# Semantic Data Labelling & Type Safety

## The Quest for sPandas

Riley Moher

# What is semantic type safety?

CompanyA

| Quarter | Profit | Operating Costs |
| --- | --- | --- |
| Q1-2014 | 158 000 000 | 35 000 000 |
| Q2-2014 | 200 000 000 | 38 000 000 |
| ... | ... | ... |

CompanyB

| Quarter | Profit | Operating Costs |
| --- | --- | --- |
| Q1-2014 | 16 | 2.6 |
| Q2-2014 | 30 | 3.3 |
| ... | ... | ... |

[Profit Comparison] = CompanyA.Profit - CompanyB.Profit

- How do we ensure this computation is correct and meaningful?
- What could cause misinterpretation?
  - Different profit currencies
  - Different profit scale
  - Different quarters
  - Different profit definitions

# What does the solution look like?

CompanyA

| Quarter | Profit | Operating Costs |
|---|---|---|
| Q1-2014 | 158 000 000 | 35 000 000 |
| Q2-2014 | 200 000 000 | 38 000 000 |
| ... | ... | ... |

Company A Unit Definitions:

- Profit is net profit

Company B Unit Definitions:

- Profit is gross profit

sPandas Ontology Layer

CompanyB

| Quarter | Profit | Operating Costs |
|---|---|---|
| Q1-2014 | 16 | 2.6 |
| Q2-2014 | 30 | 3.3 |
| ... | ... | ... |

IN:

```
CompanyA = SemanticDataFrame(data=CompanyA, units = CompanyAUnits)
CompanyB = SemanticDataFrame(data=CompanyB, units = CompanyBUnits)
[Profit Comparison] = CompanyA.Profit - CompanyB.Profit
```

OUT:

```
    WARNING: Semantic dtype mismatch: ‘profit type: expected {net} got {gross}’
```

# How is type safety enforced?

- sPandas will contain a semantic units representation that underlies every semantic dataframe. These contain axioms which describe the different concepts that can be represented and how they are represented.
- When performing an operation requiring type-safety, types must be verified as semantically compatible:
    a. First and foremost, they must be representing the same concept
    b. They must be measured with the same scale and unit, or automatically converted to make this true (according to conversion semantics defined within the unit definitions)
    c. They must have the same stratification(s), or be automatically converted to make this true (according to stratification semantics defined within the unit definitions)

# Logic of Type-safety

- Components of units
  - Scale
  - Dimension (what are they actually measuring?)
  - Measure (what is the measure of the unit)
    - Has a numerical value
  - Domain-specific properties
    - Can be comparable or not
- Comparability
  - Properties of a given unit or dimension may be comparable or not
  - Non-comparable property means automatic conversion between distinct values of that property is not possible.
  - Some properties may only be comparable with additional information

# Logic of Type-safety

- Using previous basic example of comparing profits of two different companies

**CompanyA**

- Profit(A)
- Profitkind(A,Net)
- hasScale(A, 1.0)
- hasCurrency(A,USD)

**CompanyB**

- Profit(B)
- Profitkind(B,Gross)
- hasScale(B, 10 000 000)
- hasCurrency(B,USD)

**Unit Axioms**

- hasMeasure(Profit, Money)
- NoncomparableDataProperty(ProfitKind)
- Dimension(Profit)

- We can convert the scale, and the currencies are comparable, but the different kinds of profits will cause a semantic error

# Classifying sPandas Operations

- Different operations of sPandas can be classified in 3 ways
    - Operations enforcing type safety
    - Operations producing new types
    - Type neutral operations

# Type-Safe Enforced Operations

- Addition/Subtraction
  - dfA[‘col1’] +/- dfB[‘col2’] -> verify concept being represented is the same, automatically change scale/unit if conversion definition exists, otherwise raise semantic type mismatch error
- Filtering
  - df[df[‘price’] >=  sPandasUnit{value: 10M, scale: 1, curr: USD, agg: {mean across 7 days}] -> check for equivalence of each price row according to the semantics of equivalence for that unit, performing any automatic conversions if applicable.
- Concat, append, join, merge
  - df1.join(df2), df1.append(df2) -> any columns from one dataframe being added to another must be verified for compatability

# Type-Changing Operations

- Groupby
  - Will attempt to convert units of columns other than the one being grouped-by according to groupby aggregation method invoked
- Statistic operations: mean, count, max, min, median, std
  - Concept being represented is the same, but stratification is put in place according to the statistic
- Apply
  - Any functions called with df.apply(func) must have a static-typed return
- Importing
  - Importing will automatically apply typing to certain variables (binary, categorical), will convert other units to basic dimensionless unit type
- Multiplication/Division
  - New units defined through geometric combination of multiple units, ex: cm^2 == [cm]*[cm], $/person (annual),

# Type-Neutral Operations

- Sorting operations
    - Sorting changes order, no type awareness necessary
- Exporting
    - Exporting will behave as in pandas, could possibly export some other metadata file for units?

# Inference

- One of the advantages of using an ontology-based approach is the ability to leverage an automated reasoner
    - Can check if a given unit class subsumes another
    - Can also use multiple inheritance and multiple instantiation
    - Can also find units that are equivalent, if subsumption check is true in both directions (automatically done with Hermit reasoner)

# Unit Types

- Units of Measure
  - Include familiar scientific units of measurement for length, volume, area, etc.
  - Also includes measurable concepts like money-based concepts, profit, GDP, etc.
- Time dimension
  - Quantification of time-based validity
  - Can be combined with units of measure
  - Could be instantaneously defined (timestamped measurement) or defined over a time-period
- Statistical dimension
  - Representation of aggregational operation(s)
  - Can be combined with units of measure
- Geospatial dimension
  - Similar to time and statistical, modifier for measure unit
  - Defined over some geographical space

# Unit Types

- Categorical unit
    - Basic unit to define categorical units
    - Semantics of these units will be harder to generalize, could provide some basic framework to be extended on a per-case basis

# Avoiding Consequences

- Through real failure analysis in 3 domains, we can show how a system like sPandas can mitigate real consequences

# Predicting Congestive Heart Failure with 100% Accuracy?

- In a paper published January 2020, Porumb et al present a CNN model that accurately identifies congestive heart failure (CHF) on the basis of one raw electrocardiogram (ECG) heartbeat only
- The 100% accuracy boasted is misleading, heterogeneous data sources and failure to properly validate caused a big problem
  - 100% accuracy is on the training dataset, 97.8% accuracy on testing set
  - Only 33 subjects as datasources
  - **Positive and negative samples came from different ECG sources, with different sampling frequencies**
  - Data sources were all from lead-1 ECG (single lead)
- Having data source information encoded within the data could have prevented this inaccurate analysis
  - When analyzing the model's predictions, semantic information about the data source could have revealed biases
  - Semantic data provenance would have revealed the artificial downsampling difference in the positive vs negative classes

# Use Case 1: Python for Finance - Time Series Analysis

- Data sources are TSLA financial information and US macroeconomic data
- Utilizes many of pandas' timeseries functions
  - Resample
  - Shift
  - Rolling
- Uses statistical models for time series modeling
  - ETS
  - EWMA
  - ARIMA
- Will be useful to demonstrate time series representations
  - Aggregations over periods of time
  - Timeseries data with an implied causality

# Use Case 1: Python for Finance - Time Series Analysis

**What's the challenge?**

- All timeseries information is contained within the index
- No data provenance across timeseries operations like shifts, reindexing, etc
  - Units transformed through these operations will have some assumptions in the way they're aggregated, these assumptions are lost through these transformations

**How can sPandas address it?**

- Integrating timeseries information in the units
  - Ex: print(stock['price'].dtype) → sPandas Unit {scale: 1, curr: USD, agg: {mean across 7 days}
- Automatic data provenance through operations

# Use Case 2: Cervical Cancer Risk Classification

- Data contains many different categorical variables
- No intuitive units for this data, physical quantities would not be useful
- Good example of filling missing categorical data
  - What are the semantics for values filled by taking median, etc?
- Contains many ML data formatting and processing tasks
  - Can be used to evaluate sPandas

# Use Case 2: Cervical Cancer Risk Classification

**What's the challenge?**

- Categorical values may have different names for same concepts - large vocabulary in medical field
- Filling missing values contains assumptions that are not transparent and are not easily traceable
- Normalization of values reduces information in data, provenance not maintained

**How can sPandas address it?**

- Automatic conversion of categorical variables to a categorical unit type, which can be later refined to some specific concept
- Ontological definition of medical terms from a medical domain ontology
- Integrated data provenance

# Use Case 3: Identifying Invalid GPS Taxi Data

- Combines geospatial with timeseries
  - A good opportunity to integrate both representations
- In this specific use case, pandas should make the task of identifying invalid trip data much simpler
  - Semantics of geospatial units will have axioms that are analogous to the functions being manually implemented in this use case
  - Reduces work required in geospatial data cleaning

# Use Case 3: Identifying Invalid GPS Taxi Data

**What's the challenge?**

- Same challenges as in use case 1 with regards to the timeseries element
- Identifying outliers in geospatial data requires considerable manual work
  - Manually defining a filtering function
  - Requires a high degree of exploratory data analysis that could be reduced with some base-level geospatial semantics

**How can sPandas address it?**

- With embedded geospatial semantics, tasks like filtering out invalid GPS data will be much faster
  - Semantics of distance, travelling, etc.
  - Axioms will be able to error-check data like this with much less effort

# Implications for Reproducible Research

- Automatic data provenance, especially as an extension of a widely-used tool in data science, makes results clearer to follow, mistakes easier to spot.