Persist: Persistent and Reusable Interactions in Computational Notebooks

Kiran Gadhave, Zach Cutler, Alexander Lex
http://vdl.sci.utah.edu
PERSISTENT AND REUSABLE INTERACTIONS IN COMPUTATIONAL NOTEBOOKS

A Jupyter Plugin

https://vdl.sci.utah.edu/persist/

pip install persist_ext

Demo Notebook: https://tinyurl.com/5db7nynn
Avalanches are a **major hazard** in Utah.

Utah Avalanche Center collects data about avalanches, including **where it occurred** (location, elevation), **how it occurred**, **how big it was**, etc.
Supposed you’re doing data analysis in Python

What’s the pandas code...

• …to change the order of columns?
• …to drop a column?
• …to change the label of a column?

Nothing here is hard, but it’s annoying.
Have you ever plotted something and wished you could just “fix” things as you spot them?

**WHAT IS THIS TALK ABOUT?**

- Elevation where the avalanche occurred
- How deep (big) the avalanche was
- Utah’s Elevation Range

Elevation where the avalanche occurred

Utah’s Elevation Range
PERSIST MAKES THIS EASY
SO WHAT’S SPECIAL HERE?

Lots of **vis tools** support these operations

Most **data wrangling happens in code**: it’s just more powerful

Opportunity: bring **interactive operations to code**!

Persist works **INSIDE your Jupyter Notebook**
BRIDGING BETWEEN DATA ANALYSIS MODALITIES
What are Modalities?

1. Interactive Vis
2. Code
INTERACTIVE VISUALIZATION: BENEFITS

Intuitive
Easy to use
Uses human perceptual capabilities
INTERACTIVE VISUALIZATION: **DOWNSIDES**

**Limited Expressivity**

Some operations are difficult

* e.g., conditional queries..

**Not reusable**

need to redo analysis when data changes

**Not reproducible**
Flexible and powerful
you basically can do anything

Reusable
if your data changes, re-run

Reproducible
everything is documented
It’s hard
requires extensive training
reading documentation
not discoverable

It’s time consuming
even simple things require effort

Some operations are difficult
e.g., labeling data points
COMPUTATIONAL NOTEBOOKS: A MIDDLE GROUND?

Observable

R Markdown

Jupyter Notebooks
Yes!

Afford both scripting and interactive visualization

But visualizations are a dead end

can’t “use” interaction in code
e.g., changing a label, or filtering a value
GAPS BETWEEN CODE AND INTERACTIVE OUTPUTS

**Semantic Gap**
Information only flows from code to visualization

**Temporal Gap**
Changes made to code are preserved
Changes made to vis are lost

**Layout Gap**
Changes in code are messy

*Wu, Hellerstein, Satyanarayan, UIST 2020*
THESIS: BRIDGING BETWEEN CODE AND INTERACTIVE VIS IS USEFUL

Easy handoffs are important!
RELATED WORK

B2 – Wu, Hellerstein, Satyanarayan, UIST 2020

Mage – Kery et al., UIST 2020
THE PERSIT APPROACH
Track events in interactive visualizations

Map them to data frame operations

Operations then applied to data frame
HOW IT WORKS

**Code to create chart**

```python
df = pd.read_csv('cars.csv')
PersistChart(scatterplot(df))
```

**Provenance**

- **df1**: Select 3 points
- **df2**: Remove selections
- **df3**: Select 2 points
- **df4**: Assign category V4 to selection
- **df5**: Select 3 points

**Interactive Visualization**

- **Initial plot & selection**
- **Remove selections**
- **Assign Category V4**
- **Select 3 points**

**Dataframe updates**

**In code**

```python
df5.head()
```

**Provenance**:

<table>
<thead>
<tr>
<th>Name</th>
<th>MPG</th>
<th>Cylinders</th>
<th>origin</th>
<th>Engine</th>
<th>is_selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>ford</td>
<td>18</td>
<td>8</td>
<td>USA</td>
<td>NA</td>
<td>FALSE</td>
</tr>
<tr>
<td>dodge</td>
<td>15</td>
<td>8</td>
<td>USA</td>
<td>NA</td>
<td>FALSE</td>
</tr>
<tr>
<td>volkswagen</td>
<td>22</td>
<td>4</td>
<td>Europe</td>
<td>V4</td>
<td>FALSE</td>
</tr>
<tr>
<td>amc</td>
<td>16</td>
<td>8</td>
<td>USA</td>
<td>NA</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

27 rows x 8 columns
PERSIST WORKFLOW
```python
import altair as alt
from vega_datasets import data
import persist_ext as PR

cars_df = data.cars()  # Load cars dataset

brush = alt.selection_interval()  # Create a 2d brush

# create scatterplot and link to brush
scatterplot = alt.Chart(cars_df).mark_point().encode(
    x="Miles_per_Gallon:Q",
    y="Weight_in_lbs:Q",
    color=alt.condition(brush, alt.value("steelblue"), alt.value("gray"))
).add_params(
    brush
)

PR.PersistChart(scatterplot)
```
import altair as alt
from vega_datasets import data
import persist_ext as PR

cars_df = data.cars()  # Load cars dataset
brush = alt.selection_interval()  # Create a 2d brush

# create scatterplot and link to brush
scatterplot = alt.Chart(cars_df).mark_point().encode(
    x='Miles_per_Gallon:Q',
    y='Weight_in_lbs:Q',
    color=alt.condition(brush, alt.value('steelblue'), alt.value('gray'))
).add_params(brush)

PR.PersistChart(scatterplot)
import altair as alt
from vega_datasets import data
import persist_ext as PR

cars_df = data.cars()  # Load cars dataset
brush = alt.selection_interval()  # Create a 2d brush

# create scatterplot and link to brush
scatterplot = alt.Chart(cars_df).mark_point().encode(
    x="Miles_per_Gallon:Q",
    y="Weight_in_lbs:Q",
    color=alt.condition(brush, alt.value("steelblue"), alt.value("gray"))
).add_params(brush)

PRPersistChart(scatterplot)
Branches and Choosing a State in provenance support non-linear analysis, addressing the layout gap.
Branches and **Choosing a State** in provenance support *non-linear* analysis, addressing the layout gap.
Persist **re-runs the interactions** in the output, addressing the **temporal gap**.
Persist applies interactions to data frames that can be accessed in code, addressing the **semantic gap**.
VISUALIZATION OPTIONS

Arbitrary Vega-Altair Charts

An Interactive Data Table
Persist works with most Vega-Altair charts

“Listens” to native operations (selections)

Updates Vega charts:

- Use original chart spec when possible (e.g., filters)
- Update spec when necessary (categories, labels)
EXAMPLE CHARTS
DEMO
EVALUATION
IN-LAB STUDY

---

**STUDY DESIGN**

We recruited ELEVEN participants for the study. Participants all had prior experience with Python and Pandas.

- **Full Factorial Design**
  - 2 Datasets × 2 Conditions
- The order of conditions was randomly assigned.
  - For each condition, datasets were randomly assigned. Participants never saw the same dataset twice.

---

**STUDY METRICS**

- Subjective Performance
- Time
- Error
- Reproducibility

---

**CONDITIONS**

**Pandas Coding**

1. `import pandas as pd
   df = pd.read_csv(...)`
2. `df = df.drop("age")
   df = df.rename(columns={"job": "profession"})
   df.loc["row"] = "principal"

**Persist Extension**

1. `import persist_ext as pr
   pr = pandas.read_csv(...)`  
2. `pr.plot(scatterplot(...))`

---

**DATASETS**

- **Video Games**
- **Avalanches**
IN-LAB STUDY

TASKS

PARTICIPANTS MADE THE FOLLOWING CHANGES TO A DATASET

- REMOVE COLUMNS
- RENAME COLUMNS
- CHANGE DATA TYPE
- EDIT VALUES
- FILTER DATA
- ADD CATEGORICAL COLUMN
RESULTS

- 11/11 notebooks using Persist were reproducible
- 3x times faster with Persist
- 97% tasks correctly using Persist, compared to 85% for Pandas
- only 7/11 using Pandas were reproducible
“so much easier than manually coding.”  
  - M4

“easier as compared to the code and everything was visible [...] and it didn’t take much time.”
  - M2

“Changing the category type, or adding new categories or removing anomalies from data, they were very much easier in [Persist] than coding.”
  - M7

“The thing I really liked about is version control, which shows the history of all operations [...] and also saves the changes [...] into a data frame.”
  - M14
DISCUSSION
Provenance **better for most cases**

- No code clutter
- Undo/redo
- Consistent semantics

But code generation might be **more robust**

- works outside of notebooks
- works w/o the library
GENERATE CODE ON DEMAND!

[Beta]
BEYOND JUPYTER
Other Interactive **Plotting Libraries**

Plot.ly, Bokeh

Other **data types**

Maps, Networks, …

[ Nobre et al., 2020 ]
Persist: Persistent and Reusable Interactions in Computational Notebooks

Kiran Gadhave, Zach Cutler, Alexander Lex
http://vdl.sci.utah.edu

Thanks to Jake Wagoner
Thanks to the NSF for Funding (IIS 1751238, CNS 213756, and CNS-2313998)