LUX Always-on Visualization Recommendations

Doris Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang,
Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, Aditya G. Parameswaran

Berkeley
UNIVERSITY OF CALIFORNIA

Ponder

pandas: Swiss-Army Knife of Data Science

Rich API: 600+ functions for data loading, cleaning/preparation, and analysis

pd.read_csv(...)

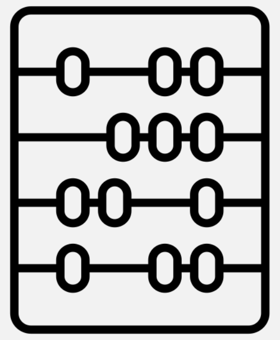
pd.read_json(...)

pd.read_excel(...)

df.dropna(...)

df.fillna(...)

pd.get_dummies(...)

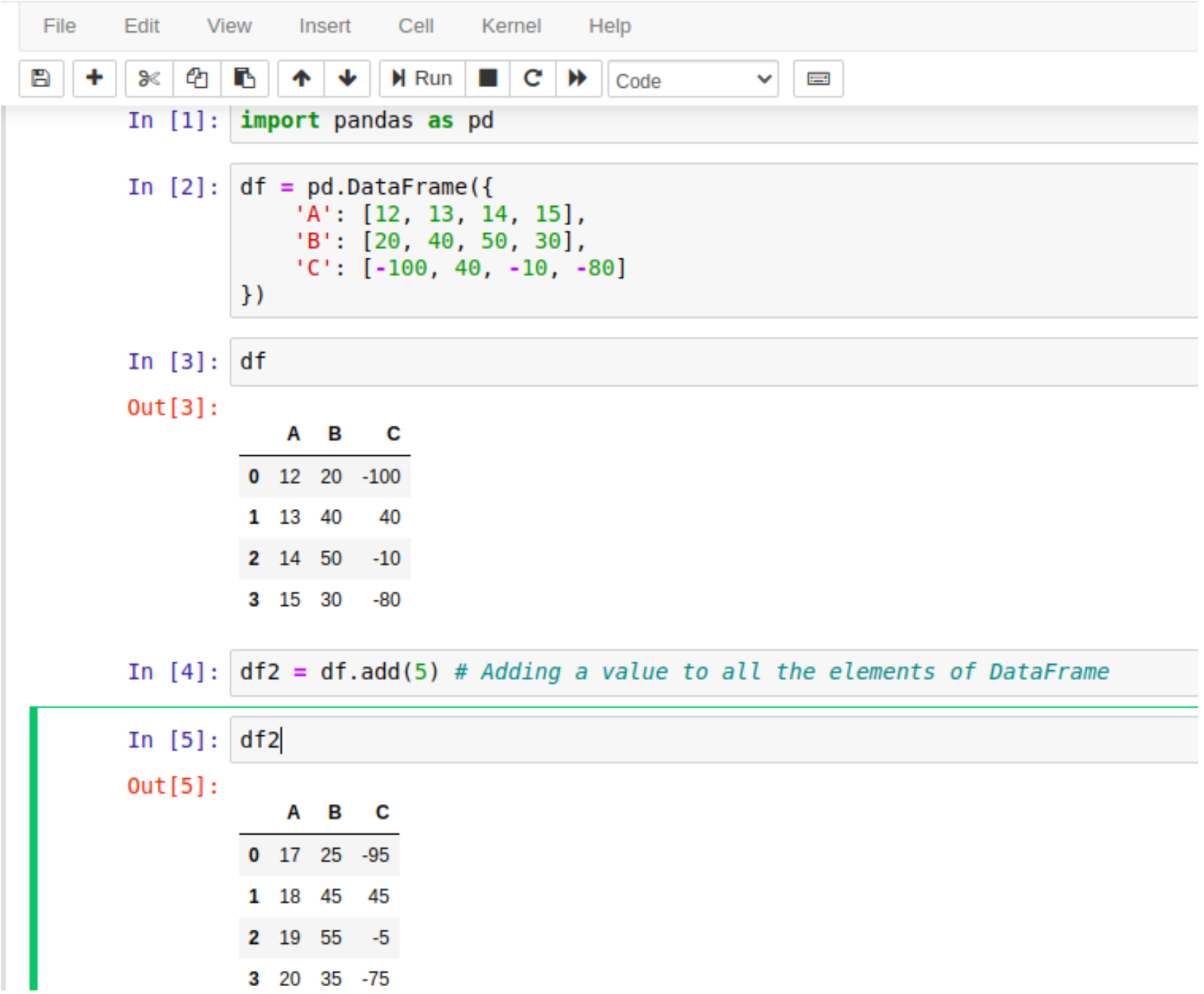


pd.join(...)

df.transpose(...)

df.pivot(...)

Interoperability: Easy to integrate with existing data tools, e.g., Jupyter Notebook



File Edit View Insert Cell Kernel Help

Run Code

In [1]:
```
import pandas as pd
```

In [2]:
```
df = pd.DataFrame({
    'A': [12, 13, 14, 15],
    'B': [20, 40, 50, 30],
    'C': [-100, 40, -10, -80]
})
```

In [3]:
```
df
```

Out[3]:

	A	B	C
0	12	20	-100
1	13	40	40
2	14	50	-10
3	15	30	-80

In [4]:
```
df2 = df.add(5) # Adding a value to all the elements of DataFrame
```

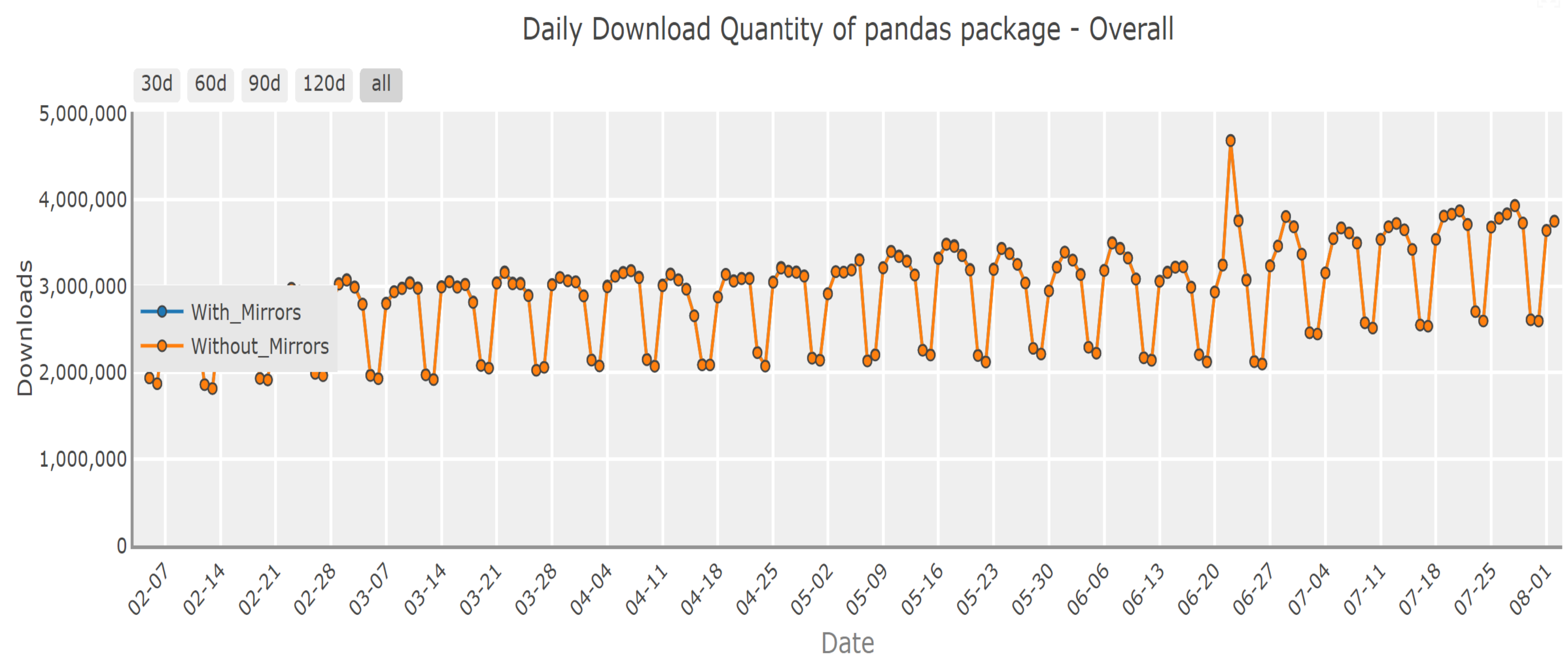
In [5]:
```
df2
```

Out[5]:

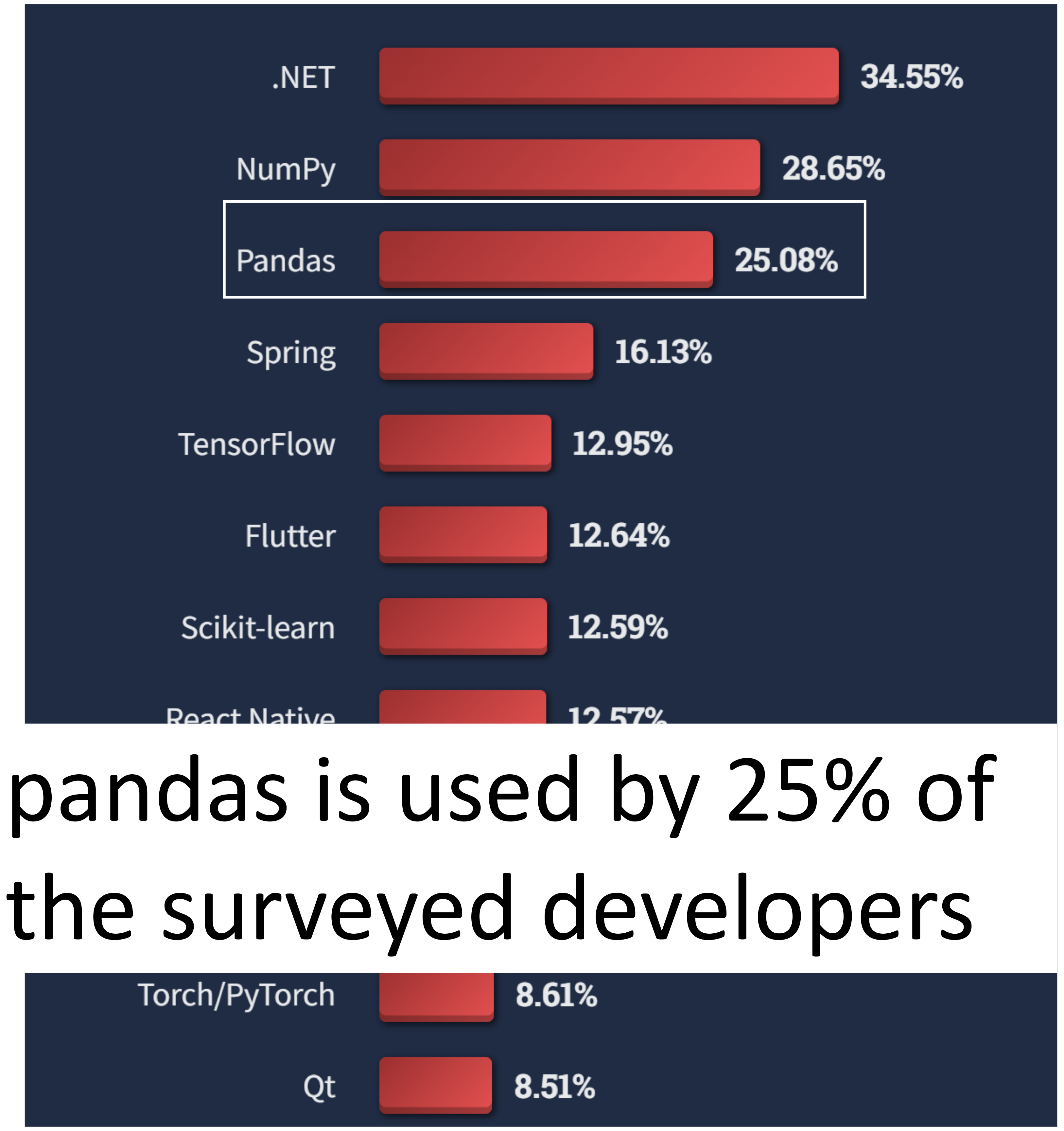
	A	B	C
0	17	25	-95
1	18	45	45
2	19	55	-5
3	20	35	-75

Berkeley UNIVERSITY OF CALIFORNIA

People Love pandas



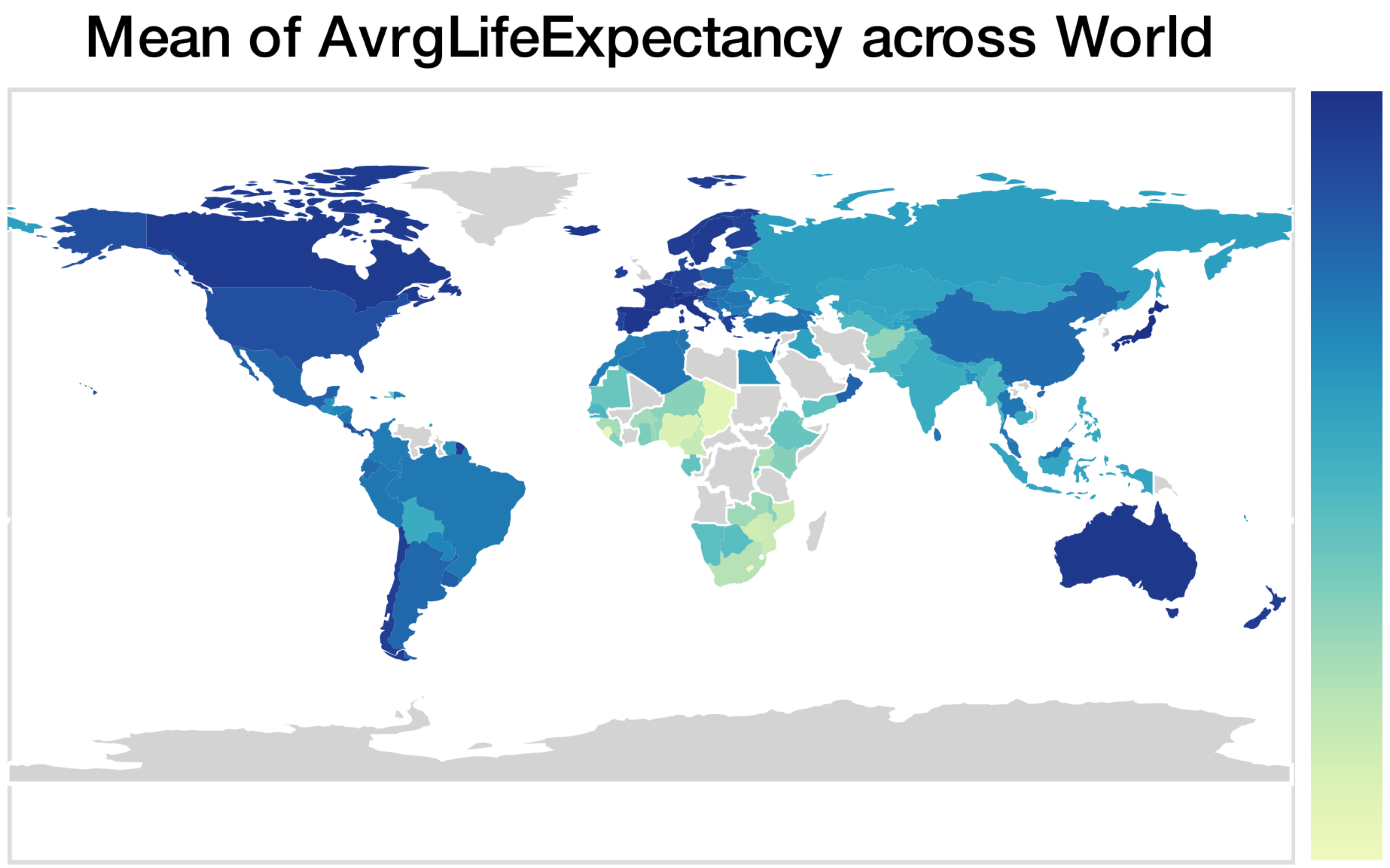
3M+ daily downloads



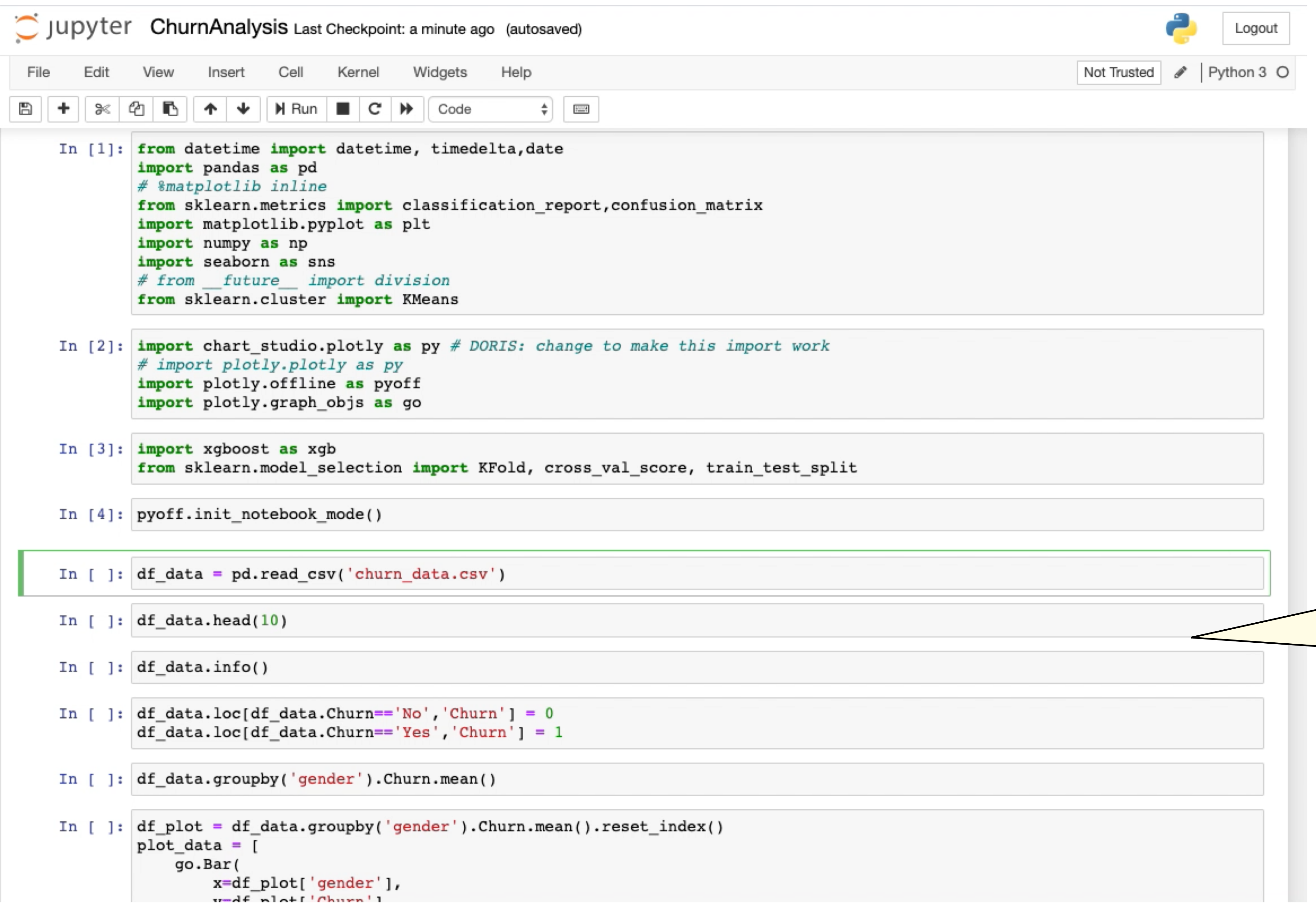
pandas is used by 25% of the surveyed developers

Visualizations Come in Handy when Exploring Data with pandas

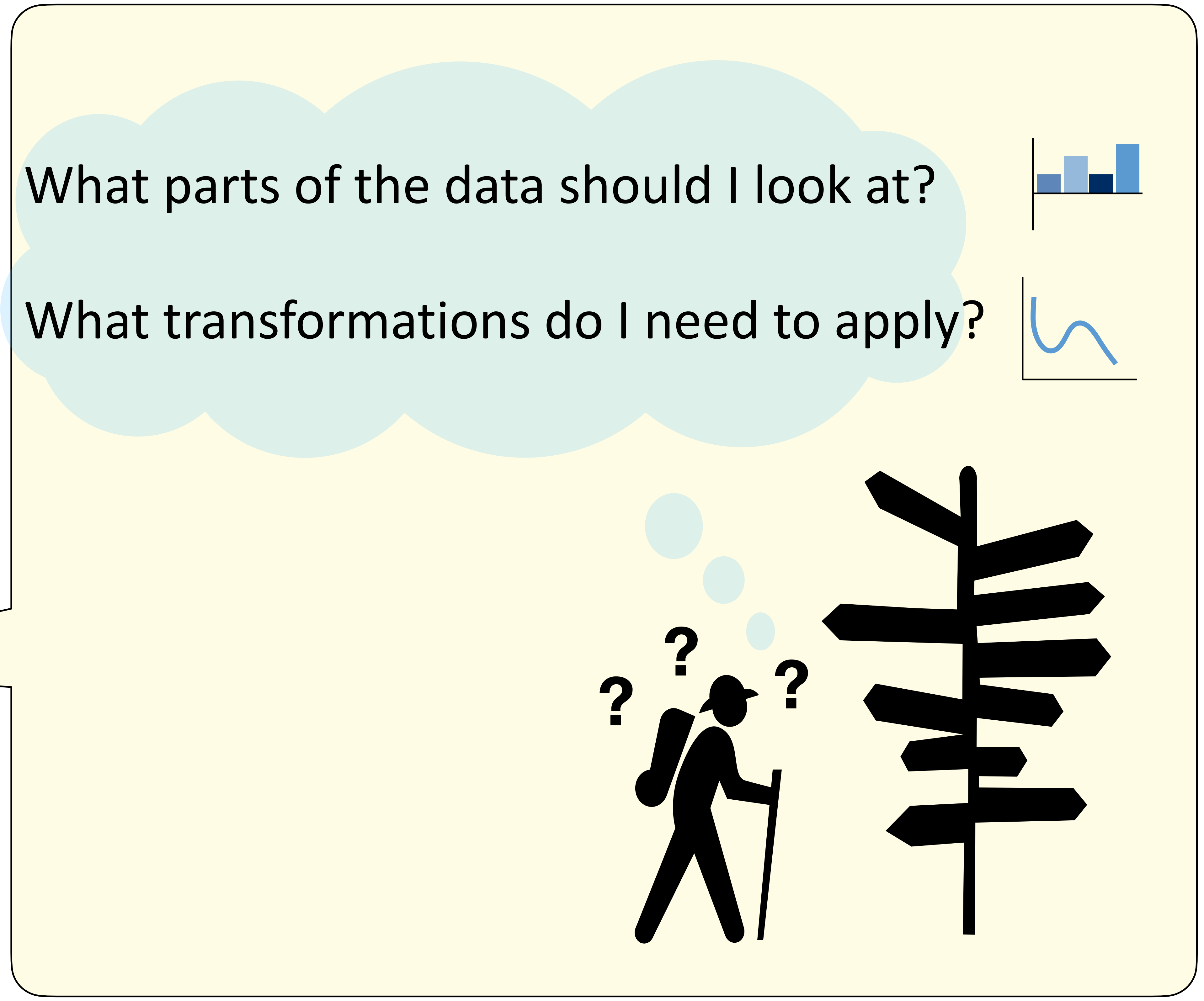
	HPIRank	Country	GeoPoliticalRegion	AvrgLifeExpectancy	AvrgWellBeing	HappyLifeYears
0	110	Afghanistan	Middle East and North Africa	59.7	3.8	12.4
1	13	Albania	Post-communist	77.3	5.5	34.4
2	30	Algeria	Middle East and North Africa	74.3	5.6	30.5
...	...	...	...	...	...	...



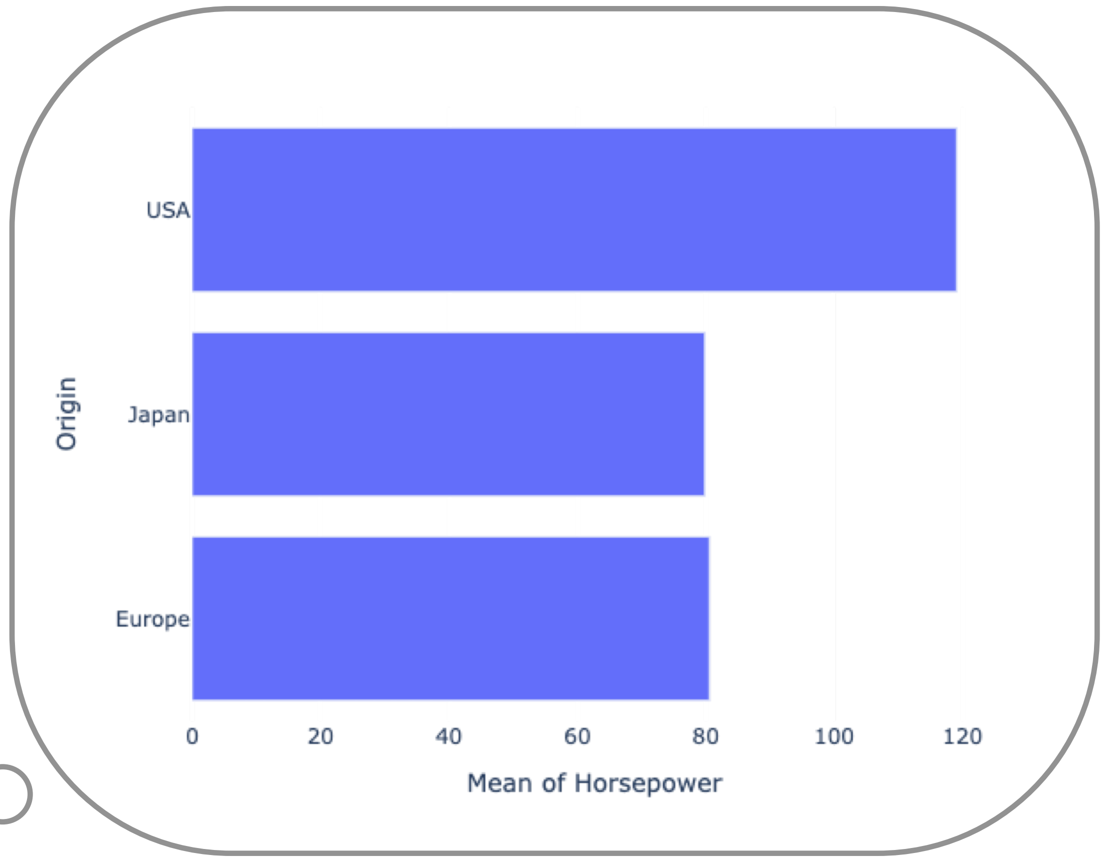
Preparing Data Involves a Substantial Amount of Code




```
In [1]: from datetime import datetime, timedelta,date
        import pandas as pd
        # %matplotlib inline
        from sklearn.metrics import classification_report,confusion_matrix
        import matplotlib.pyplot as plt
        import numpy as np
        import seaborn as sns
        # from __future__ import division
        from sklearn.cluster import KMeans

In [2]: import chart_studio.plotly as py # DORIS: change to make this import work
        # import plotly.plotly as py
        import plotly.offline as pyoff
        import plotly.graph_objs as go

In [3]: import xgboost as xgb
        from sklearn.model_selection import KFold, cross_val_score, train_test_split

In [4]: pyoff.init_notebook_mode()

In [ ]: df_data = pd.read_csv('churn_data.csv')

In [ ]: df_data.head(10)

In [ ]: df_data.info()

In [ ]: df_data.loc[df_data.Churn=='No','Churn'] = 0
        df_data.loc[df_data.Churn=='Yes','Churn'] = 1

In [ ]: df_data.groupby('gender').Churn.mean()

In [ ]: df_plot = df_data.groupby('gender').Churn.mean().reset_index()
        plot_data = [
            go.Bar(
                x=df_plot['gender'],
```



Creating Many Visualizations Involves Much Code




```python
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv("data/cars.csv")
barVal = df.groupby("Origin").mean()["Horsepower"]
y_pos = range(len(barVal))
plt.barh(y_pos,barVal, align='center', alpha=0.5)
plt.yticks(y_pos,list(barVal.index))
plt.xlabel('Mean of Horsepower')
plt.ylabel('Origin')
plt.show()
```

matplotlib

```python
import plotly.graph_objects as go
import pandas as pd
df = pd.read_csv("data/cars.csv")
barVal = df.groupby("Origin").mean()["Horsepower"]
fig = go.Figure(go.Bar(
            x= barVal,
            y= barVal.index,
            orientation='h'))
fig.update_layout(
    xaxis_title="Mean of Horsepower",
    yaxis_title="Origin",
)
fig.show()
```

plotly

What should my visualizations look like?

What encodings or chart types should I pick?

"over 20%—the majority—of duplicated code in notebooks is vis code"

[Koenzen et al. VL/HCC 2020]

users only visualize during "the late stages of their workflow"

[Batch et al. TVCG 2018]

LUX An Always-On Dataframe Visualization Tool [VLDB'22]

Star 4.2k

400k+ downloads

Lux: Always-on Visualization Recommendations for Exploratory Dataframe Workflows

Doris Jung-Lin Lee, Dixin Tang, Kunal Agarwal, Thyne Boonmark, Caitlyn Chen, Jake Kang,
Ujjaini Mukhopadhyay, Jerry Song, Micah Yong, Marti A. Hearst, Aditya G. Parameswaran
UC Berkeley
{dorislee,totemtang,kagarwal2,thyneboonmark,caitlynachen,cjache,ujjaini,jerrysong,micahtyong,hearst,adityagp}@berkeley.edu

ABSTRACT

Exploratory data science largely happens in computational notebooks with dataframe APIs, such as pandas, that support flexible means to transform, clean, and analyze data. Yet, visually exploring data in dataframes remains tedious, requiring substantial programming effort for visualization and mental effort to determine what analysis to perform next. We propose LUX, an *always-on* framework for accelerating visual insight discovery in dataframe workflows. When users print a dataframe in their notebooks, LUX recommends visualizations to provide a quick overview of the pat-

the fluid, iterative process of data science, for two reasons: cumbersome boilerplate code and challenges in determining the next steps.

Cumbersome Boilerplate Code. Substantial boilerplate code is necessary to simply generate a visualization from dataframes. In a formative study, we analyzed a sample of 587 publicly-available notebooks from Rule et al. [63] to understand current visualization practices. A surprising number of notebooks apply a series of *data processing* operations to wrangle the dataframe into a form amenable to visualization, followed by a set of highly-templatized

Used by many data practitioners

Technical University of Denmark

NCM
National Center of Meteorology

UNIVERSITY OF MARYLAND

Key Challenges

- How do we display visual recommendations to users in a seamless manner?
- What recommendations should we show to advance analysis?
- How do we allow users to steer recommendations?
- How do we return results in a reasonable amount of time?

Key Challenges

- How do we display visual recommendations to users in a seamless manner?
- What recommendations should we show to advance analysis?
- How do we allow users to steer recommendations?
- How do we return results in a reasonable amount of time?

Berkeley
UNIVERSITY OF CALIFORNIA

Always-on Dataframe Visualization Beyond Tables

```
import lux
import pandas as pd
```

```
df = pd.read_excel("data.xls")
df
```

```
df = df[df["date"]>"2020-03"]
df
```

```
cleaned = df.dropna()
cleaned
```

```
df
```

Toggle Pandas/Lux

	Entity	Code	Day	stringency	Country	Region	AvrgLifeExpectancy	AvrgWellBeing	HappyLifeYears	Footprint	Inequality	HappyPlanetIndex
0	Afghanistan	AFG	2020-03-11	27.78	Afghanistan	Middle East	59.7	3.8	12.4	0.8	0.43	20.2
1	Albania	ALB	2020-03-11	51.85	Albania	Post-communist	77.3	5.5	34.4	2.2	0.17	36.8
2	Algeria	DZA	2020-03-11	16.67	Algeria	Middle East	74.3	5.6	30.5	2.1	0.24	33.3
3	Argentina	ARG	2020-03-11	25.00	Argentina	Americas	75.9	6.5	40.2	3.1	0.16	35.2
4	Australia	AUS	2020-03-11	19.44	Australia	Asia Pacific	82.1	7.2	53.1	9.3	0.08	21.2
...	...	...	...	...	...	...	...	...	...	...	...	...
124	Venezuela	VEN	2020-03-11	11.11	Venezuela	Americas	73.9	7.1	41.5	3.6	0.19	33.6
125	Vietnam	VNM	2020-03-11	47.22	Vietnam	Asia Pacific	75.5	5.5	32.8	1.7	0.19	40.3
126	Yemen	YEM	2020-03-11	0.00	Yemen	Middle East	63.3	4.1	15.2	1.0	0.39	22.8

Launcher | hpi_covid_demo.ipynb

Download GitHub Binder Code Python 3 (ipykernel)

In this demo, we will be exploring how world developmental indicators are related to a country's early effort in COVID-19 response.

Visualizations of dataframes beyond simple tables

To enable Lux, simply add `import lux` along with your Pandas import statement.

```
import lux
import pandas as pd
```

Lux preserves the Pandas dataframe semantics -- which means that you can apply any command from Pandas's API to the dataframes in Lux and expect the same behavior. For example, we can load the Happy Planet Index (HPI) dataset via standard Pandas `read_csv` command.

```
df = pd.read_csv("https://github.com/lux-org/lux-datasets/blob/master/data/hpi_cleaned.csv?raw=True")
```

We can quickly get an overview of the dataframe, simply by print out the dataframe `df`.

```
df
```

From the Pandas table view, we see that the dataframe contains country-level data on sustainability and well-being. By clicking on the Toggle button, you can now explore the data visually through Lux, you should see several tabs of visualizations recommended to you that includes scatterplots, bar charts, and maps. In Lux, we recommend visualizations that may be relevant or interesting to you across different actions, which are displayed as different tabs.

By inspecting the `Correlation` tab, we learn that there is a negative correlation between `AvrgLifeExpectancy` and `Inequality`. In other words, countries with higher levels of inequality also have a lower average life expectancy. We can also look at other tabs, which show the Distribution of quantitative attributes and the Occurrence of categorical attributes.

Steering analysis with intent

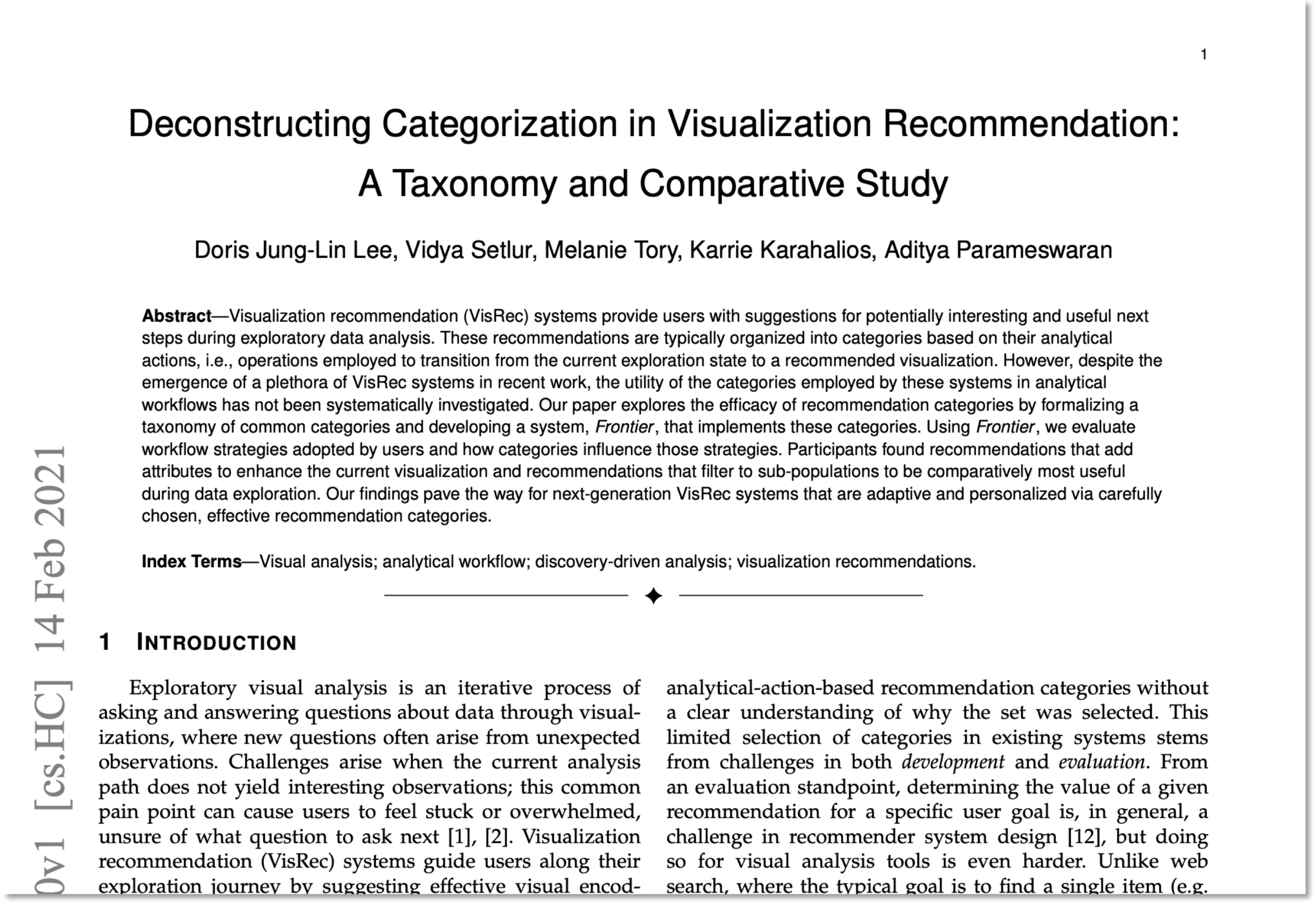
Let's say that we want to investigate whether any country-level characteristics explain the observed negative correlation between inequality and life expectancy. Beyond the basic recommendations, you can further specify your analysis *intent*, i.e., the data attributes and values that you are interested in visualizing.

We can do this by specifying our analysis intent to Lux via `df.intent`:

Key Challenges

- How do we display visual recommendations to users in a seamless manner?
- What recommendations should we show to advance analysis?
- How do allow users to express the intent with minimal efforts?
- How do we return results in a reasonable amount of time?

The Choice of Visualization Recommendations



1

Deconstructing Categorization in Visualization Recommendation:
A Taxonomy and Comparative Study

Doris Jung-Lin Lee, Vidya Setlur, Melanie Tory, Karrie Karahalios, Aditya Parameswaran

Abstract—Visualization recommendation (VisRec) systems provide users with suggestions for potentially interesting and useful next steps during exploratory data analysis. These recommendations are typically organized into categories based on their analytical actions, i.e., operations employed to transition from the current exploration state to a recommended visualization. However, despite the emergence of a plethora of VisRec systems in recent work, the utility of the categories employed by these systems in analytical workflows has not been systematically investigated. Our paper explores the efficacy of recommendation categories by formalizing a taxonomy of common categories and developing a system, *Frontier*, that implements these categories. Using *Frontier*, we evaluate workflow strategies adopted by users and how categories influence those strategies. Participants found recommendations that add attributes to enhance the current visualization and recommendations that filter to sub-populations to be comparatively most useful during data exploration. Our findings pave the way for next-generation VisRec systems that are adaptive and personalized via carefully chosen, effective recommendation categories.

Index Terms—Visual analysis; analytical workflow; discovery-driven analysis; visualization recommendations.

1 Introduction

Exploratory visual analysis is an iterative process of asking and answering questions about data through visualizations, where new questions often arise from unexpected observations. Challenges arise when the current analysis path does not yield interesting observations; this common pain point can cause users to feel stuck or overwhelmed, unsure of what question to ask next [1], [2]. Visualization recommendation (VisRec) systems guide users along their exploration journey by suggesting effective visual encod-

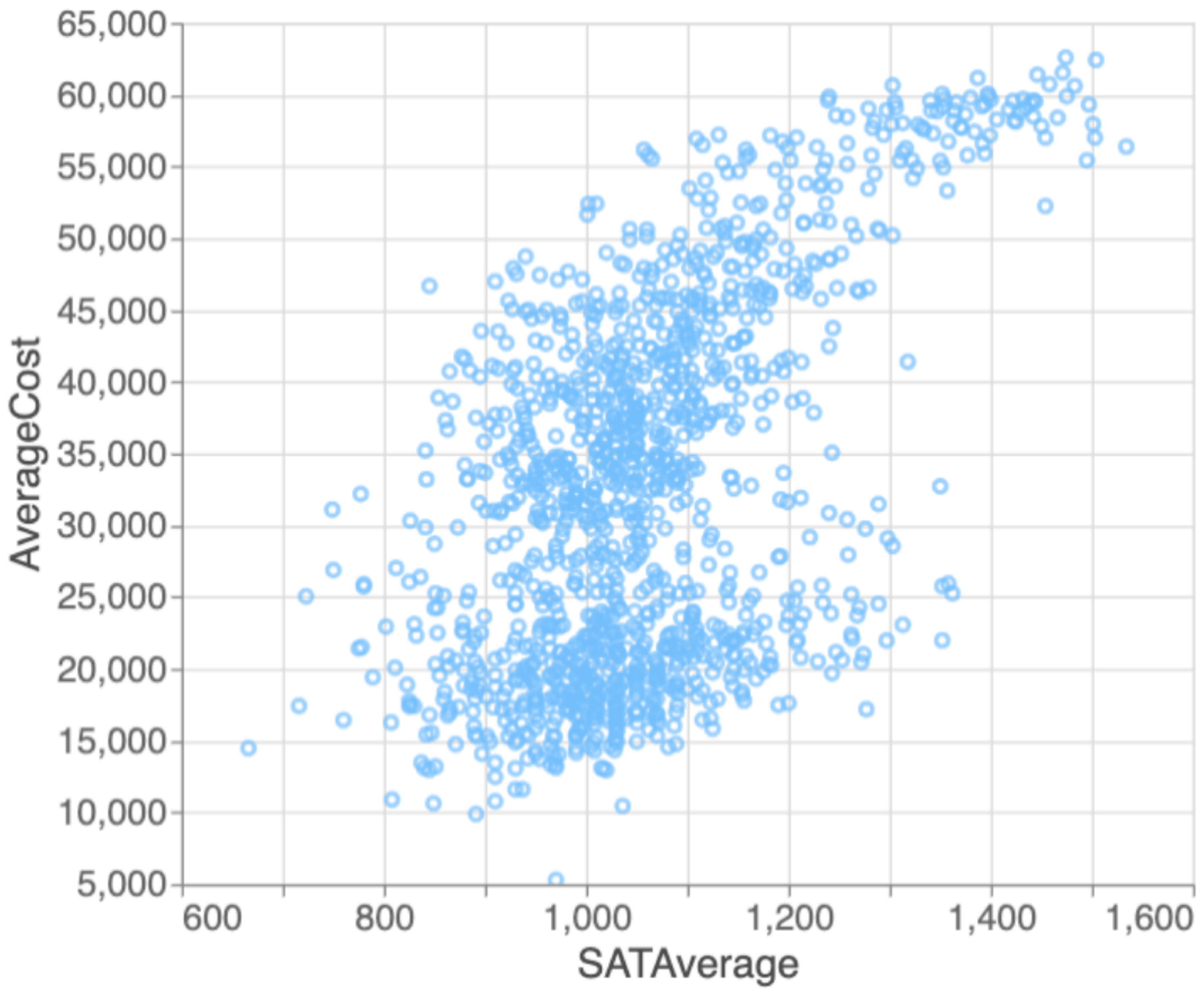
analytical-action-based recommendation categories without a clear understanding of why the set was selected. This limited selection of categories in existing systems stems from challenges in both *development* and *evaluation*. From an evaluation standpoint, determining the value of a given recommendation for a specific user goal is, in general, a challenge in recommender system design [12], but doing so for visual analysis tools is even harder. Unlike web search, where the typical goal is to find a single item (e.g.

To mirror human analyst behavior, visualization recommendations map to *analytical actions* from their current exploration state.

How do we represent the current state of exploration?

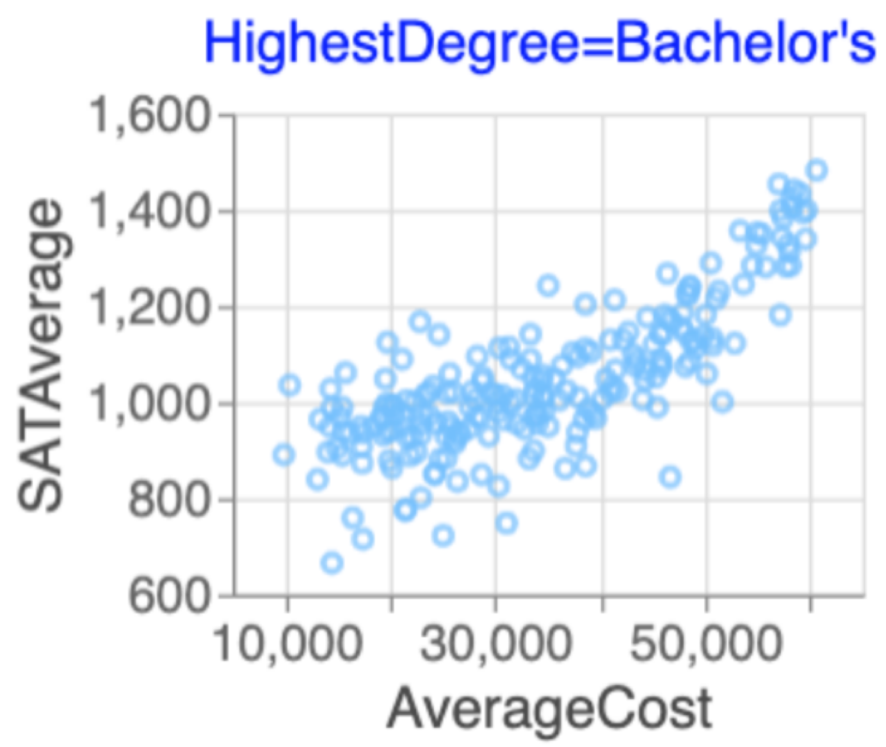
[*Attributes* being visualized, *Filters* being applied]

Transitions in the Visualization Space



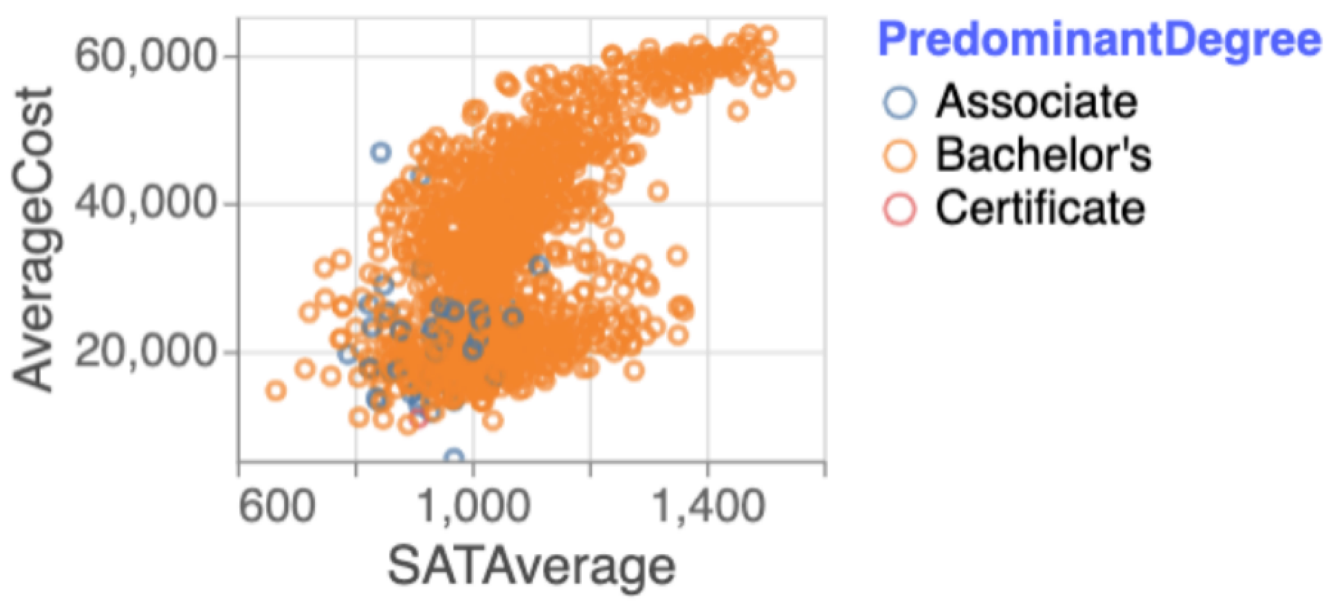
Attributes: {AverageCost, SATAverage}
Filters: { }

Transition in filter space



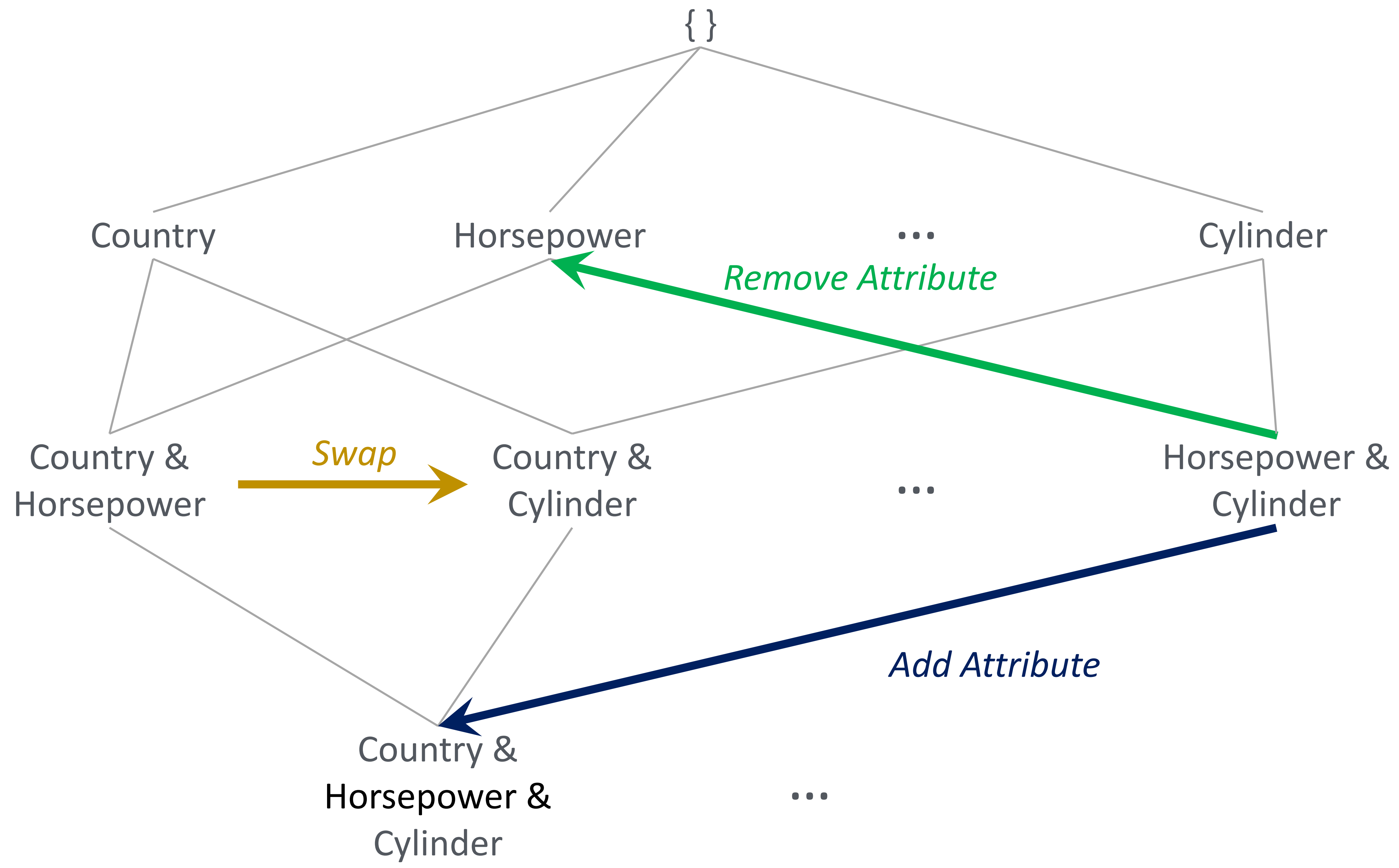
Attributes: {AverageCost, SATAverage}
Filters: {"HighestDegree=Bachelor's"}

Transition in attribute space

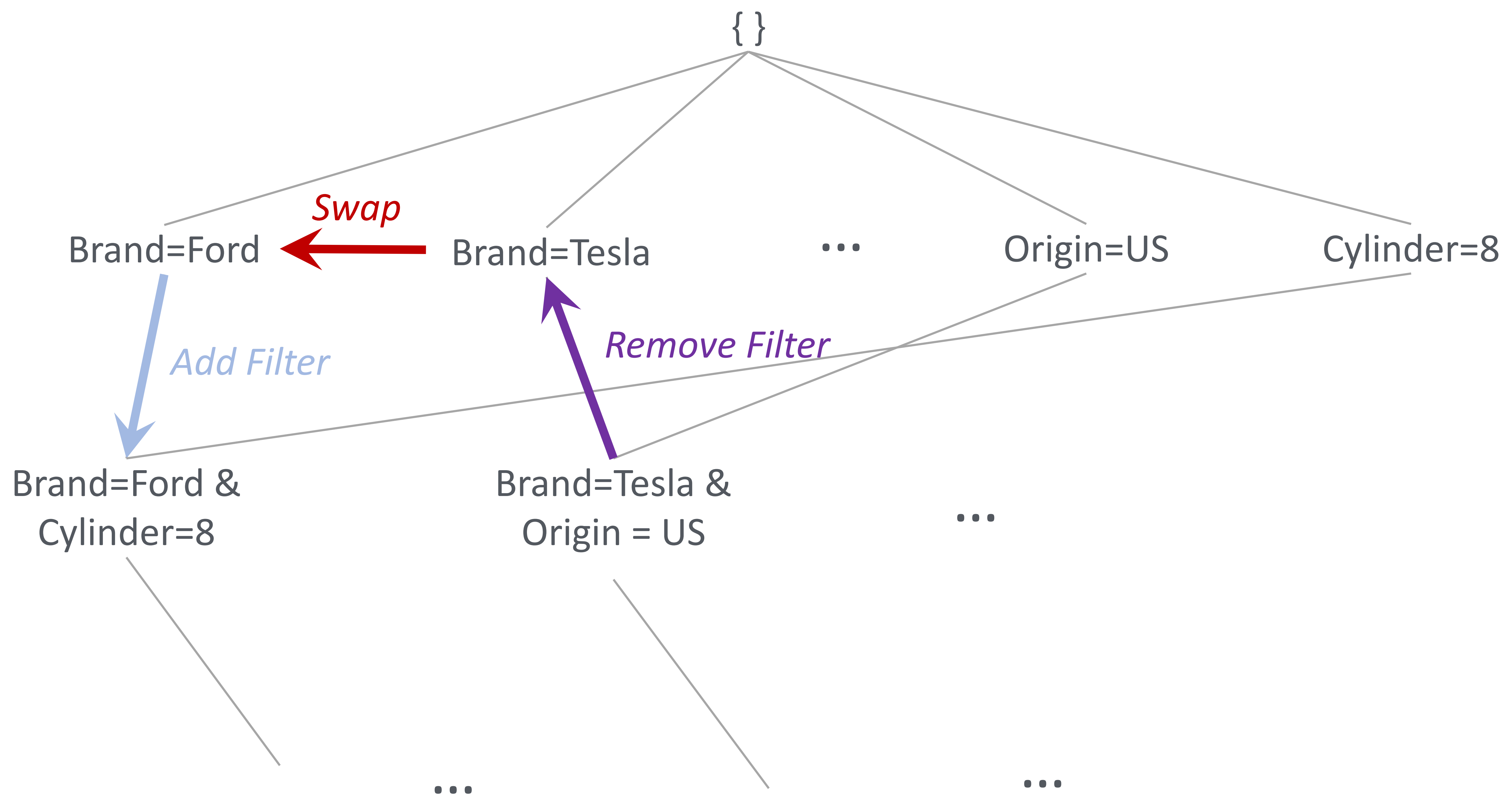


Attributes: {AverageCost, SATAverage, PredominantDegree}
Filters: { }

Space of Analytical Actions



Attribute Hierarchy



Filter Hierarchy

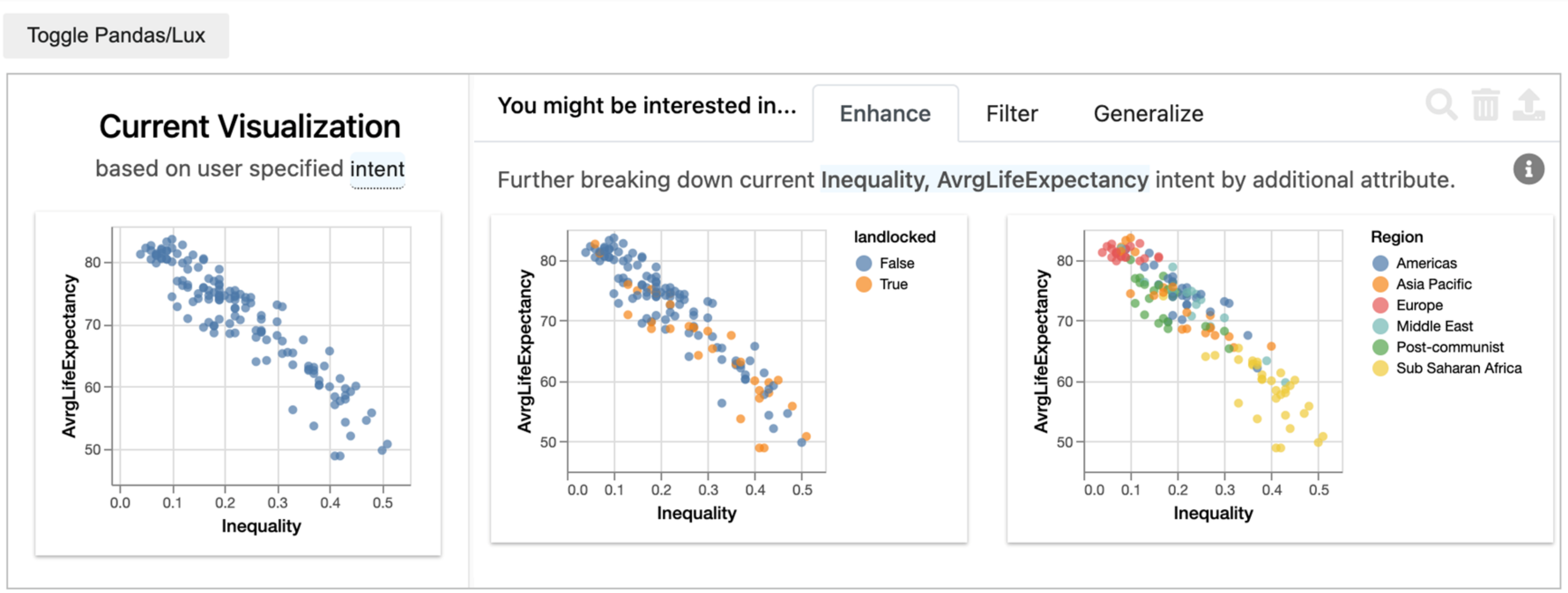
Key Challenges

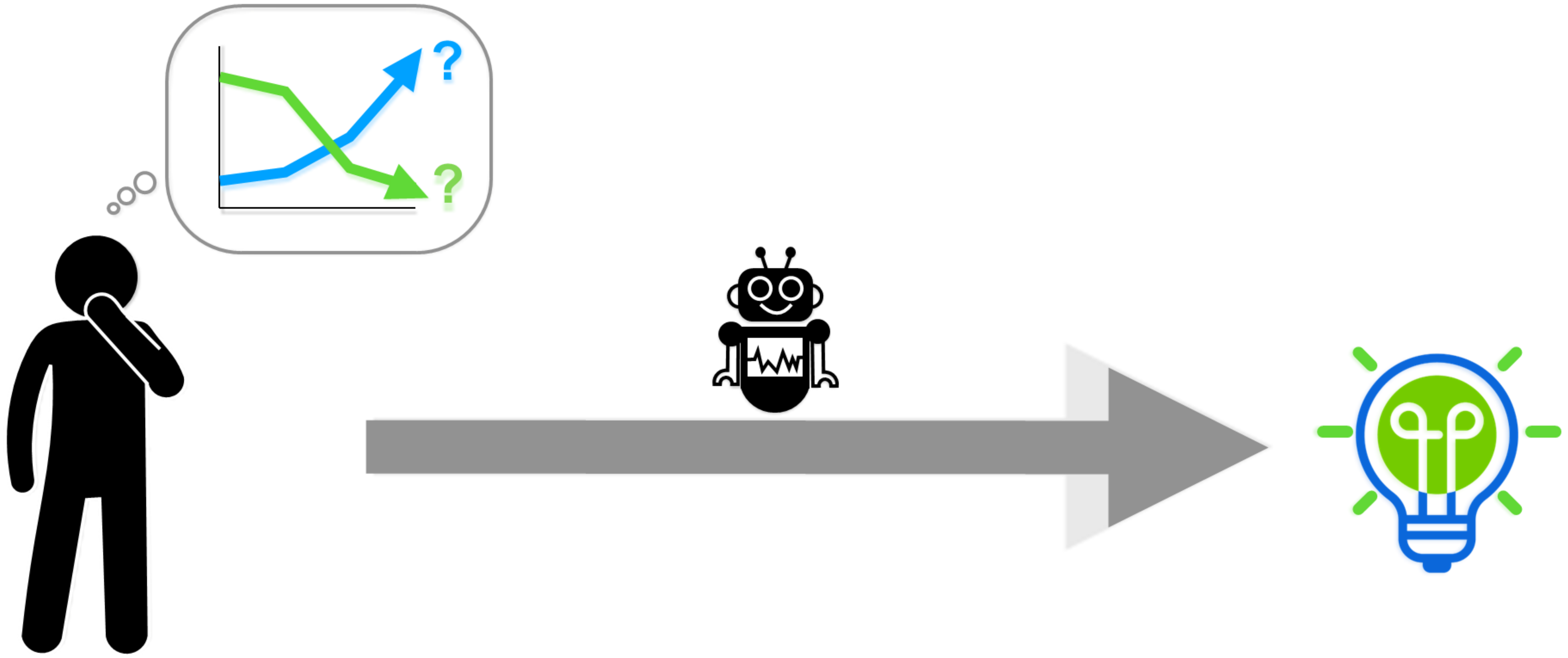
- How do we display visual recommendations to users in a seamless manner?
- What recommendations should we show to advance analysis?
- How do we allow users to steer recommendations?
- How do we return results in a reasonable amount of time?

Data-centric Intent

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

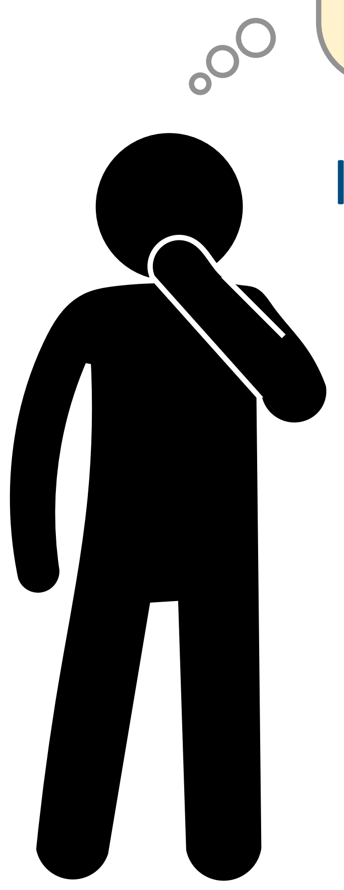
```
df
```





Key Insight: Empower users to specify data-centric aspects in a lightweight manner, with the system automatically filling in the "gaps"

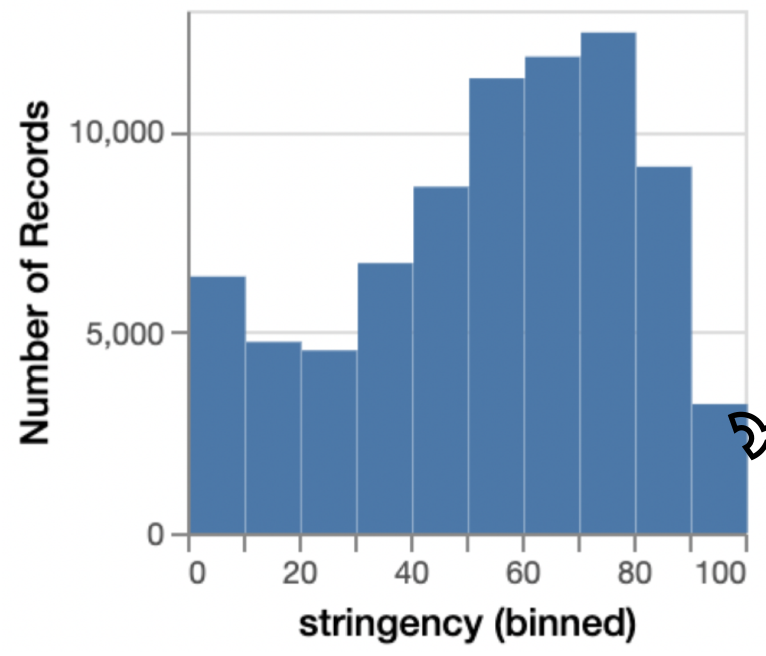
Creating Visualizations on Demand

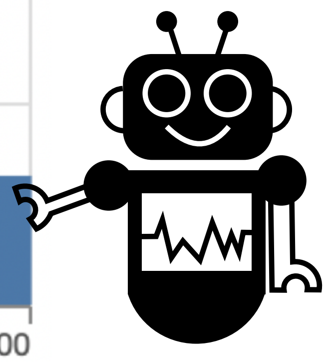


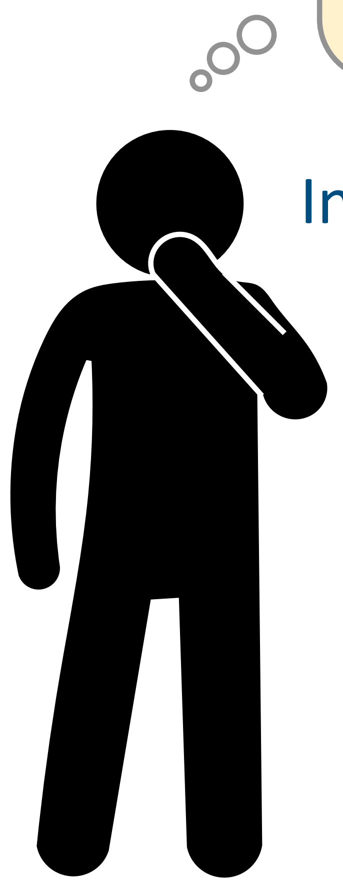
Let’s start by looking at stringency!

In[*]:

```
Vis([“stringency”],df)
```



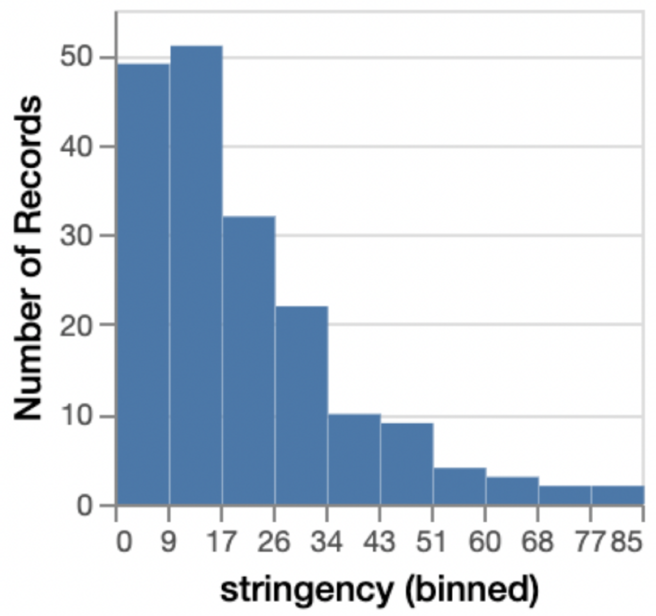


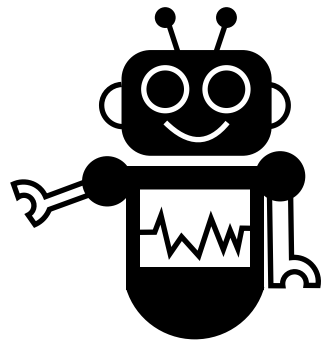


How did that change my chart?

In[*]:

```
Vis([“stringency”],df)
```

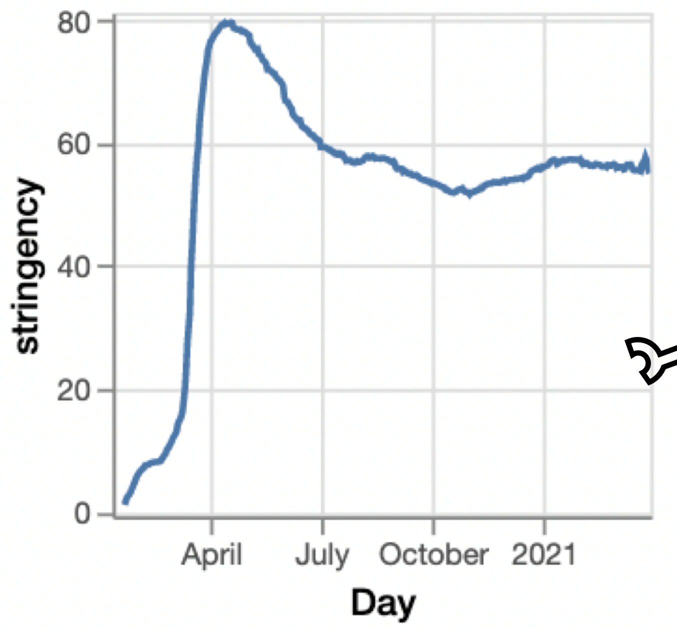


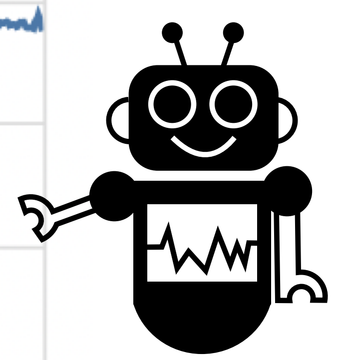


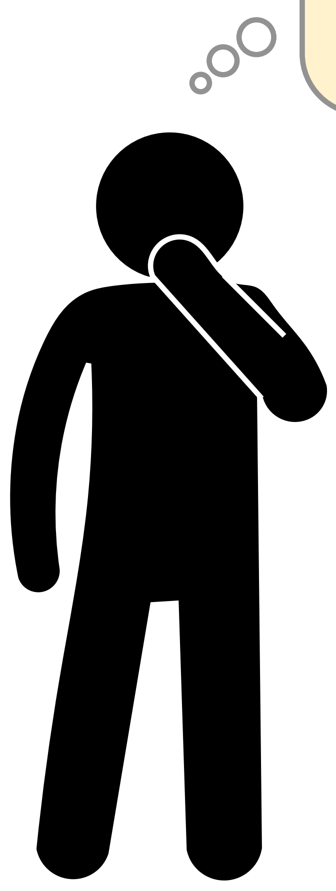
Ok, but how does it change over time?

In[*]:

```
Vis([“stringency”,”Day”],df)
```



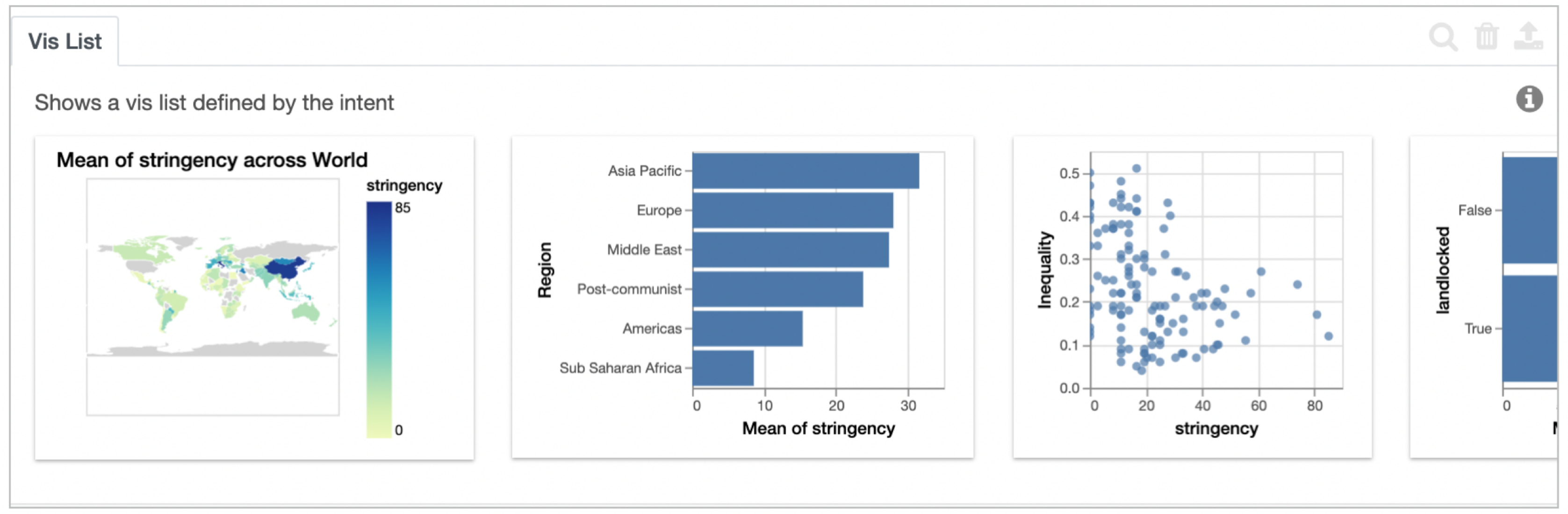


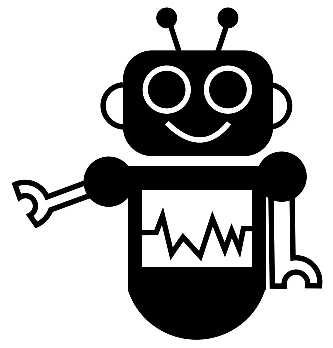


Show me other visualizations involving stringency

In[*]:

```
VisList([“?”,”stringency”],df)
```





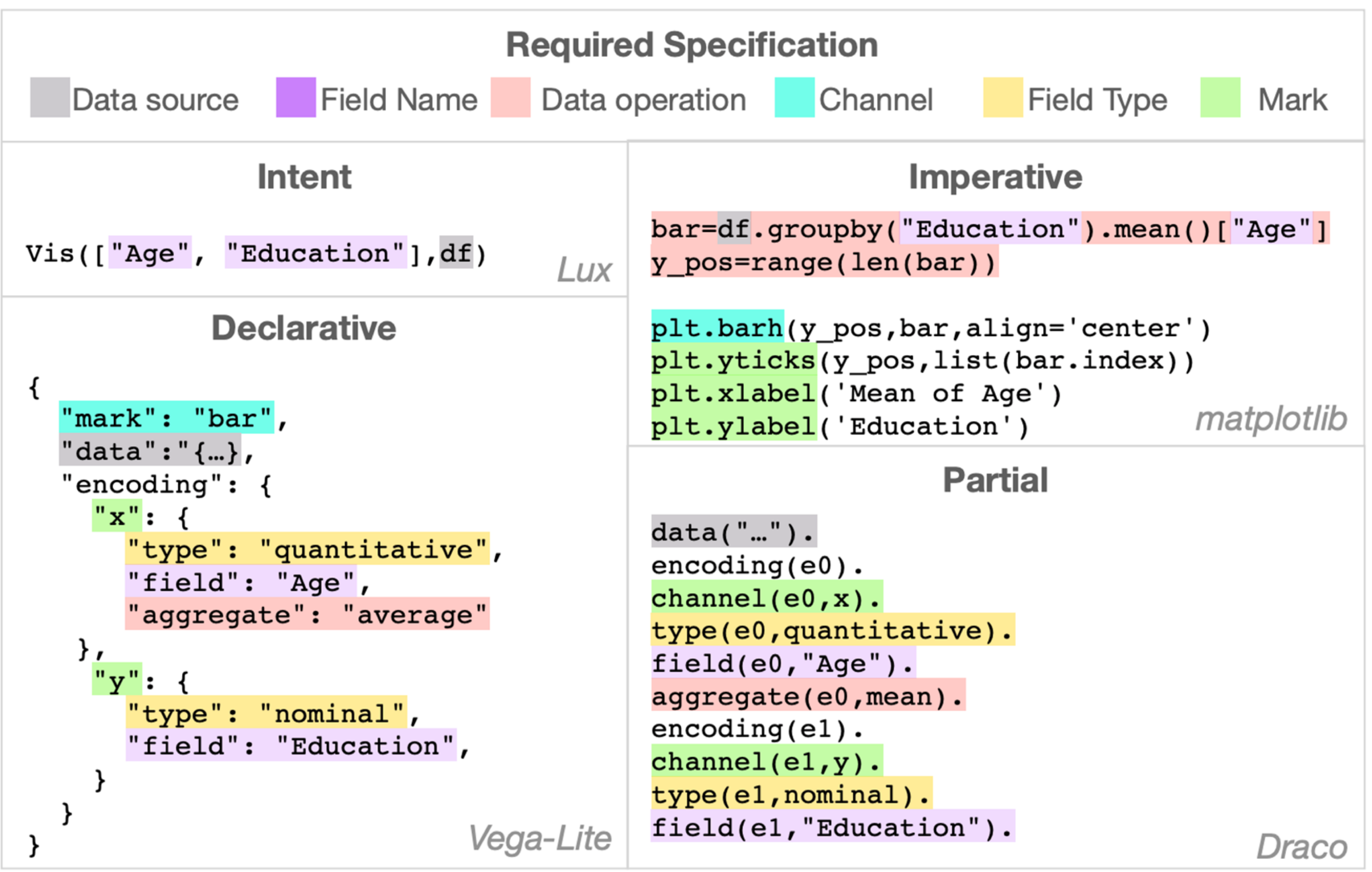
Berkeley
UNIVERSITY OF CALIFORNIA

Formal Basis and Comparison

Based on a formal regex language

$$\langle Intent \rangle \rightarrow \langle Clause \rangle^{+}$$
$$\langle Clause \rangle \rightarrow \langle Axis \rangle \mid \langle Filter \rangle$$
$$\langle Filter \rangle \rightarrow \langle attribute \rangle \; [=><\leq\geq\neq] \; \langle value \rangle$$
$$\langle attribute \rangle \rightarrow \text{attribute} \cup \langle attribute \rangle^{*} \mid ? \; \langle constraint \rangle$$
$$\langle value \rangle \rightarrow \text{value} \cup \langle value \rangle^{*} \mid ?$$

More succinct than other frameworks



Key Challenges

- How do we display visual recommendations to users in a seamless manner?
- What recommendations should we show to advance analysis?
- How do we allow users to steer recommendations?
- How do we return results in a reasonable amount of time?

Visualization Recommendation is Costly

Collecting metadata for deciding data types and chart types → Recommend visualizations

Min/max and cardinality for each unique value for each column

Computing the interestingness scores for candidate visualizations

System Optimizations

- Intelligent workflow-based optimizations (WFLOW)
- Approximate early pruning of search space (PRUNE)
- Cost-based scheduling of actions (ASYNC)

Intelligent workflow-based optimizations (WFLOW)

No optimization   WFLOW

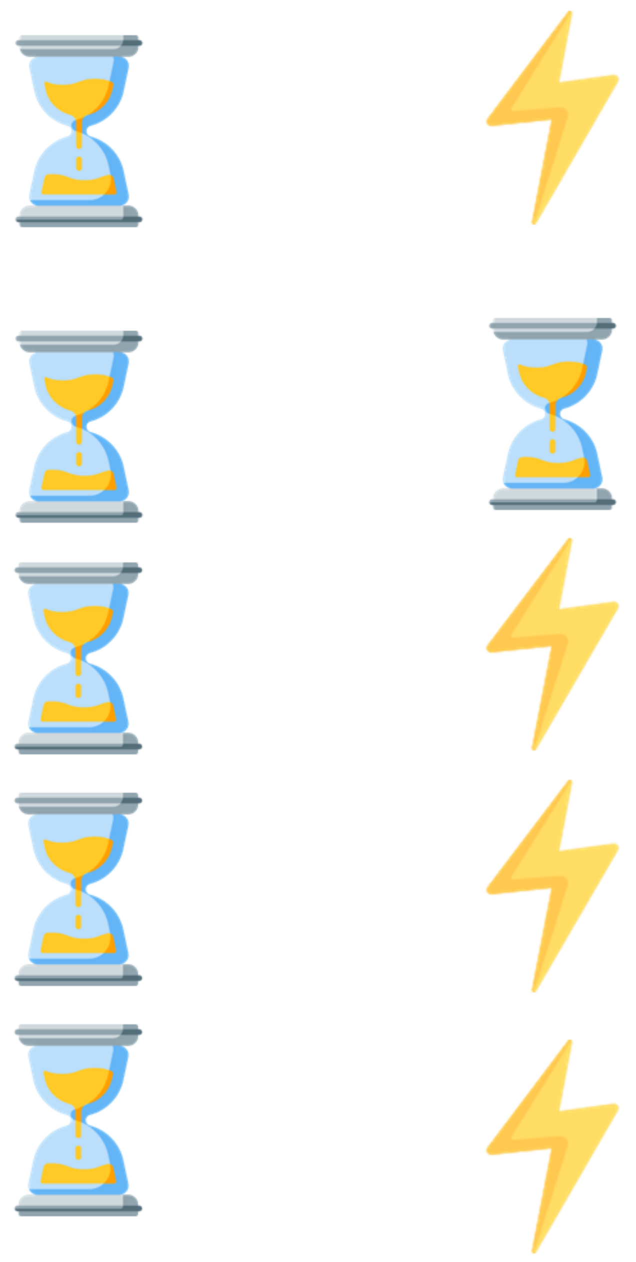
[1]
```
df['review_date'] = pd.to_datetime(df["review_date"],format="%Y")
df.drop(columns=['Unnamed: 0'],inplace=True)
```

[2]
```
df
```

[3]
```
df.groupby("company_location").mean()
```

[4]
```
df.info()
```

[5]
```
df
```



Optimizations

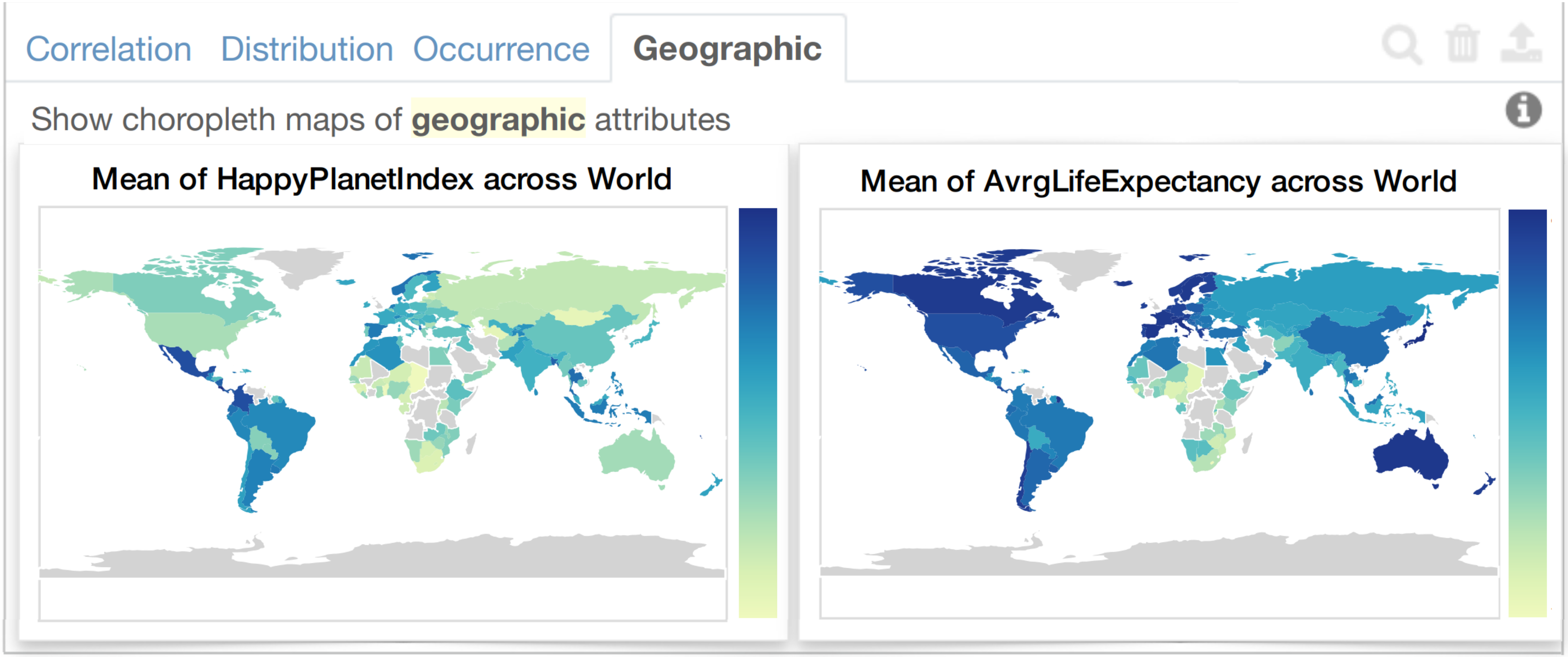
1) We can lazily compute only when users print df
2) We can cache and reuse results across the session (with careful accounting…)

Approximate early pruning of search space (PRUNE)

Collecting metadata for deciding data types and chart types → Recommend visualizations

Approximate the interestingness scores using sampled data

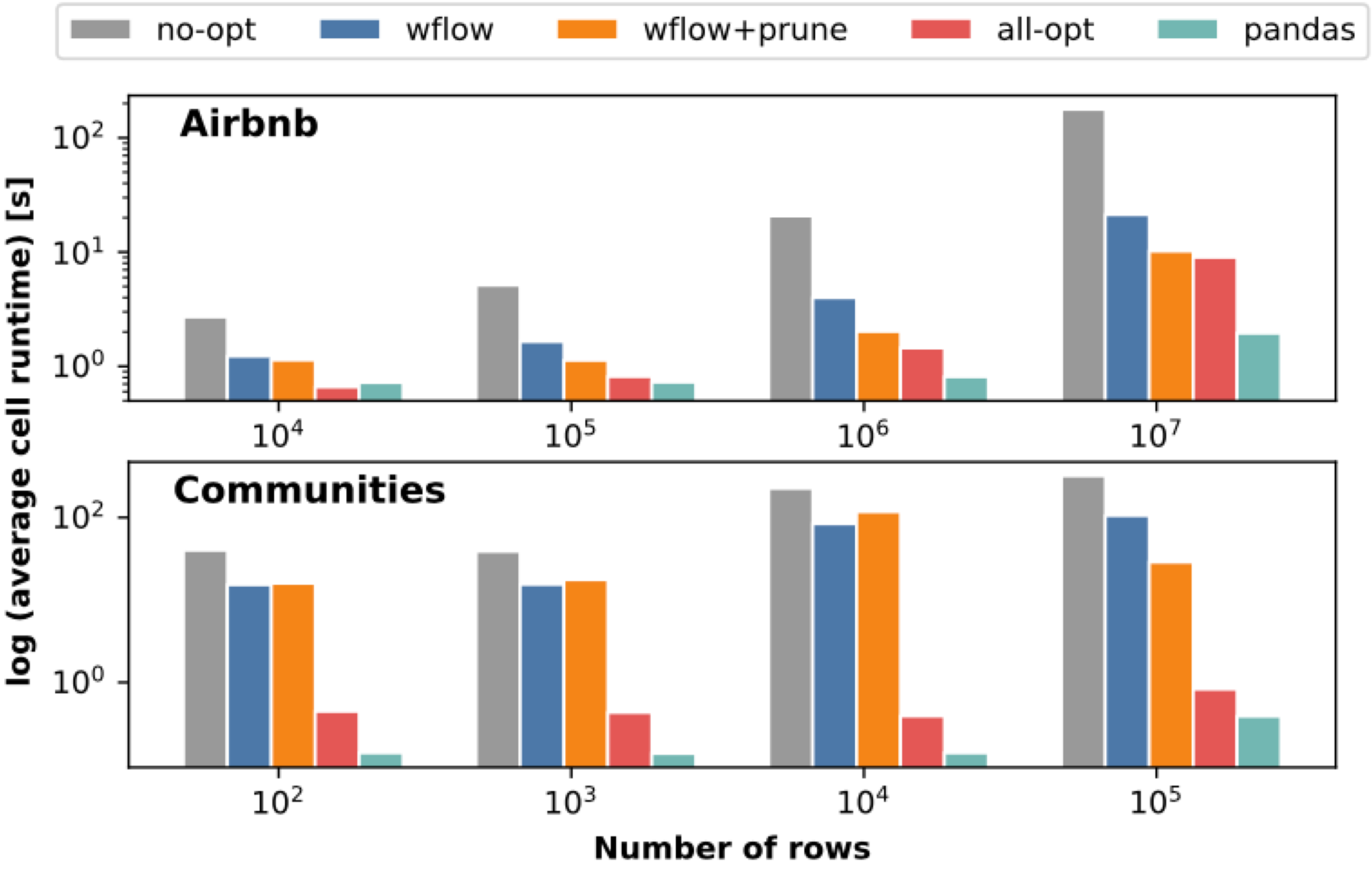
Cost-based scheduling of actions (ASYNC)



Experiment Evaluation

	Num of Columns	Num of Print df	Num of Print series
AirBnb	12	14	7
Communities	128	14	4

Experiment Evaluation



Berkeley
UNIVERSITY OF CALIFORNIA

Modin: A “drop-in” Scalable Replacement for pandas

```python
# import pandas as pd
import modin.pandas as pd
```

pandas API	SQL API	[Future] API

MODIN

🔥7k daily downloads

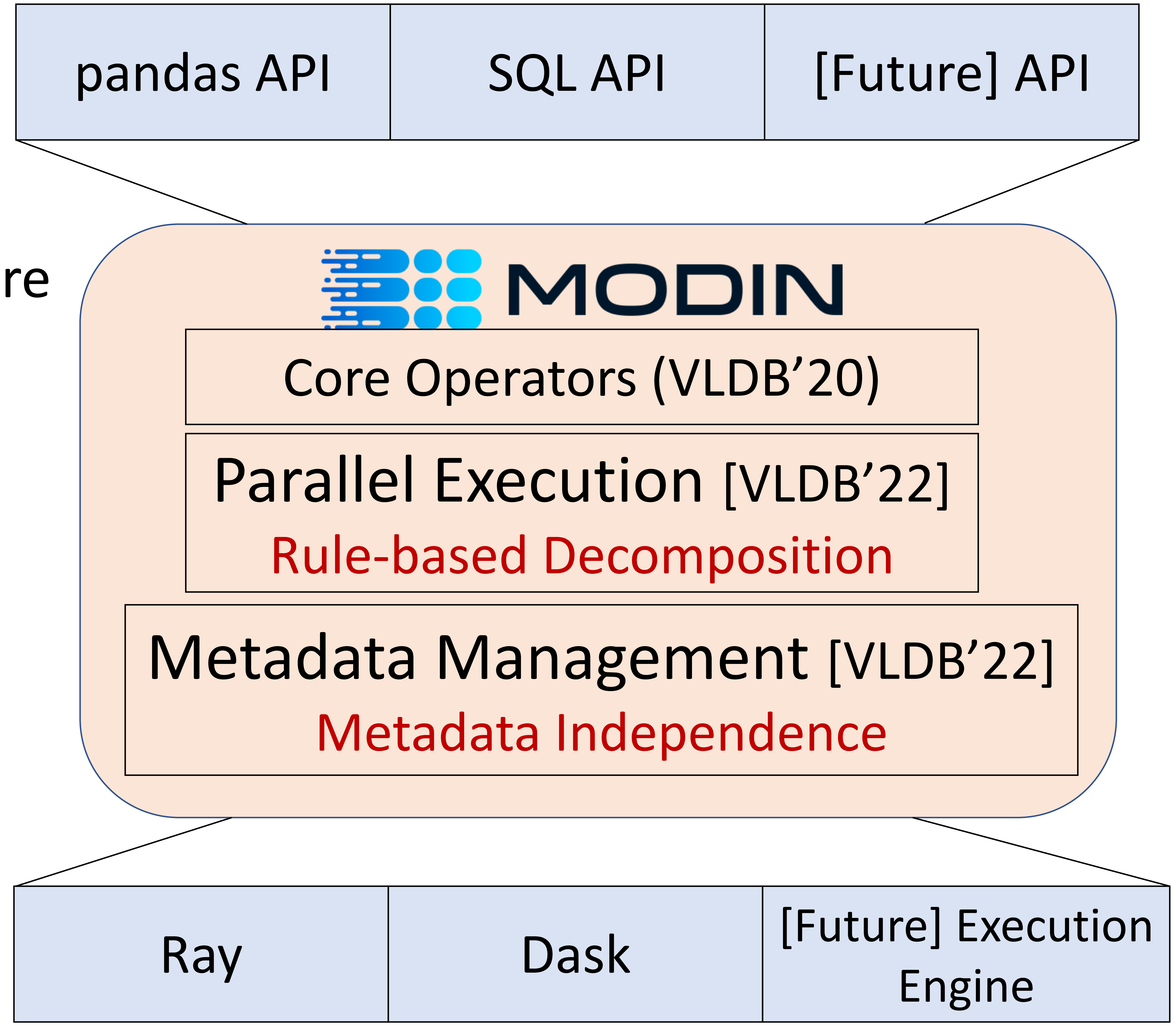
⭐ 7.7k Github stars

🤝 100+ contributors

Ray	Dask	[Future] Execution Engine

Modin: A “drop-in” Scalable Replacement for pandas

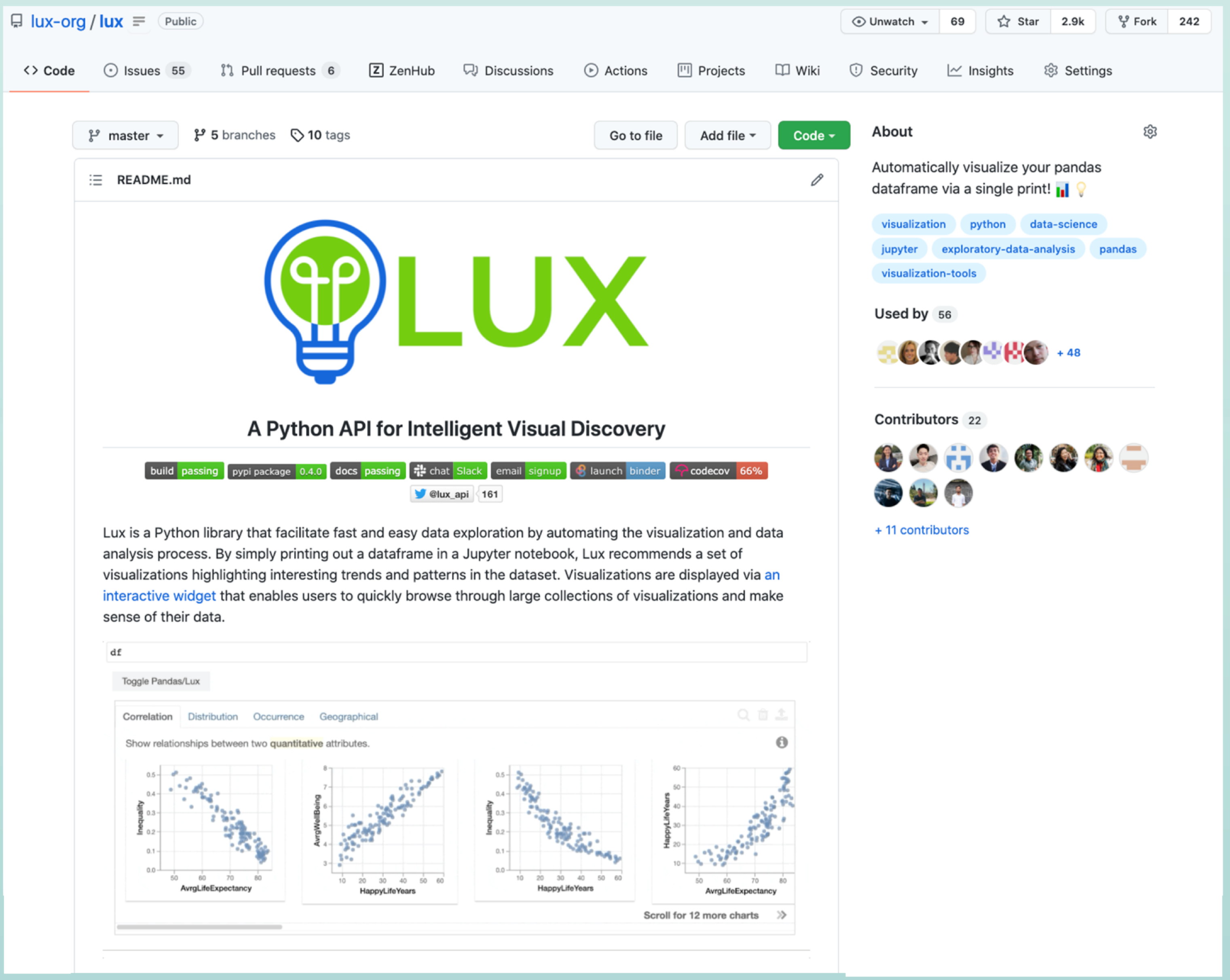
Please check out Modin here



Lessons Learned from Building Lux

- Integration with existing workflows
- Integration with downstream tools
- The ability for customization

github.com/lux-org/lux



Thank You!

EPIC DATA lab

Democratizing Data Work via No-code and Low-code interfaces

Berkeley UNIVERSITY OF CALIFORNIA

Backup Slides

About me

- PhD'20 at UChicago
- Jan. 2021 – now: Postdoc with Prof. Aditya G. Parameswaran at UC Berkeley
- My Research: building usable, scalable, and cost-effective solutions for data scientists