
Parquet vs PostgreSQL
Comparing column stores to a traditional DB

Modeling in data science

- Typically comparing multiple columns to each other
- Or repeatedly selecting values in multiple columns for training a model
- Raw data can contain
  - large numbers of columns
  - huge numbers of rows

Modeling in data science

- Selecting multiple entire columns will be a common operation
- In traditional SQL, queries would take the following form:

```
select c1, c2, …, cn from table;
```

Advantages of SQL tables

- SQL tables are common, well understood and well supported.
- The SQL language allows easy query, export and transportability to other formats.
- SQL is considered speedy for most uses and databases can be hand-configured for extra speed.

Problems with SQL tables

- Insert speed is slow (typically, inserts take 10 times longer than updates).
- Database tables might require security and IT support.
- Result data might have to travel over a network.

Column stores

Column stores are best understood in relation to row stores, which are more common. This represents a **row store**.

a	b	c	d
a1	b1	c1	d1
a2	b2	c2	d2
a3	b3	c3	d3

Here, the data is arranged by row, and the first row is: a, b, c, d.

Column stores

This represents a **column store**. It is the same data as the last example.

a	a1	a2	a3
b	b1	b2	b3
c	c1	c2	c3
d	d1	d2	d3

Here, the data is arranged by column, and the first column is: a, a1, a2, a3.

Data is data, so what’s the difference?

Clearly, these two examples are very similar. Why would column stores be better and when might they be worse than row stores?

An test to measure differences

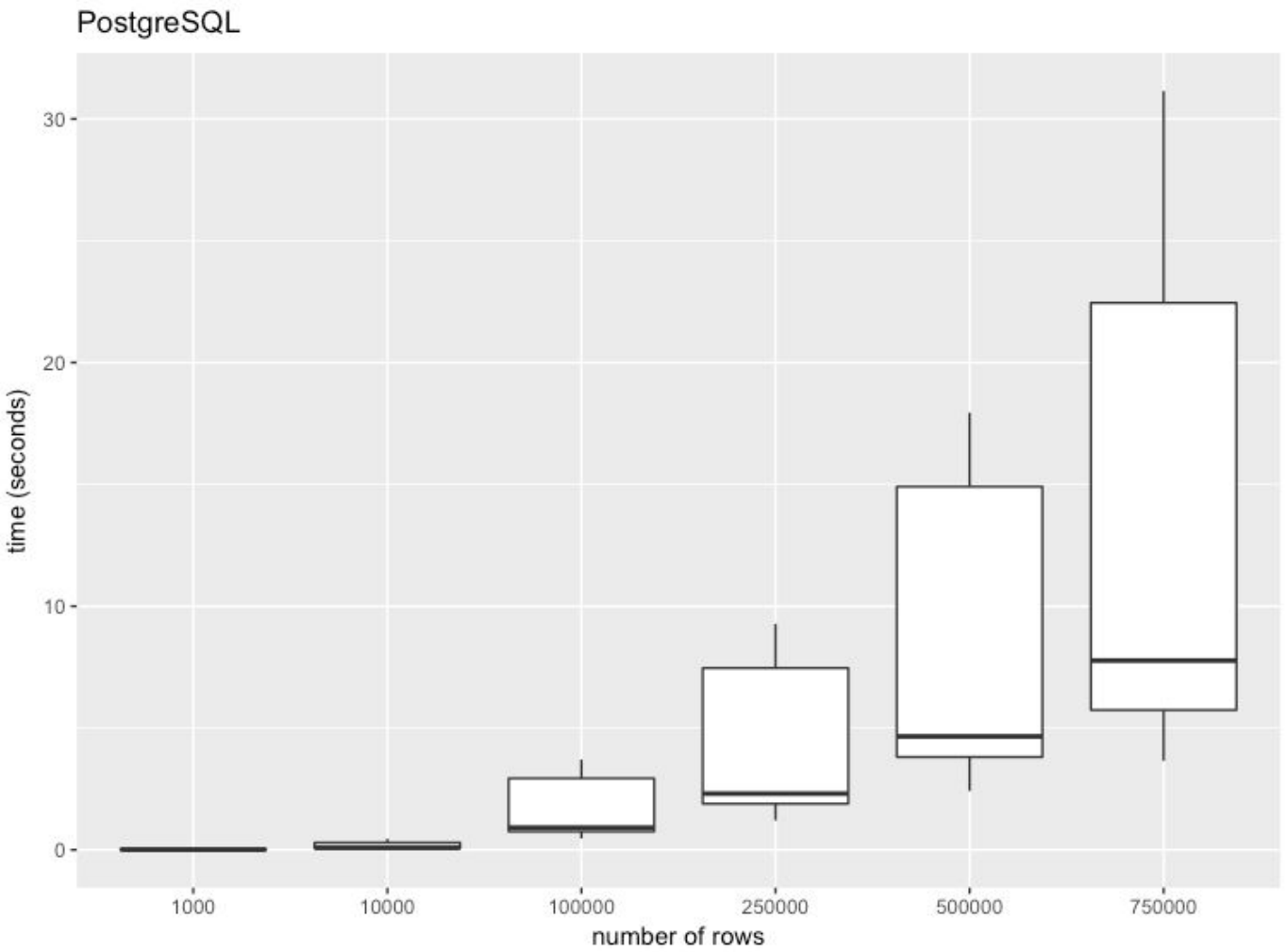
Select a subset of columns from a single table and measure the speed to retrieve all of the data in all selected columns.

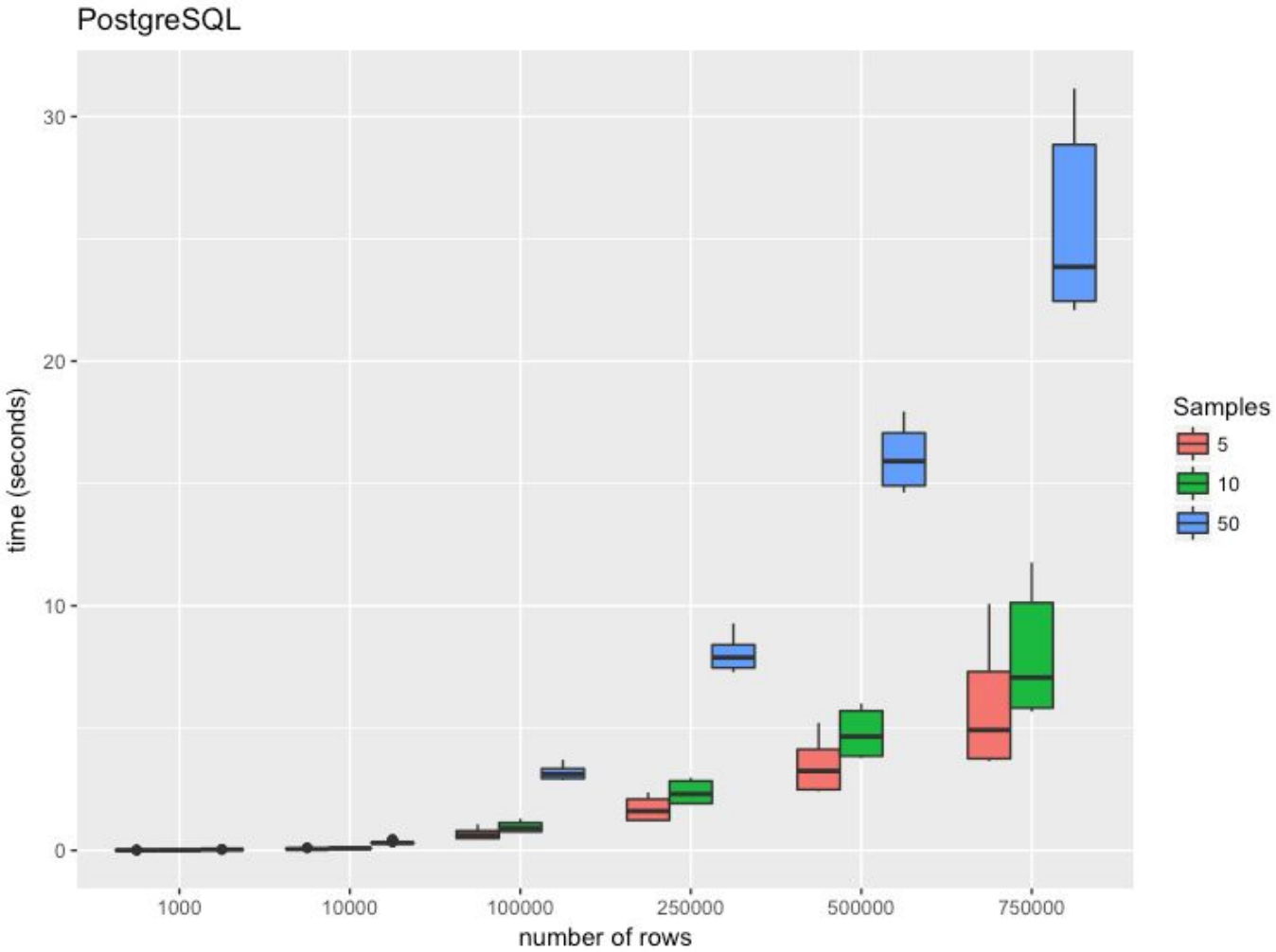
The table will have 100, 500, or 1000 columns.

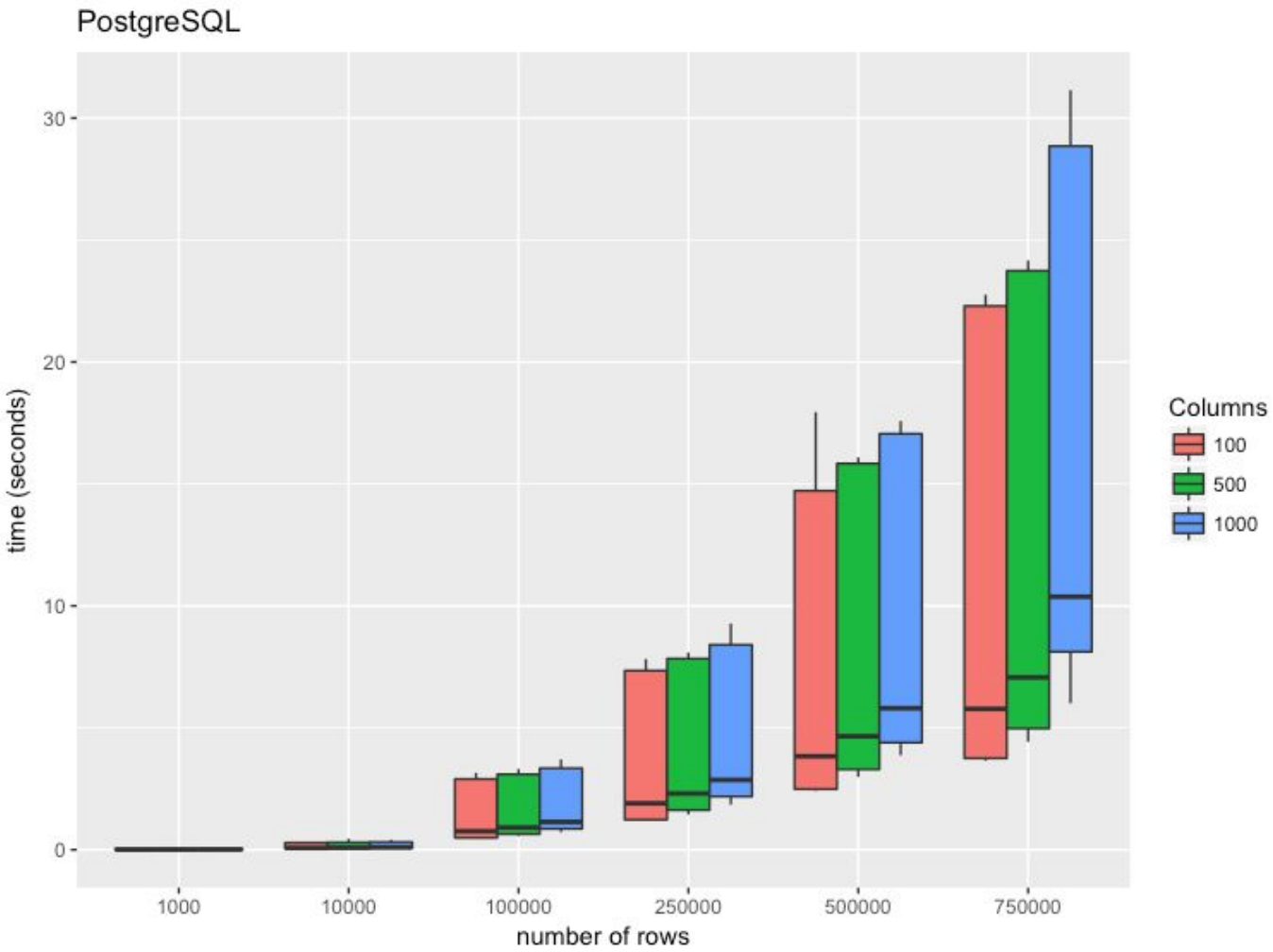
The table will have 1k, 10k, 100k, 250k, 500k or 750k rows.

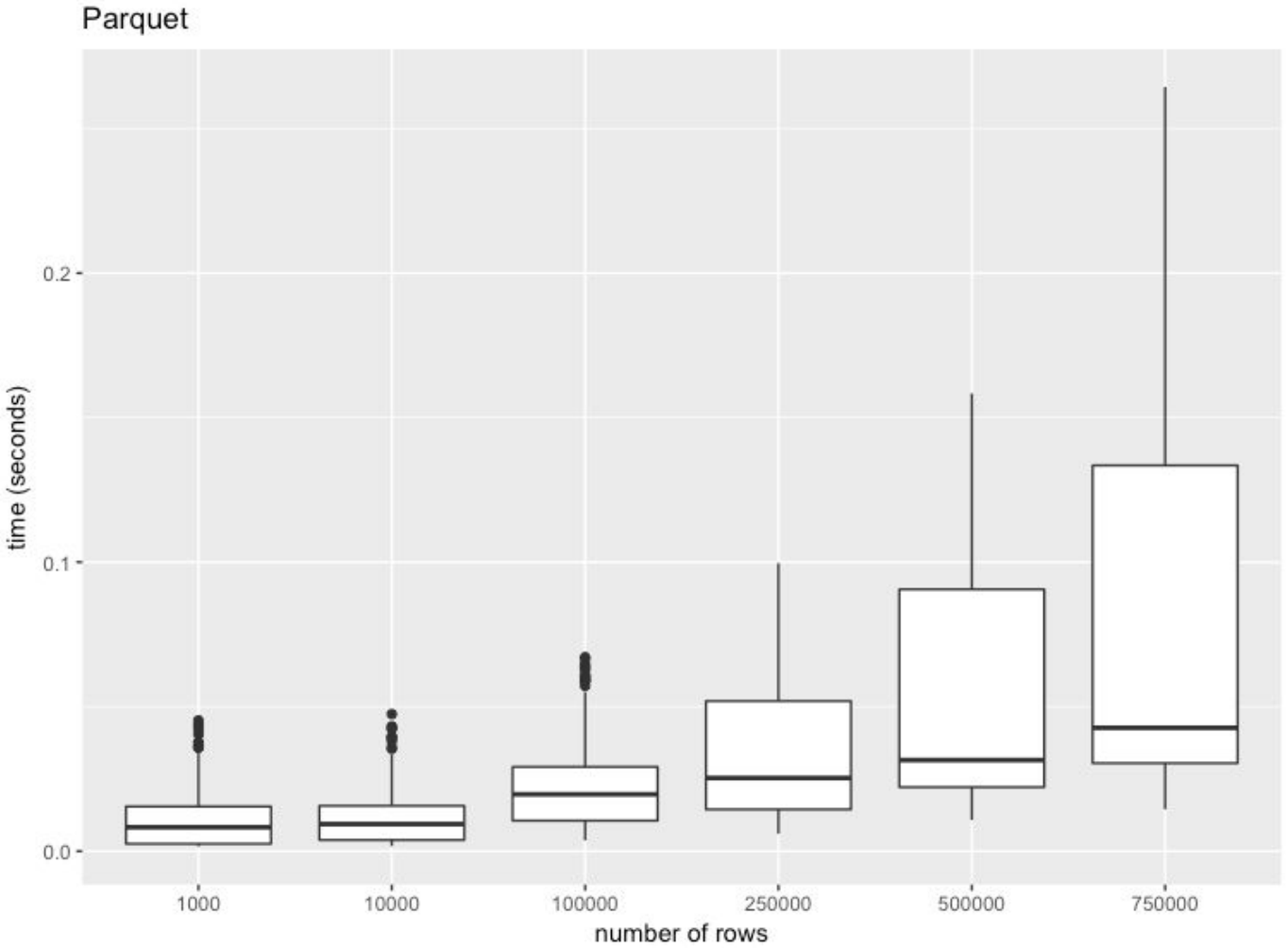
We will select 5, 10 or 50 columns chosen randomly.

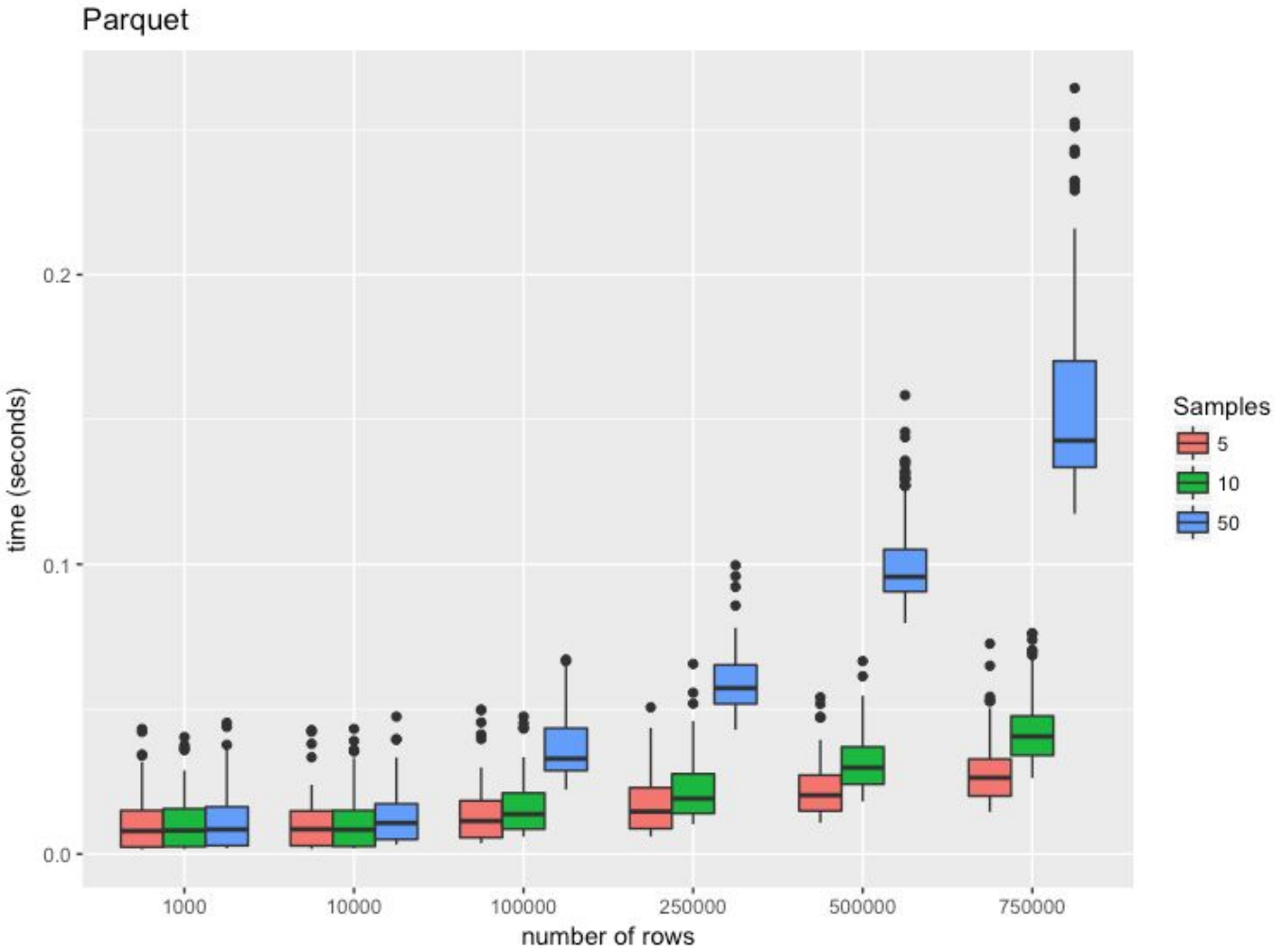
We will perform 100 different random selections per experiment.

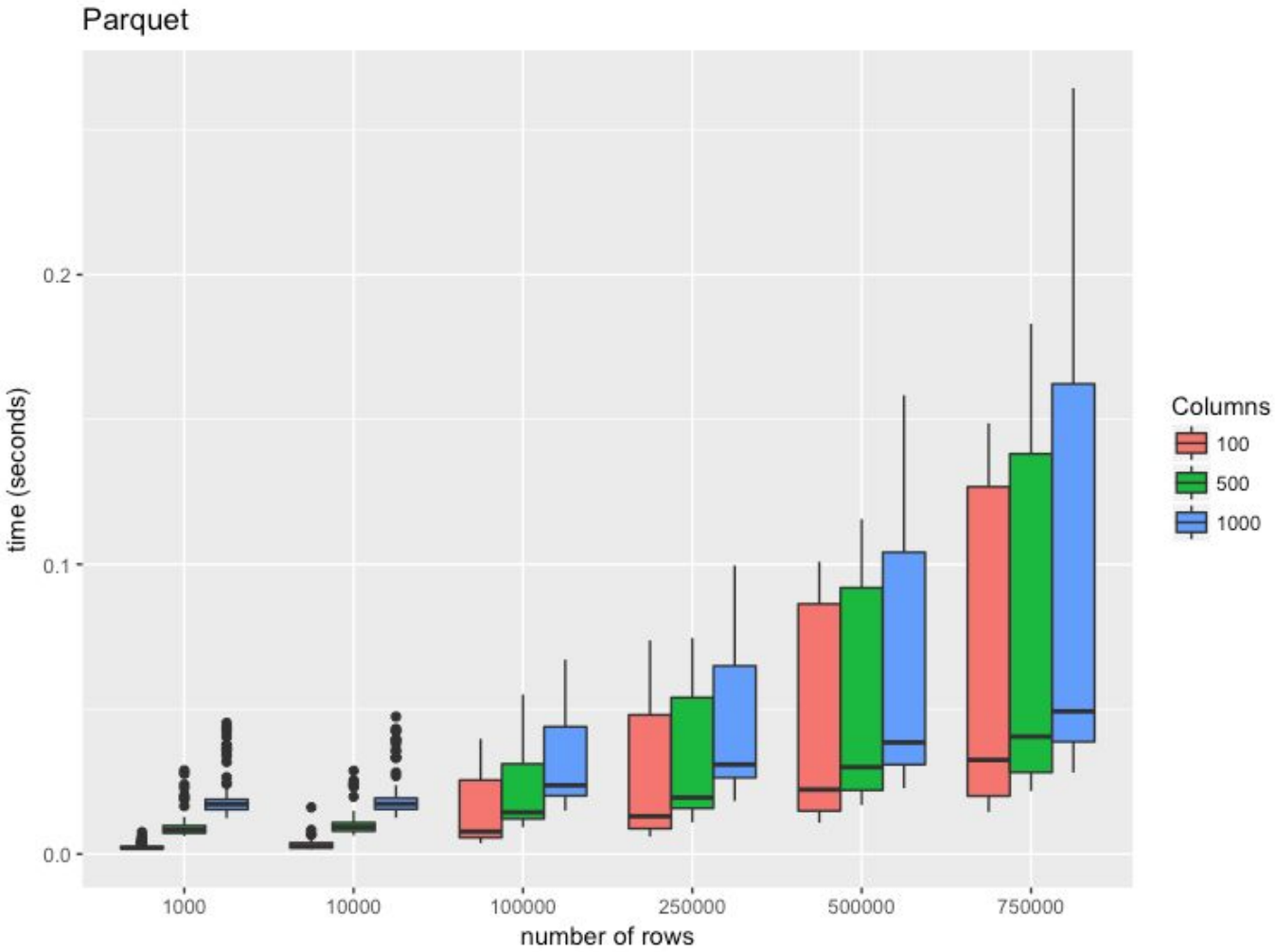

PostgreSQL
time (seconds)
30
20
10
0
1000
10000
100000
250000
500000
750000
number of rows


PostgreSQL
time (seconds)
0
10
20
30
1000
10000
100000
250000
500000
750000
number of rows
Samples
5
10
50


PostgreSQL
time (seconds)
0
10
20
30
1000
10000
100000
250000
500000
750000
number of rows
Columns
100
500
1000


Parquet
time (seconds)
0.0
0.1
0.2
1000
10000
100000
250000
500000
750000
number of rows


Parquet
time (seconds)
0.0
0.1
0.2
1000
10000
100000
250000
500000
750000
number of rows
Samples
5
10
50


Parquet
time (seconds)
0.0
0.1
0.2
1000
10000
100000
250000
500000
750000
number of rows
Columns
100
500
1000

Populating the table: Postgres

```
import os
from sqlalchemy import create_engine, Table, MetaData
from numpy.random import random_sample


def load_table(engine, table, nrows, ncols):
   for i in range(nrows):
       randomdata = random_sample(ncols)
       data = { 'col{:03d}'.format(i): randomdata[i] for i in range(ncols) }
       engine.execute(table.insert(), data)
```

Taking a sample: Postgres

```
import time, os, pdb
import numpy as np
from sqlalchemy import create_engine, Table, MetaData, text
from sqlalchemy.sql import select, column

def select_random(engine, table, nsel=5):
    columns = [col.name for col in table.columns][1:]  # Skip id column
    selected_cols = np.random.choice(columns, nsel, False)
    sql = text("select {} from {}".format(", ".join(selected_cols), table.name))
    results = engine.execute(sql)
    data = []
    for row in results:
        data.append(row)
    return data
```

Creating the table: Postgres

```
import os
from sqlalchemy import *
from optparse import OptionParser

def create_table(engine, metadata, ncols=100):
   cols = [Column('id', Integer, primary_key=True)]
   for i in range(ncols):
       cols.append(Column('col{:03d}'.format(i), Float))
   coltest = Table('column_test', metadata, *cols)
   metadata.create_all(engine)
```

Populating the table: Parquet

```
import numpy as np
from numpy.random import random_sample
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq

def load_table(filename, nrows, ncols=100):
   data = { 'col{:03d}'.format(i): random_sample(nrows) for i in range(ncols) }
   dataframe = pd.DataFrame(data)
   table = pa.Table.from_pandas(dataframe)
   pq.write_table(table, filename)
```

Taking a sample: Parquet

```
import time, os
import numpy as np
import pyarrow.parquet as pq

def select_random(table, ncols, nsel):
   column_names = [ 'col{:03d}'.format(i) for i in range(ncols) ]
   selected_cols = np.random.choice(column_names, nsel, False)
   data = pq.read_table(table, columns=selected_cols)
   return data
```

The PostgreSQL setup

I used a two-container docker-compose instance.

```
version: '3'

services:
  db:
    image: postgres:9.6.11
    restart: unless-stopped
    env_file: env
    ports:
      - "5432:5432"
    volumes:
      - ./tmp/db:/var/lib/postgresql/data
    environment:
      - POSTGRES_PASSWORD=password
  web:
    build: .
    command: python3 app.py
    env_file: env
    volumes:
      - .:/code
    ports:
      - "5000:5000"
    depends_on:
      - db
```

docker-compose.yml

requirements.txt

```
Flask-SQLAlchemy
numpy
redis
jupyter
psycopg2-binary
```

The Parquet setup

I used a single container docker-compose instance.

```
version: '3'
services:
  app:
    build:
      context: .
    command: python app.py
    ports:
      - "8888:8888"
    volumes:
      - .:/code
```

docker-compose.yml

requirements.txt

```
Flask-SQLAlchemy
numpy
pyarrow
jupyter
psycopg2-binary
```