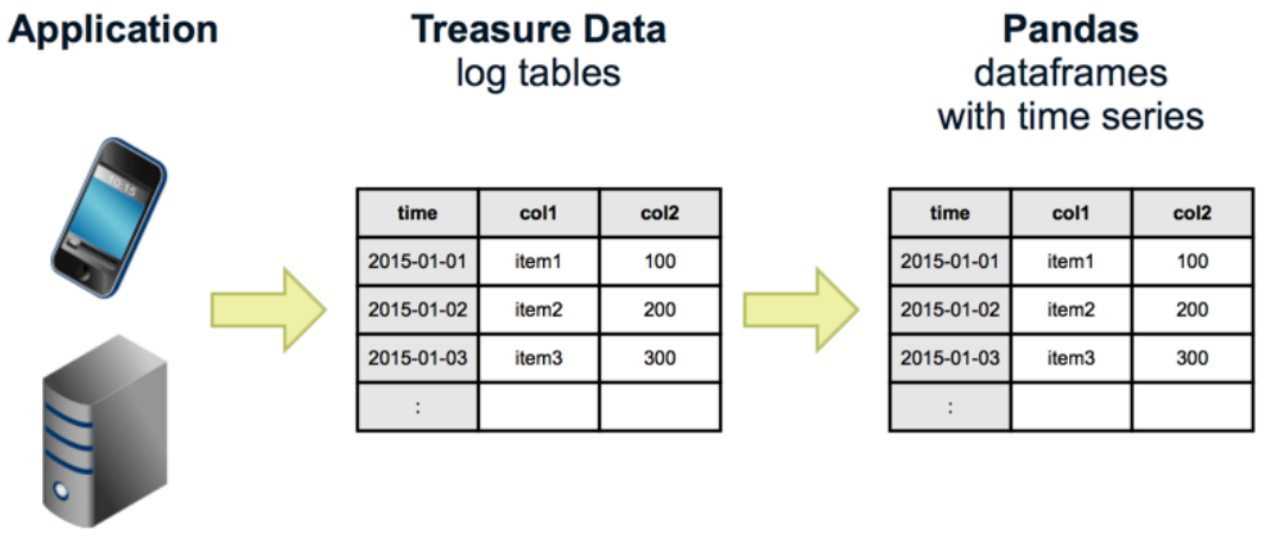
Pandas and Jupyter Configuration for Treasure Data

Treasure Data provides a cloud-based analytics infrastructure accessible via SQL. Interactive engines like Presto enable you to crunch billions of records easily. However, writing a SQL query is sometimes painful for data scientists, and you'll still need to use external tools like Excel or Tableau to visualize the result. You can use Treasure Data with the Python-based data analysis tool called Pandas and visualize the data interactively via Jupyter Notebook.



Prerequisites

- Basic knowledge of Python.
- Basic knowledge of Treasure Data.

Set Treasure Data API Key

Set your master API key as an environment variable before launching Jupyter. The master API KEY can be retrieved from the TD Console profile.

```
$ export TD_API_KEY="1234/abcde..."
```

You can set your environment variables with a command such as the following in Jupyter Notebook cell.

```
%env TD_API_KEY = "123c/abcdefghjk..."
```

Set Treasure Data API Endpoint

Set Treasure Data API Endpoint as an environment variable if your account does not belong to the US Region. You can see Endpoint info here

```
$ export TD_API_SERVER="https://api.treasuredata.co.jp"
```

You can set your environment variables with a command such as the following in Jupyter Notebook cell.

```
%env TD_API_SERVER = "https://api.treasuredata.co.jp"
```

Install the Necessary Packages and Configure your Environment

For more information and instructions, see Installing Conda, Pandas, matplotlib, Jupyter Notebook, and pytd.

Run Jupyter and Create your First Notebook

We'll use Jupyter as a frontend for our analysis project.

1. Run Notebook using the following syntax:

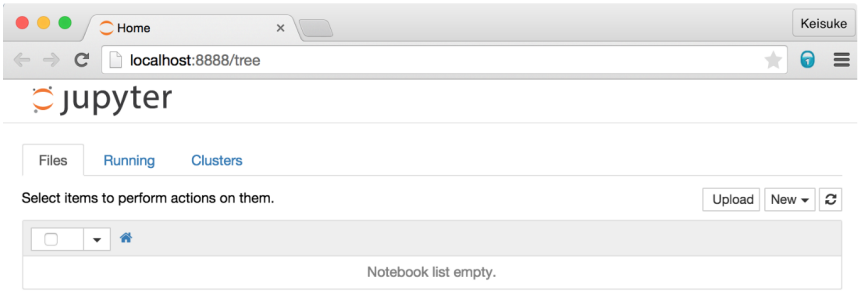
   ```
   (analysis)$ ipython notebook
   ```

2. Your web browser will open:
3. Select **New** > **Python 3**.
4. Copy and paste the following text into your notebook:

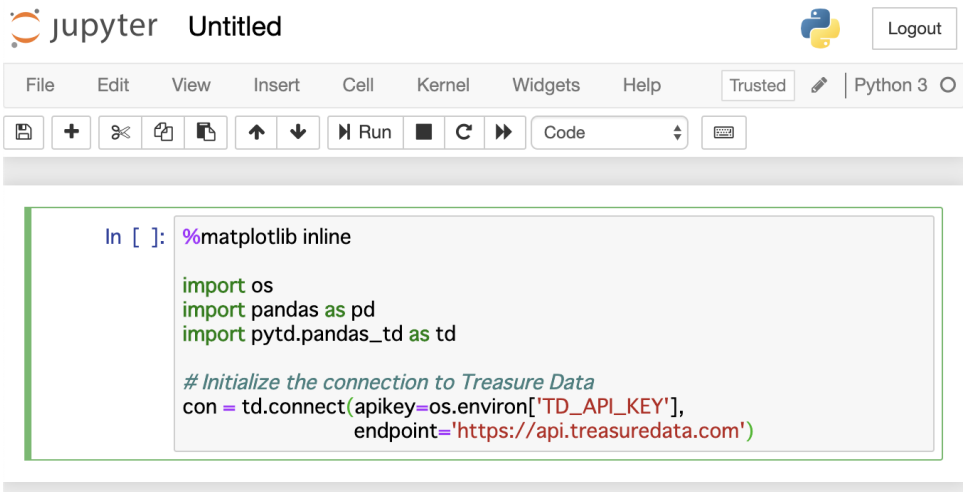
   ```
   %matplotlib inline

   import os
   import pandas as pd
   import pytd.pandas_td as td
   # Initialize the connection to Treasure Data

   con = td.connect(apikey=os.environ['TD_API_KEY'], endpoint='https://api.treasuredata.com')
   ```



5. Your notebook should now look similar to



6. Type Shift-Enter.
   If you get "KeyError: 'TD_API_KEY'" error, **try "apikey='<your master apikey>'"** instead of "apikey=os.environ['TD_API_KEY']".
   If it works, Jupyter didn't recognize the TD_API_KEY variable from the OS.
   Confirm the TD_API_KEY again and re-launch Jupyter.
7. Optionally, save your notebook.

Explore Data

There are two tables in `sample_datasets`. You can use the magic command `td_tables` to view all the tables in your database.

```
In [42]: %td_tables sample_datasets
```

Out[42]:

	db_name	name	count	estimated_storage_size	last_log_timestamp	created_at
0	sample_datasets	nasdaq	8807279	133215435	1978-01-03 16:00:00+00:00	2014-10-08 02:57:38+00:00
1	sample_datasets	www_access	5000	114834	2014-10-04 01:13:15+00:00	2014-10-04 01:13:12+00:00

Let's explore the `nasdaq` table.

In Jupyter, type the following syntax:

```
engine = td.create_engine("presto:sample_datasets")
client =td.Client(database='sample_datasets')
client.query('select symbol, count(1) as cnt from nasdaq group by 1 order by 1')
```

For example:

```
In [2]: engine = td.create_engine("presto:sample_datasets")

In [7]: client =td.Client(database='sample_datasets')

In [9]: client.query('select symbol, count(1) as cnt from nasdaq group by 1 order by 1')

Out[9]: {'data': [['AAIT', 590],
                  ['AAL', 82],
                  ['AAME', 9252],
                  ['AAOI', 253],
                  ['AAON', 5980],
                  ['AAPL', 8522],
                  ['AAVL', 41],
                  ['AAWW', 2428],
                  ['AAXJ', 1539],
                  ['ABAC', 1052],
                  ['ABAX', 5712],
                  ['ABCB', 5125],
                  ['ABCD', 1194],
                  ['ABCO', 3239],
                  ['ABDC', 98],
                  ['ABGB', 237],
                  ['ABIO', 1426],
                  ['ABMD', 6846],
                  ['ABTL', 3899],
```

Running a Query in Jupyter

For the purposes of this example, Presto is used as the query engine for this session.

In Jupyter, type the following syntax:

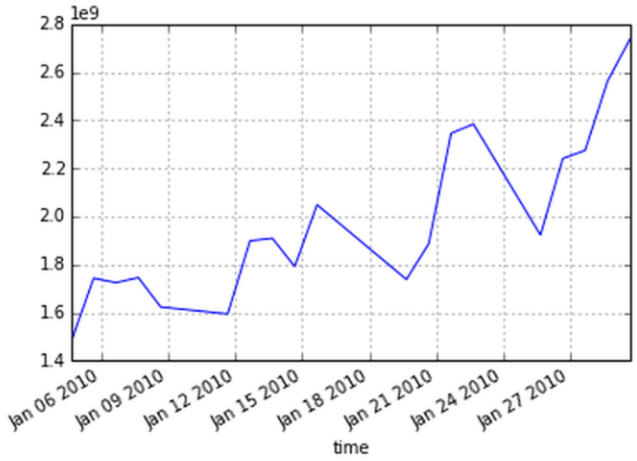
```
import pytd.pandas_td as td
con = td.connect(apikey=apikey, endpoint="https://api.treasuredata.com")
engine = td.create_engine("presto:sample_datasets")
td.read_td_query(query, engine, index_col=None, parse_dates=None, distributed_join=False, params=None)
```

You can also use the `time_range` parameter to retrieve data within a specific time range:

```
In [4]: # Get all rows from 2010-01-01 to 2010-02-01
df = td.read_td_table('nasdaq', engine, limit=None,
                      time_range=('2010-01-01', '2010-02-01'),
                      index_col='time', parse_dates={'time': 's'})

# Plot the sum of volumes, grouped by time (= index level 0)
df.groupby(level=0).volume.sum().plot()
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x10a908ac8>
```



Your data is stored in the local variable `df` as a DataFrame. Because the data is located in the local memory of your computer, you can analyze it interactively using the power of Pandas and Jupyter. See Time Series / Date functionality for the details of time-series data.

Sample Data

As your data set grows very large, the method from the previous step doesn't scale very well. We don't recommend that you retrieve more than a few million rows at a time due to memory limitations or slow network transfer. If you're analyzing a large amount of data, you need to limit the amount of data getting transferred.

There are two ways to do this:

- You can sample data. For example, the "Nasdaq" table has 8,807,278 rows. Setting a limit of 100000 results in 100,000 rows, which is a reasonable size to retrieve:

```
In [12]: df = td.read_td_table('nasdaq',engine,limit=100000,
                               index_col='time',parse_dates={'time':'s'})
         len(df)
```

```
Out[12]: 100000
```

- Write SQL and limit data from the server-side. For example, as we are interested only in data related to "AAPL", let's count the number of records, using `read_td_query`:

```
In [7]: td.read_td_query('''
        select count(*) from nasdaq where symbol='AAPL'
        ''', engine)
```

Out[7]:

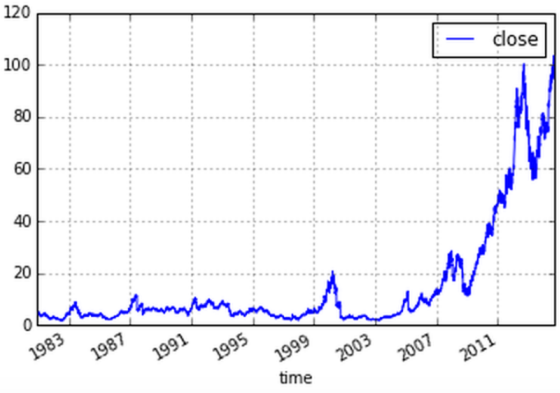
	_col0
0	8522

It's small enough, so we can retrieve all the rows and start analyzing data:

In [8]:
```
df = td.read_td_query('''
select time, close from nasdaq where symbol='AAPL'
''', engine, index_col='time', parse_dates={'time': 's'})

df.plot()
```

Out[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x107557438>
```



See the contents below for further information.

- Python for Data Analysis (Book by O'Reilly Media)

Jupyter Notebooks are supported by GitHub and you can share the result of your analysis session with your team:

- GitHub + Jupyter Notebooks = <3