JUPYTERLAB_VOYAGER: A DATA VISUALIZATION ENHANCEMENT IN JUPYTERLAB

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Ji Zhang

June 2018

COMMITTEE MEMBERSHIP

TITLE:	JupyterLab_Voyager: a Data Visualization Enhancement in JupyterLab
AUTHOR:	Ji Zhang
DATE SUBMITTED:	June 2018
COMMITTEE CHAIR:	Brian Granger, Ph.D. Associate Professor of Physics
COMMITTEE MEMBER:	Franz Kurfess, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	Jennifer Klay, Ph.D. Associate Professor of Physics

ABSTRACT

JupyterLab_Voyager: a Data Visualization Enhancement in JupyterLab

Ji Zhang

With the emergence of big data, scientific data analysis and visualization (DAV) tools are critical for the data science software ecosystem; the usability of these tools is becoming extremely important to facilitate next-generation scientific discoveries. JupyterLab has been considered as one of the best polyglot, web-based data science tools. As the next phase of extensible interface for the classic iPython notebooks, it supports interactive data science and scientific computing across multiple programming languages with great performances. Despite these advantages, heuristics evaluation studies have shown that JupyterLab has some significant flaws in the data visualization side. Its DAV system heavily relies on users' understanding and familiarity with visualization libraries, and doesn't support the golden visual-information-seeking mantra of 'overview first, zoom and filter, then details-on-demand'. These limitations often lead to a workflow bottleneck at the start of a project.

In this thesis, we present 'JupyterLab_Voyager', an extension for JupyterLab that provides a graphical user interface (GUI) for data visualization operations and couples faceted browsing with visualization recommendation to support exploration of multivariate, tabular data, as a solution to improve the DAV system. The new plugin works with various types of datasets in the JupyterLab ecosystem; using the plugin you can perform a high-level graphical analysis of fields within your dataset sans coding without leaving the JupyterLab environment. It helps analysts learn about the dataset and engage in both open-ended exploration and target specific answers from the dataset. User testings and evaluations demonstrated that this implementation has good usability and significantly improves the DAV system in JupyterLab.

ACKNOWLEDGMENTS

Thanks to:

- Professor Brian Granger for his guidance and good advice.
- My wife and family for supporting me throughout my graduate education.
- Professor Kurfess and Professor Klay in my thesis defense committee.
- Project Jupyter team at Cal Poly.
- The entire Cal Poly Computer Science Department.
- The Data Voyager team in the Interactive Data Lab at University of Washington.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1 Introduction 1
- 1.1 DAV Tools 1
- 1.2 JupyterLab: the Data Science Tool 2
- 1.3 JupyterLab_Voyager Extension 4

2 Related Works 5
- 2.1 Graphic Grammars in Visualization Systems 5
- 2.2 Visual Data Analysis 6

3 Design 9
- 3.1 Problem Specification 9
- 3.2 Voyager Interface 11
- 3.3 File Transition 12
- 3.4 Feature Engineering 14
- 3.5 Usability Test 16

4 Implementation 18
- 4.1 Voyager Class Overview 18
- 4.2 Open and Save for Local File Types 20
- 4.3 Open and Save for Notebook Charts 22
- 4.4 Transition between Voyager and Notebooks 23

5 System Evaluation 27
- 5.1 User-testing Design 27
- 5.2 User-testing Methods 29
  - 5.2.1 Materials and Devices 29
  - 5.2.2 Procedures 29
  - 5.2.3 Confidentiality 30
- 5.3 Result Analysis 30

6 Conclusion 35
7 Future Directions 36
7.1 Temporary File System 36
7.2 Improving Workflow between Voyager and Notebooks 37
7.3 Improving Voyager Functions 38
BIBLIOGRAPHY 39
APPENDICES
A JupyterLab User Interview 42
B Beginner Level User Persona 44
C Expert Level User Persona 45
D Pre-screening Survey 46
E Usability Test Script 47
F Consent Form 52
G Usability Test Result Summary 54
H Participates Background Summary 57

LIST OF TABLES

Table Page

5.1 User Testing Instructions 34

LIST OF FIGURES

Figure Page

1.1 JupyterLab Interface: The JupyterLab interface consists of a main work area (contains tabs of documents and activities), a collapsible left sidebar (contains a file browser, the list of running kernels and terminals, the command palette, the notebook cell tools inspector, and the tabs list), and a top menu bar. 3

2.1 Voyager Interface: (A) Bookmark gallery and undo commands. (B) Data panel contains the dataset name. (C) Data fields. (D) Wildcard fields. (E) More custom wildcards with desired fields. (F) The encoding panel contains shelves for mapping fields to visual channels via drag-and-drop, and a control for selecting mark type. (G) A wildcard shelf lets users add fields without selecting a specific channel, allowing the system to suggest appropriate encodings. (H) The filter panel for dynamic query controls for filtering. (I) The primary focus view displays the currently specified chart. (J) Related views show recommended plots relevant to the focus view. (K) Related summaries suggest aggregate plots to summarize the data. (L) Field suggestions show the results of encoding one additional field within the focus view. 8

3.1 Typical Data Science Workflow in JupyterLab. 10

3.2 JupyterLab_Voyager Interface. 15

3.3 Designed Workflow between File System, Voyager and Notebooks. . 16

4.1 The System Design for Opening Local Files 21

4.2 The System Design for Saving Back to Local Files 22

4.3 Operating Options: Saving Back to Local Files 23

4.4 System Design: from Notebook to Voyager 24

4.5 System Design: from Voyager to Notebook 25

4.6 Operating Options: from Voyager to Notebook 26

5.1 Error Rate of User's First Action 32

5.2 Fail Rate of Each Task 33

Chapter 1

INTRODUCTION

With the emergence of big data, scientific data analysis and visualization tools are critical components of the data science software ecosystem, and the usability of these tools is becoming extremely critical to facilitate next-generation scientific discoveries. JupyterLab has been considered as one of the best polyglot, web-based, open-source data science tools [9, 6] . As the next phase of extensible interface for the classic iPython notebooks, this tool supports interactive data science and scientific computing across multiple programming languages with great performances. Despite these advantages, previous heuristics evaluation studies [16] showed that the DAV system in JupyterLab has some significant flaws in the usability fields and needs to be improved.

1.1 DAV Tools

Our so called information age more often seems like an era of information overload. A publication in 2013 estimated that new data generated everyday can reach up to 2.5 billion Gigabyte (GB) [3]. These excessive amounts of information are overwhelming, how to handle them is becoming a hot topic. Raw data becomes useful when we apply methods of deriving insight from it. For humans, who are 'intensely visual creatures' [7], it is easy to interpret graphic charts and understand the meaning from those data's visual representations, but hard to detect any patterns among raw data, for example, rows of numbers. Data visualization, the process of encoding raw data as visual objects contained in graphics, is a powerful exercise and is a good way to communicate information clearly and efficiently to others. And the tools for scientific data analyses and visualization are becoming critical to next-generation scientific discoveries.

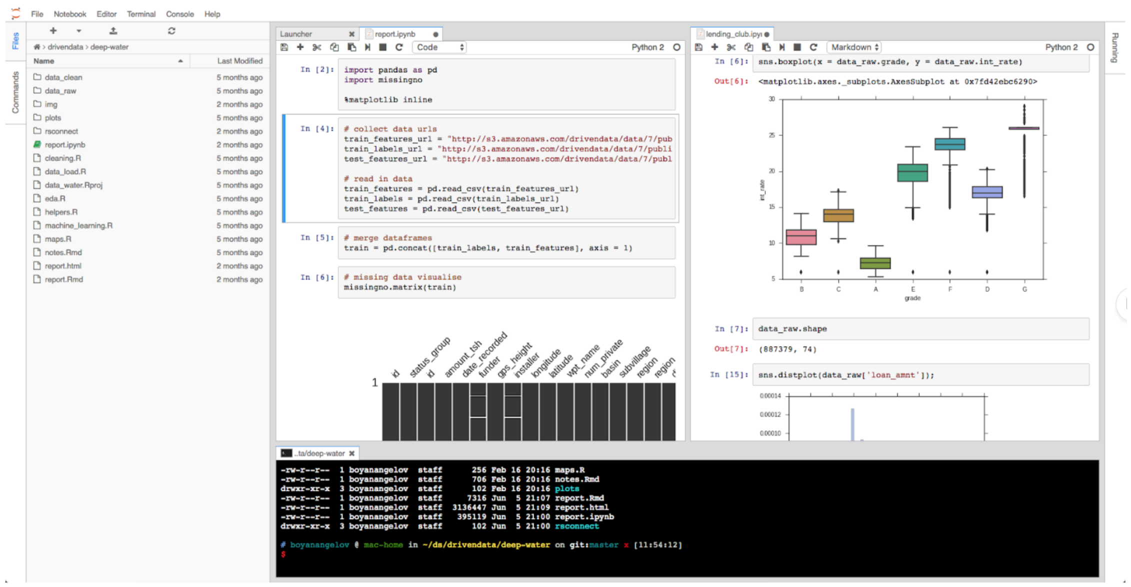
DAV tools are developed to enable analyzing data and building visualization entities to help users understand the data better, and the usability of these tools plays an important role in determining how useful they are to the users. For softwares, usability is an important attribute of quality and one of the key factors that explains users' learnability, efficiency, memorability and satisfaction. The ISO 9241 standard [14] defines usability as the extent to which a product can be used with efficiency and satisfaction by specific users to achieve specific goals in specific environments. Today, it is common practice to do usability evaluations to ensure those tools can meet users' needs. When evaluators examine an interface, they need to follow a set of usability principles, called heuristics [8]. This heuristics evaluation technique has been applied to different contexts including DAV tools and provides an impartial way to determine the effectiveness of these softwares [16].

1.2 JupyterLab: the Data Science Tool

JupyterLab is emerging as the next UI generation for Project Jupyter [6], which offers users with an integrated development environment (IDE) -like experience [10]. Here, data scientists don't need to handle scattered tools, JupyterLab already puts most of the necessary instruments together and offers a customizable and flexible working environment, which allows window docking, window combination and dynamic dashboard creation on demand. It totally changes the way users see the notebooks since the original iPython version. As the next phase of extensible interface for the classic iPython notebooks, JupyterLab focuses on interactive, exploratory computing across multiple programming languages (figure 1.1), and keeps many features found in traditional IDEs.

Despite these advantages, heuristics evaluation studies have shown that Jupyter-

Lab actually has some significant flaws on data visualization side [16]. Ideal DAV tools should support the visual-information-seeking mantra of overview first, zoom and filter, then details-on-demand [13]. Following this standard, good DAV systems should allow users to interact with data through mouse clicks, menus, functions, and even screen-touch to generate visualizations to discern patterns, trends and outliers. Although most of those aspects are achievable inside JupyterLab, users have to use special libraries and coding languages to operate, to manage and to customize the graph, which are not easy tasks and can really distract the users from focusing on the data analysis. Those issues significantly bring down the user experience, thus in the heuristics evaluation [16], it scored low on fields like user control, analytical reasoning and visualization support.



Name	Last Modified
data_clean	5 months ago
data_raw	5 months ago
img	2 months ago
plots	5 months ago
rsconnect	2 months ago
report.ipynb	2 months ago
cleaning.R	5 months ago
data_load.R	5 months ago
data_water.Rproj	5 months ago
eda.R	5 months ago
helpers.R	5 months ago
machine_learning.R	5 months ago
maps.R	5 months ago
notes.Rmd	5 months ago
report.html	2 months ago
report.Rmd	2 months ago

```
In [2]:  import pandas as pd
         import missingno

         %matplotlib inline

In [4]:  # collect data urls
         train_features_url = "http://s3.amazonaws.com/drivendata/data/7/pub
         train_labels_url = "http://s3.amazonaws.com/drivendata/data/7/publi
         test_features_url = "http://s3.amazonaws.com/drivendata/data/7/publ

         # read in data
         train_features = pd.read_csv(train_features_url)
         train_labels = pd.read_csv(train_labels_url)
         test_features = pd.read_csv(test_features_url)

In [5]:  # merge dataframes
         train = pd.concat([train_labels, train_features], axis = 1)

In [6]:  # missing data visualise
         missingno.matrix(train)
```

```
In [6]:  sns.boxplot(x = data_raw.grade, y = data_raw.int_rate)

Out[6]:  <matplotlib.axes._subplots.AxesSubplot at 0x7fd42ebc6290>

In [7]:  data_raw.shape

Out[7]:  (887379, 74)

In [15]: sns.distplot(data_raw['loan_amnt']);
```

```
-rw-r--r--  1 boyanangelov  staff      256 Feb 16 20:16 maps.R
-rw-r--r--  1 boyanangelov  staff      706 Feb 16 20:16 notes.Rmd
drwxr-xr-x  3 boyanangelov  staff      102 Feb 16 20:16 plots
-rw-r--r--  1 boyanangelov  staff     7316 Jun  5 21:07 report.Rmd
-rw-r--r--  1 boyanangelov  staff  3136447 Jun  5 21:09 report.html
-rw-r--r--  1 boyanangelov  staff   395119 Jun  5 21:00 report.ipynb
drwxr-xr-x  3 boyanangelov  staff      102 Jun  5 21:00 rsconnect

# boyanangelov @ mac-home in ~/ds/drivendata/deep-water on git:master x [11:54:12]
$
```

Figure 1.1: JupyterLab Interface: The JupyterLab interface consists of a main work area (contains tabs of documents and activities), a collapsible left sidebar (contains a file browser, the list of running kernels and terminals, the command palette, the notebook cell tools inspector, and the tabs list), and a top menu bar.

1.3 JupyterLab_Voyager Extension

To solve the usability issues in the Data Visualization system inside JupyterLab, and to enhance the visualization support with better analytical reasoning through graphics, we designed and developed 'JupyterLab_Voyager', a plugin extension for enhancing the DAV system in JupyterLab. This new extension incorporated a third-party data visualization tool 'Voyager', which provides a GUI for data visualization operations and couples faceted browsing with visualization recommendations, into the original system. With this enhancement, now JupyterLab can blend manual and automated chart specification together to help users explore the multivariate, tabular data, and engage in both open-ended exploration and targeted question answering. We also performed user-testings to evaluate the extension's usability. The reason for choosing Voyager and the details of design and implementation will be discussed thoroughly in the following chapters.

Chapter 2

RELATED WORKS

JupyterLab_Voyager draws on and extends prior research on graphics grammars, visual analysis tools and focuses on usability improvements.

2.1 Graphic Grammars in Visualization Systems

In data science field, graphic grammars is quite a popular way to create statistical graphics. Expressive lower-level grammars, including those of Protovis [1], D3 [2] and Vega [12], provide users with a expressive design space with abstracting data models, graphical marks, visual encoding channels, scales, axes and legends, and allow them to construct graphics for exploratoration and other analysis [4]. In the past, those graphic grammars have been widely used for creating explanatory and highly-customized graphics. But due to their nature of lower-level grammars, they are not easy for human to understand or edit, which significantly limits their usage.

Vega-Lite is a high level grammar based on Vega [11]. It uses a set of encoding definitions that map data attributes to graph position, color, shape, size, and other visual channels. It also includes common data transformations such as binning, aggregation, sorting, and filtering. Vega-Lite uses a portable JavaScript Object Notation (JSON) syntax that can be generated from a variety of programming languages, which makes it more user-friendly than Vega. When using Vega-Lite, its specifications need to be compiled to full Vega specifications, hence it may lose some expressiveness. However, the dramatic gains in the conciseness and clarity of specification in usage fields made it a popular grammar using in various DAV tools including JupyterLab and Voyager.

2.2 Visual Data Analysis

When doing exploratory data analysis and exploratory search [17], users might be unfamiliar with their datasets resources, undecided about their goals, or unsure about how to reach their objectives. To perform exploratory tasks, users either query for specific information or browse to gain an overview and discover the unexpected. As they gain new information, users may clarify their goals and engage in alternative approaches. Therefore, data visualization tools should employ interfaces such as faceted browsers and dynamic queries [20] to let users focus on interested items. With these interfaces, users express their intent in the form of partial specifications that convey criteria for desired items. For large collections, recommender systems [5] can populate a seed set to help users begin exploring or suggest relevant alternatives to selected items.

Voyager is a new DAV tool aiming at supporting exploratory data analysis in an analogous fashion [18]. It is developed by the Data Voyager team in Interactive Data Lab at University of Washington in Seattle. The current version of Voyager was released in 2017. This Voyager system suggests univariate summaries to help analysts begin an exploration. It offers a graphical interface in which users can drag-and-drop data fields onto visual encoding shelves. These interactions produce complete view specifications using the VizQL visual analysis grammar. Inspired by previous works [15], Voyager integrates a graphical specification recommendation interface, which recommends charts related to the current focus view. All of those interactions are included in this unified tool (figure 2.1), enabled by a query language that supports partial specification of visualizations.

Voyager system uses a Compass recommender engine to generate and present a gallery of recommended charts to facilitate breadth-oriented exploration. This engine accepts user-selected data fields and summary functions to steer the recommendations

[19]. This feature helps Voyager blend manual and automatic chart specification in a unified system, which enables pivoting among multiple interaction methods. Users no longer need to create every graph for all the variable pairs, and can easily browse suggested views related to their current focus chart. Voyager itself can also be considered as a module, which allows it to be incorporated into other systems. The Application Programming Interface (API) of Voyager module exposes one function:

CreateVoyager(container, config, data)

This function can create an instance of the Voyager application and return it. The first parameter 'container' refers to a css selector or an HTMLElement that will be the parent of the newly created Voyager. The second and third parameter describe the detail configuration and the data source of this Voyager.

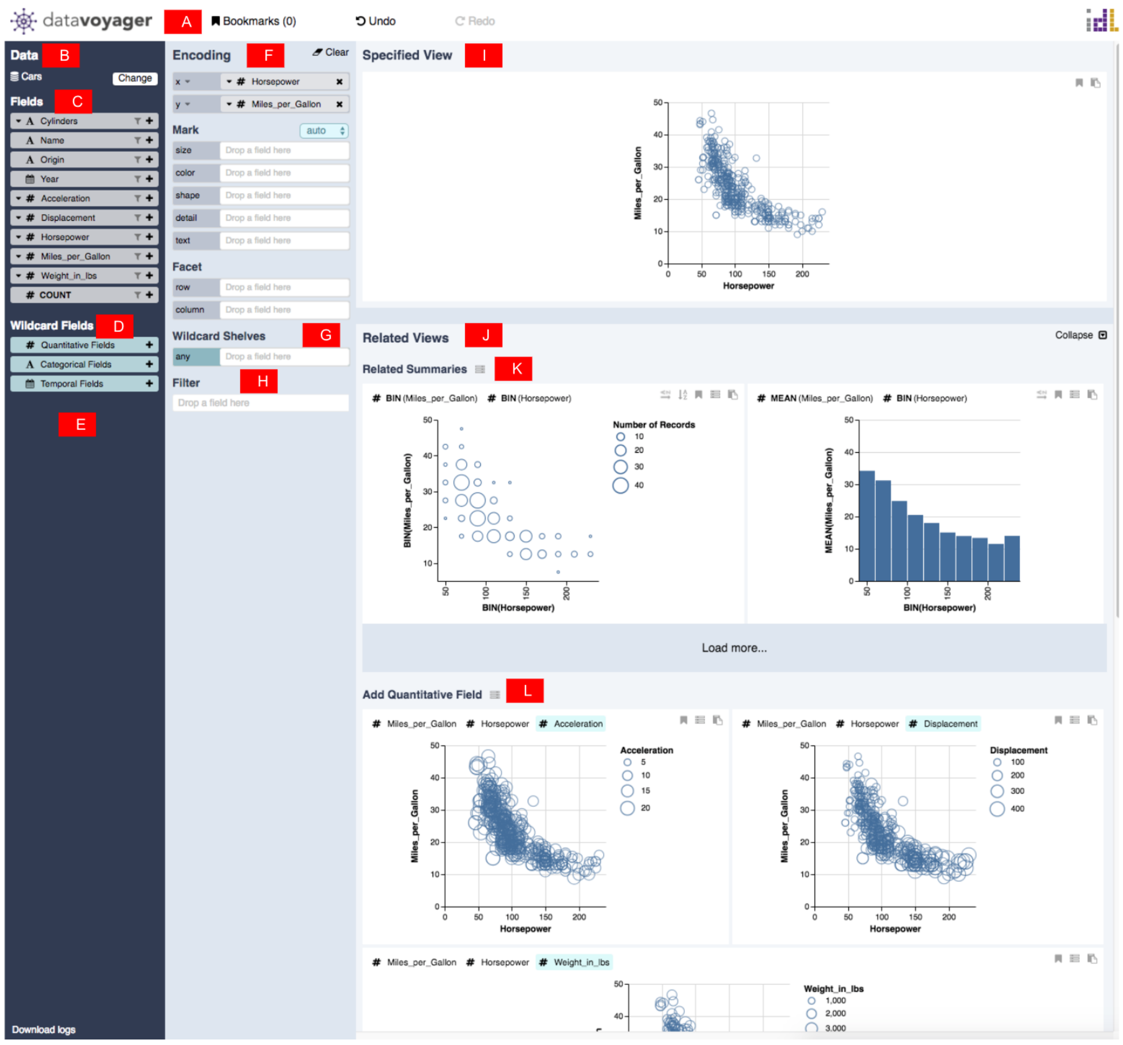


Figure 2.1: Voyager Interface: (A) Bookmark gallery and undo commands. (B) Data panel contains the dataset name. (C) Data fields. (D) Wildcard fields. (E) More custom wildcards with desired fields. (F) The encoding panel contains shelves for mapping fields to visual channels via drag-and-drop, and a control for selecting mark type. (G) A wildcard shelf lets users add fields without selecting a specific channel, allowing the system to suggest appropriate encodings. (H) The filter panel for dynamic query controls for filtering. (I) The primary focus view displays the currently specified chart. (J) Related views show recommended plots relevant to the focus view. (K) Related summaries suggest aggregate plots to summarize the data. (L) Field suggestions show the results of encoding one additional field within the focus view.

Chapter 3

DESIGN

This chapter specifies the current issues with the JupyterLab DAV system, and explains the design of the proposed solution. Major components of the section include user study, feature engineering, and software usability test.

3.1 Problem Specification

Although the previous research paper has discussed certain visualization flaws in Jupyterlab, the description in that manuscript was quite abstract. Jupyter and other softwares were only investigated by two user-interface (UI) experts following heuristics evaluation principles, so the paper didn't provide any details about the needs of the real users like data scientists. In this project, we first conducted user investigations on data scientists with different experience levels. The content of the user interview is listed in Appendix A. Through that way, we studied data scientists' usual workflow and investigated the detailed usability problems they have with JupyterLab. Generally, when data scientists receive a new dataset to work on, first, they tend to look for descriptions of the dataset, like listing of columns, a header file telling what each field is, to learn what this dataset is about. Once they feel they have got enough information, they will come up with an idea about what is the current task. After loading the data into JupyterLab, they start to work different variables against each other in two steps: 1) plot each variable by itself to look how it varies; 2) explore interactions between variables, like testing a few variables in a graph to see if there's any co-relations. At last, they will call external mathematical libraries to do a complex analysis via regression, differentiation, integration, complex dynamics to dig into details, and interpret the final results. The whole process is shown below in figure 3.1.

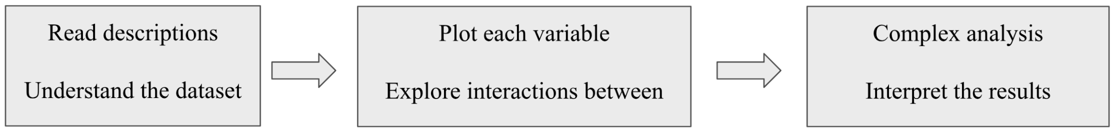


Figure 3.1: Typical Data Science Workflow in JupyterLab.

The issues raised by the data scientists during interviews are collected and summarized. To help us understand the problem, two personas were created based on the information. The first persona (Appendix B) represents a junior student with less experience, and the second one (Appendix C) represents a more experienced data scientist. From these two personas, we can see that, despite the difference between their experience, the usability problems they are facing are quite similar:

A. It's difficult to get an overview when learning about a new dataset;

B. Visualization library code: DAV in JupyterLab relies on using visualization libraries (ex: Altair), which requires user to memorize lots of coding formulars and definitions. Although experienced users don't have much trouble remembering the library code, GUI is still a prefered choice;

C. It's hard to export the data visualization work inside notebook;

The first problem happens during the data exploration stage. Since this process is pure coding based, and the users need to be aware of all the existing variables while exploring the interactions between, this work is tedious, inefficient, and really takes time. When the dataset is large, users don't have a very good overview even after long time of exploration.

The second problem is caused by the fact that the visualization library like Altair is huge, and it's not easy to remember all the APIs. To use it, beginner level users have to look for online references from time to time. And even for experienced users, they will have some trouble once a new version of Altair library is released with API changes. Both problems 1&2 are caused by the nature of using a coding based DAV tool inside JupyterLab. When doing analysis using such a tool, it requires the user to focus on 3 things at the same time: the target (dataset information), the tool (Altair code), and the goal (data relations). We all know people's ability to concentrate is quite limited, when they try to focus on too many things at the same time, they can easily lose focus, get interrupted in their workflow and get mis-impression about the dataset. Clearly, coding based DAV tools are not the best practice for JupyterLab users.

Complaints about the third problem mainly focuses on the fact that JupyterLab doesn't have very good support for graph edit, import and export operations in notebooks. Export as an image, viewing source code in a new browser or in an online Vega editor, are the only available graph operating options in current JupyterLab. Those operations don't treat the graph as an editable file, and there are no proper transitions between them. It's relatively hard to import and export graphs with proper editable form. Those choices made the DAV system disjoint with the notebook ecosystem. To solve this problem, we need to make modifications on the current graph file system in the workflow, to achieve smooth transitions between file system, notebook and graph editors.

3.2 Voyager Interface

As mentioned in section 2.2, Voyager has a good UI interface which facilitates the dataset operations. On top of that, its visualization is also based on Vega-Lite, which means it uses the same graphic language as the current JupyterLab. When Voyager was designed, APIs and external library functions were included, which makes it relatively easy to use in other web-based systems. Those are some important reasons why we choose Voyager for this DAV improvements project.

When using Voyager to do the analysis, all data fields were extracted, listed, and marked with their specific types (quantitative or categorical). You can easily create and modify a graph through drag and drop data fields into different 'Encoding' boxes. Almost all Altair library functions are achievable through this way. And the suggested views can greatly help users' data exploration process. With this tool (figure 3.2), users no longer need to pay much attention to the target (dataset information) and the tool (Altair code), and can focus solely on their goals (data relations) during the analysis.

Therefore, we consider Voyager as a perfect solution for problems 1&2 in the previous section.

3.3 File Transition

Another issue with the DAV system in JupyterLab is about graph file transition within the notebook ecosystem. Jupyter notebooks are documents that combine live runnable code with narrative text, equations, images, interactive visualizations and other rich output, and are brilliant data analysis tools. In the current design, the data visualization process is kept inside notebook, and there are not many options for graph file transitions. When exporting to local files, users can only get the image in a TIFF or PNG format, with no data attached. And the only other editor option for people to use is an external web based Vega-editor, and this operation is one-direction only: from notebook to Vega-editor web page, and it's rather hard to get the content back to notebook after editing.

To solve this issue, we plan to build a graph file transition system for Jupyter-

Lab. The first step is to select a proper file type to represent the graph. Vega-Lite (VL.JSON type file) is the ideal choice here. It's an expressive grammar and can contains all the necessary informations inside. It enables rapid specification of interactive data visualizations, and the entire information of the data visualization, including data source (inline data or URL links), encoding fields, chart type and details, can be stored in JSON form, namely a VL.JSON file type; Besides, JupyterLab has very good support for Vega-Lite, the original build-in data visualization system is largely based on this grammar. And Voyager, the GUI tool we plan to use, is also based on Vega-Lite. Therefore, Vega-Lite (VL.JSON) file type is the one to choose.

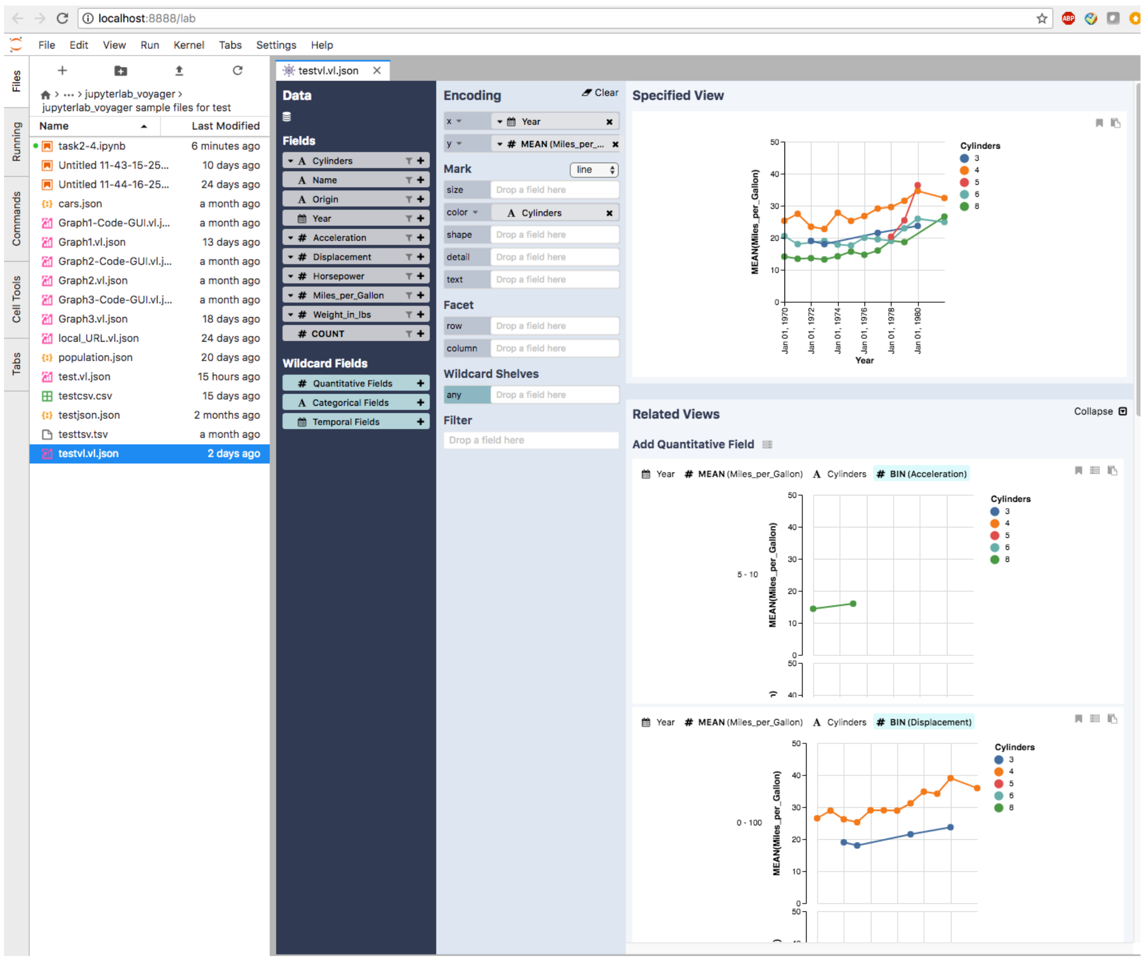
Second, to facilitate a smooth file transition, we need to design operating paths for the workflow. A typical one would be: local dataset file - Voyager interface - save back to local file. This path looks quite straight-forward: read and load the local file content into the Voyager tool, do visualization and analysis there, and at the end store the work back to the local file. However, considering the various file types of data sources, situations can be quite complex in the stage of save&store. Common file types used as data source include CSV, TSV, JSON and VL.JSON. For VL.JSON type file, the content in Voyager can be saved back to the original local file without any problem. But for typical tabular data formats like CSV and TSV, which can only contain columns and rows of data, any attempts to save the visualization specifics back would ruin the files. And for JSON, although the entire content of Vega-Lite can be saved here, users usually don't expect a JSON file to have the whole body of Vega-Lite content like VL.JSON file. So if we allow system to save Voyager content back to a JSON file, it may cause conflicts between the JSON file content and user's expectations. Therefore, for dataset file types other than VL.JSON, we plan to use an 'Export' option to store the current work inside Voyager in a newly created Vega-Lite(.VL.JSON) file.

Third, we need to build transition paths connecting Voyager interface and notebooks. After all, the notebook ecosystem is the most powerful and complete data science tool inside JupyterLab. When considering transferring data from notebook to Voyager, the only operable data sources inside notebook are charts like graphs and tables. Passing the charts' data to Voyager is not difficult, and we can build a command to do so and embed it in the context menu. When it comes to the save&storage stage, since there's no real local file for this type of dataset, we don't really have a place to save the Voyager visualization work. So, similar with CSV, TSV and JSON file types, the 'Export' option seems to be the only choice here. When considering transferring data from Voyager to notebook, the most direct way is to build a special 'Copy to notebook' command to extract Voyager content in Vega-Lite form and paste it back to notebook.

3.4 Feature Engineering

As discussed above, we plan to incorporate Voyager into JupyterLab to enhance the visualization support in DAV system as well as to provide better analytical reasoning through graphics.

To incorporate the Voyager interface into the JupyterLab ecosystem. Our design here is to use Voyager as a special 'Editor' for files. So, for local files, the 'Open With Voyager' command should stay together with other opening options within the 'Open With ...' context menu. And for charts inside notebooks, it should serve as an editing option and therefore goes into the field context menu. When users try to save or export their works in Voyager, options should be given in a way similar to other editors. In JupyterLab, all editors are linked with the main 'File' menu and the 'Save' and 'Export' buttons inside it are the default ways for save and export actions. In addition, the notebook provides a toolbar with easy access to those functions as well. So, we implemented both ways in JupyterLab_Voyager for users' convenience



Name	Last Modified
task2-4.ipynb	6 minutes ago
Untitled 11-43-15-25...	10 days ago
Untitled 11-44-16-25...	24 days ago
cars.json	a month ago
Graph1-Code-GUI.vl.j...	a month ago
Graph1.vl.json	13 days ago
Graph2-Code-GUI.vl.j...	a month ago
Graph2.vl.json	a month ago
Graph3-Code-GUI.vl.j...	a month ago
Graph3.vl.json	18 days ago
local_URL.vl.json	24 days ago
population.json	20 days ago
test.vl.json	15 hours ago
testcsv.csv	15 days ago
testjson.json	2 months ago
testtsv.tsv	a month ago
testvl.vl.json	2 days ago

Fields: Cylinders, Name, Origin, Year, Acceleration, Displacement, Horsepower, Miles_per_Gallon, Weight_in_lbs, COUNT

x	Year
y	MEAN (Miles_per_...
Mark	line
size	Drop a field here
color	Cylinders
shape	Drop a field here
detail	Drop a field here
text	Drop a field here
Facet row	Drop a field here
Facet column	Drop a field here
any	Drop a field here
Filter	Drop a field here

Year # MEAN (Miles_per_Gallon) A Cylinders # BIN (Acceleration)

Year # MEAN (Miles_per_Gallon) A Cylinders # BIN (Displacement)

Figure 3.2: JupyterLab_Voyager Interface.

(figure 3.3).

However, the incorporation of Voyager into JupyterLab disturbs the original workflow of JupyterLab users. Previously, data scientisst do all the data manipulation, analysis and visualization inside the notebook interface. With JupyterLab_Voyager, when it comes to the visualization part, the data is actually extracted and passed to a new Voyager interface, and the following operations will be outside of the notebook. If the users want to continue working inside the notebook widget, they will have to save the current voyager work and go back to the previous position. Although we tried very hard to base the design on JupyterLab user habits, changes in operations are unavoidable and we are not sure about how those changes will affect

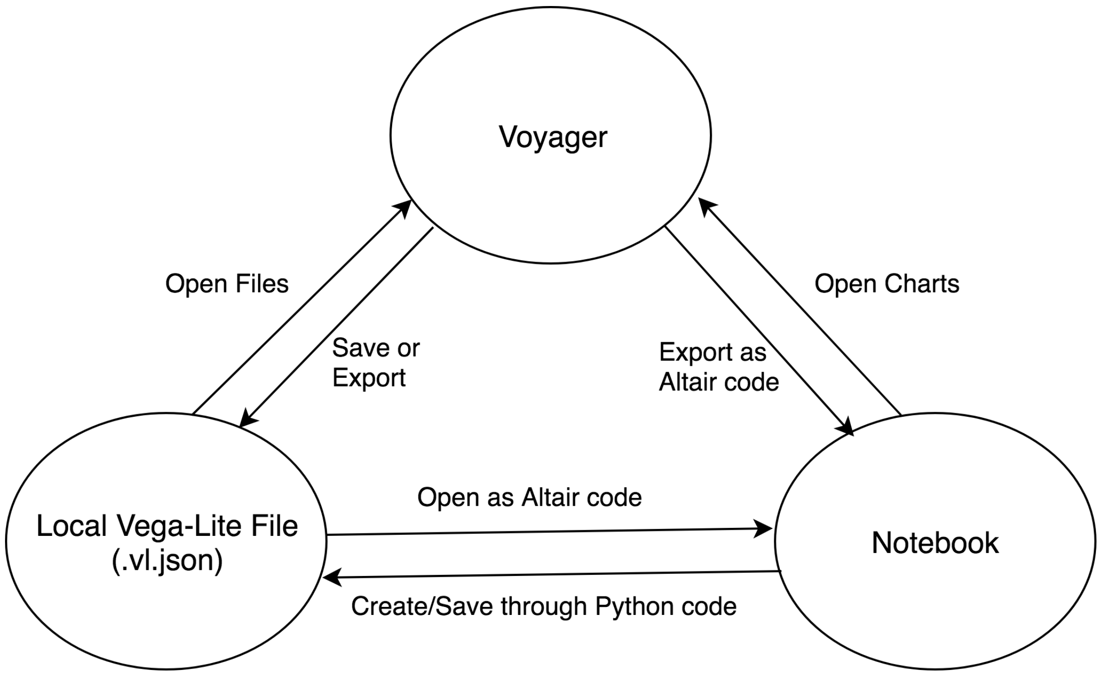


Figure 3.3: Designed Workflow between File System, Voyager and Notebooks.

the real users. Thus we need to perform user-testings and evaluate our design and development based on the results.

3.5 Usability Test

User-testings were performed to demonstrate the usability of this JupyterLab_Voyager. The tests included establishing a baseline of user performance, establishing and validating user performance measures, and identifying potential design concerns to be addressed in order to improve the software efficiency, productivity, and end-user satisfaction.

In this usability test, participants' responsibilities are to attempt to complete a set of representative task scenarios presented to them in as efficient and timely a manner as possible, and to provide feedback regarding the usability and acceptability of the user interface. The participants were directed to provide honest opinions regarding the usability of the application, and to participate in post-session subjective questionnaires and debriefing. Those tasks require the participants to be familiar with data science and coding, have experience with both JupyterLab and data visualization. Here at Cal Poly, students in the Data Science 301 class usually have a solid background in Computer Science. JupyterLab and data operation were taught in this class. So those students are the suitable candidates. We recruit 8 participants for the user testing (see Appendix F for informed consent). After we collect the feedback, future plans will be made to solve any newly discovered issues.

Chapter 4

IMPLEMENTATION

This section describes the technical implementation of JupyterLab_Voyager. The goal of this section is to provide an overview of the basic system structure of JupyterLab_Voyager as an extension to JupyterLab, explain the functions and designed workflow.

4.1 Voyager Class Overview

JupyterLab is a web-based open-source data science platform. It enables the user to work with documents and activities such as Jupyter notebooks, text editors, terminals, and custom components in a flexible, integrated, and extensible manner. JupyterLab offers a unified model for viewing and handling data formats. JupyterLab understands many file formats (images, CSV, JSON, Markdown, PDF, Vega, Vega-Lite, etc.) and can also display rich kernel output in these formats.

Fundamentally, JupyterLab is designed as an extensible environment. JupyterLab extensions can customize or enhance any part of JupyterLab. They can provide new themes, file viewers and editors, or renderers for rich outputs in notebooks. Extensions can add items to the menu or command palette, keyboard shortcuts, or settings in the settings system. In fact, the whole of JupyterLab itself is simply a collection of extensions. The base JupyterLab application includes a core set of extensions, which provide the features described in this user guide (notebook, terminal, text editor, etc.)

To use Voyager as a special 'Editor' for graph files in this JupyterLab_Voyager extension, first, we plan to create a special widget type called VoyagerPanel to hold

Voyager objects as a way to render targeted dataset. The class is described as follows:

class VoyagerPanel extends Widget implements DocumentRegistry.IReadyWidget

And its constructor function is :

constructor(options: VoyagerPanel.IOptions, app:JupyterLab, docManager: IDocumentManager)

Inside this constructor function, we record the file path, analyze the data type, and create Voyager interface inside a build-in HTMLElement. Due to the various data sources, the data type analysis is quite complex here. First, we need to check if the source file contains direct inline data (CSV, TSV); then we read its content and try to look for a 'data' field. If such field exists, the next step will be checking if it's JSON pairs or URL link. For URL link, we need to distinguish 'Local URL' and 'Web URL'. After that, we can finally process the data properly and pass it through Voyager API.

Certain attributes and functions need to be added as well besides its constructor function:

get context(): DocumentRegistry.Context

This function is used by JupyterLab to get the content of this widget, here we defined it as the Vega-Lite information for data visualization. So when this function gets called, the content will be extracted, and then exported or saved in VL.JSON form.

get ready(): Promise<void>

This function is to signal JupyterLab with a promise that resolves when VoyagerPanel is ready for further operations.

protected onActivateRequest(msg: Message): void

This function is to handle 'active-request' type of message including clicking and other operations. It will bring the selected Voyager interface to the front.

get isDirty(): boolean

This function is to test whether the widget content has been modified and needs saving. The 'close' icon will change based on the result.

private _onPathChanged(): void

This function is to re-link the widget with its local source file once the local file moves to a different place.

The entire source code of this project can be accessed online through the following link:

https://github.com/zzhangjii/jupyterlab_voyager

4.2 Open and Save for Local File Types

The opening file process is through factory options (figure 4.1), JupyterLab has a unified form of supporting for this type of service. First, we created a new widgetfactory class:

class VoyagerWidgetFactory extends ABCWidgetFactory <VoyagerPanel, DocumentRegistry.IModel>

Inside this class, a special createNewWidget function needs to be written to create the target widget for local file and return it.

protected createNewWidget(context: DocumentRegistry.Context): VoyagerPanel

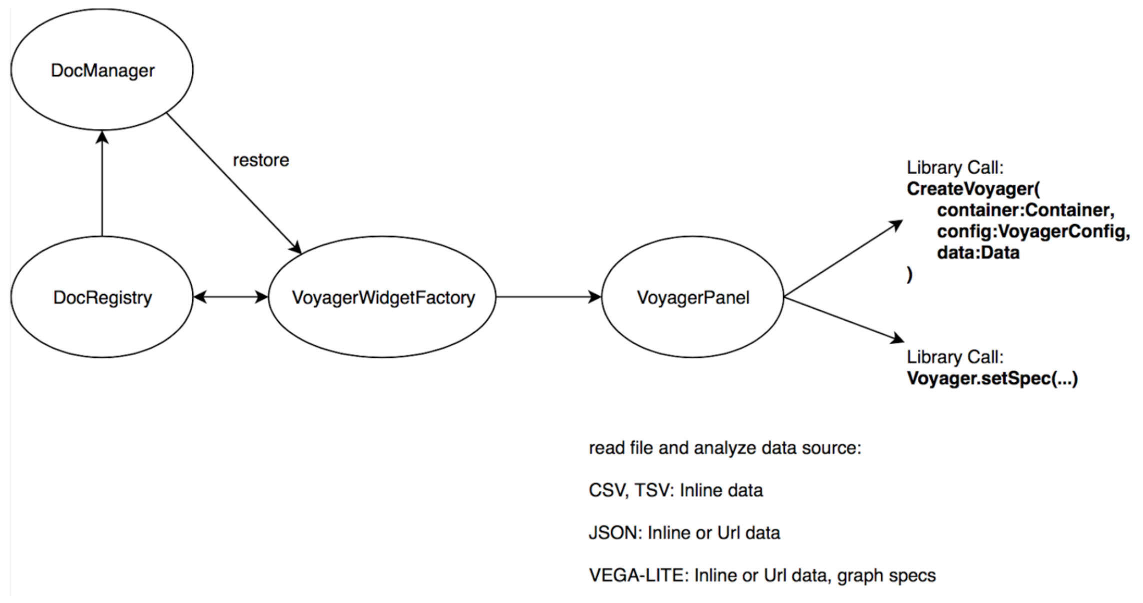


Figure 4.1: The System Design for Opening Local Files

When activating this plugin, we track certain file types through the Document Registry, namely: CSV, JSON, TSV, and Vega2 (which is a collection of filetypes, including VL.JSON file). Then, we build VoyagerWidgetFactory linking to the target file, and also add this new widget to tracker and Document Manager, which will help us restore this interface when refresh or system crashing happens.

In the save process, VoyagerPanel collects the current content of the graph by calling the context() function, and save the result back to the original file through the Document Registry. As discussed before, except for VL.JSON files, other file types cannot hold the whole visualization information. Therefore, to save the voyager content for those files, users have to create a VL.JSON file through Document Manager, and then export the content to this file (figure 4.2). To facilitate various kinds of user habits, we placed the export command in multiple places based on common usage: Main File button, Context Menu inside Voyager, and a special Toolbar (figure 4.3).

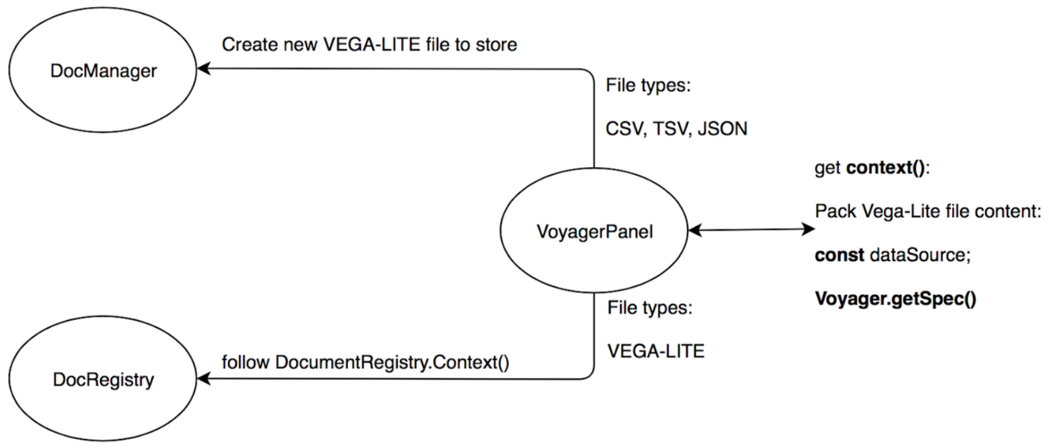


Figure 4.2: The System Design for Saving Back to Local Files

4.3 Open and Save for Notebook Charts

Notebooks are the core of the entire JupyterLab data science platform and ecosystem. Most of the data analysis happened here. A lot of time, the datasets only exist inside the notebook and don't really have an individual local file. The above implementation in Section 4.2 will not work if we plan to use the charts inside notebooks as the dataset source.

So, we created an additional widget class for this type of data

class VoyagerPanel_DF extends Widget implements DocumentRegistry.IReadyWidget

This VoyagePanel_DF class is quite similar to the previous VoyagePanel class, but since it's not linked with a real local file in JupyterLab's Document Registry, it doesn't have any file related attributes (filetype, filepath) or functions(onPathChanged...). The save option will not work and the only way to store the Voyager work is to use the Export button.

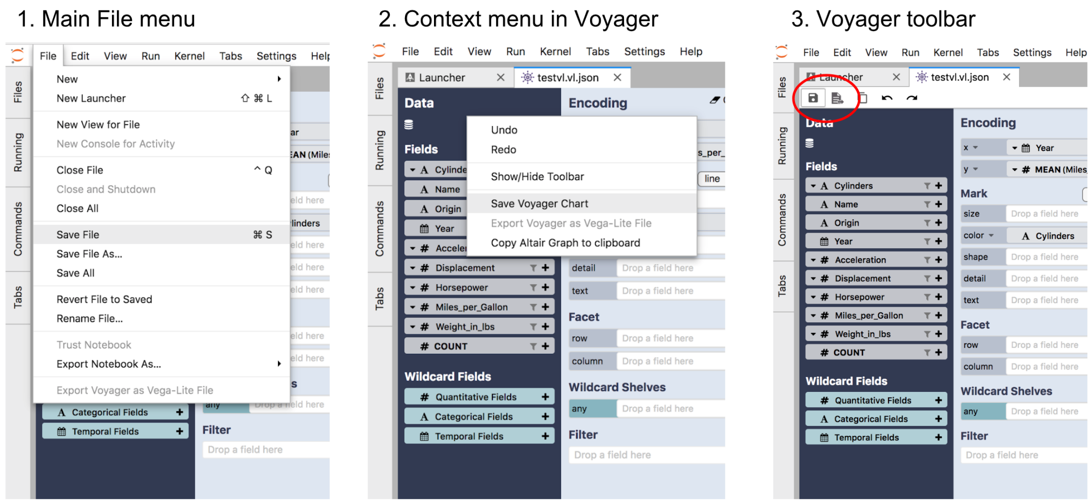


Figure 4.3: Operating Options: Saving Back to Local Files

One flaw of this design is that since there's no local file link for VoyagerPanel_DF class and currently we have no way to locate the original notebook chart during a restoration process; the whole VoyagerPanel_DF widget will be gone and all the previous work within it will be lost after a restoration. Therefore, users must save their work frequently. JupyterLab team is developing a new strategy to solve this problem using a temporary file system. This will be explained in 'Future Work' section in Chapter 7. Other than this, the VoyagerPanel_DF interface appears identical with the VoyagerPanel interface to the users. The main operations are the same (figure 4.4).

4.4 Transition between Voyager and Notebooks

One important goal this project tries to achieve is to build a smooth workflow path between the notebook, Voyager and possibly local files. We have discussed the workflow from the notebook to Voyager in section 4.3, and in this section, we plan to build the reverse way back: from Voyager to notebook.

The ideal workflow between notebook and Voyager would be: open notebook

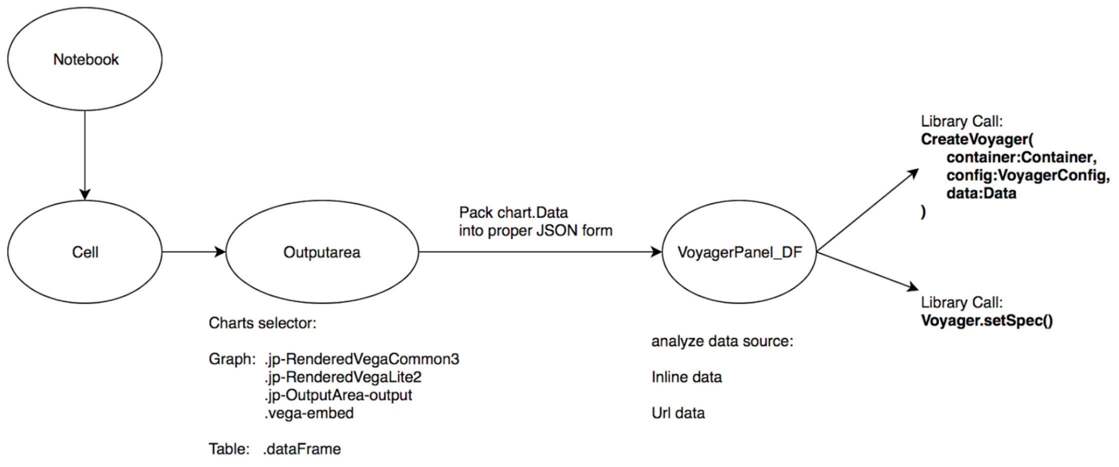


Figure 4.4: System Design: from Notebook to Voyager

chart in Voyager - do visualization analysis - export this graph back to the notebook (figure 4.5). This idea is simple, but we encountered some problems in the design phase. A Voyager interface could be opened from a local file or from a notebook chart, so it's not guaranteed that there is an original notebook to export the Voyager content to in the third step. We have tried creating a new notebook file as a solution, but after testing, we quickly decline it since it left too many unwanted notebook files after working. When using notebook charts as dataset sources, users usually prefer to export the Voyager back to the same cell or the cell next to the original one. However, we cannot achieve that in the current JupyterLab, since the notebook system doesn't have a way to trace each notebook cell. Eventually, we decided to allow user to copy the Voyager content to clipboard, and let the user decide which notebook and which cell to paste to (figure 4.6).

Notebooks are mainly running in Python environment, so in order to create the graph, we have to build the Altair code using the Vega-Lite content. However, the readable Altair code is very complex. Since Altair has too many variables, cross dependencies may happen and forms may change in new releases, pre-written static

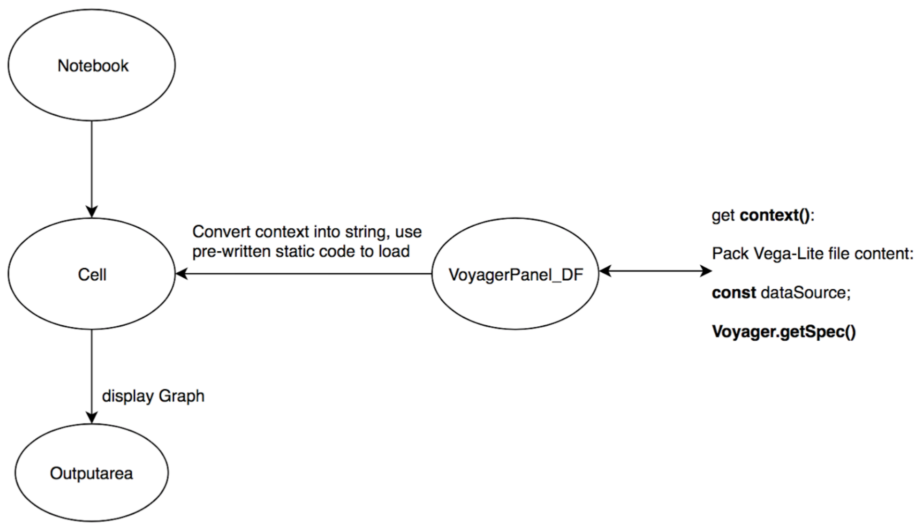


Figure 4.5: System Design: from Voyager to Notebook

python code couldn't work well to handle all those cases. Besides, the idea of developing JupyterLab_Voyager is to free the user from memorizing Altair library code, so to use and display a readable Altair code structure for users should not be our focus. Therefore, the solution we use here is to copy all the Vega-Lite information as strings and use 'from_dict()' function to convert it into graph displays inside notebook cells.

Additionally, we provide a link between a local graph file and the notebook interface, to allow users directly load this graph inside notebook. This command is developed using the same strategy, so it's facing the similar problem: no readable Altair codes in current version.

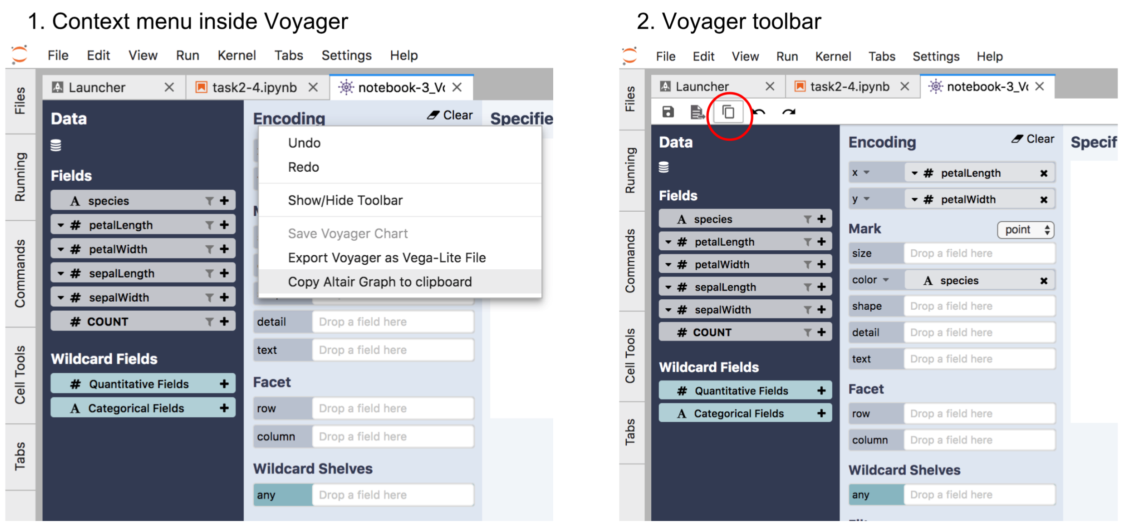


Figure 4.6: Operating Options: from Voyager to Notebook

Chapter 5

SYSTEM EVALUATION

As the main goal of this project is to improve JupyterLab's DAV system, our strategy is to incorporate the Voyager interface, which is a well-established visualization tool, into JupyterLab. The hope is that JupyterLab users can make full use of JupyterLab_Voyager system. To prove that, a usability test is needed here to determine the extent to which this extension is understood, easy to learn, easy to operate and attractive to the data scientists using the original JupyterLab.

For software developers, systems may be built 100% in accordance with the specifications. Yet, they may be 'unusable' when it lands in the hands of the real users. So, we designed the usability test for our JupyterLab_Voyager extension, which includes all the operations in designed work-flows involving Voyager, notebook and local file system. The purpose of this event is to review the user interface and other human factors of JupyterLab_Voyager with the people who will be using the application. This is to ensure that the layout, sequence and other designs enable the desired functions to be executed as easily and intuitively as possible.

5.1 User-testing Design

To design a usability test, first, we need to understand the following issues:

1) Who are the users of the system.

2) What are their business needs.

3) Try to mimic their behaviors.

Clearly, the users of this JupyterLab_Voyager extension are data scientists, who also have to be JupyterLab users. Since our work is an extension to the JupyterLab platform, if a data scientist is not familiar with it, then our test result would more likely to reflect the usability of JupyterLab itself. So, to find the right target users, we designed a pre-screening survey (Appendix D) and send it out to Cal Poly students who have taken Data Science 301 class (major content is learning data science in JupyterLab). Through this pre-screening, we selected 8 users who are familiar with JupyterLab and have some basic knowledge about Vega-Lite, Altair and other key data visualization components.

The user-needs here are relatively complex. In general, users want a better data analysis and visualization tool in JupyterLab, and our work here is to incorporate a good DAV tool: Voyager, into JupyterLab. If we simply test whether users can perform data visualization better with this extension, the result will largely reflect the quality of Voyager itself, not our JupyterLab_Voyager. And because the team developed Voyager has already conducted plenty of usability test on it, our work here would be redundant. Therefore, our real focus should be the easiness and effectiveness of the extension itself: to test if users can easily use Voyager interface inside JupyterLab, and if the designed workflow can help with the data analysis and visualization.

To find out if users can successfully perform the designed workflow between Voyager and the notebook ecosystem using JupyterLab_Voyager, we tried to mimic target users' behaviors and designed a series of tasks. Then, based on that, we compiled a script containing very detailed operations in each step (table 5.1 and Appendix E). The first section includes opening Voyager from various types of data sources, and testing the usability of importing and exporting Voyager data. The second section focuses on testing the data workflow between Voyager interface and notebook.

5.2 User-testing Methods

5.2.1 Materials and Devices

Candidates will be invited to the Project Jupyter office (Graphic Arts Building on Cal Poly campus) to participate in the user test, which includes performing certain tasks on a computer, and a post-test survey focusing on their feelings and concerns. The computer and all other necessary devices will be prepared by the investigator.

5.2.2 Procedures

Pre-screening surveys will be sent out to Cal Poly students who have taken Data Science 301 Class from the faculty advisor Prof. Brian Granger. Candidates passed the screening will be invited to participate the user test. The investigator will schedule a 30 - 45 minutes time period with a participant to hold this face-to-face interview & user test.

At the beginning of the test, a consent form (Appendix F) will be read to the participant and her/his signature will be obtained. The student investigator will conduct the user test only after the subject has given her/his written consent.

First, the participant will be asked to perform a series of data visualization tasks on a computer using the JupyterLab Voyager extension. Those tasks will be clearly instructed by the investigator step by step. Participant's operations (mouse clicking, file dragging...) on the screen will be recorded by a screen-tracking software and the data will be stored on this computer.

Second, the participant will be interviewed about their feelings and concerns.

Third, after finishing all the user tests, the investigator will analyze the original data to calculate the success rate for each task and identify flaws in design. Once all of those information are extracted, organized and listed in tables, the original data files will be deleted.

After doing the user test with all participants, the investigator will evaluate the usability of this JupyterLab_Voyager extension, and summarize this research in the master thesis paper, and potentially a peer reviewed academic paper.

5.2.3 Confidentiality

No personally identifiable information (like respondent's name ,age, gender, address...) in any form (text/video/audio/photographic recording...) will be collected during this whole process of this research (pre-screening, user test...). A participant will be asked to use an unidentifiable nickname and code to represent herself/himself in this research.

5.3 Result Analysis

We successfully recruited 8 participants, and arranged 5 of them for the first round of user testing. Their background information are summarized in Appendix H. After conducting the usability testing, all the data were analyzed task by task. Each recording was watched through, step-by-step, to see whether a participant could complete each task in table 5.1. During each task, user's first attempt (real action: clicking...) and user's final attempt will be specially marked. For user's first attempt, it directly reflects user's thought about how to operate or solve this problem. If a user keeps failing and eventually did not complete the task as expected, it would be marked as a task failure, and notes were taken on what action was made as well as the time in the video when it occurred. The summarized results can be found in Appendix G. For each participant, the status of each task will be analyzed together with his/her technical background information in the pre-screening survey.

As we can see from figure 5.1, the error rates of step 1, 3, 5, 7 are minimum, indicating 'Open file in Voyager' is not a problem for JupyterLab users, which is understandable as opening through context menu is the default option in JupyterLab and the users should be well aware of it already. But for the saving and exporting part (step 2, 4, 6, 8), the error rates are higher. All user actually noticed the difference between saving Voyager and saving the default file type like CSV, but not everyone realized the export is the correct action for non-Vega-Lite files at the first place. But with enough exploration time, all of the users managed to find the right button and finished the tasks at the end, as indicated by the zero task failing rate of step 1-8 in figure 5.2.

The designed workflow between notebook and Voyager (step 9-16) is more challenging than the file system part. As shown in figure 5.1, error rate of step 09 is not high, meaning most users don't have trouble with opening notebook charts in Voyager through context menu. But from figure 5.2, we learnt that the user having error here eventually failed this task. More errors happened during the export stage (step 10 and 12). Lots of users started to click the main menu to look for these export options. Even with enough exploration time, several users still failed these tasks at the end. The reason behind this is that they never used the right click to open the context menu either on notebook charts or inside Voyager interface. Considering the previous failure in step 09 is also related with the right click, the current design for putting those export options only in context menu is probably not the best choice.

Sometimes, errors and failures are inevitable when using software for the first time, and the best we can do is to make sure it is easy to learn so user won't make the same mistake again. To test the learning curve of Jupyterlab_Voyager, our usability test contains some repeats about similar actions. Opening&Saving work can be considered as same group of actions, so step 5-8 can be considered as repeats for step 1- 4. The second part of testing on notebook and Voyager (step 9-16) also includes similar

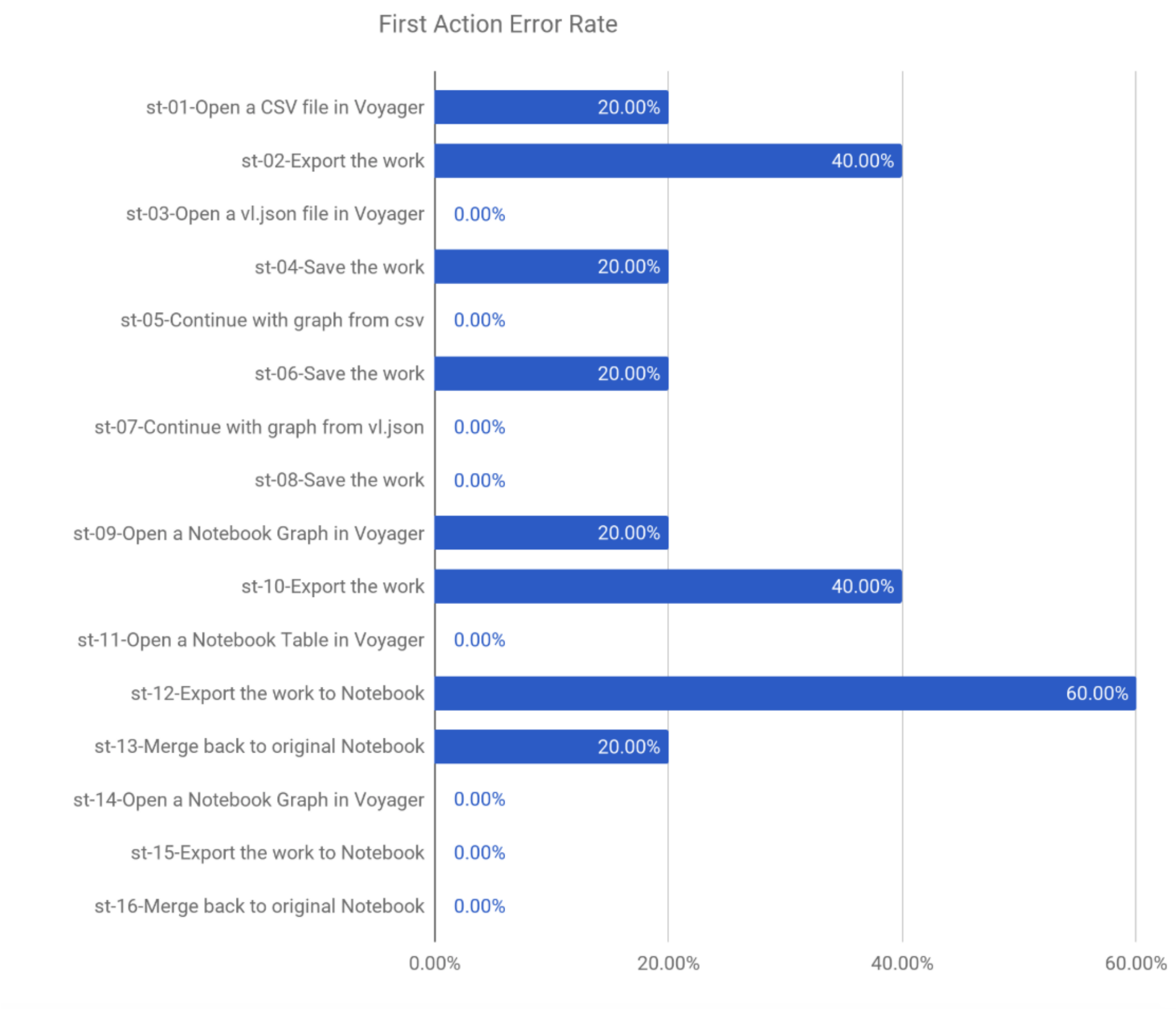


Figure 5.1: Error Rate of User's First Action

repeats. We have step 09, 10 as preparations, step 11-13 as the first round and step 14-16 as the second round. As indicated in figure 5.1 and 5.2, after the first time error or failure, the success rate of the second time action was close to 100%, indicating a steep learning curve for this JupyterLab_Voyager extension. In this usability test, the overall user performance is very good, as indicated by the low error rate and failing rate. We conclude that the JupyterLab_Voyager extension has a good usability. And it's very clear that the performance is positively correlated with the experience of using JupyterLab notebooks, since all the failures happened on the least experienced

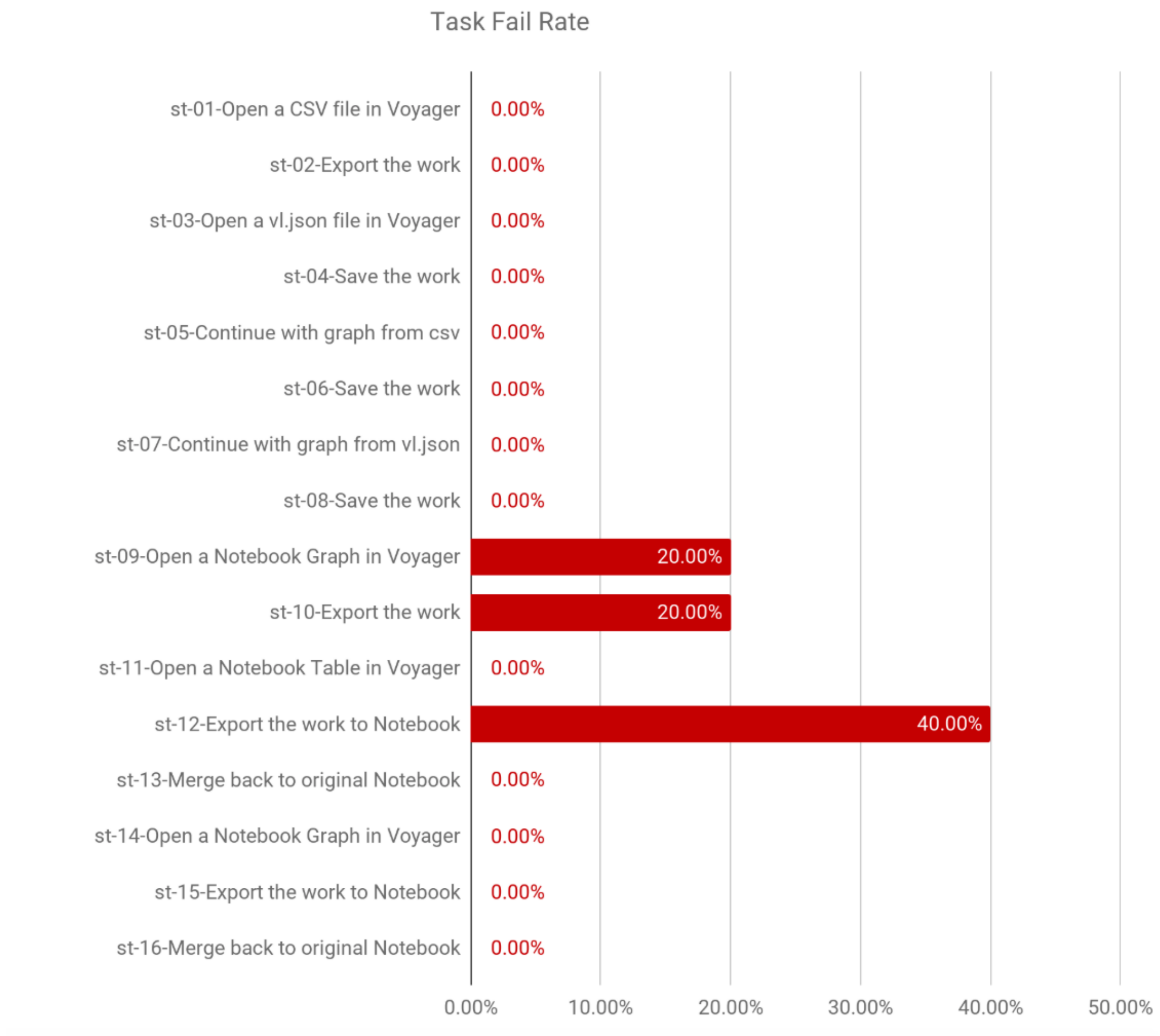


Figure 5.2: Fail Rate of Each Task

JupyterLab users.

Step	Instructions
FILE SYSTEM and VOYAGER	
st01	Open a csv file in Voyager
	Describe the dataset and create a graph
st02	Export the work
st03	Open a vl.json file in Voyager
	Describe the dataset and create a graph
st04	Save the work
st05	Continue with graph from csv
	Modify the graph: change fields, color, chart type
st06	Save the work
st07	Continue with graph from vl.json
	Modify the graph: change fields, color, chart type
st08	Save the work
NOTEBOOK and VOYAGER	
st09	Open a Notebook Graph in Voyager
	Modify the graph: change fields, color, chart type
st10	Export the work
st11	Open a Notebook Table in Voyager
	Create a graph
st12	Export the work to Notebook
st13	Merge back to original Notebook
st14	Open a Notebook Graph in Voyager
	Modify the graph: change fields, color, chart type
st15	Export the work to Notebook
st16	Merge back to original Notebook

Table 5.1: User Testing Instructions

Chapter 6

CONCLUSION

This thesis presents JupyterLab_Voyager, a plugin extension which incorporates a well established visualization tool: Voyager, into the DAV system of JupyterLab.

The JupyterLab_Voyager extension is an enhancement to the original DAV system. It's designed to solve the usability issues reported previously. As we interviewed data scientists and analyzed their pain points, we concluded that Voyager is the right tool to use. To improve the DAV system in JupyterLab, we implemented the extension using Voyager as a new graph editor supporting CSV, TSV, JSON and VEGA-LITE type files. We designed several workflows between the Voyager and notebook for users, hope they can make use of this GUI based data visualization system while still enjoying the powerful Jupyter notebook platform.

To prove the usability of this design, we recruit real JupyterLab users from data science class students, who have some experience with both data visualization and JupyterLab notebook, to conducted usability tests. The test focused on the file system of JupyterLab_Voyager (Open, Save and Export from various types of data sources) and the data workflow between Voyager interface and the notebook. At the end, most users performed well in this usability test. The fail rate is low and the learning curve is steep, indicating this JupyterLab_Voyager extension has a good usability.

Although it's only this first edition and it needs improvements in the future, this software can help analysts learn about the dataset and engage better in both open-ended exploration and targeted question answering inside JupyterLab.

Chapter 7

FUTURE DIRECTIONS

In this project, we designed and developed JupyterLab_Voyager to improve the DAV system in JupyterLab, and we demonstrated its usability through user-testing. Although this extension is effective and useful, it's still in the early stage. Flaws and limitations exist in the current design due to technical difficulty and designing problems. These issues can be our future working directions in terms of both development and usability study. Further areas of research to improve the data analysis and visualization in JupyterLab will be discussed as follows:

7.1 Temporary File System

In the current JupyterLab_Voyager, all those VoyagerPanel_DF widgets were created based on the datasets in notebook charts. They are not linked with local files in Document Registry, and there's no file path for the restorer. So, these interfaces cannot be restored, meaning once crash happens or user does a refresh, all the data inside those VoyagerPanel_DF will be gone. This issue is a huge flaw for JupyterLab_Voyager. At the beginning, we have tried other designs like creating a local file for the selected chart, but it was declined immediately after testing, since users left too many newly created files during data exploration, and most of those files are totally unwanted afterwards.

The JupyterLab core developing team is working on a temporary storage system. With this new design, the JupyterLab will allocate a certain amount of space in the system for storing the temporary files during working. This change will provide us the necessary tools to fix this flaw. In the future version of JupyterLab, when users open a notebook chart with Voyager, we will create a temporary VL.JSON file inside the special allocated space to store the Vega-Lite information for this Voyager. So it will be linked with this temporary file in Document Registry. Its file path will be recorded in the restorer, so the restoration will work properly. And, upon exiting, this temporary storage space will be wiped out, and all the unwanted temporary files will be cleaned up.

7.2 Improving Workflow between Voyager and Notebooks

The designed workflow from notebook to Voyager is through copying all the Vega-Lite information as strings and use 'from_dict()' function and pre-written static python code to convert it into graph displays inside notebook cell. During this process, users need to do 4 actions: Copy - Select a notebook cell - Paste - Run. When using notebook charts as dataset sources, users usually prefer to export the Voyager back to the same cell or the cell next to the original one. However, we cannot achieve that in the current JupyterLab, since the notebook system doesn't have a way to trace each notebook cell.

Another team in the JupyterLab is working on an improved notebook cell management system right now. With this new system, each cell will have a unique ID. With this ID, we can to track which cell does the Voyager source data come from, and will be able to trace back to that cell for more operation like replace the original graph with the modified one from Voyager. We will be able to simplify the current 4-step workflow and provide user with better experience.

7.3 Improving Voyager Functions

The Voyager itself is a newly developed software tool for graphics and also needs improvements. As we test it inside JupyterLab, we also found out a lot of flaws and issues:

(1) Wildcard Fields are confusing, difficult to understand or use;

(2) Limited operations for each graph;

(3) Only work for Units, doesn't support layers;

(4) Cannot personalize the data range in file;

(5) Cannot set groups among fields;

(6) No interactive function implemented;

Therefore, we are also collaborating with the Data Voyager team in Interactive Data Lab at University of Washington to solve those issues.

BIBLIOGRAPHY

[1] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics*, 15(6), 2009.

[2] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[3] B. Data. for better or worse: 90% of worlds data generated over last two years. *SCIENCE DAILY, May*, 22(3), 2013.

[4] J. Heer and B. Shneiderman. Interactive dynamics for visual analysis. *Queue*, 10(2):30, 2012.

[5] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[6] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[7] S. Murray. *Interactive Data Visualization for the Web: An Introduction to Designing with.* " O'Reilly Media, Inc.", 2017.

[8] J. Nielsen. 10 usability heuristics for user interface design. *Nielsen Norman Group*, 1(1), 1995.

[9] F. Perez and B. E. Granger. Project jupyter: Computational narratives as the engine of collaborative data science. *Retrieved September*, 11:207, 2015.

[10] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonnier. The jupyter/ipython architecture: a unified view of computational research, from interactive exploration to communication and publication. In *AGU Fall Meeting Abstracts*, 2014.

[11] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017.

[12] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2016.

[13] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *The Craft of Information Visualization*, pages 364–371. Elsevier, 2003.

[14] I. Standardization. Ergonomics of human-system interaction: Part 210: Human-centred design for interactive systems, 2010.

[15] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[16] S. Swaid, M. Maat, H. Krishnan, D. Ghoshal, and L. Ramakrishnan. Usability heuristic evaluation of scientific data analysis and visualization tools. In *International Conference on Applied Human Factors and Ergonomics*, pages 471–482. Springer, 2017.

[17] R. W. White and R. A. Roth. Exploratory search: Beyond the query-response paradigm. *Synthesis lectures on information concepts, retrieval, and services*, 1(1):1–98, 2009.

[18] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE transactions on visualization and computer graphics*, 22(1):649–658, 2016.

[19] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2648–2659. ACM, 2017.

[20] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408. ACM, 2003.

APPENDICES

Appendix A

JUPYTERLAB USER INTERVIEW

1. When you get a new data set to work with. How much of an idea do you have of what you'll be doing with that data set?

2. At what point in your process do you usually feel like you have a clear goal in mind?

3. Think about a time you analyzed a data set you were initially unfamiliar with.

—- a. What program(s) did you use to to explore the data set?

——— i. [If not Jupyter] Can you think an example where you explored it with the Jupyter Notebook?

—- b. Let's talk a little bit about exploring data in Jupyter, go ahead and keep a specific example in mind.

——— i. What are the first steps you took to explore the data set?

——— ii. And then what did you do?

——— iii. At what point did you feel you were ready to begin analysis?

——— iv. Tell me a little about your understanding of the data set at that point?

—————- 1. How confident were you with your understanding at that point?

—————- 2. Jumping ahead a bit, now that you have analyzed the data set, how did you understanding of the data at that point compare to your understanding of it now?

—- c. Tell me about a time you thought you were familiar enough with a data set to work with it, but it turned out to be more complex than you originally thought

——— i. Thinking back, what helped you conclude you were more familiar with the set than you actually were?

——— ii. How often would you say this happens?

——— iii. What are some things you do to avoid this?

————- 1. Are they effective?

——— iv. What impact do you think exploring the data visually would have in this process?

4. Think about a time you analyzed a data set, one that really sticks out in your memory. What did you do after you'd familiarized yourself with the data set? (This could be for the same, or a different data than you've been previously talking about)

—- a. Is this typical of your workflow?

——— i. [if no] What was different about this particular instance?

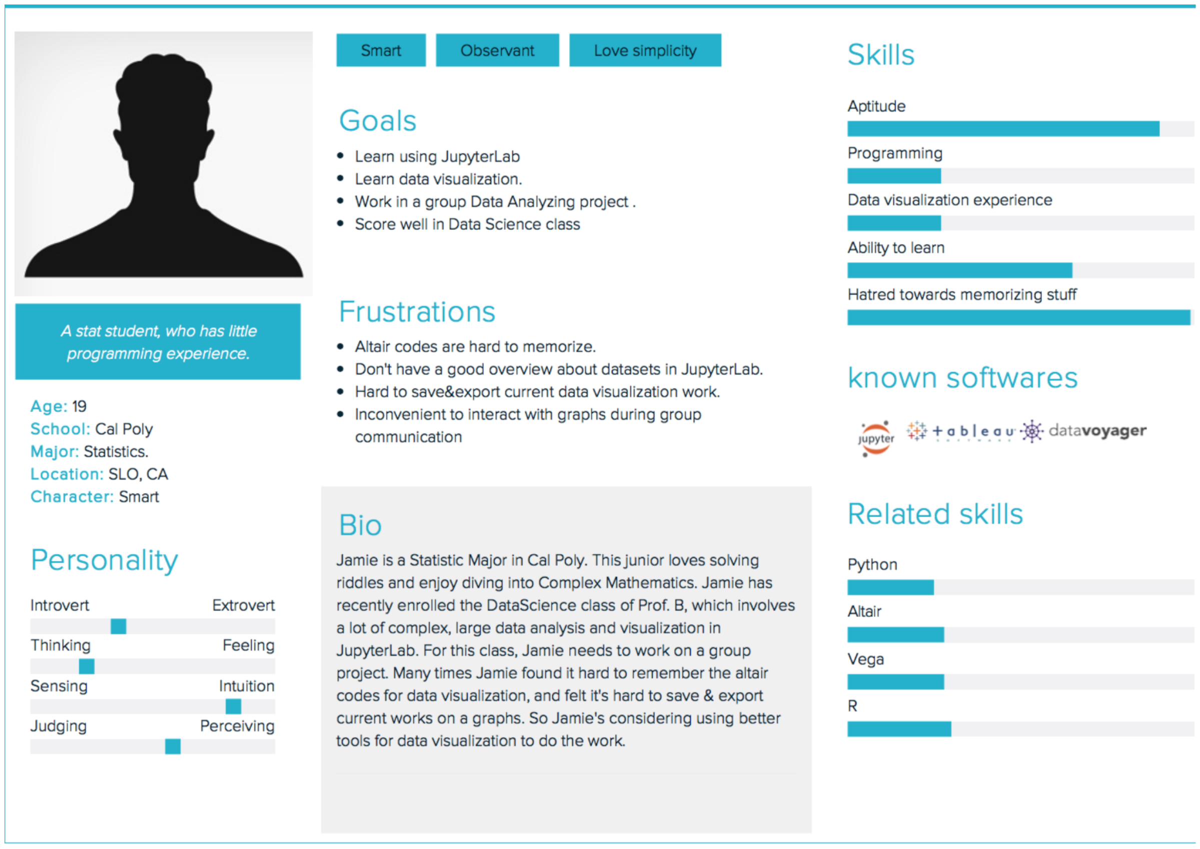
——— ii. How would your normal workflow be in this situation?

—- b. How do you build visualizations in your current workflow?

——— i. What impact would it have on your workflow, if you could access data from your notebooks in Voyager Extension edit the visualization, and export those results back into your notebook?

Appendix B

BEGINNER LEVEL USER PERSONA



Smart · Observant · Love simplicity

A stat student, who has little programming experience.

Age: 19
School: Cal Poly
Major: Statistics.
Location: SLO, CA
Character: Smart

Personality

Introvert
Extrovert
Thinking
Feeling
Sensing
Intuition
Judging
Perceiving

Goals

- Learn using JupyterLab
- Learn data visualization.
- Work in a group Data Analyzing project .
- Score well in Data Science class

Frustrations

- Altair codes are hard to memorize.
- Don't have a good overview about datasets in JupyterLab.
- Hard to save&export current data visualization work.
- Inconvenient to interact with graphs during group communication

Bio

Jamie is a Statistic Major in Cal Poly. This junior loves solving riddles and enjoy diving into Complex Mathematics. Jamie has recently enrolled the DataScience class of Prof. B, which involves a lot of complex, large data analysis and visualization in JupyterLab. For this class, Jamie needs to work on a group project. Many times Jamie found it hard to remember the altair codes for data visualization, and felt it's hard to save & export current works on a graphs. So Jamie's considering using better tools for data visualization to do the work.

Skills

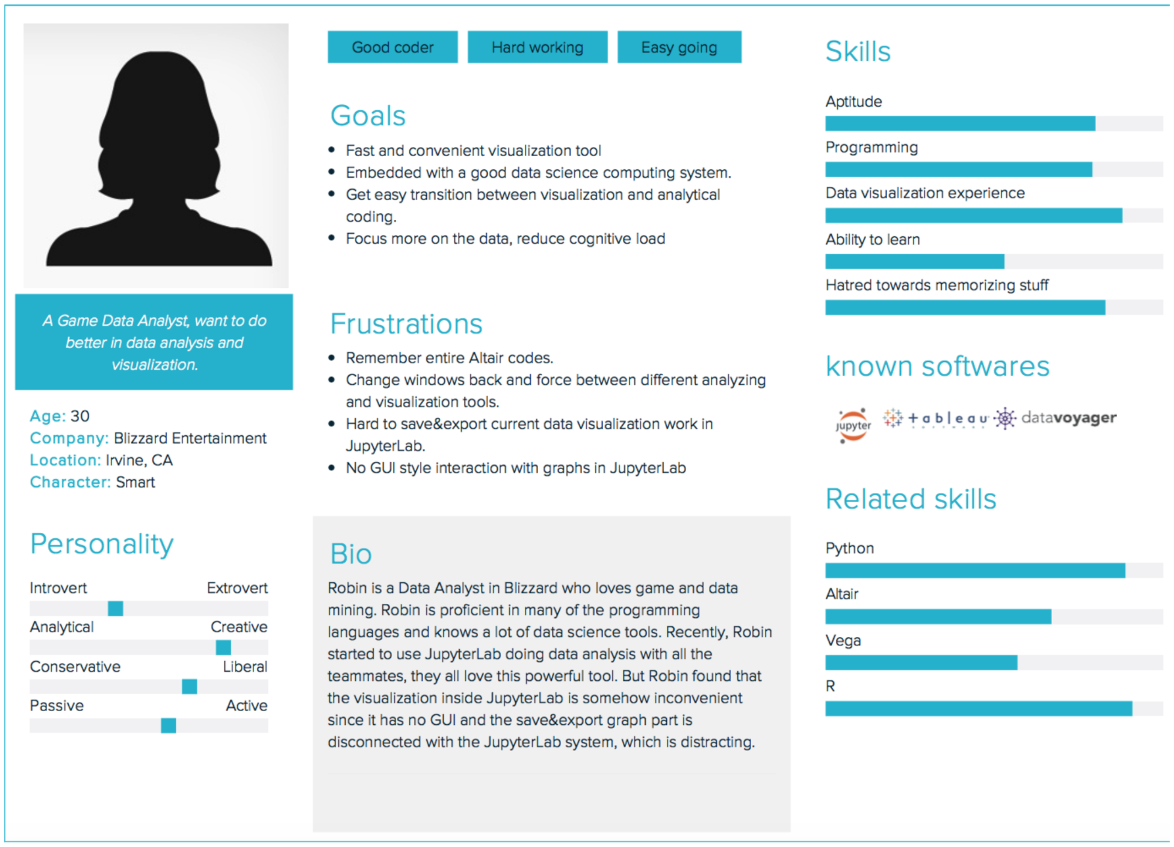
Aptitude
Programming
Data visualization experience
Ability to learn
Hatred towards memorizing stuff

known softwares

Related skills

Appendix C

EXPERT LEVEL USER PERSONA



A Game Data Analyst, want to do better in data analysis and visualization.

Age: 30
Company: Blizzard Entertainment
Location: Irvine, CA
Character: Smart

Personality

Introvert — Extrovert
Analytical — Creative
Conservative — Liberal
Passive — Active

Good coder | Hard working | Easy going

Goals

- Fast and convenient visualization tool
- Embedded with a good data science computing system.
- Get easy transition between visualization and analytical coding.
- Focus more on the data, reduce cognitive load

Frustrations

- Remember entire Altair codes.
- Change windows back and force between different analyzing and visualization tools.
- Hard to save&export current data visualization work in JupyterLab.
- No GUI style interaction with graphs in JupyterLab

Bio

Robin is a Data Analyst in Blizzard who loves game and data mining. Robin is proficient in many of the programming languages and knows a lot of data science tools. Recently, Robin started to use JupyterLab doing data analysis with all the teammates, they all love this powerful tool. But Robin found that the visualization inside JupyterLab is somehow inconvenient since it has no GUI style interaction with graphs and the save&export graph part is disconnected with the JupyterLab system, which is distracting.

Skills

Aptitude
Programming
Data visualization experience
Ability to learn
Hatred towards memorizing stuff

known softwares

jupyter, tableau, datavoyager

Related skills

Python
Altair
Vega
R

Appendix D

PRE-SCREENING SURVEY

Survey

1. Email Address for Contact:

2. Job Title (Major if you're a student):

3. Known programming languages:

4. Programming Experience:

—- Less than a year

—- 1-2 years

—- 2-5 years

—- 5+ years

5. Rank your familiarity with Jupyter Notebooks (from 1 - 6)

With 1 as minimum familiarity and 6 as an expert

6. Rank your familiarity with Altair (from 1 - 6)

With 1 as minimum familiarity and 6 as an expert

7. Rank your familiarity with Matplotlib (from 1 - 6)?

With 1 as minimum familiarity and 6 as an expert

8. Rank your familiarity with Vega-Lite (from 1 - 6)?

With 1 as minimum familiarity and 6 as an expert

9. Briefly describe your experience with data analysis and visualization.

Appendix E

USABILITY TEST SCRIPT

* Verbal instructions*

For this section of the test, I want to you open various files using the voyager extension. Please remember to speak what you're thinking out loud.

Note: this section is to test the import and export data for jupyterlab_voyager

1. Open testcsv.csv file in Voyager (through context menu);

- Describe what fields does the dataset has?
- Create a graph to show the relationship between X(State) and Y(total raised money amount).
- Use X(state) based color in this graph.
- Which state raised the most amount of money
- Now save&export your work to a new file A.

2. Open testvl.VL.JSON file in Voyager (through context menu);

- Describe what fields does the dataset has?
- Create a graph to show the relationship between X(the number of cylinders) and Y(average horsepower).
- Change to view to show the relationship between X(the number of cylinders) and Z(the max horsepower).
- Some modification in shape&color (color the graph based on origins)

- What did you learn about Cylinder and horsepower?
- Now save your work, and export your work to a new file B.

3. Continue working on the previous file in step A in Voyager;

- Describe your thought about which file to choose for continue working,
- Open your target file;
- Change the view to show the relationship between Z(funded date) and aggregated Y(max raised amount).
- Some modification in shape&color (color based on category, use line)
- Which category provide the most funding?
- Now save your work.

4. Continue working on the previous file in step B in Voyager;

- Describe your thought about which file to choose for continue working,
- Open your target file
- Change to view to show the relationship between Y(Year) and aggregated W(average of MPG).
- Some modification in shape&color (Color it based on Cylinder)
- What did you learn about the energy efficiency and cylinder number.
- Now save your work.

5. For this next task, I'm going to put you in the middle notebook in progress. I want you to locate and open the graph you see using the voyager extension. (This should already be open in a workspace call this task2-4)

- Select a graph and describe it (age&sum of people);
- Open this graph inside Voyager;
- Describe what you see during the process;
- Change to view to show the relationship between Y(year) and aggregated W(sum of people).
- Close this Voyager widget

6. Continue working in the middle notebook. I want you to locate and open the table you see using the voyager extension.

- Select a table and describe it;
- Open a table inside Voyager
- Describe what you see during the process ,
- Change to view to show the relationship between Y(max heart rate) and aggregated W(chest pain level).
- Now export your work to a local file C.
- Close the widget.
- Re-open your previous work in C;
- Continue working on the previous graph created from notebook table, make some modification in shape&color (mark it with color)

- Save your work and exit.

Transition between notebook and voyager

Note: this section is to test the data workflow between voyager and notebook

1. Open a local VL.JSON file, which contains the datasource and the specs for a Vega-Lite graph, inside a new notebook(through context menu); (For this step have a voyager file ready and a relating notebook D with a 'bad' graph ready open in a workspace.Ask the user to export the 'good' voyager graph into the notebook.)

- Locate a VL.JSON file (the previous file A/B/C) inside JupyterLab
- Open it in a notebook through context menu
- Describe what you see
- Compare it with the Graph in the already-opened notebook D
- Move this cell to D and save it.

2. For this next task, I'm going to put you in the middle notebook in progress. I want you to locate a graph.

- Open this graph inside voyager
- Change the graph type and fields
- Copy the graph back into notebook

3. You in the middle notebook in progress. I want you to locate a table;

- Open this table inside voyager
- Create a graph to show the relation between X and Y

- Export this graph to a local file
- Copy the graph back into notebook

Appendix F

CONSENT FORM

Informed Consent Form

INFORMED CONSENT TO PARTICIPATE IN A RESEARCH PROJECT:

JupyterLab_Voyager: a data visualization enhancement in JupyterLab

A research project on JupyterLab_Voyager is being conducted by Ji Zhang, a graduate student in the Department of Computer Science, and Prof. Brian Granger in Department of Physics, at Cal Poly, San Luis Obispo. The purpose of the study is review the user interface and other human factors of JupyterLab_Voyager with the people who will be using the application through software usability test. This is to ensure that the layout, sequence and other designs enable the desired functions to be executed as easily and intuitively as possible.

You are being asked to take part in this study by perform a series of data visualization tasks on computer using JupyterLab_Voyager, and to complete an interview with the researcher, and finish a survey at the end. Your participation will take approximately 30-45 minutes. Please be aware that you are not required to participate in this research and you may discontinue your participation at any time without penalty. If you decide to participate, you also don't have to answer any questions you choose not to answer.

There's no possible risks associated with participation in this study. If you should experience nervousness when you failed to perform a task using this software on computer during the test and feel you need help, please be aware that you may contact the campus Psychological Services, Health Center at 805-756-1211 for assistance.

Your confidentiality will be protected. In this research, only participant's operations (mouse clicking, file dragging...) on the screen will be recorded by a screen-tracking software. Once the investigator has finished the research, those original data will be deleted. During the research, no personally identifiable information in any form will be collected during this whole process of this research. A participant will be asked to use an un-identifiable nickname (pick some random letters plus a random number) to represent herself/himself in this research. Potential benefits associated with the study include general feeling of reward for being able to help with this research, and general feeling of reward for being able to contribute to Jupyter community.

If you have questions regarding this study or would like to be informed of the results when the study is completed, please feel free to contact the student researcher Ji Zhang at jzhang49@calpoly.edu, or the student researcher's faculty advisor Prof. Brian Granger at bgranger@calpoly.edu. If you have concerns regarding the manner in which the study is conducted, you may contact Dr. Michael Black, Chair of the Cal Poly Institutional Review Board, at (805) 756-2894, mblack@calpoly.edu, or Ms. Debbie Hart, Compliance Officer, at (805) 756-1508, dahart@calpoly.edu.

If you agree to voluntarily participate in this research project as described, please indicate your agreement by signing below. Please keep one copy of this form for your reference, and thank you for your participation in this research. If interested, you will received a copy of the thesis paper about this research.

Signature of Volunteer:

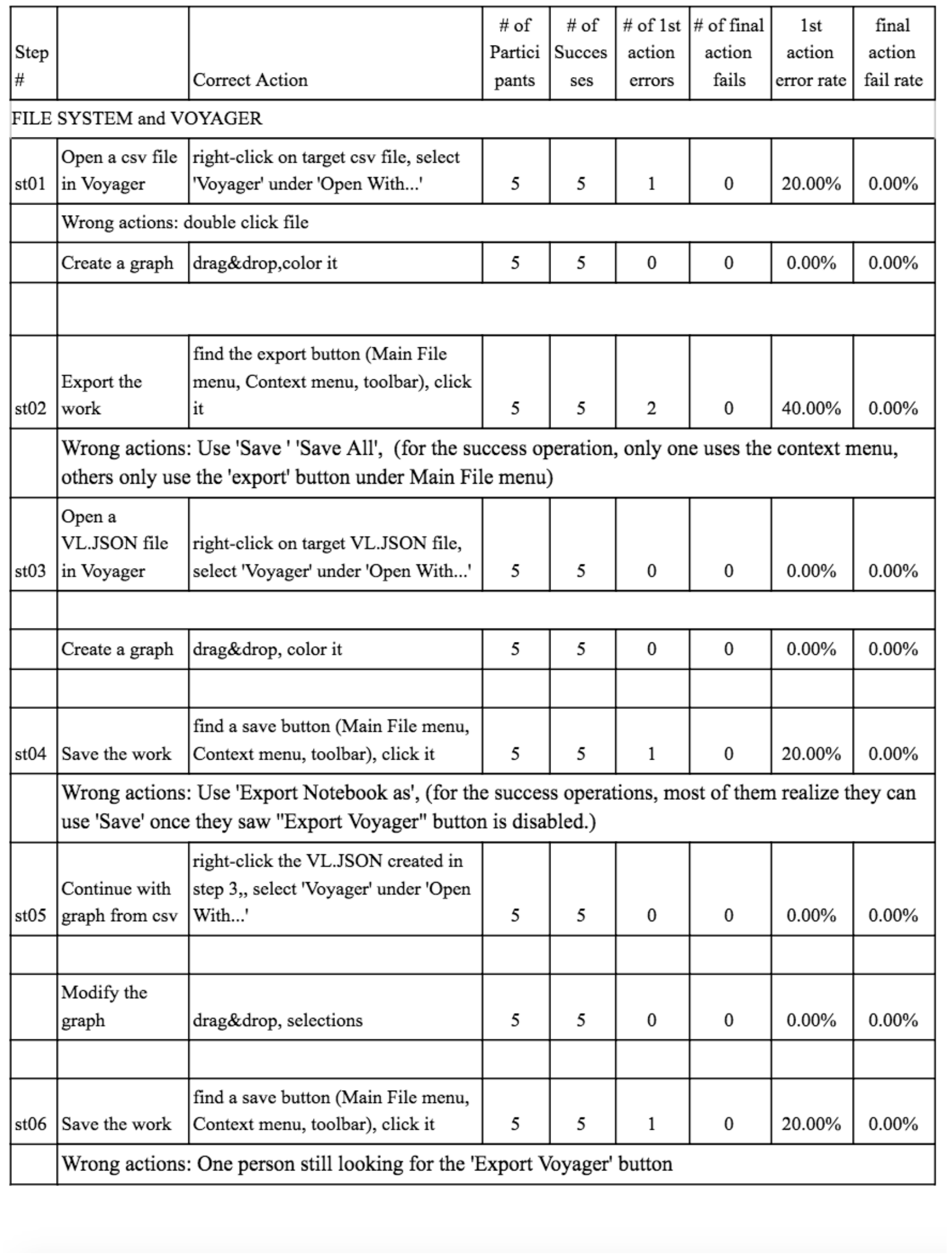
Date:

Signature of Researcher:

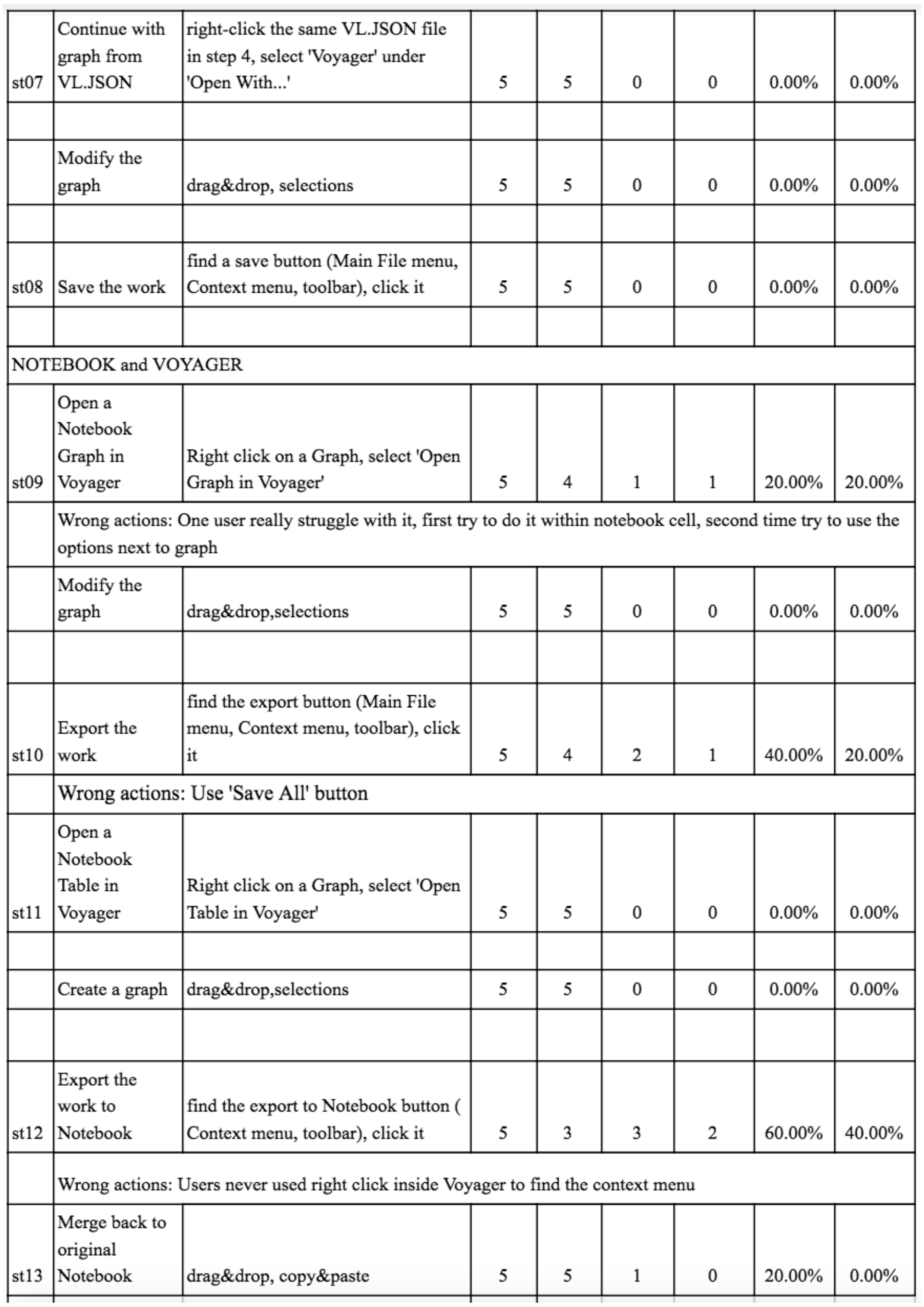
Date:

Appendix G

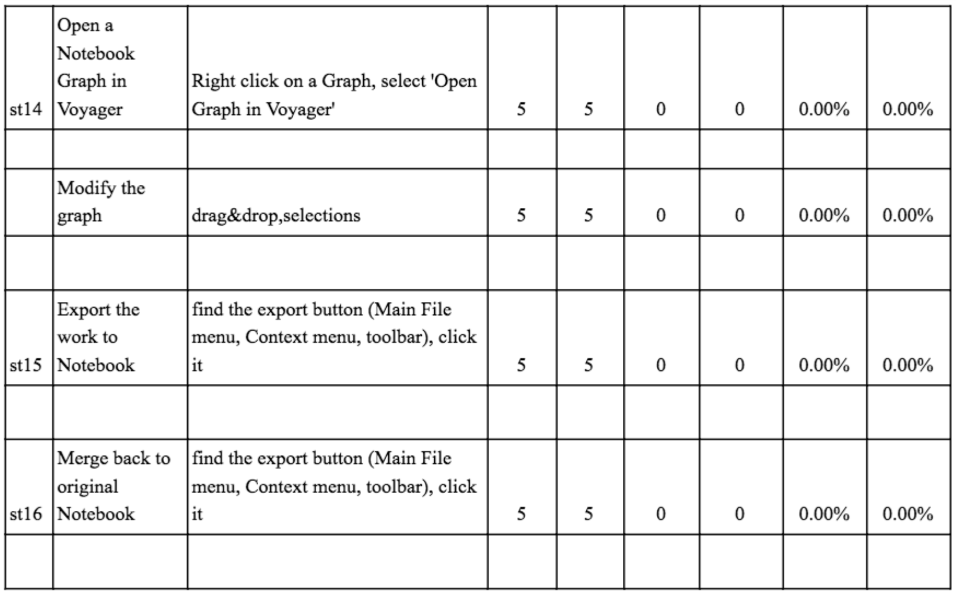
USABILITY TEST RESULT SUMMARY



Step #		Correct Action	# of Participants	# of Successes	# of 1st action errors	# of final action fails	1st action error rate	final action fail rate
FILE SYSTEM and VOYAGER								
st01	Open a csv file in Voyager	right-click on target csv file, select 'Voyager' under 'Open With...'	5	5	1	0	20.00%	0.00%
	Wrong actions: double click file							
	Create a graph	drag&drop,color it	5	5	0	0	0.00%	0.00%
st02	Export the work	find the export button (Main File menu, Context menu, toolbar), click it	5	5	2	0	40.00%	0.00%
	Wrong actions: Use 'Save ' 'Save All', (for the success operation, only one uses the context menu, others only use the 'export' button under Main File menu)							
st03	Open a VL.JSON file in Voyager	right-click on target VL.JSON file, select 'Voyager' under 'Open With...'	5	5	0	0	0.00%	0.00%
	Create a graph	drag&drop, color it	5	5	0	0	0.00%	0.00%
st04	Save the work	find a save button (Main File menu, Context menu, toolbar), click it	5	5	1	0	20.00%	0.00%
	Wrong actions: Use 'Export Notebook as', (for the success operations, most of them realize they can use 'Save' once they saw "Export Voyager" button is disabled.)							
st05	Continue with graph from csv	right-click the VL.JSON created in step 3,, select 'Voyager' under 'Open With...'	5	5	0	0	0.00%	0.00%
	Modify the graph	drag&drop, selections	5	5	0	0	0.00%	0.00%
st06	Save the work	find a save button (Main File menu, Context menu, toolbar), click it	5	5	1	0	20.00%	0.00%
	Wrong actions: One person still looking for the 'Export Voyager' button							



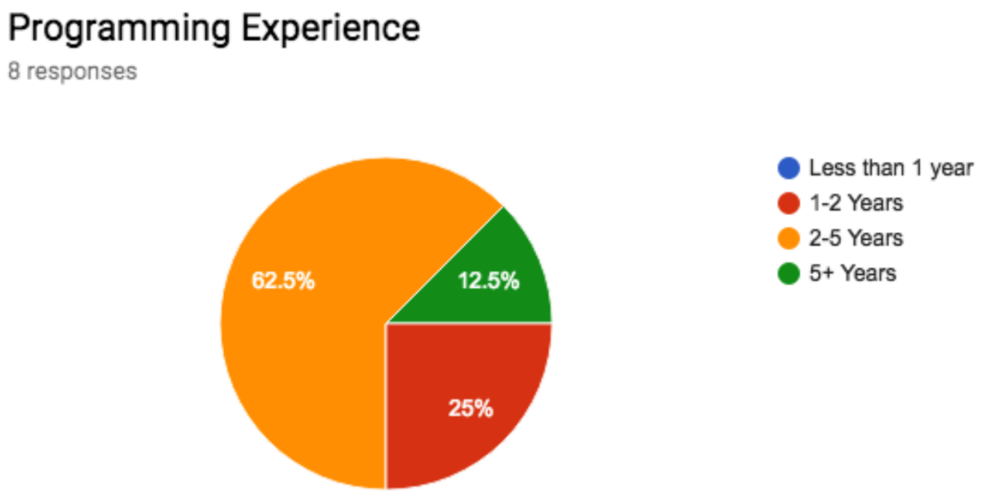
st07	Continue with graph from VL.JSON	right-click the same VL.JSON file in step 4, select 'Voyager' under 'Open With...'	5	5	0	0	0.00%	0.00%
	Modify the graph	drag&drop, selections	5	5	0	0	0.00%	0.00%
st08	Save the work	find a save button (Main File menu, Context menu, toolbar), click it	5	5	0	0	0.00%	0.00%
NOTEBOOK and VOYAGER								
st09	Open a Notebook Graph in Voyager	Right click on a Graph, select 'Open Graph in Voyager'	5	4	1	1	20.00%	20.00%
	Wrong actions: One user really struggle with it, first try to do it within notebook cell, second time try to use the options next to graph							
	Modify the graph	drag&drop,selections	5	5	0	0	0.00%	0.00%
st10	Export the work	find the export button (Main File menu, Context menu, toolbar), click it	5	4	2	1	40.00%	20.00%
	Wrong actions: Use 'Save All' button							
st11	Open a Notebook Table in Voyager	Right click on a Graph, select 'Open Table in Voyager'	5	5	0	0	0.00%	0.00%
	Create a graph	drag&drop,selections	5	5	0	0	0.00%	0.00%
st12	Export the work to Notebook	find the export to Notebook button (Context menu, toolbar), click it	5	3	3	2	60.00%	40.00%
	Wrong actions: Users never used right click inside Voyager to find the context menu							
st13	Merge back to original Notebook	drag&drop, copy&paste	5	5	1	0	20.00%	0.00%

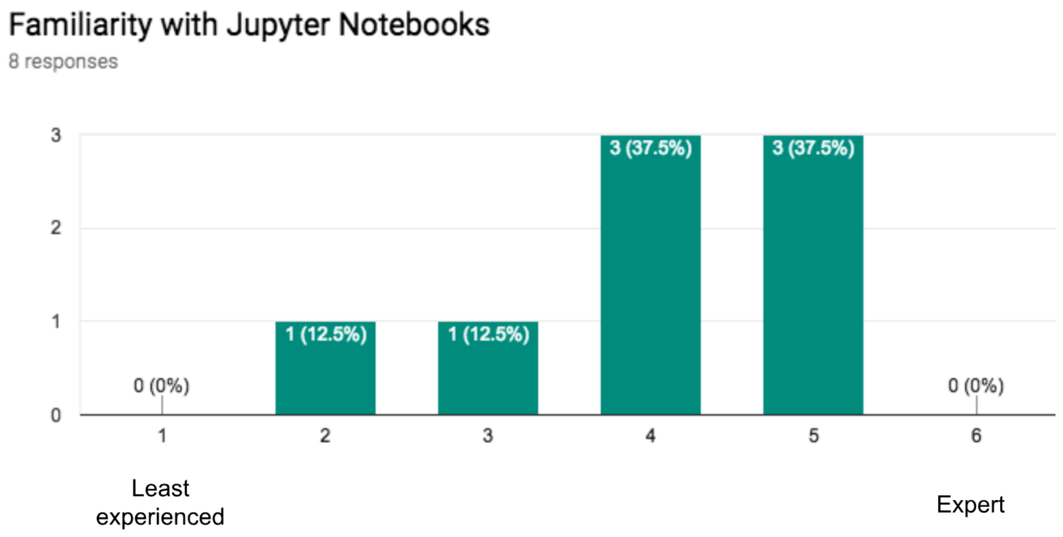


st14	Open a Notebook Graph in Voyager	Right click on a Graph, select 'Open Graph in Voyager'	5	5	0	0	0.00%	0.00%
	Modify the graph	drag&drop,selections	5	5	0	0	0.00%	0.00%
st15	Export the work to Notebook	find the export button (Main File menu, Context menu, toolbar), click it	5	5	0	0	0.00%	0.00%
st16	Merge back to original Notebook	find the export button (Main File menu, Context menu, toolbar), click it	5	5	0	0	0.00%	0.00%

Appendix H

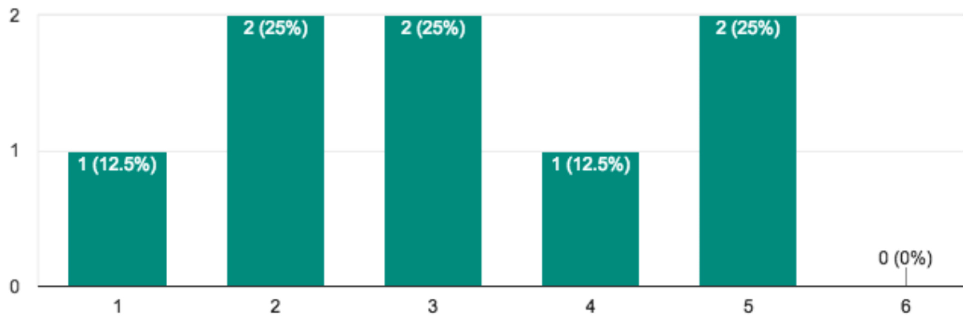
PARTICIPATES BACKGROUND SUMMARY





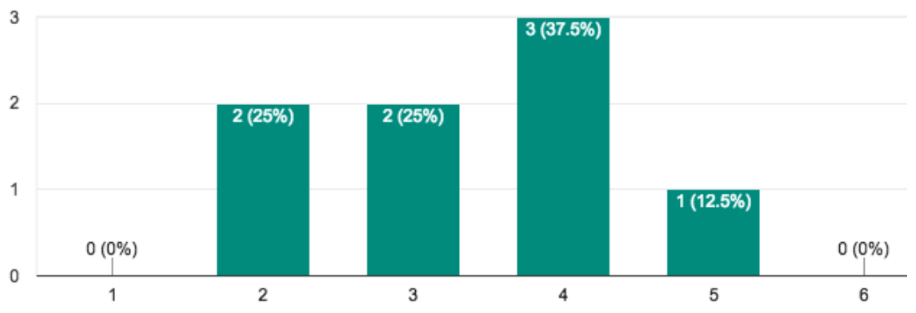
Familiarity with Altair

8 responses



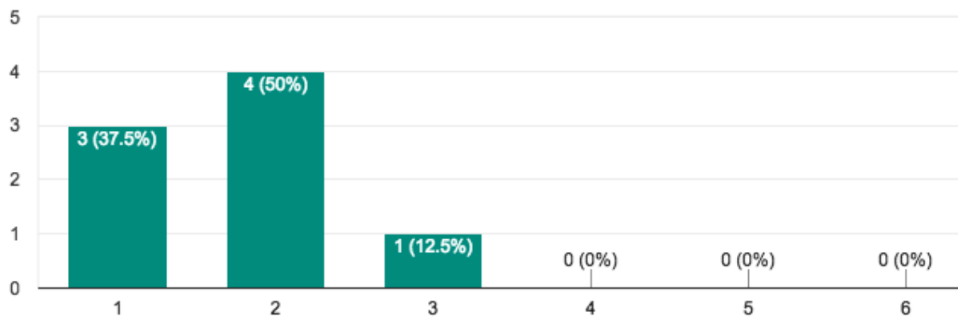
Familiarity with Matplotlib?

8 responses



Familiarity Vega-lite?

8 responses



5
4
3
2
1
0
3 (37.5%)
4 (50%)
1 (12.5%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5
6