Technology Training Session 1

Databricks AWS:
Data Wrangling with Python and Spark

November 2019

This training will be successful if…

- Python: move from academic to business/practical
  - Monetize python in production environment
  - Sample codes and best practice tactics for marketing efforts
- Data Management: move to AWS, file processing, data engineering best practice, in ETL software or in native

Instructor: Matthew DePoint, PhD

linkedin.com/in/msdepoint

- 15+ years hands-on and director-level experience in marketing science working for Target, Intuit, HSBC, USAA, Banner Health etc…
- Learned the craft in financial services marketing analytics, campaign execution and credit risk (SAS and SQL). Applied those skills in other industries.
- Updated skills in the past 10 years using modern tools, technologies and approaches, prompted by my work with a Silicon Valley based firm.
- PhD Economics, adjunct Professor of Economics and Data Science.

Today's Agenda

- Industry Trends Data Analytics
- Database Architecture for Data Analysis
- Databricks Overview
- Data Wrangling, Profiling, and Visualization Basics in Databricks
- Takeaways from Q&A and Future Knowledge Share

Common Trends across Industries

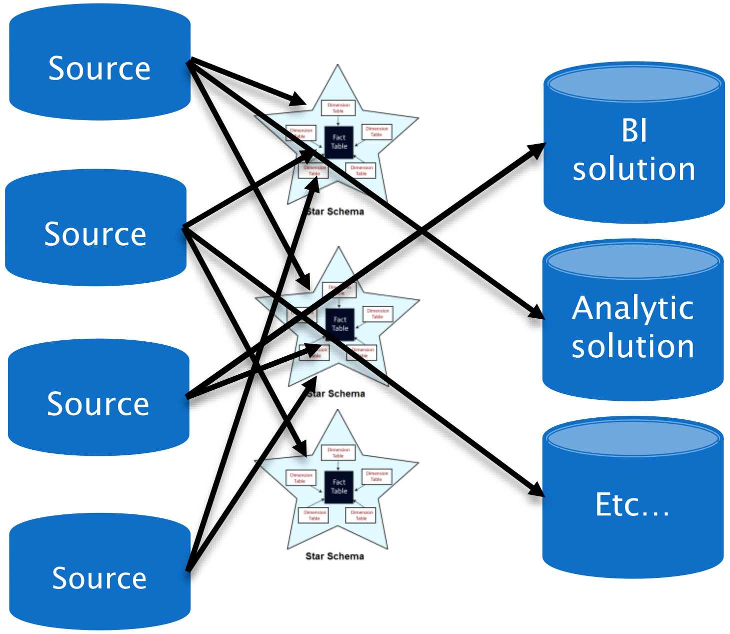
Trend	Tools/concepts that exploit this trend
Social > Email	Stackoverflow, github
Collaborative > Siloed	PopSQL, G Suite, Evernote, Slack, Teams, GitHub, GoToMeeting
Agile > Waterfall	Change is constant
Buy > Build	SaaS, PaaS,NaaS, RaaS …. Salesforce,
Cloud > in-house	AWS
Open Source > Proprietary	Linux, R, python, Hadoop, mySQL
Data Science > Analytics	Machine learning
Schema on-read > Schema on-write	HDFS, Splunk, Databrick,
Petabytes > millions of records	Big data solutions

Tradition SAS Shop → Today's Architecture

Same	Different
Messy Data	Default access to data is YES but versus "why do you need that?"
Pulling from multiple sources	Security built in at the table or row level
No/incomplete data dictionaries	More source data dumps
	Social / collaboration code
	Late "data binding"

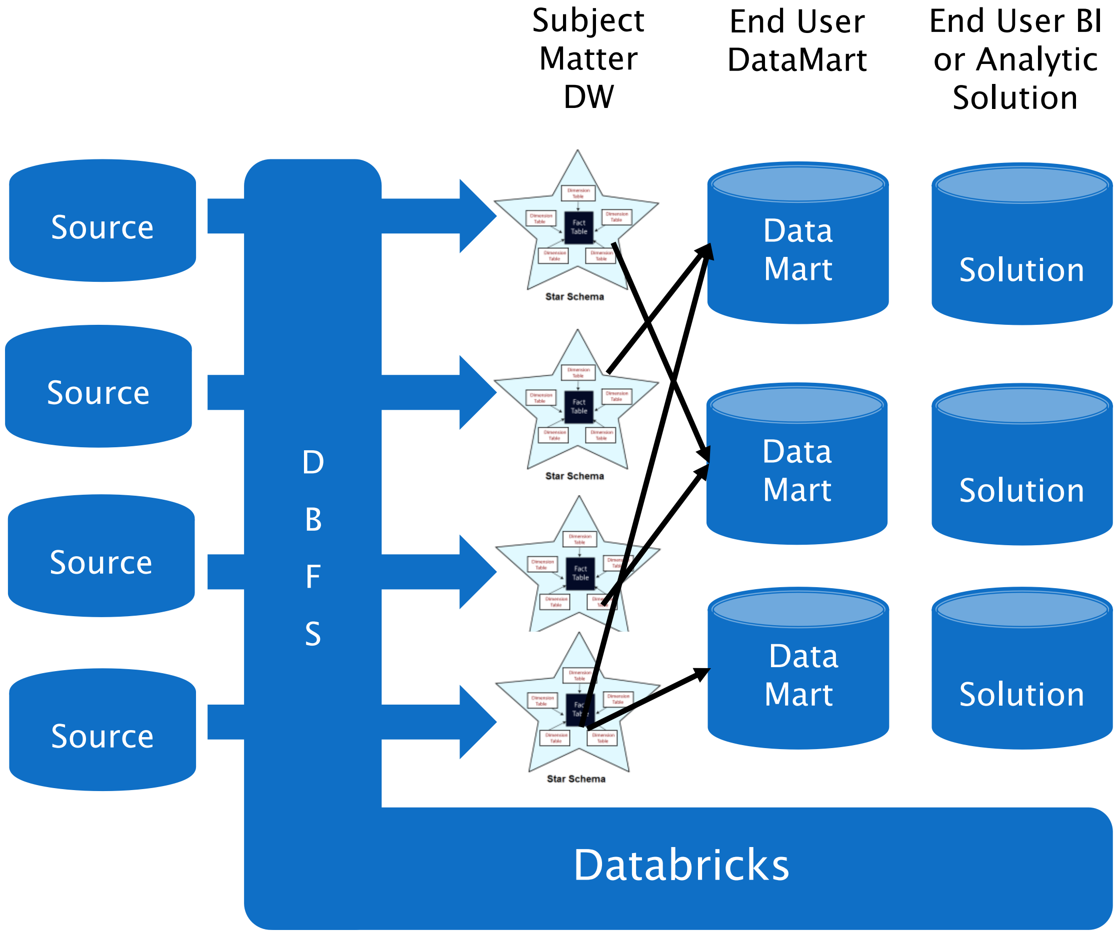
I am not an architect, but I did stay at Holiday Inn Express

Old Way



Multiple pulls against sources, incomplete warehouses and end user solutions.

With New Data Architecture

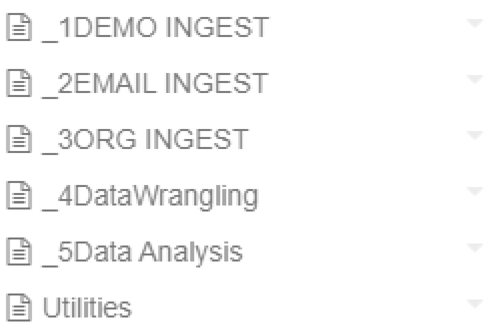


Efficient pulls, storage, and data models with late binding of end user solutions. Huge Sand box for analytical development

Deep Dive Topics

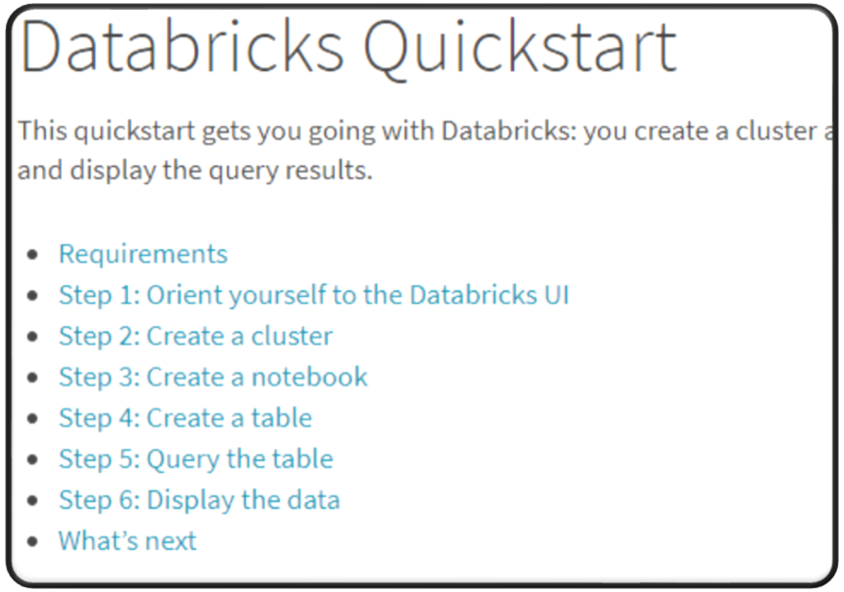
- Data Integration – data engineering light
  - Ingest modest sized data from various data sources
- Data Wrangling
  - Setup permanent model dataset with pre-created variables
  - Example creation of simple and complex aggregate variables
  - What tools are the best? Python, SQL, Spark, etc
- Data Analysis
  - Pre-model profile analysis
  - Visualization
- Best Practices
  - Efficient code
  - Model related data work -mostly covered in next sessions
  - Model production – mostly covered in next sessions
- OUT OF SCOPE – Data Engineering
  - Develop and maintain data architectures aligned with business needs
  - Identify data sources and productionalize ingestion at scale
  - Security and access

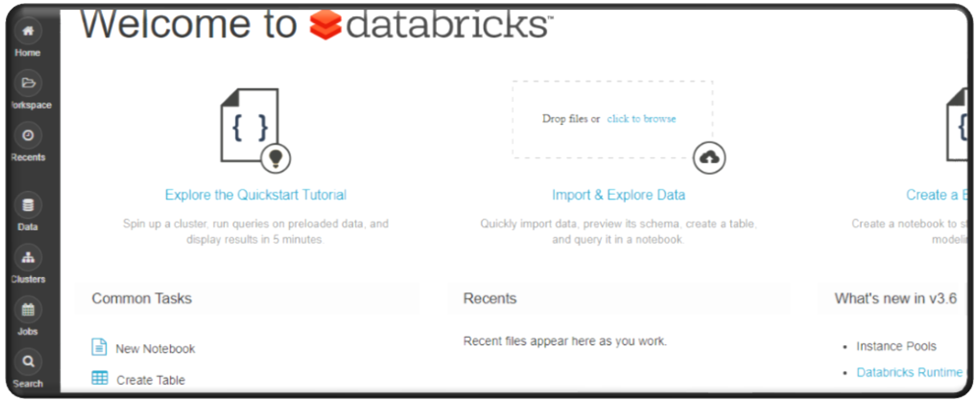
Databricks



AWS/Databricks Overview

- Getting started: Databricks from Apache Spark
- Documentation: https://docs.databricks.com/index.html
- Databricks File System
- Personal login, free version of Databricks: https://databricks.com/try-databricks





Feels like Jupyter notebook/ipyhton

- Visualize your code and output together
- Interactive mode

Data Imports / Ingestion

- Pull from raw sources (schema on the fly)
  - Ingest Company Sample files
  - CSV ingestion options
- Pull from modeled data (SQL in python) OUT OF SCOPE
- Pull from real time data sources like twitter using an API call
  - Need external SQL environment
- Pull from RedShift OUT OF SCOPE
  - Need external RedShift environment

Uploading to /FileStore

- Utilize Drag and Drop functionality. UI or Notebook
  https://docs.databricks.com/data/tables.html

```
# File location and type
file_location = "/FileStore/tables/Demo_training.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

display(df)
```

(3) Spark Jobs

df: pyspark.sql.dataframe.DataFrame

```
ID: integer
monthid: integer
household_income: string
age_agg_ind: integer
sec_age_agg_act: integer
```

Write permanent table

```
df.write.format("parquet").saveAsTable(permanent_table_name)
```

(1) Spark Jobs

Command took 3.83 seconds -- by sdepoint@gmail.com at 11/14/2019, 7:10:38 PM on TestCluster1

Data Wrangling

- Cleansing and data prep
  - Spark DataFrame and SparkSQL or Pandas DataFrame?
- Data transformations
  - Apply "formats" using functions and dictionary mappings or Lamba functions
  - Datetime manipulate in Pandas, formats and date differences
  - Simple string parsing
- Conversion to and from SparkDF and Pandas DF
- Merge dataframes together to create analysis table

DateTime efficiencies

- Declare the format if you know it!

```
#cleanup the date time variables in Pandas
EMPDF['sentDate'] = EMPDF['sent_ts'].apply(pd.to_datetime)
EMPDF['openDate'] = EMPDF['open_ts'].apply(pd.to_datetime)
EMPDF['clickDate'] = EMPDF['click_ts'].apply(pd.to_datetime)
#FYI this conversion to datetime will be a lot quicker if I tell pandas the da
EMPDF['sent2openDiff'] = (EMPDF['openDate']- EMPDF['sentDate']).dt.seconds
EMPDF['click2openDiff'] = (EMPDF['clickDate']- EMPDF['sentDate']).dt.seconds
# EMPDF[:15]

ommand took 42.66 seconds -- by sdepoint@gmail.com at 11/15/2019, 6:22:59 AM on TestCluster1
```

```
#cleanup the date time variables in Pandas
EMPDF['sentDate'] = EMPDF['sent_ts'].apply(pd.to_datetime,format='%m/%d/%Y %H:%M')
EMPDF['openDate'] = EMPDF['open_ts'].apply(pd.to_datetime,format='%m/%d/%Y %H:%M')
EMPDF['clickDate'] = EMPDF['click_ts'].apply(pd.to_datetime,format='%m/%d/%Y %H:%M')
#FYI this conversion to datetime will be a lot quicker if I tell pandas the datetime format
EMPDF['sent2openDiff'] = (EMPDF['openDate']- EMPDF['sentDate']).dt.seconds
EMPDF['click2openDiff'] = (EMPDF['clickDate']- EMPDF['sentDate']).dt.seconds
# EMPDF[:15]

mmand took 22.79 seconds -- by sdepoint@gmail.com at 11/15/2019, 6:24:47 AM on TestCluster1
```

Data Analysis & Visualization

- Profile your data
  - Income Distribution
  - Time to click distribution
  - Click-through rate
- Count, Sum, Mean, median, std of variables
- Time series plots
- Basic in-line plotting and visualization so you can drop into PPT using DataBricks

Visualization w/ Databricks SparkSQL

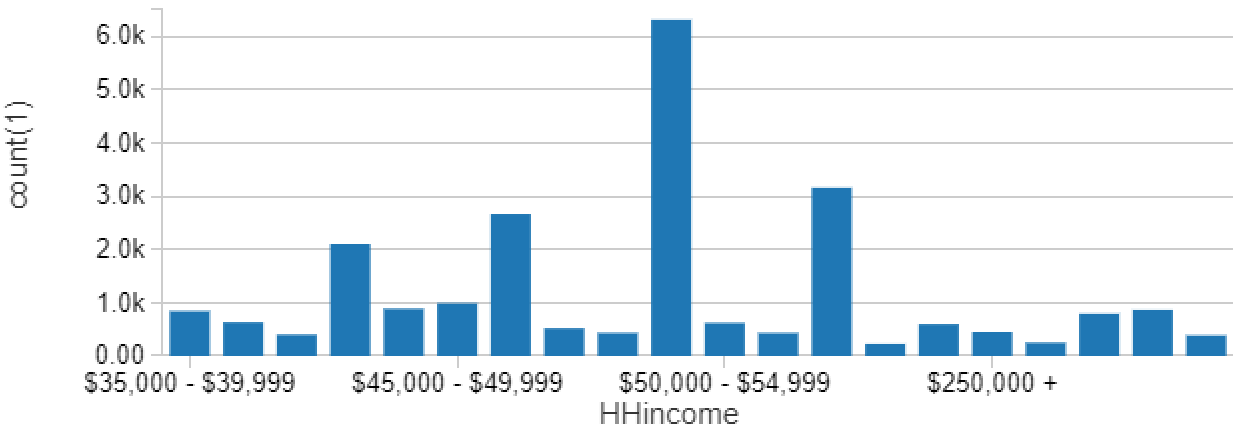
```
%sql
SELECT HHincome, count(*) FROM EM_DEMO_sDF group by HHi
```

(5) Spark Jobs

HHincome	count(1)
nan	6317
$75,000 - $99,999	3162
$100,000 - $149,999	2667
$65,000 - $74,999	2104
$45,000 - $49,999	991
$55,000 - $59,999	889
$60,000 - $64,999	867
$35,000 - $39,999	844

```
%sql
SELECT HHincome, count(*) FROM EM_DEMO_sDF group by HHincome
```

(5) Spark Jobs



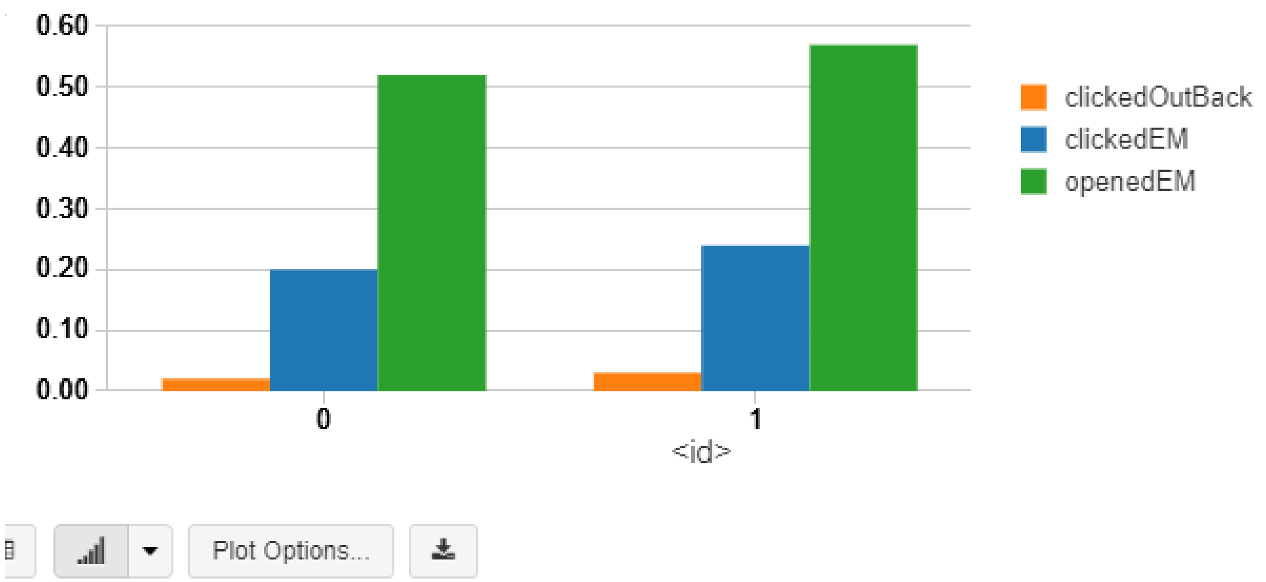
Plot Options...

Command took 3.04 seconds -- by sdepoint@gmail.com at 11/15/2019, 10:06:28 AM on TestCluster1

Spark SQL

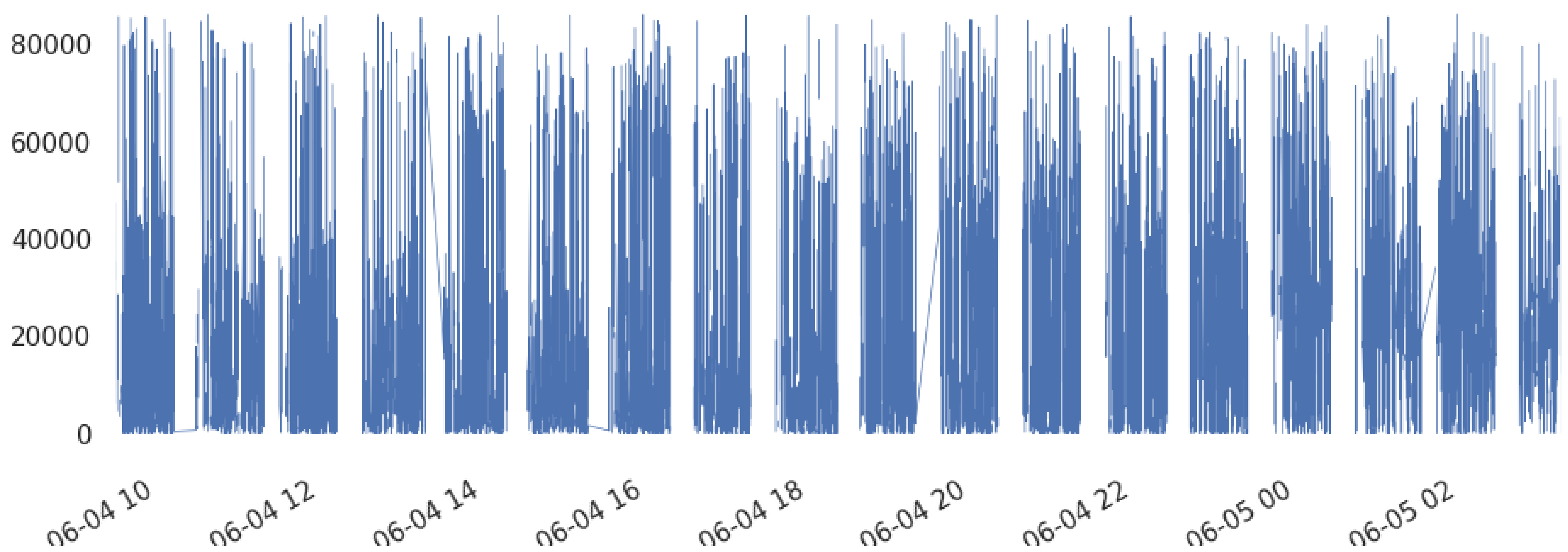
```
%sql
SELECT oldPeople
  ,CAST(AVG(openedEM) AS decimal(12,2)) AS openedEM
  ,CAST(AVG(clickedEM) AS decimal(12,2)) AS clickedEM
  ,CAST(AVG(clickedOutback) AS decimal(12,2)) as clickedOutBack
FROM EM_DEMO_sDF
GROUP BY oldPeople
```

(5) Spark Jobs



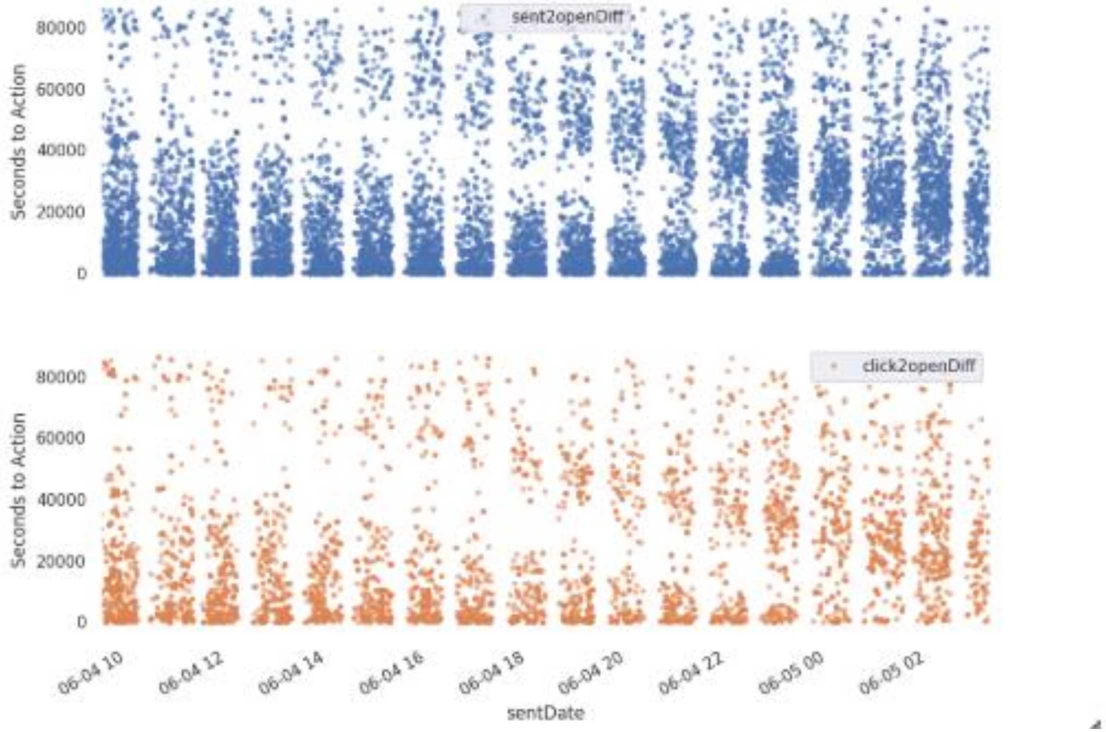
Time series displays with pandas / seaborn

```
#time series displays
# Use seaborn style defaults and set the default figure size
sns.set(rc={'figure.figsize':(11, 4)})
ts['sent2openDiff'].plot(linewidth=0.5);
display()
```



Time series displays with pandas / seaborn

```python
cols_plot = ['sent2openDiff', 'click2openDiff']
axes = ts[cols_plot].plot(marker='.', alpha=0.5, linestyle='None', figsize=(11, 9), subplots=True)
for ax in axes:
    ax.set_ylabel('Seconds to Action')
display()
```



Best practices

- Processing efficiency is directly tied to your environment and data structures.
  - Try to stay in one framework (i.e. Pandas) then optimize (rewrite for Spark)
  - Spark SQL queries utilize Hive efficiencies. Conversion to dataframe brings that data into memory
- You still need to know command line for cleaning up the HIVE files in databricks. It still is buggy
  - CLI – command line interface

APPENDIX

- Resources
  - Python for Data Analysis https://github.com/wesm/pydata-book
  - Great cartoons http://hadoopknowledgebasebyabhi.blogspot.com/2015/03/hdfs-through-cartoon.html
  - https://ipython.org/
- Training
  - Data Camp
  - Coursera
  - codeacademy

Distributed File Systems