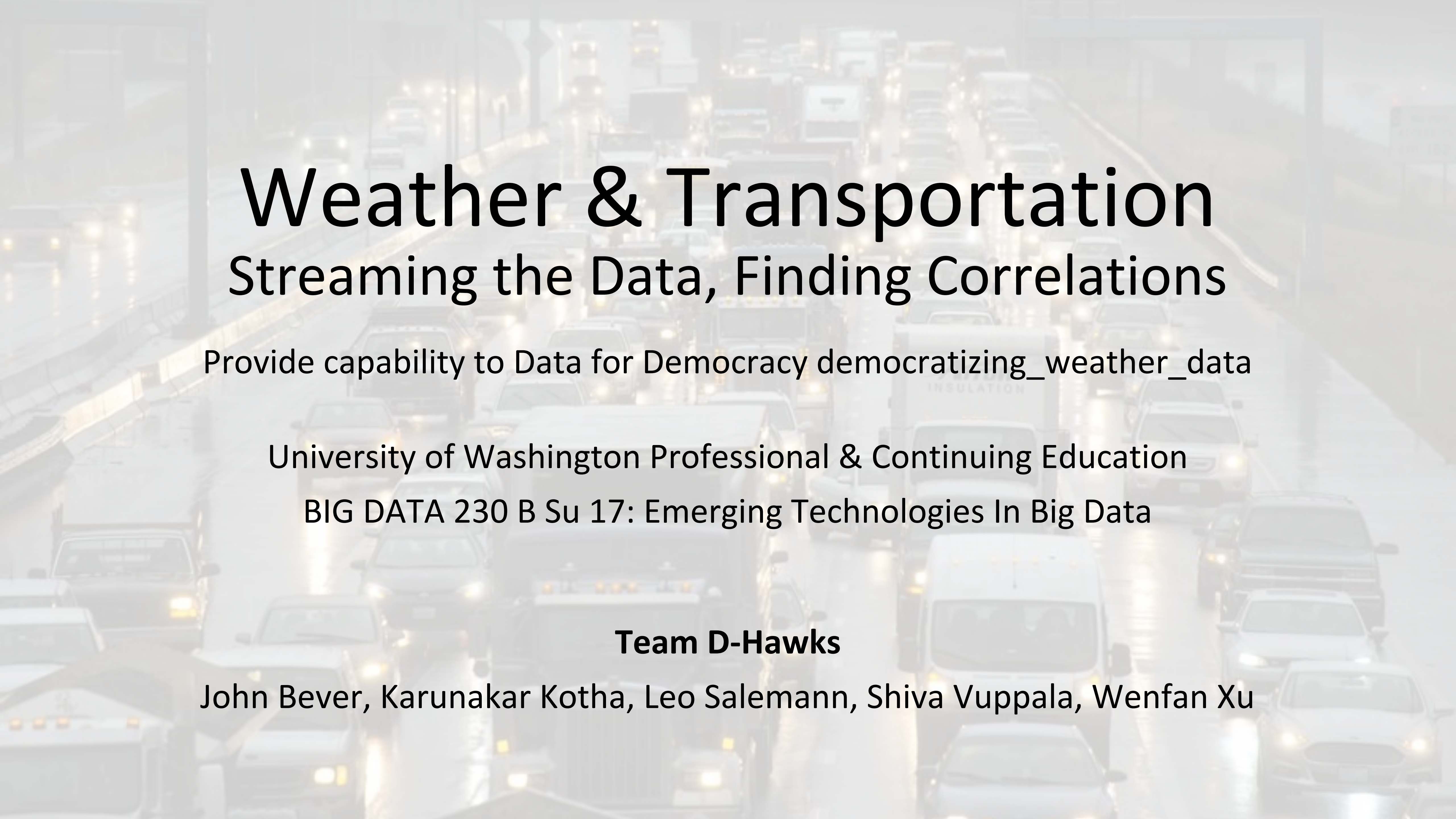


Weather & Transportation

Streaming the Data, Finding Correlations

Provide capability to Data for Democracy democratizing_weather_data

University of Washington Professional & Continuing Education

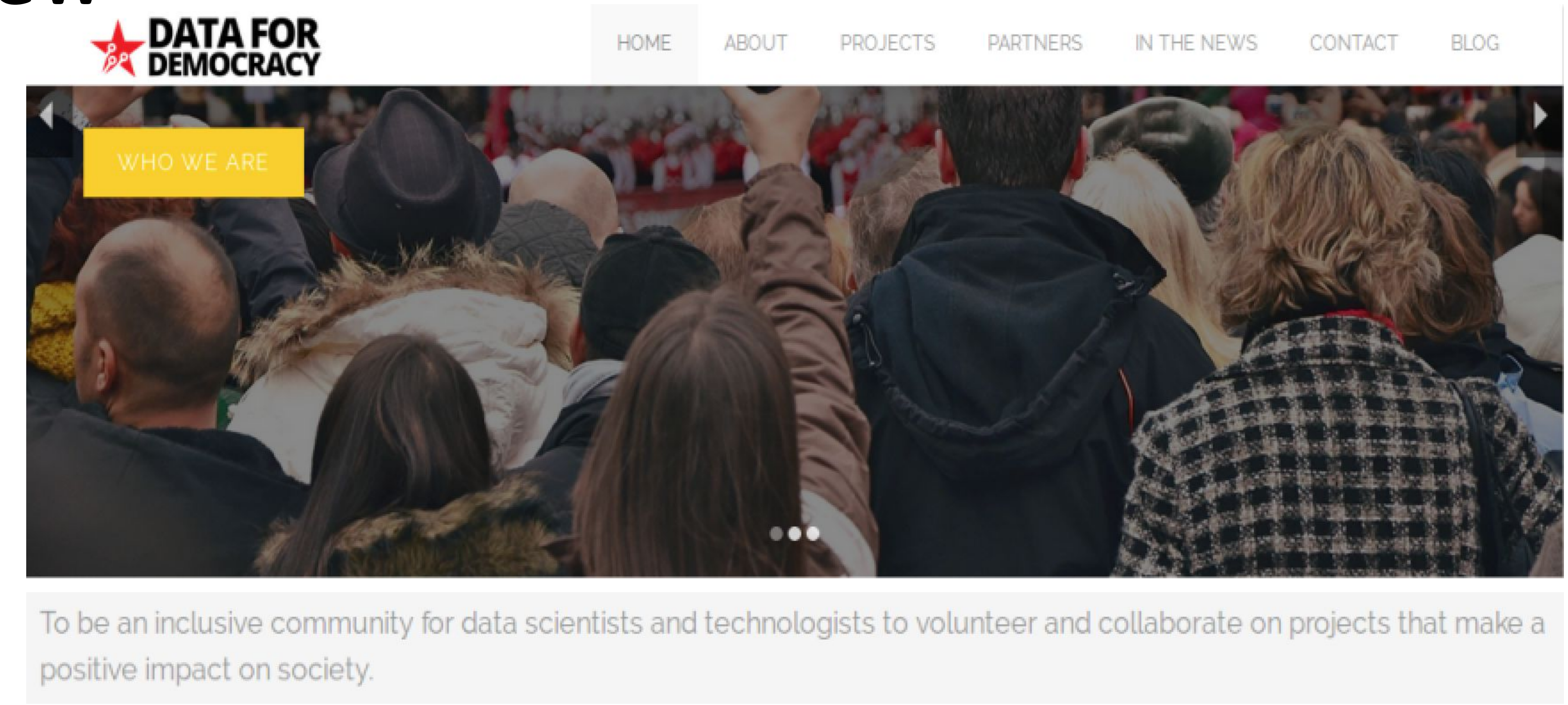
BIG DATA 230 B Su 17: Emerging Technologies In Big Data

Team D-Hawks

John Bever, Karunakar Kotha, Leo Salemann, Shiva Vuppala, Wenfan Xu

Overview

Our "Client"



Their Mission

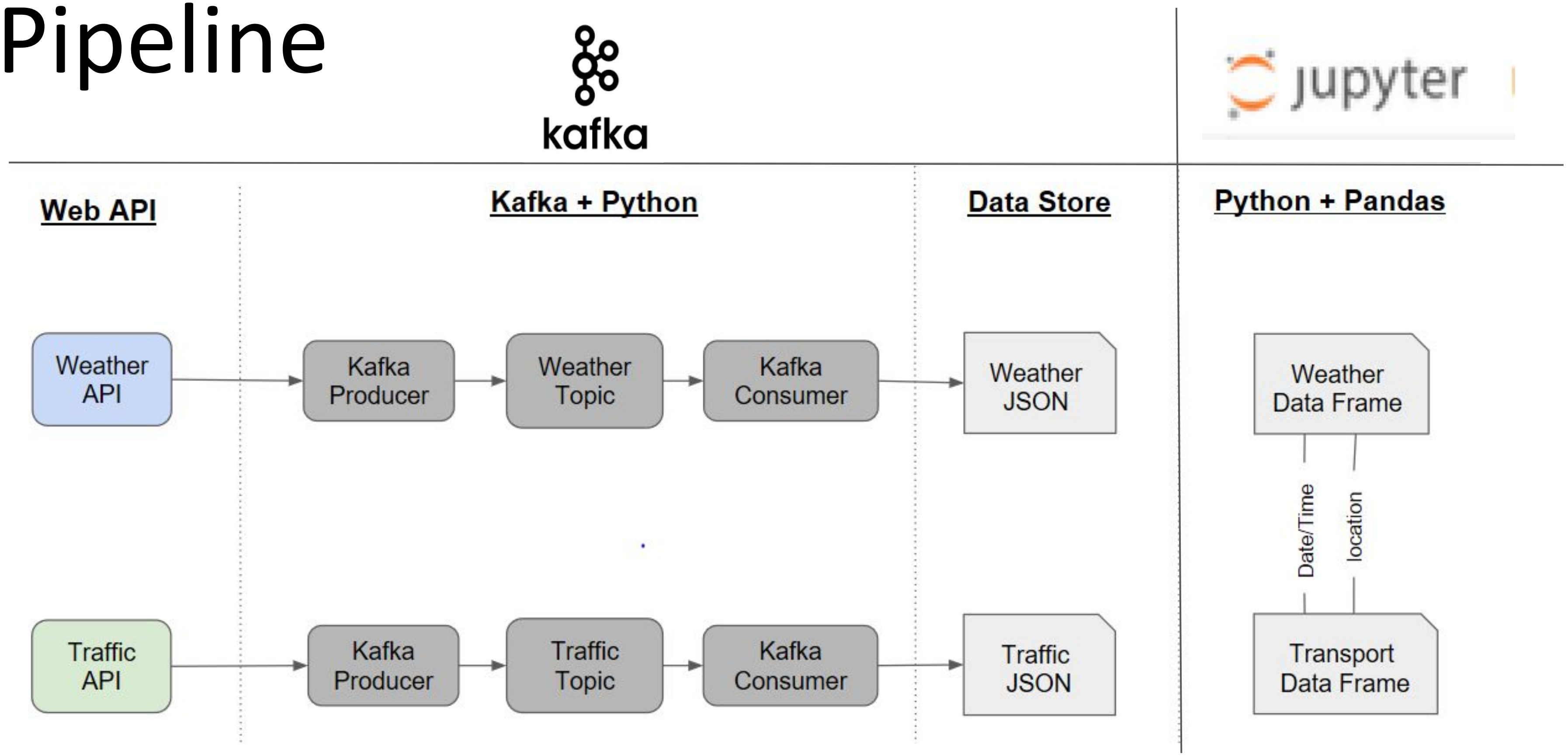
To be an inclusive community for data scientists and technologists to volunteer and collaborate on projects that make a positive impact on society.

Our Mission

- Provide a streaming capability to extract weather and traffic data from multiple Web API's, and produce a clean merged dataframe suitable for Machine Learning and other Data Science analysis.
- Deliver code to D4D's Github Repository
- Use vendor-neutral, opensource solutions, implemented in python and Jupyter notebooks

Learn More www.datafordemocracy.org https://github.com/Data4Democracy democratizing_weather_data/streaming

Pipeline

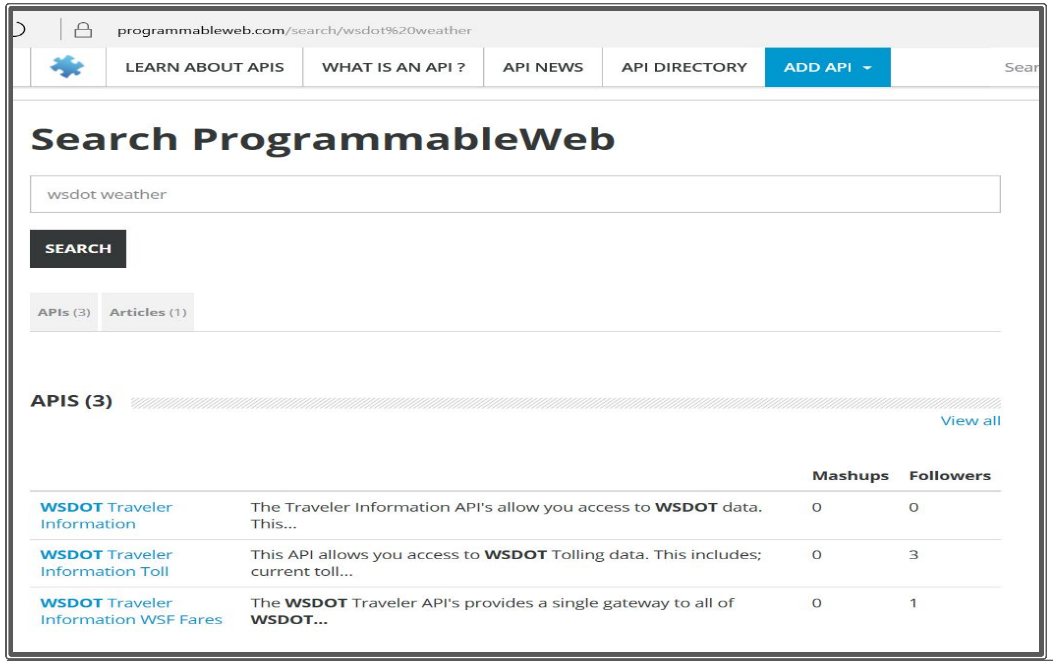


- Kafka transport mechanism (vendor-neutral, open source)
- Message value is an entire JSON document
- One topic per source API, guarantees consistent schema
- Multiple json documents (sharing same schema) combined into a single dataframe
- Dataframe records joined based on space and time

Web APIs

ProgrammableWeb.com

- A massive searchable directory of over 15,500 web APIs that are updated daily
- Includes sample source code for APIs



Search ProgrammableWeb

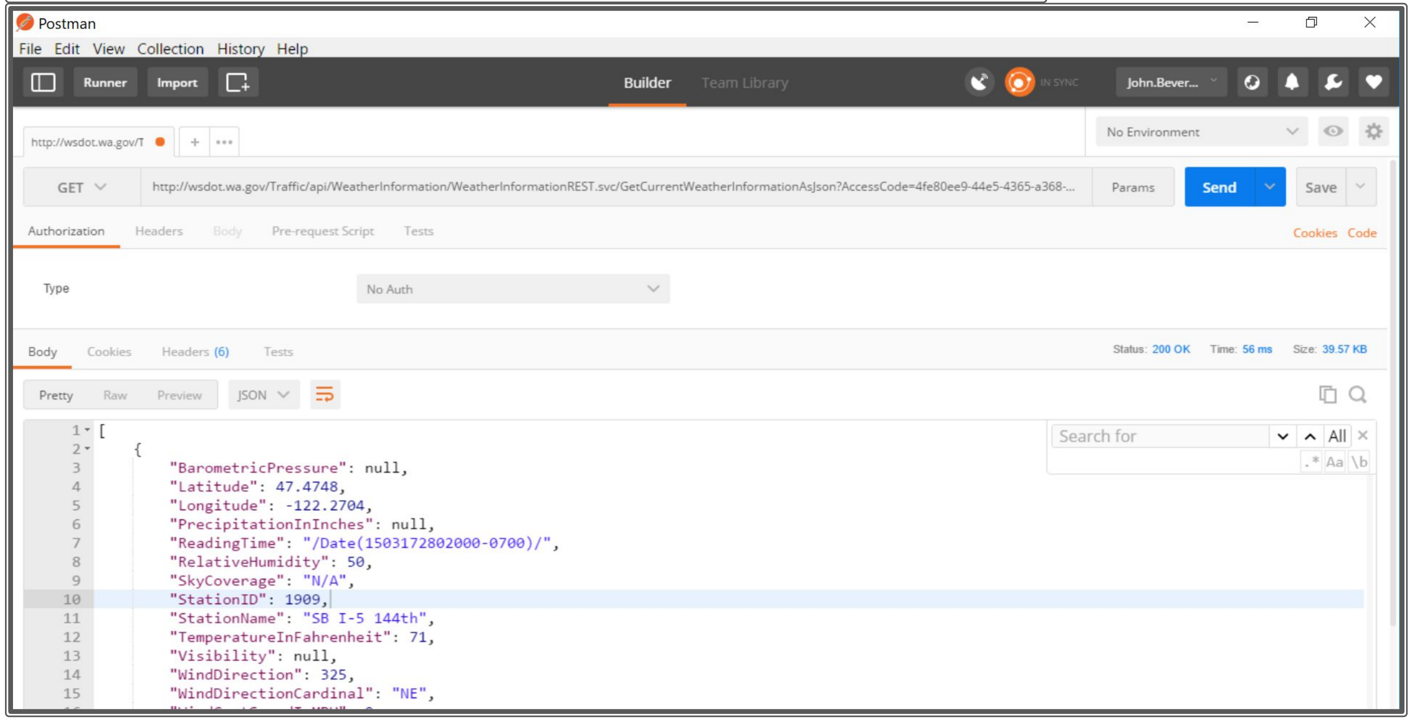
wsdot weather

APIS (3)

		Mashups	Followers
WSDOT Traveler Information	The Traveler Information API's allow you access to **WSDOT** data. This...	0	0
WSDOT Traveler Information Toll	This API allows you access to **WSDOT** Tolling data. This includes; current toll...	0	3
WSDOT Traveler Information WSF Fares	The **WSDOT** Traveler API's provides a single gateway to all of **WSDOT...**	0	1

Postman

- Great tool for interacting with potential APIs.
- Friendly GUI for constructing requests and reading responses.
- **Provided JSON files before pipeline was completed. Allowed analysis of data in parallel**



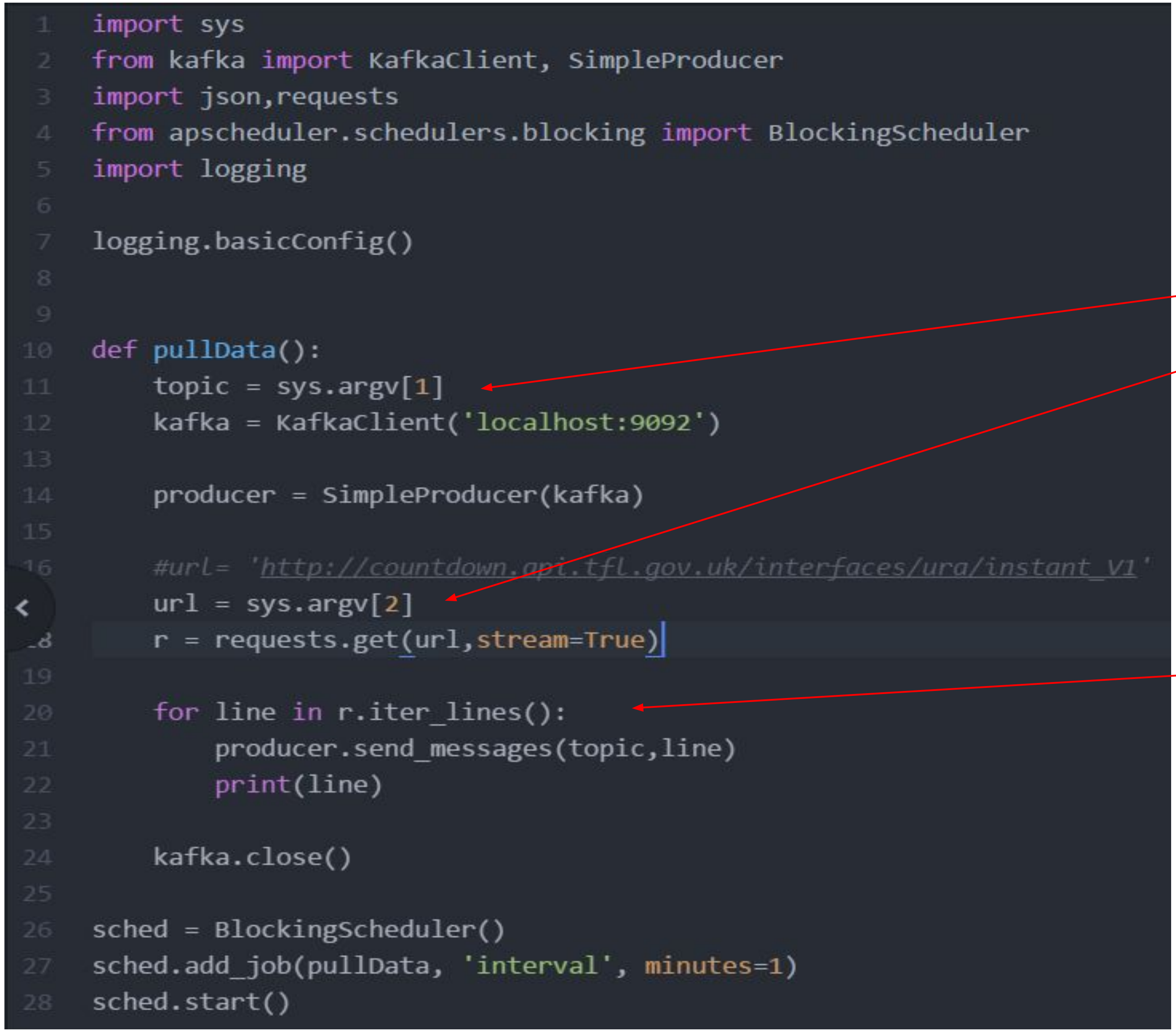
GET http://wsdot.wa.gov/Traffic/api/WeatherInformation/WeatherInformationREST.svc/GetCurrentWeatherInformationAsJson?AccessCode=4fe80ee9-44e5-4365-a368-...

Type: No Auth

Status: 200 OK Time: 56 ms Size: 39.57 KB

```
[
    {
        "BarometricPressure": null,
        "Latitude": 47.4748,
        "Longitude": -122.2704,
        "PrecipitationInInches": null,
        "ReadingTime": "/Date(1503172802000-0700)/",
        "RelativeHumidity": 50,
        "SkyCoverage": "N/A",
        "StationID": 1909,
        "StationName": "SB I-5 144th",
        "TemperatureInFahrenheit": 71,
        "Visibility": null,
        "WindDirection": 325,
        "WindDirectionCardinal": "NE",
```

Producers



```python
import sys
from kafka import KafkaClient, SimpleProducer
import json,requests
from apscheduler.schedulers.blocking import BlockingScheduler
import logging

logging.basicConfig()


def pullData():
    topic = sys.argv[1]
    kafka = KafkaClient('localhost:9092')

    producer = SimpleProducer(kafka)

    #url= 'http://countdown.api.tfl.gov.uk/interfaces/ura/instant_V1'
    url = sys.argv[2]
    r = requests.get(url,stream=True)

    for line in r.iter_lines():
        producer.send_messages(topic,line)
        print(line)

    kafka.close()

sched = BlockingScheduler()
sched.add_job(pullData, 'interval', minutes=1)
sched.start()
```

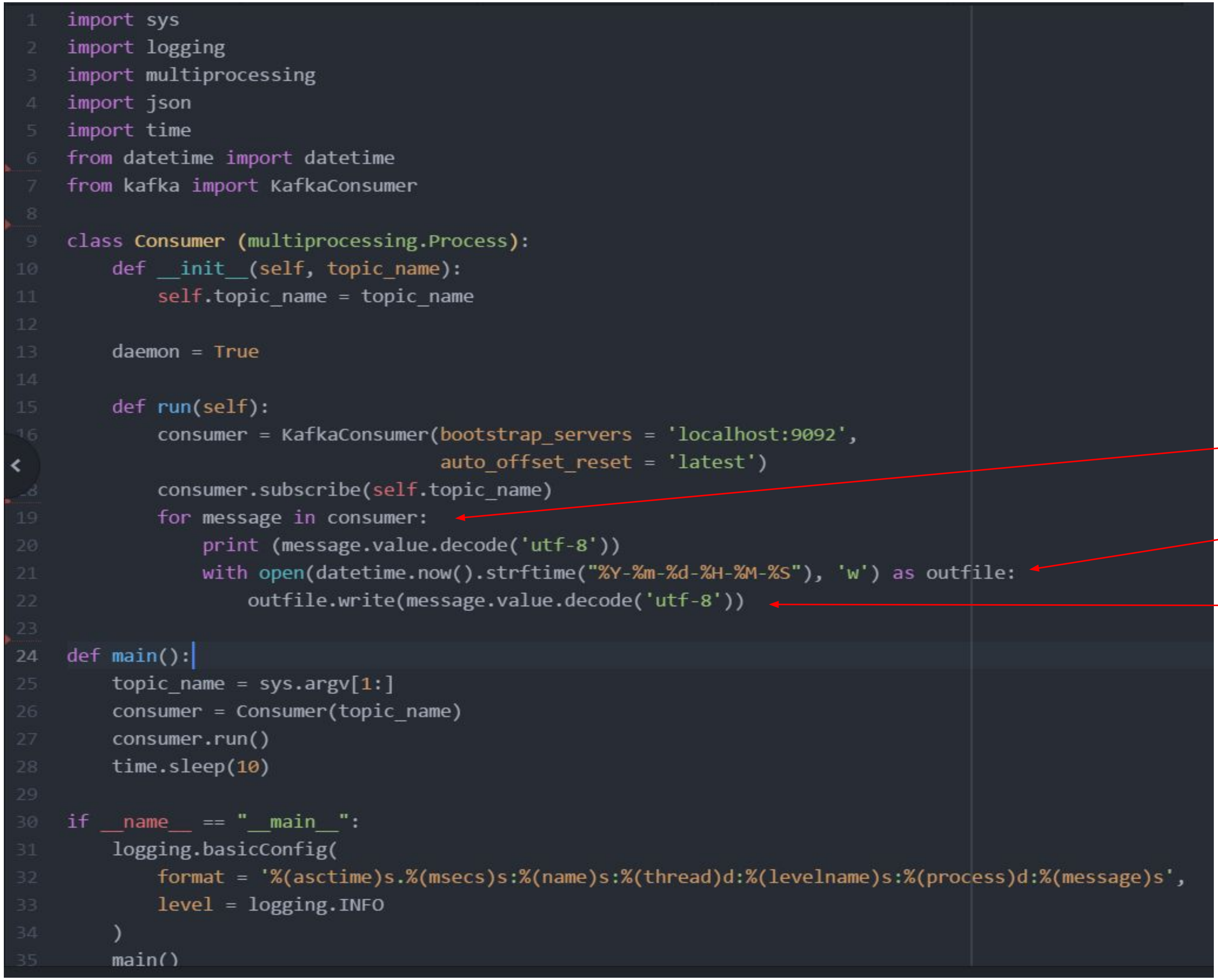
Arguments

- Topic
- URL + Access Key

Message.Value

- JSON document

Consumers



```
import sys
import logging
import multiprocessing
import json
import time
from datetime import datetime
from kafka import KafkaConsumer

class Consumer (multiprocessing.Process):
    def __init__(self, topic_name):
        self.topic_name = topic_name

    daemon = True

    def run(self):
        consumer = KafkaConsumer(bootstrap_servers = 'localhost:9092',
                                 auto_offset_reset = 'latest')
        consumer.subscribe(self.topic_name)
        for message in consumer:
            print (message.value.decode('utf-8'))
            with open(datetime.now().strftime("%Y-%m-%d-%H-%M-%S"), 'w') as outfile:
                outfile.write(message.value.decode('utf-8'))

def main():
    topic_name = sys.argv[1:]
    consumer = Consumer(topic_name)
    consumer.run()
    time.sleep(10)

if __name__ == "__main__":
    logging.basicConfig(
        format = '%(asctime)s.%(msecs)s:%(name)s:%(thread)d:%(levelname)s:%(process)d:%(message)s',
        level = logging.INFO
    )
    main()
```

- One complete JSON file on disk per message
- Filename includes timestamp
- “utf-8” decoded text file

Analysis

7 days of data (includes eclipse!)   30 minutes between readings

Load Json file, normalize, save as dataframe.
Repeat for next json file, append to prior.

54 Weather Json Files from Yahoo (54 rows x 31 columns)

394 Weather Json Files from WSDOT (40,931 rows x 16 columns)

395 Traffic Json Files from WSDOT (70,998 rows x 20 columns)

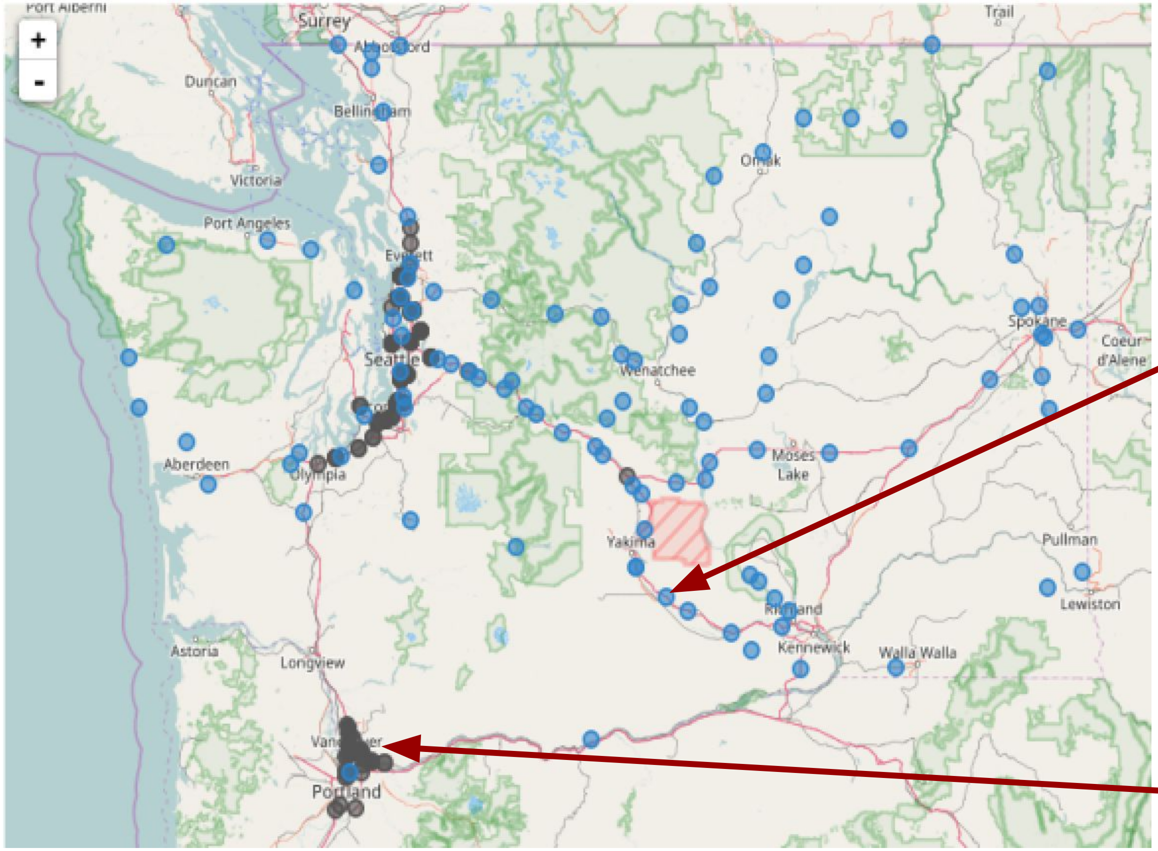
Merge WSDOT & Yahoo   Weather Dataframes (use columns common to both)
Merge Traffic/Weather Dataframes. Each Row has:

- Traffic data from a specific Traffic dataframe row
- Weather data from a weather station within *20 miles* and *30 minutes* of traffic reading.

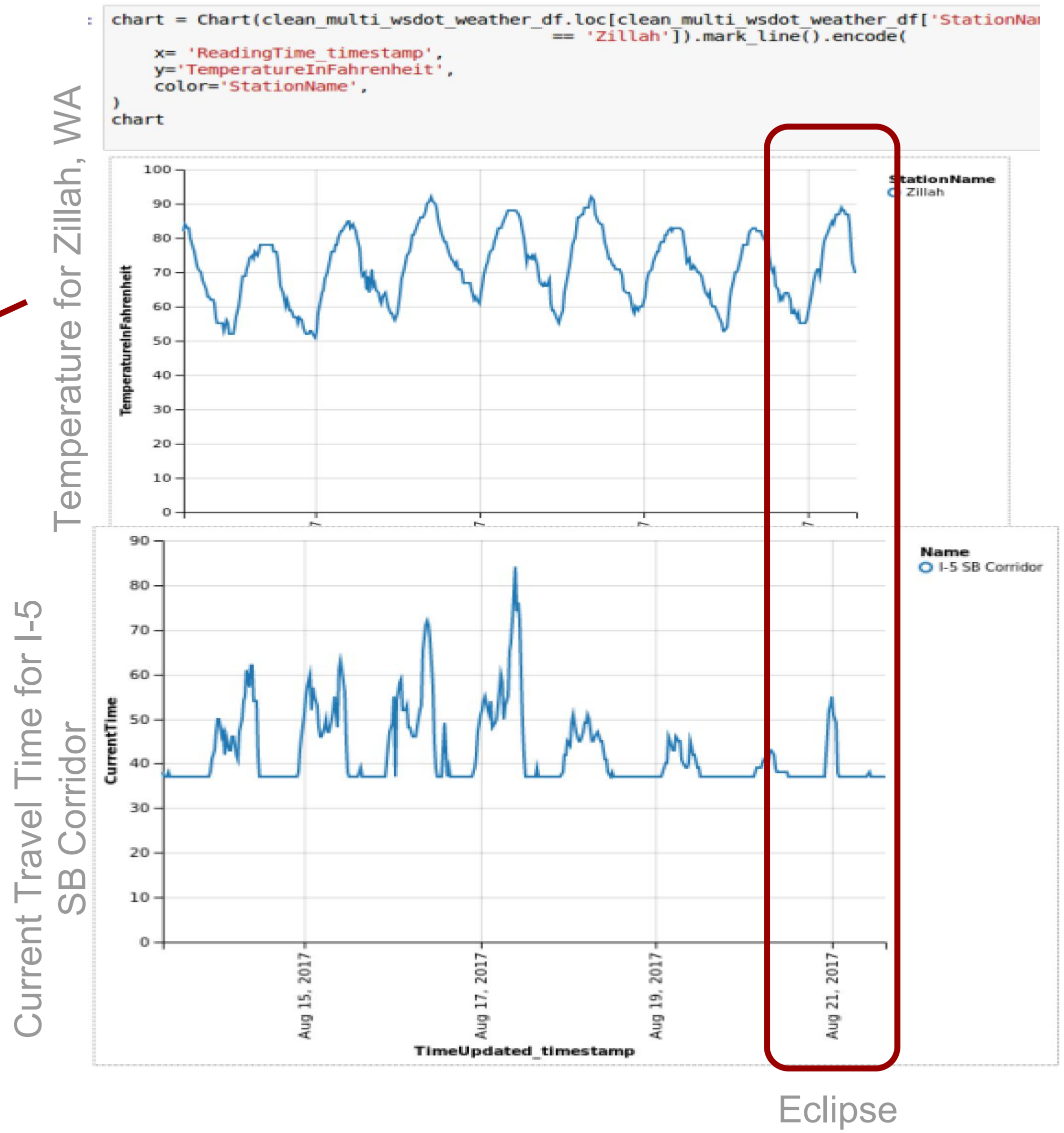
1 Merged Traffic/Weather Table (52,975 rows x 30 columns)

Visualization

Mapping with Folium (traffic in black; weather in blue)



Charting with Altair



```
chart = Chart(clean_multi_wsdot_weather_df.loc[clean_multi_wsdot_weather_df['StationNa
                                              == 'Zillah']).mark_line().encode(
    x= 'ReadingTime_timestamp',
    y='TemperatureInFahrenheit',
    color='StationName',
)
chart
```

Temperature for Zillah, WA

Current Travel Time for I-5 SB Corridor

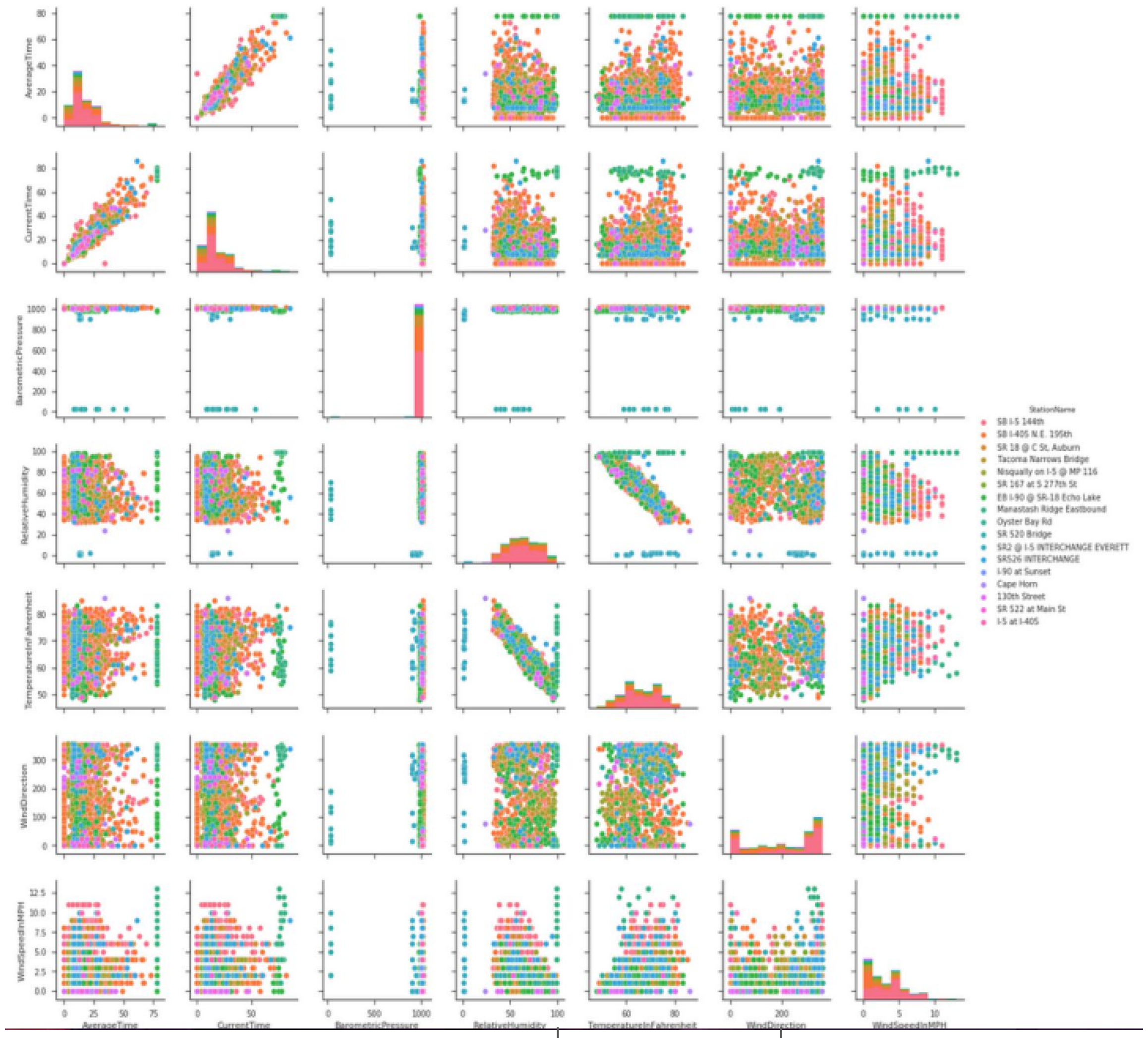
Eclipse

Analyzing the Merged/Traffic Weather Dataset

Scatterplot Matrix with Seaborn (10% random sample)

Average Travel Time

Current Travel Time



Barometer

Temp.

Wind Speed

Humidity

Wind Direction

Wrapping Up ...

Key Takeaways

- Choose your python libraries carefully (2 lines of code for a fully-labeled lineplot vs. dozens)
- Spatial plots first, data-joins later (I-5 traffic data vs. statewide weather, also Portland)
- The fastest way to count records in a dataframe is df.shape[0]

Conclusion

- Data for Democracy has a repeatable way to extract weather and transportation data from WSDOT and Yahoo
- Jupyter Notebook provides a teaching/coding environment
- Bitnami provides low-cost simple Kafka infrastructure

Further Work

- Upload csv and zipped json's to data.world
- Better parameters for Producer scripts (ex. Longitude, Latitude, Date, Time)
- Config files for access keys
- More matrix plots, Data Science, Machine Learning
  - Gather data for longer time frames (fewer readings per day?)
  - Isolate matrix plots to specific locations and/or time.

THANK YOU!