

Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows

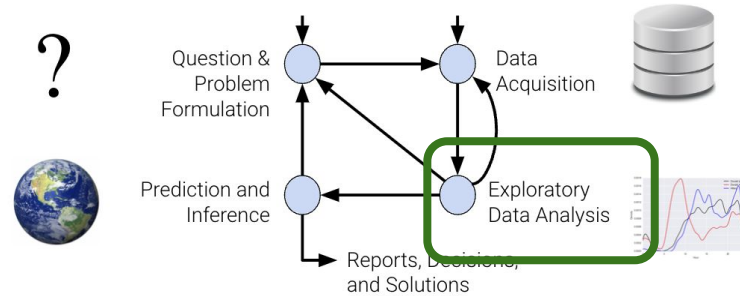
Authors: Doris Jung-Lin Lee et al. at ISchool, UC Berkeley

Year: 2021-May

Code Repository at: **<https://github.com/lux-org/lux>**

Author Role Play: Mayank Sethi

Introduction & Inspiration



Typical Data Science Lifecycle

- **EDA** is an essential part of performing Data Science and Analytics task.
- This exploratory data science is further **an iterative, trial-and-error process**, involving many interleaved **stages of data cleaning, transformation, analysis, and visualization**. And **>75% Data Scientists** normally use **DataFrame APIs** like **Pandas** for performing such operations.

Inspiration

- But **Visualizing dataframes** is an **unwieldy and error-prone process**, adding substantial friction to the fluid, iterative process of data science.

Reasons:

a. Cumbersome Boilerplate Code

- **Substantial load** is necessary to generate visualizations that users have to context-switch: **thinking about data operations and visual elements**.
- **Consequence:** Barriers hinder exploration and users often visualize during the late stages of workflow rather than experimenting.

b. Challenges in Determining Next Steps.

- The many choices of the many combinations make it hard to determine what **visualization to generate** to advance analysis, and automated assistance is not provided.

Proposed Solution



A Python API for Intelligent Visual
Discovery [↗](#)

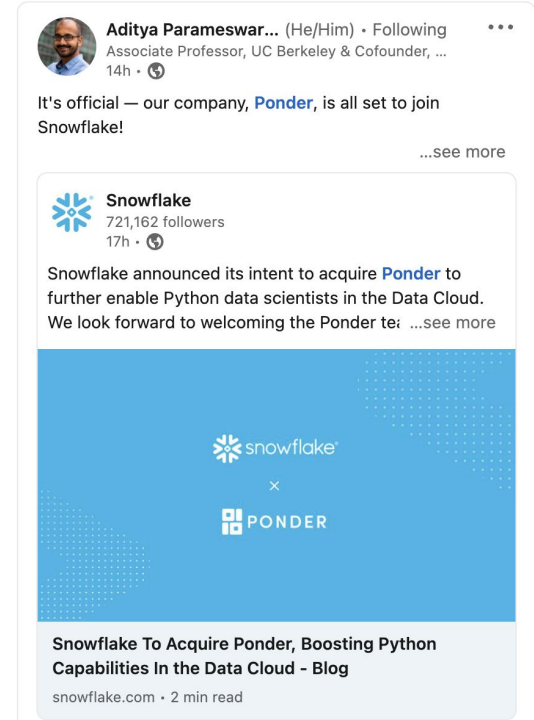


- The authors introduce **Lux**, a seamless extension to pandas that retains its powerful API, but enhances the tabular outputs with **automatically-generated visualizations highlighting interesting patterns and suggesting next steps for analysis.**
- It has monthly downloads around 9k (with a total of 62k downloads), and over 3.1k stars on Github, as of November 2021.

Interesting fact about Lux

- Lux was started as an initiative by a Phd student **Dorris Lee** at **Information School, Berkeley** with efforts from **Marti Hearst and Dr. Aditya**. The initiative got funded by top VCs like Lightspeed Partners, and is currently part of a company called **Ponder** run by the professors.

Congratulations !!🎉
To the team of PONDER





Aditya Parameswarar... (He/Him) · Following
Associate Professor, UC Berkeley & Cofounder, ...
14h · 🌐

It's official — our company, **Ponder**, is all set to join Snowflake!
...see more

Snowflake
721,162 followers
17h · 🌐

Snowflake announced its intent to acquire **Ponder** to further enable Python data scientists in the Data Cloud. We look forward to welcoming the Ponder team to Snowflake!
...see more

 snowflake
×
 PONDOR

Snowflake To Acquire Ponder, Boosting Python Capabilities In the Data Cloud - Blog
snowflake.com · 2 min read

Contributions

1. A **novel, always-on framework** that provides visualizations for the dataframe as it stands at any point in the workflow.
 - This multi-tiered dataframe interaction framework supports **pandas' 600+ operators** without compromising the ease and flexibility of data transformation and analysis.
2. An expressive and succinct intent language powered by a **formal, algebra** that allows users to specify their fuzzy intent at a high level.
 - It allows users to create one or more visualizations but also flexibly indicate their high-level analysis interest, without worrying about how the data elements map onto aspects of the visualization.
3. A **novel recommendation system** that uses automatically extracted information about dataframes to infer the appropriate visualizations.
 - Lux is one of the first of such systems that is designed to fit into a programmatic dataframe workflow.

Approach

1. Lux draws insights from **Visualization Recommendation (VisRec)**
 - Interactive GUI-based tools like **Tableau** are not widely used by data scientists due to their **lack of customizability and integration** with the data science workflow. Lux takes recommendation principles from this literature and explores how visualization recommendations can support a dataframe workflow.
2. Lux is built on top of imperative and declarative frameworks for **Visualization Specification**.
 - It synthesizes visualization code to enable users to customize as needed like defining axes, labels, etc.
 - Lux's intent language reduces the specification burden on users, allowing them to provide lightweight intent as opposed to writing long code fragments for visualization.
3. Visual Data Exploration with **Dataframes**
 - Lux lowers the barrier to visualizing dataframes by adopting an always-on approach so that dataframe visualizations are always recommended to users at all times, unlike pandasgui.

Demonstration

Overview Video: [Lux JupyterCon2020 Lightning Talk](#)

Demonstration: <https://github.com/lux-org/lux>

Framework for DF Interaction

In Lux's always-on framework, users inspect a dashboard of recommended visualizations, as part of a multi-tiered framework (blue), all of which is driven by a user- or system specified intent (orange)

Intent: Users can indicate aspects of the DF that they are interested in via a lightweight intent specification

```
df.intent = ["Inequality", "AvrgLifeExpectancy"]
df
```

Toggle Pandas/Lux

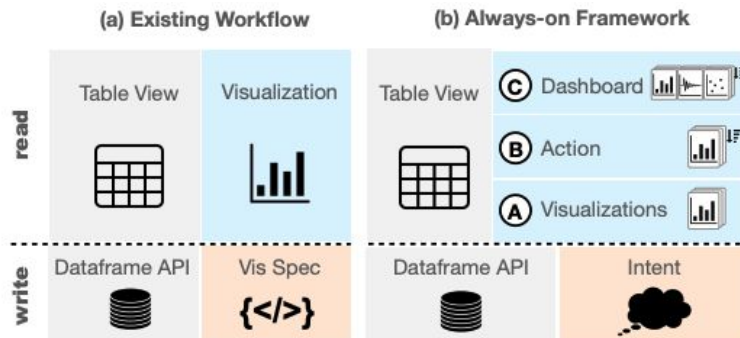


Figure 5: Conceptual framework for dataframe interaction. Users can make changes to anything below the dotted line (write), elements displayed to the user are shown above the dotted line (read). (a) In existing workflows, users write visualization specification code to create one or more visualizations. (b) In Lux's always-on framework, users can optionally make changes to the intent, which steers the recommended visualizations (Visualizations, Action, Dashboard).

Grammar behind Intent

The intent language is a lightweight, succinct means for users to declaratively specify their high-level interests.

$$\langle \text{Intent} \rangle \rightarrow \langle \text{Clause} \rangle^+$$
$$\langle \text{Clause} \rangle \rightarrow \langle \text{Axis} \rangle \mid \langle \text{Filter} \rangle$$
$$\langle \text{Axis} \rangle \rightarrow \langle \text{attribute} \rangle^* \langle \text{channel} \rangle \langle \text{aggregation} \rangle \langle \text{bin_size} \rangle$$
$$\langle \text{Filter} \rangle \rightarrow \langle \text{attribute} \rangle [= > < \leq \geq \neq] \langle \text{value} \rangle$$

Benefits:

Versatility: It serves both as a mechanism for steering recommendations and as a way of directly creating visualizations on top of dataframes.

Convenience: Minimalistic language design is intended to alleviate the common challenge in exploratory analysis where users struggle to translate their high-level data questions to exact visualization specifications

Specifying Intent to Dataframe

Attaching Intent to DF

```
axis1 = lux.Clause(attribute="Age")  
axis2 = lux.Clause(attribute="Education")  
df.intent = [axis1,axis2]
```

```
df.intent = ["Age", "Education"]
```

```
axis = "Age"  
filter = "Department=Sales"  
df.intent = [axis, filter]
```

Constructing Visualizations

```
axis1 = lux.Clause(attribute="Age")  
axis2 = lux.Clause(attribute="Education")  
Vis([axis1,axis2],df)
```

Searching across attribute space

Q6. Browse through relationships between any two quantitative columns in the dataframe.

```
any = lux.Clause("?",data_type = "quantitative")  
VisList([any, any],df)
```

Specification Comparison

```
axis = "Age"  
filter = "Department=Sales"  
df.intent = [axis, filter]
```

Required Specification	
Legend: Data source (grey), Field Name (purple), Data operation (red), Channel (cyan), Field Type (yellow), Mark (green)	
Intent <code>Vis(["Age", "Education"], df)</code> <i>Lux</i>	Imperative <code>bar=df.groupby("Education").mean()["Age"] y_pos=range(len(bar)) plt.barh(y_pos, bar, align='center') plt.yticks(y_pos, list(bar.index)) plt.xlabel('Mean of Age') plt.ylabel('Education')</code> <i>matplotlib</i>
Declarative <pre>{ "mark": "bar", "data": {...}, "encoding": { "x": { "type": "quantitative", "field": "Age", "aggregate": "average" }, "y": { "type": "nominal", "field": "Education", } } }</pre> <i>Vega-Lite</i>	Partial <pre>data(...). encoding(e0). channel(e0,x). type(e0,quantitative). field(e0,"Age"). aggregate(e0,mean). encoding(e1). channel(e1,y). type(e1,nominal). field(e1,"Education").</pre> <i>Draco</i>

Figure 6: Comparison between the level of specification required from Lux versus other existing approaches for Query 3.

System generated Recommendation

We can attach an intent to a dataframe, or this intent can be **programmatically generated** as part of Lux's recommendations.

Programmatically, two types can be there:

1. Structure-based recommendations.

- The dataframe **“structure” reveals strong signals** for what the users subsequently choose to visualize, thus providing implicit information on what recommendations to display automatically by Lux

2. History-based recommendations.

- Lux displays **history-based recommendations** based on whether the dataframe has been filtered or aggregated in its recent history

System Architecture

1. A **client-server model**.
2. **LuxDataFrame**: a wrapper around pandas, and supports all pandas operations, while storing additional information, such as the intent, metadata, structure, and history, for generating visual recommendations.
3. Design is **modular and extensible** so that alternatives can be swapped in at different layers, e.g., Altair and Matplotlib visualization rendering libraries.

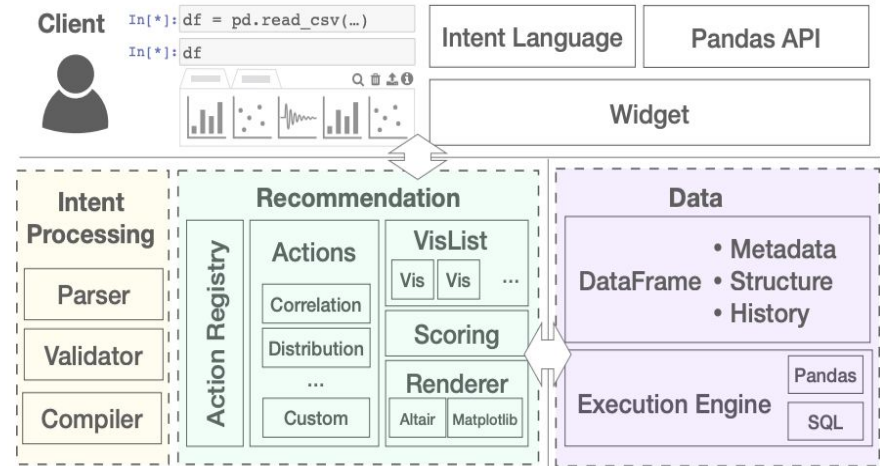


Figure 8: System architecture for Lux

Execution & Optimization

1. Metadata Computation

- a. It includes **attribute level statistics and data types**. The statistics include the list of unique values, cardinality, and min/max values for each attribute

2. Visualization Processing

- a. Execution engine translates each visualization to queries responsible for processing the data required for the visualizations. **It is time consuming so techniques are followed like below.**

3. Workflow Based Optimizations(WFLOW)

- a. **Lazily** compute the metadata and recommendations when users explicitly print data.
- b. cache and reuse results later on in the session.

Execution & Optimization

4. Approximate Pruning of Search Space (PRUNE)

- a. Each visualization in an action is ranked based **on a scoring function**, computed based on the data associated with each visualization.
- b. Approximate query processing to **reduce the cost** by estimating the scores using sampled data
- c. Optimization only performed when: $N \times t_{exact} \gg N \times t_{approx} + k \times t_{exact}$
where t_{exact} and t_{approx} are the cost of computing the exact and approximate scores.

5. Cost Based Scheduling of Actions (ASYNC)

- a. Scheduling the cheapest action to compute first, followed by computing the remaining in the background.
- b. The async optimization provides users with early results and returns interactive control back to the user, instead of incurring a high wait time during their analysis session

Contributions and Impact

- The idea of exploiting asynchronous execution during user wait time has been well-established but it is the **first to apply** this technique in a visualization recommendation context.
- **Lazy computation and caching** and **reuse** are well-studied but identifying usage patterns and determining to expire metadata are novel.
- Leveraging cost estimates to prioritize **cheaper-to-compute visualizations** is also novel.
- The cost model across different visualization types is an **independent valuable** contribution.

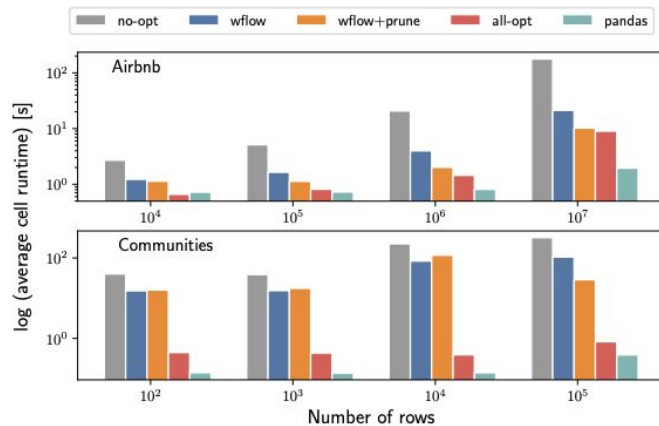
Performance Evaluation

Datasets

Airbnb(12 columns) and Community dataset(128 columns)

- **no-opt**: Baseline condition with no optimization applied, representing a naive implementation of LUX where the results are explicitly computed at the end of every cell involving a reference to the dataframe⁴.
- **wflow**: Condition with the WFLOW optimization applied.
- **wflow+prune**: Both WFLOW and PRUNE applied.
- **all-opt**: All WFLOW, PRUNE, and ASYNC applied, representing the best achievable performance.
- **pandas**: Condition with only pandas and *without* using LUX, representing the raw performance of dataframe workflows without the benefits of always-on visualizations.

	Airbnb		Communities		Distr.
	N	overhead [s]	N	overhead [s]	
Print df	14	21.18	14	1.41	
Print Series	7	0.61	4	0.07	
Non-Lux	17	0	25	0	



Field Study and Evaluation

- Study did with Industry Data Scientists, 3 participants.
- Reviews:

it really helps speed up my exploratory analysis. If not, it will take me forever to go through these many variables

Summary:

1. Average System Usability Scale (SUS) score across participants is 70/100.
2. All three participants were interested in continuing to use Lux in their data science work
3. Participants are still attached to their existing visualization tool for this functionality.
4. Concerns around customizability and the inability to express their desired visualizations, need for improving the flexibility of the intent language.