



CREATING A **FAIR** DATA CATALOG TO SUPPORT SCIENTIFIC MODELING

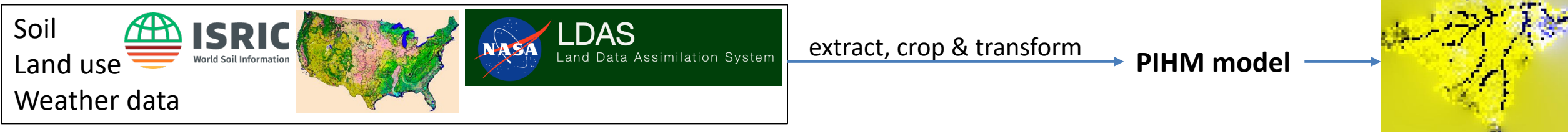
Basel Shbita, Binh Vu, Minh Pham, Dan Feldman

Craig Knoblock, Jay Pujara, Yao-Yi Chiang



Motivating Example

- Finding, preparing, and cleaning datasets dominate time devoted to scientific inquiry
 - Example: flooding prediction takes **months** for data prep.

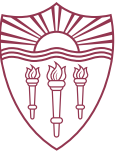


- Need a Data Catalog
 - **Retrieve datasets** by variables (e.g., SVO) and other metadata (temporal and geospatial coverage)
 - **Transform dataset** for different use cases with minimal effort
 - Reprojection, cropping, or format conversion
 - Joining multiple datasets
 - **Visualize dataset** to get insight of the data



Challenges in building the Data Catalog

- Registering data in the data catalog
 - Huge number of datasets with **massive amount of data**
 - Lots of **manual effort** for curating the datasets
 - Ambiguous data definition: different communities using **diverse terms** for same phenomena
 - Dataset are stored in **different formats** and layouts: NetCDF, CSV, spreadsheets...
- (Semi)-automatic data transformation
 - Generate & execute a transformation plan based on input and **desired output**
 - Easy to **reuse** existing transformation libraries or add new transformation

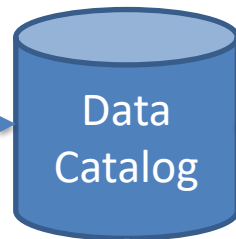
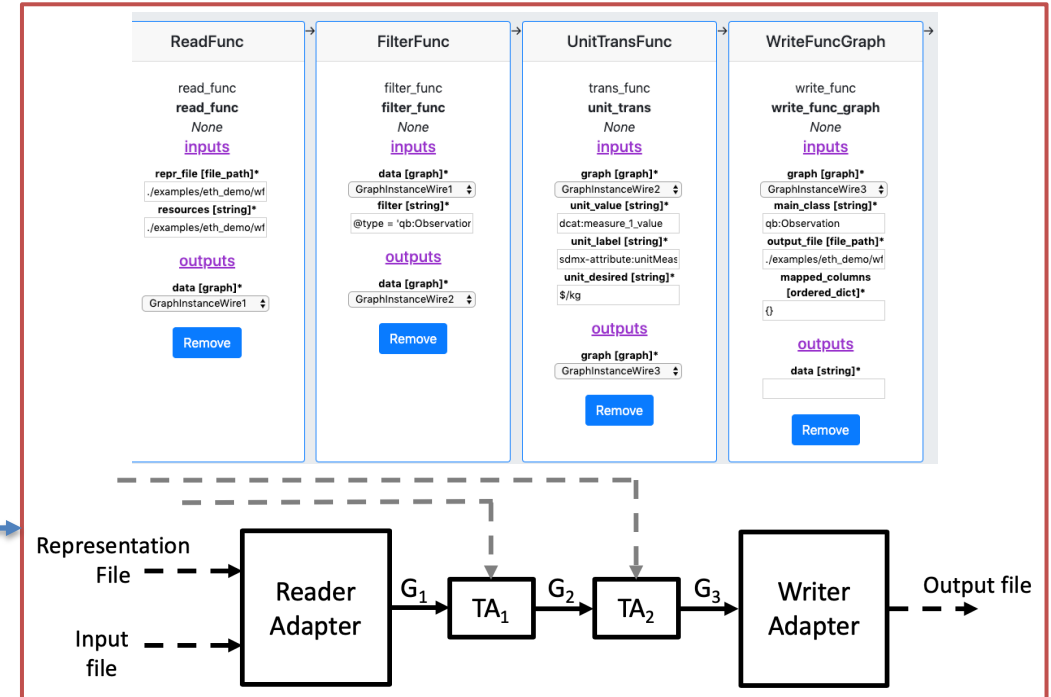


Overall approach

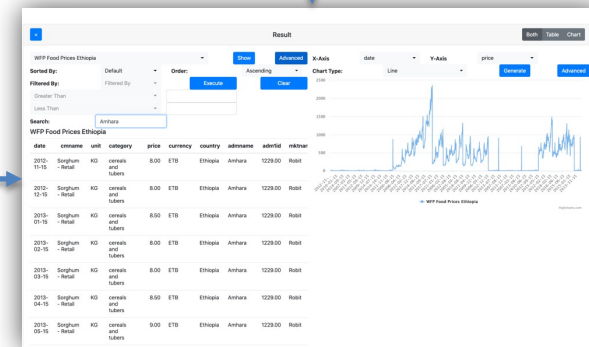
Indicator Name	Units	2005	2006
Cargo Traffic (M.Tor Million tonne		6915000.00	7100000.00
Cargo Traffic (M.Tor Million tonne		204929.00	182810.00
Container Traffic (TE TEUs			
Cargo Traffic (M.Tor Million tonne		5432353.00	5489586.00
Cargo Traffic, Annual Volume			
Cargo Traffic, Annual %			

dataset

Data Transformation



Visualization



Registration

MINT Dataset Registration

Dataset Name: World Development Indicators for Ethiopia

Resource URL: http://api.worldbank.org/v2/en/country/ETH

Dataset Description: Contains socioeconomic variables such as economic growth, education, healthcare.

File Type: CSV

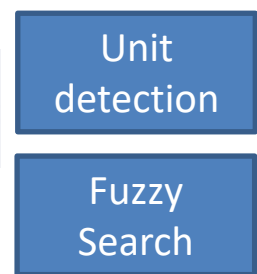
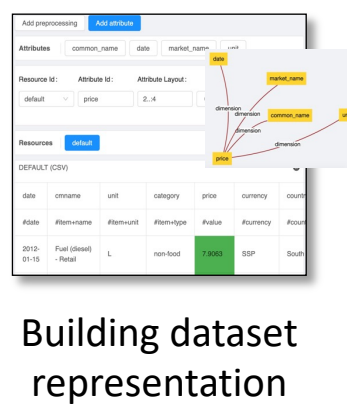
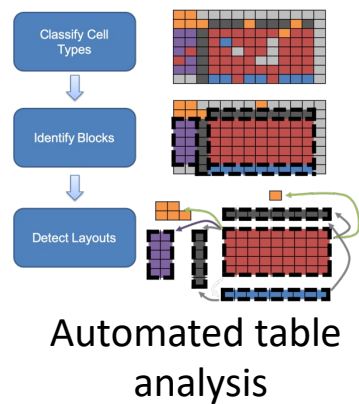
Start Time: 1990-01-01T00:00:00

End Time: 2018-12-31T23:59:59

Spatial Coverage: Ethiopia

Variables: Variable, Fuzzy Search

[Register]



Registering data

- Dataset registration
 - Can be done via API or UI
 - Provide basic metadata: variable, spatial and temporal coverage
- Finding correct variable (fuzzy search)
 - Semantic knowledge in Wordnet
 - Statistical associations with Word2Vec
 - Topic modeling and other string similarity metrics

MINT Dataset Registration

Dataset Name	<input type="text" value="World Development Indicators for Ethiopia"/>
Resource URL	<input type="text" value="http://api.worldbank.org/v2/en/country/ETH"/>
Dataset Description	<input type="text" value="Contains socioeconomic variables such as economic growth, education, healthcare"/>
File Type	<input type="text" value="CSV"/>
Start Time	<input type="text" value="1990-01-01T00:00:00"/>
End Time	<input type="text" value="2018-12-31T23:59:59"/>
Spatial Coverage	<input type="text" value="Ethiopia"/>
Variables	<input type="text" value="Variable"/> <input type="text" value="Fuzzy Search"/> <input type="button" value="+"/>
<input type="button" value="Register!"/>	

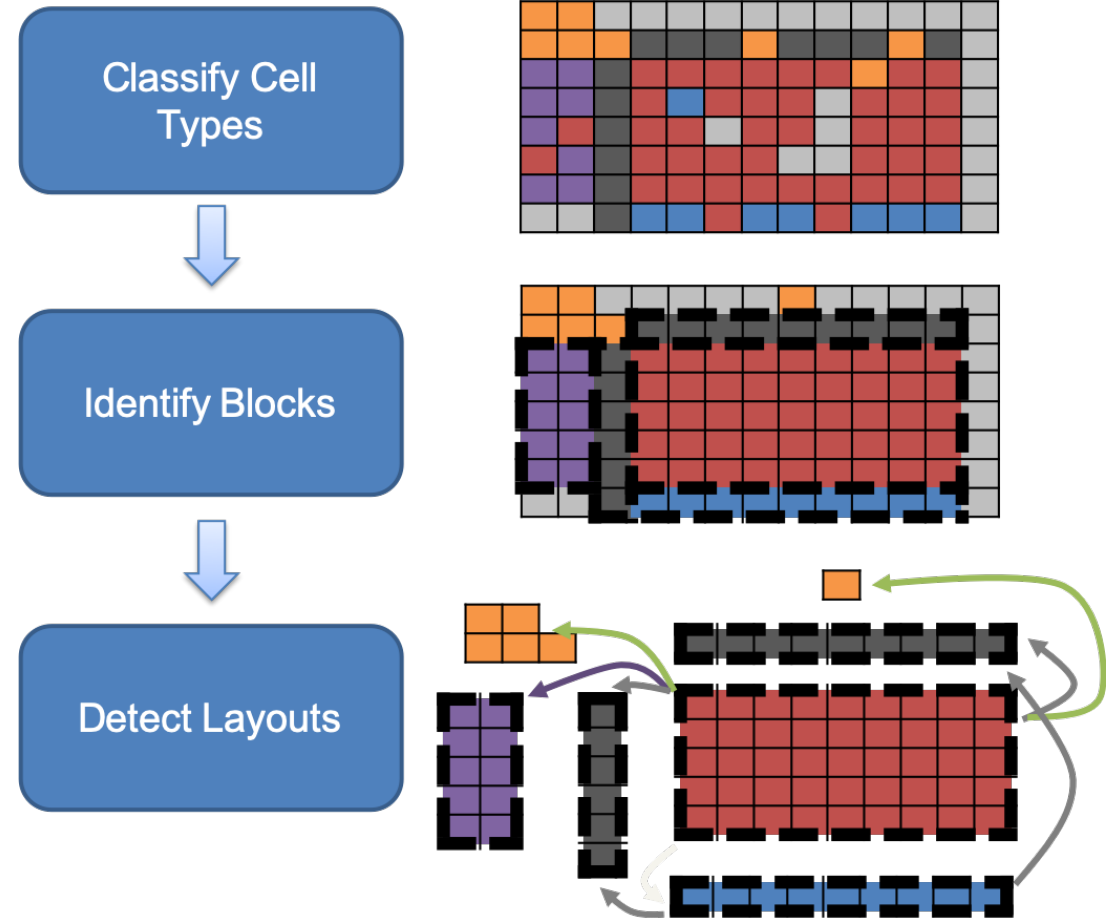
MINT Fuzzy Search

Matches	
Description	Score
atmosphere water precipitation leq volume flux	0.08106219740955758
atmosphere water one-month time integral of precipitation leq volume flux	0.036781124763790314
atmosphere water one-day time integral of precipitation leq volume flux	0.03655978314283351



Building rich data understanding

- Automated table analysis [2]
 - Input: tabular datasets
 - Output: dataset layout
 - Header/Attribute/Value blocks
 - Their relationships





Building rich data understanding (cont.)

- Dataset representation (D-REPR [3]). Why?
 - Different **formats** (NetCDF, spreadsheet, CSV, JSON)
 - Different **layouts** (matrix tables, hierarchical tables)
 - **Same interface** to access to many kinds of datasets
- Automatically generate D-REPR file
 - Table understanding
 - D-REPR GUI for curation

The screenshot shows the D-REPR GUI interface. At the top, there are buttons for 'Add preprocessing' and 'Add attribute'. Below, there are input fields for 'Attributes' (common_name, date, market_name, unit), 'Resource Id' (default), 'Attribute Id' (price), and 'Attribute Layer' (2..:4). A diagram on the right shows a network of nodes (date, market_name, common_name, unit, price) connected by 'dimension' labels. Below the diagram is a table titled 'DEFAULT (CSV)' with columns: date, cmname, unit, category, price, currency, country. The 'price' column for the row '2012-01-15 Fuel (diesel) - Retail' is highlighted in green.

date	cmname	unit	category	price	currency	country
#date	#item+name	#item+unit	#item+type	#value	#currency	#country
2012-01-15	Fuel (diesel) - Retail	L	non-food	7.9063	SSP	South

FY 2008	JAN	FEB	MAR
From domestic sugar beets	661,586	485,126	423,775
From imported sugar beets	0	37,160	0
Subtotal	661,586	522,287	423,775
Cane production:			
Florida	321,414	253,438	242,560
...			
Subtotal	378,919	283,190	289,237
Total	1,040,505	805,476	713,012

Indicator Name	Units	2005	2006
Cargo Traffic (M.Tor	Million tonne	6915000.00	7100000.00
Cargo Traffic (M.Tor	Million tonne	204929.00	182810.00
Container Traffic (TE	TEUs		
Cargo Traffic (M.Tor	Million tonne	5432353.00	5489586.00
Cargo Traffic, Annual	Volume		
Cargo Traffic, Annual	%		



Building rich data understanding (cont.)

- Four steps to create a D-REPR file

- Specify resources
- List attributes (or variables) in the datasets
- Simple rules to join values of the attributes
- Map attributes to predicates, classes in domain ontologies

Step 1

a. life table.csv

		2016	
Indicator	Age Group	Male	Female
LIFE_0035	<1 year	57.7	59.6
LIFE_0035	1-4 years	60.6	62.1

b. indicators.json

```
[
  {
    "indicator": "LIFE_0035",
    "url": "http://apps.who.int/.../indicator.aspx?iid=35"
  },
  {
    "indicator": "LIFE_0029",
    "url": "http://apps.who.int/.../indicator.aspx?iid=29"
  }
]
```

Step 2

gender	observation	indicator_column	indicator	url
(1,2) male	(2,2) 57.7	(2,0) LIFE_0035	(0,indicator) LIFE_0035	(0,url) http://...?iid=35
(1,3) female	(2,3) 59.6	(3,0) LIFE_0035	(1,indicator) LIFE_0029	(1,url) http://...?iid=29
	(3,2) 60.6			
	(3,3) 62.1			

Step 3

qb:Observation			eg:Indicator	
observation	gender	indicator_column	indicator	url
57.7	male	LIFE_0035	LIFE_0035	http://...?iid=35
59.6	female	LIFE_0035	LIFE_0029	http://...?iid=29
60.6	male	LIFE_0035		
62.1	female	LIFE_0035		

Step 4

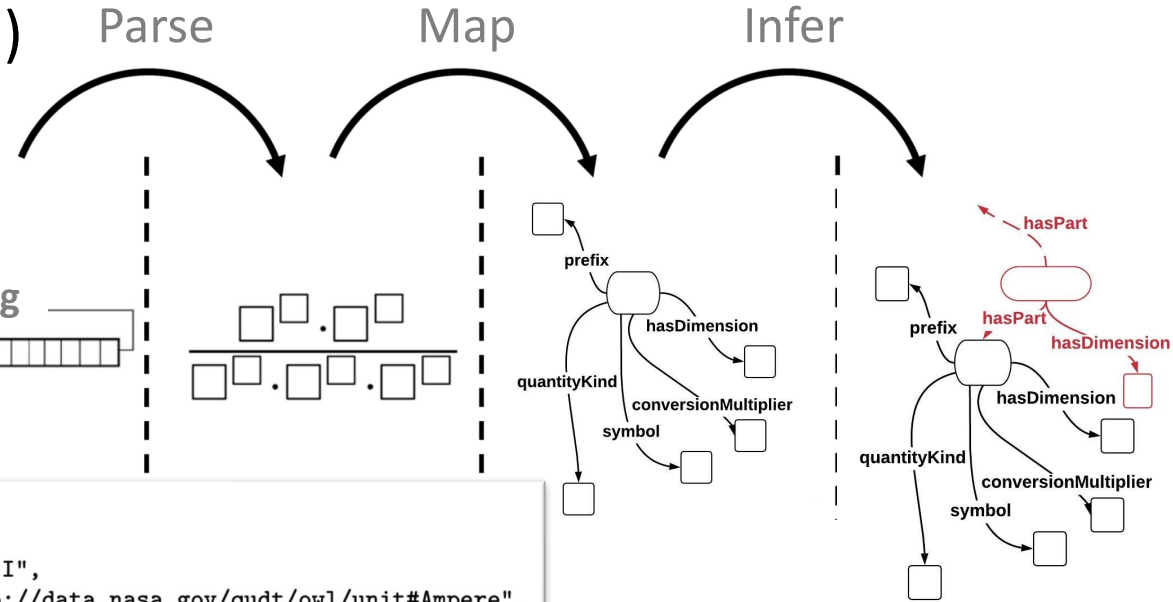
```

graph TD
    qbObservation((qb:Observation)) -- "sdmx-m:obsValue" --> observation[observation]
    qbObservation -- "sdmx-d:refPeriod" --> year[year]
    qbObservation -- "eg:indicator" --> egIndicator((eg:Indicator))
    qbObservation -- "eg:ageGroup" --> ageGroup[age group]
    qbObservation -- "sdmx-d:sex" --> gender[gender]
    egIndicator -- "drepr:uri" --> url[url]
    egIndicator -- "eg:code" --> indicator[indicator]
  
```




Building rich data understanding (cont.)

- Unit detection (CCUT [1])
 - **Identify**, parse, and map compound units of measurement to QUDT ontology (**semantic representation**)
 - Enable **automatic unit conversions**
 - Enhance the dataset **representation**



	A	B	C
2	Year in Production	Units	200
3	Technology Generation		
4	Physical Lgate (Low-Standby-Power)	nm	75
5	EOT (Equivalent Oxide Thickness)	A	22
6	Gate Poly Depletion & Inversion-Layer Thickness	A	8
7	Inversion Gate Dielectric Thickness Value	A	30
8	Maximum Gate Leakage Limit	A/cm ²	4.4E
9	Power Supply Voltage	v	1.2
10	Saturation Threshold Voltage	V	0.5

```

qudt:_hasPart_: [
  {
    qudt:hasDimension: "I",
    qudt:quantity: "http://data.nasa.gov/qudt/owl/unit#Ampere",
    qudt:symbol: "A"
  },
  {
    UNK:exponent: "-2",
    UNK:prefix: "http://data.nasa.gov/qudt/owl/unit#Centi",
    UNK:prefix_conversion_multiplier: 0.01,
    qudt:hasDimension: "L",
    qudt:quantity: "http://data.nasa.gov/qudt/owl/unit#Meter",
    qudt:symbol: "cm"
  }
]
qudt:abbreviation: "A cm-2",
qudt:hasDimension: "L-2 I"

```





Querying data from data catalog

- Search datasets (API-based) by
 - dataset name
 - variables
 - temporal coverage
 - spatial coverage (bounding box)
- Return a subset of data in different format
 - Using data transformation component

Example queries can be found in the [Api Demo](#) notebook

Searching for datasets/resources
After registering datasets/variables/resources, we can now programmatically search of relevant information. Below, you'll see 3 examples of searching for data using standard variable names, temporal, and spatial coverages. Currently, these are the only search filters we support, but we'll be adding more as we get more feature requests. If you would like to search data catalog by other keywords, please let me know at dianf@usc.edu

```
In [ ]: # 1) Searching by standard_names
search_query_1 = {
    "standard_variable_names_in": [temperature_standard_variable("name")]
}
resp = requests.post(f"{url}/datasets/find",
                    headers=request_headers,
                    json=search_query_1).json()
if resp['result'] == 'success':
    found_resources = resp['resources']
    print(f"Found {len(found_resources)} resources")
    print(found_resources)
```

```
In [ ]: # 2) Searching by spatial_coverage
# Bounding box search parameter is a 4-element numeric array (in WGS84 coordinate system) [xmin, ymin, xmax, ymax]
# As a reminder, x is longitude, y is latitude
bounding_box = [
    spatial_coverage["value"]["xmin"],
    spatial_coverage["value"]["ymin"],
    spatial_coverage["value"]["xmax"],
    spatial_coverage["value"]["ymax"]
]
search_query_2 = {
    "spatial_coverage_within": bounding_box
}
resp = requests.post(f"{url}/datasets/find",
                    headers=request_headers,
                    json=search_query_2).json()
if resp['result'] == 'success':
    found_resources = resp['resources']
    print(f"Found {len(found_resources)} resources")
    print(found_resources)
```

```
In [ ]: # 3) Searching by temporal_coverage and standard_names
# Bounding box search parameter is a 4-element numeric array (in WGS84 coordinate system) [xmin, ymin, xmax, ymax]
# As a reminder, x is longitude, y is latitude
start_time = "2018-01-01T00:00:00"
end_time = "2018-01-21T23:59:59"
search_query_3 = {
    "standard_variable_names_in": [temperature_standard_variable("name")],
    "start_time_gte": start_time,
    "end_time_lte": end_time
}
resp = requests.post(f"{url}/datasets/find",
                    headers=request_headers,
                    json=search_query_3).json()
if resp['result'] == 'success':
    found_resources = resp['resources']
    print(f"Found {len(found_resources)} resources")
    pprint(found_resources)
```

```
In [ ]: # 4) Searching by dataset_names
search_query_4 = {
    "dataset_names_in": ["Temperature recorded outside my house"]
}
resp = requests.post(f"{url}/datasets/find",
```

1. Semantically search contents*

Are there rainfall data for Juba?

Potentially. There is "2018_temperature_Juba.csv", containing "precipitation", which is semantically similar to "rainfall".

2. Return just a subset of data*

Give me a list of daily highs in Juba for July 2018

Date	T, High
Jul 1	30
...	...
Jul 31	34

3. Return data in a different format*

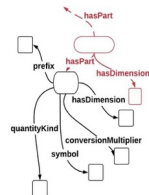
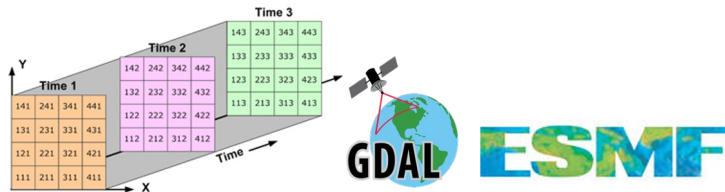
Give me 2018_temperature_Juba data in NetCDF format

2018_temperature_Juba.nc

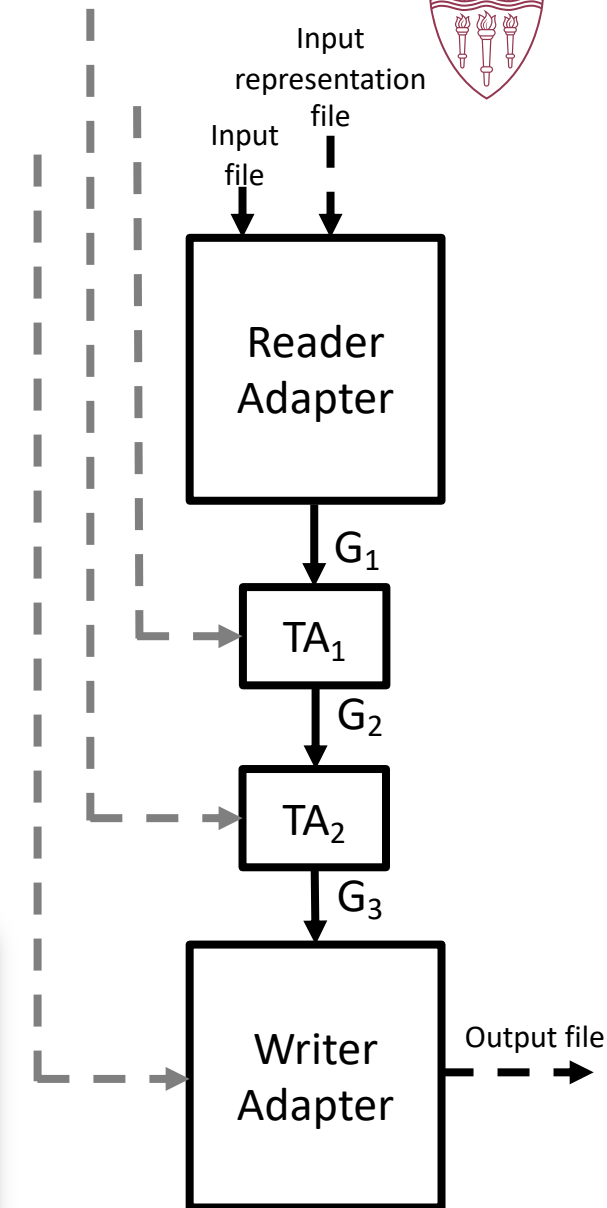


Data Transformation

- Construct **transformation pipeline**
- **Structured Representation**
 - Uses D-REPR representation
 - Supports **complex data** (N-dim, data across multiple files)
 - Captures and leverages a **semantic meaning** of the data
 - Can be mapped to any **ontology** (i.e. SVO, DataCube)
 - Makes the data **format-independent**
- **Building blocks architecture**
- No more manual-coding for each transformation
 - Easy to **reuse** existing building blocks

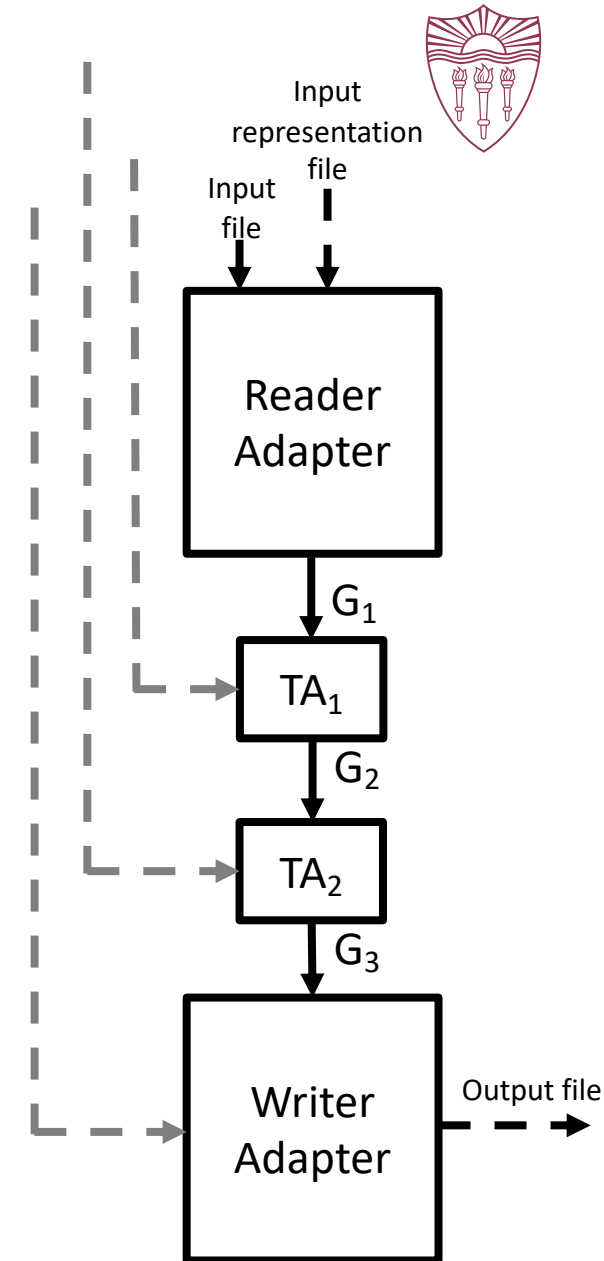


ReadFunc	FilterFunc	UnitTransFunc	WriteFuncGraph
read_func read_func None inputs repr_file (file_path)* ./examples/leth_demo/wf resources (string)* ./examples/leth_demo/wf outputs data (graph)* GraphInstanceWire1	filter_func filter_func None inputs data (graph)* GraphInstanceWire1 filter (string)* @type = "qb:Observation" outputs data (graph)* GraphInstanceWire2	trans_func unit_trans None inputs graph (graph)* GraphInstanceWire2 unit_value (string)* dict:measure.T.value unit_label (string)* sdmx-attribute:unit:Meas unit_desired (string)* outputs graph (graph)* GraphInstanceWire3	write_func write_func_graph None inputs graph (graph)* GraphInstanceWire3 main_class (string)* qb:Observation output_file (file_path)* ./examples/leth_demo/wf mapped_columns (ordered_dict)* {} outputs data (string)*



Data Transformation Architecture

- Adapters (building blocks/‘components’)
 - Three types:
 - Reader (entry point)
 - Reads input file (data) and description of it (variables, relations, semantics)
 - Transformer
 - Does not materialize the data, just reproduces it
 - Writer (exit point)
 - Writes output file (data) based on a description of it (variables, relations, semantics)
 - Enable input data validation and compatibility checking
- Pipeline
 - Define the required **inputs** for some adapters
 - **Wire** some inputs to outputs (‘concatenate’ the components)
 - Execute the pipeline!





Data Transformation UI

MINT-DT Home Browse Pipeline

Search

1. Explore adapters

Choose an adapter:

0 read_func/ReadFunc

ReadFunc

read_func
read_func
None

inputs

repr_file [file_path]*
resources [string]*

outputs

data [graph]*

Choose an adapter:

- ✓ 0 read_func/ReadFunc
- 1 filter_func/FilterFunc
- 2 unit_trans/UnitTransFunc
- 3 write_func_graph/WriteFuncGraph
- 4 write_func_ndarray/WriteFuncNDimArray
- 5 graph_str2str_func/GraphStr2StrFunc

ReadFunc

read_func
read_func
None

inputs

repr_file [file_path]*
./examples/eth_demo/wf
resources [string]*
./examples/eth_demo/wf

outputs

Create graph instance
✓ GraphInstanceWire1
GraphInstanceWire2
GraphInstanceWire3

Input files

"wires" = graph instances

2. Construct pipeline

add to pipeline

ReadFunc

read_func
read_func
None

inputs

repr_file [file_path]*
./examples/eth_demo/wf
resources [string]*
./examples/eth_demo/wf

outputs

data [graph]*
GraphInstanceWire1

FilterFunc

filter_func
filter_func
None

inputs

data [graph]*
GraphInstanceWire1
filter [string]*
@type = 'qb:Observator'

outputs

data [graph]*
GraphInstanceWire2

UnitTransFunc

trans_func
unit_trans
None

inputs

graph [graph]*
GraphInstanceWire2
unit_value [string]*
dcat:measure_1_value
unit_label [string]*
sdmx-attribute:unitMeas
unit_desired [string]*
\$/kg

outputs

graph [graph]*
GraphInstanceWire3

WriteFuncGraph

write_func
write_func_graph
None

inputs

graph [graph]*
GraphInstanceWire3
main_class [string]*
qb:Observation
output_file [file_path]*
./examples/eth_demo/wf
mapped_columns [ordered_dict]*
{}

outputs

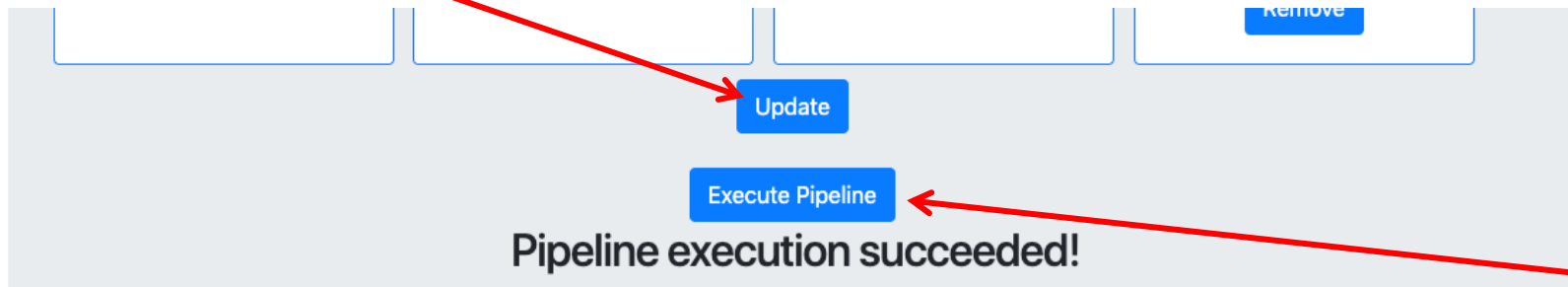
data [string]*

remove from pipeline



Data Transformation UI (cont'd)

Apply (online) updates and modifications

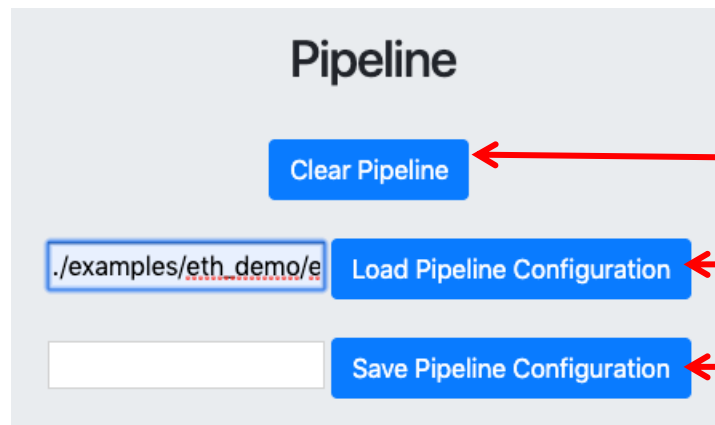


3. Execute pipeline

Generates output file

wfp_food_prices_ethiopia.modified

(optional) Save/Load pipelines



Clear pipeline

Load from a ready config file

Save to config file



Future work

- Dataset Registration
 - **Data discovery** for finding and adding more datasets
 - **Registering high volume** of data sitting behind servers (RESTful API)
 - Improve **automatic data understanding**
 - Table understanding
 - Unit Detection
- Dataset Query
 - GUI for **browsing and visualizing datasets** in the Data Catalog
- Data Transformation
 - Generating transformation plan (semi-)**automatically** (“transformation reasoning”)



Summary

- The Data Catalog allows dataset **registration, search, transformation** and **visualization**
- **Automated** tools to support adding additional **meta-data** to the Catalog
- Transformation framework that allows **constructing, executing** and **validating** a **transformation** pipeline
 - Easy to use and easily extended (no need to re-implement a complete flow)
- More information can be found at
 - <https://mint-project.org>
 - <https://github.com/mintproject>





References

- [1] B. Shbita, A. Rajendran, J. Pujara, and C. Knoblock, Parsing, Representing and Transforming Units of Measure, in Modeling the World's Systems, 2019
- [2] J. Pujara, A Common Framework for Developing Table Understanding Models, ISWC 2020
- [3] B. Vu, J. Pujara, and C. Knoblock, D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF, K-CAP 2019