# **Geist**

# A multimodal data transformation, query, and reporting language

Meng Li, Timothy McPhillips, Bertram Ludäscher

Copyright © 2023-2025, CIRSS

# Table of contents

1. Welcome to Geist documentation	3
1.1 Features	3
1.2 Demo for SciPy 2024	3
2. Setup	4
3. Geist Templates	5
3.1 Geist Templates	5
3.2 Tags	12
4. CLI	24
4.1 Command report	24
4.2 Command create	26
4.3 Command destroy	28
4.4 Command export	29
4.5 Command graph	30
4.6 Command load	31
4.7 Command query	32
5. Python API	34
5.1 Class Connection	34
5.2 Functions	54
6. Changelog	69

#### 1. Welcome to Geist documentation

Geist is a new templating language for declarative data manipulation, query, and report generation. Building on the Jinja2 template engine, Geist is designed to support diverse data backends and query engines via predefined tags and filters, and may be extended with custom tags. A single Geist template may include multiple queries expressed in different languages, e.g. SQL and SPARQL, to leverage the strengths of each for clarity and ease of maintenance. Because Geist both can generate reports in diverse formats and perform inserts and updates on new or existing databases during template expansion, Geist templates may orchestrate data extraction, transformation, and load operations spanning multiple tools and data storage systems. The Geist Python package can be installed easily and accessed via the command line. If your dataset is stored in DuckDB and SPARQL queries are more suitable for your problem, then Geist might be for you! Check out our Poster for SciPy 2024!

At the moment, Geist supports DuckDB and RDFLib. More types of data backends will be available in the near future.

#### 1.1 Features

Both CLI and Python API provide the following features:

• report feature: expand a report using dataset(s)

• create feature: create a new dataset

• destroy feature: delete a dataset

• export feature: export a graph

• graph feature: visualize a dataset

• load feature: import data into a dataset

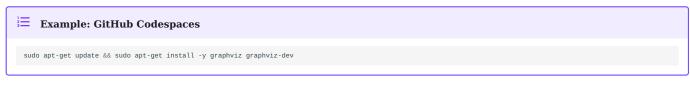
• query feature: perform a query on a dataset

#### 1.2 Demo for SciPy 2024

A Geist report that employs two different query languages. We demonstrate how Geist can be used to extract triples from a relational database, store them as a RDF dataset, and perform SPARQL queries on it. Instead of purely in-memory operations, Geist can be used to migrate data. With the hamming numbers dataset stored in DuckDB as an input, we generate a report to describe the original dataset and the subgraph extracted via SQL and SPARQL queries using a single Geist script.

# 2. Setup

Before installing Geist, please make sure Graphviz is installed.



Example: Google Colab (Jupyter Notebook)

apt install libgraphviz-dev

#### Install Geist:

```
pip install geist-p
```

To check Geist is working, run geist in the command line. You should get the following output:

```
Usage: geist [OPTIONS] COMMAND [ARGS]...

Options:
--help Show this message and exit.

Commands:
create Create a new dataset
destroy Delete a dataset
export Export a dataset
graph Visualize a dataset
load Import data into a dataset
query Perform a query on a dataset
report Expand a report using dataset(s)
```

# 3. Geist Templates

#### 3.1 Geist Templates

#### 3.1.1 What is a Geist template?

A Geist template is a text file without a specific extension requirement although adding a "geist" extension is recommended. It is an extension of a Jinja template, therefore it follows the default Jinja delimiters:

- {% ... %} for Statements
- $\{\{\ldots\}\}$  for Expressions to print to the template output
- $\{\# \dots \#\}$  for Comments not included in the template output

#### 3.1.2 How to write a Geist template?

A Geist template relies on tags and filters.

#### Tags

Tags are used within the statements, i.e., {% ... %}. There are two types of tags, StandaloneTag and ContainerTag. While the StandaloneTag does not require a closing tag, the ContainerTag does. Besides the Jinja predefined tags (e.g., for), Geist supports the following tags:

#### StandaloneTag:

- destroy
- graph
- graph2
- use

#### ContainerTag:

- create
- loadquery
- component
- map
- html
- img
- table

Custom tags can be defined through files with the use tag.

#### **Filters**

Filters are used to modify variables. Each filter can only take one variable as input. Multiple filters can be applied to a single variable in sequence. For example, {{ var|filter1|filter2|filter3 }} denotes the variable var will be processed through filter1 first, then filter2, and filter3 at the end.

Besides the Jinja predefined filters, Geist supports the following filters:

- head: extract the first 5 rows of a Pandas data frame
- ullet csv2df: convert a CSV string to a Pandas data frame
- dict2df: convert a dictionary to a Pandas data frame
- json2df: convert a JSON string to a Pandas data frame
- json2dict: convert a JSON string to a dictionary
- df2json: convert a Pandas data frame to a JSON string
- df2htmltable: convert a Pandas data frame to an HTML table
- $\bullet \ escape\_quotes : \ escape \ both \ double \ and \ single \ quotation \ marks \\$
- process\_str\_for\_html: preprocess a string to be displayed within an HTML document, e.g., replace < with &lt

3.1.3 How to execute (expand) a Geist template?



report command can expand a report (Geist template) using dataset(s).

Here are options of the *report* command:

```
Usage: geist report [OPTIONS]
Expand a report using dataset(s)
Options:
-ifile, --inputfile FILENAME Path of the file containing the report
-oroot, --outputroot TEXT Path of the directory to store the expanded report (default: current directory)
-so, --suppressoutput BOOLEAN Suppress output or not (default: False)
-a, --args <TEXT TEXT>... Arguments to be passed to the report
                                          template, e.g., (arg, value) indicates that {{ arg }} in the report template will be
                                             replaced by value
                                          Show this message and exit.
```

#### **Example 1: expand a report from stdin**

```
geist report << END TEMPLATE
{% create "test", datastore="rdflib", inputformat="nt", isfilepath=False %}
   <http://example.com/drewp> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
   <http://example.com/drewp> <http://example.com/says> "Hello World" .
{% endcreate %}
{% query "test", datastore="rdflib", isfilepath=False as all_triples %}
SELECT ?s ?p ?o
      ORDER BY ?s ?p ?o
{% endquery %}
 \{\% \  \, \text{for $\_$, row in all\_triples.iterrows() $\%$} \\ \text{Subject: $\{\{ \  \, \text{row}["s"] \ \}\}$, Predicate: $\{\{ \  \, \text{row}["p"] \ \}\}$, Object: $\{\{ \  \, \text{row}["o"] \ \}\}$.} 
{% endfor %}
{% destroy "test", datastore="rdflib" %}
END_TEMPLATE
```

#### **Example 2: expand a report from a file**

qeist report --inputfile report.qeist

Here is the report.geist file:

```
 \{\% \ \mathsf{create} \ \mathsf{"test"}, \ \mathsf{datastore="rdflib"}, \ \mathsf{inputformat="nt"}, \ \mathsf{isfilepath=False} \ \% \} 
   {% endcreate %}
{% query "test", datastore="rdflib", isfilepath=False as all_triples %}
    SELECT ?s ?p ?o
    WHERE {
    ORDER BY ?s ?p ?o
{% endquery %}
{% for _, row in all_triples.iterrows() %}
\label{eq:subject: optimizer} Subject: $$\{ row["s"] \}$, Predicate: $$\{ row["p"] \}$, Object: $$\{ row["o"] \}$. $$$ endfor $$$$
{% destroy "test", datastore="rdflib" %}
```

# Example 3: expand a report from a file with external arguments

geist report --inputfile report.geist --args sentence "Hello World" --args feeling Happy

#### Here is the report. geist file:



#### i Python API

report function can expand a report (Geist template) using dataset(s).

Parameters description for report():

Name	Description
inputfile	String. A report to be expanded ([required])
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content (by default, False)
outputroot	String. Path of the directory to store the expanded report (by default, current directory, i.e., ./)
suppressoutput	Bool. True to suppress output (by default, True)
args	Dict. External arguments, e.g., {"arg1": "value1", "arg2": "value2"} denotes that {{ arg1 }} and {{ arg2 }} in the report template will be replaced by value1 and value2 respectively (by default, {})

# Example 1: expand a report from a string

```
import geist
report = """
{% create "test", datastore="rdflib", inputformat="nt", isfilepath=False %}
    <http://example.com/drewp> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
    <http://example.com/drewp> <http://example.com/says> "Hello World" .
{% endcreate %}
{% query "test", datastore="rdflib", isfilepath=False as all_triples %}
      SELECT ?s ?p ?o
      ORDER BY ?s ?p ?o
{% endquery %}
  \{\% \  \, \text{for} \  \, \_, \  \, \text{row in all\_triples.iterrows() } \\  \quad \text{Subject: } \{\{\  \, \text{row["s"] }\}\}, \  \, \text{Predicate: } \{\{\  \, \text{row["p"] }\}\}, \  \, \text{Object: } \{\{\  \, \text{row["o"] }\}\}. \\ \{\% \  \, \text{endfor } \%\} 
{% destroy "test", datastore="rdflib" %}
# Return the expanded report as a string variable named expanded_report
expanded_report = geist.report(inputfile=report)
```

#### **Example 2: expand a report from a file**

```
import geist
# Return the expanded report as a string variable named expanded_report
expanded_report = geist.report(inputfile='report.geist', isinputpath=True)
```

#### Here is the report.geist file:

```
{% create "test", datastore="rdflib", inputformat="nt", isfilepath=False %}
    <http://example.com/drewp> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
    <http://example.com/drewp> <http://example.com/says> "Hello World" .
{% endcreate %}
{% query "test", datastore="rdflib", isfilepath=False as all_triples %}
SELECT ?s ?p ?o
      ORDER BY ?s ?p ?o
\{\% \ \text{endquery } \%\}
 \{\% \ for \ \_, \ row \ in \ all\_triples.iterrows() \ \% \}   Subject: \ \{\{\ row["s"] \ \}\}, \ Predicate: \ \{\{\ row["p"] \ \}\}, \ Object: \ \{\{\ row["o"] \ \}\}. 
{% endfor %}
{% destroy "test", datastore="rdflib" %}
```

# Example 3: expand a report from a file with external arguments

```
import geist

# Return the expanded report as a string variable named expanded_report
expanded_report = geist.report(
    inputfile='report.geist',
    isinputpath=True,
    args={
        "sentence": "Hello World",
        "feeling": "Happy"
    }
)
```

#### Here is the report.geist file:

## 3.2 Tags

#### 3.2.1 Tag create

The create tag creates a dataset based on the given string. By default, the given string is a file path. However, it can be updated by setting the isfilepath field to False. Here are parameters of the create tag:

Name	Description
dataset	Name of RDF dataset to create (by default, kb)
datastore	Data backend. duckdb and rdflib are available for now. (by default, rdflib)
inputformat	Format of the file to be loaded as triples (by default, json-ld). It has to be one of $\{xml, n3, turtle, nt, pretty-xml, trix, trig, nquads, json-ld, hext, csv\}$
infer	Inference to perform on update choosing from { none, rdfs, owl, rdfs_owl} (by default, none). Please check OWL-RL document for detailed information.
isfilepath	A bool value to denote if the given data is a file path or not (by default: True, which denotes the given data is a file path)
table	Table name. Available for duckdb data backend only.
colnames	Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2],] when the input format is csv (by default, None). Available for rdflib data backend only.

# **Example 1:** the given string is not a file path

{% create "test", datastore="rdflib", inputformat="nt", isfilepath=False %}
 <http://example.com/drewp> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
 <http://example.com/drewp> <http://example.com/says> "Hello World" .
{% endoreate %}

#### **Example 2: the given string is a file path**

#### Here is the test.nt file:

 $$$ \begin{array}{ll} <& \text{http://example.com/drewp} & \text{http://www.w3.org/1999/02/22-rdf-syntax-ns\#type} & \text{http://xmlns.com/foaf/0.1/Person} & \text{http://example.com/drewp} & \text{http://example.com/says} & \text{Hello World} & . \\ \end{aligned}$ 

#### Code:

 $\{\% \text{ create "test", datastore="rdflib", inputformat="nt", isfilepath=True \%} \} \text{ test.nt } \{\% \text{ endcreate } \% \}$ 

#### 3.2.2 Tag destroy

The destroy tag deletes a dataset. Here are parameters of the destroy tag:

Name	Description
dataset	Name of RDF dataset to be removed (default  kb)
datastore	Data backend. duckdb and rdflib are available for now. (by default, rdflib)
quiet	Suppress error messages if the provided dataset does not exist

# Example: delete the test dataset

{% destroy dataset="test" %}

OR

{% destroy "test" %}

The <code>.geistdata/test.pkl</code> file will be removed after this operation. By default, you will get an error message if the provided dataset (in this case, it is the <code>test</code> dataset) does not exist. To suppress this error message, you can add the <code>quiet</code> parameter:

{% destroy "test", quiet=True %}

## 3.2.3 Tag graph

The graph tag visualizes a dataset. Here are parameters of the graph tag:

Name	Description
dataset	Name of RDF dataset to be visualized (default kb)
datastore	Data backend. duckdb and rdflib are available for now. (by default, rdflib)
rankdir	Direction of the graph (default TB): TB or BT or LR or RL
mappings	File of the mappings to shorten text (str): path of a JSON file, where the key is the original text and the value is the shorter text.
on	Column(s) to be mapped (default None, which means all columns will be mapped)
samecolor	A bool value to denote if all edges are filled with the same color (default: True)



{% graph "test", datastore="rdflib" %}

## 3.2.4 Tag graph2

The graph2 tag visualizes a dataset. It does not rely on the PyGraphviz pacakge, which makes it more flexible compared to the graph tag. Here are parameters of the graph2 tag:

Name	Description
dataset	Name of RDF dataset to be visualized (default kb)
rankdir	Direction of the graph (default TB): TB or BT or LR or RL
mappings	File of the mappings to shorten text (str): path of a JSON file, where the key is the original text and the value is the shorter text.
on	Column(s) to be mapped (default None, which means all columns will be mapped)
	Graph attributes of Graphviz



## 3.2.5 Tag load

The  $\ensuremath{\operatorname{load}}$  tag imports data into a dataset. Here are parameters of the  $\ensuremath{\operatorname{load}}$  tag:

Name	Description
dataset	Name of RDF dataset to be removed (default kb)
datastore	Data backend. duckdb and rdflib are available for now. (by default, rdflib)
inputformat	Format of the file to be loaded as triples (default json-ld)
isfilepath	A bool value to denote if the given data is a file path or not (default True, which denotes the given data is a file path)
table	Table name to be loaded. Available for duckdb data backend only.
colnames	Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2],] when the input format is csv. Available for rdflib data backend only.

## **Example: load a file into the** test dataset

{% load "test", datastore="rdflib" %} test\_add.jsonld {% endload %}

#### 3.2.6 Tag query

The query tag performs a query on a dataset and returns a Pandas DataFrame. Here are parameters of the query tag:

Name	Description
dataset	Name of a dataset to query (default kb)
datastore	Data backend. duckdb and rdflib are available for now. (by default, rdflib)
isfilepath	A bool value to denote if the given data is a file path or not (default $True$ , which denotes the given data is a file path)

# 

# Example 2: the given string is a file path (% query "test", datastore="rdflib", isfilepath=True %} query\_file {% endquery %} Here is the query\_file's content: SELECT ?s ?p ?o WHERE { ?s ?p ?o } ORDER BY ?s ?p ?o

#### 3.2.7 Tag component

The component tag finds connected components in a graph. It will return a dict where the key is the index of a component (e.g., 0, 1, 2, ...) and the value is a connected component. By default, the given string is a file path. However, it can be updated by setting the isfilepath field to False. Here are parameters of the component tag:

Name	Description
isfilepath	A bool value to denote if the given data is a file path or not (by default: True, which denotes the given data is a file path)
edges	A list of list. [[start_node1, end_node1], [start_node2, end_node2],] or [[start_node1, end_node1, label1], [start_node2, end_node2, label2],] where these items are column names

#### 3.2.8 Tag map

The map replaces the original string (JSON string) on selected columns (if provides) with the shorter ones based on the given mappings. By default, the given string is a file path. However, it can be updated by setting the <code>isfilepath</code> field to False. A Pandas DataFrame will be returned. Here are parameters of the <code>map</code> tag:

Name	Description
isfilepath	A bool value to denote if the given data is a file path or not (by default: True, which denotes the given data is a file path)
mappings	File of the mappings to shorten text (str): path of a JSON file, where the key is the original text and the value is the shorter text.
on	A column or a list of selected columns. All columns will be selected by default ( None )

```
data.json

{
    "v1": {"0":"test_a1", "1":"test_b1", "2":"test_c1"},
    "v2": {"0":"test_a2", "1":"test_b2", "2":"test_c2"),
    "v3": {"0":"test_a3", "1":"test_b3", "3":"test_c3"}
}
```

```
mapping.json
{"test_": ""}
```

```
Example 1: replace all columns

{%- map mappings="mappings.json" as res %} data.json {% endmap %}
{{ res }}

Expected output:

v1,v2,v3
a1,a2,a3
b1,b2,b3
c1,c2,c3
```

```
Example 2: replace selected columns

{% map mappings="mappings.json", on=["v1","v2"] as res %} data.json {% endmap %}

{{ res }}

Expected output:

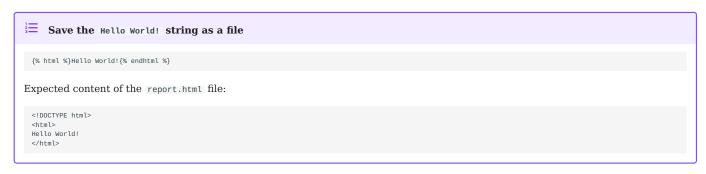
v1,v2,v3
a1,a2,test_a3
b1,b2,test_b3
c1,c2,test_c3

If only "v1" column need to be replaced, you can replace on=["v1","v2"] with on="v1".
```

## 3.2.9 Tag html

The  $\mbox{\sc html}$  tag formats and saves the string as a HTML file. Here is a parameter of the  $\mbox{\sc html}$  tag:

Name	Description
path	Path of the HTML file to be saved. By default, report.html



#### 3.2.10 Tag img

The img tag renders Graphviz code as an image and embeds it into HTML. Here are parameters of the img tag:

Name	Description
src	Path of the rendered image to be saved. Various extensions are supported. Check PyGraphviz Docs for the whole list. Note: dot or gv will show code directly.
	Attributes of the HTML or the HTML tag

## **Example 1: render as svg**

{% img src="test.svg" %}digraph test\_graph { node1 -> node2 }{% endimg %}

A file named  $\ensuremath{\operatorname{test.svg}}$  will be created and the Geist template will be updated as:

<img src="test.svg" width="100%" >

# **Example 2: render as gv**

A file named test.gv will be created and the Geist template will be updated as:

<code>digraph test\_graph { node1 -> node2 }</code>

#### 3.2.11 Tag table

The table tag embeds query results to HTML as a table. Please make sure the stdin is a JSON string. Here are parameters of the table tag:

Name	Description
mappings	File of the mappings to shorten text (str): path of a JSON file, where the key is the original text and the value is the shorter text.
on	A column or a list of selected columns. All columns will be selected by default ( None )

#### 3.2.12 Tag use

The use tag can be used to define custom tags. Here is a parameter of the use tag:

Name	Description
filepath	Path of a file to define custom tags

Here is the structure of tags to be defined within the file at the path filepath:

```
{% template TAG_NAME PARAM1 PARAM2 %}
CONTENT
{% endtemplate %}
```

You need to update TAG\_NAME, PARAM1, PARAM2, and CONTENT based on your use case. TAG\_NAME must be unique, which means you cannot define multiple tags with the same name. You can have any number of parameters, which means {% template TAG\_NAME %} and {% template TAG\_NAME PARAM1 PARAM2 PARAM3 %} are also valid. Nested tags are also supported, which means you can put another tag within the CONTENT part.

#### Example: define predicate\_term and format\_output tags

- 1. Write {% use "templates.geist" %} at the beginning of a Geist template, where you want to use the custom tags, i.e., predicate\_term and format\_output tags.
- 2. Define custom tags in file with the path of "templates.geist":

```
{% template predicate_term %}says{% endtemplate %}
{% template format_output person sent %}
{{ person }} {% predicate_term %} {{sent}}
{% endtemplate %}
```

3. Use custom tags in the Geist template as other predefined tags (e.g., create)

4. Expected output:

```
<http://example.com/test1> says Hello World.
<http://example.com/test2> says What a Nice Day.
```

## 4. CLI

#### 4.1 Command report

report command can expand a report (Geist template) using dataset(s).

Here are options of the *report* command:

```
Usage: geist report [OPTIONS]

Expand a report using dataset(s)

Options:
-ifile, --inputfile FILENAME
-oroot, --outputroot TEXT
-oroot, --outputroot TEXT
-so, --suppressoutput BOOLEAN
-a, --args <TEXT TEXT>...

Path of the file containing the report template to expande [required]
Path of the directory to store the expanded report (default: current directory)

Suppress output or not (default: False)
Arguments to be passed to the report template, e.g., (arg, value) indicates that {arg} in the report template will be replaced by value

Show this message and exit.
```

## **Example 1: expand a report from stdin**

#### **Example 2: expand a report from a file**

```
geist report --inputfile report.geist
```

#### Here is the report.geist file:

# **Example 3: expand a report from a file with external arguments**

{% destroy "test", datastore="rdflib" %}

#### 4.2 Command create

The create command has two subcommands, both of which create a new dataset on disk. The dataset name :memory: is a reserved value for datasets that exist only in memory and is not allowed in the CLI.

```
Usage: geist create [OPTIONS] COMMAND [ARGS]...
Create a new dataset
Options:
--help Show this message and exit.
duckdb Create a new SQL dataset using DuckDB rdflib Create a new RDF dataset using RDFLib
```

# geist create duckdb [OPTIONS] Usage: geist create duckdb [OPTIONS] Create a new SQL dataset using DuckDB Options: -d, --dataset TEXT -ifile, --inputfile FILENAME Name of SQL dataset to create (default "kb") Path of the file to be loaded as a Pandas DataFrame [required] -iformat, --inputformat [csv|json] onj Format of the file to be loaded as a Pandas DataFrame (default csv) Name of the table to be created (default "df") -t, --table TEXT --help Show this message and exit. Example 1: create a test SQL dataset from stdin geist create duckdb --dataset test --inputformat csv --table df << \_\_END\_INPUT\_\_ v1, v2, v3

# 1,2,3 4,5,6 7,8,9 \_\_END\_INPUT\_

#### Example 2: create a test dataset from a file

Here is the test.csv file:

v1, v2, v3 1,2,3 4,5,6 7,8,9

Code:

geist create duckdb --dataset test --inputfile test.csv --inputformat csv --table df

# geist create rdflib [OPTIONS]

Usage: geist create rdflib [OPTIONS]

Create a new RDF dataset Options: -d, --dataset TEXT Name of RDF dataset to create (default "kb")
-ifile, --inputfile FILENAME Path of the file to be loaded as triples [required]
-iformat, --inputformat [xml|n3|turtle|nt|pretty-xml|trix|trig|nquads|json-ld|hext|csv]

Format of the file to be loaded as triples (default json-ld)
Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2], ...] when the input format is csv --colnames TEXT

--infer [none|rdfs|owl|rdfs\_owl]

Inference to perform on update [none, rdfs, owl, rdfs\_owl] (default "none")
Show this message and exit.

--help

#### Example 1: create a test RDF dataset from stdin

geist create rdflib --dataset test --inputformat nt --infer none << \_\_END\_INPUT\_\_</pre> < http://example.com/drewp> < http://www.w3.org/1999/02/22-rdf-syntax-ns#type> < http://xmlns.com/foaf/0.1/Person> .<http://example.com/drewp> <http://example.com/says> "Hello World" \_\_END\_INPUT\_

## Example 2: create a test dataset from a file

Here is the test.nt file:

#### Code:

geist create rdflib --dataset test --inputfile test.nt --inputformat nt --infer none

#### 4.3 Command destroy

destroy command can delete a dataset. The .duckdb or the .pkl file of the corresponding dataset will be discarded.

There are two subcommands for destroy:

```
Usage: geist destroy [OPTIONS] COMMAND [ARGS]...

Delete a dataset

Options:
--help Show this message and exit.

Commands:
duckdb Delete a SQL dataset
rdflib Delete an RDF dataset
```

#### i geist destroy duckdb [OPTIONS]

```
Usage: geist destroy duckdb [OPTIONS]

Delete a SQL dataset

Options:
-d, --dataset TEXT Name of SQL dataset to be removed (default "kb")
-q, --quiet Suppress error messages if the provided dataset does not exist
--help Show this message and exit.
```

#### **Example:** delete the test dataset

geist destroy duckdb --dataset test

The \_\_geistdata/duckdb/test\_duckdb file will be removed after this operation. By default, you will get an error message if the provided dataset (in this case, it is the \_\_test\_dataset) does not exist. To suppress this error message, you can add \_\_-quiet:

geist destroy duckdb --dataset test --quiet

## geist destroy rdflib [OPTIONS]

```
Usage: geist destroy rdflib [OPTIONS]

Delete an RDF dataset

Options:
-d, --dataset TEXT Name of RDF dataset to be removed (default "kb")
-q, --quiet Suppress error messages if the provided dataset does not exist
--help Show this message and exit.
```

#### **Example:** delete the test dataset

geist destroy rdflib --dataset test

The .geistdata/rdflib/test.pkl file will be removed after this operation. By default, you will get an error message if the provided dataset (in this case, it is the test dataset) does not exist. To suppress this error message, you can add --quiet:

geist destroy rdflib --dataset test --quiet

#### 4.4 Command export

export command can export a dataset.

There are two subcommands for *export*:

```
Usage: geist export [OPTIONS] COMMAND [ARGS]...

Export a dataset

Options:
--help Show this message and exit.

Commands:
duckdb Export a SQL dataset
rdflib Export an RDF dataset
```

#### geist export duckdb [OPTIONS]

```
Usage: geist export duckdb [OPTIONS]
Export a SQL dataset
Options:
-d, --dataset TEXT
                                            Name of SQL dataset to be exported (default
                                            "kb")
Path of the directory to store the exported
-oroot, --outputroot TEXT
                                            table (default: current directory). If the given path (i.e., --outputfile) is None or a relative path, it will be ignored.
Path of the file to store the exported table
-ofile, --outputfile TEXT
                                            (default: None)
-oformat, --outputformat [csv|json]
                                            Format of the exported table (default csv)
Name of the table to be exported (default
-t, --table TEXT
                                             "df")
--help
                                            Show this message and exit.
```

#### **Example: export the** df **table in** test **dataset**

By default, the exported table will be printed in terminal:

geist export duckdb --dataset test --table df

# geist export rdflib [OPTIONS]

#### **Example:** export the test dataset

By default, the exported triples will be printed in terminal:

geist export rdflib --dataset test

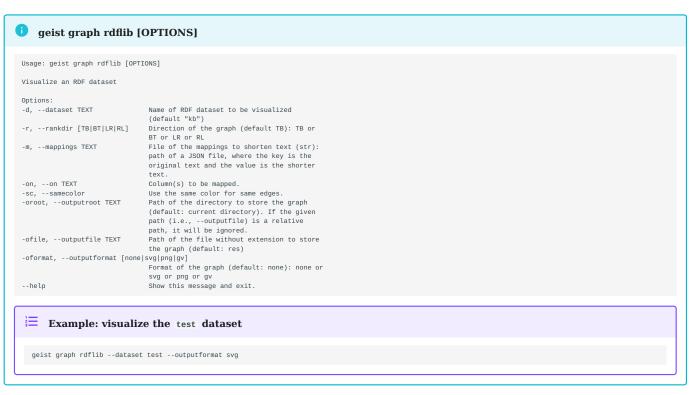
## 4.5 Command graph

graph command can visualize a dataset. Only rdflib is supported for now.

```
Usage: geist graph [OPTIONS] COMMAND [ARGS]...

Visualize a dataset

Options:
--help Show this message and exit.
```



#### 4.6 Command load

load command can import data into an existing dataset.

There are two subcommands for load:

```
Usage: geist load [OPTIONS] COMMAND [ARGS]...

Import data into a dataset

Options:
--help Show this message and exit.

Commands:
duckdb Import data into a SQL dataset
rdflib Import data into a RDF dataset
```

# geist load duckdb [OPTIONS]

```
Usage: geist load duckdb [OPTIONS]

Import data into a SQL dataset

Options:
-d, --dataset TEXT
Name of SQL dataset to load a file (default "kb")
-ifile, --inputfile FILENAME [required]
-iformat, --inputformat [csv|json]
Format of the file to be loaded as a table (default csv)
-t, --table TEXT
Name of fthe file to be loaded as a table (default csv)
-t, --table TEXT
Show this message and exit.
```

# **Example:** load a file into the test dataset

geist load duckdb --dataset test --inputfile test\_add.csv --inputformat csv --table df

#### i geist load rdflib [OPTIONS]

Here are options of the *load* command:

```
Usage: geist load rdflib [OPTIONS]

Import data into a RDF dataset

Options:

-d, --dataset TEXT

Name of RDF dataset to load a file (default "kb")

-ifile, --inputfile FILENAME

-iformat, --inputformat [xml|n3|turtle|nt|pretty-xml|trix|trig|nquads|json-ld|hext|csv]

Format of the file to be loaded as triples (default json-ld)

--colnames TEXT

Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2], ...] when the input format is csv

--help

Show this message and exit.
```

#### **Example:** load a file into the test dataset

geist load rdflib --dataset test --inputfile test\_add.jsonld

#### 4.7 Command query

query command can perform a query on a dataset.

There are two subcommands for query:

```
Usage: geist query [OPTIONS] COMMAND [ARGS]...

Perform a query on a dataset

Options:
--help Show this message and exit.

Commands:
duckdb Perform a SQL query on a dataset
rdflib Perform a SPARQL query on a dataset
```

# i geist query duckdb [OPTIONS]

```
Usage: geist query duckdb [OPTIONS]

Perform a SQL query on a dataset

Options:
-d, --dataset TEXT

Name of RDF dataset to be queried (default "kb")

-ifile, --inputfile FILENAME

-oroot, --outputroot TEXT

Path of the directory to store the query results (default: current directory). If the given path (i.e., --outputfile) is None or a relative path, it will be ignored.

-ofile, --outputfile TEXT

-ofile, --outputfile TEXT

-ofile, --outputfile TEXT

Show this message and exit.
```

# Example 1: all rows of the df table in test dataset from stdin

```
geist query duckdb --dataset test << _END_QUERY_
SELECT * FROM df
_END_QUERY_
```

## **Example 2: all rows of the** test dataset from a query file

geist query duckdb --dataset test --inputfile query\_file

Here is the query\_file's content:

SELECT \* FROM df

#### i geist query rdflib [OPTIONS]

```
Usage: geist query rdflib [OPTIONS]

Perform a SPARQL query on a dataset

Options:
-d, --dataset TEXT

Name of RDF dataset to be queried (default "kb")

-ifile, --inputfile FILENAME

Specify either the path of the file containing the SPARQL query to execute or provide the SPARQL query to execute or provide the SPARQL query itself via stdin [required]

-oroot, --outputroot TEXT

Path of the directory to store the query results (default: current directory). If the given path (i.e., --outputfile) is None or a relative path, it will be ignored.

-ofile, --outputfile TEXT

--help

Show this message and exit.
```

#### **Example 1: all triples of the test dataset from stdin**

```
geist query rdflib --dataset test << _END_QUERY_

SELECT ?s ?p ?o

WHERE {

    ?s ?p ?o

}

ORDER BY ?s ?p ?o

_END_QUERY_
```

## Example 2: all triples of the test dataset from a query file

geist query rdflib --dataset test --inputfile query\_file

Here is the query\_file's content:

# 5. Python API

#### 5.1 Class Connection

 $\textbf{Connection} \ class \ can \ interact \ with \ a \ dataset \ with \ \textit{create} \ , \ \textit{close} \ , \ \textit{destroy} \ , \ \textit{export} \ , \ \textit{graph} \ , \ \textit{load} \ , \ \textit{and} \ \ \textit{query} \ \ \textit{methods}.$ 

#### 5.1.1 What is a Connection class?

A Connection class has three attributes:

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String. Name of the dataset. Note that ':memory:' is a reserved value for datasets that exist only in memory ([required])
conn	DuckPyConnection object OR GeistGraph object (by default, None)

#### 5.1.2 How to instantiate a Connection class?

If the dataset exists, the Connection class can be instantiated using its connect method:

```
# create a Connection object to an existing dataset named test
connection = geist.Connect(datastore='duckdb', dataset='test')
```

If the dataset does not exist, there are two approaches to create and connect:

Approach 1: use the create function, then initialize the Connection class

#### 0

#### create function

create function can create a new dataset on disk or in memory.

Parameters description for create():

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String. Name of the dataset to be created. Note that <code>":memory:"</code> is a reserved value for datasets that exist only in memory ( <code>[required]</code> )
inputfile	String. A file to be loaded ([required])
inputformat	String. Format of the file to be loaded ( [required] )
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content ( $[required]$ )
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the config parameter:



Key	Description
table	String. Name of the table to be created (by default, 'df')

## Example 1: create a test SQL dataset on disk from a string

 $The \ . \texttt{geistdata/duckdb/test.duckdb} \ file \ is \ created \ and \ a \ \texttt{DuckDBPyConnection} \ object \ is \ returned.$ 

```
import geist

csv_str = """
v1,v2,v3
1,2,3
4,5,6
7,8,9
"""

# Create a DuckPyConnection object
conn = geist.create(datastore='duckdb', dataset='test', inputfile=csv_str, inputformat="csv", isinputpath=False, config={"table": "df"})
```

## Example 2: create a test SQL dataset on disk from a file

The .geistdata/duckdb/test.duckdb file is created and a DuckDBPyConnection object is returned.

Here is the test.csv file:

```
v1, v2, v3
1, 2, 3
4, 5, 6
7, 8, 9
```

### Code:

```
import geist

# Create a DuckPyConnection object
conn = geist.create(datastore='duckdb', dataset='test', inputfile="test.csv", inputformat="csv", isinputpath=True, config={"table": "df"})
```

## **Example 3:** create a SQL dataset in memory from a string

A DuckDBPyConnection object is returned.

```
import geist

csv_str = """
v1,v2,v3
1,2,3
4,5,6
7,8,9
"""

# Create a DuckPyConnection object
conn = geist.create(datastore='duckdb', dataset=':memory:', inputfile=csv_str, inputformat="csv", isinputpath=False, config={"table": "df"})
```

# Example 4: create a SQL dataset in memory from a file

A  ${\tt DuckDBPyConnection}$  object is returned.

Here is the test.csv file:

v1, v2, v3 1, 2, 3 4, 5, 6 7, 8, 9

### Code:

import geist

# Create a DuckPyConnection object conn = geist.create(datastore='duckdb', dataset=':memory:', inputfile="test.csv", inputformat="csv", isinputpath=True, config={"table": "df"})



Key	Description
colnames	String. Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2],] ([required] when inputformat='csv')
infer	String. Inference to perform on update, i.e., 'none', 'rdfs', 'owl', or 'rdfs_owl' (by default, 'none')

## Example 1: create a test RDF dataset on disk from a string

The .geistdata/rdflib/test.pkl file is created and a GeistGraph object is returned.

```
import geist
csv str = """
subject, predicate, object
<http://example.com/drewp>,<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,<http://xmlns.com/foaf/0.1/Person>
<http://example.com/drewp>,<http://example.com/says>,"Hello World"
# Create a GeistGraph object: a dictionary with 'rdf_graph' and 'infer' keys conn = geist.create(datastore='rdflib', dataset='test', inputfile=csv_str, inputformat="csv", isinputpath=False, config={"colnames": "[['subject',
'predicate', 'object']]"})
```

## Example 2: create a test RDF dataset on disk from a file

The .geistdata/rdflib/test.pkl file is created and a GeistGraph object is returned.

Here is the test.csv file:

```
subject, predicate, object \\ < http://example.com/drewp>, < http://www.w3.org/1999/02/22-rdf-syntax-ns\#type>, < http://xmlns.com/foaf/0.1/Person> \\ < http://xmlns.com/foaf/0.1/Person
<http://example.com/drewp>,<http://example.com/says>,"Hello World"
```

### Code:

import geist

import geist

# Create a GeistGraph object: a dictionary with 'rdf\_graph' and 'infer' keys conn = geist.create(datastore='rdflib', dataset='test', inputfile="test.csv", inputformat="csv", isinputpath=True, config={"colnames": "[['subject', 'predicate', 'object']]"})

## **Example 3: create a RDF dataset in memory from a string**

A GeistGraph object is returned.

```
csv_str = """
subject, predicate, object
\verb|\disp| < | \text{com/drewp}| < | \text{com/drewp}| < | \text{com/drewp}| < | \text{com/foaf}| < | \text{com
<a href="http://example.com/drewp">http://example.com/says</a>, "Hello World"
# Create a GeistGraph object: a dictionary with 'rdf_graph' and 'infer' keys
conn = geist.create(datastore='rdflib', dataset=':memory:', inputfile=csv_str, inputformat='csv', isinputpath=False, config={"colnames":
"[['subject', 'predicate', 'object']]"})
```

## Example 4: create a RDF dataset in memory from a file

A GeistGraph object is returned.

Here is the test.csv file:

subject, predicate, object subject, predicate, object 
http://example.com/drewp>,http://xmlns.com/foaf/0.1/Person>
<http://example.com/drewp>,http://example.com/says>, "Hello World"

### Code:

import geist

# Create a GeistGraph object: a dictionary with 'rdf\_graph' and 'infer' keys
conn = geist.create(datastore='rdflib', dataset=':memory:', inputfile='test.csv', inputformat='csv', isinputpath=True, config={"colnames":
"[['subject', 'predicate', 'object']]"})

```
import geist
csv_str = """
v1,v2,v3
1,2,3
7,8,9
"""
 # create a Connection object
# create a commection object connection object connection geist.create(datastore='duckdb', dataset=':memory:', inputfile=csv_str, inputformat="csv", isinputpath=False, config={"table": "df"}) connection = geist.Connection(datastore='duckdb', dataset=':memory:', conneconn)
```

Approach 2: use the create method of the Connection class	

## 0

## create method of the Connection class

create method of the Connection class creates a new dataset on disk or in memory. It is very similar to the create() function.
The only difference is that the datastore and the dataset parameters do not need to be passed as they have already been specified while initializing the Connection class.

Parameters description for create method of the Connection class:

Name	Description
inputfile	String. A file to be loaded ([required])
inputformat	String. Format of the file to be loaded ( [required] )
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content ( $[required]$ )
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the config parameter:



Key	Description
table	String. Name of the table to be created (by default, 'df')

## **Example 1: create a test SQL dataset from a string**

The .geistdata/duckdb/test.duckdb file is created and a Connection instance is returned.

```
import geist

csv_str = """
v1,v2,v3
1,2,3
4,5,6
7,8,9
"""

# Create a Connection instance
connection = geist.Connection(datastore='duckdb', dataset='test')
connection.create(inputfile=csv_str, inputformat="csv", isinputpath=False, config={"table": "df"})
```

## **Example 2: create a test SQL dataset from a file**

The .geistdata/duckdb/test.duckdb file is created and a Connection instance is returned.

Here is the test.csv file:

```
v1,v2,v3
1,2,3
4,5,6
7,8,9
```

## Code:

```
import geist

# Create a Connection instance
connection = geist.Connection(datastore='duckdb', dataset='test')
connection.create(inputfile="test.csv", inputformat="csv", isinputpath=True, config={"table": "df"})
```

## **Example 3:** create a SQL dataset in memory from a string

A Connection instance is returned.

```
import geist

csv_str = """
v1,v2,v3
1,2,3
4,5,6
7,8,9

# Create a Connection instance
connection = geist.Connection(datastore='duckdb', dataset=':memory:')
connection.create(inputfile=csv_str, inputformat="csv", isinputpath=False, config={"table": "df"})
```

# **Example 4:** create a SQL dataset in memory from a file

A Connection instance is returned.

Here is the test.csv file:

v1, v2, v3 1, 2, 3 4, 5, 6 7, 8, 9

### Code:

import geist

# Create a Connection instance
connection = geist.Connection(datastore='duckdb', dataset=':memory:')
connection.create(inputfile="test.csv", inputformat="csv", isinputpath=True, config={"table": "df"})



Key	Description
colnames	String. Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2],].([required] when inputformat='csv')
infer	String. Inference to perform on update, i.e., 'none', 'rdfs', 'owl', or 'rdfs_owl' (by default, 'none')

## Example 1: create a test RDF dataset from a string

The .geistdata/rdflib/test.pkl file is created and a Connection instance is returned.

```
import geist
csv str = """
subject, predicate, object
<http://example.com/drewp>,<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,<http://xmlns.com/foaf/0.1/Person>
<http://example.com/drewp>,<http://example.com/says>,"Hello World"
# Create a Connection instance
connection = geist.Connection(datastore='rdflib', dataset='test')
connection.create(inputfile=csv_str, inputformat="csv", isinputpath=False, config={"colnames": "[['subject', 'predicate', 'object']]"})
```

## Example 2: create a test RDF dataset from a file

The .geistdata/rdflib/test.pkl file is created and a Connection instance is returned.

Here is the test.csv file:

```
subject, predicate, object \\ < http://example.com/drewp>, < http://www.w3.org/1999/02/22-rdf-syntax-ns\#type>, < http://xmlns.com/foaf/0.1/Person> \\ < http://xmlns.com/foaf/0.1/Person
<http://example.com/drewp>,<http://example.com/says>,"Hello World"
```

### Code:

```
import geist
# Create a Connection instance
connection = geist.Connection(datastore='rdflib', dataset='test')
connection.create(inputfile="test.csv", inputformat="csv", isinputpath=True, config={"colnames": "[['subject', 'predicate', 'object']]"})
```

## **Example 3: create a RDF dataset in memory from a string**

A Connection instance is returned.

```
import geist
csv_str = """
subject, predicate, object
\verb|\disp| < | \text{com/drewp}| < | \text{com/drewp}| < | \text{com/drewp}| < | \text{com/foaf}| < | \text{com
<a href="http://example.com/drewp">http://example.com/says</a>, "Hello World"
# Create a Connection instance
connection of geist. Connection (datastore='rdflib', dataset=':memory:')
connection.create(inputfile=csv_str, inputformat="csv", isinputpath=False, config={"colnames": "[['subject', 'predicate', 'object']]"})
```



### 5.1.3 How to interact with a Connection class?

Once a Connection class is instantiated, we can interact with it using close, destroy, export, graph, load, and query methods.



**close** method of the *Connection* class is to close the dataset connection, i.e., reset all attributes to None. No parameters are required.

## **Example: close the connection**

Suppose connection is the instance of the Connection class.

# Close the connection
connection.close()

## destroy method

destroy method of the Connection class is to delete the dataset and close the dataset connection.

## **Example:** delete the dataset and close the connection

Suppose connection is the instance of the *Connection* class for a DuckDB dataset named test stored on disk. The following code will delete the .geistdata/duckdb/test.duckdb file.

# Delete the dataset and close the connection
connection.destroy()

### export method

export method of the Connection class exports a dataset. It is very similar to the export() function. The only difference is that the datastore and the dataset parameters do not need to be passed as they have already been specified while initialzing the Connection

Parameters description for export method of the Connection class:

Name	Description
hasoutput	Bool. True to export as a file or print it out ([required])
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the *config* parameter:



### datastore: duckdb

Key	Description
outputroot	String. Path of the directory to store the exported table (by default, $ \cdot\>./\cdot\>)$
outputfile	String. Path of the file to store the exported table (by default, $\ensuremath{None}$ )
outputformat	String. Format of the exported table, i.e., 'csv' or 'json' (by default, 'csv')
table	String. Name of the table to be exported (by default, 'df')

## Example 1: export all rows of the df table in test dataset on disk

There exist a file with the path of <code>.geistdata/duckdb/test.duckdb</code>. The following code returns a Pandas data frame named data.

import geist # Create a Connection instance connection = geist.Connection.connect(datastore='duckdb', dataset='test')
# Export the df table of the test dataset data = connection.export(hasoutput=False, config={'table': 'df'})

## Example 2: export all rows of the df table in test dataset in memory

Suppose conn is a DuckPyConnection object points to a DuckDB dataset in memory. The following code returns a Pandas data frame named data.

import deist # Create a Connection instance connection = geist.Connection(datastore='duckdb', dataset=':memory:', conn=conn)
# Export the df table of the test dataset data = connection.export(hasoutput=False, config={'table': 'df'})



Key	Description
outputroot	String. Path of the directory to store these exported triples (by default, './')
outputfile	String. Path of the file to store these exported triples (by default, None)
outputformat	String. Format of the exported triples, i.e., 'json-ld', 'n3', 'nquads', 'nt', 'hext', 'pretty-xml', 'trig', 'trix', 'turtle', 'longturtle', or 'xml' (by default, 'nt')



## Example 1: export all triples of the test dataset on disk

 $There\ exist\ a\ file\ with\ the\ path\ of\ .geist data/rdflib/test.pkl\ .\ The\ following\ code\ returns\ a\ string\ named\ data\ .$ 

import geist

# Create a Connection instance
connection = geist.Connection.connect(datastore='rdflib', dataset='test')
# Export all triples of the test dataset as a string with the 'nt' format data = connection.export(hasoutput=False)

## Example 2: export all triples of the test dataset in memory

Suppose conn is a GeistGraph object points to a RDF dataset in memory. The following code returns a string named data.

import geist

# Create a Connection instance
connection = geist.Connection(datastore='rdflib', dataset=':memory:', conn=conn)
# Export the df table of the test dataset data = connection.export(hasoutput=False)

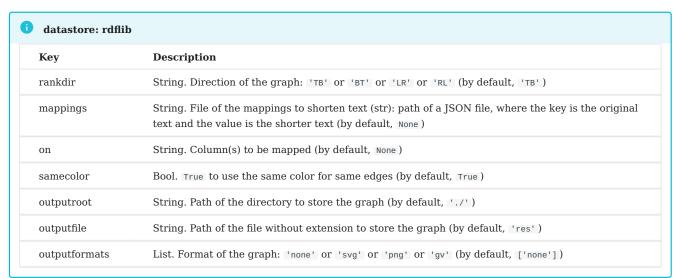
### graph method

graph method of the Connection class exports a dataset. Only rdflib is supported for now. It is very similar to the graph() function. The only difference is that the datastore and the dataset parameters do not need to be passed as they have already been specified while initialzing the Connection class.

Parameters description for graph method of the Connection class:

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String OR GeistGraph object. Dataset to load an object: (1) A string indicates the name of the dataset stored on disk OR (2) a GeistGraph object for dataset in memory ( [required] )
hasoutput	Bool. True to export as a file or print it out ([required])
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the *config* parameter:



## Example 1: visualize the test dataset on disk

There exist a file with the path of <code>.geistdata/duckdb/test.duckdb</code>. The following code visualizes the test dataset as a graph and saves it as the res.svg file.

```
import geist
# Create a Connection instance
{\tt connection = geist.Connection.connect(datastore='duckdb', \ dataset='test')} \ {\tt\#\ Visualize\ the\ test\ dataset}
connection.graph(hasoutput=True, config={'outputformats': ['svg']})
```

## Example 2: visualize the test dataset in memory

Suppose conn is a DuckPyConnection object points to a DuckDB dataset in memory. The following code visualizes the test dataset as a graph and saves it as the res.svg file.

```
import geist
# Create a Connection instance
{\tt connection = geist.Connection(datastore='duckdb', \ dataset=':memory:', \ conn=conn)} \ {\tt\#\ Visualize\ the\ test\ dataset}
connection.graph(hasoutput=True, config={'outputformats': ['svg']})
```

### load method

load method of the Connection class imports data into an existing dataset on disk or in memory. It is very similar to the load() function. The only difference is that datastore, dataset, and immemory parameters do not need to be passed as they have already been specified while initialzing the Connection class.

Parameters description for load method of the Connection class:

Name	Description
inputfile	String. A file to be loaded ([required])
inputformat	String. Format of the file to be loaded ([required])
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content ( [required] )
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the *config* parameter:



### datastore: duckdb

Key	Description
table	String. Name of the table to be loaded ( [required] )

## Example: load a table into the test dataset

There exist a file with the path of <code>.geistdata/duckdb/test.duckdb</code>. The <code>csv\_str</code> will be imported into the <code>df</code> table. Note that the order of table columns should be consistent with the imported data.

```
import geist
csv_str = """
v1, v2, v3
1,1,1
3,3,3
# Create a Connection instance
# Greate a Connection = geist.Connection.connect(datastore='duckdb', dataset='test')
# Load csv_str to the df table of the test dataset
connection.load(inputfile=csv_str, inputformat='csv', isinputpath=False, config={'table': 'df'})
```



Key	Description
colnames	String. Column names of triples with the format of [[subject1, predicate1, object1], [subject2,
	<pre>predicate2, object2],] ([required] when inputformat='csv')</pre>

## **Example:** load a triple into the test dataset

 $There \ exist \ a \ file \ with \ the \ path \ of \ .geistdata/rdflib/test.pkl. \ The \ csv\_str \ will \ be \ imported \ into \ the \ test \ RDF \ dataset.$ 

```
import geist
csv_str = """
subject, predicate, object
<http://example.com/drewp>, <http://example.com/feels>, "Happy"
"""
# Create a Connection instance
connection = geist.Connection.connect(datastore='rdflib', dataset='test')
# Load csv_str to the test dataset
connection.load(inputfile=csv_str, inputformat='csv', isinputpath=False, config={"colnames": "[['subject', 'predicate', 'object']]"})
```

### query method

query method of the Connection class can query a dataset stored on disk or in memory. It is very similar to the query() function. The only difference is that the datastore and the dataset parameters do not need to be passed as they have already been specified while initialze the Connection class.

Parameters description for query method of the Connection class:

Name	Description
inputfile	String. File containing the query ( [required] )
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content ( $[required]$ )
hasoutput	Bool. True to store the query results as a CSV file or print them out ( $[required]$ )
config	Dict. A dictionary with configurations when hasoutput=True. See below for detailed information

Description for the *config* parameter:

Key	Description
outputroot	String. Path of the directory to store the query results (by default, ''./'
outputfile	String. Path of the file to store the query results (by default, None

### Example 1: all rows of the df table in test dataset on disk (query from a string)

 $There\ exist\ a\ file\ with\ the\ path\ of\ . geist data/duckdb/test.duckdb\ .\ The\ following\ code\ returns\ a\ Pandas\ data\ frame\ named\ res$ with query results.

```
import geist
# Ordered a Commection = geist.Connection.connect(datastore='duckdb', dataset='test')
# Query the df table of the test dataset
res = connection.query(inputfile="SELECT * FROM df;", isinputpath=False, hasoutput=False)
```

## Example 2: all rows of the df table in test dataset on disk (query from a file)

 $There\ exist\ a\ file\ with\ the\ path\ of\ .geistdata/duckdb/test.duckdb\ .\ The\ following\ code\ returns\ a\ Pandas\ data\ frame\ named\ res$ with query results.

Here is the query.txt file:

```
SELECT * FROM df;
```

### Code:

```
import geist
# Create a Connection instance
{\tt connection = geist.Connection.connect} ({\tt datastore='duckdb'}, \ {\tt dataset='test'}) \\ {\tt \# Query the df table of the test dataset}
res = connection.query(inputfile="query.txt", isinputpath=True, hasoutput=False)
```

## Example 3: all rows of the df table in test dataset in memory (query from a string)

Suppose conn is a DuckPyConnection object points to a DuckDB dataset in memory. The following code returns a Pandas data frame named res with query results.

```
import geist
# Create a Connection instance
# Greate a connection = geist.Connection(datastore='duckdb', dataset=':memory:', conn=conn)
# Query the df table of the test dataset
res = connection.query(inputfile="SELECT * FROM df;", isinputpath=False, hasoutput=False)
```

## Example 4: all rows of the df table in test dataset in memory (query from a file)

Suppose conn is a DuckPyConnection object points to a DuckDB dataset in memory. The following code returns a Pandas data frame named res with query results.

Here is the query.txt file:

SELECT \* FROM df;

### Code:

import geist

# Create a Connection instance # Grate a connection = geist.Connection(datastore='duckdb', dataset=':memory:', conn=conn)
# Query the df table of the test dataset
res = connection.query(inputfile="query.txt", isinputpath=True, hasoutput=False)

- 53/69 -

## 5.2 Functions

### 5.2.1 Function report

**report** function can expand a report (Geist template) using dataset(s).

Parameters description for report():

Name	Description
inputfile	String. A report to be expanded ([required])
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content (by default, False)
outputroot	String. Path of the directory to store the expanded report (by default, current directory, i.e., 7)
suppressoutput	Bool. True to suppress output (by default, True)
args	Dict. External arguments, e.g., {"arg1": "value1", "arg2": "value2"} denotes that {{ arg1 }} and {{ arg2 }} in the report template will be replaced by value1 and value2 respectively (by default, {})

# Example 1: expand a report from a string

## **Example 2: expand a report from a file**

```
import geist
 # Return the expanded report as a string variable named expanded_report
expanded_report = geist.report(inputfile='report.geist', isinputpath=True)
Here is the report.geist file:
 {% endcreate %}
  {% query "test", datastore="rdflib", isfilepath=False as all_triples %}
SELECT ?s ?p ?o
     ORDER BY ?s ?p ?o
  \{\% \text{ endquery } \%\}
 {% for _, row in all_triples.iterrows() %}
Subject: {{ row["s"] }}, Predicate: {{ row["p"] }}, Object: {{ row["o"] }}.
```

## Example 3: expand a report from a file with external arguments

```
# Return the expanded report as a string variable named expanded_report
expanded_report = geist.report(
   inputfile='report.geist',
     isinputpath=True,
    args={
    "sentence": "Hello World",
    "feeling": "Happy"
```

## Here is the report.geist file:

{% endfor %}

{% destroy "test", datastore="rdflib" %}

```
{% endcreate %}
{% query "test", datastore="rdflib", isfilepath=False as all_triples %}
   SELECT ?s ?p ?o
WHERE {
     ?s ?p ?o
   ORDER BY ?s ?p ?o
{% endquery %}
{% for _, row in all_triples.iterrows() %}
   Subject: {{ row["s"] }}, Predicate: {{ row["p"] }}, Object: {{ row["o"] }}.
{% endfor %}
{% destroy "test", datastore="rdflib" %}
```

## 5.2.2 Function create

create function can create a new dataset on disk or in memory.

Parameters description for create():

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String. Name of the dataset to be created. Note that <code>':memory:'</code> is a reserved value for datasets that exist only in memory ( <code>[required]</code> )
inputfile	String. A file to be loaded ([required])
inputformat	String. Format of the file to be loaded ( [required] )
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content ([required])
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the *config* parameter:

## datastore: duckdb

import geist

Key	Description
table	String. Name of the table to be created (by default, 'df')

## Example 1: create a test SQL dataset on disk from a string

The .geistdata/duckdb/test.duckdb file is created and a DuckDBPyConnection object is returned.

```
csv_str = """
v1,v2,v3
1,2,3
4,5,6
7,8,9
"""

# Create a DuckPyConnection object
conn = geist.create(datastore='duckdb', dataset='test', inputfile=csv_str, inputformat="csv", isinputpath=False, config={"table": "df"})
```

## Example 2: create a test SQL dataset on disk from a file

The .geistdata/duckdb/test.duckdb file is created and a DuckDBPyConnection object is returned.

Here is the test.csv file:

```
V1, V2, V3
1, 2, 3
4, 5, 6
7, 8, 9
```

### Code:

```
import geist

# Create a DuckPyConnection object
conn = geist.create(datastore='duckdb', dataset='test', inputfile="test.csv", inputformat="csv", isinputpath=True, config={"table": "df"})
```

## **Example 3: create a SQL dataset in memory from a string**

A DuckDBPyConnection object is returned.

```
import geist

csv_str = """
v1,v2,v3
1,2,3
4,5,6
7,8,9
"""

# Create a DuckPyConnection object
conn = geist.create(datastore='duckdb', dataset=':memory:', inputfile=csv_str, inputformat="csv", isinputpath=False, config={"table": "df"})
```

## **Example 4:** create a SQL dataset in memory from a file

A DuckDBPyConnection object is returned.

Here is the test.csv file:

v1, v2, v3 1,2,3 4,5,6 7,8,9

### Code:

import geist

# Create a DuckPyConnection object conn = geist.create(datastore='duckdb', dataset=':memory:', inputfile="test.csv", inputformat="csv", isinputpath=True, config={"table": "df"})



Key	Description
colnames	String. Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2],] ([required] when inputformat='csv')
infer	String. Inference to perform on update, i.e., 'none', 'rdfs', 'owl', or 'rdfs_owl' (by default, 'none')



## **Example 1:** create a test RDF dataset on disk from a string

The .geistdata/rdflib/test.pkl file is created and a GeistGraph object is returned.

```
import geist
subject.predicate.object
<http://example.com/drewp>,<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,<http://xmlns.com/foaf/0.1/Person>
<a href="http://example.com/drewp">http://example.com/says>,"Hello World"</a>
# Create a GeistGraph object: a dictionary with 'rdf_graph' and 'infer' keys conn = geist.create(datastore='rdflib', dataset='test', inputfile=csv_str, inputformat="csv", isinputpath=False, config={"colnames": "[['subject',
'predicate', 'object']]"})
```

## Example 2: create a test RDF dataset on disk from a file

The <code>.geistdata/rdflib/test.pkl</code> file is created and a <code>GeistGraph</code> object is returned.

Here is the test.csv file:

```
<http://example.com/drewp>,<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,<http://xmlns.com/foaf/0.1/Person>
<http://example.com/drewp>,<http://example.com/says>,"Hello World"
```

### Code:

```
import geist
# Create a GeistGraph object: a dictionary with 'rdf_graph' and 'infer' keys
```

## Example 3: create a RDF dataset in memory from a string

A GeistGraph object is returned.

```
import geist
subject, predicate, object
<http://example.com/drewp>,<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,<http://xmlns.com/foaf/0.1/Person>
<a href="http://example.com/drewp">http://example.com/says>, "Hello World"</a>
# Create a GeistGraph object: a dictionary with 'rdf_graph' and 'infer' keys conn = geist.create(datastore='rdflib', dataset=':memory:', inputfile=csv_str, inputformat='csv', isinputpath=False, config={"colnames": "[['subject',
'predicate', 'object']]"})
```

## **Example 4:** create a RDF dataset in memory from a file

A GeistGraph object is returned.

Here is the test.csv file:

subject, predicate, object subject, predicate, object

http://example.com/drewp>,
http://example.com/drewp>,
http://example.com/drewp>,
http://example.com/drewp>,
http://example.com/drewp>,
http://example.com/drewp>,
http://example.com/drewp>
<pr

### Code:

import geist

# Create a GeistGraph object: a dictionary with 'rdf\_graph' and 'infer' keys conn = geist.create(datastore='rdflib', dataset=':memory:', inputfile='test.csv', inputformat='csv', isinputpath=True, config={"colnames": "[['subject', 'predicate', 'object']]"})

## 5.2.3 Function destroy

destroy function can delete a dataset.

Parameters description for *destroy()*:

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String. Name of the dataset to be removed ([required])
quiet	Bool. True to suppress error messages if the provided dataset does not exist (by default, False)

## Example: delete the test dataset

geist.destroy(datastore='rdflib', dataset='test')

The .geistdata/rdflib/test.pkl file will be removed after this operation. By default, you will get an error message if the provided dataset (in this case, it is the test dataset) does not exist. To suppress this error message, you can set quiet=True:

import geist
geist.destroy(datastore='rdflib', dataset='test', quiet=True)

## 5.2.4 Function export

export function can export a dataset.

Parameters description for export():

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String OR DuckPyConnection object OR GeistGraph object. Dataset to load an object: (1) A string indicates the name of the dataset stored on disk OR (2) a DuckPyConnection object OR a GeistGraph object for dataset in memory ( [required] )
hasoutput	Bool. True to export as a file or print it out ([required])
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the config parameter:



### datastore: duckdb

Key	Description
outputroot	String. Path of the directory to store the exported table (by default, $^{\shortmid}./^{\shortmid}$ )
outputfile	String. Path of the file to store the exported table (by default, None )
outputformat	String. Format of the exported table, i.e., csv or json (by default, 'csv')
table	String. Name of the table to be exported (by default, 'df')

## Example 1: export all rows of the df table in test dataset on disk

 $There\ exist\ a\ file\ with\ the\ path\ of\ .geistdata/duckdb/test.duckdb\ .\ The\ following\ code\ returns\ a\ Pandas\ data\ frame\ named\ data$ and a  ${\tt DuckPyConnection}$  object named  ${\tt conn}$  .

import geist # Export the df table of the test dataset (data, conn) = geist.export(datastore='duckdb', dataset='test', hasoutput=False, config={'table': 'df'})

## Example 2: export all rows of the df table in test dataset in memory

 $Suppose \ \ conn \ \ is \ a \ \ DuckPyConnection \ \ object \ points \ to \ a \ DuckDB \ dataset \ in \ memory. \ The following \ code \ returns \ a \ Pandas \ data$ frame named data and the same DuckPyConnection object named conn.

import geist # Export the df table of the test dataset
(data, conn) = geist.export(datastore='duckdb', dataset=conn, hasoutput=False, config={'table': 'df'})

Key	Description
outputroot	String. Path of the directory to store these exported triples (by default, "./')
outputfile	String. Path of the file to store these exported triples (by default, None)
outputformat	String. Format of the exported triples, i.e., 'json-ld', 'n3', 'nquads', 'nt', 'hext', 'pretty-xml', 'trig', 'trix', 'turtle', 'longturtle', or 'xml' (by default, 'nt')

## Example 1: export all triples of the test dataset on disk

 $There\ exist\ a\ file\ with\ the\ path\ of\ . geist\ data/rdflib/test.pkl\ .\ The\ following\ code\ returns\ a\ string\ named\ data\ and\ a\ Geist\ Graph$ object named conn.

import geist

# Export all triples of the test dataset
(data, conn) = geist.export(datastore='rdflib', dataset='test', hasoutput=False)

## Example 2: export all triples of the test dataset in memory

Suppose conn is a GeistGraph object points to a RDF dataset in memory. The following code returns a string named data and the same GeistGraph object named conn.

import geist

# Export all triples of the test dataset

(data, conn) = geist.export(datastore='rdflib', dataset=conn, hasoutput=False)

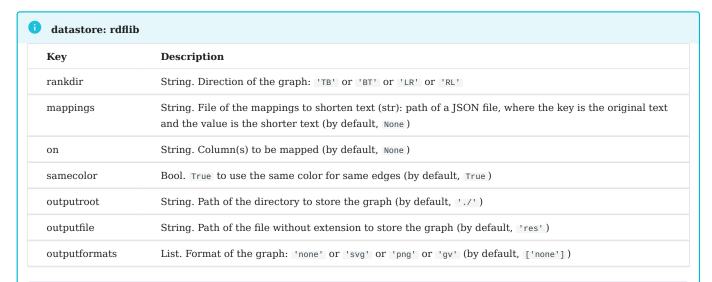
## 5.2.5 Function graph

graph function can visualize a dataset. Only rdflib is supported for now.

Parameters description for export():

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String OR GeistGraph object. Dataset to load an object: (1) A string indicates the name of the dataset stored on disk OR (2) a GeistGraph object for dataset in memory ([required])
hasoutput	Bool. True to export as a file or print it out ([required])
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the config parameter:



## **Example: visualize the test dataset**

import geist

# Visualize the test dataset as a graph and save it as the res.svg file
geist.graph(datastore='rdflib', dataset='test', hasoutput=True, config={'outputformats': ['svg']})

## 5.2.6 Function load

load function can import data into an existing dataset.

Parameters description for query():

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String OR DuckPyConnection object OR GeistGraph object. Dataset to load an object: (1) A string indicates the name of the dataset stored on disk OR (2) a DuckPyConnection object OR a GeistGraph object for dataset in memory ([required])
inputfile	String. File to be loaded ( [required] )
inputformat	String. Format of the file to be loaded ( [required] )
isinputpath	Bool. True if the inputfile is the file path, otherwise the inputfile is the content ( $[required]$ )
config	Dict. A dictionary with configurations for certain backend store. See below for detailed information

Description for the *config* parameter:



## datastore: duckdb

Key	Description
table	String. Name of the table to be loaded ( [required] )

import geist

## Example: load a table into the test dataset

 $There \ exist \ a \ file \ with \ the \ path \ of \ .geist data/duckdb/test.duckdb \ . \ The \ csv\_str \ will \ be \ imported \ into \ the \ df \ table. \ Note that \ the$ order of table columns should be consistent with the imported data.

```
csv_str = """
v1,v2,v3
1,1,1
2,2,2
3,3,3
geist.load(datastore='duckdb',\ dataset='test',\ inputfile=csv\_str,\ inputformat='csv',\ is input path=False,\ config=\{'table':\ 'df'\})
```



Key	Description
inmemory	Bool. True if the new dataset (after loading data) is stored in memory only, otherwise it is stored on disk (by default, False)
colnames	String. Column names of triples with the format of [[subject1, predicate1, object1], [subject2, predicate2, object2],] ([required] when inputformat='csv')

## Example: load a triple into the test dataset

There exist a file with the path of <code>.geistdata/rdflib/test.pkl</code> . The <code>csv\_str</code> will be imported into the <code>test RDF</code> dataset.

```
import geist
csv_str = """
subject,predicate,object
<http://example.com/drewp>,<http://example.com/feels>,"Happy"
"""
# Load csv_str to the df table of the test dataset
geist.load(datastore='rdflib', dataset='test', inputfile=csv_str, inputformat='csv', isinputpath=False, config={"colnames": "[['subject', 'predicate', 'object']]"})
```

## 5.2.7 Function query

query function can perform a query on a dataset.

Parameters description for query():

Name	Description
datastore	String. A backend datastore, i.e., 'rdflib' or 'duckdb' ([required])
dataset	String OR DuckPyConnection object OR GeistGraph object: (1) A string indicates the name of the dataset stored on disk OR (2) a DuckPyConnection object OR a GeistGraph object for dataset in memory ([required])
inputfile	String. File containing the query ([required])
isinputpath	Bool
hasoutput	Bool
config	Dict

## Description for the config parameter:

Key	Description
outputroot	String. Path of the directory to store the query results (by default, $\ '\cdot .'\ )$
outputfile	String. Path of the file to store the query results (by default, None)

### 1 =

## Example 1: all rows of the df table in test dataset on disk (query from a string)

There exist a file with the path of .geistdata/duckdb/test.duckdb. The following code returns a Pandas data frame named res with query results, and a DuckPyConnection object.

import geist
# Query the df table of the test dataset
(res, conn) = geist.query(datastore='duckdb', dataset='test', inputfile="SELECT \* FROM df;", isinputpath=False, hasoutput=False)

### 1 2

## Example 2: all rows of the df table in test dataset on disk (query from a file)

There exist a file with the path of <code>.geistdata/duckdb/test.duckdb</code> . The following code returns a Pandas data frame named <code>res</code> with query results, and a <code>DuckPyConnection</code> object.

Here is the query.txt file:

SELECT \* FROM df;

### Code:

import geist

# Query the df table of the test dataset
(res, conn) = geist.query(datastore='duckdb', dataset='test', inputfile="query.txt", isinputpath=True, hasoutput=False)

## Example 3: all rows of the df table in test dataset in memory (query from a string)

Suppose conn is a DuckPyConnection object points to a DuckDB dataset in memory. The following code returns a Pandas data frame named  $\,$  res  $\,$  with query results, and the same  $\,$  DuckPyConnection  $\,$  object.

import geist # Query the df table of the test dataset
(res, conn) = geist.query(datastore='duckdb', dataset=conn, inputfile="SELECT \* FROM df;", isinputpath=False, hasoutput=False)

## Example 4: all rows of the df table in test dataset in memory (query from a file)

Suppose conn is a DuckPyConnection object points to a DuckDB dataset in memory. The following code returns a Pandas data frame named res with query results, and the same DuckPyConnection object.

Here is the query.txt file:

SELECT \* FROM df;

### Code:

import geist

# Query the df table of the test dataset
(res, conn) = geist.query(datastore='duckdb', dataset=conn, inputfile="query.txt", isinputpath=True, hasoutput=False)

## 6. Changelog

### v0.3.1 (2025-01-06)

- Fix minor bug of the report feature: update TAGS and filters
- Fix minor bug of the load feature: update the rdflib load() function
- Add external arguments for the report feature: {{ arg }} will be replaced by value in the report template

### v0.3.0 (2024-12-26)

- Code refactoring: datastore -> api -> commands (e.g., code under the api folder is based on code under the datastore folder)
- Add Python API: (1) Connection class with create, close, destroy, export, graph, load, and query methods; and (2) create, destroy, export, graph, load, query, and report functions
- $\bullet$  Update CLI of report and query commands: replace --file with --inputfile
- Update the documentation

### v0.2.1 (2024-08-17)

- Add the Geist Poster for SciPy 2024
- Update the documentation: add descriptions for the demo of SciPy 2024

### v0.2.0 (2024-03-15)

- SQL queries are supported by GEIST based on duckdb
- Update ContainerTag: return objects of any type, not just strings

### v0.1.0 (2024-03-04)

- Add documentation
- $\bullet$  Add the component tag to extract connected components of a given graph
- Add the process\_str\_for\_html filter
- Make the map tag more flexible: make it possible to map selected columns
- Fix the quotes bug: keep the cell's original format

## v0.0.1 (2023-09-21)

- The first version of GEIST with create, load, query, destroy, graph, graph2, map, use, html, img, and table tags
- SPARQL queries are supported by GEIST based on RDFLib