Mission Statement

High level summary of project
What are you doing?
Who is it for?

We are adding a simplified version of revision control to the data analytics tool Jupyter. We are adding the ability to tag significant points in the history of a document and the ability to move between these tagged points, make updates, and arrange the history of a document.

Problem

What is the problem you are solving?
What is the need you are filling?

The problem is that there is a young data science tool with a growing user base that lacks a formal source control system. In addition, the trajectory of a data science document is different than that for a piece of software. In software, there is generally a forward, linear flow of changes, where we are maintaining and improving a piece of code. In analytics, the document is meant to answer some question, and the significant states we want to capture are the different solutions to this problem. This means instead of needing to clone or duplicate parts of a project, a developer can tag the current state and then follow a different line of inquiry, without losing track of the previous point.

Context

Describe the context or circumstances
What are the existing solutions?
Why is something new needed here?

Data scientists need to collaborate to solve a problem, they need to be able to see the progression from idea to idea, and not just the final result. Moreover, Jupyter documents are not just code, they are descriptions and analyses as well. This means that in a Jupyter doc, the documentation is more important than the code, which in turn is temporary and used more to supplement the documentation.

Currently there is no dedicated revision control experience. You can fake it by applying external source control to a filesystem in Jupyter, but the existing source control systems have no concept of Jupyter or the Jupyter Notebook, which is the name for set of documents that make up a Jupyter project. In addition, the current best practice for sharing revisions is through JupyterHub, which is a multi-user environment where each user collaborates by sharing the same view of a filesystem, but works on their own single-user copy. Having a concept of history would allow more detailed change tracking in multi-contributor scenarios.

Something new is needed because the current approaches to source control don't fit elegantly with the Jupyter Notebook model. The process to reach a result is important and it is useful to see how a particular solution was reached.

Customer

Who is the customer?
What is the value to them?
Are there other stakeholders?
What is their interest?

The customers of our project are the users of Jupyter. The Jupyter users are not expert software engineers, they are expert analysts; Jupyter is not a tool to build a program, it is a tool to tell a story. This means they want a clear and efficient way to step between possible outcomes for their problem. As an example, let's say some analysts want to model flood overruns based on the height of sandbag barriers on a river. One possible solution would be to not even build a sandbag barrier. While this is clearly not the endpoint of the document, it is still a significant state that the analyst will want to return to. Instead of duplicating work, the analyst can tag this state and rerun the experiment in different settings, which unlocks a more natural way of navigating the evolution of a document.

Challenges

What are the critical make-or-break features?
Examples: form, budget, responsiveness
Other special circumstances?

For the timeline functionality, the three critical features will be the history (made up of a series of snapshots of some form), the ability to tag certain points in the progression of the Jupyter notebook, and the ability to remove parts of the history permanently. For this feature to be useful, it must be easy to go back and forth in history and to comment on particular snapshots. This would allow clients to easily tag meaningful states in the analysis process. The ability to permanently delete a section in history is also crucial for legal reasons, as sometimes the content may not be appropriate to leave in the final version of a notebook. Two wonderful features to add (as a stretch goals) would be some integration with git on the notebook cell level and also some level of real-time collaboration.

Expertise

What special technologies or knowledge will you need?
How will you acquire the necessary expertise?
What sources are available, accessible?

Explain the underlying theory of the problem domain

The number one technology we need to learn is the Jupyter ecosystem. Jupyter was born out of the IPython project, which is spin-off of Python that is aimed directly at data science and data analytics. The Jupyter Project, as it is named, is made up of Jupyter and JupyterHub, where Jupyter is the single user case, and JupyterHub allows multiple users on the same system. The unit of work in Jupyter is called a Notebook, which in turn is built up of cells, which can either contain code or documentation. We need to know the technical subtleties of the Jupyter environment, and how that environment differs from JupyterHub.
More precisely, Jupyter and JupyterHub both operate in a sandbox, and we need to understand the scope and limitations of that sandbox, as well as what else is running in the sandbox, so we know what operations and features are allowed in each environment.

Secondly we need to know how to “scroll back time” in Jupyter. In a traditional code-based repo like Git, changes are stored in a line-based protocol, and we can look back through changesets to find the correct state, but in Jupyter, the data is not line based. The output is a “live” web page style document, stored as an entry in a database. This means finding and tracking changes is a matter of queries and transformations on a database.

In that vein, we need to understand more about databases and database logs. Specifically, Jupyter is build on top of SQLite, so we need to know how SQLite stores changes and timestamps, and how we can recover states from the SQLite log.

Risks

What are the risks to success?
What are your contingency plans?
How can you mitigate these risks?
Examples?

One risk we face is not understanding the client’s expectations for the features and how they will be used. If we implement the wrong thing, our features will not be useful and may even make their daily workflow worse. To minimize this, we need to ensure that we communicate often with both Alva and our end users, and to ensure that we are all on the same page.

Another risk we face is information leak. This can come through not handling the history backups securely, and potentially allowing unauthorized users to be able to view sensitive information. This can also occur by not correctly implementing the delete feature, and thus leaving the risk of inappropriate content being left visible when it should be gone. There is no concrete way to prevent this, outside of being careful and thoroughly checking our work. It may be a good idea to leave a disclaimer to the users until we feel confident that we handle the data securely.

On top of data leaks there is also a risk of data loss. We are planning on “deleting history”, which can be hazardous if done incorrectly, especially when moving between states, so we want to make sure we do not inadvertently corrupt someone’s project while they are trying to splice a point in time.

Ethical questions

Are there ethical questions about the project?
Who is impacted?
What is the underlying ethical issue?
How will you investigate and
mitigate this issue?

The main users of Jupyter are analysts. Having revision control will allow the analysts to show their thought process and document how completely they have considered a problem space. However, the history we present is incomplete and malleable. Having only a small amount of information can be misleading, and there could ways for an analyst to distort a history to present a biased or malicious view. While this is not a problem intrinsic in source control, it is something we need to consider if we are building tools to support analysts.

Goal

What is the desired result?
Describe what success looks like

Our goal is to improve the workflow of data scientists using Jupyter by simple revision control in the form of tagging. We'd like to add a timeline feature and allow improved integration with Git (*ie* not the XML nonsense it does now).

This means that as a developer is working, she should be able to tag important Notebook states, check in on old states for information or fixes/updates, and then (as a stretch goal) be able to check the Notebook into GitHub as a Git repo, where each file in the Git repo is a cell in the Notebook.

If we are able to do these two things, then our project will be a success. As a stretch goal, we would like to use the Git integration to add more sophisticated collaboration workflows to JupyterHub.