pvta-bus-stuff

April 11, 2024

0.0.1 In this notebook, you will:

1. Get "live" data from all the visible PVTA buses
2. Store that data into a SQLite or Parquet file
3. Query that data using one of:
    1. Raw SQL
    2. A Python ORM called `Peewee`
    3. Panda Dataframe operations

0.1 Start with a request to the API:

[29]:
```
import requests
response = requests.get("https://bustracker.pvta.com/InfoPoint/rest/routes/
 ↪getvisibleroutes")
```

0.1.1 Now we extract the list of vehicles:

[46]:
```
routes = (rt for rt in response.json())
vehicles = [v for rt in routes for v in rt['Vehicles']]
```

0.1.2 It's useful to convert the `LastUpdated` field into this format:

[47]:
```
import re
import datetime
for v in vehicles:
    timestamp_ms = re.search(r'\d+', v['LastUpdated']).group()[:]
    v['LastUpdated_timestamp_ms'] = timestamp_ms
# Printing just one vehicle's data, see the peewee model below for more info on␣
 ↪each field:
vehicles[:1]
```

[47]:
```
[{'BlockFareboxId': 24,
  'CommStatus': 'GOOD',
  'Destination': 'Five Town Plaza via Union Station',
  'Deviation': 0,
  'Direction': 'S',
  'DirectionLong': 'Southbound',
  'DisplayStatus': 'On Time',
```

```
 'StopId': 679,
 'CurrentStatus': None,
 'DriverName': None,
 'DriverLastName': None,
 'DriverFirstName': None,
 'DriverFareboxId': 0,
 'VehicleFareboxId': 1866,
 'GPSStatus': 2,
 'Heading': 186,
 'LastStop': 'Grove / Court (2)',
 'LastUpdated': '/Date(1712865680000-0400)/',
 'Latitude': 42.156162,
 'Longitude': -72.585326,
 'Name': '1866',
 'OccupancyStatus': 0,
 'OnBoard': 2,
 'OpStatus': 'ONTIME',
 'RouteId': 20001,
 'RunId': 1557322,
 'Speed': None,
 'TripId': 1555,
 'VehicleId': 1866,
 'SeatingCapacity': 40,
 'TotalCapacity': 56,
 'PropertyName': 'SATCO',
 'OccupancyStatusReportLabel': 'Empty',
 'LastUpdated_timestamp_ms': '1712865680000'}]
```

0.1.3 Approach 1: SQLite with Peewee

[48]:
```
# First, let's create a DB file if it does not exist already:
with open('pvta-sqlite-data.sqlite', 'a+') as f:
    pass
```

[49]:
```
# Now, create the peewee sqlite model:
import json, re, peewee
import datetime
from peewee import Model, CharField, DoubleField, IntegerField, DateTimeField

db = peewee.SqliteDatabase("pvta-sqlite-data.sqlite")

class VehiclePositionRecord(Model):
    # Different values, but both appear to ID a vehicle:
    Name =  CharField()
    VehicleId = IntegerField()
    # Position info:
    Latitude = DoubleField()
```

```
    Longitude = DoubleField()
    # Time info:
    LastUpdated = DateTimeField()
    LastUpdated_timestamp_ms = IntegerField()
    # Ex: 10043 for the B43
    RouteId = IntegerField()
    # Lateness in minutes. (Could be more precise by retreiving the schedule␣
↪data, but still useful)
    Deviation = IntegerField()
    # NOT UNIQUE; RunId seems to update everytime the vehicle starts running␣
↪for the day (or leaves the garage?)
    RunId = IntegerField()
    # NOT UNIQUE; updated everytime the vehicle starts moving from the 1st stop␣
↪in a route. The value is simply the scheduled time (7am -> 700)
    TripId = IntegerField()
    # Other useful info
    Direction =  CharField()
    Destination = CharField()
    LastStop = CharField(null = True)

    # Not too important, a lot of these are NULL:
    OpStatus = CharField(null = True)
    Heading = IntegerField(null = True)
    OnBoard = IntegerField(null = True)
    TotalCapacity = IntegerField(null = True)
    BlockFareboxId = IntegerField(null = True)
    CommStatus =  CharField(null = True)
    OccupancyStatus = IntegerField(null = True)
    DirectionLong =  CharField(null = True)
    DisplayStatus =  CharField(null = True)
    DriverFareboxId = IntegerField(null = True)
    VehicleFareboxId = IntegerField(null = True)
    GPSStatus = IntegerField(null = True)
    SeatingCapacity = IntegerField(null = True)
    PropertyName =  CharField(null = True)
    OccupancyStatusReportLabel =  CharField(null = True)
    StopId = IntegerField(null = True)

    class Meta:
        database = db
        # This prevents having two records of the same vehicle at the same time
        indexes = (
          (('LastUpdated', 'VehicleId'), True),
        )
```

[50]:
```
# Needed for first run, doesn't matter if you run this or not afterwards.␣
 ↪Initialize tables:
db.create_tables([VehiclePositionRecord], safe = True)
```

Saving the vehicle records to the database:

[ ]:
```
# Add the vehicles:
for v in vehicles:
    vehicle_record = VehiclePositionRecord(
            Name=v['Name'],
            VehicleId=v['VehicleId'],
            Latitude=v['Latitude'],
            Longitude=v['Longitude'],
            # the '-3' removes the last 3 digits, converting␣
 ↪milliseconds->seconds
            LastUpdated=datetime.datetime.fromtimestamp(int(re.search(r'\d+',␣
 ↪v['LastUpdated_timestamp_ms']).group()[:-3])),
            LastUpdated_timestamp_ms = v['LastUpdated_timestamp_ms'],
            RouteId=v['RouteId'],
            Direction=v['Direction'],
            Destination=v['Destination'],
            Deviation=v['Deviation'],
            RunId=v['RunId'],
            TripId=v['TripId'],
            LastStop=v['LastStop'],
            OpStatus=v['OpStatus'],
            Heading=v['Heading'],
            OnBoard=v['OnBoard'],
            TotalCapacity=v['TotalCapacity'],
            BlockFareboxId=v['BlockFareboxId'],
            CommStatus=v['CommStatus'],
            OccupancyStatus=v['OccupancyStatus'],
            DirectionLong=v['DirectionLong'],
            DisplayStatus=v['DisplayStatus'],
            DriverFareboxId=v['DriverFareboxId'],
            VehicleFareboxId=v['VehicleFareboxId'],
            GPSStatus=v['GPSStatus'],
            SeatingCapacity=v['SeatingCapacity'],
            PropertyName=v['PropertyName'],
            OccupancyStatusReportLabel=v['OccupancyStatusReportLabel'],
            StopId=v['StopId']
        )
    vehicle_record.save()
```

Let's select B43 buses, going west, since 9pm April 10th:

[36]:
```
start_date = datetime.datetime(2024, 4, 10, 21)
end_date = datetime.datetime.now()

s = VehiclePositionRecord.select().where(VehiclePositionRecord.RouteId == 10043)
# Constraints can be chained separately or together using '&' (AND) or '|' (OR)
s = s.where(
    (VehiclePositionRecord.Direction == 'W') &
    (VehiclePositionRecord.LastUpdated.between(start_date, end_date))
)

for vRecord in s:
    print()
    print(vRecord.LastUpdated, vRecord.RunId, vRecord.VehicleId, vRecord.
 ↪LastStop, vRecord.Direction)
    print()
```

```
2024-04-10 21:31:27 1545622 701 Prospect Street W


2024-04-10 21:31:28 1545613 502 Haigis Mall W


2024-04-10 21:46:43 1545622 701 Prospect Street W


2024-04-10 21:48:19 1545613 502 Russell/Russell (The Stables) W


2024-04-11 16:01:27 1545616 701 Russell/Rte 9 (Holiday Inn Express) W


2024-04-11 16:01:29 1545610 412 Fearing Street (In) W
```

Let's show the percentage of B43 buses over 5 minutes late:

[52]:
```
five_mins_late = VehiclePositionRecord.select().where(VehiclePositionRecord.
 ↪RouteId == 10043).where(VehiclePositionRecord.Deviation > 5)
all_b43 = VehiclePositionRecord.select().where(VehiclePositionRecord.RouteId ==␣
 ↪10043)
# Could be 0 if you don't have enough data, it was around 18% on my copy with␣
 ↪~170k records.
(five_mins_late.count() / all_b43.count()) * 100
```

[52]:
```
9.090909090909092
```

Since all the data is in a `sqlite` file, you can also use sqlite to query the data

0.1.4 Approach 2: Pandas

[38]:
```
# First, create a dataframe from the vehicles list:
import pandas as pd
import json
df = pd.read_json(json.dumps(vehicles))

# Add a datetime column for easier querying of the `LastUpdated` field:
df['LastUpdated_datetime'] = pd.to_datetime(df['LastUpdated_timestamp_ms'],␣
 ↪unit='ms')
```

[39]:
```
# Let's show rows from the B43:
df[df['RouteId'] == 10043]
```

[39]:

	BlockFareboxId	CommStatus	Destination	Deviation	\
82	4308	GOOD	Amherst College via UMass	1	
83	4303	GOOD	Amherst College via UMass	0	
84	4304	GOOD	Northampton via Hampshire Mall	6	
85	4309	GOOD	Northampton via Hampshire Mall	0	
86	4305	GOOD	Amherst College via UMass	1	

	Direction	DirectionLong	DisplayStatus	StopId	CurrentStatus	DriverName	\
82	E	East	On Time	0	NaN	NaN	
83	E	East	On Time	0	NaN	NaN	
84	W	West	Late	0	NaN	NaN	
85	W	West	On Time	0	NaN	NaN	
86	E	East	On Time	0	NaN	NaN	

	…	RunId	Speed	TripId	VehicleId	SeatingCapacity	TotalCapacity	\
82	…	1545622	NaN	1540	411	40	56	
83	…	1545608	NaN	1520	501	49	69	
84	…	1545616	NaN	1530	701	40	56	
85	…	1545610	NaN	1550	412	40	56	
86	…	1545620	NaN	1600	419	40	56	

	PropertyName	OccupancyStatusReportLabel	LastUpdated_timestamp_ms	\
82	VATCO	Many Seats Available	1712865684000	
83	VATCO	Many Seats Available	1712865685000	
84	VATCO	Many Seats Available	1712865687000	
85	VATCO	Many Seats Available	1712865689000	
86	VATCO	Many Seats Available	1712865696000	

	LastUpdated_datetime
82	2024-04-11 20:01:24
83	2024-04-11 20:01:25
84	2024-04-11 20:01:27
85	2024-04-11 20:01:29

```
86   2024-04-11 20:01:36

[5 rows x 35 columns]
```

Append the pandas data to a parquet file:

[43]:
```python
# Creates file if not already existing:
try:
    old_df = pd.read_parquet('pvta-parquet-data.parquet')
    df = pd.concat([df, old_df])
except FileNotFoundError:
    pass
df.to_parquet('pvta-parquet-data.parquet')
```

[ ]:
```python
#TODO: select buses in a particular time frame.
```