UC San Diego
JACOBS SCHOOL OF ENGINEERING

Recommendation Predictions Data Exploration

Final Presentation

Group Members:
Lauren Coden
Jared Goldsmith
Jaime Ryan
Pradeep Saddi
Laura Wilke

December 8, 2017

Data Exploration Objective

Issue: In order to increase sales, our retailer would like to provide customized recommendations to its customers, suggesting other items they may enjoy based on past behavior, demographics, etc. The overall goal is to create a recommendation engine from existing order data.

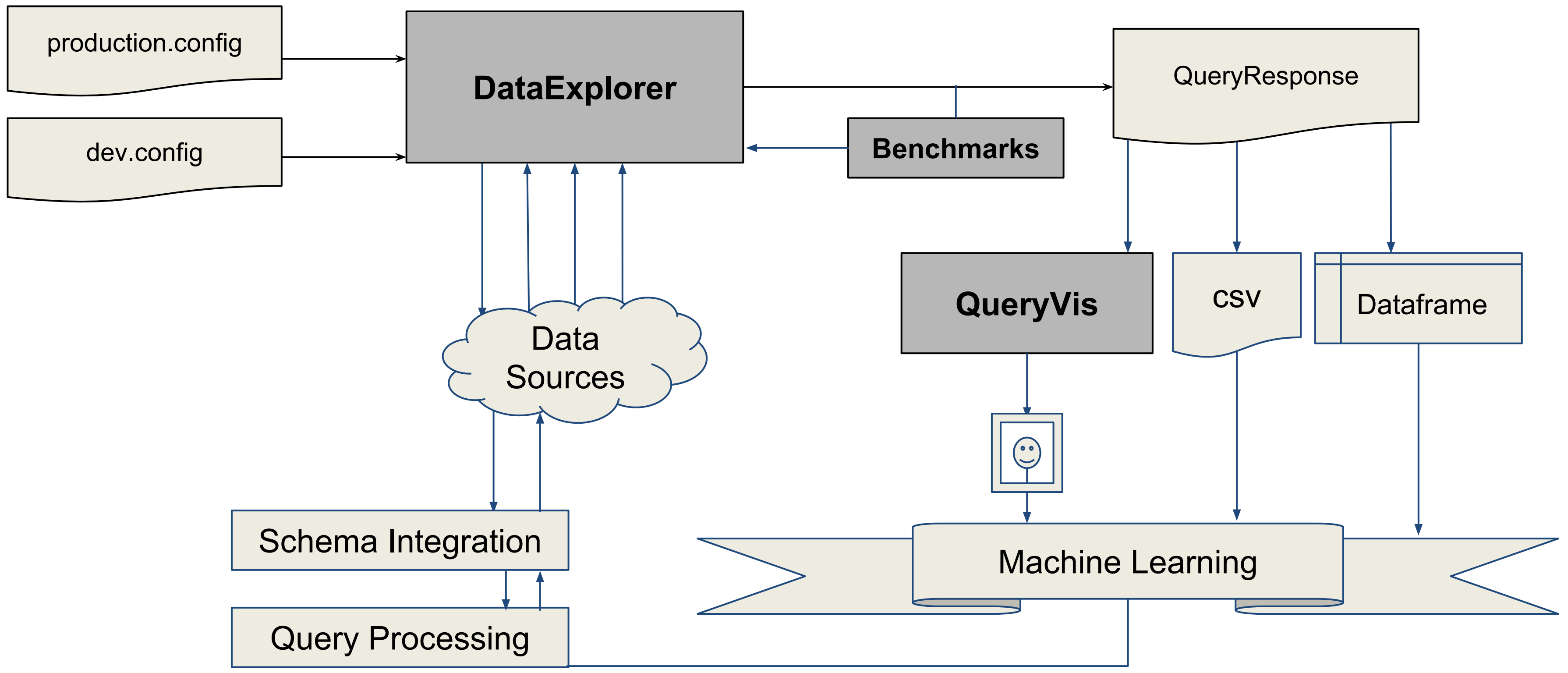
Data Exploration Objectives:

- Understand the data
- Develop an API that will be utilized by the Machine Learning Team
    - Functions that allow for data exploration
    - Create user friendly interface
    - Optimize functions for usability on a large scale

UC San Diego
JACOBS SCHOOL OF ENGINEERING

Data Exploration API Structure

Dependencies: Python 3, psycopg2, pysolr, pandas, sklearn



API Function Definitions

Customers

clusterCustomers(*feature_set, n_clusters, algorithm, init, cluster_on, scale*)
membersOfHousehold(*householdID, sample_size)*
productsByHousehold(householdID,*min_date*, *max_date*, *sample_size*)
statsByCustomer(customerID,*min_date*, *max_date*, *sample_size*)
statsByHousehold(householdID,*min_date*, *max_date*, *sample_size*)
idsForCustomer(customermatchedid)

Orders

statsByProduct(*min_date*, *max_date*, *sample_size*)
statsByZipcode(*min_date*, *max_date*, *sample_size)*

Reviews

asinByTerms(*terms*)
termsByAsin(*asin)*

Recommendations

statsByProduct(productid,*min_date*, *max_date*, *sample_size*)

Benchmarks

clientActivity(*aggregate*, *min_date'*, *max_date*, *clientid_filter*, *sample_size*)
statsByFunction(*min_date*, *max_date*, *function_filter*, *sample_size)*
insert(*function*, *args*, *kwargs*, *start*, *end*, *is_cached*, *client_id*)

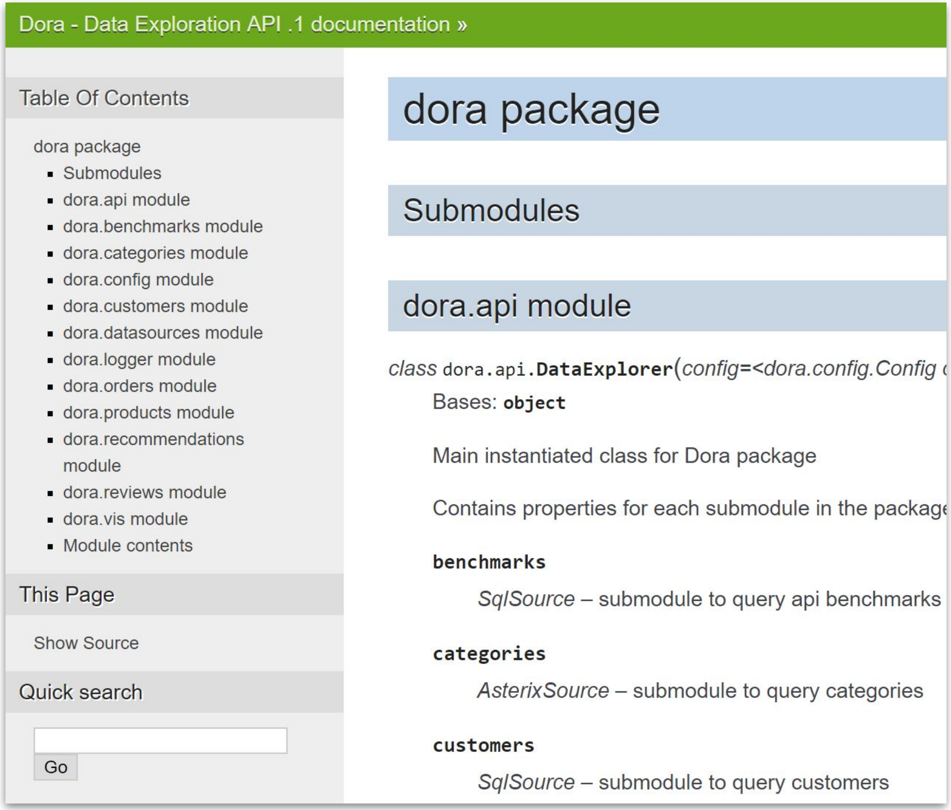
Products

byCategory(*node_id*)
clusterProducts(*feature_set, n_clusters, algorithm, init, cluster_on, scale, random_state, asin, PCA, n_components)*
coPurchases(asin, *min_date*, *max_date*, *sample_size*)
priceDistribution(bins)
ratingsDistribution(asin, *min_date*, *max_date*, *sample_size*)
seasonalOrderDistribution(asin)

Categories

search(*search_str*)
childrenOf(*node_id)*
parentOf(*node_id*)

Documentation Examples



Dora - Data Exploration API .1 documentation »

Table Of Contents

dora package
- Submodules
- dora.api module
- dora.benchmarks module
- dora.categories module
- dora.config module
- dora.customers module
- dora.datasources module
- dora.logger module
- dora.orders module
- dora.products module
- dora.recommendations module
- dora.reviews module
- dora.vis module
- Module contents

This Page

Show Source

Quick search

Go

dora package

Submodules

dora.api module

class `dora.api.`**`DataExplorer`**(*config=<dora.config.Config*

Bases: **`object`**

Main instantiated class for Dora package

Contains properties for each submodule in the package

`benchmarks`

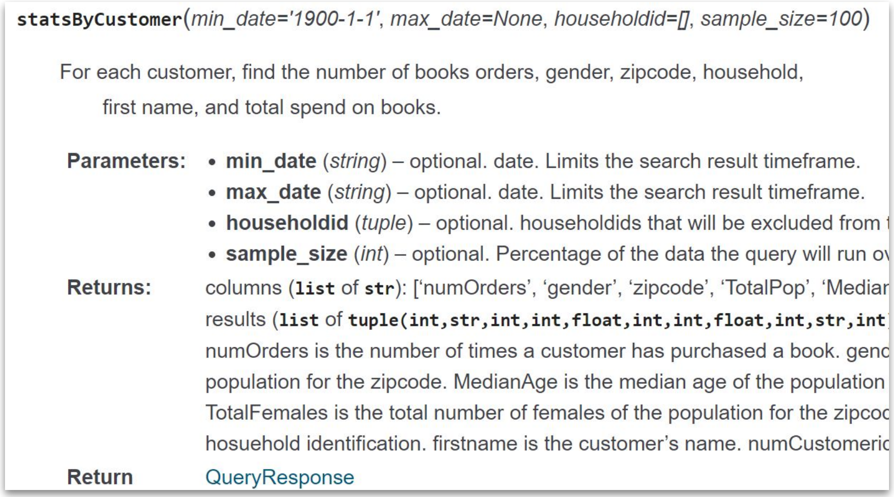
SqlSource – submodule to query api benchmarks

`categories`

AsterixSource – submodule to query categories

`customers`

SqlSource – submodule to query customers



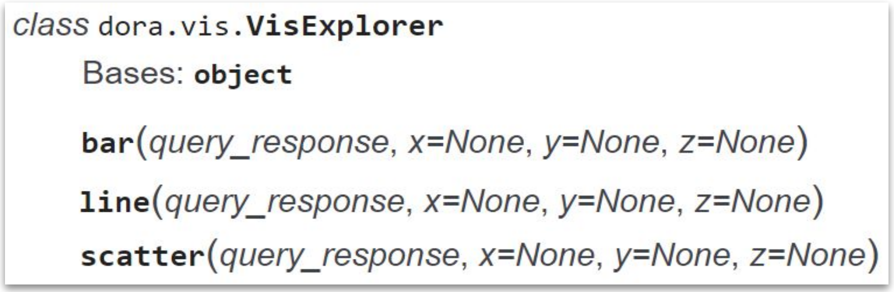
`statsByCustomer`(*min_date='1900-1-1', max_date=None, householdid=[], sample_size=100*)

For each customer, find the number of books orders, gender, zipcode, household, first name, and total spend on books.

Parameters:
- **min_date** (*string*) – optional. date. Limits the search result timeframe.
- **max_date** (*string*) – optional. date. Limits the search result timeframe.
- **householdid** (*tuple*) – optional. householdids that will be excluded from t
- **sample_size** (*int*) – optional. Percentage of the data the query will run ov

Returns: columns (**`list`** of **`str`**): ['numOrders', 'gender', 'zipcode', 'TotalPop', 'Mediar
results (**`list`** of **`tuple(int,str,int,int,float,int,int,float,int,str,int`**
numOrders is the number of times a customer has purchased a book. genc
population for the zipcode. MedianAge is the median age of the population
TotalFemales is the total number of females of the population for the zipcoc
hosuehold identification. firstname is the customer's name. numCustomeric

Return QueryResponse



class `dora.vis.`**`VisExplorer`**

Bases: **`object`**

`bar`(*query_response, x=None, y=None, z=None*)

`line`(*query_response, x=None, y=None, z=None*)

`scatter`(*query_response, x=None, y=None, z=None*)

API Function Collaborative Development

- Distribution Functions
  - priceDistribution
    - ML Team is using this to see the distribution of prices
  - seasonalOrderDistribution
    - Utilized to determine when products are purchased seasonally
- Clustering
  - clusterCustomers
    - Utilized to cluster customers into like groups to specify clustering for those customers
- Recommendations Exploration
  - Returns the number of times a product is recommended

Collaborated with Machine Learning Team to Develop Data Exploration Functions

API Challenges: Design

Consistent & Logical Organization

- Conventions
  - ClassName, functionName, attribute_name, parameter_name
  - Dates, Sample Sizes
  - QueryResponse across all DataSources, ok for Machine Learning functions?
- Hierarchical class structure
- Functional Module splits
  - Products vs Orders vs Orderlines
  - Customers vs Census

Flexibility

- Good default arguments
- Enough options to allow exploration
- Progressive disclosure

Dependency Management

- Interacting with Pandas + Sklearn, internal or not?
- Fewest dependencies without reinventing the wheel
- Solr libraries - variety of age and upkeep.

API Challenges: Implementation

Python Packaging

- Readme, setup.py
- pip install . [--upgrade]
- IPython notebook code vs .py code

Documentation

- Sphinx-Quickstart -> Sphinx-Apidoc -> Sphinx-Build
- Docstrings for each function.
    - Description, Arguments, Return Value
- Complete vs Redundant

Rapid Code Changes

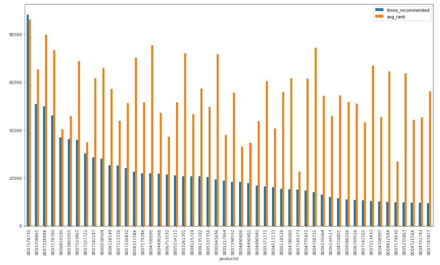
- Data Exploration minimum viable product needed first
- After a point, no breaking changes allowed.

ML Based Exploration

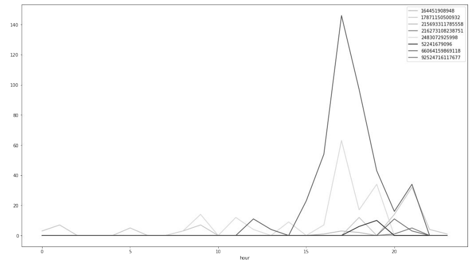
- Feature Development
- Clustering
- PCA

API Challenges: Visualization Capabilities

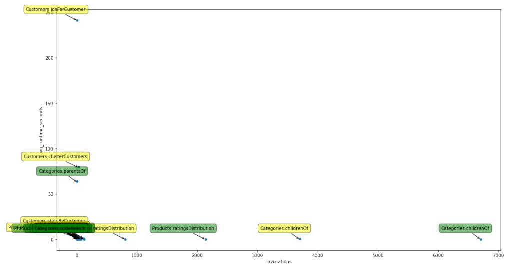
Goal: A module that takes a QueryResponse and produces useful default graphs.



Bar



Line



Scatter

Too many different use cases.
PyPlot.Line2D has 41 parameters just in the constructor.

Pandas, Numpy, Pyplot faster than dora.

API Challenges: Performance Optimization

Data Sampling

- AsterixDB - chose not to implement data sampling
  - Did not want to limit the categories dataset
- Solr - chose not to implement data sampling
  - Solr is efficient enough to manage full dataset

Indexing

- Focus on minimum viable product
- Need for rapid development and code changes from collaboration with other groups
  - Index selection change with each iteration
  - Based on benchmarking feedback

Query Response Time

- Original AsterixDB schema with nesting at each level
  - Query time-out or > 60 seconds for complex queries
  - Improved with flattened schema
- Clustering on 100,000+ “people,” accessing person from materialized view (customer matched, customer matched customerid)
  - Query time
  - Full table scan, indexing no help.
  - Cache locally, 3 minutes vs < 1 second

Materialized Views

- AsterixDB - Flattened category collection
- Postgres - Matched Customerid’s
  - (firstname, gender, householdid, [customerids])

UC San Diego
JACOBS SCHOOL OF ENGINEERING

Performance Optimization

🟩 Implemented	🟧 Pending	🟥 Not planned

	Postgres	**AsterixDB**	**Solr**
Client side caching	🟩	🟩	🟩
Server side caching	🟩 *	🟩 **	🟩 ***
Data sampling	🟩	🟥	🟥
Indexing	🟩	🟩	🟩
Benchmarking	🟩	🟩	🟩

* Default table and index caching
** Default buffer cache
*** Default filter, queryResponse, and document caches

API Demo