Time Series Analytics with Simple Relational Database Paradigms

Ben Leighton, Julia Anticev, Alex Khassapov

LAND AND WATER & CSIRO IMT SCIENTIFIC COMPUTING
www.csiro.au

Context

Energy Use Data Model (EUDM) endeavours to deliver a wide range of energy datasets to a broad set of stakeholders especially the research community

As part of this provision transformations and views on data provide value by reducing complexity, building fit for purpose structures, and aligning observations to reduce the gap between data and insight

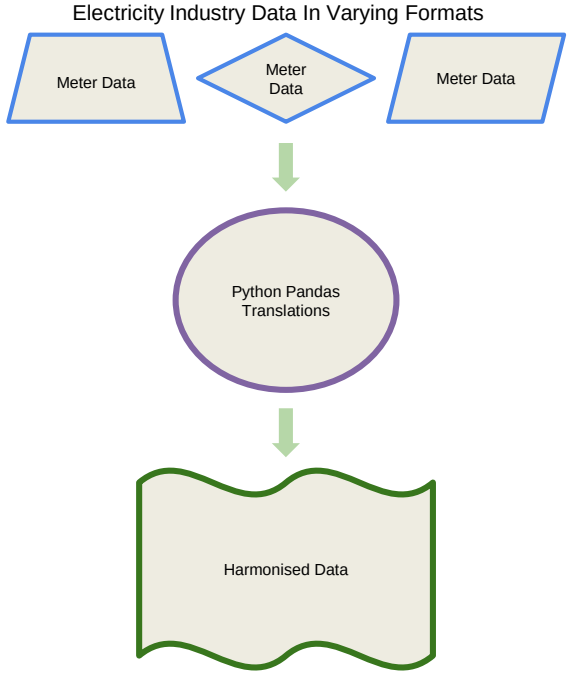
As EUDM developers: Is there a, performant, rapidly deployable, conceptual simple approach, that leverages existing technology and is industry proven for providing an analytics platform?

Building a Foundation
EUDM Data Harmonisation

Many data providers, for example energy utility companies

Not Surprisingly: They all structure data differently. However some of this data is similar, for example many utilities meter readings from substations

For researchers it is useful to harmonize the format across such similar datasets. EUDM do this for substation data to provide a single format for further analysis



Characteristics of Harmonised EUDM Data

218 million substation sensor records

100s of substations

Data for > 10 years

Varying Power Units

Fairly Static (we don't get new data that often yet)

Up to 3 measurements per record
Active Power
Reactive Power
Apparent Power

45gb of csv data but a fair bit of intentional redundancy in columns like units

Varying timesteps
Usually observations every 30 minutes

Harmonized to consistent column schema

Views and Analytics in EUDM

Column Based CSV
- Conciseness

Consistent Timesteps
- Interoperability

Aggregated Observations
- Summaries, Statistics, Tractability

Unit Harmony / Conversions
- Interoperability

(Future) Real time Analytics Platforms

Index	StartDeliveryTime	EndDeliveryTime	UtilityName	-|Wanniassa Zone|ApparentPower	-|Wanniassa Zone|ApparentPowerUnits
0	2004-07-01 00:00:00	2004-07-01 00:30:00	ActewAGL	41.29	MVA
1	2004-07-01 00:30:00	2004-07-01 01:00:00	ActewAGL	38.14	MVA
2	2004-07-01 01:00:00	2004-07-01 01:30:00	ActewAGL	35.71	MVA
3	2004-07-01 01:30:00	2004-07-01 02:00:00	ActewAGL	33.74	MVA
4	2004-07-01 02:00:00	2004-07-01 02:30:00	ActewAGL	32.23	MVA
5	2004-07-01 02:30:00	2004-07-01 03:00:00	ActewAGL	31.47	MVA
6	2004-07-01 03:00:00	2004-07-01 03:30:00	ActewAGL	31.52	MVA
7	2004-07-01 03:30:00	2004-07-01 04:00:00	ActewAGL	31.09	MVA
8	2004-07-01 04:00:00	2004-07-01 04:30:00	ActewAGL	31.61	MVA
9	2004-07-01 04:30:00	2004-07-01 05:00:00	ActewAGL	32.92	MVA
10	2004-07-01 05:00:00	2004-07-01 05:30:00	ActewAGL	35.24	MVA
11	2004-07-01 05:30:00	2004-07-01 06:00:00	ActewAGL	39.59	MVA
12	2004-07-01 06:00:00	2004-07-01 06:30:00	ActewAGL	46.42	MVA
13	2004-07-01 06:30:00	2004-07-01 07:00:00	ActewAGL	54.93	MVA
14	2004-07-01 07:00:00	2004-07-01 07:30:00	ActewAGL	64.53	MVA

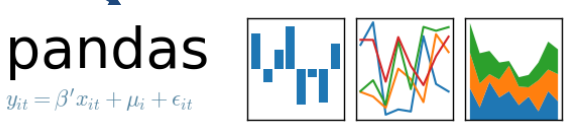
index	min	max	avg	UtilityName	SupplyRegionName
0	2010-12-31 20:30	2010-12-31 21:00	28.46233	WesternPower	Beechboro
1	2010-12-31 21:00	2010-12-31 21:30	27.35437	WesternPower	Beechboro
2	2010-12-31 21:30	2010-12-31 22:00	26.25408	WesternPower	Beechboro
3	2010-12-31 22:00	2010-12-31 22:30	25.3218	WesternPower	Beechboro
4	2010-12-31 22:30	2010-12-31 23:00	24.29368	WesternPower	Beechboro
5	2010-12-31 23:00	2010-12-31 23:10	23.5839	WesternPower	Beechboro
6	2010-12-31 22:40	2010-12-31 23:00	123.5725	WesternPower	Boddington Gold Mine
7	2010-12-31 23:00	2010-12-31 23:30	120.64	WesternPower	Boddington Gold Mine
8	2010-12-31 23:30	2011-01-01 00:00	124.475	WesternPower	Boddington Gold Mine

Options for Time Series Analysis

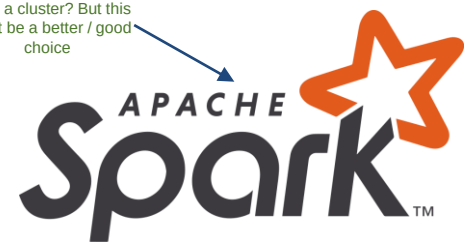
No schema, Kind of free but you pay for clustering Online skepticism about performance

2014 - InfluxDB - https://www.influxdata.com/
Land and Water Weather Station Data Hybrid SOS system

Pandas working well for us but not when we run out of memory. Clunky querying.



Learning Curve, Do I need a cluster? But this might be a better / good choice



Already started by the time it was suggested down the track but might be a good option

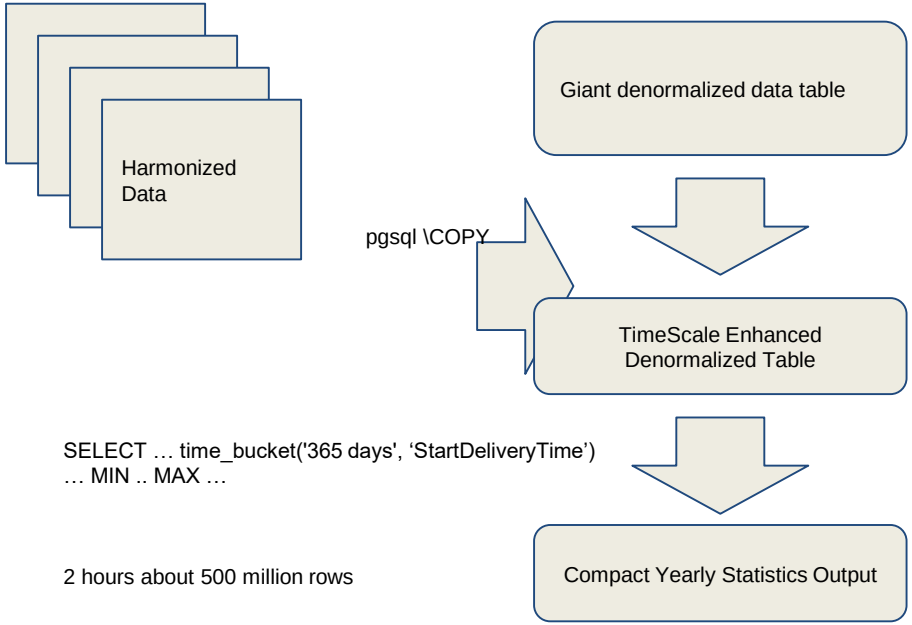
Very interesting as built on top of postgresql and at least I know that is high quality

TimeScale DB for Quality Assurance

To debug the data harmonization process we needed a way of spotting systemic errors quickly

TimeScale provided a way to quickly aggregate data to yearly average maximum and minimum (thanks pre built Docker Image and Python + Jupyter)

Physically unrealistic averages and outlier maximums and minimums could indicate potential process errors but also might reflect errors in original data



Postgres for Time Series Analysis

TimeScale is not well documented and doesn't seem capable of much more than Aggregation

What about native PostgreSQL then?

A PostgreSQL Solution
AWS, Features Used, A Simple Schema

Easy and Powerful - AWS RDS

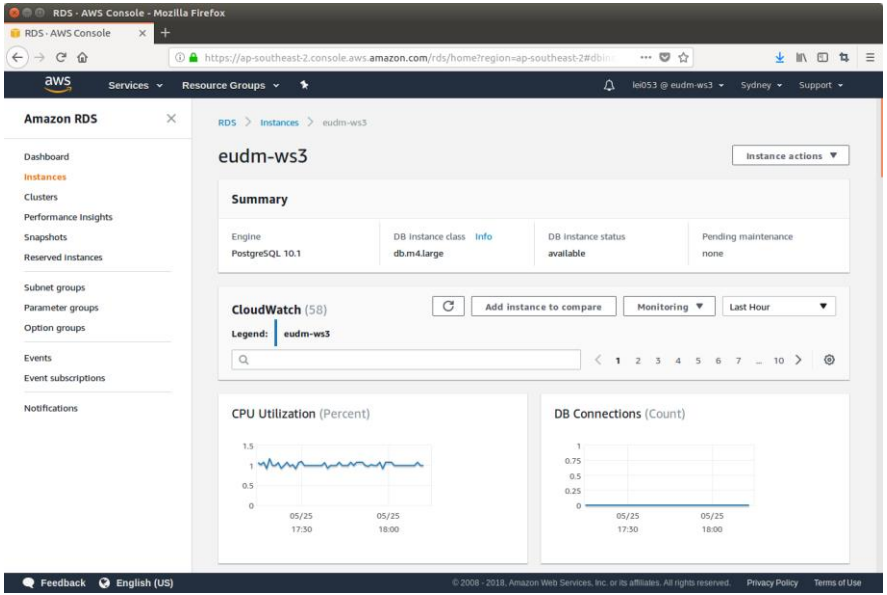
Out of the box scalable DB instances including Postgresql

PostgreSQL 10 supported at up to 64cpus 256GB RAM

Performance Visualisation For Optimisation

Relatively Cheap

Not AWS specific maintaining portability avoiding vendor lock in

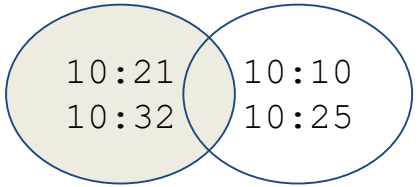


Modern PostgreSQL Features Used

```
In [10]: %%bash
psql -h $DBHOST -U $DBUSER -d eudm -c "SELECT tablename FROM pg_catalog.pg_tables WHERE tablename = 'data' OR tablename LI

        tablename
-----------------------
 data
 data_westernpower
 data_westernpower_1985
 data_westernpower_1986
 data_westernpower_1987
 data_westernpower_1988
 data_westernpower_1989
 data_westernpower_1990
 data_westernpower_1991
 data_westernpower_1992
(10 rows)
```

Partitioned Tables Matching Likely Query Domain



Many date range operators including overlaps

GiST Indexes

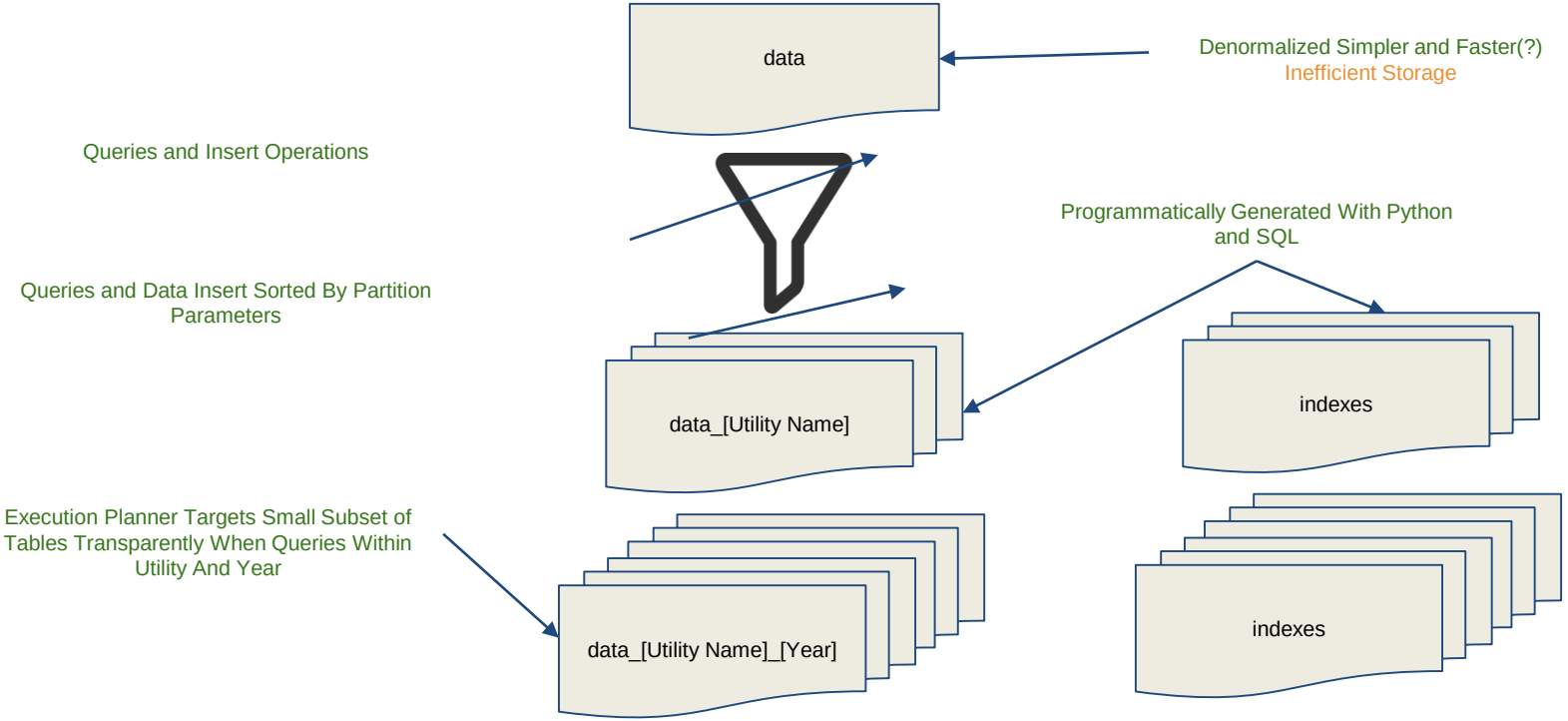
```
->  Append  (cost=9.84..534.71 rows=139 width=117)
      ->  Bitmap Heap Scan on weather_2010  (cost=9.84..534.71 rows=139 width=117)
            Recheck Cond: (("stationNumber" = zone_weather."StationId") AND (data_westernpower_2010."TimeR
eRange"))
```

Parallel Queries

```
QUERY PLAN
------------------------------------------------------------------------------------------------------------------------
 Gather Merge  (cost=116900019.81..141184894.43 rows=199132344 width=251)
   Workers Planned: 6
   ->  Sort  (cost=116899019.71..116981991.52 rows=33188724 width=251)
         Sort Key: data_westernpower_2010."StartDeliveryTime", weather_2010."startObservationTime"
         ->  Nested Loop  (cost=23.77..108782719.31 rows=33188724 width=251)
               ->  Hash Join  (cost=13.93..510120.93 rows=201963 width=193)
                     Hash Cond: (data_westernpower_2010."SupplyRegionID" = zone_weather."ZoneId")
                     ->  Append  (cost=0.00..484702.80 rows=2672522 width=188)
                           ->  Parallel Seq Scan on data_westernpower_2010  (cost=0.00..484702.80 rows=2672522 width=188)
```

A fair bit of inspiration from https://dba.stackexchange.com/a/39599 Erwin Brandstetter

A Simple Schema for Time Series Analysis



Ephemeral Reproducible Database

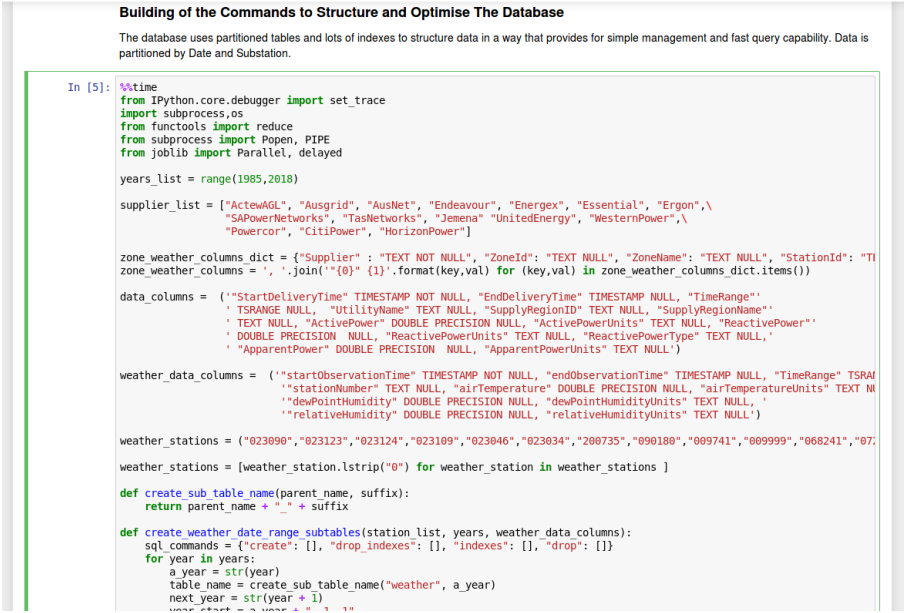
Database creation and data upload via Jupyter for rapid development

DDL/SQL via Templates with Python

Programmatic Index Creation Mitigating Duplication (required until Postgresql11)

Needs refactoring into robust libraries

Python Workflow Engines?



Building of the Commands to Structure and Optimise The Database

The database uses partitioned tables and lots of indexes to structure data in a way that provides for simple management and fast query capability. Data is partitioned by Date and Substation.

In [5]:

```
%%time
from IPython.core.debugger import set_trace
import subprocess,os
from functools import reduce
from subprocess import Popen, PIPE
from joblib import Parallel, delayed

years_list = range(1985,2018)

supplier_list = ["ActewAGL", "Ausgrid", "AusNet", "Endeavour", "Energex", "Essential", "Ergon",\
                 "SAPowerNetworks", "TasNetworks", "Jemena" "UnitedEnergy", "WesternPower",\
                 "Powercor", "CitiPower", "HorizonPower"]

zone_weather_columns_dict = {"Supplier" : "TEXT NOT NULL", "ZoneId": "TEXT NULL", "ZoneName": "TEXT NULL", "StationId": "TI
zone_weather_columns = ', '.join('"{0}" {1}'.format(key,val) for (key,val) in zone_weather_columns_dict.items())

data_columns =  ('"StartDeliveryTime" TIMESTAMP NOT NULL, "EndDeliveryTime" TIMESTAMP NULL, "TimeRange"'
                 ' TSRANGE NULL,  "UtilityName" TEXT NULL, "SupplyRegionID" TEXT NULL, "SupplyRegionName"'
                 ' TEXT NULL, "ActivePower" DOUBLE PRECISION NULL, "ActivePowerUnits" TEXT NULL, "ReactivePower"'
                 ' DOUBLE PRECISION  NULL, "ReactivePowerUnits" TEXT NULL, "ReactivePowerType" TEXT NULL,'
                 ' "ApparentPower" DOUBLE PRECISION  NULL, "ApparentPowerUnits" TEXT NULL')

weather_data_columns =  ('"startObservationTime" TIMESTAMP NOT NULL, "endObservationTime" TIMESTAMP NULL, "TimeRange" TSRAI
                          '"stationNumber" TEXT NULL, "airTemperature" DOUBLE PRECISION NULL, "airTemperatureUnits" TEXT NU
                          '"dewPointHumidity" DOUBLE PRECISION NULL, "dewPointHumidityUnits" TEXT NULL, '
                          '"relativeHumidityUnits" DOUBLE PRECISION NULL, "relativeHumidityUnits" TEXT NULL')


weather_stations = ("023090","023123","023124","023109","023046","023034","200735","090180","009741","009999","068241","07

weather_stations = [weather_station.lstrip("0") for weather_station in weather_stations ]

def create_sub_table_name(parent_name, suffix):
    return parent_name + "_" + suffix

def create_weather_date_range_subtables(station_list, years, weather_data_columns):
    sql_commands = {"create": [], "drop_indexes": [], "indexes": [], "drop": []}
    for year in years:
        a_year = str(year)
        table_name = create_sub_table_name("weather", a_year)
        next_year = str(year + 1)
        year_start = a_year + "-1-1"
```

Time Series Queries

PostgreSQL SQL is succinct and simple for Time Series Analytics

```
SELECT min("StartDeliveryTime"),max("EndDeliveryTime),
avg("ActivePower"),"UtilityName","SupplyRegionName"
```

Aggregate time series to half hourly

```
FROM DATA WHERE

"UtilityName" = 'WesternPower'

GROUP BY  "UtilityName",  "SupplyRegionName",

floor(EXTRACT(epoch FROM  "StartDeliveryTime") / 60 / 30)

ORDER BY "UtilityName",  "SupplyRegionName", min("StartDeliveryTime")
```

convert date to seconds since 1/1/1970

get the fractional number of half hourly increments since 1/1/1970

floor to specify which half hour this time belongs to

group all times in the same half hour together and apply functions to aggregate values to a new half hourly record

Find any overlapping weather observations and substation sensor measurements even if they are at different timesteps with unaliigned start and end dates

```
SELECT ... [LOTS OF COLUMNS] ...
JOIN zone_weather ON zone_weather."StationId" = weather."stationNumber"
JOIN data ON
data."SupplyRegionID" = zone_weather."ZoneId"
AND
(data."TimeRange" && weather."TimeRange")
WHERE data."UtilityName" = 'WesternPower' AND ...
[LIMIT THIS TO A HAPPEN WITHIN A YEAR... ] ... AND
zone_weather."Supplier" = 'WesternPower' ORDER BY ...  [TIME] ...
```

date range overlaps operator

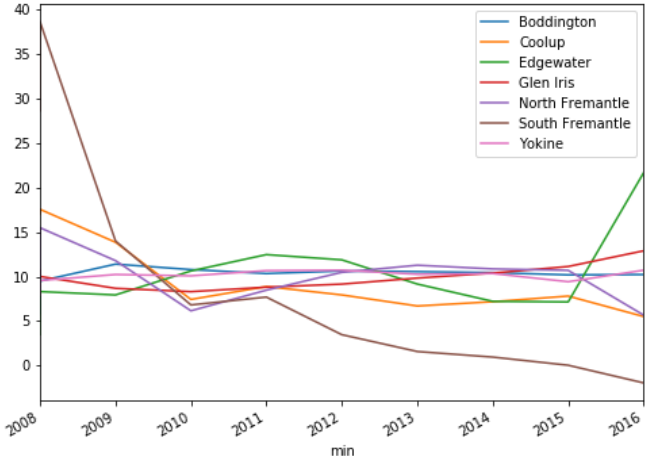
Performance

Aggregation 100 million Observations constituting 150 Substations over > 10 years at from 5 minute to 30 minute ~ 20 million output rows ~ 18 minutes

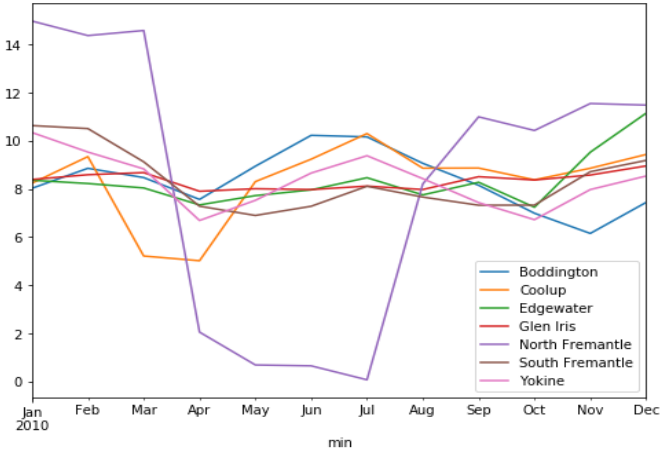
Overlaps join 2010 weather data (30 Minute) and 2010 Western Power Data (5 Minute) over 150 Substations ~ 14 Million Output Rows ~ 8 Minutes

Aggregation 100 million Observations constituting 150 Substations over > 10 years at from 5 minute to Yearly ~ 14 minutes

Research Use Case and Visualisation



Percentage summed average active power consumption over 8 years happening per year for a selection of Perth Substations in 2010



Percentage of average active power consumption happening per month for a selection of Perth Substations in 2010

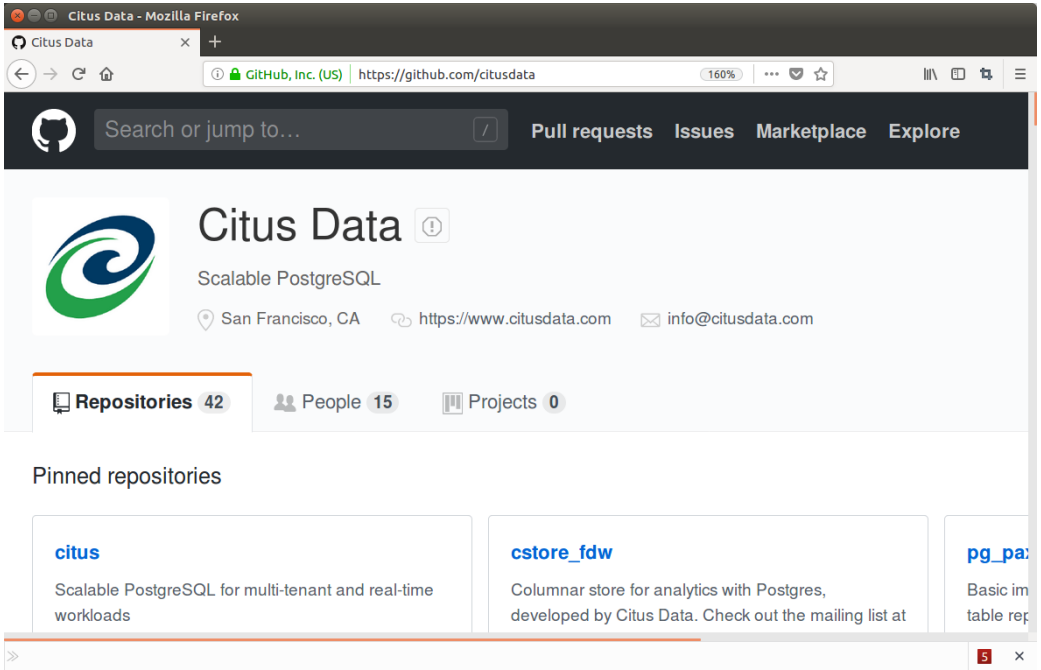
Potential For Real Time Analytics

Columnar Stores

Sharding

Pre Canned Aggregations

Limiting Workloads via Query Estimation



Future Work

Devops for cheaper AWS usage or inhouse

Further Ingest workflow automation

Provenance Reporting and Approaches

Live Sensors?

Personalized Analytics Environments

Other Technologies: Dask, Apache Spark, Hybrid with Influx, NetCDF

Postgresql 11

Thanks

Land and Water
Ben Leighton
Scientific Compute
Julia Anticev, Alex Khassapov

e ben.leighton@csiro.au