Empowering Quants to Trade Faster

Aneesh Karve, CTO at Quilt Data
Trading firms say they're data driven...
Miss the data, Miss the trade, Miss the profit
Data registry integrates data sources
Data should be managed like source code
GFS Bank

- Fortune 500
- Global banking & financial services
- Over 1.5 Trillion USD assets under management
Data pipeline at GFS

- Parent bank **purchases** data assets as indicators
- **Distributes** assets to 20 Quants across 8 subsidiaries
- Data assets feed **predictive models**
Before: life of an Excel file

<table>
<thead>
<tr>
<th>Activity</th>
<th>Risks and challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upload to blob storage (S3, Azure)</td>
<td>Manual, links break, files corrupt</td>
</tr>
<tr>
<td>Post links to files on wiki</td>
<td>Manual, no versioning, no access control, no logging</td>
</tr>
<tr>
<td>Quants use different APIs to access data (Python, R, Excel)</td>
<td>Manual, time-consuming, focus on data prep</td>
</tr>
</tbody>
</table>
Files

In [1]:
import glob
import os
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Extract, transform, and load

List input files

Each file is the result of an experiment to measure p-values for genes based on CRISPR knock-out experiments. The data in question has been generated to protect the identity of the experimenters from being revealed.

In [2]:
# load .txt files

Determine format of .txt files

In [3]:
# load .xlsx files

Convert TSV files to dataframes

In [4]:
def read_tsv(filename):
    df = pd.read_csv(filename, sep='	')
    return df

CPU times: user 229 ms, sys: 105 ms, total: 334 ms
Wall time: 448 ms

Convert XLSX files to dataframes

In [5]:
def read_excel(filename):
    df = pd.read_excel(filename)
    return df

CPU times: user 347 ms, sys: 121 ms, total: 468 ms
Wall time: 468 ms

Analysis

Join all of the experimental data on a common key (in this case 'gene_id').

In [6]:
# Join data across all dataframes on a common key

Packages

Before you start

In [7]:
# pip install quilt

# pip install matplotlib

import quilt

Import packages

In [8]:
# time from quilt-data-shares -h

Analysis

Join the data across all dataframes on a common key.

In [9]:
# def merge_frames(df):

Visualize

In [10]:
# if data is numeric data, we show it just as a solid block of data

plt.scatter(sample, sample, color='blue')
Data science is only 21% science

- It's 79% data prep (find, clean, organize data)
- Source
Files are a drag

Now what?
Package = virtualized data + meta-data
Files

```python
In [3]:
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Extract, transform, and load

# List input files

# Determine format of .txt files

# Convert TSV files to dataframes

# Convert XLSX files to dataframes

Analysis

Join all of the experimental data in a common key (in this case gene id).

```
Old way (manual)

- Purchase asset files
- Upload files to blob storage
- Post links to storage on wiki
- Quants download files
- Write data prep code for different platforms
- Load data into code
- Forecast
- Trade

New way (auto)

- Build package
- Push package
- Install package
- Import package
- Model
- Trade

Data registry
Data registry

Missing in old way

- Discoverability
- Reproducibility
- Auditability
- Security
- Compliance

Automatic in new way

- Discoverability
- Reproducibility
- Auditability
- Security
- Compliance
Data science with packages is **80% science**

- 75% less data prep
- 10X faster I/O
- Trade faster
"Packages are a seamless way to make Quants successful."
## Objections

<table>
<thead>
<tr>
<th>Objection</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Don't want to change existing data infrastructure</td>
<td>Data packages are compatible + additive</td>
</tr>
<tr>
<td>No time for long integration cycles</td>
<td>Can quickly integrate one analyst at time</td>
</tr>
<tr>
<td>Don't want to teach new skills or systems</td>
<td>&quot;Manage data like code&quot; means commands already known from git, Docker</td>
</tr>
</tbody>
</table>
Surprises

- Data engineers are the champions
- Users document data if others are watching
- Data can be managed like code
Packages standardize data
$ quilt build usr/pkg

Compute hash version, parse files into data frames, serialize data frames, and create package.

$ quilt push usr/pkg

Upload package to central registry.

$ quilt install usr/pkg

Download the package.

from quilt.data.usr import pkg

Lazily load data frames into memory.
Demo
Architecture

Registry
- Meta-data, storage, permissions
- Flask, MySQL, S3

Catalog
- Browse, search via web UI
- JavaScript

Data compiler
- Build packages
- Python, R (future)
Data compiler

- **Builds** packages
- **Checks** for compliance
- **Indexes** for discoverability
- [https://github.com/quiltdata/quilt-compiler](https://github.com/quiltdata/quilt-compiler)
Data registry

- Inspired by **Docker**
- **Additive** to existing infrastructure
- **Downstream** of databases, data warehouses
- Ideal **endpoint for ETL pipeline**
- Manages **security and compliance**
- **De-duplicates** data
- [https://github.com/quiltdata/quilt-registry](https://github.com/quiltdata/quilt-registry)
Data catalog

- Search
- Browse
- Docs
How fast can you get data into code?
It doesn't matter how much data you collect, it matters how much data you leverage.
Maximize return on data

- Discoverable
- Reproducible
- Auditable
- Compliant
- Secure
Manage **data like code**

<table>
<thead>
<tr>
<th>Return on data</th>
<th>Code-style service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discoverability</td>
<td>Cataloging, compilation</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>Packaging, versioning</td>
</tr>
<tr>
<td>Auditability</td>
<td>Packaging, logging</td>
</tr>
<tr>
<td>Security</td>
<td>Registry as hub for data</td>
</tr>
<tr>
<td>Compliance</td>
<td>Compilation, linting, checks</td>
</tr>
</tbody>
</table>
Stay in touch

@akarve
aneesh@quiltdata.io

https://quiltdata.com/docs/quants.pdf