalies:

    def __init__(self):
        self.sensors = []
        self.times = []
        self.values = []

    def update(self, event):
        self.sensors.append(event["coordinates"])
        self.times.append(event["created_at"])
        self.values.append(float(event["PM2.5_CF1_ug/m3"]))

        return self

flow.fold_window("count_sensors", cc, wc, Anomalies, Anomalies.update)
```

Output to Slack

```python
# Calculate some statistics and use rules to separate smoke events from malfunctions
def convert(key__anomalies):
    # removed for brevity - see github.com/awmatheson/current22
    return (key, {sensor_data})

flow.map(convert)

def output_builder(worker_index, worker_count):

    def send_to_slack(key__sensor_data):
        location, sensor_data = key__sensor_data
        if sensor_data['malfunction']:
            message = f'''In {location} was a malfunctioning sensor at location
                          {sensor_data['sensors']} at {sensor_data['min_event']}'''
            title = (f"Malfunctioning Sensor")
        else:
            message = f'''In {location} there is a suspected smoke event from a fire reported
                        by sensors at {sensor_data['sensors']} at {sensor_data['min_event']}'''
            title = (f"Suspected Smoke Event")
        slack_data = {
            # removed for brevity - see github.com/awmatheson/current22
        }
        byte_length = str(sys.getsizeof(slack_data))
        headers = {'Content-Type': "application/json", 'Content-Length': byte_length}
        response = requests.post(WEBHOOK_URL, data=json.dumps(slack_data), headers=headers)
        if response.status_code != 200:
            raise Exception(response.status_code, response.text)

    return send_to_slack

flow.capture(
    ManualOutputConfig(output_builder))
```

```python
from scipy.stats import variation

from river import anomaly

WEBHOOK_URL = os.getenv("WEBHOOK_URL")

# Define Dataflow and input configuration
flow = Dataflow()
flow.input(
    "aqi_state",
    KafkaInputConfig(
        brokers=["localhost:9092"],
        topic="sensor_data",
        starting_offset = "beginning",
        tail = False
        )
    )

# Deserialize input for processing
def deserialize(key_bytes__payload_bytes):
    key_bytes, payload_bytes = key_bytes__payload_bytes
    key = json.loads(key_bytes) if key_bytes else None
    sensor_data = json.loads(payload_bytes) if payload_bytes else None
```

PROBLEMS 25 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

zsh - current22

(current22) awmatheson@Alexanders-MacBook-Pro-2 current22 %

bytewax

The End

@MathesonZander

@awmatheson

Repo github.com/awmatheson/current22