Data Versioning control

Data versioning control is a system for managing and tracking changes to data over time, allowing for retrieval of previous versions, comparison of differences, and preservation of a history of modifications. This is useful for maintaining data integrity, ensuring data consistency, and facilitating collaboration and recovery from errors. Examples of data versioning control systems include Git for source code, and version control systems for databases such as Microsoft SQL Server or MongoDB.

Here is an updated tabulated comparison of the open-source tools for dataset versioning control discussed, including links to the tools:

Tool	**Versioning Strategy**	**Advantages**	**Disadvantages**	**Link**
DVC	Git-like	Easy integration with ML workflows, support for large datasets	Limited metadata versioning, limited collaboration features	https://dvc.org/
Quilt Data	Content-addressable file system	Support for very large datasets, easy sharing and collaboration, tracking of lineage and dependencies	Steep learning curve, limited integration with existing workflows	https://quiltdata.com/
Pachyderm	Git-based	End-to-end solution for ML workflows, including data versioning, pipeline management, and reproducibility	Complex architecture, higher resource requirements	https://pachyderm.com/
DataLad	Git-based	Support for complex data structures and metadata, integration with scientific workflows	Steep learning curve, limited collaboration features	https://datalad.org/
Dolt	Git-like for SQL database	Ability to version structured data, familiarity of SQL for querying	Limited collaboration features, limited support for large datasets	https://www.dolthub.com/
Firecamp	Cloud-based	Easy to use, support for popular databases, collaboration features	Need for cloud-based solution, need for paid subscription for some features	https://firecamp.io/

Tool	Versioning Strategy	Advantages	Disadvantages	Link
OpenRefine	Built-in versioning	Wide range of data formats supported, ease of use, ability to clean and transform data	Limited collaboration features, manual intervention required for version management	http://openrefine.org/
Reversion	Versioning for Django models	Integration with Django framework, versioning for structured data, ease of use	Limited collaboration features, limited support for large datasets	https://github.com/etianen/django-reversion

This is not an exhaustive list of all open-source tools for dataset versioning control.

Here are some disadvantages of using a tool-based versioning strategy:

1. Dependency on the tool: The versioning strategy is dependent on the availability and reliability of the tool being used. If the tool is unavailable, the versioning process may be disrupted.
2. Limited compatibility: Some tools may be compatible with certain file formats, but not with others. This can limit the types of data that can be versioned.
3. Complexity: The versioning process may become complex, especially if the tool has a steep learning curve or requires specialized skills to use effectively.
4. Performance issues: The versioning process may consume a significant amount of computational resources and slow down the overall performance of the system.
5. Cost: Some versioning tools may be expensive, especially if they are proprietary or have a large user base.
6. Privacy and security concerns: The use of third-party tools may raise privacy and security concerns, especially if the data being versioned contains sensitive information.
7. Integration difficulties: Integrating the tool into existing systems and workflows may be challenging and require additional resources.

The alternative to a tool-based data versioning strategy is a manual versioning strategy. In this approach, data versioning is performed manually, without relying on any tools. This approach can be useful in small-scale or low-complexity scenarios where the amount of data being versioned is relatively small, and the number of changes is minimal.

Versioning Strategy 1:

Here's a simple manual versioning process:

1. Create a new folder for each version of the data.

2. Copy the current version of the data into the new folder.

3. Rename the copied data to include the version number or a timestamp.

4. Repeat the process each time a new version of the data is created.

The main advantage of this approach is that it does not depend on any tools, so it is simple, straightforward, and easy to implement. However, it can become cumbersome and error-prone as the number of versions grows, and the amount of data being versioned increases. In summary, manual data versioning is a good option for small-scale and low-complexity scenarios, while tool-based data versioning is better suited for larger and more complex scenarios where more sophisticated versioning capabilities are needed.

Versioning Strategy 2:

Here's an offline data versioning strategy for a bunch of CSV files that takes inspiration from the above tools:

1. Store the CSV files in a directory structure: Arrange the CSV files in a directory structure that makes sense for your use case, e.g., create subdirectories based on the type of data, the date range, or the data source.
2. Use a version control system: Use a version control system, such as Git, to track changes to the CSV files. This will provide you with a history of changes and the ability to revert to previous versions if necessary.
3. Version file names: Consider using versioned file names for the CSV files, e.g., append the date or a version number to the file name. This will make it easy to identify the different versions of the files and track changes over time.
4. Document metadata: Document the metadata for each file, such as the file format, the date range of the data, and any relevant descriptions. This metadata can be stored in a separate file or included in the file name.
5. Use checksums: Consider using checksums, such as SHA256, to verify the integrity of the CSV files. Checksums can be stored in a separate file or included in the file name.
6. Back up the data: Regularly back up the directory structure, version control system, and metadata to ensure that the data is not lost.

By following this strategy, you can ensure that your CSV files are well-organized, versioned, and protected.

popular naming conventions followed in data versioning:

1. Timestamp-based naming: This involves appending a timestamp, such as the date and time, to the file name to indicate when the file was created or modified. For example: "data_2022-01-01_12-00-00.csv"
2. Version-based naming: This involves appending a version number to the file name to indicate the current version of the file. For example: "data_v1.csv", "data_v2.csv"
3. Semantic versioning: This involves using a version number that follows the semantic versioning specification, which includes a major version, a minor version, and a patch version. For example: "data_1.0.0", "data_1.1.0"
4. Combination of timestamp and version-based naming: This involves combining the timestamp-based and version-based naming conventions to provide a more comprehensive picture of the file's history. For example: "data_2022-01-01_12-00-00_v1.csv", "data_2022-01-01_12-00-00_v2.csv"

Choosing the right naming convention depends on your specific use case and requirements.

Here's a naming convention that can be used when there are two types of CSV files: data files and referential files:

1. Data Files: Append "_data" to the end of the file name to indicate that the file contains data. For example: "sales_data_2022-01-01.csv".
2. Referential Files: Append "_ref" to the end of the file name to indicate that the file contains referential data. For example: "products_ref_2022-01-01.csv".

In addition to these prefixes, you could also use timestamp-based or version-based naming conventions to provide a more comprehensive picture of the file's history. For example: "sales_data_2022-01-01_v1.csv", "products_ref_2022-01-01_v1.csv".

This naming convention provides a clear and concise picture of the file's history and makes it easy to identify the different types of files and track changes over time.

To display data version, reference files, and connected model version as a web dashboard in SharePoint, you could follow these steps:

1. Set up a SharePoint site: Create a SharePoint site or use an existing one to store your data versioning information.
2. Create a custom list: Create a custom list in SharePoint to store the data versioning information. The list should include columns for the data version, reference files, and model version.

3. Import data: Import the data versioning information from your data source into the custom list in SharePoint.
4. Create a view: Create a view in SharePoint to display the data versioning information in a tabular format.
5. Create a dashboard: Create a dashboard in SharePoint that displays the data versioning information from the custom list in a visual and interactive format. You can use the out-of-the-box SharePoint dashboards, or you can use a third-party dashboard solution like Power BI to create the dashboard.
6. Publish the dashboard: Once the dashboard is complete, publish it to the SharePoint site, and share the link with the relevant stakeholders.

What information can be listed in SharePoint dashboard:

1. Data version information: A table displaying the data version, date of creation, and a brief description of changes made in each version.
2. Reference files information: A table displaying the reference files used in each version of the data, along with the date of creation and any relevant notes.
3. Connected model version information: A table displaying the model version that is connected to each data version, along with the date of creation and any relevant notes.
4. Data lineage visualization: A visual representation of the relationship between data versions, reference files, and model versions. This could be displayed as a flowchart or a diagram showing the dependencies between each component.
5. Interactive filters: Users should be able to filter the data versioning information based on data version, reference files, and model version, making it easy to focus on specific data sets or models.

The dashboard could be displayed in SharePoint, allowing stakeholders to easily access the data versioning information and track changes over time. The visual representation of data lineage and the ability to filter the information makes it easier to understand the relationships between different components, and provides a comprehensive view of the data versioning process.

why do we need a web based dashboard in SharePoint

A web-based dashboard in SharePoint can be useful for data modeling tasks for several reasons:

1. Centralized repository: SharePoint provides a centralized repository for all the data and files related to the modeling task. This makes it easy for stakeholders to access the latest versions of the data and reference files and to view the associated model versions.
2. Collaboration: SharePoint provides a platform for collaboration and communication between stakeholders. This helps ensure that everyone is working with the latest versions of the data and models and can provide feedback and suggestions.

3. Real-time updates: A web-based dashboard in SharePoint can be updated in real-time, making it easy to track changes to the models and data over time. This helps ensure that stakeholders are working with the latest information and helps prevent errors and inconsistencies.
4. Accessibility: SharePoint provides access to the dashboard from anywhere with an internet connection, making it easy for stakeholders to access the information they need, even if they are working remotely.
5. Customizable: SharePoint provides a flexible platform for customization, so you can tailor the dashboard to meet the specific needs of your modeling task. This can include adding charts, graphs, and other visualizations to help stakeholders understand the data and models more easily.

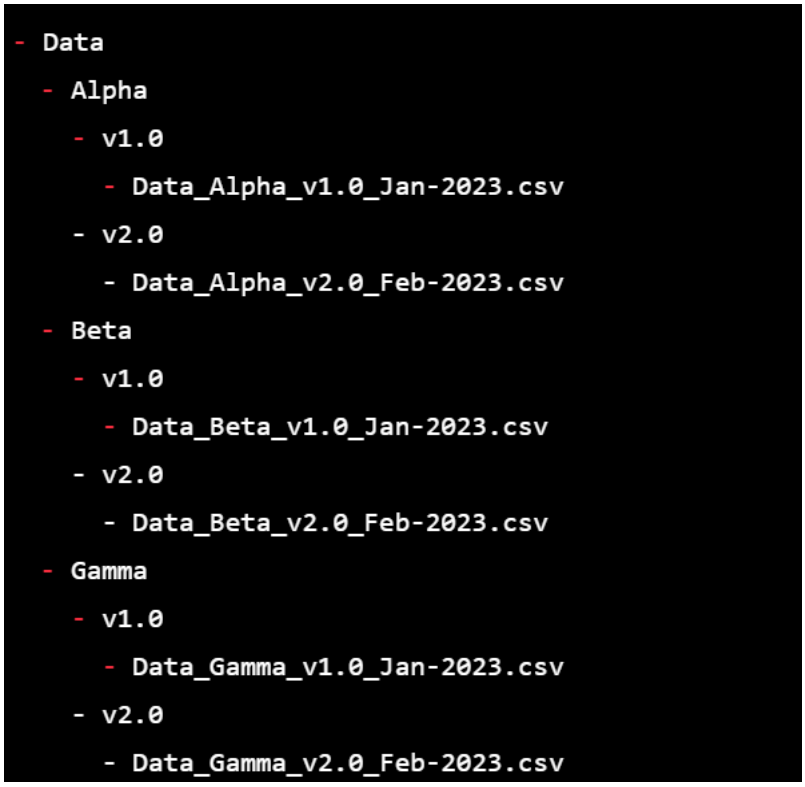
In general, a web-based dashboard in SharePoint provides a centralized repository, collaboration platform, real-time updates, accessibility, and customization options to support data modeling tasks.

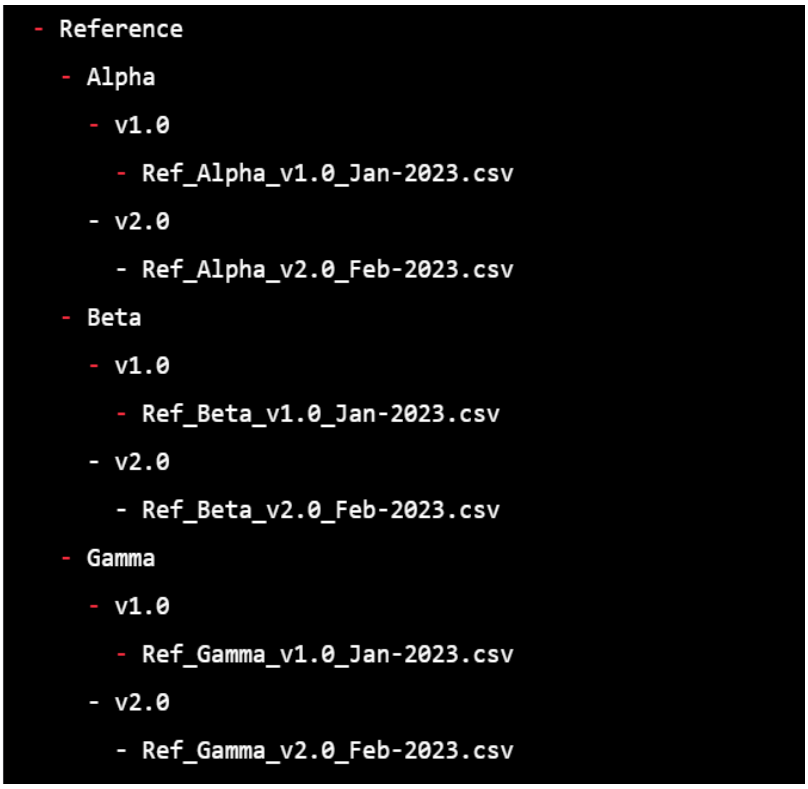
The proposed folder structure for manual data versioning and reference files with a naming convention, following the format of a SharePoint web dashboard:

Dashboard structure:

Model Name	Month/Year	Link to Data Folder	Link to Reference Fo
Alpha_v1.0	Jan-2023	[Link to Data v1.0]	[Link to Ref v1.0]
Beta_v1.0	Jan-2023	[Link to Data v1.0]	[Link to Ref v1.0]
Gamma_v1.0	Jan-2023	[Link to Data v1.0]	[Link to Ref v1.0]
Alpha_v2.0	Feb-2023	[Link to Data v2.0]	[Link to Ref v2.0]
Beta_v2.0	Feb-2023	[Link to Data v2.0]	[Link to Ref v2.0]
Gamma_v2.0	Feb-2023	[Link to Data v2.0]	[Link to Ref v2.0]

Folder structure:





The Data folder for each version may contain files with names such as Data_Alpha_v1.0_Jan-2023.csv and Data_Beta_v1.0_Jan-2023.csv, while the Reference folder for each version may contain files with names such as Ref_Alpha_v1.0_Jan-2023.csv and Ref_Beta_v1.0_Jan-2023.csv.

Option 2:

Dashboard view

Model Name	Model Version	Data Version	Reference Version	Link
Alpha	v1.0	v1.0	v1.0	[Data_
Alpha	v2.0	v2.0	v2.0	[Data_
Beta	v1.0	v1.0	v1.0	[Data_
Beta	v2.0	v2.0	v2.0	[Data_
Gamma	v1.0	v1.0	v1.0	[Data_
Gamma	v2.0	v2.0	v2.0	[Data_

Data view:

```
- Models
  - Alpha
    - v1.0
      - Data
        - Data_Alpha_v1.0_Jan-2023.csv
      - Reference
        - Ref_Alpha_v1.0_Jan-2023.csv
    - v2.0
      - Data
        - Data_Alpha_v2.0_Feb-2023.csv
      - Reference
        - Ref_Alpha_v2.0_Feb-2023.csv
  - Beta
    - v1.0
      - Data
        - Data_Beta_v1.0_Jan-2023.csv
      - Reference
        - Ref_Beta_v1.0_Jan-2023.csv
```

Option 3:

Dashboard

Model	Data Folder	Data File	Reference Folder	Reference File
Alpha	Data_Alpha_2022_01	Alpha_Data_v1_2022_01.csv	Ref_Alpha_2022_01	Alpha_Ref_v1_2022_01.csv
Alpha	Data_Alpha_2022_01	Alpha_Data_v2_2022_02.csv	Ref_Alpha_2022_01	Alpha_Ref_v2_2022_02.csv
Beta	Data_Beta_2022_01	Beta_Data_v1_2022_01.csv	Ref_Beta_2022_01	Beta_Ref_v1_2022_01.csv
Beta	Data_Beta_2022_01	Beta_Data_v2_2022_02.csv	Ref_Beta_2022_01	Beta_Ref_v2_2022_02.csv
Gamma	Data_Gamma_2022_01	Gamma_Data_v1_2022_01.csv	Ref_Gamma_2022_01	Gamma_Ref_v1_2022_01.csv
Gamma	Data_Gamma_2022_01	Gamma_Data_v2_2022_02.csv	Ref_Gamma_2022_01	Gamma_Ref_v2_2022_02.csv

Folder Structure:

```
Data Versioning:

Model: Alpha
- Data Folder: Data_Alpha_2022_01
  - Alpha_Data_v1_2022_01.csv
  - Alpha_Data_v2_2022_02.csv
- Reference Folder: Ref_Alpha_2022_01
  - Alpha_Ref_v1_2022_01.csv
  - Alpha_Ref_v2_2022_02.csv
```