Turbine

Data import, export, and transformation library for PostgreSQL and Pandas

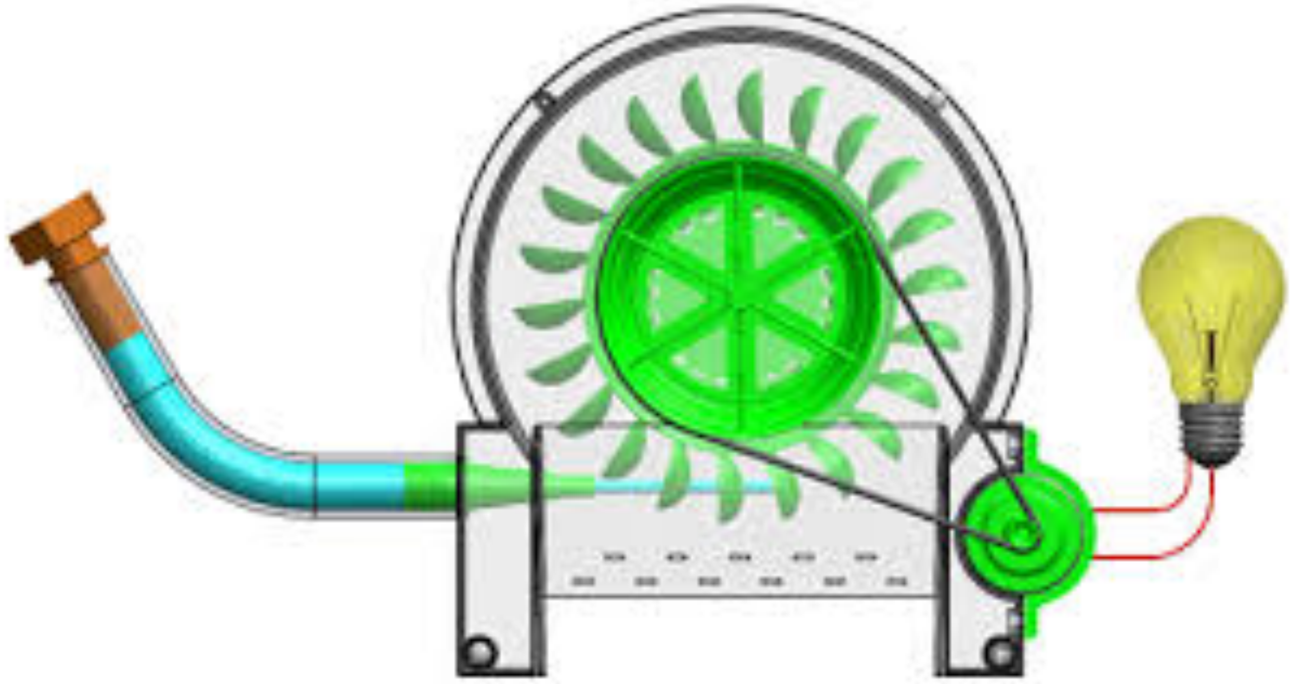
November 22, 2019

WHAT IS TURBINE?

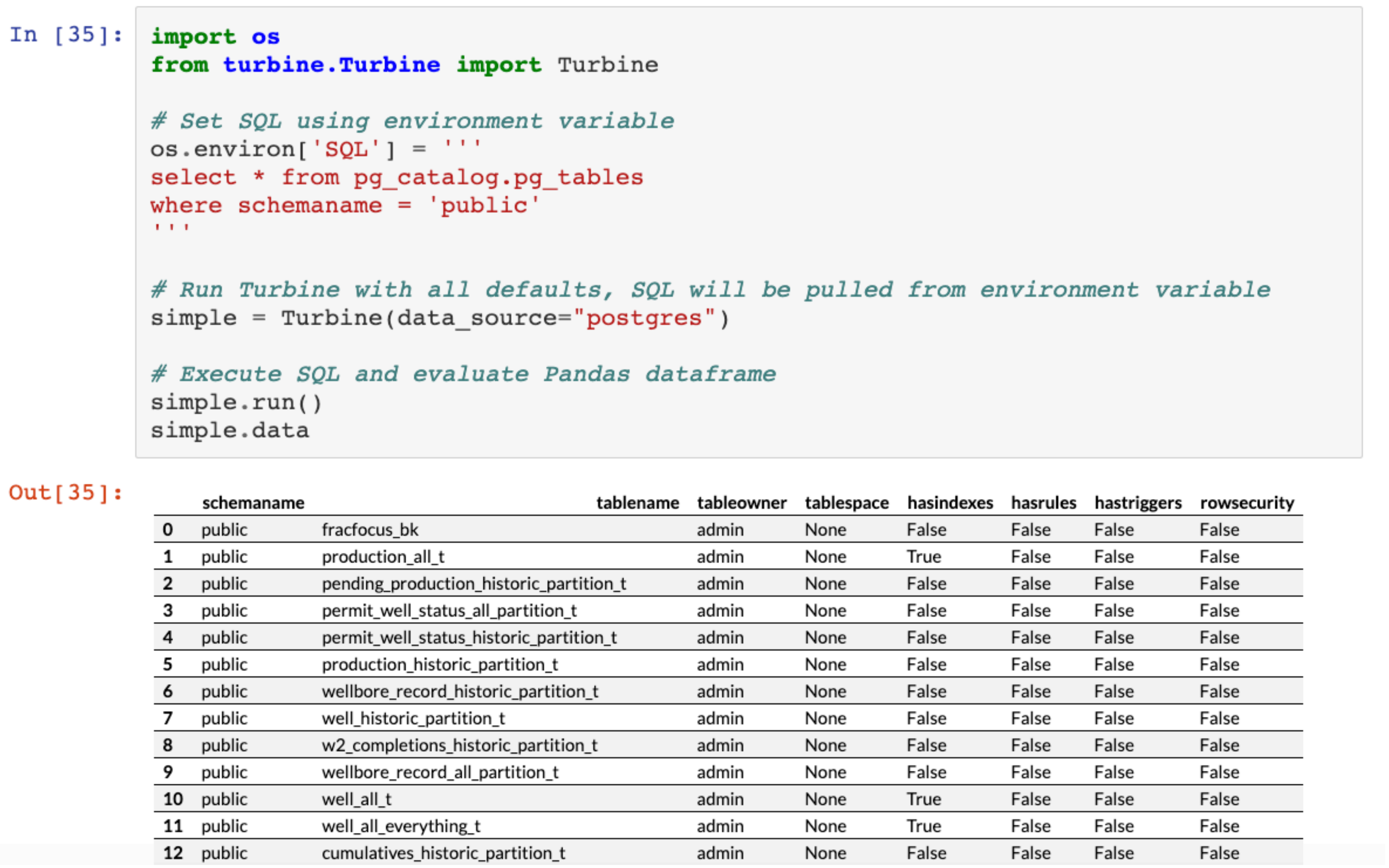
- Turbine is an open source library that works with Jupyter
  - Jupyter is an open-source web application that lets you create and share notebooks that contain live code, saved data, equations, visualizations and narrative text
  - Jupyter is used for:
    - Data cleaning and transformation
    - Numerical simulation
    - Statistical modeling
    - Data visualization
    - Machine learning
- Turbine helps streamline data analysis and engineering in PostgreSQL and Pandas
- It makes it easy to create a SQL file or python variable with SQL, which Turbine will use to extract your target data and transform it into a new data frame, where analysis and visualizations can be performed
- Analysis can be output to a file or to a database table for further use and sharing

WHY TURBINE?

- If data engineering is a pipeline, and databases are reservoirs, then you need a turbine to quickly and efficiently process data to or from the database



EXAMPLE TURBINE USE



In [35]:

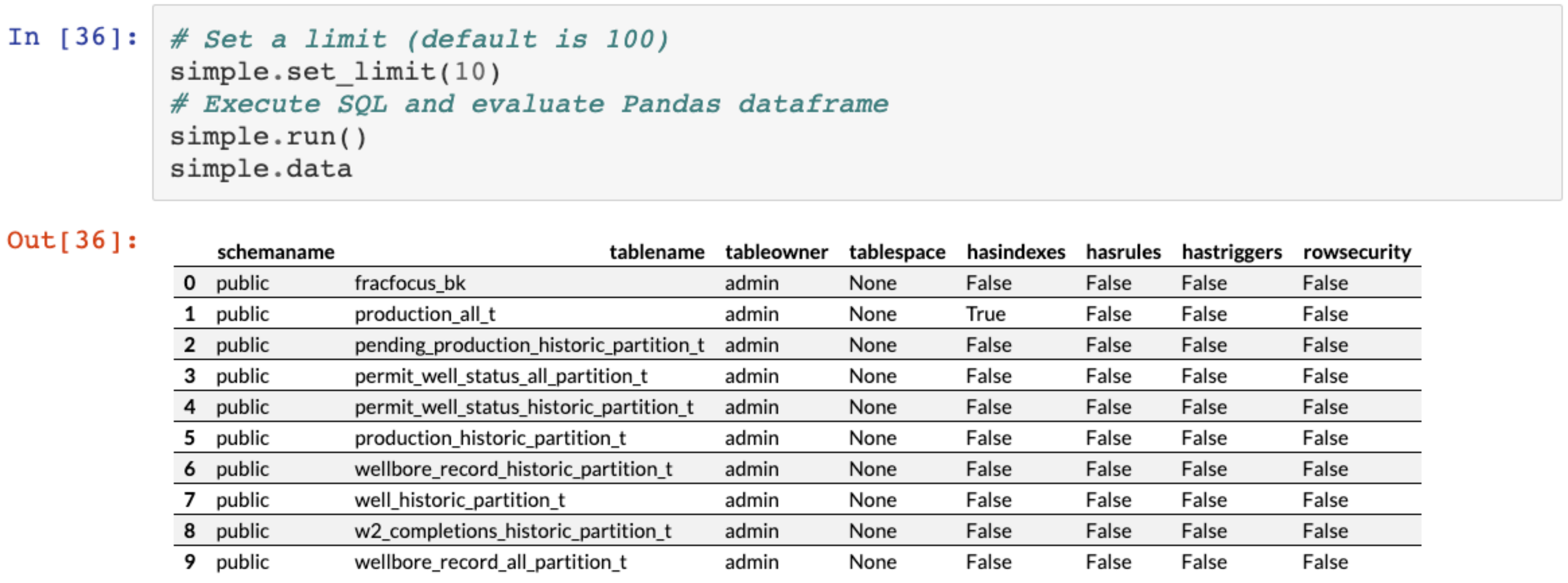
```
import os
from turbine.Turbine import Turbine

# Set SQL using environment variable
os.environ['SQL'] = '''
select * from pg_catalog.pg_tables
where schemaname = 'public'
'''

# Run Turbine with all defaults, SQL will be pulled from environment variable
simple = Turbine(data_source="postgres")

# Execute SQL and evaluate Pandas dataframe
simple.run()
simple.data
```

Out[35]:

	schemaname	tablename	tableowner	tablespace	hasindexes	hasrules	hastriggers	rowsecurity
0	public	fracfocus_bk	admin	None	False	False	False	False
1	public	production_all_t	admin	None	True	False	False	False
2	public	pending_production_historic_partition_t	admin	None	False	False	False	False
3	public	permit_well_status_all_partition_t	admin	None	False	False	False	False
4	public	permit_well_status_historic_partition_t	admin	None	False	False	False	False
5	public	production_historic_partition_t	admin	None	False	False	False	False
6	public	wellbore_record_historic_partition_t	admin	None	False	False	False	False
7	public	well_historic_partition_t	admin	None	False	False	False	False
8	public	w2_completions_historic_partition_t	admin	None	False	False	False	False
9	public	wellbore_record_all_partition_t	admin	None	False	False	False	False
10	public	well_all_t	admin	None	True	False	False	False
11	public	well_all_everything_t	admin	None	True	False	False	False
12	public	cumulatives_historic_partition_t	admin	None	False	False	False	False

SET LIMITS

- To ensure your queries are running as expected it’s good to think about adding a limit
  - You can start small and work your way up



In [36]:

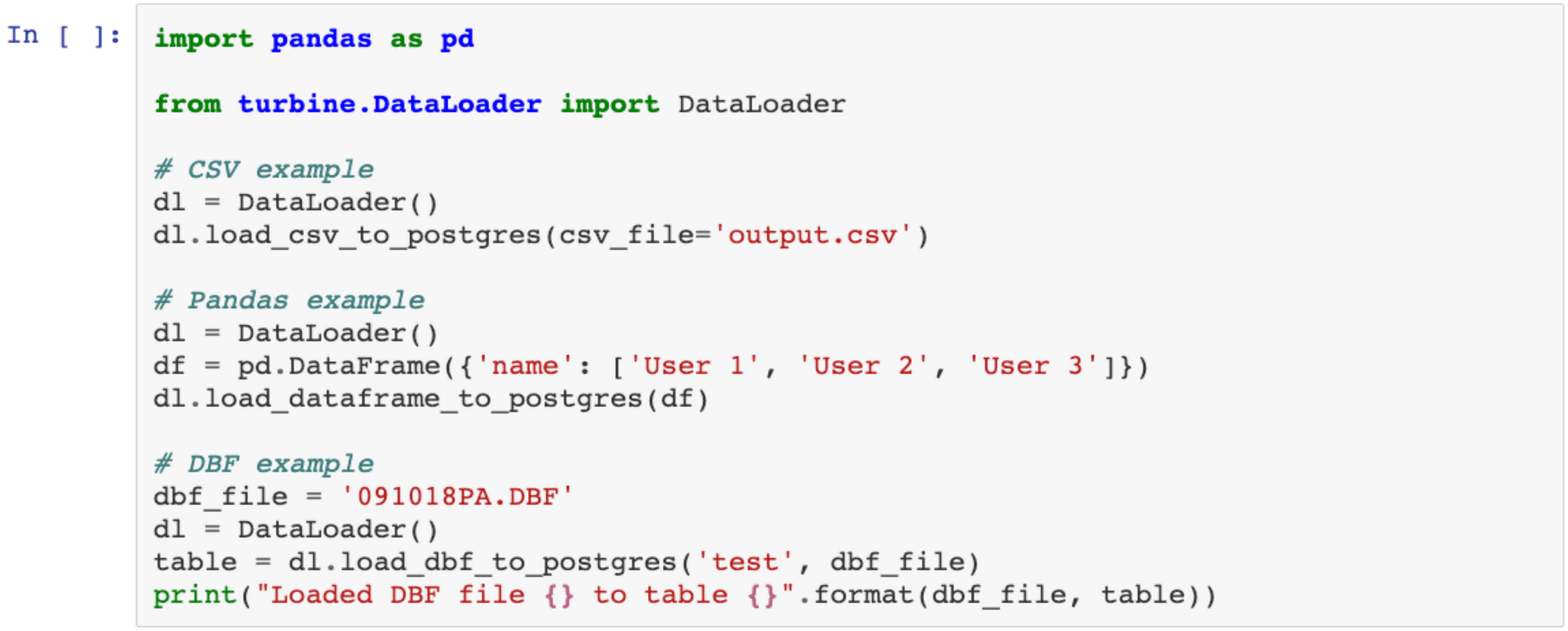
```
# Set a limit (default is 100)
simple.set_limit(10)
# Execute SQL and evaluate Pandas dataframe
simple.run()
simple.data
```

Out[36]:

	schemaname	tablename	tableowner	tablespace	hasindexes	hasrules	hastriggers	rowsecurity
0	public	fracfocus_bk	admin	None	False	False	False	False
1	public	production_all_t	admin	None	True	False	False	False
2	public	pending_production_historic_partition_t	admin	None	False	False	False	False
3	public	permit_well_status_all_partition_t	admin	None	False	False	False	False
4	public	permit_well_status_historic_partition_t	admin	None	False	False	False	False
5	public	production_historic_partition_t	admin	None	False	False	False	False
6	public	wellbore_record_historic_partition_t	admin	None	False	False	False	False
7	public	well_historic_partition_t	admin	None	False	False	False	False
8	public	w2_completions_historic_partition_t	admin	None	False	False	False	False
9	public	wellbore_record_all_partition_t	admin	None	False	False	False	False

DIRECT CSV OR DATA FRAME TO POSTGRES

- Let’s say you have a CSV data file that you’ve discovered (or a Pandas data frame that you’ve created), and you want to get it into a sandbox table to play with or share with others
- Turbine makes this easy:

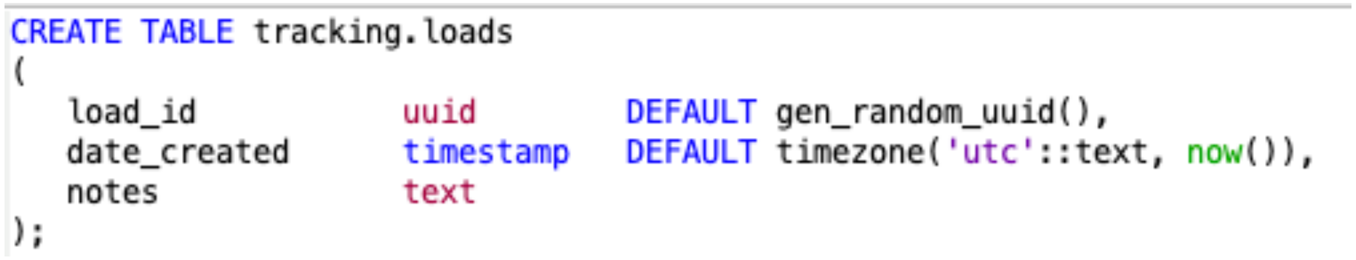


```
In [ ]: import pandas as pd

        from turbine.DataLoader import DataLoader

        # CSV example
        dl = DataLoader()
        dl.load_csv_to_postgres(csv_file='output.csv')

        # Pandas example
        dl = DataLoader()
        df = pd.DataFrame({'name': ['User 1', 'User 2', 'User 3']})
        dl.load_dataframe_to_postgres(df)

        # DBF example
        dbf_file = '091018PA.DBF'
        dl = DataLoader()
        table = dl.load_dbf_to_postgres('test', dbf_file)
        print("Loaded DBF file {} to table {}".format(dbf_file, table))
```

TRACKING LOADS

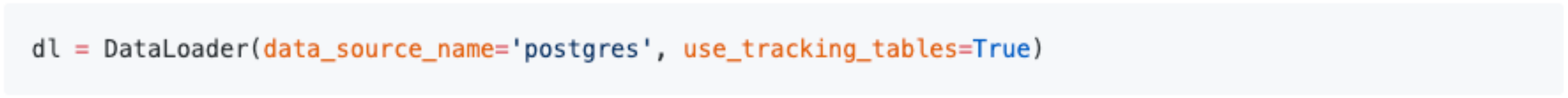
- Turbine has built-in functionality for tracking every load that you make in a table specifically designed for this purpose

- Turbine adds a ‘load_id’ field to your database tables that refers to a tracking.loads table in Postgres that tracks date/time that it was loaded and any notes you want to add



```
CREATE TABLE tracking.loads
(
   load_id            uuid         DEFAULT gen_random_uuid(),
   date_created       timestamp    DEFAULT timezone('utc'::text, now()),
   notes              text
);
```

- To enable this functionality, simply pass the parameter use_tracking_tables:



```
dl = DataLoader(data_source_name='postgres', use_tracking_tables=True)
```

- Then load your CSV or data frame and it will be automatically tracked

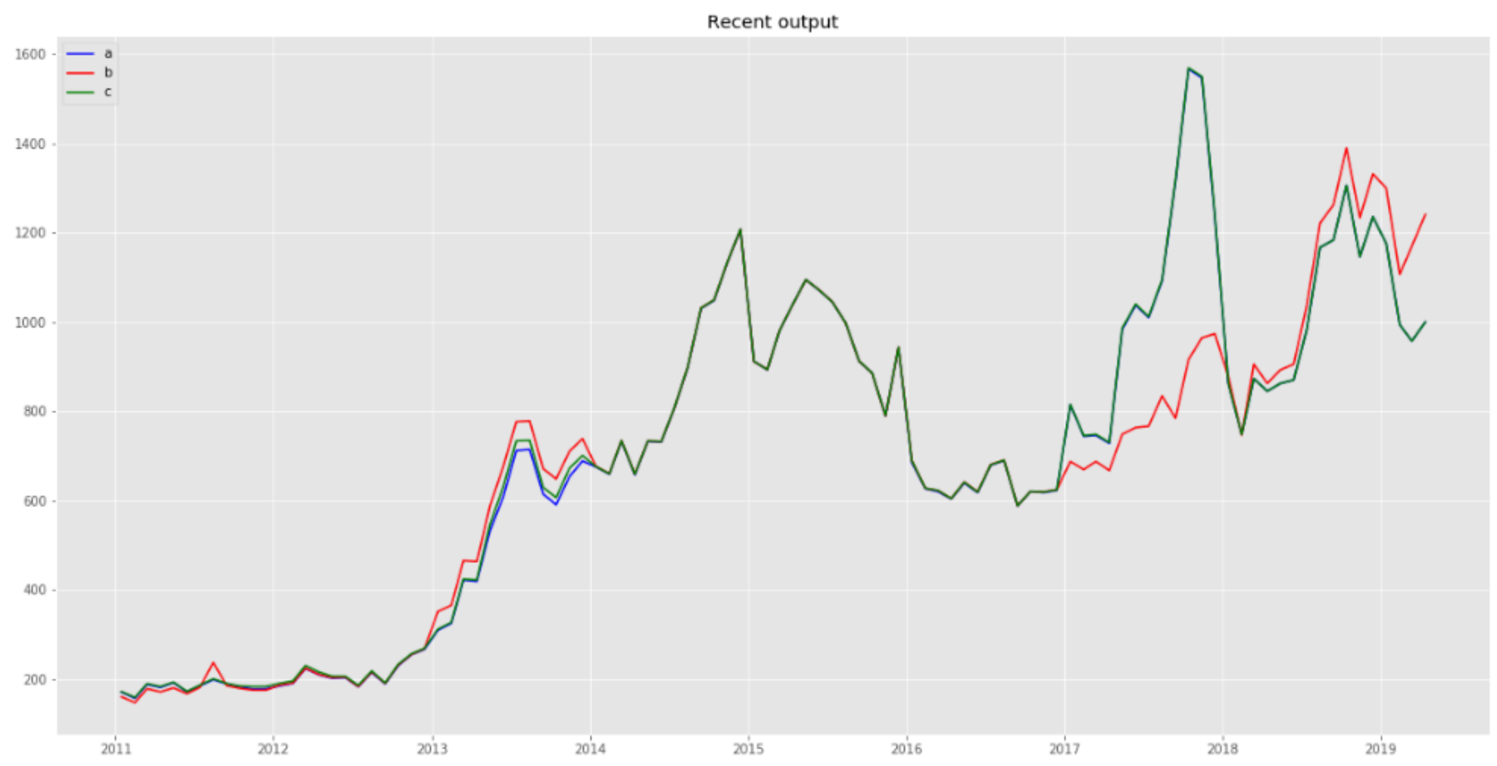
INSTALLATION

- Install python 3.6+ and Jupyter
  - Anaconda Python is the easiest method
- Install Turbine
  - Option A: install from github directly
    - pip install git+https://github.com/bazean/public-turbine.git
  - Option B: Clone the git repo and install the package
    - git clone https://github.com/bazean/public-turbine
    - pip install dist/turbine-1.3-py2.py3-none-any.whl
- Copy config.json.template to config.json and set your own database user and password
- Copy turbine/SQL/Example.sql to make your own SQL file for retrieving data

TURBINE USE AT BAZEAN

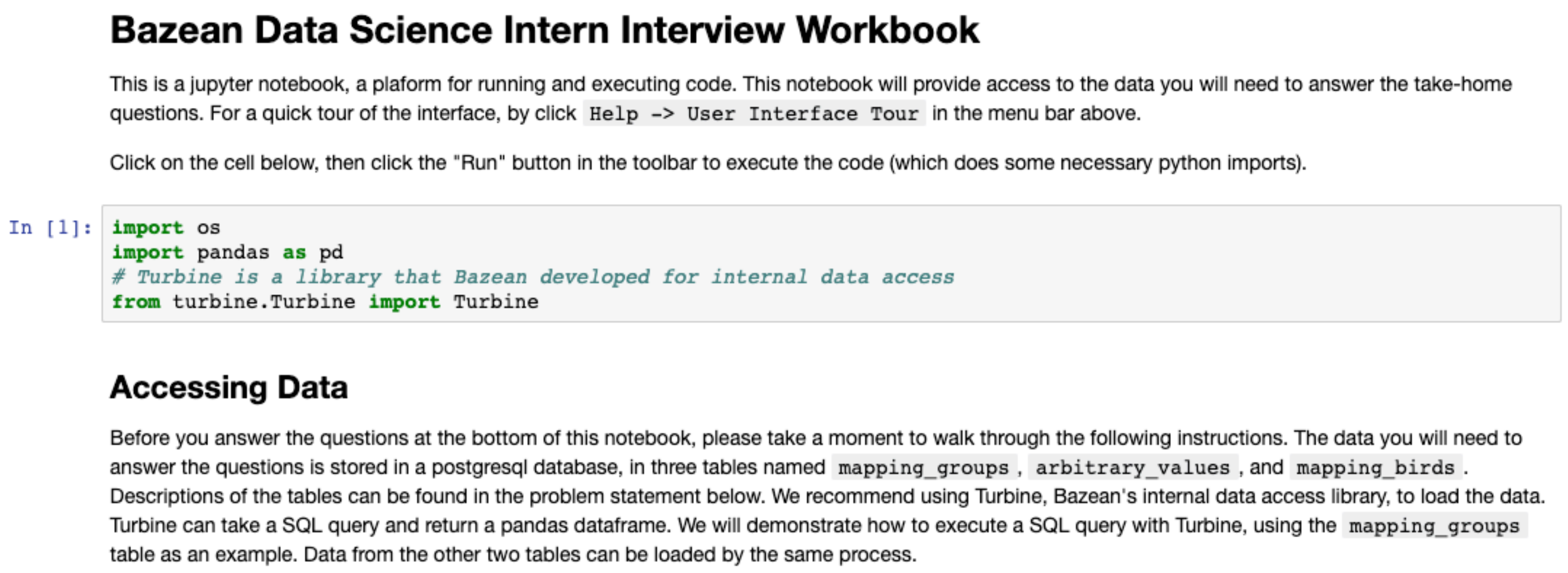
- Turbine is used as an everyday tool to perform analysis on our data
- It allows us to visually compare multiple sets of data from different Postgres databases on different cloud providers and other external sources including CSV/XLS/DBF data, and to gain insights into the data

```
Out[36]: <matplotlib.legend.Legend at 0x7fdf5b334400>
```



TURBINE USE FOR RECRUITMENT

- At Bazean, we also use Turbine for take home assessments for our summer interns which really allows us to see how candidates handle the kind of data that they'll be working with during their internship



Bazean Data Science Intern Interview Workbook

This is a jupyter notebook, a plaform for running and executing code. This notebook will provide access to the data you will need to answer the take-home questions. For a quick tour of the interface, by click `Help -> User Interface Tour` in the menu bar above.

Click on the cell below, then click the "Run" button in the toolbar to execute the code (which does some necessary python imports).

In [1]:
```python
import os
import pandas as pd
# Turbine is a library that Bazean developed for internal data access
from turbine.Turbine import Turbine
```

Accessing Data

Before you answer the questions at the bottom of this notebook, please take a moment to walk through the following instructions. The data you will need to answer the questions is stored in a postgresql database, in three tables named `mapping_groups`, `arbitrary_values`, and `mapping_birds`. Descriptions of the tables can be found in the problem statement below. We recommend using Turbine, Bazean's internal data access library, to load the data. Turbine can take a SQL query and return a pandas dataframe. We will demonstrate how to execute a SQL query with Turbine, using the `mapping_groups` table as an example. Data from the other two tables can be loaded by the same process.

TURBINE OPEN SOURCE LINKS

These will be publicly available by Feb 2020

- Bazean.com/open-source
  - Contact us at opensource@Bazean.com
- Github.com/Bazean/public-turbine
  - We’re always open to hearing your ideas for improving Turbine
  - Or you can just submit pull requests for fixes/improvements

KEY TAKEAWAYS

- Turbine provides easy database access capabilities for data scientists and analysts to be able to merge SQL with Pandas to create visualizations and run experimental models on the data
- It allows users to be able to analyze and compare data from multiple different sources including different databases, which could reside on different cloud providers, and CSV/XLS/DBF data
- Once data has been consolidated and any required manipulation has been performed it is simple to load that data into your Postgres database with tracking to enable you to see when each version of the data was loaded